State Management in React with Redux

Student's Name

Department, Institutional Affiliation

Course Number and Name

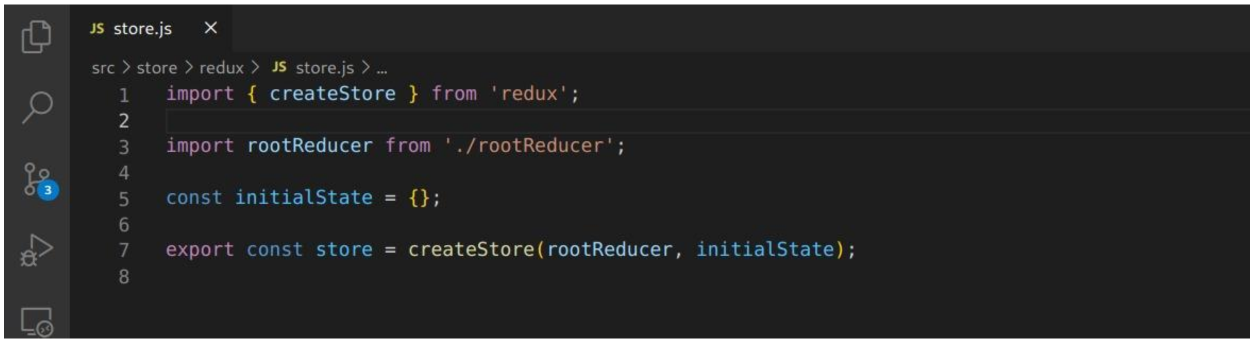
Instructor's Name

Due Date

State Management in React with Redux

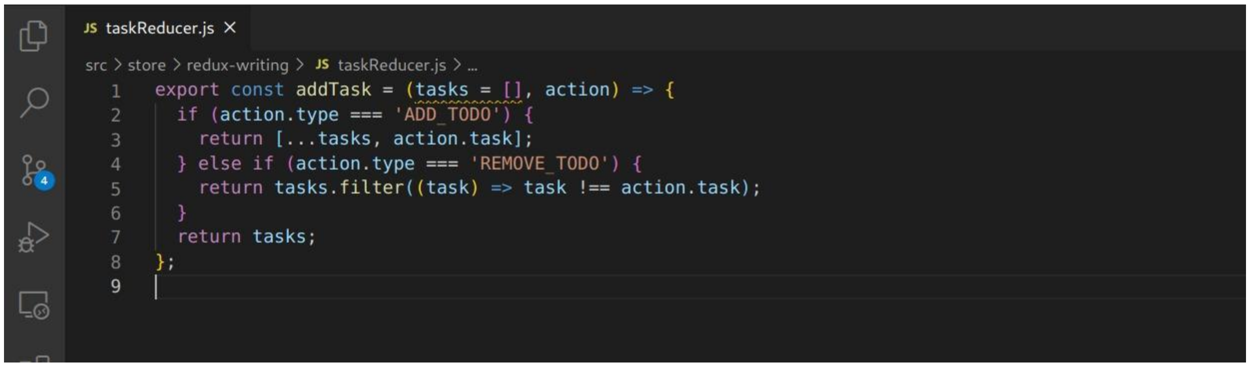
Redux is a state management tool that ensures that data binding floes in one direction. It integrates well with React applications and ensures that there is no prop drilling, all state is stored in one place, and it provides efficient updates for components (Aguirre, 2021). Redux is recommended for state management due to its small implementation and design simplicity. It applies three significant concepts in its work: store, reducers, and actions.

A Redux store is an immutable object that holds the application's state. The state can only be changed by dispatching actions. Redux supports a single store with only one root reducer (Aguirre, 2021). To add the store to a React application, we first install Redux with the command *npm install redux.* Then add the store using the *createStore()* method. The code below shows how it is implemented.

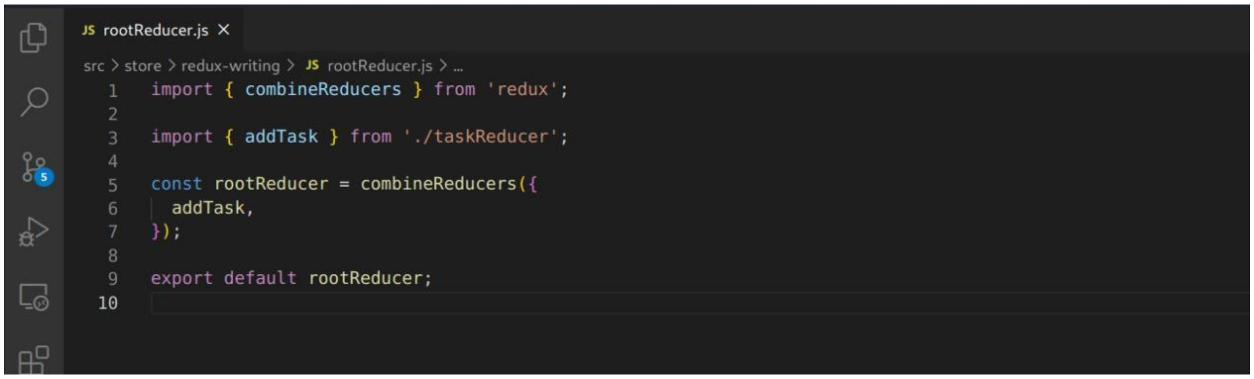


```
import { createStore } from 'redux';

import rootReducer from './rootReducer';

const initialState = {};

export const store = createStore(rootReducer, initialState);

```

Redux reducers determine changes in applications' state. They take in actions and return the App's state (*Redux Fundamentals,* n.d.). Ideally, any Redux app comprises one reducer function referred to as the root reducer. This function is passed to the *createStore()* method to handle all dispatched actions. Besides, the root reducer updates the application's entire state after any changes. Redux reducers do not mutate the state directly. Instead, they copy the existing state and update the copied values' changes (*Redux Fundamentals,* n.d.). This is shown below.

