Welcome to Full Stack Web Development!

About me...

Your turn!

Class Expectations & Syllabus

Learning Path

- 9 modules across 41 lectures. Classes every Saturday & Sunday
- Outcome: a solid software engineering foundation. Ability to independently build applications using MERN stack

What you can expect from me?

- Real-world examples, live coding and hand-on exercises
- Open Q&A, discussions and collaborative problem-solving

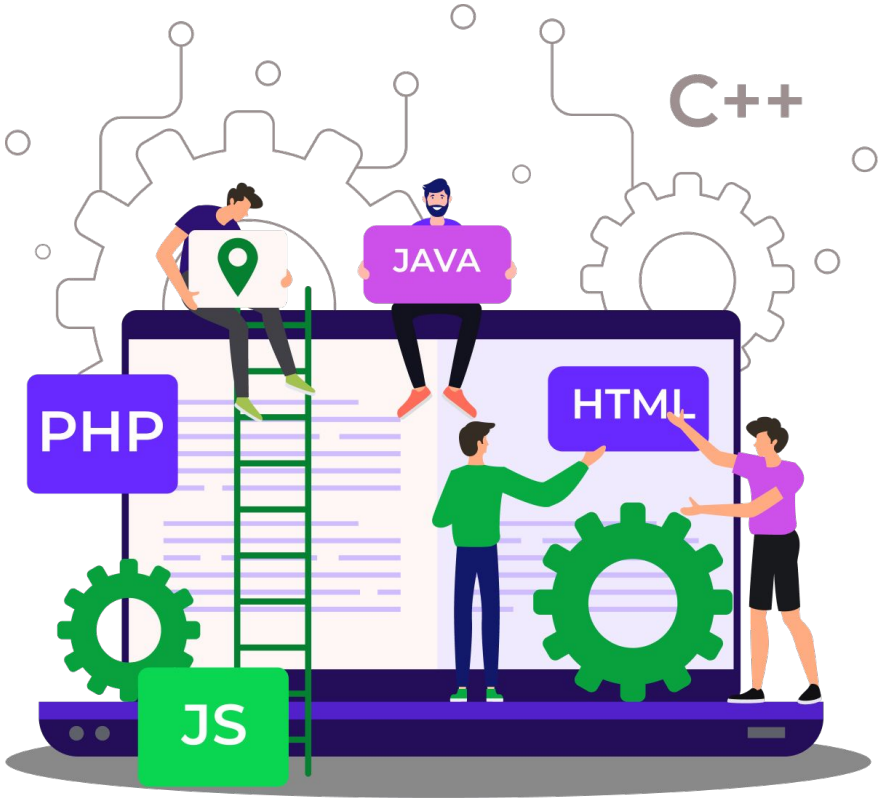
What will I expect from each of you?

- Participation: your presence, ask questions and engage in discussions
- Consistent practice: complete tasks, explore references and apply to projects
- Use of LLMs (ex: ChatGPT) to understand the “why” vs writing the code for you

Commitment outside of class

- Self-study: Allocate time for reading, practice and revisiting session materials
- Ask for help: Share the topics you find difficult so that we can address them together

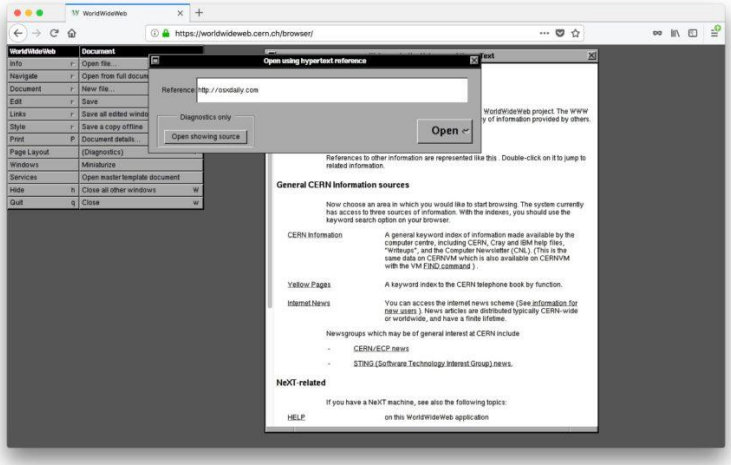
The Evolution of Web Development



The Big Bang - Web 1.0 (1990s)

Features:

- HTML and CSS helped create the static content
- Interactivity was limited as Javascript hadn't come into the picture
- Content were served directly from servers; databases such as MySQL came only in 1995



Challenges:

- Functionally limited as most content was static
- Tedious to maintain as any change to your webpage meant manual changes to HTML files
- Poor user experience: no personalization

State of the web: Websites were static, like online brochures.

