Calling API with Redux

Learning Objectives

By the end of this lesson, you will be able to:

- Outline the best practices for API integration with Redux for managing asynchronous operations
- Implement CRUD operations using JSON server API and Axios and built-in Fetch API
- Develop a React application to implement error handling and loading states in API calls to enhance the user experience

Best Practices for API Integration with Redux

Best Practices for API Integration with Redux

- Separate API-related actions from other actions in the Redux codebase
- Define action types related to API calls explicitly
- Create reusable action creators for API requests
- Leverage middleware like Redux Thunk for handling asynchronous actions
- Implement consistent error handling mechanisms for API requests

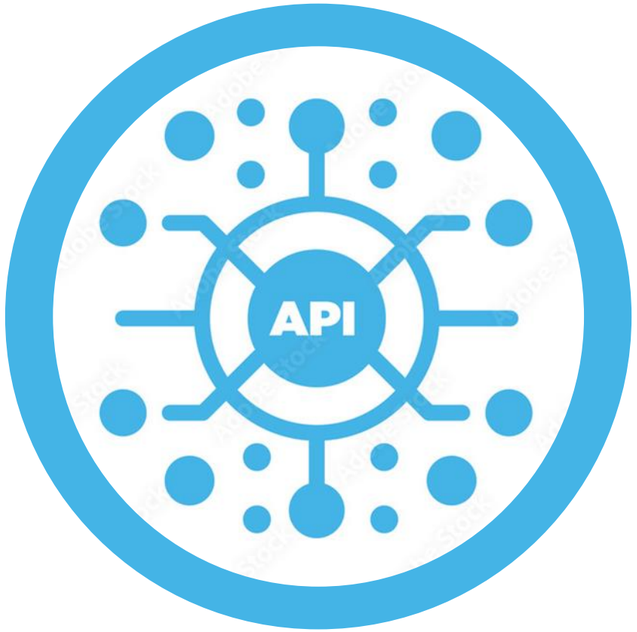
Best Practices for API Integration with Redux

- Manage loading states to provide feedback to users during API requests
- Implement optimistic updates for a better user experience
- Integrate Redux DevTools for debugging and monitoring API-related actions
- Write unit tests for API-related actions and reducers

Performing CRUD Operations Using Axios and Built-in Fetch API

Understanding Representational State Transfer (RESTful) API

It is a set of principles for designing and developing web APIs that are scalable, flexible, and easy to maintain.

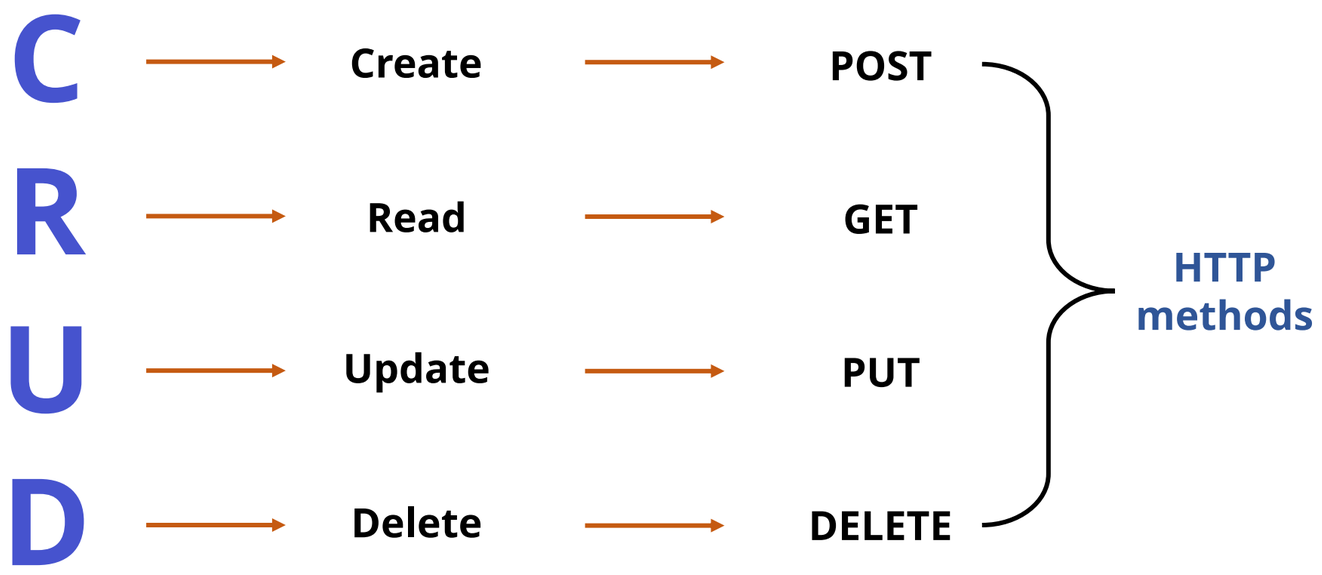


It is a popular architectural approach used for constructing web services that can be consumed by diverse clients, including web, mobile, and desktop applications.

