Backend Development with NodeJS

AMIT DHOJU

Collaboration With :

WorldLink
CORPORATE SOCIAL RESPONSIBILITY

THE ALGORITHM
Tech for inclusive democracy

Mr. Amit Dhoju
Sr. Software Engineer

TOC

Day 1: JavaScript Fundamentals & Node.js Introduction

Section 1: JavaScript Basics

Javascript Scope

- scope
- closure
- hoisting
- variable declarations
- functions
- data Types

AMIT DHOJU

Javascript Scope

- global scope
- function scope
- block scope

```
let globalVar = "Global"; // Global scope

function test() {
  let localVar = "Function Scope"; // Function scope
  if (true) {
    let blockVar = "Block Scope"; // Block scope
    console.log(blockVar); // ✅ Accessible
  }
  // console.log(blockVar); ❌ Error
}

console.log(globalVar); // ✅
```

AMIT DHOJU

Javascript Closure

- A closure happens when a function remembers variables from its outer scope, even after that outer function has finished running.
- Why?
  - Because in JavaScript, functions carry their scope with them.

```
function outer() {
  let counter = 0; // Outer
  return function inner() {
    counter++;
    console.log(counter);
  };
}

const increment = outer();
increment(); // 1
increment(); // 2
increment(); // 3
```

AMIT DHOJU

Javascript Hoisting

- hoisting means JavaScript moves **declarations** (not initializations) to the top of their scope **before execution**.

For variables:

- var is hoisted **and** initialized with undefined.
- let and const are hoisted but **not initialized** (Temporal Dead Zone).

For functions:

- **Function declarations** are hoisted completely (you can call them before they're defined).
- **Function expressions** are hoisted like variables.

AMIT DHOJU

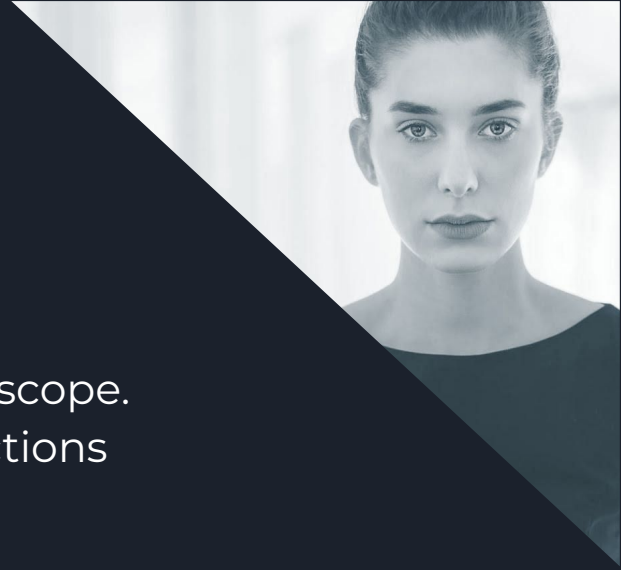
Javascript Hoisting Example

```
console.log(a); // undefined (var is hoisted)
var a = 5;

sayHi(); // ✅ Works
function sayHi() {
  console.log("Hi!");
}

// greet(); ❌ Error
const greet = function() {
  console.log("Hello!");
};
```

AMIT DHOJU

In Brief

- **Scope** decides where variables exist.
- **Closure** allows a function to remember its scope.
- **Hoisting** changes when variables and functions become available.



AMIT DHOJU

Variable declarations

- var
- let
- const
- which one to use?

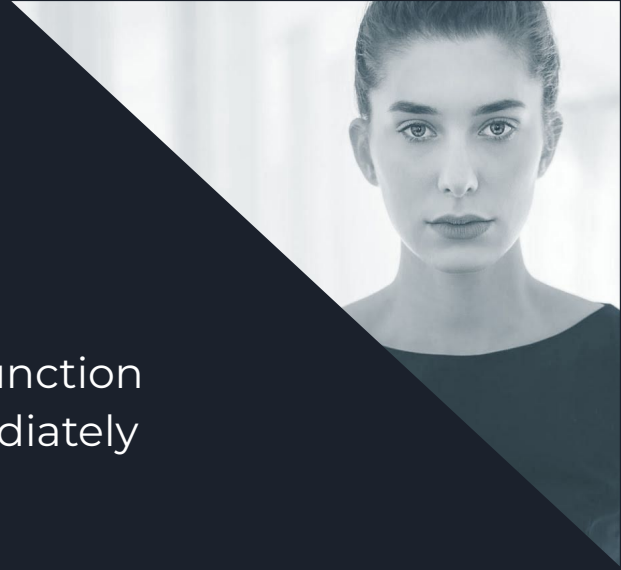
AMIT DHOJU

Javascript Functions

- regular functions (link)
- arrow functions (link)
- high order functions

AMIT DHOJU

Regular functions

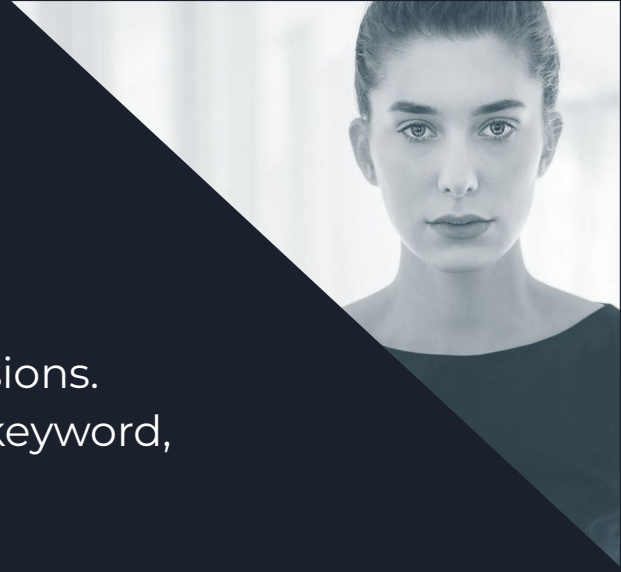


- javaScript functions are defined with the function
- declared functions are not executed immediately
- executed after its been called
- Link
- example

```
function functionName(parameters) {
  // code to be executed
}
```

AMIT DHOJU

Arrow functions



- allows a shorter syntax for function expressions.
- no need the function keyword, the return keyword, and the curly brackets.
- executed after its been called
- Link
- example

Before Arrow:

```
let hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
let hello = () => {
  return "Hello World!";
}
```

AMIT DHOJU

High order functions

- A Higher-Order Function (HOF) in JavaScript is simply a function that does at least one of these
  - Takes another function as an argument
  - Returns a function as its result
- Example
  - Array methods: map, filter, reduce, forEach, sort

```
function multiplier(factor) {
  return function (number) {
    return number * factor;
  };
}

const double = multiplier(2);
console.log(double(5)); // 10
```

AMIT DHOJU

Asynchronous Programming

1. Callbacks: ***"I will call back later!"***
2. Asynchronous : “***I will finish later!***”
3. Promises: "***I Promise a Result!***"
4. Async/await: “***i will make promises easier to write***“

AMIT DHOJU

Callbacks Functions

- function passed as an argument to another function
- allows a function to call another function
- can run after another function has finished
- simplified use.
- Link: W3 schools Callbacks

AMIT DHOJU

Asynchronous Functions

- functions running in parallel with other functions are called asynchronous
- A good example is JavaScript setTimeout()
- Right: setTimeout(myFunction, 3000);
- LInk: [W3 schools Async]

AMIT DHOJU

Promises Functions

- "Producing code" is code that can take some time
- "Consuming code" is code that must wait for the result
- object that links Producing code and Consuming code
- A JavaScript Promise object can be:
  a. Pending
  b. Fulfilled
  c. Rejected
- Links: W3 schools Promise

AMIT DHOJU

Async/await Functions

- **async** makes a function return a Promise
- **await** makes a function wait for a Promise
- The await keyword makes the function pause the execution and wait for a resolved promise before it continues:
- Link: W3Schools Async/await

AMIT DHOJU


Any Questions??
AMIT DHOJU

Section 2: Introduction to **Node.js**

Some history facts.