The Rise of Interactivity: Web 2.0 (2000s)

Features:

- Rich user interactions (e.g., social media, e-commerce)
- Use of databases for dynamic content delivery
    - Relational DBs remained popular; NoSQL starts to gain popularity
- Rise of JavaScript for client-side interactivity.

Challenges:

- Complex server-side programming
    - The PHP/ASP.NET/etc backend was tightly couple with the client
- Scalability issues with monolithic architectures
    - Traditional server-client models struggled when dealing with high traffic and required expensive “vertical” scaling. Microservices start solving this.
- Developers needed specialized tools for both client and server.

State of the web: Websites became more interactive, enabling greater user contribution and engagement

Modern Challenges Demand Modern Tools

User expectations:

- Fast and responsive
- Real-time updates (eg: chat apps, live dashboards)

Developer needs:

- Modular, reusable codebases
- Tools for managing state, routing, and large datasets

Modern Challenges Demand Modern Tools

Challenges:

- Handling increased client side complexity
    - Richer UIs mean we need specialized tools to manage state (Redux Toolkit, Remix, etc), routing (react-router), form handling (react hooks form) and more
- Developer efficiency when switching between frontend and backend
    - Building a Java or Python based backend and Javascript & React based frontend means that engineers will need to learn both technologies to build end-to-end products.

MERN Assemble

The Rise of MERN: MongoDB

Traditional relational databases (MySQL, Oracle) struggled with:

- Rigid schema requirements
- Poor scalability for modern web apps

In 2007, three engineers created MongoDB to address these challenges:

- Schema-less: Flexible data models for rapid iteration
- JSON-like storage: Aligned with web standards (easier data exchange)
- Horizontal scaling: Distributed architecture for large-scale apps
- Real-time performance: Optimized for high-speed read/write operations

Became a go-to choice for scalable, real-time applications (social media, IoT)

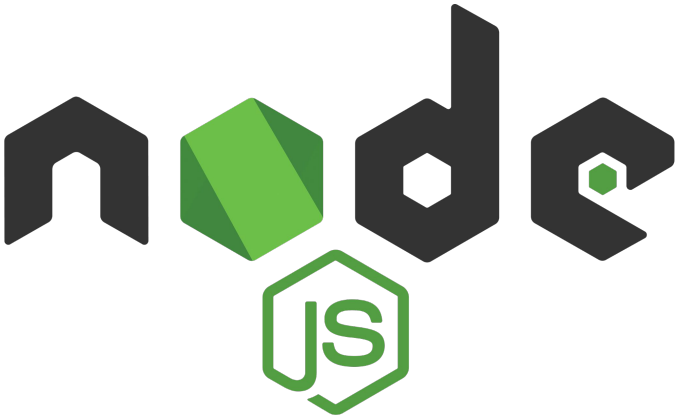
The Rise of MERN: Node (& Express)

Ryan Dahl needed an efficient way to handle file uploads asynchronously, but existing server-side technologies (PHP, Ruby) were blocking and inefficient

In 2009, Node.js was created by running Google's V8 JavaScript engine outside the browser:

- Non-blocking I/O: Handles thousands of concurrent connections efficiently
- Single programming language: JavaScript for both frontend & backend
- Ecosystem growth: Community-built tools like Express.js for web servers

Enabled scalable backend services and full-stack JavaScript applications



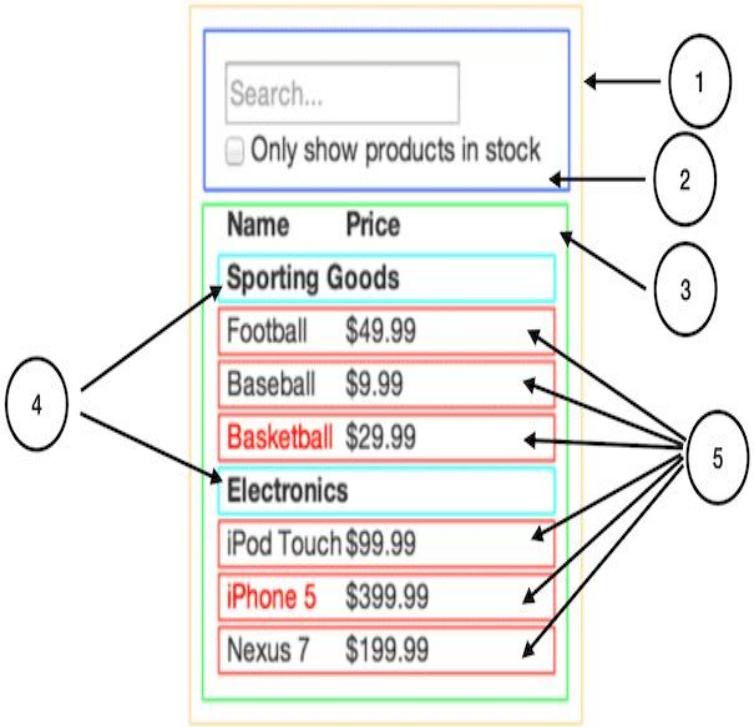
The Rise of MERN: React

Facebook's complex UI (news feed, chat, notifications) was hard to manage with jQuery and DOM manipulations, leading to performance bottlenecks

