<u>**Harshil Tandel**</u>

Table Contents:

Topic:Inception

Q: What is Emmet?

A: Emmet is the essential toolkit for web developers. It allows you to type shortcuts that are then expanded into full pieces of code for writing HTML and CSS, based on an abbreviation structure most developers already use that expands into full-fledged HTML markup and CSS rules.

Q: Difference between a Library and Framework?

A: A library is a collection of packages that perform specific operations whereas a framework contains the basic flow and architecture of an application. The major difference between them is the complexity. Libraries contain a number of methods that a developer can just call whenever they write code. React js is a library and Angular is a framework. The framework provides the flow of a software application and tells the developer what it needs and calls the code provided by the developer as required. If a library is used, the application calls the code from the library.

Q: What is CDN? Why do we use it?

A: A content delivery network (CDN) refers to a geographically distributed group of servers that work together to provide fast delivery of Internet content. The main use of a CDN is to deliver content through a network of servers in a secure and efficient way.

Q: Why is React known as React?

A: React is named React because of its ability to react to changes in data. React is called React because it was designed to be a declarative, efficient, and flexible JavaScript library for building user interfaces. The name React was chosen because the library was designed to allow developers to "react" to changes in state and data within an application, and to update the user interface in a declarative and efficient manner. React is a JavaScript-based UI development library. Facebook and an open-source developer community run it.

Q: What is crossorigin in script tag?

A: The crossorigin attribute sets the mode of the request to an HTTP CORS Request. The purpose of the crossorigin attribute is to share the resources from one domain to another domain. Basically, it is used to handle the CORS request. It is used to handle the CORS request that checks whether it is safe to allow for sharing the resources from other domains.

Q: What is difference between React and ReactDOM?

A: React is a JavaScript library for building User Interfaces whereas ReactDOM is also a JavaScript library that allows React to interact with the DOM. The react package contains React.createElement(), React.Component, React.Children, and other helpers related to elements and component classes. You can think of these as the isomorphic or universal helpers that you need to build components. The react-dom package contains ReactDOM.render(), and in react-dom/server we have server-side rendering support with ReactDOMServer.renderToString() and ReactDOMServer.renderToStaticMarkup().

Q: What is the difference between react.development.js and react.production.js files via CDN?

A: Development is the stage of an application before it's made public while production is the term used for the same application when it's made public. Development build is several times (maybe 3-5x) slower than the production build.

Q: What is async and defer?

A: Async - The async attribute is a boolean attribute. The script is downloaded in parallel (in the background) to parsing the page, and executed as soon as it is available (do not block HTML DOM construction during downloading process) and don’t wait for anything.

Defer - The defer attribute is a boolean attribute. The script is downloaded in parallel (in the background) to parsing the page, and executed after the page has finished parsing (when browser finished DOM construction). The defer attribute tells the browser not to wait for the script. Instead, the browser will continue to process the HTML and build DOM.

Unless you're supporting ancient legacy systems, always add type="module" to all your script tags:

html
```
<script type="module" src="main.js"></script> and place the tag inside <head>
```

html
```
<script defer nomodule> can be used as a legacy fallback.
```

As the name suggests, it allows you to import modules, which makes it easier to organize your code. It enables strict mode by default. This makes your code run faster and reports more runtime errors instead of silently ignoring them. It executes your code only after the DOM has

initialized, which makes DOM manipulation easier. Thanks to this, you won't need to listen to load/readystatechange/DOMContentLoaded events. It prevents top-level variables from implicitly polluting the global namespace. It allows you to use top-level await in supported engines. It loads and parses your code asynchronously, which improves load performance.

Topic:Igniting our App

Q1: What is `NPM`?

A: NPM is a tool used for package management and the default package manager for Node projects. NPM is installed when NodeJS is installed on a machine. It comes with a command-line interface (CLI) used to interact with the online database of NPM, called the NPM Registry, which hosts public and private packages. To add or update packages, we use the NPM CLI to interact with this database. An alternative to NPM is `yarn`.

How to initialize `npm`?

A:

```
npm init
```

`npm init -y` can be used to skip the setup step. NPM takes care of it and creates the `package.json` file automatically but without configurations.

Q2: What is `Parcel/Webpack`? Why do we need it?

A: Parcel/Webpack is a type of web application bundler used for development and production purposes. It offers blazing-fast performance utilizing multicore processing and requires zero configuration. Parcel can take any type of file as an entry point, but an HTML or JavaScript file is a good place to start. These bundlers power our applications with various functionalities and features.

Parcel Features:

- HMR (Hot Module Replacement) - parcel keeps track of file changes via a file watcher algorithm and renders the changes in the files
- File watcher algorithm - made with C++
- Minification
- Cleaning our code
- DEV and production Build
- Super fast building algorithm
- Image optimization
- Caching while development
- Compresses

- Compatible with older versions of browsers
- HTTPS in dev
- Port Number
- Consistent hashing algorithm
- Zero Configuration
- Automatic code splitting

Installation commands:

- Install:

```
npm install -D parcel
```

`-D` is used for development and as a development dependency.