- 1995 – Birth of JavaScript
  - Created by Brendan Eich at Netscape in just 10 days.
  - Originally called Mocha, then LiveScript, and finally JavaScript.
  - Designed to make web pages interactive inside the Netscape Navigator browser.
- 2009 – Creation
  - Created by Ryan Dahl.
  - Used Google's V8 JavaScript Engine (from Chrome) to run JS outside the browser.
  - Introduced non-blocking, event-driven I/O, perfect for scalable network apps.

What is Node.js

- Node.js is an open-source, cross-platform runtime environment that lets you run JavaScript outside the browser.

- Built on Google's V8 JavaScript Engine (the same engine powering Chrome).

- Uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient — perfect for real-time applications.

Why Node.js?

- One language for full stack
  - JavaScript on both frontend and backend.
- High performance
  - Non-blocking I/O handles thousands of requests without slowing down.
- Rich ecosystem
  - npm has millions of packages ready to use.
- Scalability
  - Ideal for microservices, APIs, streaming apps, chats, and more.

AMIT DHOJU

Installing

- Visit https://nodejs.org
  - LTS (Long-Term Support) version → stable for most users
  - Current version → latest features (may be less stable)
- Verify installation
  - `node -v` # Check Node.js version
  - `npm -v` # Check npm version
  -

AMIT DHOJU

Running Node.js code

- One language for full stack
  - JavaScript on both frontend and backend.
- High performance
  - Non-blocking I/O handles thousands of requests without slowing down.
- Rich ecosystem
  - npm has millions of packages ready to use.
- Scalability
  - Ideal for microservices, APIs, streaming apps, chats, and more.

AMIT DHOJU

Running Node.js/Hello World

- Run in console
- Run a function in browser console
- Run a .js file

Understanding the Event Loop

- Node.js uses single-threaded, non-blocking architecture.
- The Event Loop is the heart of Node.js — it lets Node handle many tasks asynchronously.
- This means Node can handle thousands of requests without creating new threads.

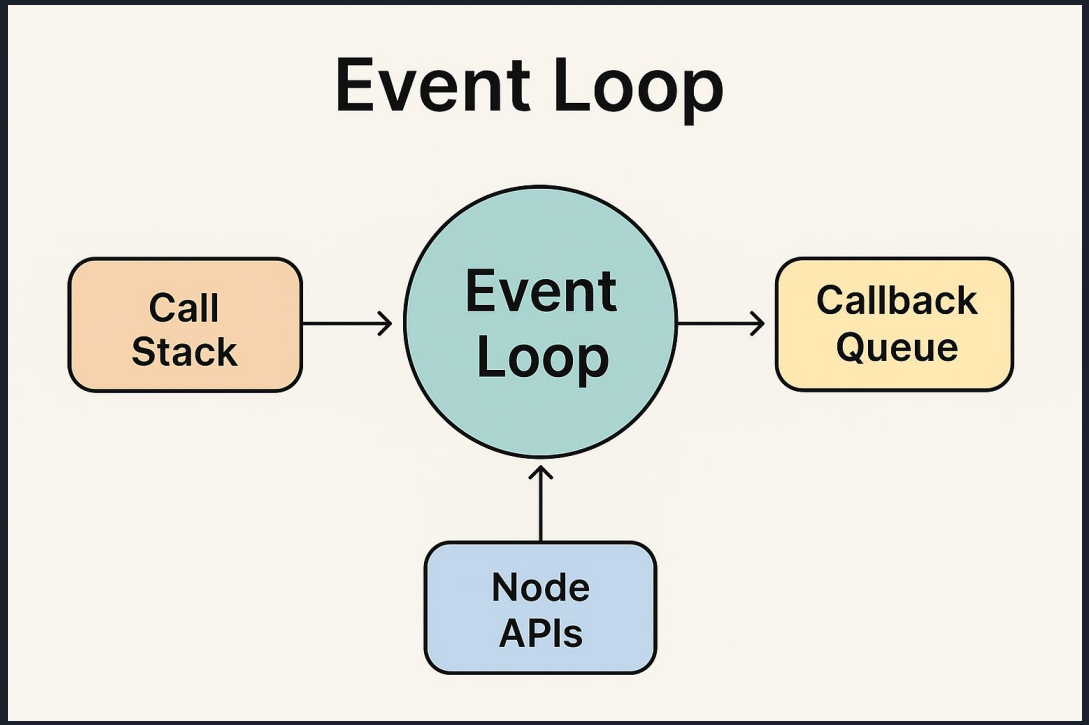
AMIT DHOJU

Event Loop

- The mechanism that allows Node.js to handle many tasks at once without creating multiple threads.
- Single thread runs JavaScript code.
- When you do something slow (like reading a file or calling an API), Node hands it off to the system to do in the background.
- Once the task is done, the callback (or promise) is queued.
- The Event Loop checks if the main thread (call stack) is free — if yes, it runs the callback.

AMIT DHOJU

Event Loop



AMIT DHOJU

How Event Loop Works?

- Call Stack executes JS code.
- When Node encounters async tasks (e.g., file read, HTTP request), it sends them to the Node APIs.
- When tasks complete, callbacks are moved to the Callback Queue.
- Event Loop checks if the Call Stack is free → moves queued callbacks into the stack to execute.

AMIT DHOJU

Examples

Code

```
console.log("Start");

setTimeout(() => {
  console.log("Timeout finished");
}, 2000);

console.log("End");
```

Output

```
Start
End
Timeout finished
```

AMIT DHOJU

Hands-On Exercise: Print triangle pattern


Any Questions??
AMIT DHOJU

Section 3: Node.js Core Modules

Node Core Modules

1. File System (fs)File operations like reading, writing and manipulating files
2. Path (path)Utilities for working with file and directory paths
3. HTTP (http)Create HTTP servers and make HTTP requests
4. HTTPS (https) – Secure version of HTTP using TLS/SSL

AMIT DHOJU

Node Core Modules 2

1. Console (console) – Provides simple logging methods like console.log, console.error, console.table

2. Timers (timers) – Functions for scheduling code execution: setTimeout, setInterval, setImmediate

3. Stream (stream) – Working with streaming data (readable, writable streams, files)

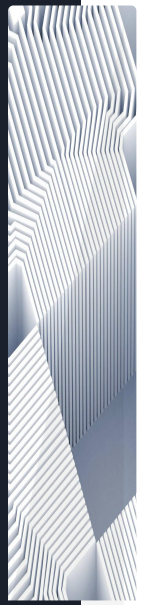
4. Buffer (buffer) – Handling binary data

AMIT DHOJU

Library VS Framework??

Library

- Pre-written code for specific tasks.
- You control when and how it’s used.
- Example: npm



Framework

- Defines overall structure
- It controls show things flow
- Examples: expressJs

AMIT DHOJU

Section 4:
Node.js Frameworks:
Express.js

Some Node js Framework

1. ExpressJS
2. NestJS
3. KoaJS
4. AdonisJS
5. FastifyJS
6. TotalJS

AMIT DHOJU

Express.js

1. Express.js is a minimalist, fast, lightweight and flexible web framework for Node.js.

2. It simplifies building web servers and APIs by providing a clean set of features like routing, middleware support, and more.

