MULTITHREADING
JDBC
WEB CLIENT UI

What is the Java Thread Model?

The **Java Thread Model** defines **how multiple threads of execution** are managed, scheduled, and coordinated within a Java program.

It allows multiple parts of a program to **run concurrently,** improving efficiency and responsiveness — especially in **networking, GUI,** and **server-side applications**.

Concept	Description
Thread	A lightweight sub-process; smallest unit of CPU execution.
Multithreading	Running multiple threads simultaneously within a single process.
Concurrency	Executing multiple tasks logically at the same time (not necessarily simultaneous).
Parallelism	Executing multiple tasks physically at the same time (on multi-core CPUs).

Java Thread Model Overview

Java provides built-in support for multithreading via:

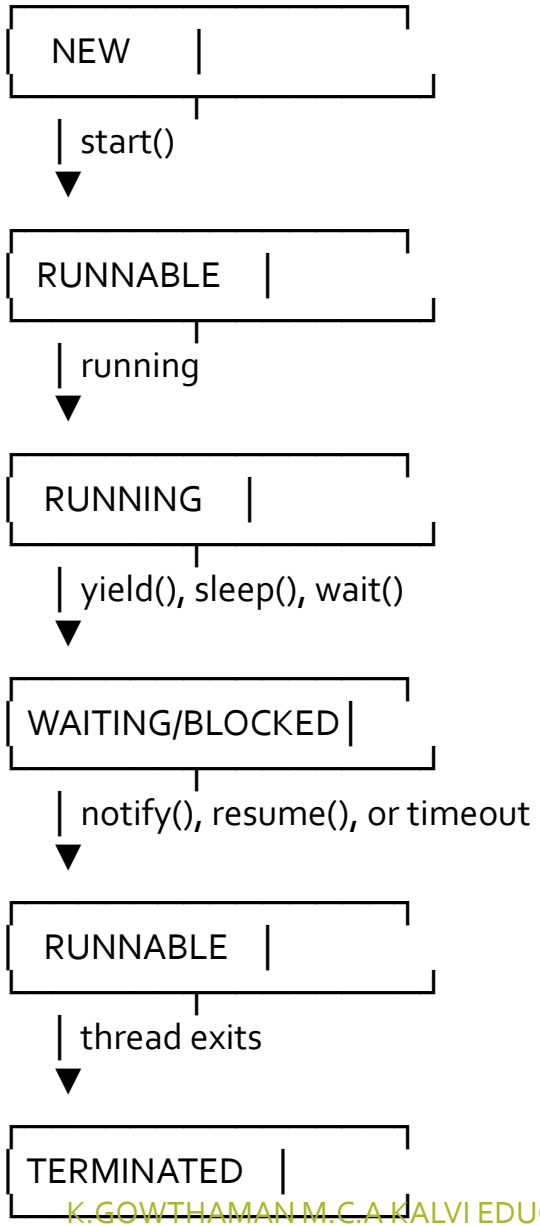
- java.lang.Thread class
- java.lang.Runnable interface
- java.util.concurrent package (advanced utilities)

Thread Life Cycle (Java Thread Model)

NEW → RUNNABLE → RUNNING → WAITING/BLOCKED → TERMINATED

Thread State	Description
NEW	Thread created but not yet started (new Thread()).
RUNNABLE	Thread ready to run but waiting for CPU time.
RUNNING	Thread currently executing.
WAITING / BLOCKED / TIMED_WAITING	Thread temporarily inactive (waiting for I/O, sleep, or monitor lock).
TERMINATED (DEAD)	Thread has completed execution.

Thread Life Cycle



Creating Threads in Java

1. Extending the Thread Class

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread running: " +
Thread.currentThread().getName());
   }

   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      t1.start();  // starts a new thread
   }
}
```

2. Implementing the Runnable Interface

```
class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Runnable thread: " +
Thread.currentThread().getName());
   }

   public static void main(String[] args) {
      Thread t = new Thread(new MyRunnable());
      t.start();
   }
}
```

Important Thread Methods

Method	Description
start()	Starts a new thread; calls run() internally
run()	Code executed by the thread
sleep(ms)	Suspends thread for given milliseconds
yield()	Temporarily pauses to let other threads execute
join()	Waits for another thread to finish
isAlive()	Checks if thread is still running
setPriority(int)	Sets thread priority (1–10)
interrupt()	Interrupts a sleeping or waiting thread

Advantages of Java Thread Model

✓ Better CPU utilization
✓ Faster response in GUI and server apps
✓ Easier program design using logical concurrency
✓ Non-blocking I/O and scalable server architectures

Thread Priorities

Every thread has a **priority** (integer 1 to 10).

Thread.MIN_PRIORITY = 1
Thread.NORM_PRIORITY = 5
Thread.MAX_PRIORITY = 10

Thread Synchronization

When multiple threads access shared resources, we must **synchronize** them to prevent inconsistent states.