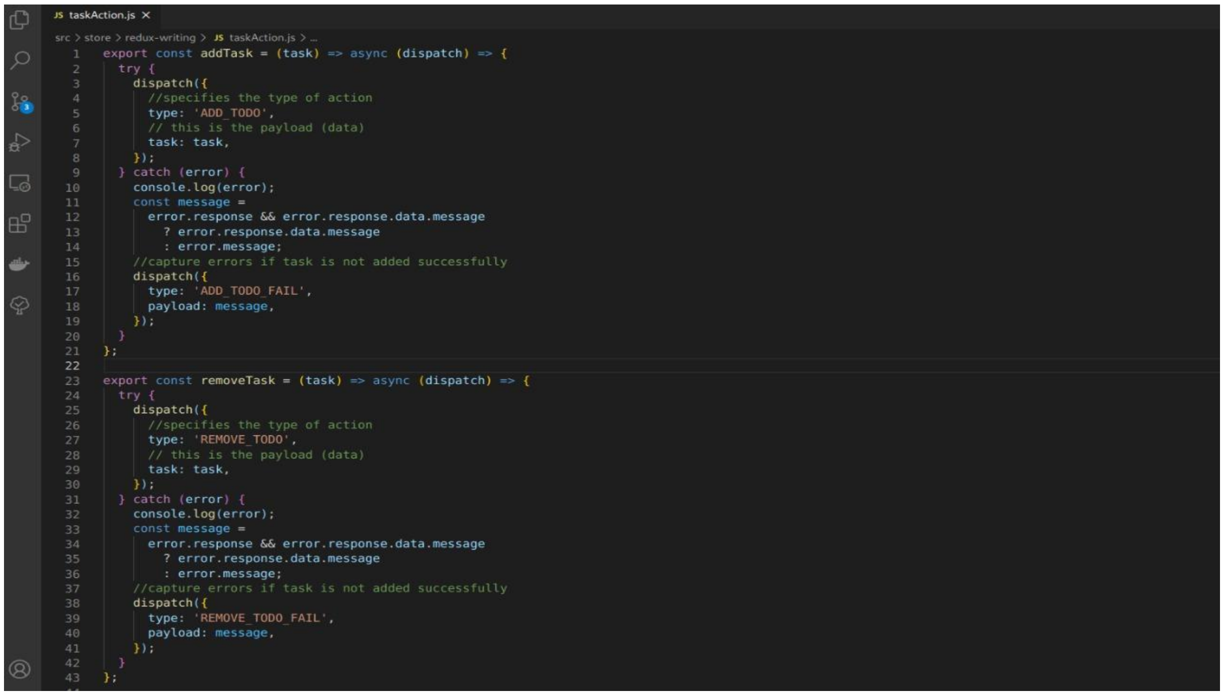


```
export const addTask = (tasks = [], action) => {
  if (action.type === 'ADD_TODO') {
    return [...tasks, action.task];
  } else if (action.type === 'REMOVE_TODO') {
    return tasks.filter((task) => task !== action.task);
  }
  return tasks;
};
```



```
import { combineReducers } from 'redux';

import { addTask } from './taskReducer';

const rootReducer = combineReducers({
  addTask,
});

export default rootReducer;
```

Redux's actions provide information for the store. They have two types of fields: the type field and data fields. The type field communicates the kind of action to perform. Action creators are functions that create actions (*Redux Fundamentals,* n.d.). The code below shows two action creators: adding a task and removing the task. The type communicates the purpose of the action to the creators.



```javascript
export const addTask = (task) => async (dispatch) => {
  try {
    dispatch({
      //specifies the type of action
      type: 'ADD_TODO',
      // this is the payload (data)
      task: task,
    });
  } catch (error) {
    console.log(error);
    const message =
      error.response && error.response.data.message
        ? error.response.data.message
        : error.message;
    //capture errors if task is not added successfully
    dispatch({
      type: 'ADD_TODO_FAIL',
      payload: message,
    });
  }
};

export const removeTask = (task) => async (dispatch) => {
  try {
    dispatch({
      //specifies the type of action
      type: 'REMOVE_TODO',
      // this is the payload (data)
      task: task,
    });
  } catch (error) {
    console.log(error);
    const message =
      error.response && error.response.data.message
        ? error.response.data.message
        : error.message;
    //capture errors if task is not added successfully
    dispatch({
      type: 'REMOVE_TODO_FAIL',
      payload: message,
    });
  }
};
```

References

Aguirre, I. N. (2021, January 21). *State management in React with Redux tutorial.* Medium. https://medium.com/swlh/state-management-in-react-with-redux-tutorial-53fa7dd60d19

Redux fundamentals, part 3: State, Actions, and Reducers. (n.d.). Redux. https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers