**REACT QUESTIONS**

**1.What is React**

* Library-> for building user interfaces
* you write small, reusable pieces of UI (called components) that update automatically when data changes.

**2. Why React**

* + React helps avoid messy manual DOM updates,
  + makes UI predictable by using state,
  + reuse UI pieces.
  + That makes large apps easier to build and maintain.

**3.Library vs Framework**

* **Library**: a toolkit you call when you need it. You control the flow.
* **Framework**: gives the structure and calls your code (Inversion of Control). The framework decides the app flow.

React = **library**. Next.js/Angular = **framework**.

**4.Single Page Application (SPA) vs Multi Page Application (MPA)**

 **SPA**: The app loads once and different screens are shown without full page reloads.

Navigation is fast, like switching views inside the same page.

 **MPA**: Each page is a new HTML document;

clicking a link reloads the browser and fetches a new page.

**5.Installation using Vite**

 **Vite** is a modern build tool and development server.

 faster than older tools like Create React App (CRA)

because it uses **native ES modules** and **on-demand file serving**.

 It supports **hot module replacement (HMR)** — meaning changes you make in code appear instantly without refreshing.

**STEP BY STEP INSTALLATION**

 npm create vite@latest → Creates a project with Vite.

 cd my-app → Opens project folder.

 npm install → Downloads needed packages.

 npm run dev → Starts live server with instant updates.

**6.What is Bundler,**

Features of Bundler,

different Types of bundlers (Vite, Parcel, Webpack)

A bundler takes all your project files (HTML, CSS, JS, images) and **packs them together** into one (or few) files so your website loads faster.

### ****Types****

1. **Vite** – Fast, modern (used for development and build).
2. **Parcel** – Zero-config, automatically detects settings.
3. **Webpack** – Powerful, highly configurable (used in large projects).

**7.Folder structure**

The folder structure is how your files are organized so you can work easily and find things quickly.

**my-app/**

**│**

**├── node\_modules/ # Installed dependencies**

**├── public/ # Static files (images, favicon)**

**├── src/ # Main source code**

**│ ├── App.jsx # Root component**

**│ ├── main.jsx # Entry point**

**│ ├── assets/ # Images, CSS**

**│ └── components/ # Reusable UI parts**

**├── index.html # Main HTML file**

**├── package.json # Project info + dependencies**

**└── vite.config.js # Vite configuration**

**8.Package.json, package-lock.json**

 **package.json** contains:

Project name, version, scripts (npm start), dependencies (React, Vite).

 **package-lock.json**:

Locks versions of each package to avoid future mismatches.

**9.Dependency, dev- dependency, scripts**

* **Dependency**: Things your project needs to work when running in production.
* **Dev-dependency**: Things needed only while developing the project (not in production).
* **Scripts**: Shortcuts to run commands (like starting your app or building it).

**10.Npm, npx ,yarn**

 **npm**: Tool to install and manage packages.

 **npx**: Runs a package without installing it permanently.

 **yarn**: Another tool like npm but faster and with extra features.

**11.Babel**

* + A JavaScript compiler that converts ES6+ (modern JavaScript) into ES5 (older JavaScript).
  + Used in React to convert **JSX** into JavaScript.

**12.React fiber**

React Fiber is the brain of React that helps it update the user interface smoothly and quickly.

**13.Difference between ^ and ~**

 **^**:caret Updates to the latest minor version.

 **~**:tilde ,, patch version.

**14.CSR and SSR**

**CSR**:

* Your browser downloads an empty HTML + JavaScript,
* then JavaScript **builds the page**.

Example: Facebook feed loading after the spinner.

**SSR**:

The server **already builds the page** and sends ready HTML to your browser.

Example: News websites loading almost instantly.

**15.Features of react**

**16.Jsx, rules of jsx**

* + JSX looks like HTML but works in JavaScript.
  + It helps write UI easily inside React code.

**Rules of JSX**

* + - Only **one root element**.
    - Tags must be **closed**.
    - Use **camelCase** for attributes (className not class).
    - JavaScript inside {}.

**17.Component and its rules**

A **component** is a small, reusable piece of UI in React.

Example: A button, navbar, or footer.

**Rules of Components**

* Component name starts with **capital letter**.
* Must return JSX.
* Reusable.
* Can take **props** (inputs).

**18.Function and class component**

 **Functional Component** → Just a JavaScript function that returns UI. Simple and preferred in modern React.

 **Class Component** → Uses ES6 class syntax, has more boilerplate, and uses lifecycle methods instead of hooks.

**19.Life cycle method**

Life cycle methods are like **events** in a component’s life:

* When it **appears** on the screen (mount)
* When it **updates** (props/state change)
* When it **disappears** (unmount)

 componentDidMount() → after first render (good for API calls)

 componentDidUpdate() → after state/props change

 componentWillUnmount() → cleanup (remove listeners)

**20.Props ,**

prop types,

children prop

default props

 **Props** → Information passed from parent to child component.

 **PropTypes** → Type-checking for props to catch errors early.

 **Children Prop** → Anything between the opening and closing tag of a component.

 **Default Props** → Fallback values when parent doesn’t pass a prop.

**21.Vdom, diffusing algorithm, reconciliation process**

 **VDOM (Virtual DOM)**: A lightweight copy of the actual DOM kept in memory.

 **Diffing Algorithm**: Compares old VDOM and new VDOM to find changes.

 **Reconciliation**: Updates only the changed parts in the real DOM for efficiency.

**22.Hook,**

**State management –** useState, useReducer, useContext

**useState** lets a component hold and update local state (like a variable that, when changed, updates the UI).

useReducer is like useState but best for complex state logic or when next state depends on previous state; it uses a reducer function and dispatch actions.

useContext lets components read values from a shared place (context) without passing props through every level.

**Side effects –**useEffect

useEffect runs code after the component renders — used for network calls, subscriptions, timers, or DOM work.

**Useref**

useRef gives you a stable object for storing a value across renders, often used to access DOM nodes.

**Optimization-** usecallback, usememo

useCallback remembers a function so it isn’t recreated every render (helps child components that depend on stable function props).

useMemo caches the result of a calculation so it doesn’t recompute unless inputs change.

React.memo wraps a component and skips re-renders when props are shallowly equal.

**Routing**- useNavigata, useParams

 useNavigate lets you programmatically change pages (like history.push).

 useParams gets route parameters from the URL (like /:id).

**23.Higher order component & custom hooks**

**Higher order component**

function -> that takes a component and returns a new component . with extra behaviour

**custom hooks :**

reusable function-> that encapsulates(bind) logic you can use across components

**24.form [controlled, uncontrolled] and form validation**

 **Controlled**: React state controls form fields. The input value comes from state; change events update state.

 **Uncontrolled**: Browser manages the input; you read values when needed using refs or FormData.

 **Validation**: Check user input (required, email format) and show errors before submission.

**25.React Fragments**

* return multiple elements from a component
* without adding extra DOM nodes (no extra <div> wrappers).

**26.axios**

* Library->to make HTTP requests (GET/POST).
* It's easier than fetch
  + - automatic JSON parsing,
    - interceptors
    - timeout support.

**27.pure components**

* Pure Component only re-renders when its props/state change
* React compares old and new props shallowly — if nothing changed, it skips re-rendering.

**28.Redux tool kit**

Redux Toolkit (RTK) is the official, recommended way to write Redux code. It reduces boilerplate by giving you createSlice, configureStore, and helpers for async actions.

Eg:Fetch and store product list globally so many components can access products without prop-drilling.