```
class Counter {
   private int count = 0;

   synchronized void increment() {  // synchronized method
      count++;
   }
}
```

Or with synchronized block:

```
synchronized(object) {
  // critical section
}
```

Inter-Thread Communication

Threads can communicate using:

- wait() — causes thread to wait until notified.
- notify() — wakes up one waiting thread.
- notifyAll() — wakes up all waiting threads.

```
synchronized (this) {
   wait();     // release lock and wait
   notify();   // wake one waiting thread
}
```

Multithreading Example

```
class PrintJob extends Thread {
   private String name;
   PrintJob(String name) { this.name = name; }

   public void run() {
      for (int i = 1; i <= 5; i++)
         System.out.println(name + " printing page " + i);
   }

   public static void main(String[] args) {
      PrintJob t1 = new PrintJob("Job1");
      PrintJob t2 = new PrintJob("Job2");
      t1.start();
      t2.start();
   }
}
```

What is JDBC?

JDBC (Java Database Connectivity) is an API (Application Programming Interface) that allows Java applications to interact with **relational databases** such as MySQL, Oracle, PostgreSQL, etc.

It provides classes and interfaces for:

- Establishing a database **connection**
- **Executing SQL** queries
- **Processing results**

```
import java.sql.*;
```

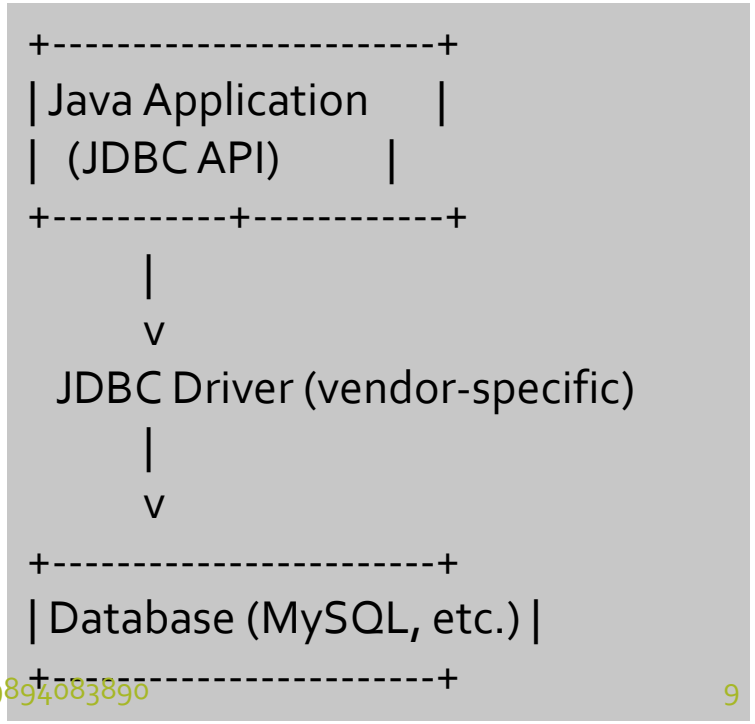
JDBC Architecture Overview

The JDBC architecture is based on **two-tier** and **three-tier** models.

Two-Tier Architecture

Client ↔ Database (direct connection)

- The Java application (client) directly communicates with the database using JDBC driver.
- Used in standalone applications.



✅ Simple and fast

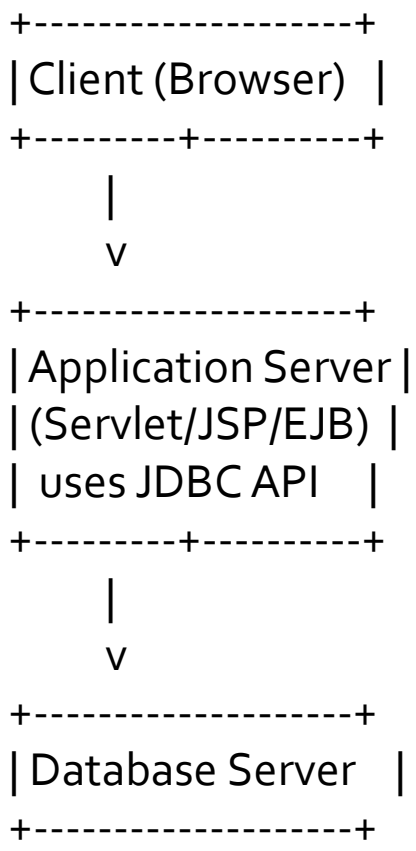
❌ Tight coupling between app and database

2) Three-Tier Architecture

Client ↔ Application Server ↔ Database

•The Java client connects to a middle-tier server (like an application server or servlet).

•The middle tier uses JDBC to communicate with the database.

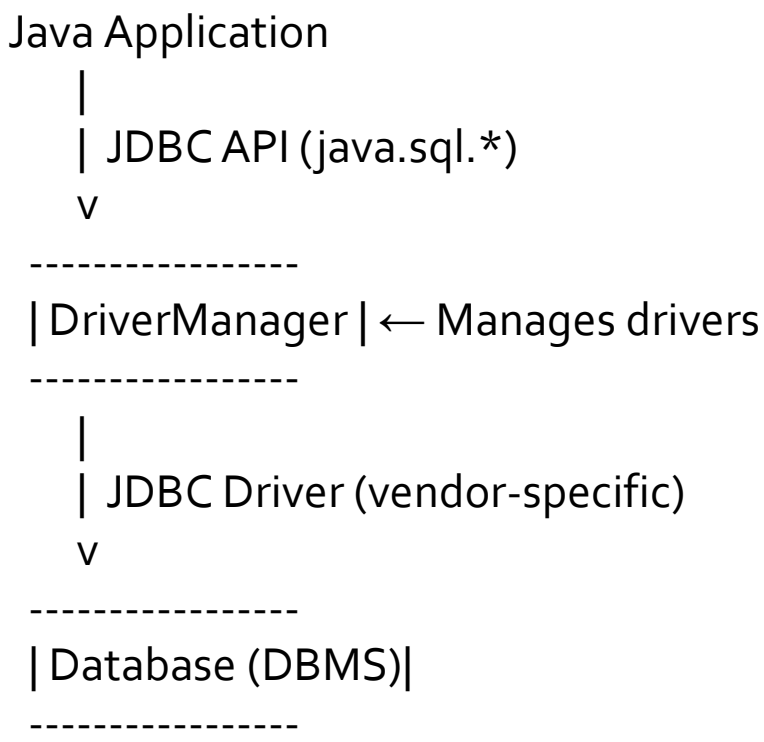


✓ Better scalability and security

✓ Useful for enterprise applications

Component	Description
JDBC API	Defines interfaces (Connection, Statement, ResultSet, etc.)
Driver Manager	Manages database drivers and connections
Driver	Translates JDBC calls into database-specific calls
Connection	Represents a session/connection with database
Statement	Used to execute SQL queries
ResultSet	Holds the results returned by a query

Jdbc Architectural Diagram



JDBC API Core Interfaces

Interface	Description
DriverManager	Loads and manages database drivers
Connection	Establishes connection to database
Statement	Executes static SQL queries
PreparedStatement	Executes parameterized SQL queries
CallableStatement	Executes stored procedures
ResultSet	Represents table of data returned from query
SQLException	Handles database access errors