Understanding RESTful API

It uses standard HTTP methods to perform operations on resources.

The four main methods are:



Create Operation Using Axios

It makes a POST request to create new data.

```
// actions.js
import axios from 'axios';
export const createData = (newData) => async (dispatch) => {
  try {
    const response = await axios.post('https://api.example.com/data', newData);
    dispatch({ type: 'CREATE_DATA_SUCCESS', payload: response.data });
  } catch (error) {
    dispatch({ type: 'CREATE_DATA_FAILURE', payload: error.message });
  } };
```

Read Operation Using Axios

It makes a GET request to fetch existing data.

```
// actions.js
import axios from 'axios';


export const readData = () => async (dispatch) => {
  try {
    const response = await axios.get('https://api.example.com/data');
    dispatch({ type: 'READ_DATA_SUCCESS', payload: response.data });
  } catch (error) {
    dispatch({ type: 'READ_DATA_FAILURE', payload: error.message });
  }
};
```

Update Operation Using Axios

It makes a PUT request to update existing data.

```
// actions.js
import axios from 'axios';


export const updateData = (id, updatedData) => async (dispatch) => {
  try {
    const response = await axios.put(`https://api.example.com/data/${id}`, updatedData);
    dispatch({ type: 'UPDATE_DATA_SUCCESS', payload: response.data });
  } catch (error) {
    dispatch({ type: 'UPDATE_DATA_FAILURE', payload: error.message });
  }
};
```

Delete Operation Using Axios

It makes a DELETE request to remove existing data.

```
// actions.js
import axios from 'axios';


export const deleteData = (id) => async (dispatch) => {
  try {
    await axios.delete(`https://api.example.com/data/${id}`);
    dispatch({ type: 'DELETE_DATA_SUCCESS', payload: id });
  } catch (error) {
    dispatch({ type: 'DELETE_DATA_FAILURE', payload: error.message });
  }
};
```

Create Operation Using Fetch API

It makes a POST request to create new data.

```
// actions.js
export const createData = (newData) => async (dispatch) => {
  try {
    const response = await fetch('https://api.example.com/data', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json’, },
      body: JSON.stringify(newData), });
    const data = await response.json();
    dispatch({ type: 'CREATE_DATA_SUCCESS', payload: data });
  } catch (error) {
    dispatch({ type: 'CREATE_DATA_FAILURE', payload: error.message });
  } };
```

Read Operation Using Fetch API

It makes a GET request to create new data.

```
// actions.js
export const readData = () => async (dispatch) => {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    dispatch({ type: 'READ_DATA_SUCCESS', payload: data });
  } catch (error) {
    dispatch({ type: 'READ_DATA_FAILURE', payload: error.message });
  }
};
```

Update Operation Using Fetch API

It makes a PUT request to update existing data.

```
// actions.js
export const updateData = (id, updatedData) => async (dispatch) => {
  try {
    const response = await fetch(`https://api.example.com/data/${id}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json’, },
      body: JSON.stringify(updatedData),  });
    const data = await response.json();
    dispatch({ type: 'UPDATE_DATA_SUCCESS', payload: data });
  } catch (error) {
    dispatch({ type: 'UPDATE_DATA_FAILURE', payload: error.message });
  } };
```

Delete Operation Using Fetch API

It makes a DELETE request to remove existing data.

```
// actions.js
export const deleteData = (id) => async (dispatch) => {
  try {
    await fetch(`https://api.example.com/data/${id}`, {
      method: 'DELETE',
    });
    dispatch({ type: 'DELETE_DATA_SUCCESS', payload: id });
  } catch (error) {
    dispatch({ type: 'DELETE_DATA_FAILURE', payload: error.message });
  }
};
```

Implementing CRUD Operations Using JSON Server API

Problem Statement:

Duration: 20 min

You have been assigned a task to implement a React application for performing CRUD operations using JSON server API.