- Parcel Commands:
    - For development build:

```
npx parcel <entry_point>
```

- For production build:

```
npx parcel build <entry_point>
```

Q3: What is `.parcel-cache`?

A: `.parcel-cache` is used by Parcel (bundler) to reduce the building time. It stores information about your project when Parcel builds it, so that when it rebuilds, it doesn't have to re-parse and re-analyze everything from scratch. This is a key reason why Parcel can be so fast in development mode.

Q4: What is `npx`?

A: `npx` is a tool that is used to execute packages. It comes with NPM, and when you install NPM above version 5.2.0, npx is installed automatically. It is an NPM package runner that can execute any package from the NPM registry without even installing that package.

Q5: What is the difference between `dependencies` vs `devDependencies`?

A: `Dependencies` contain libraries and frameworks that your app is built on and needs to function effectively, such as Vue, React, Angular, Express, JQuery, etc. `DevDependencies` contain modules/packages needed during development, such as Parcel, Webpack, Vite, Mocha. These packages are necessary only while you are developing your project, not in production. To save a dependency as a devDependency on installation:

```
npm install --save-dev
```

instead of:

```
npm install --save
```

Q6: What is `Tree Shaking`?

A: `Tree shaking` is the process of removing the unwanted code that is not used while developing the application. In computing, tree shaking is a dead code elimination technique applied when optimizing code.

Q7: What is `Hot Module Replacement`?

A: `Hot Module Replacement (HMR)` exchanges, adds, or removes modules while an application is running, without a full reload. This can significantly speed up development by retaining the application state, which is lost during a full reload.

Q8: List down your favorite `5 superpowers of Parcel` and describe any 3 of them in your own words.

A: `5 superpowers of Parcel`:

- `HMR (Hot Module Replacement)` - Adds, or removes modules while an application is running, without a full reload.
- `File watcher algorithm` - File Watchers monitor directories on the file system and perform specific actions when desired files appear.

- `Minification` - Minification is the process of minimizing code and markup in your web pages and script files.
- `Image optimization`
- `Caching while development`

Q9: What is `.gitignore`? What should we `add and not add` into it?

A: The `.gitignore file` is a text file that tells `Git` which files or folders to `ignore` in a project during `commit to the repository`. Files to consider adding to a .gitignore file are any files that do not need to get committed, such as security key files and API keys. `package-lock.json` should `not be added` to your `.gitignore` file.

Example of a .gitignore file:

```
# Ignore Mac system files

.DS_store

# Ignore node_modules folder

node_modules

# Ignore all text files

*.txt

# Ignore files related to API keys

.env

# Ignore SASS config files

.sass-cache
```

Q10: What is the difference between `package.json` and `package-lock.json`?

A: `package.json`:

- Mandatory for every project
- Contains basic information about the project
- Includes application name, version, scripts

`package-lock.json`:

- Automatically generated for operations where NPM modifies the node_module tree or package-json.
- Generated after an npm install
- Ensures future developers and automated systems can download the same dependencies as the project.
- Allows reverting to past versions of dependencies without actually committing the node_modules folder.
- Records the exact version of the installed packages.

~ or **^** in `package.json` file: These are used with the versions of the package installed.

For example, in `package.json` file:

json

Copy code

```
"dependencies": {

    "react": "^18.2.0",

    "react-dom": "^18.2.0"

  }
```

- **~** : Approximately equivalent to the version, will update you to all future patch versions, without incrementing the minor version.
- **^** : Compatible with the version, will update you to all future minor/patch versions, without incrementing the major version.

If none of them is present, only the version specified in the `package.json` file is used in the development.

Q11: Why should I not modify `package-lock.json`?

A: The `package-lock.json` file contains information about the dependencies and their versions used in the project. Deleting it would cause dependencies issues in the production environment. So don't modify it; it's handled automatically by NPM.

Q12: What is `node_modules`? Is it a good idea to push that on git?

A: The `node_modules` folder is like a cache for the external modules that your project depends upon. When you npm install them, they are downloaded from the web and copied into the node_modules folder, and Node.js looks for them there when you import them. Do not push `node_modules` to GitHub because it contains lots of files (more than 100 MB), which will cost you memory space.

Q13: What is the `dist` folder?

A: The `/dist` folder contains the minimized version of the source code. The code present in the `/dist` folder is actually the code used in production web applications. Along with the minified code, the /dist folder also includes all the compiled modules that may or may not be used with other systems.

Q14: What is `browserslist`?

A: `Browserslist` is a tool that allows specifying which browsers should be supported in your frontend app by specifying "queries" in a config file. It's used by frameworks/libraries such as React, Angular, and Vue but is not limited to them.

Topic: Laying the Foundation

Q: What is JSX?

A: JSX stands for JavaScript XML. JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods. JSX makes it easier to write and add HTML in React. JSX converts HTML tags into react elements.

Example 1 using JSX:

```
const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myElement);
```

Example 2 Without JSX:

```
const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myElement);
```

Q: Superpowers of JSX.

A: Using JSX, you can write markup inside Javascript, providing you with a superpower to write logic and markup of a component inside a single .jsx file. JSX is easy to maintain and debug.