```
import java.sql.*;

class JDBCExample {
   public static void main(String[] args) {
      try {
         // 1. Load driver (optional in newer Java versions)
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Establish connection
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root",
"password");

         // 3. Create statement
         Statement stmt = con.createStatement();

         // 4. Execute query
         ResultSet rs = stmt.executeQuery("SELECT * FROM
students");

         // 5. Process result
         while (rs.next()) {
            System.out.println(rs.getInt(1) + " " + rs.getString(2));
         }

         // 6. Close connection
         con.close();
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

DBC Drivers

JDBC drivers are the **bridge** between Java applications and specific databases.
They translate **JDBC method calls** into **DBMS-specific calls**.

Type	Name	Description	Example	Performance
Type 1	JDBC-ODBC Bridge Driver	Translates JDBC calls into ODBC calls	sun.jdbc.odbc.JdbcOdbcDriver	Slow (two levels of translation)
Type 2	Native-API Driver	Uses native client library (partly Java + native code)	Oracle OCI, DB2 driver	Faster, but platform dependent
Type 3	Network Protocol Driver	JDBC client communicates with middleware server	IDS Server, WebLogic	Good for enterprise systems
Type 4	Thin Driver (Pure Java)	Directly connects to DB using database's protocol	MySQL, PostgreSQL	Fastest, 100% Java, platform-independent

Type 1 – JDBC-ODBC Bridge

Java App → JDBC → ODBC Driver → Database

- **Converts JDBC calls into ODBC calls.**
- **Discontinued since Java 8.**

Type 4 – Thin Driver (Pure Java)

Java App → JDBC (Direct) → Database

- **Pure Java implementation (no native code).**
- **Most widely used today (e.g., MySQL Connector/J)**

Type 2 – Native API Driver

Java App → JDBC → Native API (C library) → Database

- Uses vendor-specific native code (not portable).
- Requires native client installation.

