|  |  |  |
| --- | --- | --- |
|  | **COUNTER APPLICATION** | Simple counter application – to increase decrese reset to zero . |
| 1 | **Create react app** | npx create-react-app $applicationName |
| 2 | **Add redux support**  for state management | npm install redux –save |
| 3 | **Determine Application Subjects & Model** | Counter is |
|  | The state of store is changed by Actions | Actions are objects, which have at least a field determining the *type* of the action. |
| 4 | **Determine actions:**  (INCREMENT , DECREMENT, ZERO ) | {  type: 'INCREMENT'  } |
| 5 | **Define reducer**  Let's also define a [default value](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters) of 0 for the parameter *state*  Reducer is never supposed to be called directly from the applications code. Reducer is only given as a parameter to the createStore-function which creates the store | const counterReducer = (state = 0, action) => {  switch (action.type) {  case 'INCREMENT':  return state + 1  case 'DECREMENT':  return state - 1  case 'ZERO':  return 0  default: // if none of the above matches, code comes here  return state  }  } |
| 6 | **Create store**  Reducer is never supposed to be called directly from the applications code. Reducer is only given as a parameter to the createStore-function which creates the store | import { createStore } from 'redux'  const counterReducer = (state = 0, action) => {  // ...  }  const store = createStore(counterReducer) |
|  | The store now uses the reducer to handle *actions*, which are *dispatched* or 'sent' to the store with its [dispatch](https://redux.js.org/api/store#dispatchaction)-method. | store.dispatch({type: 'INCREMENT'}) |
|  | You can find out the state of the store using the method [getState](https://redux.js.org/api/store" \l "getstate" \t "_blank). | console.log(store.getState())  store.dispatch({type: 'INCREMENT'})  store.dispatch({type: 'INCREMENT'})  store.dispatch({type: 'INCREMENT'})  console.log(store.getState()) |
| 7 | **Subscribe to store**  When the state in the store is changed, React is not able to automatically rerender the application. Thus we have registered a function renderApp, which renders the whole app, to listen for changes in the store with the store.subscribe method | store.subscribe(() => {  const storeNow = store.getState()  console.log(storeNow)  })  const renderApp = () => {  ReactDOM.render(<App />, document.getElementById('root'))  }  renderApp()// index.js will do this for you.  store.subscribe(renderApp) |
|  |
|  | **NOTES APPLICATION** | Simple note taking application.  A Note with text string, it has an order, its either important or non-important, |
|  | **We need to design an GUI to manage notes** | CRUD operation on note, change order by moving up and down (drag), toggle importance from gui |
| 1 | **Create react app, add redux,router support** | npx create-react-app $applicationName  npm install redux –save  npm i react-router-dom --save |
| 2 | **Structure your code : create folders** | **(screens) -> (App)->(components)->App.js,**  **(screens) -> (shared)->(components,constants,utils)**  **(screens)-> (shared)-> (constants)->routes.js** |
| 3 | **Create routes in (shred->constants-routes.js)** | export default Object.freeze(  { LOGIN:{ name: ‘Login’, route:’/login’} …}); |
| 4 | **Add the route info to the App** | import {    BrowserRouter,    Routes,    Route,    Link  } from "react-router-dom";  import routeConstants from '../../../shared/constants/routes'; |
| 5 | **Add Navigation links to the App** | <BrowserRouter >   <h1>App Component</h1>  <ul className="App-nav-list">          <li className="App-nav-item">            <Link to={LOGIN.route}>{LOGIN.name}</Link>          </li>          <li className="App-nav-item">            <Link to={DASHBOARD.route}>{DASHBOARD.name}</Link>          </li>  </ul>  <Routes>  <Route path={LOGIN.route}>{LOGIN.name}</Route>  <Route path={DASHOARD.route}>{DASHBOARD.name}</Route>  </Routes>  </BrowserRouter > |
|  | **Add Navigation bar component** |  |
|  | **Determine Application Subjects & Model** | Note |
| 4 | **Determine actions:** | NEW\_NOTE,TOGGLE\_IMPORTANCE,DELETE,UPDATE,GET\_ALL,REMOVE\_ALL,GET\_NOTE |
| 5 | **Define Application** | Add the boilerplate code of App.js |
| 6 | **Define reducer**  NoteReducer.js | const noteReducer = (state = [], action) => {  if (action.type === 'NEW\_NOTE') {  state.push(action.data)  return state  }  return state  } |
|  | **NOTE**: A reducer state must be composed of [immutable](https://en.wikipedia.org/wiki/Immutable_object) objects. If there is a change in the state, the old object is not changed, but it is replaced with a new, changed, object | |
| 7 | **Create Store** | import { createStore } from 'redux'  const store = createStore(noteReducer) |
| 8 | **Subscribe to Store** | const renderApp = () => {  ReactDOM.render(<App />, document.getElementById('root'))  }  store.subscribe(renderApp) |
| 9 | Test driven Development : | |
| 10 | React Testing library | npm install –save-dev @testing-library/react |
|  | Add library deep-freeze for immutable finction | Npm install –save-dev deep-freeze |
|  |  |  |
|  |  |  |