In 2013, Facebook open-sourced React to solve UI challenges:

- Virtual DOM: Efficient updates without direct manipulation of the browser DOM
- Component-based architecture: Reusable and maintainable UI pieces
- State management: Easier to track and sync changes across the app

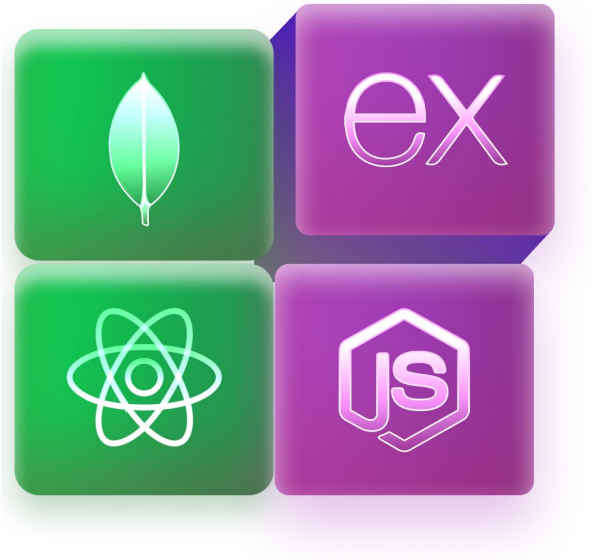
React revolutionized frontend development, making it easier to build dynamic, interactive UIs



Bringing It All Together – MERN Stack

How These Tools Form the MERN Stack:

- MongoDB: Flexible data storage for web applications
- Node.js: High-performance server-side environment
- Express.js: Lightweight backend framework built on Node.js
- React: Efficient front end UI development



Why MERN?

- Full-stack JavaScript for consistency across frontend and backend
- Scalability and flexibility for modern applications
- Strong open-source community support.

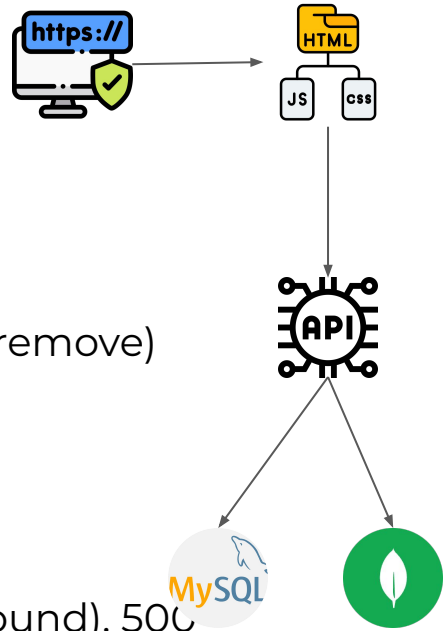
Pre-requisites

Pre-requisites: Browser

- What is a browser?
  - Software that allows users to access and interact with content on the world wide web
- What's HTML/CSS/JS?
  - Hypertext Markup Language: provides structure to a page
  - Cascading Style Sheets: provides styling to a page
  - Javascript: provides interactivity to a page
- How does the browser render the page?
  - DOM (Document Object Model) is created from the HTML
  - CSSOM (CSS Object Model) for styling
  - Javascript: execution and event handling
- Client-Server Model
  - Client (browser/application) requests data
  - Server will fetch the data (ex: from a database) and return it

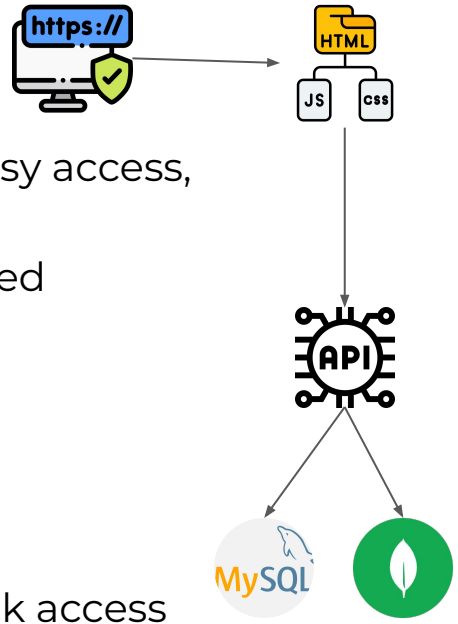
Pre-requisites: API Design

- What is HTTP/HTTPS?
  - Hypertext Transfer Protocol (Secure)
  - Structure: Headers, body, status codes
- What are HTTP methods?
  - GET (read), POST (create), PUT/PATCH (update), DELETE (remove)
- REST Principles
  - Stateless client-server communication
  - Resource-based endpoints (e.g., /users, /posts)
- What are status codes?
  - Signals success or failure of actions: 200 (OK), 404 (Not Found), 500 (Server Error)
- What is JSON?
  - Javascript Object Notation: a human-readable data format
  - Key-value pairs for data exchange