3. It’s the most popular framework in the Node.js ecosystem for backend development.

4. Link

AMIT DHOJU

Working

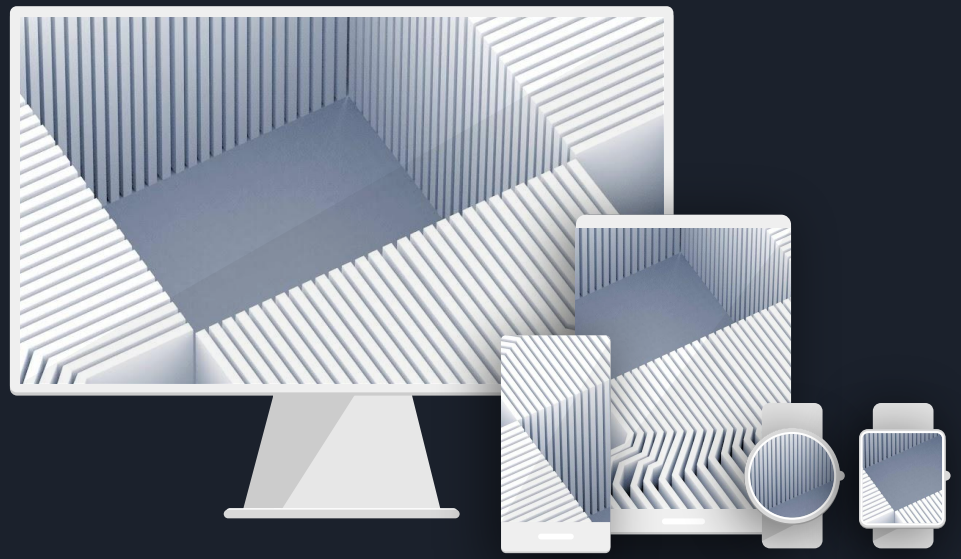
1. Listens for requests (http request from clients)
2. Parse requests (extract data from requests)
3. Match routes (find correct route like /home, /about)
4. Send response to client (returns a suitable response eg, JSON, html etc.

Installing Express JS

npm install express

Hands-On Exercise:

Express app setup



AMIT DHOJU

Overview

- ***express()-*** creates instance of the xpress application
- ***app.get(path, callback)*** - handles get requests
- ***app.post(path, callback)*** - handles post requests
- ***app.listen(port,callback)*** - start server at the the port

Section 5:
Git / Github

Git

- Version control system (VCS). It helps you track changes in your code or files over time.

- How does it work?
  You use Git on your local computer to save snapshots (commits) of your project as you work on it.

- What is it used for?
  Keeping track of code history
  Working with branches (different versions)
  Collaborating by merging changes

- Is it a website? - No, Git is software you install and run locally or on servers.

AMIT DHOJU

Github

- Web-based platform built on top of Git.

- What does it provide?
  A place to host your Git repositories online
  Tools for collaboration: Pull requests, issues, code reviews
  Social features like profiles, following, stars
  CI/CD integrations, project management, wikis

- Is it the same as Git?- No. GitHub uses Git for version control but adds a user-friendly interface, many services.

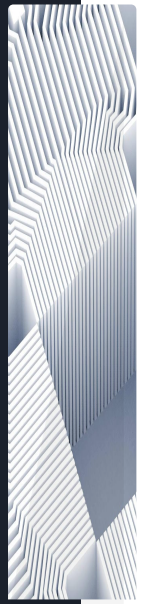
- Are there alternatives?- Yes! Examples: GitLab, Bitbucket, Azure DevOps — all are Git repository hosting platforms with their own features

AMIT DHOJU

Git vs Github

Git

- Version control system
- Track/manage code locally
- Locally on your computer
- Provides git commands & workflow
- Git is like docs file; write, edit docs



Github

- Online Git repository hosting
- Share, collaborate, manage code online
- Accessed through a web browser
- Provides web interface, collaboration tools
- Github is like google drive; you can upload, share and access from anywhere.

AMIT DHOJU

Previously

- Javascript basics (scope, variables, functions)
- Asynchronous programming
- Nodejs installation and introduction
- Nodejs modules and it usage
- Express JS introduction and a demo application

Day 2: Deep Dive on HTTP Methods & Express.js

- Server Architecture
- HTTP Request Methods
- Different URL Part
- Express project setup
- Debugging Techniques

Section 1:
Server Architecture

Server Architecture

- Stateful Server
- Stateless Server

AMIT DHOJU

Stateful Server

- Keeps track of the client’s state between requests.
- The server stores session data (e.g., in memory or in a session store) and
- uses it to serve future requests.
- State is stored on the server (e.g., in memory, sessions, database).

- Example:
  - You log in → server keeps your session in memory → you browse pages without logging in again.
  - Online banking sessions — server remembers your login and account state.
  - Multiplayer game servers tracking player positions.

AMIT DHOJU

Stateful Server: Pros & Cons

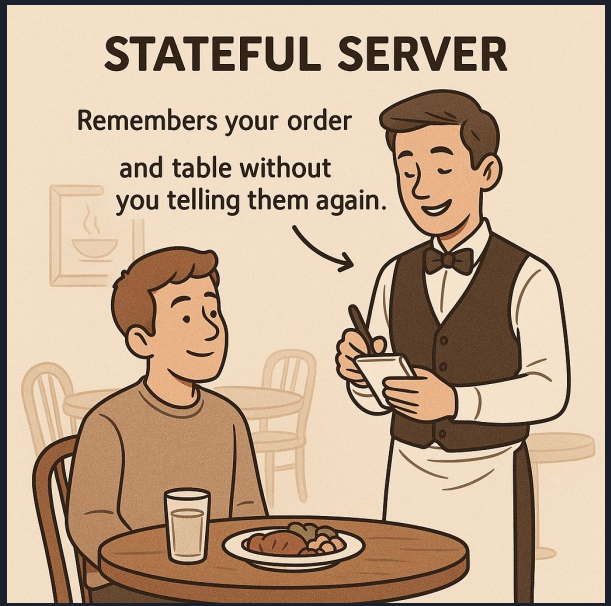
- Pros:
  - Good for long, interactive sessions (e.g., online banking, multiplayer games)
  - Easier to manage ongoing user interactions.
  - Less data sent with each request.

- Cons:
  - Harder to scale — need to share session state between servers (all servers need the same session data).
  - More memory usage on the server.

AMIT DHOJU

Stateful Server: Analogy

- Like a waiter who remembers your order and table without you telling them again.



AMIT DHOJU

Stateless Server

- Does not remember client state between requests. Each request is independent..
- **How it works**: Client must send all necessary context (e.g., authentication token, request data) with every request.
- State is stored on the client (e.g., in cookies, tokens)

- Example:
  - REST APIs using JWT tokens for authentication.
  - REST API where you send your authentication token with every request.

AMIT DHOJU

Stateful Stateless: Pros & Cons

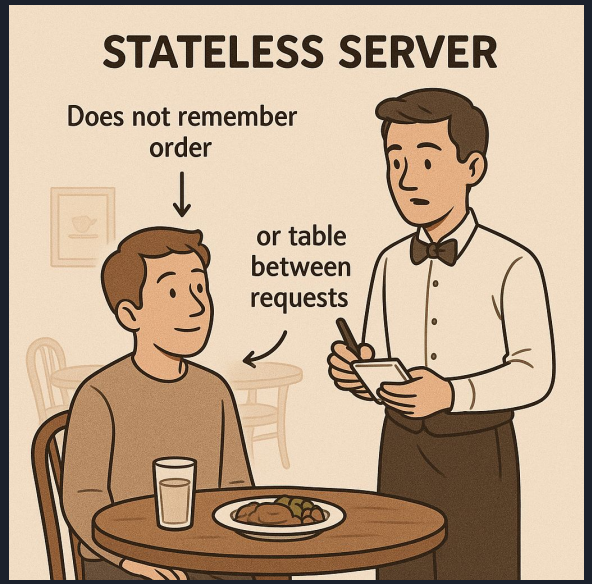
- Pros:
  - Easy to scale horizontally (add more servers without syncing state).
  - Simpler architecture.
  - More fault-tolerant.

- Cons:
  - Client sends more data with each request.
  - Can be less efficient for continuous sessions.

AMIT DHOJU

Stateful Stateless: Analogy

- Like ordering food at a fast-food counter — every order is independent, you must give your details every time..



AMIT DHOJU

Server API Style

- SOAP (Simple Object Access Protocol)
- REST (Representational State Transfer)
- GraphQL

SOAP API

- **Principle**: XML-based protocol for exchanging structured information.
- **Uses**: Often in enterprise applications (banking, government systems).
- More strict structure, with a WSDL (Web Services Description Language).