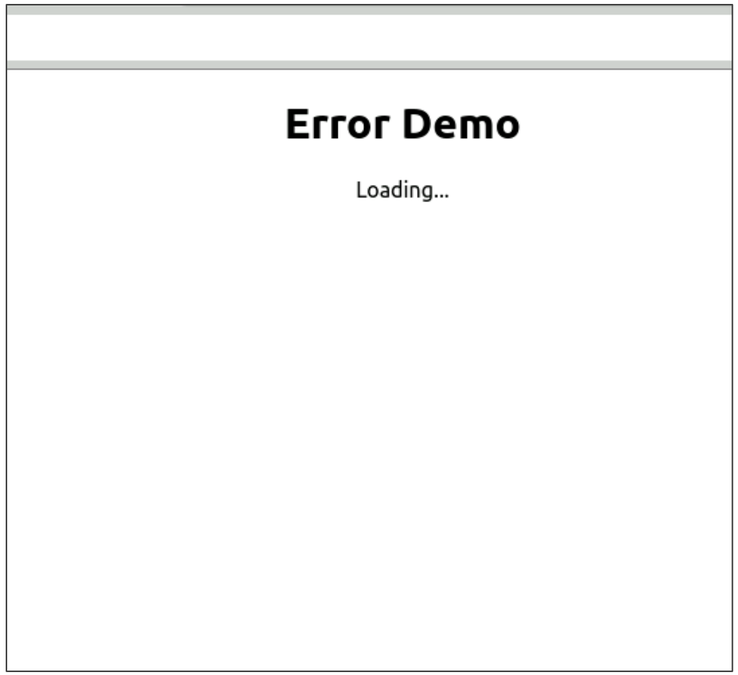
Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up the React project
2. Create the database
3. Create the components file
4. Configure the Redux
5. Modify the index.js and App.js file
6. Run the application

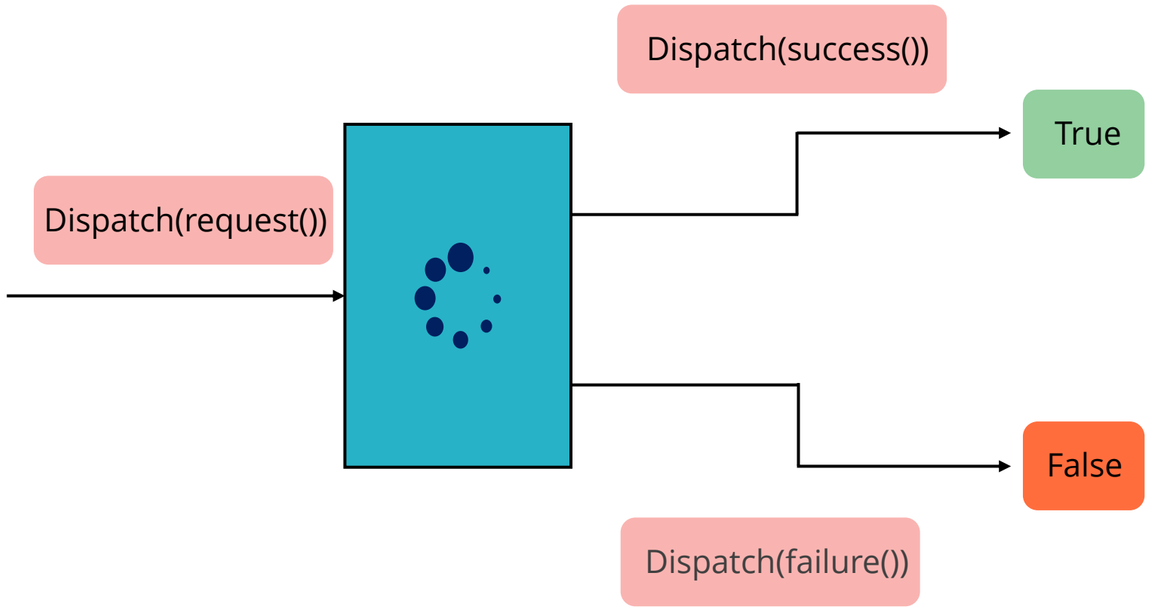
Error Handling and Managing Loading States

Error Handling



- Dispatch a failure action in case of an error during an asynchronous operation
- Provide information about the nature of the error by updating the store with an error state
- Check the error state and display appropriate error messages using the components connected to the Redux store
- Implement UI components to gracefully handle and display errors to users

Managing Loading States



Block diagram for managing loading states

- Dispatch a loading action, before initiating an asynchronous operation (Example: API request)
- Indicate data fetching by setting a loading state in the Redux store
- Update the loading state based on the dispatched loading action
- Set the loading state to true when the loading action is in progress or successful and false on failure

