Middleware in Redux: Redux Thunk

Learning Objectives

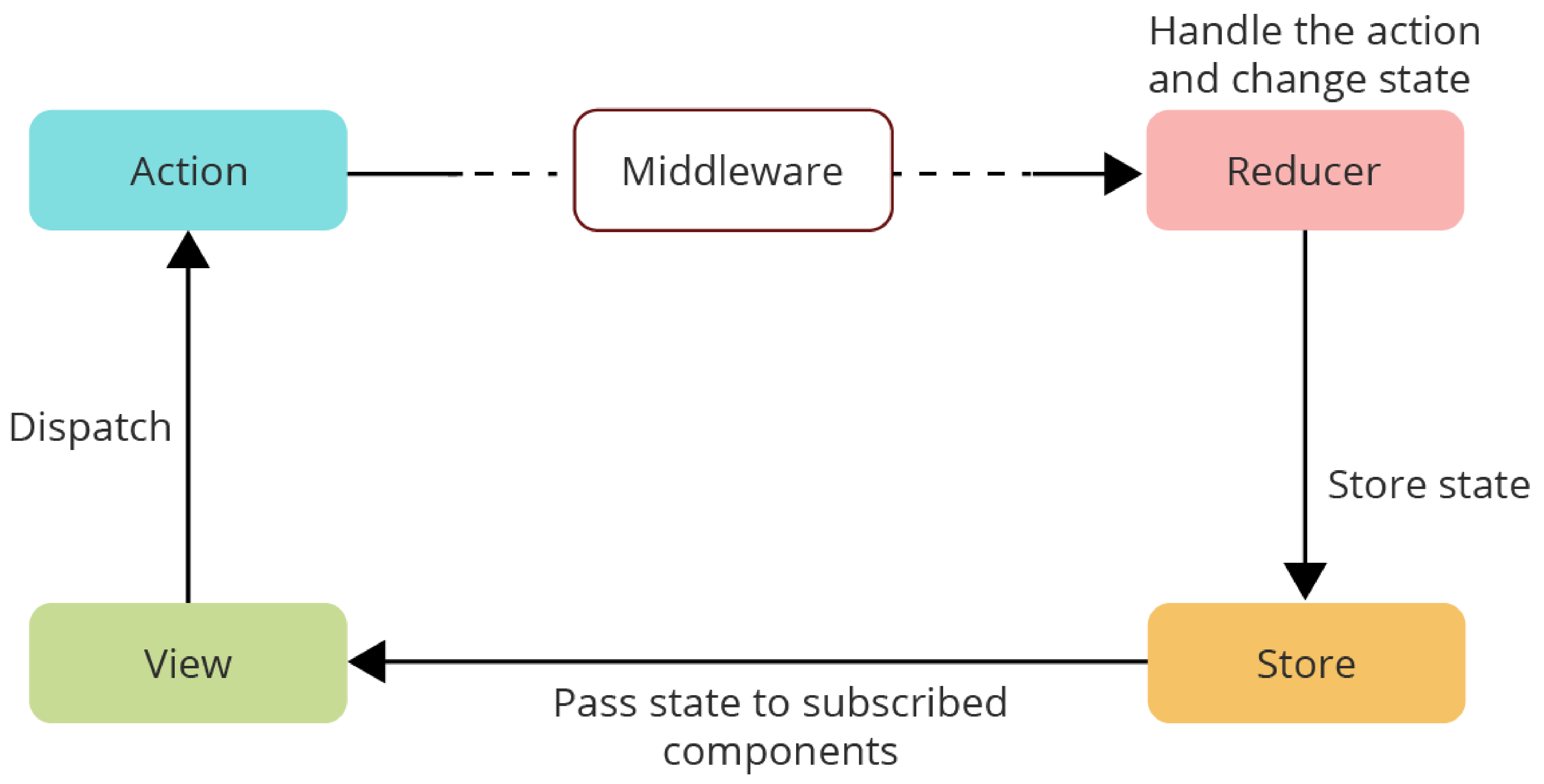
By the end of this lesson, you will be able to:

- Implement Redux Thunk middleware to handle asynchronous actions, allowing action creators to return functions for tasks like API calls
- Outline the steps for making API requests within Redux Thunk to perform API requests in an organized and manageable way
- List the steps to handle asynchronous actions and update the store with data
- Develop a React application by using React Redux Thunk and Axois for efficient state management and asynchronous API requests

Introduction to Redux Middleware

What Is Redux Middleware?

It is a third-party extension point between dispatching an action and the moment it reaches the reducer.