Pre-requisites: Databases

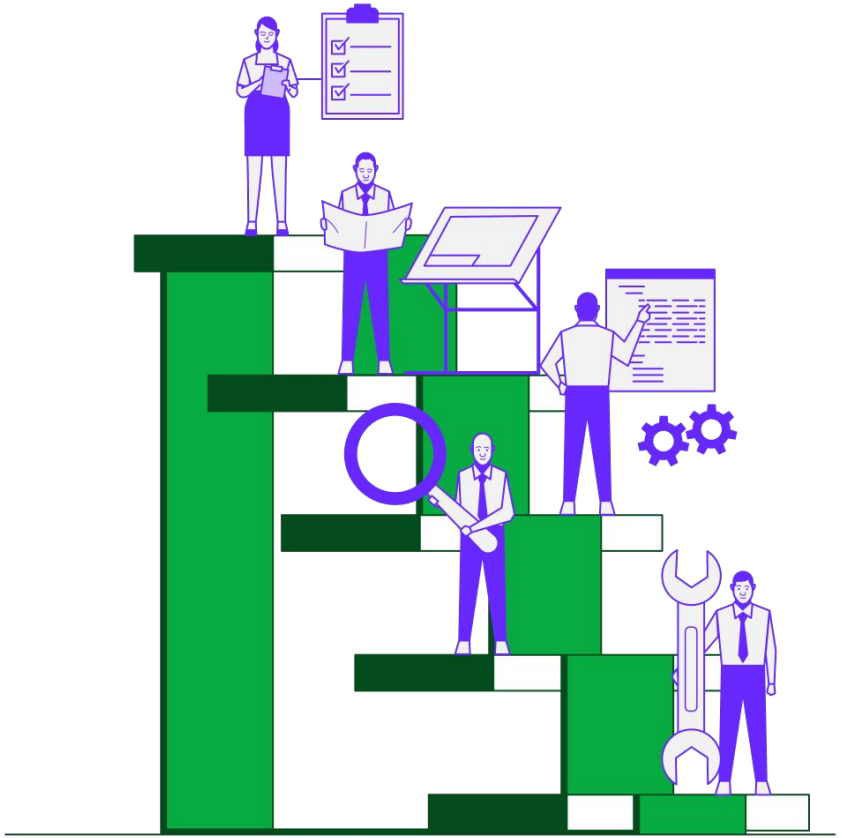
- What's a database?
  - Organized collections of data that are structured for easy access, management, and updating
  - Think of a library where books are organized and indexed
- What kinds of operations can be performed on databases?
  - Create, read, update, delete
- Types of databases
  - Relational (MySQL, Postgres)
    - Structured tables with rows & columns
    - Data is stored as a modified tree structure for quick access
  - NoSQL (MongoDB)
    - Flexible document storage
    - Data is stored as BSON (Binary JSON)



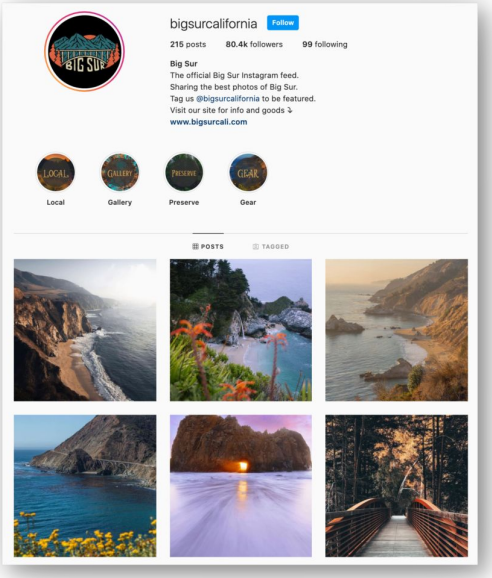
Pre-requisites: Helpful Tools

- Git (Version Control)
  - Tracks code changes, facilitates team collaboration (branches, merges)
- Package Manager (npm/Yarn)
  - Installs/manages external libraries, simplifies dependency handling
- Terminal/Command Prompt
  - Essential for running commands, scripts, version control

Use Cases



Instagram w/ MERN



MongoDB:

- Store user profiles, posts, comments, and likes in a scalable NoSQL database.
- Efficient indexing for quick content retrieval (e.g., hashtags, users).