Implementing Error Handling and Loading States

Problem Statement:

Duration: 20 min

You have been assigned a task to implement the error handling and loading states in the React application to enhance the user experience.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up the React project
2. Modify the App.js file for error handling and run the application
3. Modify the App.js file for loading states and run the application

ASSISTED PRACTICE

Creating a React Application With Redux Toolkit

Problem Statement:

Duration: 20 min

You have been assigned a task to create a React application with Redux to interact with a static JSON file for handling data and error.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up the React project
2. Create the db.json file
3. Create components and redux folder under the src folder
4. Configure the index.js file by providing store details
5. Test the application

Creating a React Toolkit Slice

Problem Statement:

Duration: 20 min

You have been assigned a task to create a React application following Redux best practices using Redux Toolkit, interacting with static JSON files for data handling and addressing errors.

ASSISTED PRACTICE

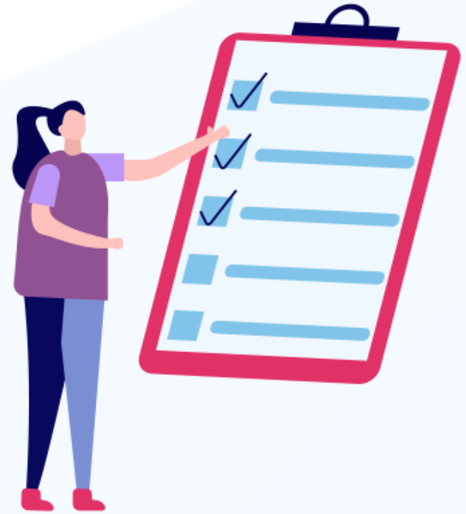
Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up the React project
2. Create a features/posts, store, and components folder
3. Modify the App.js file
4. Modify the index.js file
5. Test the application

Key Takeaways

- Best practices for API integration in a Redux are crucial in software development, including API actions, action types, action creators, and many more.
- RESTful API principles are essential for building scalable web services, and proficiency in making HTTP requests using libraries like Fetch API and Axios is crucial for effective communication in React applications.
- Error handling states include dispatching failure actions and displaying errors.
- Managing loading states include dispatching loading actions and updating loading states.



Knowledge Check

Knowledge Check 1

What is the recommended approach for structuring API-related actions in a Redux codebase?

A. Embed API actions within components

B. Integrate API actions with reducers

C. Disperse API actions across various Redux slices

D. Separate API-related actions in dedicated files or directories

Knowledge Check 1

What is the recommended approach for structuring API-related actions in a Redux codebase?

A. Embed API actions within components

B. Integrate API actions with reducers

C. Disperse API actions across various Redux slices

D. Separate API-related actions in dedicated files or directories

The correct answer is **D**

It is a best practice to separate API-related actions from other actions in dedicated files or directories to enhance maintainability and readability in a Redux codebase.

Knowledge Check 2

What is a RESTful API and why is it essential for web application development?

A. RESTful API is an architectural approach for designing scalable, flexible, and maintainable web APIs

B. RESTful API is a web design framework for creating user interfaces

C. RESTful API stands for remote, server, and transfer, enhancing server efficiency

D. RESTful API is a programming language used for web development

Knowledge Check 2

What is a RESTful API and why is it essential for web application development?

A. RESTful API is an architectural approach for designing scalable, flexible, and maintainable web APIs

B. RESTful API is a web design framework for creating user interfaces

C. RESTful API stands for remote, server, and transfer, enhancing server efficiency

D. RESTful API is a programming language used for web development

The correct answer is **A**

RESTful API is an architectural approach for designing scalable, flexible, and maintainable web APIs. It is essential for web application development as it provides a standardized approach.

Knowledge Check 3

What are the four main HTTP methods used in RESTful APIs for performing operations on resources?

A. ADD, EDIT, REMOVE, and RETRIEVE

B. INSERT, MODIFY, DELETE, and QUERY

C. POST, GET, UPDATE, and DELETE

D. FETCH, STORE, UPDATE, and ERASE

Knowledge Check 3

What are the four main HTTP methods used in RESTful APIs for performing operations on resources?

A. ADD, EDIT, REMOVE, and RETRIEVE

B. INSERT, MODIFY, DELETE, and QUERY

C. POST, GET, UPDATE, and DELETE

D. FETCH, STORE, UPDATE, and ERASE

The correct answer is **C**

The four main HTTP methods in RESTful APIs are POST (create), GET (read), UPDATE, and DELETE. They are used for performing operations on resources.