Example:

```
function greeting(user) {


  //JSX

  return <h1>{user}, How are you!!!</h1>;

}
```

Q: Role of `type` attribute in script tag? What options can I use there?

A: The `type` attribute specifies the type of the script. The type attribute identifies the content between the `<script>` and `</script>` tags. It has a default value which is “text/javascript”.

`type` attribute can be of the following types:

`text/javascript`: It is the basic standard of writing javascript code inside the `<script>` tag.
Syntax:

```
<script type="text/javascript"></script>
```

-
- `text/ecmascript`: This value indicates that the script is following the `EcmaScript` standards.
- `module`: This value tells the browser that the script is a module that can import or export other files or modules inside it.
- `text/babel`: This value indicates that the script is a babel type and requires babel to transpile it.
- `text/typescript`: As the name suggests, the script is written in `TypeScript`.

Q: `{TitleComponent}` vs `{<TitleComponent/>}` vs `{<TitleComponent></TitleComponent>}` in `JSX`.

A: The difference is stated below:

- `{TitleComponent}`: This value describes the `TitleComponent` as a javascript expression or a variable. The `{}` can embed a javascript expression or a variable inside it.
- `<TitleComponent/>`: This value represents a Component that is basically returning some JSX value. In simple terms, `TitleComponent` is a function that is returning a JSX value. A component is written inside the `{< />}` expression.
- `<TitleComponent></TitleComponent>`: `<TitleComponent />` and `<TitleComponent></TitleComponent>` are equivalent only when `<TitleComponent />` has no child components. The opening and closing tags are created to include the child components.

Example:

```
<TitleComponent>

    <FirstChildComponent />

    <SecondChildComponent />

    <ThirdChildComponent />

</TitleComponent>
```

Topic : React.js Essentials: Key Concepts and FAQs

Q: Is JSX mandatory for React?

A: JSX is an Extension Syntax that allows writing HTML and JavaScript together easily in React and is used to create React elements. These elements are then rendered to the React DOM. Each JSX element is just to make use of React easy and for calling `React.createElement(component, props, …children)` with less work. So, anything that is done with JSX can also be done with just plain JavaScript. So JSX is not mandatory but is used for writing better and clean code instead of writing code using `React.CreateElement`. Example of JSX:

```
const sample = <h2>Greetings</h2>;
```

Q: Is ES6 mandatory for React?

A: ES6 is not mandatory for React but is highly recommendable. The latest projects created on React rely a lot on ES6. React uses ES6, and you should be familiar with some of the new features like Classes, Arrow Functions, Variables (let, const). ES6 stands for ECMAScript 6. ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, published in 2015.

Q: {TitleComponent} vs {<TitleComponent/>} vs {<TitleComponent></TitleComponent>} in JSX.

A: The Difference is stated below:

- `{TitleComponent}`: This value describes the `TitleComponent` as a JavaScript expression or a variable or React element. The `{}` can embed a JavaScript expression or a variable or React element inside it.
- `<TitleComponent/>`: This value represents a Component that is basically returning Some JSX value. In simple terms `TitleComponent` a function that is returning a JSX value. If component is written inside the `{< />}` expression.

- `<TitleComponent></TitleComponent>`: `<TitleComponent />` and `<TitleComponent></TitleComponent>` are equivalent only when `<TitleComponent />` has no child components. The opening and closing tags are created to include the child components. Example:

```
<TitleComponent>

    <FirstChildComponent />

    <SecondChildComponent />

    <ThirdChildComponent />

</TitleComponent>
```

Q: How can I write comments in JSX?

A: JSX comments are written as follows:

- `{/* */}` - for single or multiline comments Example:

```
{/* A JSX comment */}

{/*

  Multi

  line

  JSX

  comment

*/}
```

Q: What is `<React.Fragment></React.Fragment>` and `<></>`?

A: `<React.Fragment></React.Fragment>` is a feature in React that allows you to return multiple elements from a React component by allowing you to group a list of children without adding extra nodes to the DOM. `<></>` is the shorthand tag for `React.Fragment`. The only difference between them is that the shorthand version does not support the key attribute. Example:

```
return (

    <React.Fragment>

        <Header />

        <Navigation />

        <Main />

        <Footer />

    </React.Fragment>

);


return (

    <>

        <Header />

        <Navigation />

        <Main />

        <Footer />

    </>

);
```

Q: What is Reconciliation in React?

A: Reconciliation is the process through which React updates the Browser DOM and makes React work faster. React uses a `diffing algorithm` so that component updates are predictable and faster. React would first calculate the difference between the real DOM and the copy of DOM (Virtual DOM) when there's an update of components. React stores a copy of Browser DOM which is called `Virtual DOM`. When we make changes or add data, React creates a new Virtual DOM and compares it with the previous one. Comparison is done by `Diffing Algorithm`. React compares the Virtual DOM with Real DOM. It finds out the changed nodes and updates only the changed nodes in Real DOM leaving the rest nodes as it is. This process is called Reconciliation.

Q: What is React Fiber?

A: React Fiber is a concept of ReactJS that is used to render a system faster, smoother and smarter. The Fiber reconciler, which became the default reconciler for React 16 and above, is a complete rewrite of React's reconciliation algorithm to solve some long-standing issues in React. Because Fiber is asynchronous, React can:

- Pause, resume, and restart rendering work on components as new updates come in
- Reuse previously completed work and even abort it if not needed
- Split work into chunks and prioritize tasks based on importance

Q: Why do we need keys in React?

A: A `key` is a special attribute you need to include when creating lists of elements in React. Keys are used in React to identify which items in the list are changed, updated, or deleted. In other words, we can say that keys are unique Identifier used to give an identity to the elements in the lists. Keys should be given to the elements within the array to give the elements a stable identity.

Example:

```
<li key={0}>1</li>

<li key={1}>2</li>

<li key={2}>3</li>
```

Q: Can we use index as keys in React?

A: Yes, we can use the `index as keys`, but it is not considered as a good practice to use them because if the order of items may change. This can negatively impact performance and may cause issues with component state. Keys are taken from each object which is being rendered. There might be a possibility that if we modify the incoming data react may render them in unusual order.

Q: What is props in React? Ways to.

A: props stands for properties. Props are arguments passed into React components. props are used in React to pass data from one component to another (from a parent component to a child component(s)). They are useful when you want the flow of data in your app to be dynamic. Example:

```
function App() {

  return (

    <div className="App">

        <Tool name="Chetan Nada" tool="Figma"/> // name and tool are
props

    </div>

  )

}
```

Q: What is Config Driven UI?

A: `Config Driven UI` are based on the configurations of the data application receives. It is rather a good practice to use config driven UIs to make application for dynamic. It is a very common & basic approach to interact with the User. It provides a generic interface to develop things which help your project scale well. It saves a lot of development time and effort. A typical login form, common in most of the Apps. Most of these forms also get frequent updates as the requirements increase in terms of Form Validations, dropdown options,.. or design changes.

Q: Difference between Virtual DOM and Real DOM?

A: DOM stands for `Document Object Model`, which represents your application UI and whenever the changes are made in the application, this DOM gets updated and the user is able to visualize the changes. DOM is an interface that allows scripts to update the content, style, and structure of the document.

- `Virtual DOM`
    - The Virtual DOM is a light-weight abstraction of the DOM. You can think of it as a copy of the DOM, that can be updated without affecting the actual DOM. It has all the same properties as the real DOM object, but doesn’t have the ability to write to the screen like the real DOM.
    - Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.
    - Reconciliation is a process to compare and keep in sync the two files (Real and Virtual DOM). Diffing algorithm is a technique of reconciliation which is used by React.
- `Real DOM`
    - The DOM represents the web page often called a document with a logical tree and each branch of the tree ends in a node and each node contains object programmers can modify the content of the document using a scripting language like JavaScript and the changes and updates to the dom are fast because of its tree-like structure but after changes, the updated element and its children have to be re-rendered to update the application UI so the re-rendering of the UI which make the dom slow all the UI components you need to be rendered for every dom update so real dom would render the entire list and not only those item that receives the update .

Topic :ES6 Exports and React Hooks

Q: What is the difference between Named export, Default export, and * as export?

A: ES6 provides us to import and export a module and use it in other files. ES6 provides two ways to export a module from a file: named export and default export.

1. **Named export:** One can have multiple named exports per file. Then import the specific exports they want surrounded in {} braces. The name of the imported module has to be the same as the name of the exported module.
   Example:
   - Exporting:
     - export const MyComponent = () => {}
     - export const MyComponent2 = () => {}
   - Importing:
     - import { MyComponent } from "./MyComponent";
     - import { MyComponent, MyComponent2 } from "./MyComponent";
     - import { MyComponent2 as MyNewComponent } from "./MyComponent";
2. **Default export:** One can have only one default export per file. The naming of import is completely independent in default export and we can use any name we like.
   Example:
   - Exporting:
     - const MyComponent = () => {}
     - export default MyComponent;
   - Importing:
     - import MyComponent from "./MyComponent";
3. *** as export:** It is used to import the whole module as a component and access the components inside the module.
   Example:
   - Exporting:
     - export const MyComponent = () => {}
     - export const MyComponent2 = () => {}
     - export const MyComponent3 = () => {}
   - Importing:
     - import * as MainComponents from "./MyComponent";
     - Usage in JSX:
       - <MainComponents.MyComponent />
       - <MainComponents.MyComponent2 />
       - <MainComponents.MyComponent3 />

Named export and default export can be used together. Example:

- Exporting:
    - export const MyComponent2 = () => {}
    - const MyComponent = () => {}
    - export default MyComponent;
- Importing:
    - import MyComponent, {MyComponent2} from "./MyComponent";

Q: What is the importance of config.js file?

A: config.js files are essentially editable text files that contain information required for the successful operation of a program. These files are structured in a particular way, formatted to be user configurable. Most computer programs we use, whether office suites, web browsers, or video games, are configured via menu interfaces. Configuration files are very simple in structure. For example, if an application only needs to know the user's preferred name, then its one and only config file could contain exactly one word: the name of the user.

Example:

- Single piece of information:
    - Chetan
- Key and value structure:
    - NAME='Chetan'
    - SURNAME='Nada'

Q: What are React Hooks?