Handle the action and change state
Action
Middleware
Reducer
Dispatch
Store state
View
Store
Pass state to subscribed components

Working of Redux middleware

Logging Middleware

```
const loggingMiddleware = (store) => (next) => (action) => {
  console.log(`Action Type: ${action.type}, Payload:
${action.payload}`);
  return next(action);
};

// Apply the logging middleware to the Redux store
const store = createStore(
  rootReducer,
  applyMiddleware(loggingMiddleware)
);
```

The above code shows how to apply logging middleware to the Redux store using the **loggingMiddleware** function.

Implementing Redux Thunk as Middleware for Handling Asynchronous Actions

What Is Redux Thunk?

It is a state management library commonly used with React.

Redux Thunk

It facilitates the handling of asynchronous operations in Redux applications.

Redux Thunk: Overview

Asynchronous actions

- Redux Thunk primarily handles asynchronous actions.
- It resolves this issue by allowing action creators to return functions rather than plain action objects.

Function returns

- Redux Thunk empowers action creators to return functions.
- These functions receive **dispatch** and **getState** functions as arguments.

Redux Thunk: Overview

Delayed actions

- Redux Thunk enables the dispatch of actions with delays or under specific conditions.
- It is valuable for scenarios involving API calls, where waiting for responses is crucial.

Middleware integration

- Redux Thunk is integrated into the Redux store as middleware.
- During store creation, apply the middleware to activate asynchronous behavior.

Redux Thunk Middleware Integration

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(rootReducer,
applyMiddleware(thunk));
```

The above code shows the integration of Redux Thunk into the Redux store using **createStore** and **applyMiddleware** functions.

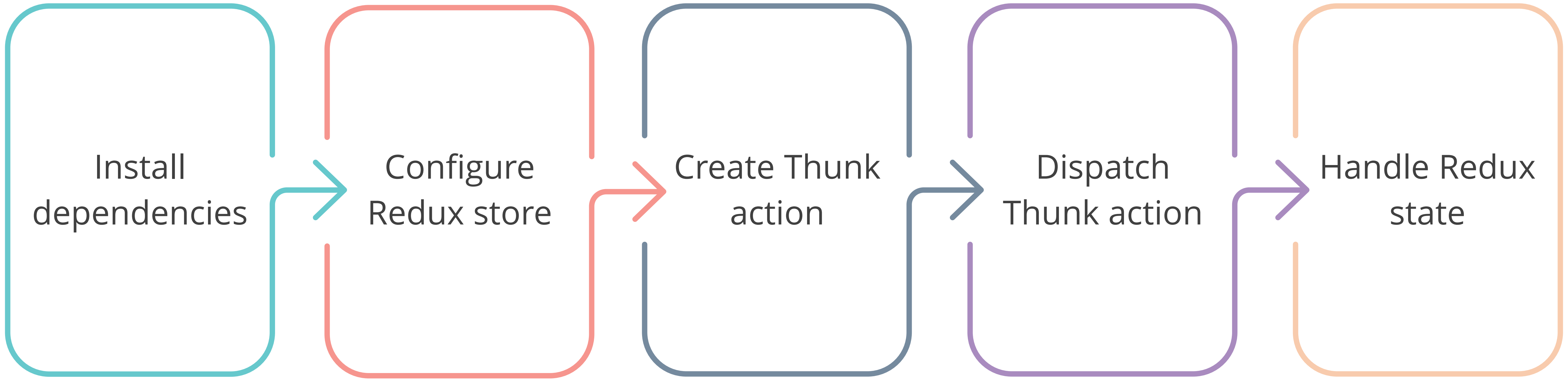
Redux Thunk Action Creator

