**Episode 4: Let’s get Hooked**

**Theory:**

1. Can we use default export along with named export?

Yes, we can do the default export along with the named export. Basically, we can export one thing using export default Component, while also exporting multiple additional values using export.

This flexibility allows us to organize and share various parts of our code efficiently, making it easy for other parts of the application to access and use the exported functionalities.

2. What is the difference between Named Export, Default Export and \* as export?

ES6 provides us to import and export a module and use it in other files. ES6 provides 2 ways to export a module from a file i.e. named export and default export. In Named export, we can export multiple named exports from a single file. Then import that specific named export they want surrounded in {} braces. The name of the imported module has to be the same as the exported module.

In Named export, the component is exported as below:

export const Component1 = () => {}

export const Component2 = () => {}

and it is imported from Components.js file as below:

import {Component1} from “./Component”;

import {Component2} from “./Component”;

In Default export, we can have only one default export per file. The naming of the import is completely independent in default import and we can use any name we like. In default export, the component is exported as below:

const Component = () => {}

export default Component;

and the component is imported as below:

import Component from “./Component”

In \* as import, it is basically used to import the whole module as a component and access the components inside the module. In \* as export, the component is exported as below:

export const Component1 = () => {}

export const Component2 = () => {}

export const Component3 = () => {}

and the component is imported as below:

import \* as MainComponents from “./Component”;

We can now use them in JSX as below:

<MainComponents.Component1 />

<MainComponents.Component2 />

<MainComponents.Component3 />

We can also use Named export and Default export together. We can export like:

export const Component = () => {}

const Component2 = () => {}

Default export Component2

And import it as below:

import {Component}, Component2 from “./Component”;

3. What is the importance of config.js file?

Config.js allows developers to configure their applications in an XML block instead of hard-coding values inside their scripts or in JSON objects.

These files are essentially editable text files that contain information required for the successful operation of a program. The files are structured in a particular way, formatted to be user configurable. Most of computer programs that we use: whether office suites, web browsers, video games etc are configured via menu interfaces. Configuration files are very simple in structure.

For example: If an application needs to store a user’s preferred name, then the config file might look like this:

Most applications require multiple settings and configurations, so the config files contain key-value pairs for various options:

NAME = “DHRUVIL”

SURNAME = “SONI”

4. What are React Hooks?

Hooks were added in React in version 16.8. Hooks basically allow functional components to have access to state and other React features.

React Hooks are simple Javascript functions that we can use to isolate the resuable part from a functional components. Hooks can be stateful and can manage side effects. Hooks allows us to reuse the stateful logic without changing our component hierarchy. This makes it easy to share Hooks among many components or with the community.

React provides a bunch of standard in-built hooks:

1. useState – To manage states. It returns a stateful value and an updater function to update it.

2. useEffect – To manage side effects like API calls, subscriptions, timers, mutations and more.

3. useContext – To return the current value for the context.

4. useReducer – A useState alternative to help with complex state management

5. useCallback – It returns a memorized version of a callback to help a child component not re-render unnecessarily.

6. useMemo – It returns a memorized value that helps in performance optimization.

7. 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.

8. useLayoutEffect – It fires at the end of all DOM mutations. It is best to use the useEffect as much as possible over this one as the useLayoutEffect fires synchronously.

9. useDebugValues – Helps to display a label in React DevTools for custom hooks.

5. Why do need a useState Hook?

useState Hook is used to manage the state of our React component. It keeps track of the state changes so basically useState has the ability to encapsulate the local state in a functional component. It is a special function that takes the initial state as an argument and returns an array of two entries. The first entry is the current state and the second entry is a function to update the entry. It will return us the updates state value whenever the setter function is called.

We can use useState hook as below:

const [state, setState] = useState(initialState);

To use useState in our React application, we will have to import useState as shown below:

Import React, {useState} from “react”;

Basically, here we are using the named import to import the useState Hook.

We can use Hooks in Functions Components as below:

const Example = () => {

//We can use Hooks here

return <div>Hello</div>;

}

**Coding:**

1. Clean up your code

2. Create a Folder Structure for your app

3. Make different files for each Components

4. Create a config file

5. Use all types of import and export

6. Create a Search Box in your app

7. Use useState to create a variable and bind it to the input box

8. Try to make your search bar work

**References:**

1. Code Link - <https://bitbucket.org/namastedev/namaste-react-live/src/master/>