1. **HTML**

* Basic HTML Tags
* Semantic HTML
* Web APIs
  + Storage API
  + Web Workers

1. **CSS**

* Box Model
* Flexbox & Grid
* Position Property
* Responsive Design Techniques
* Utility Frameworks: TailwindCSS, Bootstrap
* CSS Preprocessors: SASS, LESS

1. **JavaScript Core Concepts**

* Data Types
* Scopes: Global, Local/Functional, Block, Lexical, Dynamic
* Hoisting
* Closures
* this keyword context
* Call, Apply, and Bind
* Classical vs Prototypical Inheritance
* OOP in JS

1. **Asynchronous JavaScript**

* Event Loop: Event Queues, Micro/Macro Tasks, Web API, Call Stack
* Async Operations: Callbacks, Promises, Async/Await

1. **Functional Programming in JS**

* Pure Functions
* Higher-Order Functions
* Function Composition
* Immutability / Side Effects

1. **ES6+ Features**

* Arrow Functions
* Destructuring
* Spread and Rest Operators

1. **DOM & Client-Side JavaScript**

* DOM Manipulation
* Event Listeners
* Event Capturing & Bubbling
* Event Delegation
* Debouncing & Throttling
* Critical Rendering Path

1. **TypeScript**

* Basic Types
* Variables
* Functions
* Classes
* Interfaces
* Types vs Interfaces
* Enums
* Union and Intersection Types

1. **React**

* JSX
* Components: Class vs Function
* Props and State
* Lifecycle Methods
* Hooks:
  + useState
  + useEffect
* Composition vs Inheritance
* Controlled vs Uncontrolled Components
* Higher-Order Components (HOC)
* Virtual DOM
* Lists and Keys
* Reconciliation

1. **React Routing**

* React Router
* <Routes> and <Route>
* Routing Hooks

1. **State Management**

* Redux / Redux Toolkit
* Basics of Middleware
* Redux vs Context API

1. **Testing**

* Unit Testing
* React Testing Library / Jest
* FIRST & AAA Principles

1. **Node.js**

* Event Loop in Node
* NPM
* File System
* Async Operations: Promises, Async/Await
* Error Handling

1. **Express.js**

* Middleware
* Routing
* Create RESTful APIs: GET, POST, PUT, DELETE
* Authentication & Authorization (JWT)

1. **Architecture & Infrastructure**

* Basics of Microservices Architecture
* Basics of AWS Serverless Architecture

1. **MongoDB**

* MongoDB Basics
* CRUD Operations

1. **Version Control with Git**

* Git Basics
* Branching Strategies (e.g., GitFlow)
* Resolving Merge Conflicts

1. **Design Principles**

* SOLID
* KISS
* DRY
* YAGNI

1. **Design Patterns**

* Common JavaScript Design Patterns

1. **CI/CD**

* CI/CD Process and Tools

1. **SDLC Methodologies**

* Agile / Scrum

**Node.js & Express.js: Core Concepts**

**Why Use Express.js with Node.js?**

* What advantages does Express provide over plain Node.js?
* How does Express simplify server-side development?

**Express.js Features**

* What is middleware in Express? Provide use-cases.
* How is routing handled in Express?
* How do you serve static files in an Express application?

**Routing Without Express**

* How can you implement routing in a Node.js application without using Express?

**Asynchronous Programming & Node.js Internals**

**Event Loop & Concurrency**

* Explain the **Node.js Event Loop** and its phases.
* What is the role of the **libuv** library in Node.js?
* What is a **single-threaded** model in Node.js?
* What are the **drawbacks of single-threaded architecture**?

**Promises vs Async/Await**

* What are the differences between **Promises** and **async/await**?
* Difference between **asynchronous** and **non-blocking** code?

**Multi-tasking in APIs**

* How would you **execute two parallel processes** (e.g., update DB and upload file) in a single API call, and send a response after both are complete?

**Environment, Configuration, and Security**

**Environment**

* What is the purpose of NODE\_ENV?

**Security in Node.js APIs**

* What are the common ways to **secure APIs** in Node.js?
* How is **JWT** implemented in an application?
* What is **role-based access control (RBAC)**? How would you implement it?

**HTTP & RESTful API Design**

**HTTP Knowledge**

* Explain commonly used **HTTP status codes** (e.g., 200, 201, 400, 401, 403, 404, 500).

**RESTful API Design**

* How would you design an API to update a resource?
* What is the difference between **PUT** and **PATCH**?

**AWS Cloud & Serverless**

**Serverless Architecture**

* What is **serverless computing**?
* Why is **serverless** (e.g., AWS Lambda) preferred in certain use cases?

**AWS Lambda & API Gateway**

* How does an **API Gateway** work?
* Describe a recent use of **Lambda functions**.
* How is **authentication** handled with **AWS Cognito** in Lambda?

**AWS Services & Usage**

* How have you used the following in your projects:
  + **S3** (for file storage)
  + **SQS vs SNS** (messaging/notifications)
* From when have you been using **AWS services**?

how backend deploy on microservices

why microservices

how the different api is being communicated with each other

**🧠 Coding Challenges: UI and Logic Tasks**

**1. Counter Component with START/STOP Functionality**

**Requirements:**

* Create a component with two buttons: **START** and **STOP**.
* On clicking **START**, a counter should increment values like: 0, 1, 2, 3... every second.
* On clicking **STOP**, the counter should stop.
* If STOP is clicked when the counter value is **10**, it must stop at **10** and halt all further incrementing.

**2. Full Height Vertical Button Alignment**

**Requirements:**

* Display **three buttons** (Top, Middle, Bottom) in a vertical column.
* Buttons should be **center-aligned horizontally** and **occupy full viewport height**.
* Position:
  + **First button** at the **top**.
  + **Second button** in the **middle**.
  + **Third button** at the **bottom**.

**3. Simple Search Component**

**Requirements:**

* Create a search input box.
* Dynamically filter and display search results as the user types (assume static data or prop-based input).

**4. Debounced Search Component**

**Requirements:**

* Similar to challenge 3, but the search logic should execute **only after a delay** (e.g., 300ms) using **debouncing**.
* Prevent API calls or filtering until the user pauses typing.

**5. TODO List Component**

**Requirements:**

* Input field for user to **enter text** and **add items** to a list.
* Display the list just **below the text field**.
* Clicking a list item should **strike through** the text (mark as done).
* Clicking a **struck item** again should **un-strike** (mark as not done).
* Display:
  + **Total number** of items.
  + **Number of completed (struck)** items.