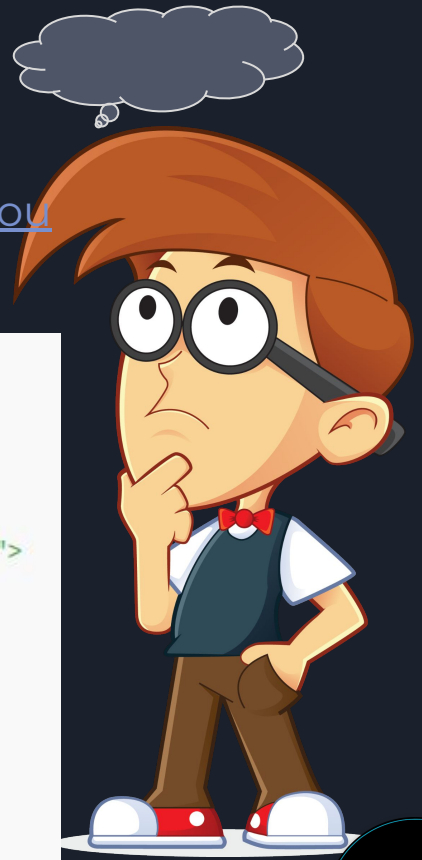
- **Pros**:
  - Strong standards and strict structure.
  - Built-in error handling and security
- **Cons**:
  - Verbose XML format.
  - More overhead compared to REST/GraphQL.

AMIT DHOJU

SOAP API : Example

- http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL

```
fetch('http://localhost:8000/wsdl', {
  method: 'POST',
  headers: { 'Content-Type': 'text/xml' },
  body: `
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <GetUser/>
  </soapenv:Body>
</soapenv:Envelope>`
})
  .then(res => res.text())
  .then(console.log);
```



AMIT DHOJU

REST API

- **Principle:** Represent resources via unique URLs, use standard HTTP methods.
- **Methods:** GET, POST, PUT, DELETE, PATCH.
- **Data Format:** JSON (most common), XML possible.

- **Pros:**
  - Simple, widely adopted.
  - Stateless by default.
- **Cons**:
  - Can result in over-fetching or under-fetching data.

REST API : Example

- https://jsonplaceholder.typicode.com/

```
fetch('http://localhost:3000/users')
  .then(res => res.json())
  .then(data => console.log(data));
```

AMIT DHOJU

GraphQL API

- **Principle:** Client specifies exactly what data it needs.
- **Endpoints:** Usually a single /graphql endpoint.
- **Data Format:** JSON
- Reduces over-fetching and under-fetching.
- **Pros:**
  - No over-fetching — returns exactly what you request.
  - Strongly typed schema.
- **Cons**:
  - More complex setup.
  - Caching can be trickier than REST.

AMIT DHOJU

GraphQL API : Example

- https://countries.trevorblades.com/

```
{
  countries {
    capital
    name
  }

  continents{
    name
code
  }
}
```

AMIT DHOJU

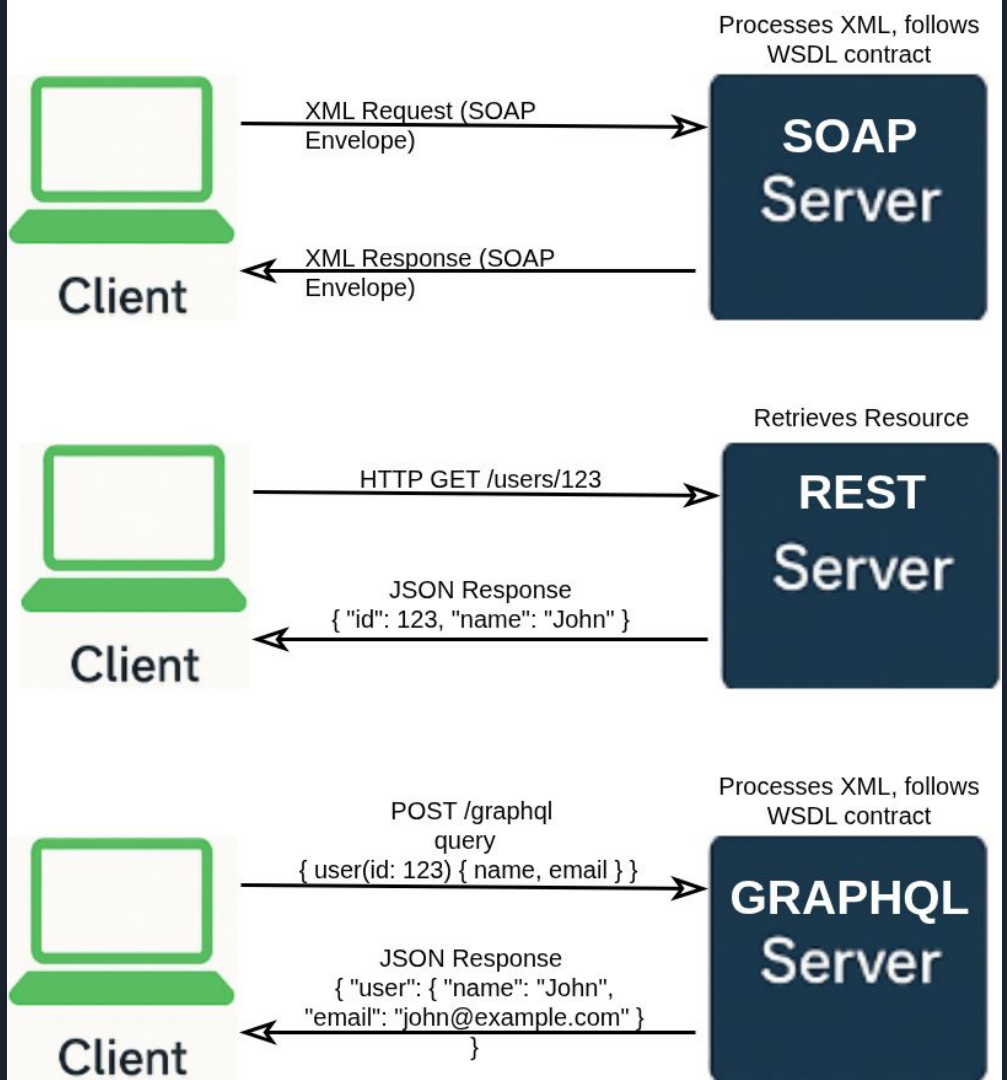
SOAP vs REST vs GRAPHQL

- **SOAP –** Like a government post office: strict rules, fixed format, reliable but heavy.

- **REST** – Like ordering from a restaurant menu: simple, predefined options, sometimes extra you don’t need (eg:fish with fries).

- **GraphQL –** Like a custom smoothie bar: you ask for exactly what you want in one go (eg: content and flavour).

AMIT DHOJU


Processes XML, follows WSDL contract
XML Request (SOAP Envelope)
SOAP Server
XML Response (SOAP Envelope)
Client
Retrieves Resource
HTTP GET /users/123
REST Server
JSON Response
{ "id": 123, "name": "John" }
Client
Processes XML, follows WSDL contract
POST /graphql
query
{ user(id: 123) { name, email } }
GRAPHQL Server
JSON Response
{ "user": { "name": "John",
"email": "john@example.com" }
}
Client

AMIT DHOJU

Any Questions??

