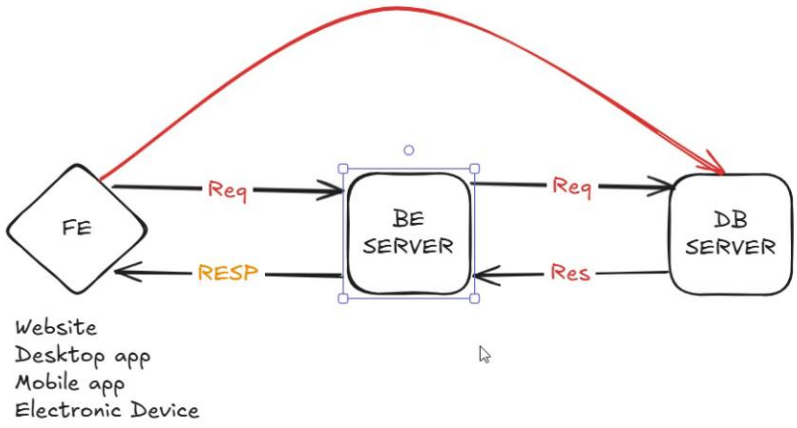
FSD MERN

INTRODUCTION TO FSD MERN

(Module-01)

Day 1 – Introduction to Foundation of Full Stack Development



---

1. What is Full Stack Web Development?

- **Definition:** Full stack development refers to the development of both frontend (client-side) and backend (server-side) portions of web applications, along with database management.
- A Full Stack Developer is someone who can work with:
    1. Frontend (UI/UX – what the user sees).
    2. Backend (business logic, APIs, authentication).
    3. Database (storage & retrieval of data).

Frontend (Client Side):

- Responsible for the visual part of the application.
- Built using HTML, CSS, JavaScript and frameworks like React, Angular, Vue.
- Example: Login forms, buttons, dashboards, animations.

Backend (Server Side):

- Handles logic & communication with databases.
- Runs code that validates user requests, applies business rules, and sends responses.
- Written using Node.js, Java, Python, PHP, etc.
- Example: Processing login credentials, fetching playlists from Spotify.

Database:

- Stores structured data (like SQL – MySQL, PostgreSQL) and unstructured data (like NoSQL – MongoDB).
- Works 24/7, accessible via backend.
- Provides persistent storage for applications.

👉 Why not connect frontend directly to database?

- Security risk – All database details would be exposed.
- Backend ensures data protection and formats the data before sending.

---

2. Static vs Dynamic Websites

- ### Static Website:
  - Content does not change.
  - Built only with HTML, CSS, JS.
  - Example: Portfolio, documentation websites.
- ### Dynamic Website:
  - Connected to backend & database.
  - Content changes based on user actions.
  - Example: Facebook, Instagram, Amazon.

---

3. Introduction to Web Browsers & JavaScript Engines

- **A web browser is software that allows users to access the web.**
- **Inside the browser:**
    - **Rendering Engine: Renders HTML & CSS into visual pages.**
    - **JavaScript Engine: Executes JavaScript code.**

JavaScript Engine:

- **Converts JavaScript into machine code for execution.**
- **V8 Engine (by Google):**
    - **Written in C++.**
    - **Used in Chrome and Node.js.**
    - **Extremely fast and efficient.**

Other Browser Engines:

- **SpiderMonkey → Firefox.**
- **JavaScriptCore (Nitro) → Safari.**
- **Chakra → Microsoft Edge (Legacy).**

👉 **Every browser has its own engine, but they all follow the ECMAScript standard for JavaScript.**

4. Browser JavaScript vs Node.js

Feature	Browser JavaScript	Node.js
Environment	Runs inside browser	Runs outside browser on server
Purpose	User interaction, DOM manipulation	Backend logic, API handling, server tasks
Engine	Uses browser's JS engine (like V8)	Uses V8 + Node APIs
Access	Can access DOM, local storage	Can access files, databases, network
Example	Validating form inputs	Handling API requests

👉 Node.js Advantages:

- Runs JavaScript outside the browser.
- Enables full-stack development using one language.
- Used in APIs, microservices, ML (TensorFlow.js), and scalable apps.

5. How the Server Interprets a URL

A URL (Uniform Resource Locator) has 4 parts:

1. **Protocol –** Defines how communication happens (HTTP, HTTPS, FTP).
    - Example: https:// (secure), http:// (non-secure).
2. **Domain Name –** The website address (e.g., www.spotify.com).
3. **Resource Path –** Specific location of a file/resource (/playlist/123).
4. **Query Parameters –** Extra information sent to the server (?id=10&type=music).

6. HTTP & HTTPS

HTTP (Hyper Text Transfer Protocol):

- A **protocol** for communication between client and server.
- Works on **port 80**.
- Not encrypted (data can be read by attackers).

HTTPS (Hyper Text Transfer Protocol Secure):

- Secure version of HTTP (encrypted using SSL/TLS).
- Works on **port 443**.

- Prevents man-in-the-middle attacks.

👉 Example:

- http://example.com → Not secure.
- https://example.com → Secure (padlock symbol in browser).

---

7. HTTP Methods

HTTP methods define what action the client wants to perform on the server.

- **GET** → Fetch data (e.g., load a profile page).
- **POST** → Send new data (e.g., register a user).
- **PUT** → Update existing data completely.
- **PATCH** → Update part of data (partial modification).
- **DELETE** → Remove data (e.g., delete a post).
- **OPTIONS** → Check what methods the server supports.

👉 Example with Spotify:

- GET → Fetch playlists.
- POST → Create new playlist.
- PUT → Rename playlist.
- DELETE → Remove playlist.

---

8. Request & Response Cycle

Step 1: User types URL → Browser sends **request**.
Step 2: Server processes → May connect to database.
Step 3: Server sends back **response** (HTML, JSON, images).
Step 4: Browser displays the result.

👉 Example:

- User enters facebook.com → Request sent → Server fetches data → Response is the homepage.

☑ Summary (Day 1)

- **Full stack** = Frontend + Backend + Database.
- **Frontend** = User interface, Backend = Business logic, Database = Data storage.
- Static websites are fixed; dynamic websites are interactive.
- Browsers run JavaScript via engines **like V8**.
- Node.js allows **JS** to run on servers.
- **URL** has protocol, domain, path, query params.
- HTTP/HTTPS are communication protocols **(HTTPS = secure).**
- HTTP methods **(GET, POST, PUT, PATCH, DELETE, OPTIONS)** enable **CRUD** operations.
- Request-Response cycle is how browsers and **servers communicate.**