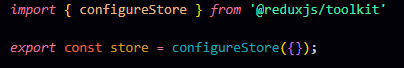
**Redux**

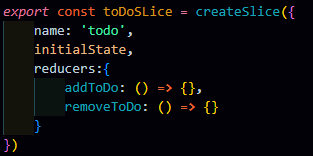
1. Redux is state management library. It is commonly used with React JS to manage and centralize the application state in a single source of truth called **store.**
   1. **Single source of truth:** The entire state of application is stored in a single **JS Object**.
   2. It is used when we have complex state management, frequent state changes or Global state sharing.
2. **How to install redux:**
   1. **npm install @reduxjs/toolkit**
   2. **npm install react-redux**
3. To start working with Redux we have to create a store:
   1. You can create a separate folder for store or you can work on same file.
   2. We will **configure and export the store**



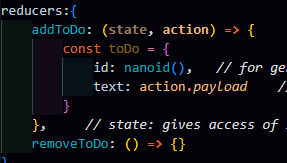
* 1. While **exporting the store** we have to pass the **object inside the configure store.**
  2. After configuring the store we have to **make reducers.**
     1. First we will make a **features named folder for this,** inside this we will create another **to-do folder** in which we will create a file for **to-do reducer. (We made a to –do folder because, we will work with to-do,** you can make any reducers for anything like: login, signup, etc.)
     2. Now, we will create an **initial state variable** in which takes the **object,** inside this object we will **add to-do (which will load on loading the page)**.



* + 1. Now, we will **make and export the reducer** using **create slice** which takes **object** and we will pass some **properties** like: name, and some methods like **add to-do, remove to-do and we will also pass our initial state.**
       1. **Add to-do & remove to-do:** This method can be made outside the to-do slice or inside.



* + 1. Functions inside the reducers have **parameters like state & action.**
       1. **State:** Whenever state changes, it will update the UI, initially we have 1 to-do, might be we will have more to-dos.
       2. **Action:** It helps in providing the values for further actions like removing the to-dos, etc.
    2. Inside our **add-to-do method:** We will create a to-do named variable in which we have **object of id & text,** id will be generated unique by using **nanoid**.



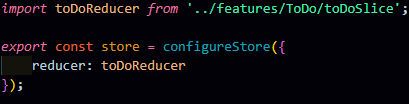
* + 1. Till now we have only made the to-do, we **haven’t updated the state**. **To update the state** we will **push** these to-dos (inside the add-to-do) in the to-dos (inside the initial state).
    2. Inside our **remove-to-do method:** We will use filter method to filter out those to-dos whose ids are not matched.
    3. After this, we will **export our all methods.**



* + 1. Since, our **store** is not **aware about our reducer,** so we have to **export our reducers also.**



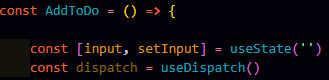
* + 1. In our **store,** we will **import our reducer** and we will add our **exported reducer** inside our store.



1. Now will create our components and learn **how to get data** and **how to send data.**
   1. Created a component folder in which we have 2 files: Add-to-do (our form to add all to-do) and To-do.
   2. **For sending data we will use a dispatch from react.** **Dispatch use reducers to update the store**.
      1. For using the dispatch we have import it.

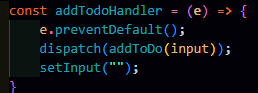


* + 1. To use dispatch:



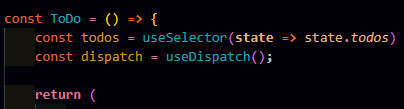
* + 1. For sending data we will **1st import our reducers (add-to-do, remove-to-do) then we will call these reducers inside our dispatch method.**
       1. add-to-to-handler is a function use to handle the to-do form.
       2. The input entered in the add-to-do inside the dispatch is the state form use-state.





* + 1. For adding to-do we will **import use-selector and use-dispatch** and also remove-to-do.
       1. For adding to-dos we will **use-selector**





* + - 1. **Now our work is completed.**

1. **Now we will add our components wherever we need to use.**
   1. To use our component we have to use **provider** & also need to **import our store** in our **main-jsx file** and **wrap our App component with provider** and we will also pass the properties.
      1. Importing provider



* + 1. Importing our store



* + 1. Wrapping our **App component**