Type 3 – Network Protocol Driver

Java App → JDBC → Middleware Server → Database

- Middleware translates JDBC calls into DB protocol.
- Best for multi-tier web apps.

JDBC DRIVER TABLE

Feature	Type 1	Type 2	Type 3	Type 4
Implementation	JDBC–ODBC bridge	Native library	Middleware server	Pure Java
Platform Dependent	Yes	Yes	No	No
Performance	Low	Medium	High	Very High
Installation Needed	ODBC driver	Native DB client	Middleware	None
Usage Today	Deprecated	Rare	Enterprise	Most Common

Advantages of JDBC

- ✓ Database-independent API
- ✓ Supports multiple databases
- ✓ Secure and reliable
- ✓ Part of standard Java (no external libraries needed)
- ✓ Works with enterprise Java (Servlets, JSP, EJB)

JDBC Workflow

1. Load driver
2. Establish connection
3. Create statement
4. Execute query
5. Process results
6. Close connection

What Are CRUD Operations?

CRUD stands for:

C – Create → INSERT

R – Read → SELECT

U – Update → UPDATE

D – Delete → DELETE

These are the four basic database operations you can perform through JDBC.

JDBC Setup (Common to All)

1 Import Required Packages

```
import java.sql.*;
```

2 Load the Driver

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

3 Establish Connection

```
Connection con = DriverManager.getConnection(
  "jdbc:mysql://localhost:3306/testdb", "root", "password");
```

4 Create Statement / PreparedStatement

Use PreparedStatement for better security (prevents SQL Injection).

```
CREATE TABLE students (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50),
   age INT
);
```

CREATE (Insert Data)

```
INSERT INTO students (name, age) VALUES (?, ?);
```

```
String sql = "INSERT INTO students (name, age) VALUES (?, ?)";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, "Gowthaman");
ps.setInt(2, 21);

int rows = ps.executeUpdate();
if (rows > 0) {
   System.out.println("Record inserted successfully!");
}
```

executeUpdate() returns number of rows affected.

READ DATA

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM
students");

while (rs.next()) {
   int id = rs.getInt("id");
   String name = rs.getString("name");
   int age = rs.getInt("age");
   System.out.println(id + " | " + name + " | " + age);
}
```

UPDATE (Modify Data)

```
String sql = "UPDATE students SET age = ? WHERE id = ?";
PreparedStatement ps = con.prepareStatement(sql);
ps.setInt(1, 22);
ps.setInt(2, 1);

int rows = ps.executeUpdate();
if (rows > 0) {
   System.out.println("Record updated successfully!");
}
```

Delete Data

```
String sql = "DELETE FROM students WHERE id = ?";
PreparedStatement ps = con.prepareStatement(sql);
ps.setInt(1, 1);

int rows = ps.executeUpdate();
if (rows > 0) {
   System.out.println("Record deleted successfully!");
}
```

After each operation, **close the resources** to prevent memory leaks.

```
ps.close();
stmt.close();
con.close();
```

Or use **try-with-resources**:

```
try (Connection con = DriverManager.getConnection(...);
    PreparedStatement ps = con.prepareStatement(...)) {
   // perform operations
}
```

Complete CRUD Operation

```
import java.sql.*;

class JDBCCRUD {
   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // --- CREATE ---
         String insert = "INSERT INTO students (name, age) VALUES (?, ?)";
         PreparedStatement ps1 = con.prepareStatement(insert);
         ps1.setString(1, "Ravi");
         ps1.setInt(2, 20);
         ps1.executeUpdate();

         // --- READ ---
         ResultSet rs = con.createStatement().executeQuery("SELECT *
FROM students");
         while (rs.next()) {
            System.out.println(rs.getInt("id") + " " +
                               rs.getString("name") + " " +
                               rs.getInt("age"));
         }

         // --- UPDATE ---
         String update = "UPDATE students SET age = ? WHERE id = ?";
         PreparedStatement ps2 = con.prepareStatement(update);
         ps2.setInt(1, 25);
         ps2.setInt(2, 2);
         ps2.executeUpdate();

         // --- DELETE ---
         String delete = "DELETE FROM students WHERE id = ?";
         PreparedStatement ps3 = con.prepareStatement(delete);
         ps3.setInt(1, 3);
         ps3.executeUpdate();

         con.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Database Summary

Operation	SQL Command	JDBC Method	Object Used
Create	INSERT	executeUpdate()	PreparedStatement
Read	SELECT	executeQuery()	Statement / ResultSet
Update	UPDATE	executeUpdate()	PreparedStatement
Delete	DELETE	executeUpdate()	PreparedStatement

HTML and CSS, the two core technologies used for building and styling web pages.

