**Week 7**

**7.1 Context API & Prop Drilling**

* Routing
* React lets you create single page applications. Dynamically changing websites.
* Client-side bundle 🡪 the bundle code that the gets back from the server.
* Client-side routing 🡪 if the client goes to messages, show them different view etc.
* Always use useNavigate inside a component which is inside <BrowserRouter></BrowserRouter>.
* Lazy loading allows to give the client the code bundle in parts and not all at once. Recommended and optional.
* Suspense API helps to load fallback when the landing page has not yet come.
* Prop Drilling
* You should push the state down as much as you can. If the state is being used by different children components then you can store it in LCA.
* Prop drilling does not stop extra re-rendering but helps to make code visually better.
* Context API
* It helps you teleport variable without drilling them down.
* It also lets you keep all state logic outside of your core react component.
* Context API is for making the code cleaner while state management tools are for performance.
* We need to create context to teleport that variable for that context. createContext() does that.
* Wrap anyone that wants to use the teleported variable inside the provider.
* Read about reducer and useReducer.

**7.2 Context API & Recoil**

* Even when using Context API, the component that does not use the count variable re-renders.
* We use context API to make syntax cleaner and get rid of prop drilling.
* Recoil also helps us to stop extra re-rendering.
* It is good to have component and state separately. Recoil helps you do that.
* Atom is similar to useState. It stores the state.
* Recoil has 3 things similar to useState(). useRecoilState is exactly similar, useRecoilValue when we only need the value, useSetRecoilValue when we only need to set the value.
* In our App, wherever we want to use any recoil logic or atom logic that needs to be wrapped inside <RecoilRoot></RecoilRoot>. Usually we do this at the top in App().
* For global variables, we use recoil instead of useState.
* Whenever we have a derived state, we use useMemo so that it only re-renders when count changes. Similarly, recoil provides you selector, when we know something completely depends on another state variable. Depending is key here.
* Atoms and selectors are good enough to know. Good to know about atom family and selector family.
* Selectors are always derived from atoms.

**7.3 Recoil Deep Dive**

* Atoms is just another way of defining state. However, it helps in state management.
* A selector is derived from atoms.
* The default value of an atom needs to be synchronous or it can be a selector which can be asynchronous.
* When we do backend requests in the selector in an atom, it takes some time to get the data back and in that meantime the screen remains white.
* atomFamily returns the atom that the component requests specific to an input.
* atomFamily lets you create multiple atoms dynamically.
* atomFamily also stops unnecessary renders because it provides separate atoms. If there is a list and an element changes then every component that uses that list would rerender.
* For asynchronous calls, you would use a selector that has async get for atom. Similarly, you would use selectorFamily for atomFamily.
* The only difference is that get has a function(id) which returns an async({get}) which returns the atom from the database.
* If you want to show loading while we get the data from the database, we simply add Loadable after useRecoilState like useRecoilStateLoadable.
* If you don’t use useRecoilValueLoadable , you can use Suspense. This would only show one loading. But if we get error, then it would crash. We would also have to use ErrorBoundary.