2. Frontend vs backend architecture

The Core Distinction

Layer	Runs On	Purpose	Primary Languages
Frontend (Client-Side)	Browser / User device	Visual interface, user interaction, displaying data	HTML, CSS, JavaScript, React, etc.
Backend (Server-Side)	Server / Cloud	Business logic, database interaction, API delivery	PHP (Laravel, CodeIgniter), Node.js, Python, etc.

Think of the frontend as the face of the app, and the backend as the brain and spine that make the face meaningful.

The Architecture Layer View

Frontend — Presentation Layer

The frontend is everything users directly see and interact with.

Responsibilities:

- Rendering UI components
- Handling user input (clicks, forms, navigation)
- Fetching and displaying data from APIs
- Managing client-side state (React state, Redux store, etc.)
- Basic validation and UX logic

Technologies:

- **HTML** → Structure
- **CSS / Tailwind / Bootstrap** → Styling
- **JavaScript / React.js** → Interactivity & dynamic data rendering

Output: Markup (HTML), style (CSS), and script (JS) that browsers render.

Backend — Application & Data Layer

The backend handles logic, computation, and data flow invisible to the user.

Responsibilities:
- Handling HTTP requests from frontend
- Authenticating users (login/signup)
- Executing business logic (orders, payments, etc.)
- Managing databases (CRUD operations)
- Securing and serving API endpoints
- Integrating with third-party services (email, payment, cloud storage)

Technologies:
- **Languages:** PHP, JavaScript (Node.js), Python, etc.
- **Frameworks:** Laravel, CodeIgniter, Express
- **Databases:** MySQL, MongoDB
- **APIs:** RESTful or GraphQL

Output: Data (often JSON) sent back to the frontend via HTTP responses.

The Data Flow
1. **Frontend Event** → User performs an action (e.g., clicks "Login").
2. **HTTP Request** → Browser sends data to backend via an API call.
3. **Backend Processing** → Server authenticates credentials, queries DB.
4. **Database Query** → Server fetches or writes data.
5. **Response Returned** → Backend sends JSON or HTML back.
6. **Frontend Render** → UI updates to reflect the result.

Example;
Frontend: fetch('https://api.site.com/login', {method: 'POST', body: JSON})
Backend: validates credentials → returns JWT token
Frontend: stores token → redirects to dashboard

Three-Layer Concept (Full Stack Mental Model)

Layer	Example	Responsibility
Frontend (Presentation)	React.js, Bootstrap	UI & UX

Backend (Application)	Laravel / Node.js	Logic, authentication, APIs
Database (Data)	MySQL / MongoDB	Persistent storage

These three communicate continuously:
Frontend ⇄ Backend ⇄ Database

Security & Communication Boundaries

- Frontend never **accesses** databases directly.
- All sensitive operations go through **backend APIs**.
- Backend enforces authentication, authorization, validation, and rate limiting.
- Frontend securely stores tokens in cookies/localStorage.

Modern Full-Stack Integration Example

Scenario: You build a “Job Portal App”.

Layer	Stack	Example Function
Frontend	React + Tailwind	Job listing UI, user forms
Backend	Laravel / Node.js	Auth, job CRUD, API routes
Database	MySQL / MongoDB	Jobs, users, applications

Flow:
User clicks → React sends API call → Laravel fetches jobs → returns JSON → React renders job cards.

Developer Takeaway

- **Frontend** = how things look and feel.
- **Backend** = how things work and connect.
- **A Full Stack Developer** orchestrates both into one seamless product.