HTML and CSS Overview

Technology	Purpose
HTML (HyperText Markup Language)	Used to structure content on the web — defines headings, paragraphs, links, images, forms, etc.
CSS (Cascading Style Sheets)	Used to style HTML — defines colors, fonts, layouts, and responsive designs.

HTML (HyperText Markup Language)

◆ **What is HTML?**

HTML is the **standard markup language** for creating web pages.
It uses **tags** to tell the browser how to display content.

Basic Structure of an HTML Document

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Welcome to HTML</h1>
  <p>This is a paragraph.</p>
</body>
</html>
```

Tag	Description
<!DOCTYPE html>	Declares the HTML5 document type
<html>	Root element of an HTML page
<head>	Contains metadata, title, links to CSS, etc.
<title>	Title shown in browser tab
<body>	Contains visible content of the web page

Tag	Purpose
<h1> – <h6>	Headings
<p>	Paragraph
<a href="">	Hyperlink
<img src="">	Image
, ,	Lists
<table>, <tr>, <td>	Tables
<form>	Form for user input
<input>, <button>, <select>	Form elements
,	Container elements for layout and styling

```
<form action="submit.jsp" method="post">
   <label>Name:</label>
   <input type="text" name="username">
   <br><br>
   <label>Password:</label>
   <input type="password" name="password">
   <br><br>
   <input type="submit" value="Login">
</form>
```

What is CSS?
CSS is used to **style HTML elements** — it controls layout, colors, fonts, spacing, and responsiveness.

Type	Description	Example
Inline CSS	Inside HTML tag using style attribute	Text
Internal CSS	Inside <style> tag in <head>	<style> p { color: blue; } </style>
External CSS	In a separate .css file linked via <link>	<link rel="stylesheet" href="style.css">

```
selector {
   property: value;
}

p {
   color: blue;
   font-size: 18px;
}
```

Selector	Description	Example
element	Selects HTML elements	p { color: red; }
.class	Selects elements with a specific class	.title { font-size: 24px; }
#id	Selects element with specific ID	#main { background-color: yellow; }
*	Selects all elements	* { margin: 0; }
element element	Descendant selector	div p { color: green; }

```
body {
  background-color: lightgray;
  color: navy;
  font-family: Arial, sans-serif;
}


.container {
  display: flex;
  justify-content: space-between;
}


div {
  float: left;
  width: 50%;
}
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Styled Page</title>
  <style>
    body {
      background-color: #fofofo;
      font-family: Arial;
    }
    h1 {
      color: darkblue;
    }
    p {
      color: #333;
      font-size: 18px;
    }
    .highlight {
      background-color: yellow;
    }
  </style>
</head>
<body>
  <h1>Welcome to Web Design</h1>
  <p>This is an example of <span class="highlight">HTML and CSS</span> working together!</p>
</body>
</html>
```

JavaScript Overview

JavaScript (JS) is a **client-side scripting language** used to make web pages **interactive, dynamic,** and **responsive**. It runs directly in the **browser** (like Chrome, Firefox, Edge) and can also be used on the **server** (with Node.js).

Role of JavaScript in Web Development

Technology	Role
HTML	Defines the structure/content of a webpage
CSS	Defines the appearance/style of the page
JavaScript	Adds interactivity and logic (behavior)

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
  <script>
    function showMessage() {
      alert("Hello, JavaScript is working!");
    }
  </script>
</head>
<body>
  <h1>Welcome</h1>
  <button onclick="showMessage()">Click Me</button>
</body>
</html>
```

Type	Description	Example
Internal JS	Inside <script> tags in HTML	<script> alert("Hi"); </script>
External JS	In a separate .js file	<script src="script.js"></script>
Inline JS	Directly inside an HTML tag	<button onclick="alert('Hi')">Click</button>

Variables

```
let name = "Gowthaman";   // Block-scoped
var age = 21;             // Function-scoped
const PI = 3.1416;        // Constant
```

Type	Example
Arithmetic	+, -, *, /, %
Assignment	=, +=, -=
Comparison	==, ===, !=, >, <
Logical	&&, || , !