Express.js:

- API layer to handle requests for user authentication, content feeds, and media uploads.
- Route management for user posts and profile actions.

Node.js:

- Handle real-time notifications, chat services, and background jobs (e.g., sending emails).
- Non-blocking architecture to handle millions of concurrent users.

React:

- Build the interactive frontend for browsing the feed, uploading posts, and engaging with content (likes, comments).
- Virtual DOM for fast UI updates when users interact with posts.

Uber w/ MERN

MongoDB:

- Store ride history, driver details, and real-time location tracking
- Geospatial queries to find nearby drivers and riders

Express.js:

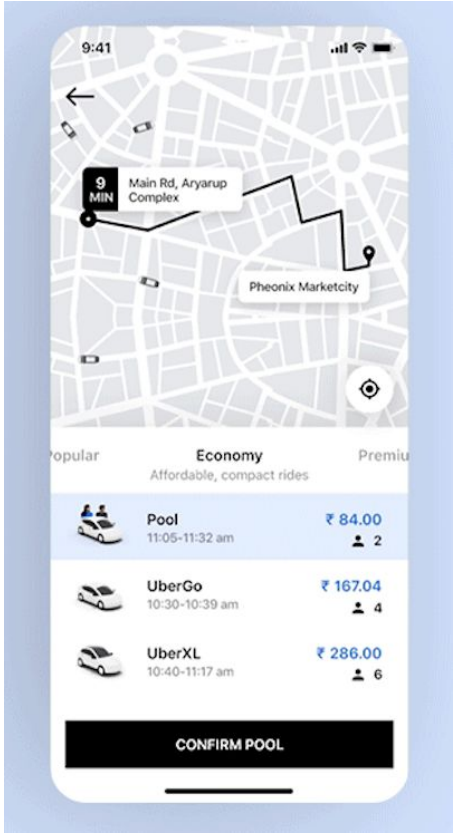
- Expose APIs for booking rides, fare calculations, and driver/rider interactions
- Middleware to handle authentication and payment processing

Node.js:

- Handle real-time ride status updates using WebSockets
- Manage concurrent requests for ride matching and ETA calculations

React:

- Component-based architecture for modular UI design
- Build the dynamic user interface for booking rides, tracking location in real time, and displaying estimated arrival times



Setup: Download & Install Required Tools

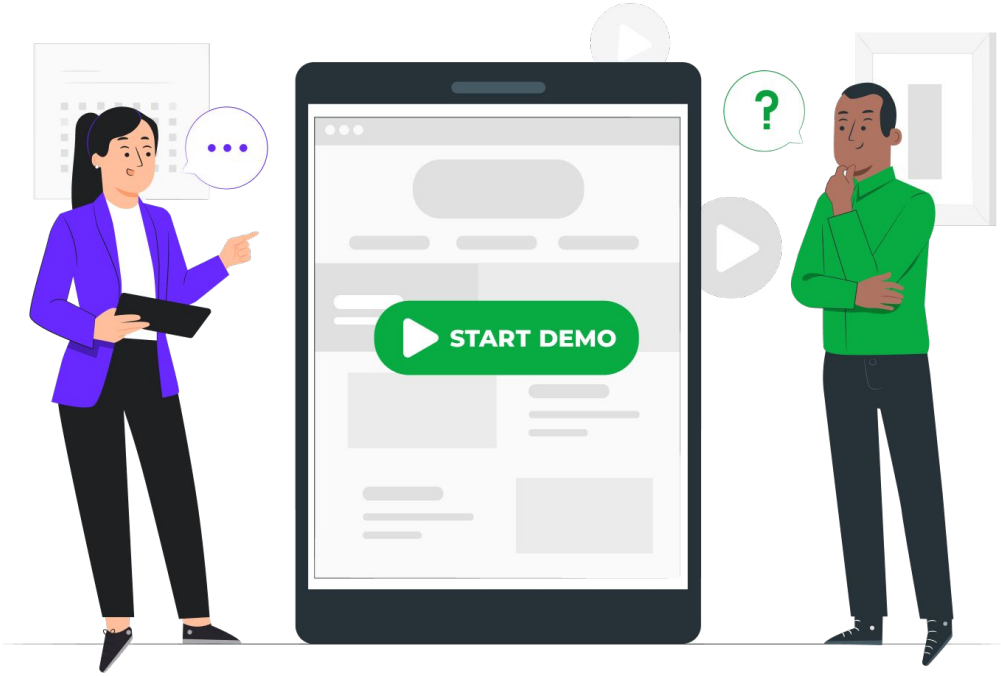
List of Tools:

- Node.js: our runtime (comparable to a server)
- NVM installation: to manage node.js versions on your system
- MongoDB: local MongoDB instance
- MongoDB Compass: tool to simplify the connections to MongoDB & visualizations
- Visual Studio Code: code editor

Documentation:

