devtown

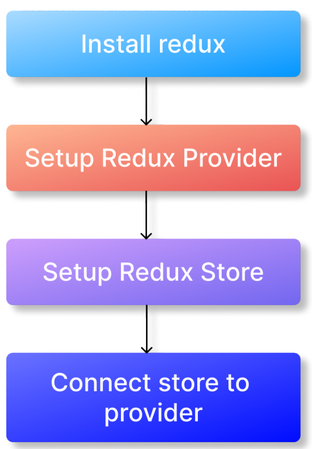
React Redux

CHEAT SHEET 01

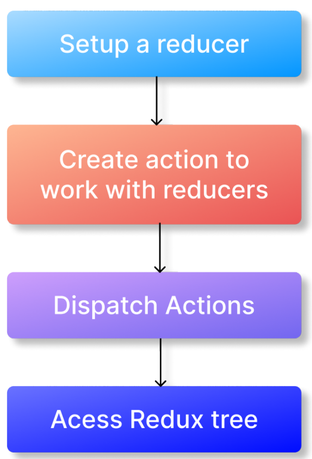
Overview

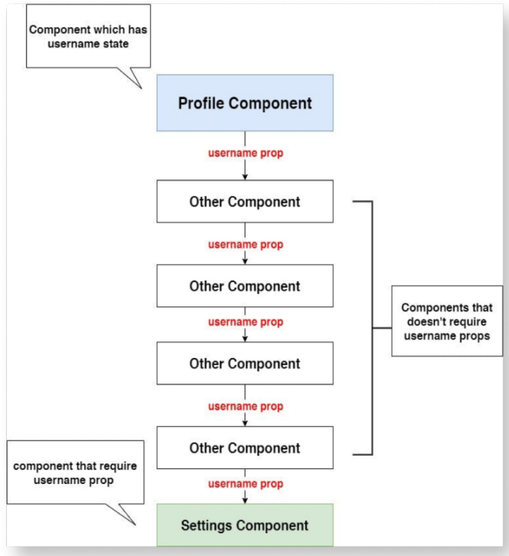
Redux is a library that holds your application state in a place that is globally accessible so that your application components can directly access the data from that global place, instead of you manually passing data throughout all the components.

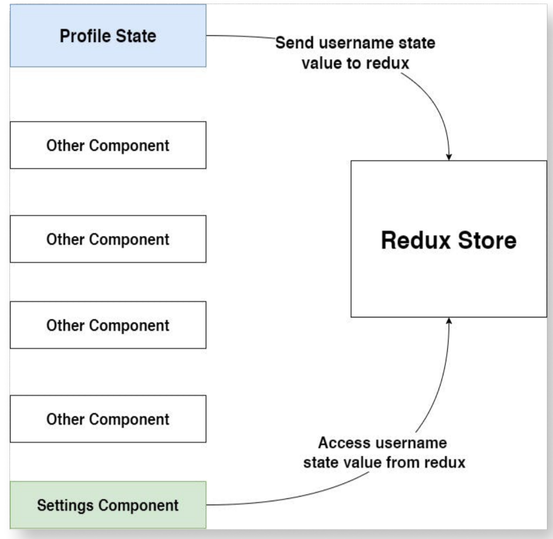
SETUP FLOW



WORK FLOW







Redux Setup

Installing

```
npm i redux react-redux
```

OR

```
yarn add redux react-redux
```

Setup Redux Provider index.js file

```
import { Provider } from "react-redux";
ReactDOM.render(
  <React.StrictMode>
    <Provider>
        <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

Redux provides a wrapper called **Provider**. You must wrap it around your whole application and index.js in the best place to do it.

Store is one of the core feature of redux. This is were our data would be stored and to create it just use the **createStore** function and pass the rootReducer as parameter.

Setup Redux Store redux/store.js

```
import { createStore } from "redux";
import rootReducer from "./reducer/rootReducer";
const store = createStore(rootReducer);
export default store;
```

Connect store to provider index.js file

```
import { Provider } from "react-redux";
// Redux Store
import store from "./Redux/store";
ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
        <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

How after setting up the store pass the store to the Provider in index.js file

devtown

React Redux

CHEAT SHEET 02

Reducers & Actions

A simple reducer

```
const INITIAL_STATE = {
  user: [],
};
const userReducer = (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case "ADD_USER":
      return {
        ...state,
        user: [...state.user, action.payload],
      };
    case "DELETE_USER":
      return {
        ...state,
        user: state.user.filter(({ id }) => id !==
        action.payload),
      };
    default:
      return {
        ...state,
      };
  }
};
```

A typical reducer structure, that updates data based on certain cases.

Now Root reducer

```
// libraries
import { combineReducers } from "redux";
// Reducers
import user from "./user/user.reducer";
// root reducer
const rootReducer = combineReducers({ user });
export default rootReducer;
```

Root reducer really required to bundle all your seperate reducers into one single object to pass on to the store. combineReducers method is used to acheve this.

Redux Action

```
// Redux Action to add new user
export const addUserAction = ({ name, email }) => {
  return {
    type: "ADD_USER",
    payload: { name, email, id: `${Math.random()}` },
  };
};
```

An action is a simple function that would specify what type of action / what type of operation that a reducer should perform. By returning the type property the reducer will perform the action based on that type.

Access redux store values

```
import { useSelector } from "react-redux";
const UserList = () => {
  // Redux state
  const reduxState = useSelector(({ user }) => ({ user}));
return (
    <>
      <div className="d-flex mt-5 flex-wrap">
       <h1>{reduxState.user.name}
      </div>
    </>
  );
}
```

Redux provides us with hooks to simplfy the access. useSelector hooks is used to get the redux state object.

By default useSelector retreives all the state object, so destructuring can be used to filter them, as we did in the example to the left.

Dispatch redux action

```
import { useDispatch } from "react-redux";
// Redux Action
import { addUserAction } from
"../Redux/reducer/user/user.action";
const UserList = () => {
// Initialize dispatch hook from react-redux
const dispatch = useDispatch();  1
const submit = () => {
dispatch(addUserAction(userData));  2
 };
return (
    <>
      <button  id=”submit-btn” onClick={submit}>
         Submit
      </button>
    </>
  );
}
```

Redux provides us with hooks to simplfy the dispatch process. useDispatch hook will return the dispatch function which can be used to dispatch action.