A: In React version 16.8, React introduced a new pattern called Hooks. React Hooks are simple JavaScript functions that we can use to isolate the reusable part from a functional component. Hooks can be stateful and can manage side-effects. They allow you to reuse stateful logic without changing your component hierarchy, making it easy to share Hooks among many components or with the community.

React provides a bunch of standard in-built hooks:

- **useState:** To manage states. Returns a stateful value and an updater function to update it.
- **useEffect:** To manage side-effects like API calls, subscriptions, timers, mutations, and more.
- **useContext:** To return the current value for a context.
- **useReducer:** A useState alternative to help with complex state management.
- **useCallback:** It returns a memorized version of a callback to help a child component not re-render unnecessarily.
- **useMemo:** It returns a memoized value that helps in performance optimizations.

- **useRef:** It returns a ref object with a current property. The ref object is mutable. It is mainly used to access a child component imperatively.
- **useLayoutEffect:** It fires at the end of all DOM mutations. It's best to use useEffect as much as possible over this one as the useLayoutEffect fires synchronously.
- **useDebugValue:** Helps to display a label in React DevTools for custom hooks.

Q: Why do we need useState Hook?

A: The useState hook is used to maintain the state in our React application. It keeps track of the state changes, encapsulating local state in a functional component. The useState hook is a special function that takes the initial state as an argument and returns an array of two entries. useState encapsulates only a singular value from the state; for multiple states, we need to have multiple useState calls.

Syntax for useState hook:

- const [state, setState] = useState(initialstate);
- Importing: To use useState you need to import it from react as shown below:
    - import React, { useState } from "react";

Usage in Functional Components:

const Example = (props) => {

- // You can use Hooks here!
- return <div />;

}

Topic : Microservices and Monolith Architecture

Q: What is `Microservice`?

A: Microservice, also known as the microservice architecture, is an architectural and organizational approach to software development where software is composed of small independent services like database, server, or a UI of the application, that communicate over well-defined APIs. These services are owned by small, self-contained teams. Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features. This approach means dividing software into small, well-defined modules enables teams to use functions for multiple purposes.

Benefits of Microservices:

- Flexible Scaling
- Easy Deployment
- Technological Freedom
- Reusable Code
- Resilience

Monolith Architecture

Q: What is `Monolith architecture`?

A: A Monolith architecture is a traditional model of a software program, built as a unified unit that is self-contained and independent from other applications. A monolithic architecture is a singular, large computing network with one code base that couples all business concerns together. To make a change to this application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming. This approach means not dividing software into small, well-defined modules, using every service like a database, server, or UI of the application in one application file.

Difference Between Monolith and Microservice

Q: What is the difference between `Monolith and Microservice`?

A: With monolithic architectures, all processes are tightly coupled and run as a single service. If one process of the application experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features becomes more complex as the code base grows, limiting experimentation and making it difficult to implement new ideas. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities, and each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

UseEffect Hook

Q: Why do we need a `useEffect Hook`?

A: The `useEffect Hook` is a JavaScript function provided by React. The useEffect Hook allows you to eliminate side effects in your components. Examples of side effects are fetching API data, directly updating the DOM, and setting up subscriptions or timers, which can lead to unwarranted side effects. `useEffect` accepts two arguments: a callback function and a dependency array. The second argument is optional.

The callback function runs when the component mounts, and the dependency array determines when the effect should re-run.

If we do not pass an empty dependency array, then the `useEffect` runs every time the UI is rendered.

Optional Chaining

Q: What is `Optional Chaining`?

A: The `Optional Chaining` (?.) operator accesses an object's property or calls a function. If the object accessed or function called is `undefined or null`, it returns `undefined` instead of throwing an error. `Optional Chaining` (?.) is a good way of accessing object keys, preventing the application from crashing if the key being accessed is not present. If the key is not present, it returns `undefined` instead of throwing a key error.

Shimmer UI

Q: What is `Shimmer UI`?

A: A `Shimmer UI` resembles the page's actual UI, so users will understand how quickly the web or mobile app will load even before the content has shown up. It gives people an idea of what's about to come and what's happening (while UI is currently loading) when a page full of content/data takes more than 3-5 seconds to load. Shimmer UI is a great way for loading applications. Instead of showing a loading circle, we can design a shimmer UI for our application that improves user experience.

JS Expression vs. JS Statement

Q: What is the difference between `JS expression and JS statement`?

A: A `JS expression` returns a value that we use in the application. For example:

1 + 2 // expresses

"foo".toUpperCase() // expresses 'FOO'

console.log(2) // logs '2'

isTrue ? true : false // returns true or false based on isTrue value

A `JS statement` does not return a value. For example:

let x; // variable declaration

if () { } // if condition

If we want to use `JS expression` in JSX, we have to wrap it in `{}`. If we want to use `JS statement` in JSX, we have to wrap it in `{()}`.

Conditional Rendering

Q: What is `Conditional Rendering`? Explain with a code example.