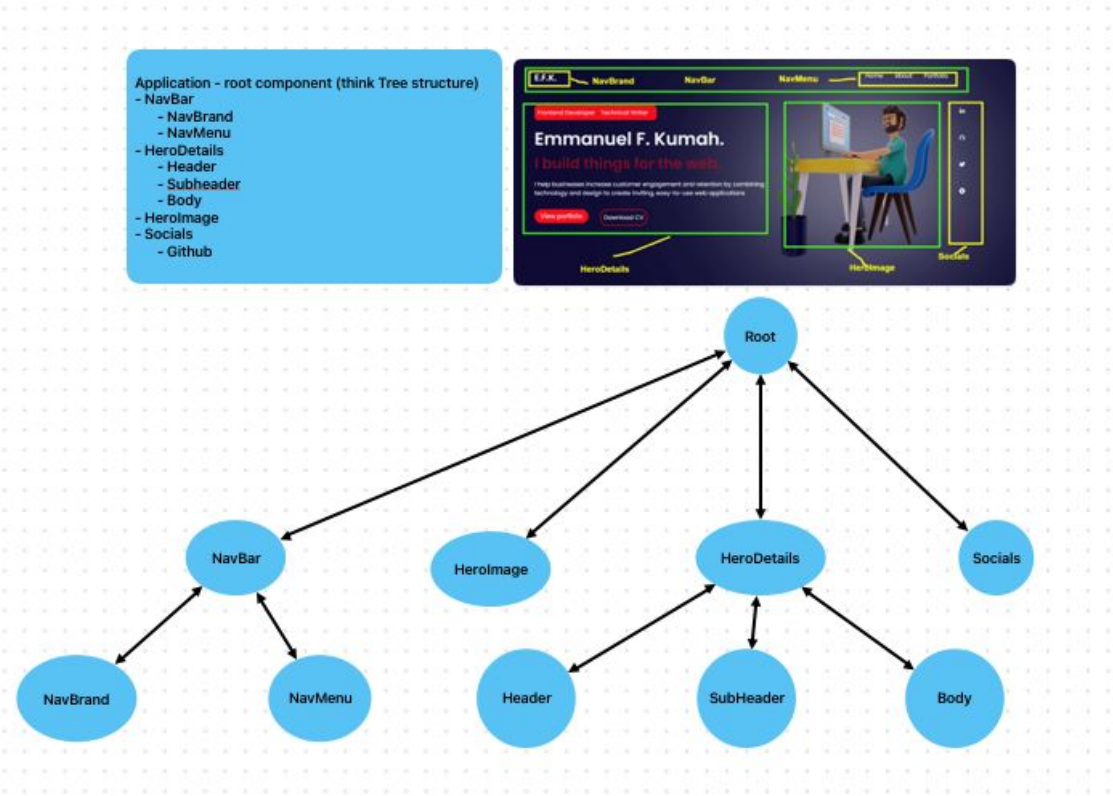
- Node.js
- MongoDB
- React
- Express

Let’s code!



