Revision Notes: Redux and Async Redux Concepts

Introduction to Redux

Redux is a popular JavaScript library used for managing state in applications. It helps in maintaining a predictable state container for JavaScript apps. Let's dive into the fundamental principles and structure of Redux.

Principles of Redux

1. **Single Source of Truth**:
   - There is only one store in the entire application which holds the complete state.
   - Multiple state slices can exist, but they all reside within the single store.
2. **State is Read-Only**:
   - The only way to change the state is to emit an action. An action is an object that describes what happened.
3. **Changes are Made with Pure Reducer Functions**:
   - Reducers specify how the application's state changes in response to actions sent to the store.
   - Reducers are pure functions which do not produce side effects. They receive the previous state and an action, and return the next state .

Redux Structure

- **Actions**: JavaScript objects that have a `type` field typically and may contain a `payload` .

- **Store**: An object that brings the actions and reducers together, containing the application's state and providing helper methods to access or update the state .

Implementing Redux with Thunk Middleware

Redux Thunk is middleware that allows you to write action creators that return a function instead of an action. Here is how they work:

1. **Return Function Instead of Action**:
   - This function can then perform asynchronous operations and dispatch actions.
   - The function gets the store's `dispatch` method as its argument .
2. **Usage**:
   - Useful for making asynchronous requests, like fetching data from an API【4:4†transcript.txt】.

Async Redux

Async Redux handles asynchronous actions such as API calls. Typically achieved using middleware like Redux Thunk, it improves how state is managed when dealing with asynchronous operations.

Implementing Async Redux

Setup for Async Redux:

- Asynchronous requests alter the state through various stages: loading, success, and error .

Middleware Implementation:

1. **Middleware Setup**:

2. **Async Function Usage**:
    - Within the action creator, async functions fetch data from APIs, updating the state upon success or error .

Example Structure

- Create slices for each state portion, specifying `initialState` and `reducers` .
- Incorporate async functions in middleware to fetch and dispatch actions upon receiving data .

Practical Implementation: Movies and User Components

1. **Movie Slice & Middleware**:
    - Incorporate a `movieSlice` to handle state specific to movies using Redux and Thunk for fetching data .
2. **User Component Example**:
    - Demonstrates fetching user data asynchronously, updating state via dispatched actions .
3. **Pagination Example**:
    - Handles page navigation state using Redux, allowing central management of page state across components .

Conclusion

Redux is a powerful tool for managing state, particularly in React applications with complex state logic or asynchronous data requirements. By understanding the core concepts and practical implementation using middleware like Thunk, developers can create scalable and maintainable applications, efficiently managing state changes and side effects.

Scaler Companion beta