A: `Conditional rendering` in React works the same way conditions work in JavaScript. Use JavaScript operators like `if` or the `conditional operator` to create elements representing the current state, and let React update the UI to match them. For example:

Using Ternary operator as a shorthand way of writing an if-else statement:

{isLoggedIn ? (return <UserGreeting />) : (return <GuestGreeting />)}

Using an if…else Statement:

if (isLoggedIn) { return <UserGreeting />; } else { return <GuestGreeting />; }

Using Logical &&:

{isLoggedIn && <button>Logout</button>}

CORS

Q: What is `CORS`?

A: Cross-Origin Resource Sharing (CORS) is an HTTP-header-based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request.

Async and Await

Q: What is `async and await`?

A: `Async`: It allows us to write promises-based code as if it was synchronous and ensures we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a promise. It makes sure that a promise is returned, and if it is not returned, then JavaScript automatically wraps it in a promise resolved with its value.

`Await`: Await function is used to wait for the promise. It can be used within the `async` block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

Use of `const json = await data.json();` in `getRestaurants()`

Q: What is the use of `const json = await data.json();` in `getRestaurants()`?

A: The `data` object, returned by the `await fetch()`, is a generic placeholder for multiple data formats. We can extract the `JSON object` from a `fetch` response by using `await data.json()`. `data.json()` is a method on the data object that lets you extract a `JSON object` from the data or response. The method returns a promise because we have used the `await` keyword. So `data.json()` returns a promise resolved to a `JSON object`.

Topic: React Image Handling

Q: What are various ways to add images into our app?

A: There are several methods to add images in a React application:

Using the full URL of the image:

```
<img src="https://reactjs.org/logo-og.png" alt="React Image" />
```

Importing images into the project: Place your image in the `src` directory or a dedicated `images` folder and import it:

```
import reactLogo from "./reactLogo.png";

export default function App() {
  return <img src={reactLogo} alt="React Logo" />;
}
```

Ensure the path is correct based on your project structure.

Organizing images in an `assets` folder: If you have multiple types of assets, including images, organize them within an `assets` folder:

```
import reactLogo from "../../assets/images/reactLogo.png";

export default function App() {
  return <img src={reactLogo} alt="React Logo" />;
}
```

Q: What would happen if we do `console.log(useState())`?

A: Calling `console.log(useState())` will output an array `[undefined, function]`. The first element (`undefined`) represents the initial state value, and the second element is a function (`setState`) used to update the state.

Q: How will `useEffect` behave if we don't add a dependency array?

A: If you omit the dependency array (`[]`) in `useEffect`, the callback function inside `useEffect` will execute every time the component re-renders. This is because React will run the effect after every render, regardless of whether the dependencies have changed.

Q: What is SPA?

A: Single Page Application (SPA) is a type of web application that dynamically updates the webpage with data from the server without reloading the entire page. SPAs load all required HTML, CSS, and JavaScript during the initial load and fetch additional data as needed.

Q: What is the difference between Client Side Routing and Server Side Routing?

A: Client-side routing and **server-side routing** differ in how they handle navigation and page rendering:

- **Server-side routing (SSR)**: Each URL change typically triggers a server request for a new webpage, causing a full page reload. The server renders and sends back the entire HTML for the new page.
- **Client-side routing (CSR)**: In this approach, the initial webpage is loaded from the server to the client. Subsequent URL changes are handled by JavaScript within the browser, which updates the page content without requiring a full server round-trip.

Client-side routing is commonly used in **Single Page Applications (SPAs)** to provide smoother and faster navigation without full page reloads.

Topic: React Router Configuration and Lifecycle Methods in React Components

Q: How do you create Nested Routes in react-router-dom configuration?

A: Nested routes can be configured in react-router-dom as follows:

To create nested routes, you can define a hierarchy of routes using the `children` property within route configurations. Here's an example:

```
const router = createBrowserRouter([
   {
      path: "/",
      element: <Parent />,
      children: [
         {
            path: "child",
            element: <Child />,
            children: [
               {
                  path: "subchild",
                  element: <SubChild />,
               }
            ],
         }
      ],
   }
])
```

This sets up nested routes where `/child` is nested within `/`, and `/subchild` is nested within `/child`.

Q: Read about `createHashRouter`, `createMemoryRouter` from React Router docs.

A:

- **`createHashRouter`**: This router uses the hash portion (`#`) of the URL, making it useful when configuring servers that can't direct all traffic to the React Router application. It's functionally similar to `createBrowserRouter`.
- **`createMemoryRouter`**: Unlike using the browser's history, this router manages its own history stack in memory. It's ideal for testing or non-browser environments like Storybook.

For more details, refer to the React Router documentation: HashRouter, MemoryRouter.

Q: What is the order of lifecycle method calls in Class Based Components?

A: The order of lifecycle method calls in Class Based Components is:

1. `constructor()`
2. `render()`
3. `componentDidMount()`
4. `componentDidUpdate()`
5. `componentWillUnmount()`

For a visual diagram, you can refer to the React Lifecycle methods diagram: React Lifecycle Methods Diagram.

Q: Why do we use `componentDidMount`?

A: `componentDidMount()` is used to execute React code after the component has been rendered to the DOM. It is part of the Mounting phase of the React lifecycle. Common use cases include making API calls or initializing subscriptions.

Q: Why do we use `componentWillUnmount`? Show with an example.