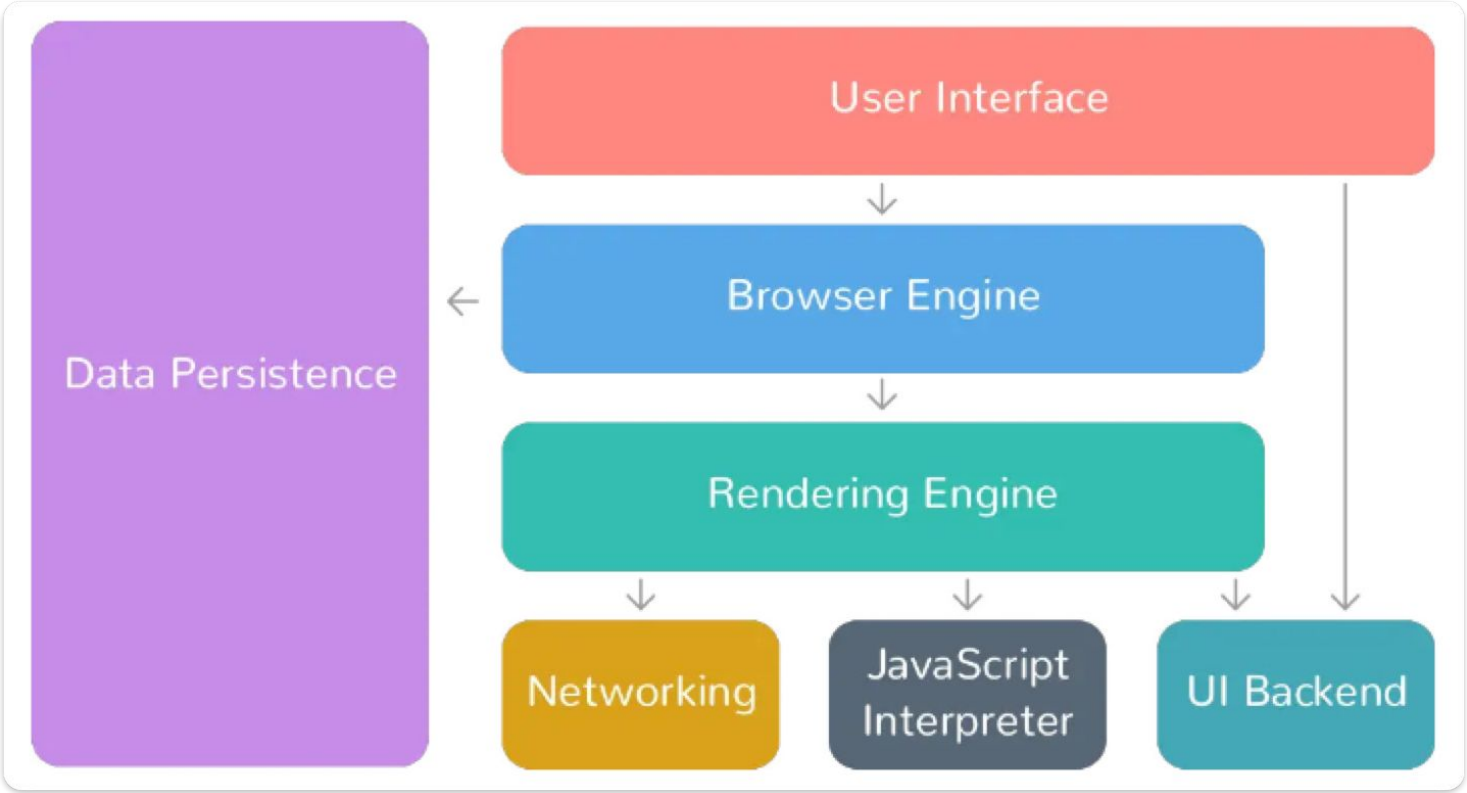
Visualisations & Exercises (Reference)