AMIT DHOJU

Section 2: HTTP Request Methods

HTTP Request Methods

- GET - Retrieve Data
- POST - Create New Data
- PUT - Update Entire Resource
- PATCH - Partial Updates
- DELETE - Remove Data

AMIT DHOJU

HTTP GET

- Used to request data from the server.
- Does not change data on the server.
- Typically used to fetch resources.

```
app.get('/users', (req, res) => {
  // Pretend we get users from database
  const users = [{ id: 1, name: 'Amit' }, { id: 2, name: 'John' }];
  res.json(users);
});
```

AMIT DHOJU

HTTP POST

- Used to send data to the server to create a new resource.
- Data sent usually in request body (JSON, form data, etc.).

```
app.post('/users', (req, res) => {
  const newUser = req.body;  // Assuming middleware like express.json() is used
  // Save newUser to database (mock)
  newUser.id = 3; // Example ID assigned
  res.status(201).json(newUser);
});
```

AMIT DHOJU

HTTP PUT

- Replaces the entire resource with new data.
- Requires sending the full updated object.

```
app.put('/users/:id', (req, res) => {
  const userId = req.params.id;
  const updatedUser = req.body;
  updatedUser.id = userId;
  // Replace user in database (mock)
  res.json(updatedUser);
});
```

AMIT DHOJU

HTTP PATCH

- Updates only specified fields of a resource.
- Does not require sending the full object.

```
app.patch('/users/:id', (req, res) => {
  const userId = req.params.id;
  const partialUpdate = req.body;  // e.g., { name: 'New Name' }
  // Update user partially in database (mock)
  const updatedUser = { id: userId, name: partialUpdate.name || 'Existing Name' };
  res.json(updatedUser);
});
```

AMIT DHOJU

HTTP DELETE

- Deletes the resource identified by the URL.

```
app.delete('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Delete user from database (mock)
  res.status(204).send();  // 204 No Content means successful deletion
});
```

AMIT DHOJU

HTTP SUMMARY

Method	Action	Idempotent	Request Body	Example Usage
GET	Retrieve data	✅	❌	Get user list or detail
POST	Create new data	❌	✅	Add a new user
PUT	Update whole data	✅	✅	Replace user with new data
PATCH	Partial update	❌	✅	Change only some user fields
DELETE	Delete resource	✅	❌	Remove a user

AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Section 3: Different URL Part

URL Parts

- Query Parameters (?key=value)
- Route Parameters (:id)
- Hash Fragment (#something)

```
https://example.com/users/123?sort=asc&page=2#contact
          |       |       |            |     |
      protocol   path   route param   query  fragment
```

Query Parameters (?key=value)

- **Syntax**: ?param1=value1&param2=value2
- **Purpose**: Send extra information to the server, usually for filtering, searching, or pagination.
- **Example:** https://example.com/products?category=boot&color=red
- **Notes**:
  - Appear after ?.
  - Multiple parameters are separated by &.
  - Order doesn't matter for most APIs (but can matter if the backend is coded that way).

AMIT DHOJU

Route Parameters (:id)

- **Syntax (in backend frameworks)** : /users/:id
- **Purpose**: Represent dynamic parts of the path.
- **Example:** https://example.com/users/123
- Notes:
  - Here **123** is the id of the user.
  - Often used for resource identification.
  - The colon : is only in the route definition (e.g., Express.js), not in the actual URL.

AMIT DHOJU

Hash Fragment (#something)

- **Syntax** : #section-name
- **Purpose**:
  - On the web: Scrolls to a specific part of the page (anchor).
  - In SPAs (Single Page Apps): Can be used for client-side routing.
- **Example:** https://example.com/about#team
- Notes:
  - Loads about page and jumps to the team section..
  - Never sent to the server — handled entirely by the browser.
  - Common in old hash-based routing: /#/home.

AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Section 4:
Express project setup

Express project setup

- Manual setup : using npm initialization
- Fast setup : using express generator

Project setup: manual setup

- Create all necessary files and folder manually

1. Initialize project with npm: `***npm init -y***`
2. Install Express: npm install express
3. Create your project structure manually:
    a. /project-root
    b. /routes
    c. /controllers
    d. /middleware
    e. /views
    f. app.js

AMIT DHOJU

Project setup: manual setup

4. Use nodemon for easier development:
   `***npm install --save-dev nodemon***`
5. Add a script in package.json:
   "scripts": {
   "dev": "nodemon app.js"
   }
6. Run development server: `***npm run dev***`

AMIT DHOJU

Project setup: Fast setup

- Using Express Generator for full setup

1. Run express generator: `***npx express-generator***`
2. This scaffolds the project with routes, views, public folder, and basic structure.
3. Install dependencies: `***npm install***`
4. Use nodemon (install if needed):
   `***npm install --save-dev nodemon***`

AMIT DHOJU

Project setup: Fast setup

5. Add dev script to package.json

```
"scripts": {
  "dev": "nodemon ./bin/www"
}
```

6. Run development server: `` `npm run dev` ``

AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Section 5: Express project directory description

```
/project-root
|
├── /routes
|     └── index.js
|
├── /controllers
|     └── userController.js
|
├── /middleware
|     └── logger.js
|
├── /views
|     └── index.pug
|
├── /public
|     ├── /css
|     └── /js
|
├── app.js
├── package.json
└── README.md
```

Express project folders

- **/routes**
  - **Purpose**: Define the app routes and map routes to controller functions.
  - Keeps routing logic organized and separate from business logic.
  - **Example:** /routes/index.js

AMIT DHOJU

Express project folders

- **/controllers**
  - Optional(better if used)
  - **Purpose**: Contains the business logic and handlers for routes.
  - Keeps route files clean and focused on routing only.
  - Can handle DB calls, validations, or other logic.
  - **Example:** /controllers/userController.js

Express project folders

- **/middleware**
  - **Purpose**: Contains custom middleware functions to handle repeated tasks
  - Middleware can process requests before they reach routes/controllers.
  - **Example:** /middleware/logger.js

Express project folders

- **/views**
  - **Purpose**: Stores template files (if using a view engine like Pug, EJS)..
  - For server-side rendering of HTML pages..
  - **Example:** /views/index.pug

```
app.set('view engine', 'pug');
app.set('views', path.join(__dirname, 'views'));
```

AMIT DHOJU

Express project folders

- **/public**
  - **Purpose**: Static files like CSS, JavaScript, images, fonts.
  - Express serves these directly to the client.
  - **Example:** /public/{*}.{css | js}

Express project folders

- **app.js**
  - **Purpose**: Main entry point of the app..
  - Sets up Express app, middleware, routes, and error handlers..
  - **Example:** app.js

Section 6: Debugging Techniques for Node.js Express App

Debugging technique

- try-catch
- console functions
- debugger usage
- node inspector
- postman

Try-Catch Blocks

- Wrap your synchronous code or async functions with try-catch to catch runtime errors.
- Use in middleware or route handlers to gracefully handle exceptions.

```
app.get('/user/:id', async (req, res) => {
  try {
    const user = await getUserById(req.params.id);
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: 'Server Error', error: error.message });
  }
});
```

AMIT DHOJU

Console Functions

- Use console.log(), console.error(), console.warn(), and console.table() for quick insights.
- Add meaningful messages and variable outputs for tracing app flow.

```
console.log('User ID:', req.params.id);
console.error('Database connection failed');
console.table([{ id: 1, name: 'Amit' }, { id: 2, name: 'John' }]);
```

AMIT DHOJU

Debugger Usage

- Use the built-in Node.js debugger.
- Start your app with
  - **node inspect app.js**
  - or **node --inspect app.js**
- Insert debugger; statement in your code where you want to pause execution.
- Then use Chrome DevTools or VSCode debugger to step through your code.

```
app.get('/data', (req, res) => {
  debugger;  // Execution will pause here if debugger is attached
  res.send('Debug me');
});
```

AMIT DHOJU

Node Inspector

- Start your app with the inspector enabled
  - **node --inspect-brk app.js**
- --inspect-brk pauses execution on the first line.
- Open Chrome browser and navigate to chrome://inspect to connect and debug.
- Supports breakpoints, call stack inspection, variable watching.

AMIT DHOJU

Testing with Postman

- Postman is a GUI tool to test API endpoints.
- Helps verify request methods, payloads, headers, and responses.
- Useful to isolate backend issues by manually triggering routes.
- Can save test collections for repeated debugging or regression tests.