A: `componentWillUnmount()` is used for cleanup tasks when a component is about to be removed from the DOM. This is crucial in Single Page Applications to prevent memory leaks and performance issues. Here's an example:

```
class Repo extends React.Component {
   componentDidMount() {
      this.timer = setInterval(() => {
         console.log("Interval running...");
      }, 1000);
   }

   componentWillUnmount() {
      clearInterval(this.timer);
      console.log("Interval cleared.");
   }

   render() {
      return <div>Repo Component</div>;
   }
}
```

In this example, `componentDidMount()` sets an interval to log messages every second. `componentWillUnmount()` clears that interval when the component is unmounted, preventing it from continuing to run in the background.

Q: (Research) Why do we use `super(props)` in constructor?

A: `super(props)` is used in a constructor of a child class component to call the constructor of the parent class. It allows the child component to access and initialize properties inherited from its parent, including `this.props`.

Without `super(props)`, you may encounter a ReferenceError stating that `this` must be called before accessing `this` in the constructor of a derived class. Using `super(props)` ensures that `this.props` is correctly initialized in the child component's constructor.

Q: (Research) Why can't we have the callback function of `useEffect` async?

A: The `useEffect` hook in React expects its callback function to return either nothing or a cleanup function. If you make the callback function `async`, it will return a Promise implicitly. React does not handle Promises returned from `useEffect` callbacks, which can lead to unexpected behavior, such as ignoring cleanup effects or causing memory leaks.

To use asynchronous operations inside `useEffect`, you can define an inner function and call it asynchronously:

```
useEffect(() => {
   const fetchData = async () => {
      // Async operation here
   };

   fetchData();

   return () => {
      // Cleanup function
   };
}, []);
```

This pattern ensures that asynchronous operations are handled correctly within the `useEffect` hook.

Topic : Optimizing React Applications with Lazy Loading and Suspense

Q: When and why do we need `lazy()`?

A: `React.lazy()` or lazy loading is used to dynamically import components or parts of code that should only be loaded when they are required.

Lazy loading offers several benefits:

- **Improved initial load time:** It reduces the amount of code that needs to be downloaded and parsed initially, thus speeding up the initial load time of your application.
- **Reduced memory usage:** By deferring the loading of resources until they are needed, lazy loading can also reduce the memory footprint of your application.
- **Improved user experience:** It can make your application feel more responsive by loading components or resources on-demand, as needed.

When to use lazy loading: Lazy loading is ideal for scenarios where certain components or resources are not immediately needed when the application loads, but rather when certain conditions are met (such as user interaction or scrolling to a specific section of a page).

Q: What is suspense?

A: `Suspense` is a feature introduced in React 16.6 that allows developers to display a fallback UI while waiting for data to load asynchronously. It's particularly useful for improving the perceived performance of your application, ensuring users don't see a blank screen during data fetching operations.

To use Suspense, you wrap components that fetch data with a `<Suspense>` component. This component renders a fallback UI until the data is ready, then displays the actual content.

For example:

jsx
Copy code
```
<Suspense fallback={<div>Loading...</div>}>
  <MyComponent />
</Suspense>
```

Suspense can also be used with `React.lazy()` to lazily load components, further optimizing your application's performance by deferring component loading until necessary.

Q: Why did we get this error: "A component was suspended while rendering a synchronous input"? How does suspense fix this error?

A: This error occurs when a component attempts to render synchronously while it's in a suspended state, waiting for an asynchronous operation (like dynamic component loading) to complete. React expects a `<Suspense>` boundary to handle the suspended state and display a fallback UI until the component can render again.

To fix this error, you should ensure that components that may suspend while rendering (due to asynchronous operations) are wrapped in a `<Suspense>` component. This ensures that React knows how to manage the loading state and display a suitable fallback until the data or component is ready.

Additionally, the `startTransition` API can be used to improve the user experience by showing the old UI while the new UI is being prepared, without removing the `<Suspense>` component or its props.

Q: What are the advantages and disadvantages of using the code splitting pattern?

A: `Code splitting` is a technique that breaks down an application's JavaScript bundle into smaller chunks, which can be loaded dynamically as needed.

Advantages:

- **Faster initial load time:** Only necessary code is loaded initially, speeding up the initial page load.
- **Improved user experience:** Users can interact with the application sooner, with non-essential code loaded asynchronously in the background.
- **Improved performance:** Less JavaScript needs to be parsed and executed initially, enhancing overall performance.

Disadvantages:

- **Increased complexity:** Development and testing processes can become more complex due to managing multiple code bundles.
- **Network requests:** More network requests may be required, potentially affecting performance on slower connections.
- **Bundle size:** Additional code and dependencies can increase the overall bundle size, although this is usually outweighed by the benefits in performance.

Q: When and why do we need `suspense`?

A: `Suspense` in React is crucial when you want to enhance user experience by displaying a temporary UI (fallback) while waiting for asynchronous operations like data fetching from APIs, especially after the initial page load.

It's particularly useful in conjunction with `lazy loading` (using `React.lazy()`), where components are loaded dynamically when needed, rather than upfront during the initial page load. This combination helps in optimizing performance and responsiveness of your application.