An overview of the DOM

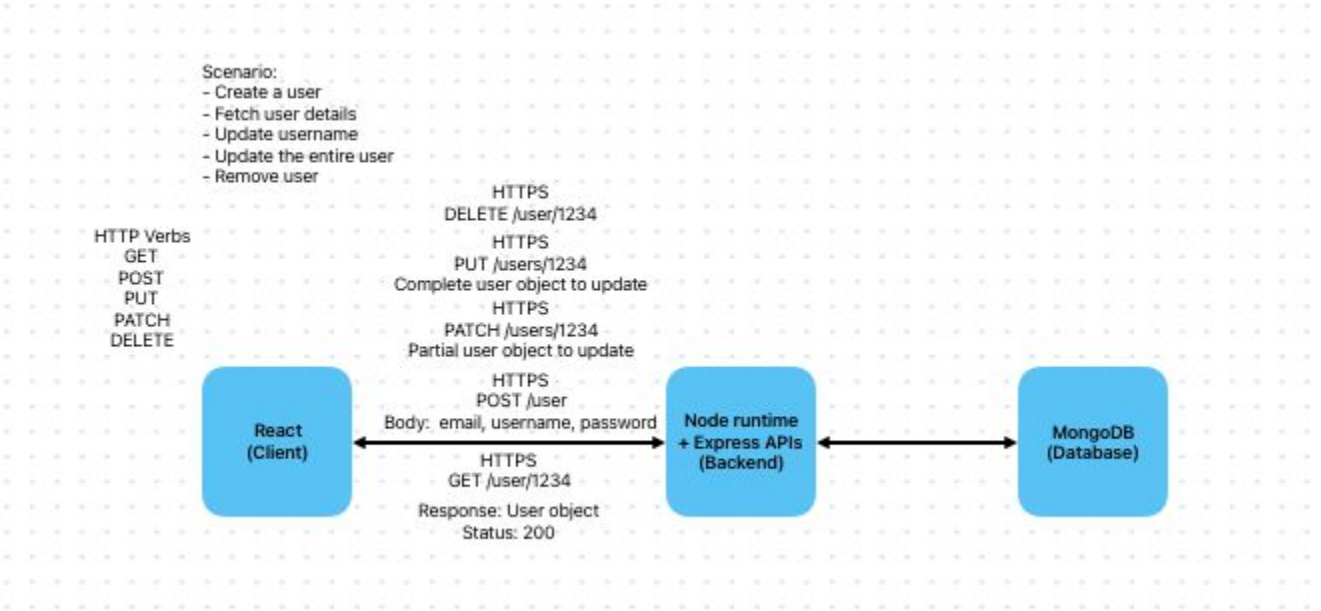


Internals of a browser

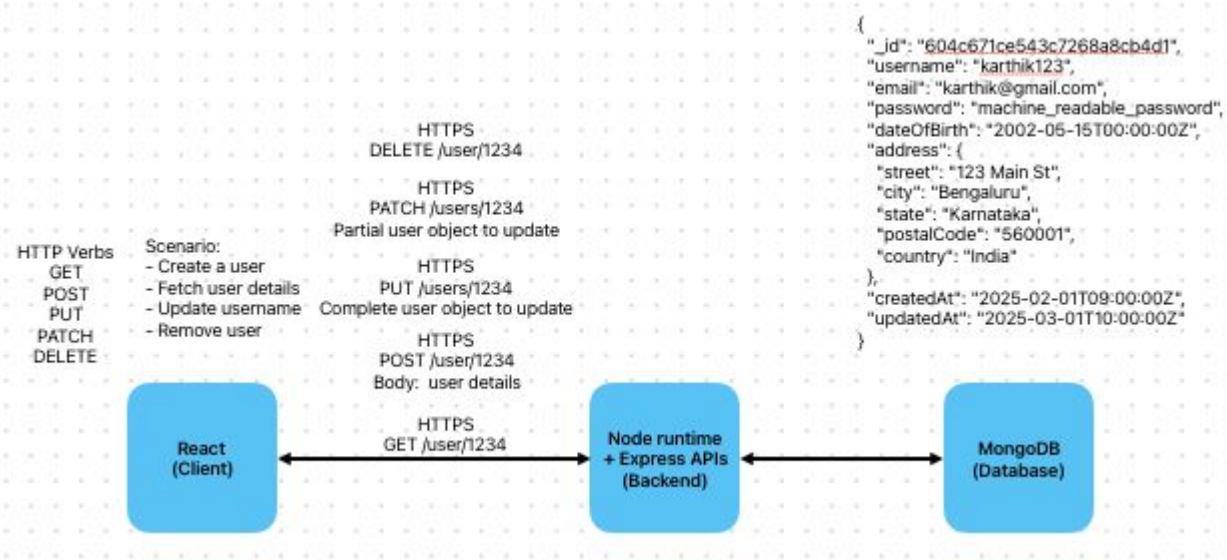
Reference



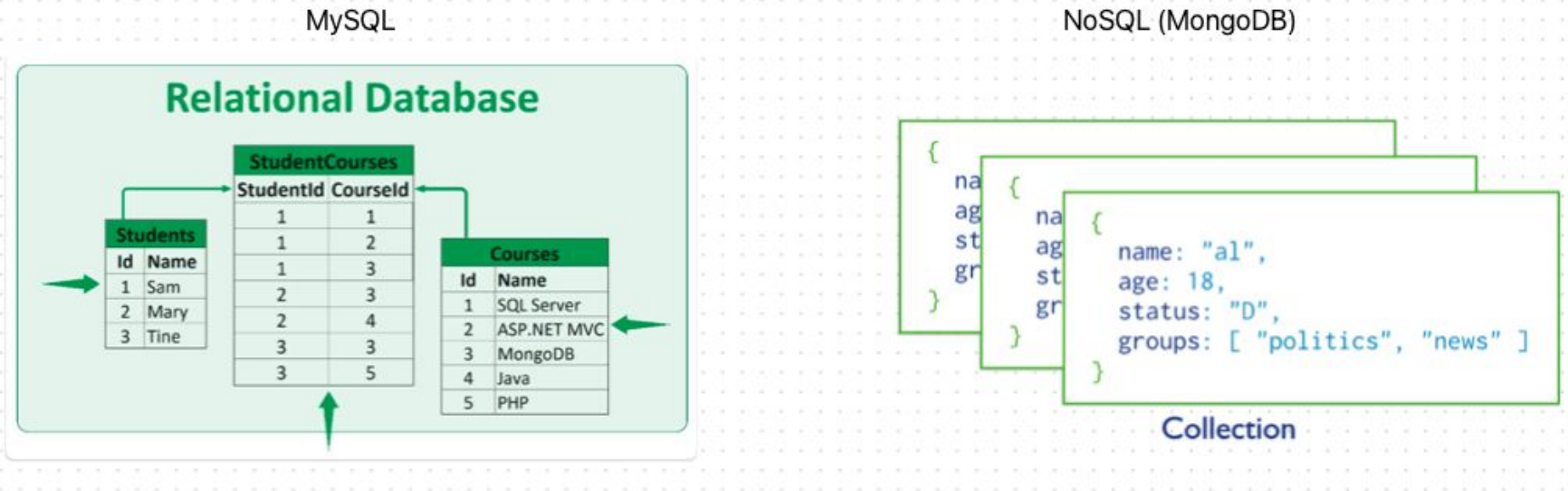
Understanding REST



Understanding JSON



An overview of the DOM



Exercise on creating Instagram