Data Types

- **String → "Hello"**
- **Number → 42, 3.14**
- **Boolean → true, false**
- **Object → { name: "Gowthaman", age: 21 }**
- **Array → ["HTML", "CSS", "JS"]**
- **Null, Undefined**

Control Statements

If-Else

```
if (age >= 18) {
   console.log("Adult");
} else {
   console.log("Minor");
}
```

Switch

```
switch(day) {
   case 1: console.log("Monday"); break;
   case 2: console.log("Tuesday"); break;
   default: console.log("Invalid day");
}
```

Loops

```
for (let i = 1; i <= 5; i++) {
   console.log(i);
}

let j = 0;
while (j < 3) {
   console.log(j);
   j++;
}
```

Functions

```
function greet(name) {
   return "Hello, " + name;
}
console.log(greet("Gowthaman"));
```

You can also use **arrow functions** (modern ES6 feature):

```
const greet = (name) => `Hello, ${name}`;
```

Objects and Arrays

Object

```
let person = {
    name: "Ravi",
    age: 22,
    greet: function() {
        console.log("Hello " + this.name);
    }
};
person.greet();
```

Array

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits[0]);  // Apple
```

DOM Manipulation (Document Object Model)
JS can dynamically **change HTML content, style, and structure**.

```
<p id="demo">Old Text</p>
<button onclick="changeText()">Change Text</button>

<script>
function changeText() {
   document.getElementById("demo").innerHTML = "New Text!";
}
</script>
```

DOM methods:

Method	Purpose
getElementById()	Selects element by ID
getElementsByClassName()	Selects elements by class
querySelector()	Selects element by CSS selector
innerHTML	Changes HTML content
style.property	Changes CSS styles dynamically

Events in JavaScript

Event	Description
onclick	When user clicks an element
onmouseover	When user hovers mouse
onload	When page loads
onchange	When form input changes
onkeyup, onkeydown	Keyboard events

```
<input type="text" onkeyup="console.log('Typing...')">
```

Form-validation

```
<form onsubmit="return validateForm()">
   <input type="text" id="name" placeholder="Enter name">
   <input type="submit" value="Submit">
</form>

<script>
function validateForm() {
   let name = document.getElementById("name").value;
   if (name == "") {
      alert("Name must be filled out!");
      return false;  // prevent form submission
   }
   return true;
}
</script>
```

JavaScript with CSS (Dynamic Styling)

```
<p id="msg">Welcome</p>
<button onclick="changeColor()">Change Color</button>

<script>
function changeColor() {
   document.getElementById("msg").style.color = "red";
}
</script>
```

Modern JavaScript (ES6+ Features)

Feature	Example
let/const	Block-scoped variables
Arrow functions	const add = (a,b)=>a+b;
Template literals	`Hello ${name}`
Destructuring	let {name, age} = person;
Promises / async-await	Handle asynchronous code
Modules (import/export)	Organize code in files

Advantages of JavaScript

✅ Runs in browser (no installation)
✅ Makes pages interactive and dynamic
✅ Works with HTML and CSS
✅ Huge ecosystem (React, Angular, Node.js)
✅ Used for both front-end and back-end

```
<!DOCTYPE html>
<html>
<head>
  <title>JS Example</title>
  <style>
    body { text-align: center; margin-top: 100px; }
    #msg { color: blue; font-size: 22px; }
  </style>
</head>
<body>
  <h2 id="msg">Hello World!</h2>
  <button onclick="changeText()">Click Me</button>

  <script>
    function changeText() {
      document.getElementById("msg").innerHTML =
"JavaScript Changed Me!";
      document.getElementById("msg").style.color = "green";
    }
  </script>
</body>
</html>
```

Responsive Web Design using Bootstrap, one of the most important modern web development topics.

Responsive Web Design (RWD) Overview

✓ **Responsive Web Design** ensures that web pages **look good and function well on all devices**

desktops, tablets, and smartphones.
It uses **flexible layouts, images,** and **CSS media queries** to automatically adjust design.

Why Use Bootstrap for Responsive Design?
Bootstrap is a popular **CSS framework** developed by Twitter that helps create responsive, mobile-first websites **quickly and easily**.

Bootstrap provides:
- Predefined **responsive grid system**
- Ready-to-use **components** (buttons, forms, navbars, cards, etc.)
- **Utility classes** for spacing, colors, typography
- Built-in **media queries** for all device sizes
- Integration with **JavaScript** for interactive features (modals, dropdowns, etc.)

Adding Bootstrap to Your Project

You can add Bootstrap via **CDN (Content Delivery Network):**

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>Bootstrap Example</title>

 <!-- Bootstrap CSS -->
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">

 <!-- Bootstrap JS (optional, for interactivity) -->
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
 <h1 class="text-center text-primary">Welcome to Bootstrap!</h1>
</body>
</html>
```

Bootstrap Grid System

Bootstrap uses a **12-column grid system** that automatically adjusts based on screen size.