Topic : Redux vs useContex

1. Explain the difference between useContext and Redux.

- **Context API:**
  - **Purpose:** Provides a way to share values between components throughout the application without having to pass props manually at every level.
  - **Integration:** Built into React, no separate installation required.
  - **Setup:** Minimal setup required.
  - **Suitability:** Ideal for static data that doesn't change frequently.
  - **Debugging:** More difficult compared to Redux.
- **Redux:**
  - **Purpose:** Manages application state in a centralized store.
  - **Integration:** Third-party library, not part of React.
  - **Setup:** Requires more setup to integrate with a React application.
  - **Suitability:** Suitable for both static and dynamic data.
  - **Debugging:** Easy with Redux DevTools.

2. What are the advantages of using Redux Toolkit over Redux?

- **Abstraction and Convenience:** Provides abstractions like `createSlice` for easier state management.
- **Immutable Updates:** Simplifies immutable state updates, reducing boilerplate code.
- **Simplified Reducers:** Automatically generates reducers with `createSlice`, reducing complexity.
- **Improved Performance:** Optimizes performance with built-in optimizations.
- **Better Debugging:** Enhanced debugging capabilities with tools like Redux DevTools.

3. Explain Dispatcher.

- **Dispatcher:** A function used to dispatch actions to the Redux store.
  - **Creation and Usage:**
    - Use `useDispatch()` hook in React to obtain the dispatcher function.
    - Dispatch actions like `dispatch(actionCreator(data))` to trigger state updates.
    - The action creator returns an action object that the reducer uses to update the state.

4. Explain Reducer.

- **Reducer:** A pure function in Redux that takes the current state and an action, and returns a new state based on the action.

Example:
```
addItem: (state, action) => {
  // Update state based on action payload
}
```

  -
  - Reducers are used to manage how the state changes in response to actions dispatched to the store.

5. Explain Slice.

- **Slice:** In Redux Toolkit, a slice is a portion of the Redux state with its own reducer and actions.
  - Created using `createSlice`, it encapsulates related state and logic within a single module.

6. Explain Selector.

- **Selector:** A function that takes the Redux state and returns derived data.
  - Used with `useSelector()` hook in React to subscribe to changes in specific parts of the Redux state.

Example:
```
const totalItemsCount = useSelector(state => state.cart.totalItemsCount);
```

  -
  - Selectors help in efficiently extracting and using data from the Redux store in React components.

7. Explain createSlice and its configuration.

- **createSlice:** A function in Redux Toolkit used to create a slice of state with reducers and actions.
  - **Configuration:**
    - **name:** Name of the slice.
    - **initialState:** Initial state of the slice.
    - **reducers:** Object containing reducer functions for state updates.

Example:

```
const cartSlice = createSlice({
  name: 'cart',
  initialState: { items: {}, totalItemsCount: 0 },
  reducers: {
    addItem: (state, action) => {
      // Logic to add an item to the cart
    },
    removeItem: (state, action) => {
      // Logic to remove an item from the cart
    },
    clearCart: (state) => {
      state.items = {};
      state.totalItemsCount = 0;
    },
  },
});
```

- `createSlice` simplifies the process of defining reducers and managing state updates in Redux.

These explanations should provide a clear understanding of the concepts related to Redux and React state management.

Topic :JavaScript Testing: Tools and Strategies

1. What are different types of testing?

- **Unit Testing:** Focuses on individual units or components of the software to ensure they work as intended.
- **Integration Testing:** Combines different units and tests their interaction to ensure they work together as a system.
- **Functional Testing:** Tests the functionality of the software to verify it meets the requirements and specifications.
- **End-to-end Testing:** Tests the entire system, simulating real-world scenarios from start to finish.
- **System Testing:** Tests the system as a whole to ensure it meets required performance, security, and reliability standards.
- **Acceptance Testing:** Tests the software from the user's perspective to ensure it meets customer expectations.
- **Performance Testing:** Tests the performance aspects like response time and scalability.
- **Security Testing:** Tests the security of the software against potential threats and vulnerabilities.
- **Regression Testing:** Tests the software after changes to ensure new bugs aren't introduced.
- **Smoke Testing:** Preliminary testing to check basic functions before comprehensive testing.

2. What is Enzyme?

Enzyme is a JavaScript testing utility specifically designed for **React** applications. It is maintained by Airbnb and supports both unit and integration testing of React components.

3. Enzyme vs React Testing Library

Features	Enzyme	React Testing Library
API	More comprehensive API for manipulating and querying React	Simplified API focusing on testing component behavior from a user's

	component tree, suited for unit testing.	perspective, suited for integration and end-to-end testing.
Approach	Tests internal implementation details like state and props.	Tests component behavior as a user would interact with it.
Maintenance	Requires more maintenance due to tight coupling with component implementation details.	Less likely to break with changes to component implementation, as it tests behavior.

4. What is Jest and why do we use it?

Jest is a JavaScript testing framework developed by Facebook, widely used for testing applications, especially React. It offers features like automatic test discovery, mocking, code coverage, and assertion libraries. Developers use Jest for its simplicity, speed, integration capabilities with popular tools, and comprehensive testing solution it provides.