```
const fetchData = () => {
return async (dispatch, getState) => {
    dispatch({ type: 'FETCH_DATA_REQUEST' });
    try {
      const response = await fetch('https://api.example.com/data');
      const data = await response.json();
      dispatch({ type: 'FETCH_DATA_SUCCESS', payload: data });
} catch (error) {
      dispatch({ type: 'FETCH_DATA_FAILURE', payload: error.message });
    }
  };
};
```

The above code shows the creation of an action creator by using **fetchData** function.

Making API Requests Within Redux Thunk

Steps for Making API Requests



The primary purpose of these steps is to enable the users to make API requests in a more organized and manageable way.

Making API Calls Using Libraries

The two ways to make API calls in React are:

Using a third-party library like Axios

Using the built-in Fetch API

The primary objective is to build modern web applications.

Axios and Built-in Fetch API

Axios

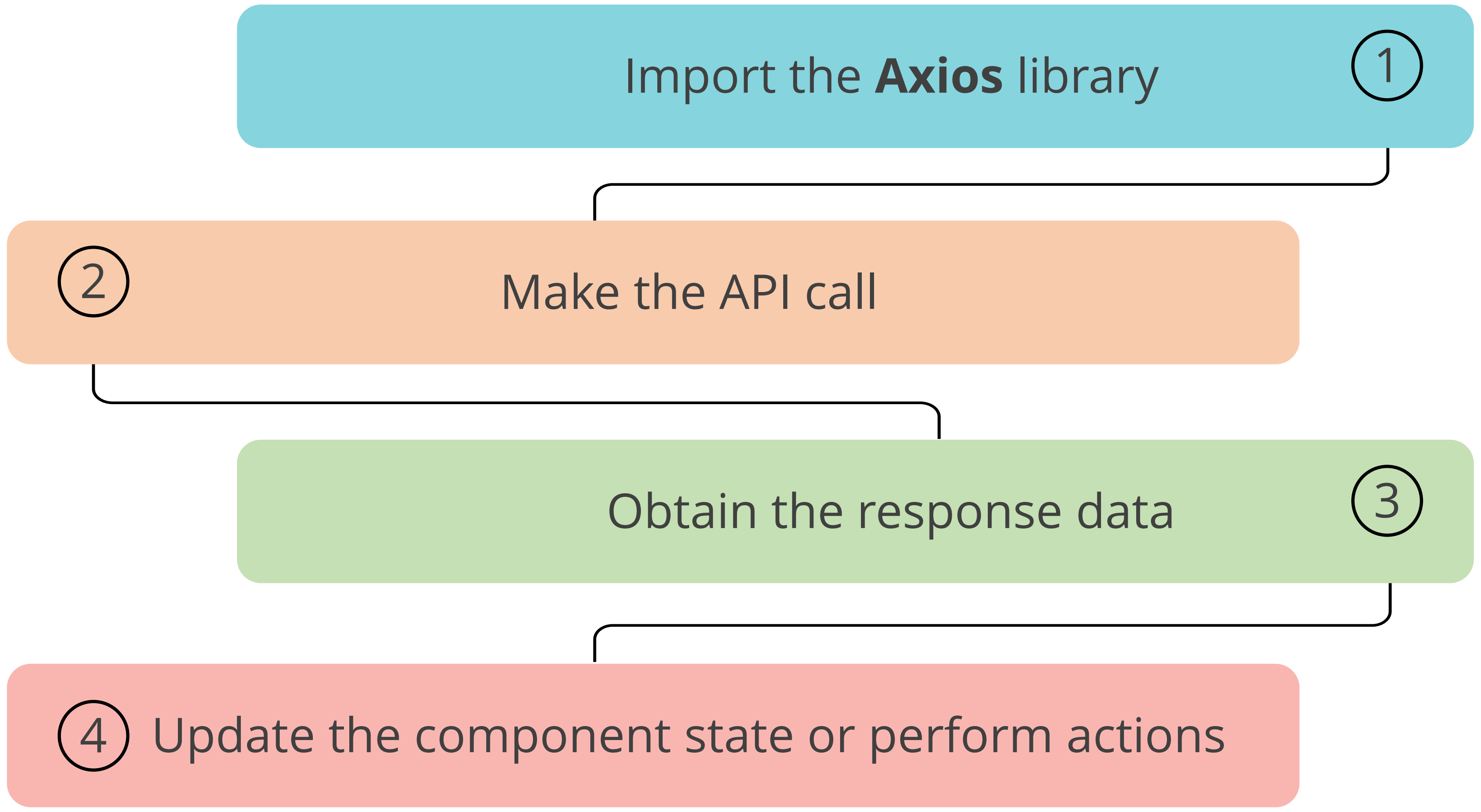
- It is a popular JavaScript library for making HTTP requests.
- It offers simplicity, flexibility, and built-in features like request or response transformations.
- It is well suited for handling API requests within Redux Thunk.

Built-in Fetch API

- It is a modern native JavaScript API for making network requests.
- It provides a simple and clean interface.
- It may require additional handling for certain features compared to Axios.

Using Axios

The steps to call an API using Axios are as follows:

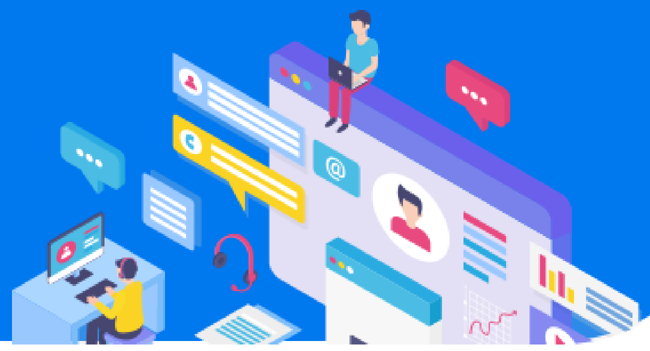

Import the **Axios** library
1
2
Make the API call
Obtain the response data
3
4
Update the component state or perform actions

Using Built-in Fetch API

The steps to call an API using Fetch API are as follows:

1. Construct the request object
2. Call the **fetch()** function
3. Handle the response
4. Update the component state or perform other actions

Creating a React Redux Thunk API for Employee Operations

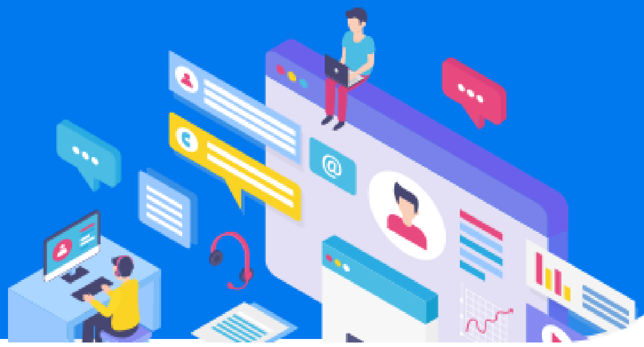


Problem Statement:

Duration: 20 min

You have been assigned a task to create a robust, React Rredux employee database that seamlessly stores and retrieves data from a JSON file, powered by Axios and Thunk middleware.

Assisted Practice: Guidelines



Steps to be followed:

1. Create and set up the React project
2. Create a JSON file with a set of static employee details
3. Create an action folder
4. Create a reducers folder
5. Create a components folder
6. Configure store and Thunk details
7. Test the application

Handling Asynchronous Actions and Updating the Store with Data

Handling Asynchronous Actions and Updating the Store with Data

Handling asynchronous actions

- Redux Thunk manages asynchronous tasks using functions (thunks) for operations like API calls.
- It is accessible to **dispatch** and **getState**, enabling asynchronous task performance and multiple actions dispatched.

Updating the store with data

- Redux Thunk dispatches a **start** action for asynchronous operations to update the store.
- It follows up with a **success** and **failure** action based on the outcome.

Handling Asynchronous Actions and Updating the Store with Data

The steps are as follows:

1. Install dependencies
2. Configure the store
3. Define asynchronous action creator
4. Update reducer
5. Connect component

Step 1: Install Dependencies