Screen Size Categories:

Device	Class Prefix	Screen Width
Extra Small	.col-	<576px
Small	.col-sm-	≥576px
Medium	.col-md-	≥768px
Large	.col-lg-	≥992px
Extra Large	.col-xl-	≥1200px

Grid Example

```
<div class="container">
  <div class="row">
    <div class="col-md-4 col-sm-6 col-12 bg-primary text-white p-3">Column 1</div>
    <div class="col-md-4 col-sm-6 col-12 bg-success text-white p-3">Column 2</div>
    <div class="col-md-4 col-sm-12 col-12 bg-danger text-white p-3">Column 3</div>
  </div>
</div>
```

This layout:

- Shows **3 columns** on large screens
- **2 columns** on tablets
- **1 column** on mobile devices

Responsive Utilities
Bootstrap provides many utility classes for responsive adjustments:

Utility	Example	Purpose
Margin/Padding	m-3, p-2	Space around/inside elements
Text Alignment	text-center, text-md-start	Align text responsively
Display	d-none d-md-block	Hide/show elements by screen size
Colors	text-primary, bg-success	Text/background colors
Width	w-50, w-100	Element width

Responsive Images

Use the .img-fluid class to make images resize automatically:

```
<img src="image.jpg" class="img-fluid" alt="Responsive image">
```

Responsive Navbar Example

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
 <div class="container-fluid">
  <a class="navbar-brand" href="#">MySite</a>
  <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav">
   <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
   <ul class="navbar-nav ms-auto">
    <li class="nav-item"><a class="nav-link active" href="#">Home</a></li>
    <li class="nav-item"><a class="nav-link" href="#">About</a></li>
    <li class="nav-item"><a class="nav-link" href="#">Contact</a></li>
   </ul>
  </div>
 </div>
</nav>
```

Responsive Cards Example

```
<div class="container my-4">
 <div class="row">
  <div class="col-md-4 mb-3">
   <div class="card">
    <img src="img1.jpg" class="card-img-top" alt="...">
    <div class="card-body">
     <h5 class="card-title">Card 1</h5>
     <p class="card-text">Some description here.</p>
    </div>
   </div>
  </div>
  <div class="col-md-4 mb-3">
   <div class="card">
    <img src="img2.jpg" class="card-img-top" alt="...">
    <div class="card-body">
     <h5 class="card-title">Card 2</h5>
     <p class="card-text">Some description here.</p>
    </div>
   </div>
  </div>
  <div class="col-md-4 mb-3">
   <div class="card">
    <img src="img3.jpg" class="card-img-top" alt="...">
    <div class="card-body">
     <h5 class="card-title">Card 3</h5>
     <p class="card-text">Some description here.</p>
    </div>
   </div>
  </div>
 </div>
</div>
```

Responsive Table

```
<div class="table-responsive">
 <table class="table table-bordered">
  <thead class="table-dark">
   <tr>
    <th>#</th>
    <th>Name</th>
    <th>Age</th>
   </tr>
  </thead>
  <tbody>
   <tr><td>1</td><td>Ravi</td><td>21</td></tr>
   <tr><td>2</td><td>Arun</td><td>22</td></tr>
  </tbody>
 </table>
</div>
```

Advantages of Bootstrap

- ✓ Speeds up development
- ✓ Responsive by default
- ✓ Predefined styles and layouts
- ✓ Mobile-first design
- ✓ Consistent design across browsers
- ✓ Easy customization

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-
scale=1">
 <title>Responsive Bootstrap Page</title>
 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.
min.css" rel="stylesheet">
</head>
<body>

<!-- Navbar -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
 <div class="container">
  <a class="navbar-brand" href="#">My Website</a>
  <button class="navbar-toggler" data-bs-toggle="collapse" data-bs-
target="#menu">
   <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="menu">
   <ul class="navbar-nav ms-auto">
    <li class="nav-item"><a class="nav-link" href="#">Home</a></li>
    <li class="nav-item"><a class="nav-link" href="#">About</a></li>
    <li class="nav-item"><a class="nav-link" href="#">Contact</a></li>
   </ul>
  </div>
 </div>
</nav>

<!-- Content Section -->
<div class="container mt-5">
 <div class="row text-center">
  <div class="col-md-4 mb-3">
   <div class="p-3 bg-primary text-white rounded">Column 1</div>
  </div>
  <div class="col-md-4 mb-3">
   <div class="p-3 bg-success text-white rounded">Column 2</div>
  </div>
  <div class="col-md-4 mb-3">
   <div class="p-3 bg-danger text-white rounded">Column 3</div>
  </div>
 </div>
</div>

<!-- Footer -->
<footer class="bg-dark text-white text-center py-3 mt-4">
 &copy; 2025 My Website
</footer>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bun
dle.min.js"></script>
</body>
</html>
```

React.js, one of the most powerful and widely used **JavaScript libraries** for building modern, fast, and dynamic web applications.

What is React.js?
React.js is a **JavaScript library** developed by **Facebook (now Meta)** for building **interactive, component-based user interfaces** — especially **Single Page Applications (SPAs)**.

