React-Redux

Here's a summary of Redux, a state management library for React:

What is Redux?

Redux is a predictable and scalable state container for managing global state in complex applications. It helps to maintain a single source of truth for your application's state, making it easier to manage and debug.

Key Concepts:

1. **Store**: The central hub that holds the entire application's state.
2. **Actions**: Payloads that trigger state changes, typically as objects with a `type` property.
3. **Reducers**: Pure functions that take the current state and an action as input, returning a new state.
4. **Dispatch**: The process of sending an action to the store.

How Redux Works:

1. An application dispatches an action to the store (e.g., a user clicks a button).
2. The store calls each reducer function in sequence, passing the current state and the dispatched action.
3. Each reducer returns a new state based on the previous state and the action received.
4. The store combines all updated states from the reducers and updates its own internal state.

Example:

Suppose we're building a simple to-do list app with two features:

1. Add a task
2. Remove a task

Here's some sample code:

```
 // actions.js
export const ADD_TASK = 'ADD_TASK';
export const REMOVE_TASK = 'REMOVE_TASK';

export function addTask(task) {
  return { type: ADD_TASK, payload: task };
}

export function removeTask(id) {
  return { type: REMOVE_TASK, payload: id };
}
```

```
 // reducers.js
import { combineReducers } from 'redux';
import { ADD_TASK, REMOVE_TASK } from './actions';

const tasks = (state = [], action) => {
  switch (action.type) {
    case ADD_TASK:
      return [...state, action.payload];
    case REMOVE_TASK:
      return state.filter((task) => task.id !== action.payload);
    default:
      return state;
  }
};

export const rootReducer = combineReducers({
  tasks,
});
```

```
 // store.js
import { createStore } from 'redux';
import rootReducer from './reducers';

const store = createStore(rootReducer);

store.dispatch(addTask({ id: 1, text: 'Buy milk' }));
console.log(store.getState().tasks); // [{ id: 1, text: 'Buy milk' }];

store.dispatch(removeTask(1));
console.log(store.getState().tasks); // []
```

In this example:

- We define two actions (`ADD_TASK` and `REMOVE_TASK`) in `actions.js`.
- We create a reducer function (`tasks`) that handles these actions in `reducers.js`.
- We combine multiple reducers using `combineReducers` from `redux`.
- We create the store with our combined reducers.
- We dispatch actions to update the state, observing changes through the `getState()` method.

This is a basic overview of Redux and how it helps manage global state in React applications.

Curated by Brajesh Kumar