AMIT DHOJU

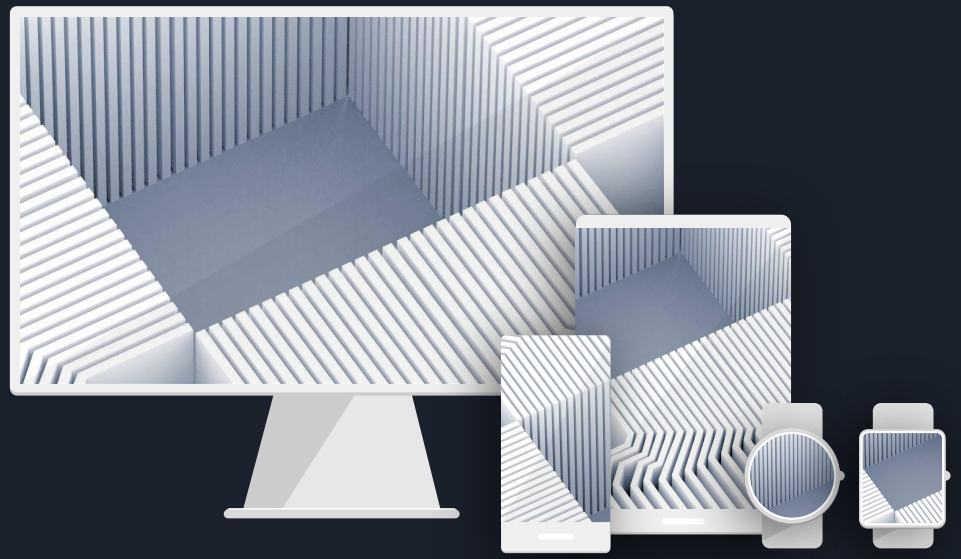
Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Hands-On Exercise:

Express app: create some more routes



AMIT DHOJU

Section 7: System Design Basics

What is System Design?

- process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

- how to build systems that work well under load, are maintainable, scalable, and fault tolerant.

AMIT DHOJU

Key Concepts

- **Scalability**: Ability of the system to handle increasing load by adding resources.
- **Vertical scaling**: Adding more power (CPU, RAM) to a single machine.
- **Horizontal scaling**: Adding more machines to distribute load.
- **Availability**: The system's ability to remain operational and accessible.

AMIT DHOJU

Key Concepts

- **Reliability**: The system works correctly even when some components fail.
- **Latency**: Time taken to respond to a request.
- **Throughput**: Number of requests handled per unit time.

AMIT DHOJU

Core Components of a System

- **Client**
- **API Layer / Load Balancer**
- **Application Servers.**
- **Database**
- **Cache**
- **Message Queues**
- **CDN: Content Delivery Network**

AMIT DHOJU

Core Components of a System

- Client: User or service that interacts with the system.
- API Layer / Load Balancer: Distributes requests, balances traffic.
- Application Servers: Handle business logic.
- Database: Stores data (SQL or NoSQL).
- Cache: Temporary fast storage for frequently accessed data.
- Message Queues: Asynchronous communication for decoupling components.
- CDN: Content Delivery Network to serve static files quickly.

AMIT DHOJU

Design Principles

- **Separation of Concerns**: Divide system into distinct features/modules.
- **Loose Coupling**: Components should be independent to reduce impact of change.
- **High Cohesion**: Related functionality grouped together.
- **Fail Fast**: Detect and handle errors quickly.
- **Idempotency**: Ensure operations can be safely repeated.

AMIT DHOJU

Common Design Patterns

- **Client-Server Architecture**
- **Microservices Architecture**: Splitting system into small independent services.
- **Event-Driven Architecture**: Components react to events/messages.
- **Load Balancing**: Distribute requests to multiple servers.

- **Caching Strategies**: Cache Aside, Write Through, Write Back.

- **Database Sharding**: Splitting data horizontally to improve performance.

AMIT DHOJU

Previously

- Javascript basics (scope, variables, functions)
- Asynchronous programming
- Nodejs installation and introduction
- Nodejs modules and it usage
- Express JS introduction and a demo application

Day 3: Database Fundamentals

- Database Fundamentals
- MySQL Implementation
- POST,PUT & PATCH
- Security & Design

Section 1: Database Fundamentals

What is a Database?

- An organized collection of data that can be easily accessed, managed, and updated.
- Types
  - SQL Databases(Structured Query Language)
    - MySQL, PostgreSQL, Oracle
  - NoSQL Databases(No SQL)
    - MongoDB, CouchDB
  - In-Memory Databases
    - Redis, Memcached

AMIT DHOJU

What is a Database?

- An organized collection of data that can be easily accessed, managed, and updated.
- Types
  - SQL Databases(Structured Query Language)
    - MySQL, PostgreSQL, Oracle
  - NoSQL Databases(No SQL)
    - MongoDB, CouchDB
  - In-Memory Databases
    - Redis, Memcached

AMIT DHOJU

SQL Databases (Relational)

- Store data in tables (rows & columns).
- Use SQL for queries.

- Examples: MySQL, PostgreSQL, Oracle.

- Pros: Strong consistency, powerful querying, ACID transactions.
- Cons: Less flexible for unstructured data.

AMIT DHOJU

NoSQL Databases (Non-relational)

- Store data in various formats: key-value, document, graph, wide-column.
- Examples: MongoDB, CouchDB.
- Pros: Flexible schema, horizontal scaling.
- Cons: Weaker consistency (eventual consistency in some cases).

AMIT DHOJU

In-Memory Databases

- Store data in RAM for ultra-fast access.
- Examples: Redis, Memcached.
- Pros: Extremely fast, good for caching.
- Cons: Data loss if not persisted to disk.

Which one to choose?

- **Use SQL** when data is highly structured & requires complex queries.
- **Use NoSQL** when data is semi-structured, unstructured, or requires scalability.
- **Use In-Memory** for performance-critical caching and real-time data.

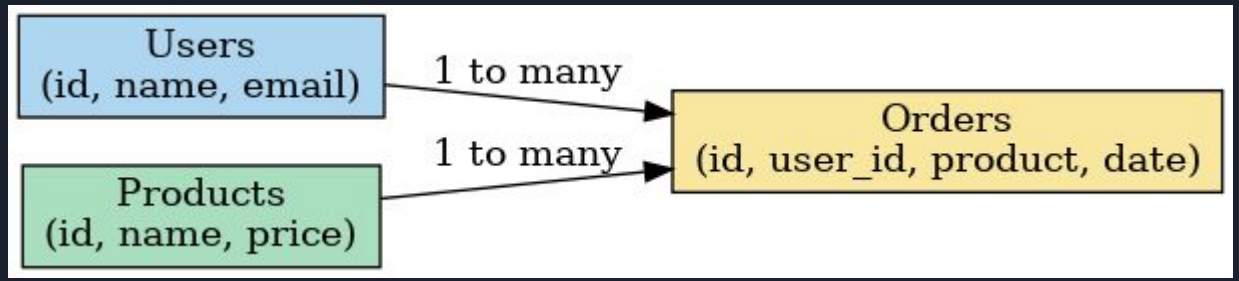
AMIT DHOJU

Section 2: Database Design

Database Design

- Use ER Diagrams to visualize entities and relationships.

- Example:
  - Users table → has many → Orders table.
  - Products table → is in many → Orders table
  - Orders table → belongs to → Users table.



AMIT DHOJU

Section 3: MySQL Implementation

Database: Create Schema

- **Understand Requirements**
  - Identify what data you need to store (e.g., students, colleges).
  - Clarify relationships between data (one-to-many, many-to-many).
- **Identify Entities & Attributes**
  - Entities = tables (e.g., Students, Colleges)
  - Attributes = columns (e.g., name, email)

Database: Create Schema

- **Define Primary Keys (PK)**
  - Unique identifiers for each record (e.g., student_id, college_id).

- **Define Relationships & Foreign Keys (FK)**
  - Link tables (e.g., Orders.user_id → Users.user_id).

AMIT DHOJU

Database: Create Schema

- **Define Primary Keys (PK)**
  - Unique identifiers for each record (e.g., user_id, order_id).

- **Define Relationships & Foreign Keys (FK)**
  - Link tables (e.g., Orders.user_id → Users.user_id).

AMIT DHOJU

Database: Create Schema

```
CREATE SCHEMA `training`
DEFAULT CHARACTER SET
utf8mb4 ;
```

AMIT DHOJU

Database: Create tables

```sql
CREATE TABLE colleges (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(255)
);

CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    unique_id VARCHAR(50) UNIQUE NOT NULL,
    college_id INT,
    FOREIGN KEY (college_id) REFERENCES colleges(id)
);
```

Database Setup with MySQL

- Install MySQL locally or use cloud DB services.
- Use MySQL Workbench for GUI-based management.
- Connecting MySQL to Node.js/Express
  - `***npm install mysql***`
- CRUD operations

AMIT DHOJU

Connecting MySQL to Node.js

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'test_db'
});

connection.connect(err => {
  if (err) throw err;
  console.log("Connected to MySQL!");
});
```

AMIT DHOJU

CRUD Operations : Create

- Create new record in database

- connection.query('INSERT INTO users (name, email) VALUES (?, ?)', ['John', 'john@example.com']);

CRUD Operations : Read

- Read data from database

- connection.query('SELECT * FROM users', (err, results) => {
- console.log(results);
- });

AMIT DHOJU

CRUD Operations : Update

- Update existing record in database.

- connection.query('UPDATE users SET name=? WHERE id=?', ['Mike', 1]);

CRUD Operations : Delete

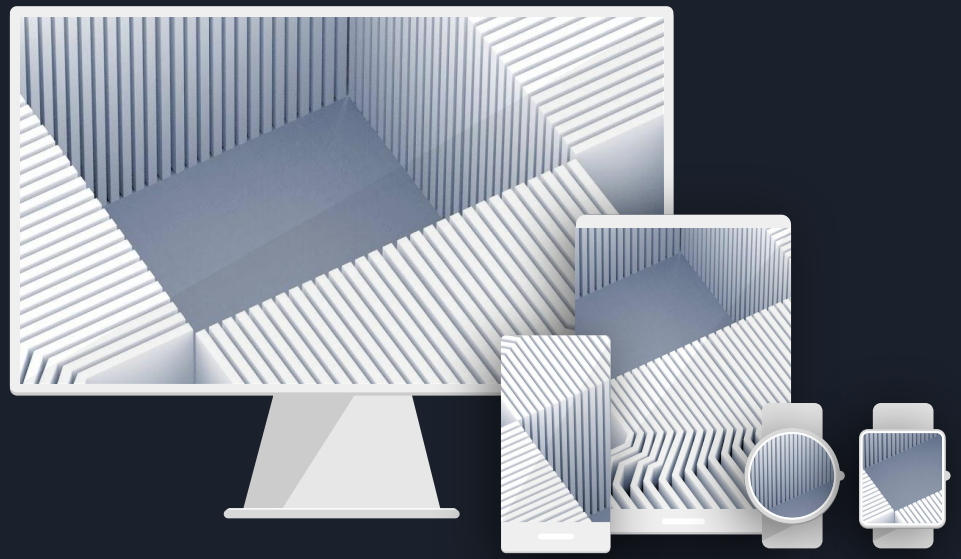
- Delete existing record by id

- connection.query('DELETE FROM users WHERE id=?', [1]);

AMIT DHOJU

Hands-On Exercise:

Express app:
Apply on express app



AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Day 4: Authentication & Authorization

Section 1:
Security & Design

Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

SQL Injection Prevention

```
// DO NOT USE - Vulnerable to SQL Injection
app.get('/user', (req, res) => {
  const userId = req.query.id; // e.g. 1 OR 1=1
  const query = `SELECT * FROM users WHERE id = ${userId}`;

  connection.query(query, (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

AMIT DHOJU

SQL Injection Prevention

📌 If a user sends:

bash

```
/user?id=1 OR 1=1
```

The query becomes:

sql

```
SELECT * FROM users WHERE id = 1 OR 1=1;
```

This returns **all users**, bypassing authentication.

AMIT DHOJU

SQL Injection: Safe Implementation

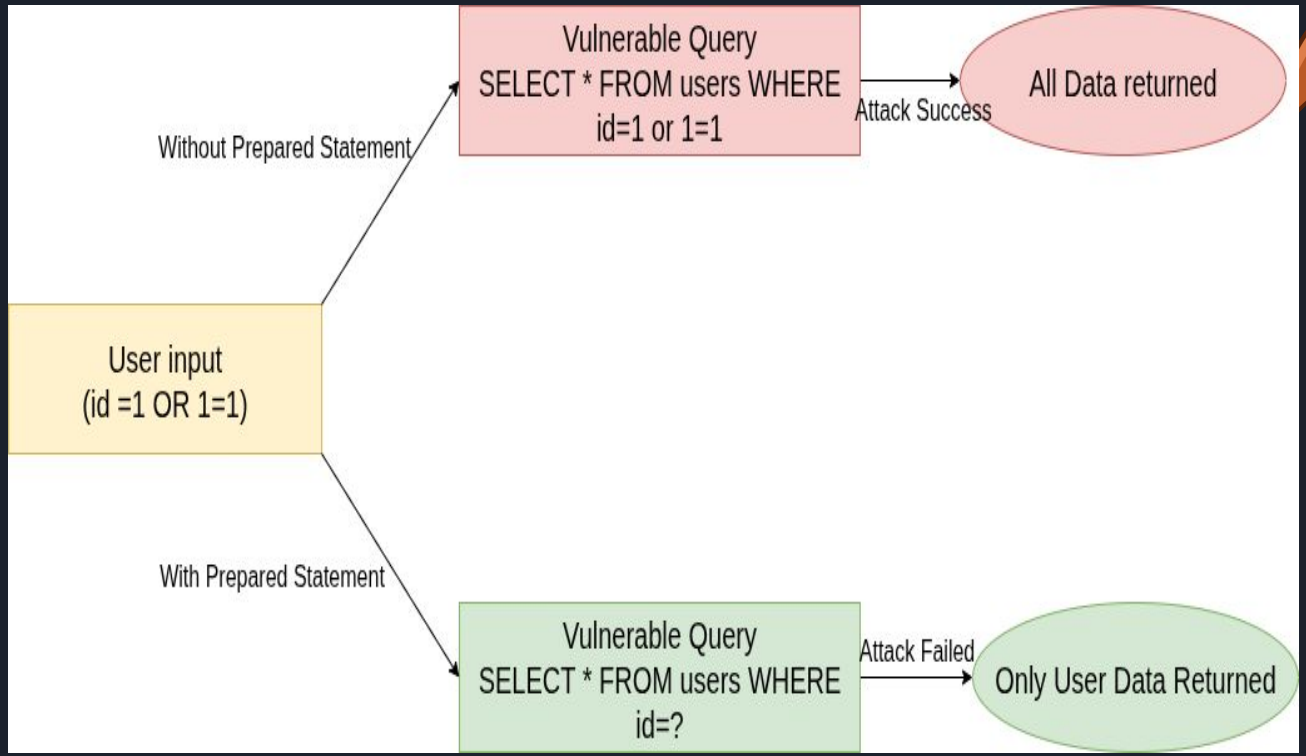
```
// SAFE - Using parameterized query
app.get('/user', (req, res) => {
  const userId = req.query.id;
  const query = 'SELECT * FROM users WHERE id = ?';

  connection.query(query, [userId], (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

Why safe?

- The `?` is replaced with escaped values — SQL code injection is impossible.

AMIT DHOJU

SQL Injection: Safe Implementation



AMIT DHOJU

XSS (Cross-Site Scripting)

🔴 **Vulnerable Example:**

```js
app.get('/search', (req, res) => {
  const term = req.query.q;
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

📌 If someone sends:

```javascript
/search?q=<script>alert('Hacked')</script>
```

The browser executes the script.

AMIT DHOJU

XSS: Safe Implementation

```
const escapeHtml = (unsafe) =>
  unsafe.replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");

app.get('/search', (req, res) => {
  const term = escapeHtml(req.query.q);
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

AMIT DHOJU

XSS: Safe Implementation

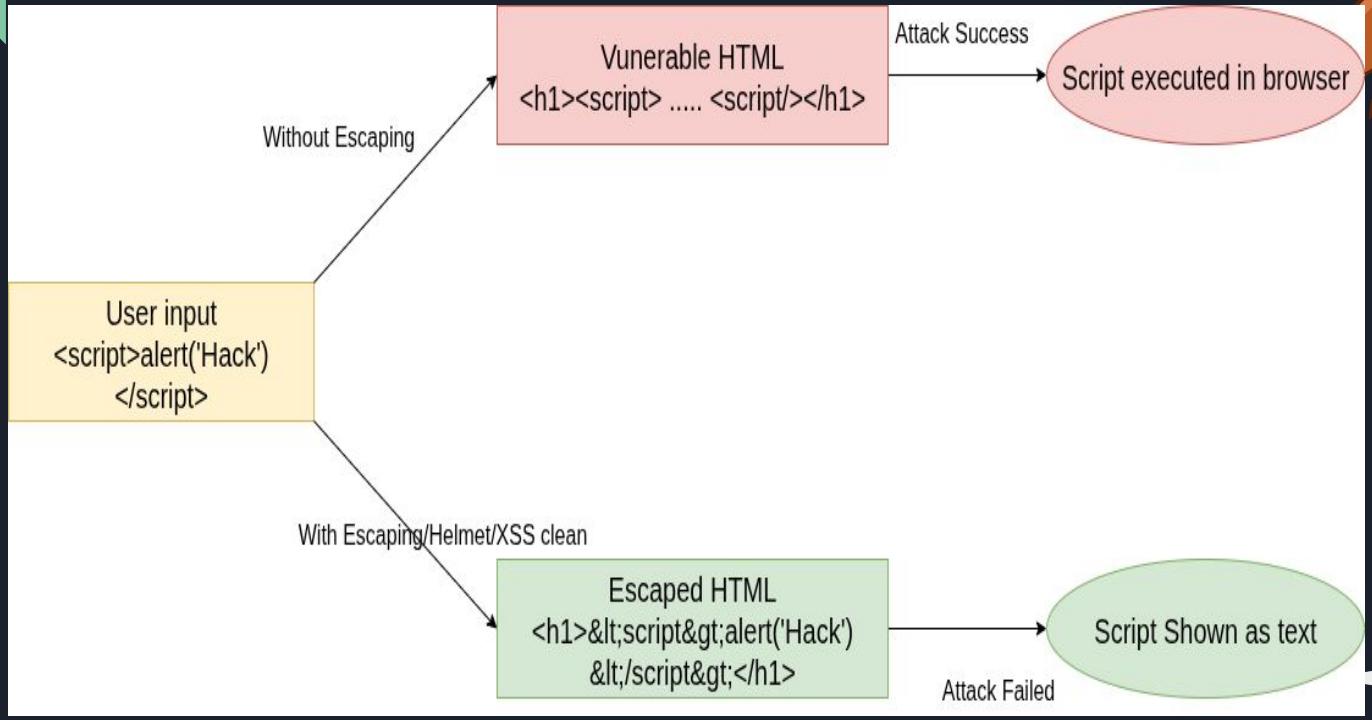
-

```bash
npm install helmet xss-clean
```

```js
const helmet = require('helmet');
const xss = require('xss-clean');

app.use(helmet()); // Security headers
app.use(xss());    // Prevent XSS
```

AMIT DHOJU

XSS: Safe Implementation



AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Section 2: Security Concepts

Authentication – Who are you?

- Process of verifying the user’s identity.
-
- Common methods:
  - Username/password
  - OTP (One-Time Password)
  - Biometric (Fingerprint, Face ID)

Authorization – What can you do?

- Process of checking the user's permissions after authentication.

- Example:
  - User A can view and edit their profile.
  - User B can only view their profile.

AMIT DHOJU

Session Management: Server-side state (Stateful)

- Stores user data (login status, preferences) on the server.
- Usually implemented with cookies and session IDs.
- Requires server memory or database storage.
- **Stateful** — server needs to remember the user.

Session Management: Server-side state (Stateful)

Session-based Authentication Flow

1. User logs in → sends credentials to Server

2. Server verifies credentials

3. Server creates session in Session Store

4. Server sends session ID in cookie

5. Client sends cookie with each request

6. Server fetches session from Session Store to validate user

AMIT DHOJU

Token-based Authentication – Stateless authentication

- No server memory of the session.
- User receives a token after logging in.
- Token is sent with every request.
- Works well for scalable APIs.

JWT (JSON Web Tokens)

- Compact, secure token format (base64-encoded).
- Contains:
  - Header (type & algorithm)
  - Payload (user data, claims)
  - Signature (ensures integrity)

- Example use:
  - Single Sign-On (SSO).
    - Google, Facebook etc login for sites
  - Token base login system
- Link: jwt.io

AMIT DHOJU

Token-based Authentication

Token-based JWT Authentication Flow

1. User logs in → sends credentials to Server

2. Server verifies credentials

3. Server creates JWT and sends to Client

4. Client stores JWT (localStorage/cookie)

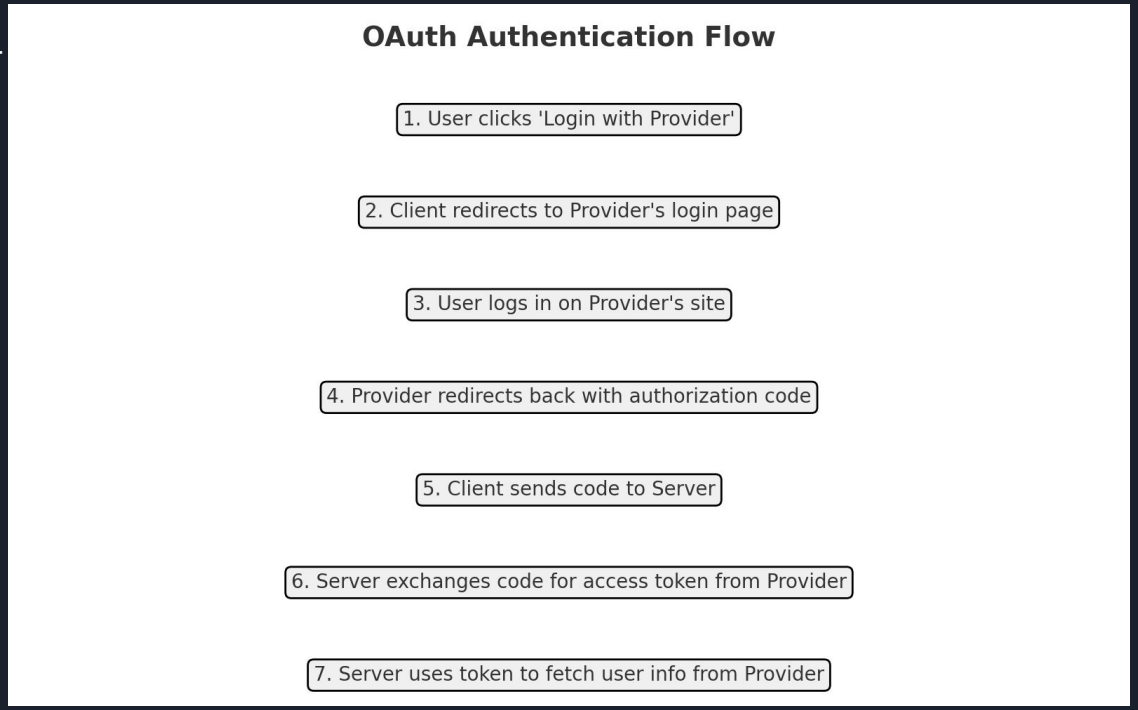
5. Client sends JWT in Authorization header with each request

6. Server verifies JWT signature and payload

OAuth – Third-party authentication

- Lets users log in via other providers (Google, Facebook, GitHub).
- Avoids storing user passwords yourself.
- Uses **access tokens** and **refresh tokens**.

OAuth – Third-party authentication



AMIT DHOJU

Thank You!

That’s all for today🫡

Thank you for your time