**🔹 HTML**

* **Semantic Tags**  
  Tags that clearly describe their meaning in a human- and machine-readable way.  
  *Examples:* <header>, <nav>, <section>, <article>, <footer>, <aside>, <main>
* **Difference between <div> and semantic tags (section, article, header)**
  + <div>: Generic container, no semantic meaning.
  + <section>: Thematic grouping of content.
  + <article>: Self-contained, independent content.
  + <header>: Introductory content or navigation.
* **Structure of an HTML Document**

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Document Title</title>

</head>

<body>

<!-- Content -->

</body>

</html>

**🔹 CSS**

* **CSS Box Model**  
  Content + Padding + Border + Margin
* **CSS position property values**
  + static (default, normal flow)
  + relative (offset relative to itself)
  + absolute (relative to nearest positioned ancestor)
  + fixed (relative to viewport, stays on scroll)
  + sticky (switches between relative and fixed based on scroll)
* **Units: em, rem, %, px**
  + px: absolute pixels
  + em: relative to font-size of parent
  + rem: relative to root (<html>) font-size
  + %: relative to parent property (e.g. width)
* **Responsive Design Techniques**
  + Media Queries
  + Flexbox
  + CSS Grid

**🔹 JavaScript**

**Core Concepts**

* **Scopes:**
  + Global scope
  + Function scope
  + Block scope (with let and const)
* **Closure:**  
  Function accessing variables from outer scope even after outer function has executed.

function outer() {

let count = 0;

return function inner() {

count++;

return count;

}

}

const counter = outer();

console.log(counter()); // 1

console.log(counter()); // 2

* **Callback vs Promise vs Async/Await**
  + Callback: function passed to be executed later
  + Promise: object representing eventual completion/failure
  + Async/Await: syntactic sugar over promises for cleaner async code
* **Function Composition (pipe):**

const pipe = (...fns) => x => fns.reduce((v, f) => f(v), x);

* **Web Workers:**  
  Background threads for running JS without blocking UI.
* **Prototype & Extending Array:**

Array.prototype.lengthCustom = function() {

return this.reduce((acc) => acc + 1, 0);

};

* **OOP in JS:**  
  Classes, constructors, inheritance using extends.
* **this context:**
  + Global: window (or undefined in strict mode)
  + Object method: object owning method
  + Arrow function: lexically bound from parent scope
  + Constructor: new instance
* **Spread vs Rest:**
  + Spread: expand iterable into arguments/array
  + Rest: collect arguments into array
* **Destructuring:**  
  Extracting array/object values into variables.
* **Function vs Class Components in React:**
  + Function components are simpler, hooks-based
  + Class components use lifecycle methods and state

**Functions & Patterns**

* **Higher-Order Function:** Function that takes or returns a function.

function hof(fn) {

return function() {

console.log('Before');

fn();

console.log('After');

}

}

* **Closure based counter:**

function createCounter() {

let count = 0;

return () => ++count;

}

const counter = createCounter();

* **Currying, Debouncing, Throttling:**  
  Techniques for function control (ask if required).
* **Call, Apply, Bind:**
  + call: invoke with this and args individually
  + apply: invoke with this and args as array
  + bind: returns new function with bound this

**🔹 TypeScript**

* **interface:** Shape of an object.

interface Employee {

id: number;

name: string;

salary?: number; // optional

}

* **enum:**

enum Role {

Admin,

User,

Guest

}

* **Generics:**

function identity<T>(arg: T): T {

return arg;

}

* **Difference type vs interface:**  
  Interfaces are extendable and declaration-mergable; types are more flexible for unions/intersections.

**🔹 React**

**Core Concepts**

* **Why React?**  
  Component-based UI, fast rendering with Virtual DOM, declarative syntax.
* **Props vs State:**  
  Props are immutable inputs; State is mutable data owned by component.
* **Controlled vs Uncontrolled Components:**  
  Controlled components have value driven by React state, uncontrolled by DOM.
* **Composition vs Inheritance:** React encourages composition.
* **Higher Order Components (HOC):**  
  Functions that take a component and return enhanced component.
* **Virtual DOM vs Real DOM:**  
  Virtual DOM is a lightweight JS representation used to minimize real DOM manipulations.

**Hooks & Patterns**

* **useState, useEffect:**
  + useState: state management in functional components
  + useEffect: side effects and lifecycle management
* **Custom Hooks (useForm, useReload):** Encapsulate reusable logic.
* **Fetch inside useEffect:** Use async function inside useEffect.
* **React Router:**
  + Basic routing with <Routes>, <Route>
  + Params, Index routes, Layout and Outlet for nested UI

**🔹 Node.js & Express**

* **Node.js:** JavaScript runtime built on Chrome V8, event-driven, non-blocking I/O.
* **Express:** Minimal web framework for routing and middleware.
* **Define routes in Express:**
* **Event Loop:** Node’s async task scheduler.
* **Middleware:** Functions that intercept requests for logging, auth, etc.
* **Event Loop:** Node’s mechanism to handle async callbacks without blocking.
* **Error Handling:** Callback style, Promises, Async/Await with try/catch.

**🔹 MongoDB**

* **Why MongoDB over SQL?**  
  Flexible schema, JSON-like documents, horizontal scaling.
* **Schema, Model, Collection, Document:**
  + Document: a record (JSON)
  + Collection: group of documents
  + Schema: structure definition for documents
  + Model: interface for schema to interact with DB
* **Mongoose Schema example:**
* **Aggregation functions:** Group, sum, avg, etc.
* **ACID support:** Limited but available in transactions (v4+).

**🔹 Git & CI/CD**

* **CI/CD:** Automate build, test, deployment to improve workflow.
* **Git stash:** Temporarily store uncommitted changes.
* **Git commands:** clone, commit, push, pull, rebase, merge, reset, revert, reflog.
* **Difference between reset, revert, reflog:**
  + reset: move HEAD pointer, possibly deleting commits
  + revert: create a new commit that undoes changes
  + reflog: history of HEAD moves (undo/reset recovery)

**🔹 Security & APIs**

* **Common vulnerabilities:**
  + XSS (Cross-Site Scripting)
  + CSRF (Cross-Site Request Forgery)
  + CORS (Cross-Origin Resource Sharing issues)
  + HTTPS: Secure communication
* **Protections:** Input sanitization, tokens, CORS policies, HTTPS setup.
* **Web APIs:** Browser APIs like Web Workers, Geolocation, localStorage.
* **API Gateway / Proxy:** Routing API calls to correct backend services.

**Testing**

* **Unit Testing:** Testing smallest units with tools like Jest.
* **Assertions:** expect(), .toBe(), .toEqual().
* **Testing API calls in React (useEffect):** Mock fetch or axios calls.

**General & Project Discussion**

* **Introduce yourself and explain your project.**
* **SOLID Principles:**  
  Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
* **DRY Principle:** Don’t Repeat Yourself.
* **Design Patterns:** Singleton, Factory, Proxy, Observer, etc.