```
npm install redux-thunk


//redux-thunk middleware for handling asynchronous actions
in Redux.
```

The above code shows the installation of dependencies using npm.

Step 2: Configure the Store

```
// store.js
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;
```

The above code shows the configuration of the Redux store by applying **createStore** and **applyMiddleware** functions.

Step 3: Define Asynchronous Action Creator

```
// actions.js

import axios from 'axios';

export const fetchData = () => async (dispatch) => {

  dispatch({ type: 'FETCH_DATA_REQUEST' });

 try {

    const response = await
axios.get('https://api.example.com/data');

    dispatch({ type: 'FETCH_DATA_SUCCESS', payload: response.data
});

  } catch (error) {

    dispatch({ type: 'FETCH_DATA_FAILURE', payload: error.message
});

  }

};
```

The above code shows the creation of an action creator that handles the asynchronous operation.

Step 4: Update Reducer

```
// reducer.js
const initialState = {
  data: [],
  loading: false, error: null,};
const dataReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'FETCH_DATA_REQUEST':
      return { ...state, loading: true, error: null };
 case 'FETCH_DATA_SUCCESS':
      return { ...state, loading: false, data: action.payload };
    case 'FETCH_DATA_FAILURE':
      return { ...state, loading: false, error: action.payload };
    default:
      return state;
  }}; export default dataReducer;
```

The above code shows that, due to the reducer, the handling of actions involves dispatching through an asynchronous operation.

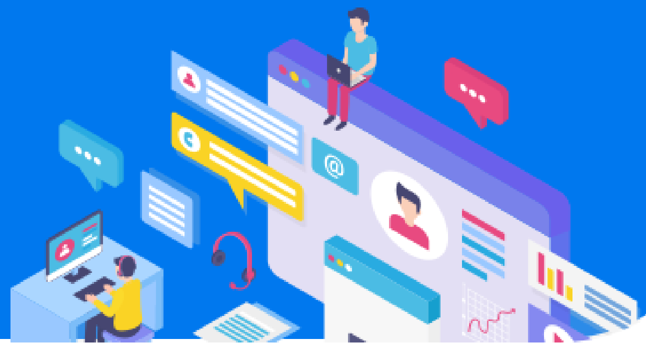
Step 5: Connect Component