Feature	Description
Component-Based	UI is built using reusable components
Declarative	You describe what UI should look like, React handles how
Virtual DOM	Efficiently updates only changed parts of the UI
Unidirectional Data Flow	Data flows in one direction (top-down)
JSX Syntax	Combines HTML and JavaScript in one file
Fast Rendering	Uses diffing algorithm and Virtual DOM
React Hooks	Enable state and lifecycle management in functional components

Setting Up React

Option 1: Using create-react-app (recommended for beginners)

```
npx create-react-app myapp
cd myapp
npm start
```

Option 2: Add React via CDN (for small demos)

```
<!DOCTYPE html>
<html>
<head>
 <title>React Demo</title>
 <script
src="https://unpkg.com/react@18/umd/react.development.
js" crossorigin></script>
 <script src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js" crossorigin></script>
 <script src="https://unpkg.com/babel-
standalone@6/babel.min.js"></script>
</head>
<body>
 <div id="root"></div>

 <script type="text/babel">
  function App() {
   return <h1>Hello, React!</h1>;
  }
  ReactDOM.render(<App />,
document.getElementById('root'));
 </script>
</body>
</html>
```

React Component Structure

A **component** is a reusable, independent piece of UI (like a button, form, or navbar).

Example (Functional Component):

```
function Welcome() {
  return <h1>Welcome to React</h1>;
}
export default Welcome;
```

Example (Class Component):

```
import React from "react";

class Welcome extends React.Component {
  render() {
    return <h1>Welcome to React (Class Component)</h1>;
  }
}
export default Welcome;
```

JSX (JavaScript XML)
JSX allows you to **write HTML inside JavaScript**.

```
const name = "Gowthaman";
const element = <h1>Hello, {name}!</h1>;
```

JSX gets compiled into JavaScript:

```
React.createElement("h1", null, "Hello, Gowthaman!");
```

Rendering Elements
In index.js:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Props (Component Properties)
Props are **read-only attributes** passed from parent to child.

```
function Welcome(props) {
  return <h2>Hello, {props.name}</h2>;
}

function App() {
  return (
    <>
      <Welcome name="Ravi" />
      <Welcome name="Arun" />
    </>
  );
}
```

State in React
State is data that **changes over time** — used for dynamic UIs.

```
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count +
1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

Handling Events

```
function ButtonClick() {
  function handleClick() {
    alert("Button Clicked!");
  }

  return <button onClick={handleClick}>Click
Me</button>;
}
```

Conditional Rendering

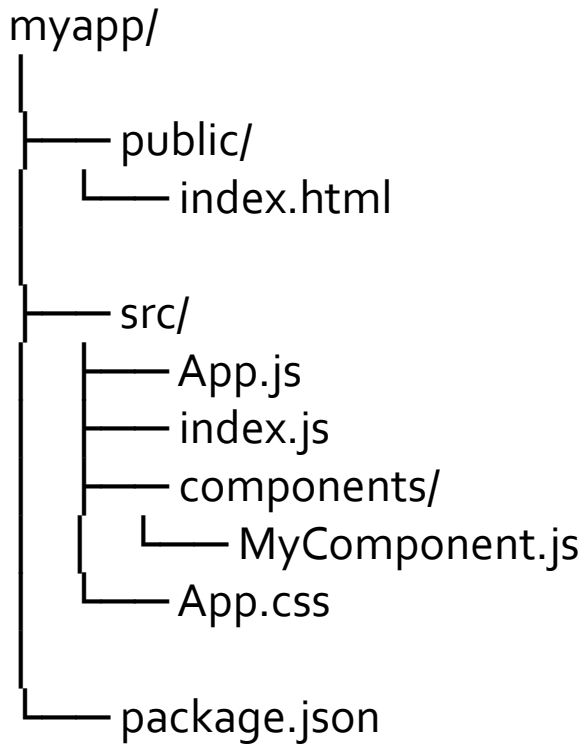
```
function UserGreeting(props) {
  return props.isLoggedIn ? <h2>Welcome Back!</h2> :
<h2>Please Sign In</h2>;
}
```

Lists and Keys

```
const fruits = ["Apple", "Banana", "Cherry"];

function FruitList() {
  return (
    <ul>
      {fruits.map((fruit, index) => <li key={index}>{fruit}</li>)}
    </ul>
  );
}
```

React Folder Structure (create-react-app)



React Component Communication

Direction	Technique
Parent → Child	Props
Child → Parent	Callback functions
Global	Context API or Redux

Advantages of React

✓ Fast and efficient (Virtual DOM)
✓ Component-based structure
✓ Reusable code and easy maintenance
✓ Large community and ecosystem
✓ Can integrate with backend APIs easily

Thank You