```
// MyComponent.js
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { fetchData } from './actions';
const MyComponent = ({ data, loading, error, fetchData }) => {
  useEffect(() => {
    fetchData();
  }, [fetchData]);
return ( <div>
      {loading && <p>Loading...</p>}
 {error && <p>Error: {error}</p>}
      {data && <p>Data: {JSON.stringify(data)}</p>}
    </div>
  ); };
```

```
const mapStateToProps = (state) => ({
  data: state.data,
  loading: state.loading,
  error: state.error,
});
export default connect(mapStateToProps, { fetchData
})(MyComponent);
```

The above code shows the connection of the React component to the store and dispatches the asynchronous action.

Creating an Airport Search React Redux Thunk Application



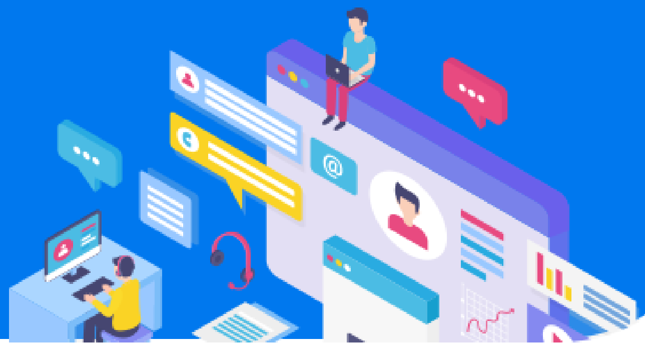
Problem Statement:

Duration: 20 min

You have been assigned a task to create a React application for airport search with Redux Thunk, enabling users to efficiently search and select airports by name, code, or city.

ASSISTED PRACTICE

Assisted Practice: Guidelines



Steps to be followed:

1. Create and set up the React project
2. Create an airport.json file
3. Create a services folder
4. Create a component folder
5. Configure the index.js file
6. Test the application

Key Takeaways

- Redux Thunk serves as middleware in a Redux application, enabling the handling of asynchronous actions.
- Making API calls is a fundamental aspect of building modern web applications, and they can be made using the built-in Fetch API or a third-party library like Axios.
- Redux Thunk dispatches a start action, which follows success and failure actions and updates the store for managing asynchronous tasks in the Redux application.

Knowledge Check

Knowledge Check

1

What is the primary purpose of Redux Thunk in a Redux application?

A. Style components

B. Handle asynchronous actions

C. Optimize reducer performance

D. Manage component state

Knowledge Check

1

What is the primary purpose of Redux Thunk in a Redux application?

A. Style components

B. Handle asynchronous actions

C. Optimize reducer performance

D. Manage component state

The correct answer is **B**

Redux Thunk serves as middleware specifically designed to handle asynchronous operations, such as making API calls, within a redux application.

Knowledge Check

2

What is the primary purpose of Redux middleware?

A. Handle CSS styling

B. Manage state in React components

C. Provide an extension point between action dispatch and reducer execution

D. Facilitate form validation

Knowledge Check

2

What is the primary purpose of Redux middleware?

A. Handle CSS styling

B. Manage state in React components

C. Provide an extension point between action dispatch and reducer execution

D. Facilitate form validation

The correct answer is **C**

Redux middleware acts as an extension point between action dispatch and reducer execution, allowing customization of the Redux data flow.

Knowledge Check

3

How is Axios different from the Fetch API?

A. Axios is for making HTTP requests, while Fetch API is for making network requests.

B. Fetch API is more flexible and feature-rich than Axios.

C. Axios requires additional handling for certain features compared to Fetch API.

D. Axios is a library, while Fetch API is specifically designed for making HTTP requests.

Knowledge Check

3

How is Axios different from the Fetch API?

A. Axios is for making HTTP requests, while Fetch API is for making network requests.

B. Fetch API is more flexible and feature-rich than Axios.

C. Axios requires additional handling for certain features compared to Fetch API.

D. Axios is a library, while Fetch API is specifically designed for making HTTP requests.

The correct answer is **A**

Axios is a popular JavaScript library designed for making HTTP requests, while Fetch API is a native JavaScript API for making network requests.