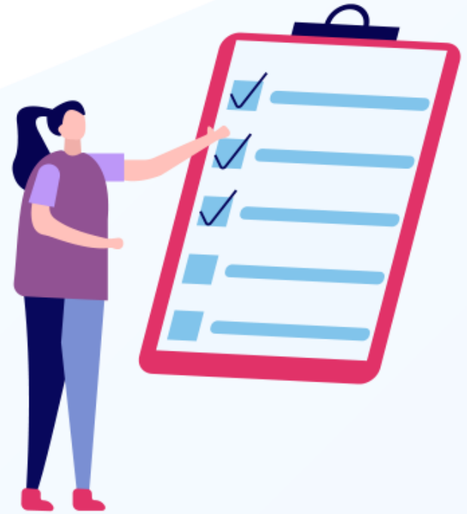
# Redux Toolkit (RTK)

# Learning Objectives

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

- Identify the process of setting up the Redux Toolkit in a project to streamline the state management
- Apply middleware integration techniques to extend and modify the Redux Toolkit functionality for enhanced state management and action handling
- Create the React application with the Redux toolkit to store, delete, edit, and display course details
- Create the React application with Redux using the Thunk module to retrieve details from a JSON file



# Introduction to Redux Toolkit

# Redux Toolkit

It is an advanced, opinionated, and practical toolkit designed to simplify the process of writing Redux logic.


It aims to address the common complaints about Redux being too verbose and complicated by providing a set of tools that streamline Redux development.

# Redux Toolkit

Some key concepts and functions that enhance the Redux experience are:

| | |
|---|---|
| **configureStore function** | Simplifies the process of store setup |
| **createSlice function** | Abstracts the process of writing action creators and reducers |
| **Immer function** | Reduces the boilerplate code (computer language text that users can reuse) and the chances of making immutability-related mistakes |
| **createAsyncThunk function** | Abstracts the process of dispatching actions related to an asynchronous request |

Normalization of best practices encourages writing more consistent, scalable, and maintainable Redux codes.

# Advantages of RTK

### Simplified store configuration

Sets up the Redux Thunk middleware for asynchronous actions and integrates the Redux DevTools extension

### Reduced boilerplate code

Leads to cleaner and more concise codebases

### Built-in immutability with Immer

Simplifies reducer functions, allowing developers to write more readable and intuitive state update logic without worrying about accidental state mutations

### Simplified asynchronous logic

Abstracts the standard pattern of dispatching actions during an asynchronous request and reduces the need for boilerplate code in async logic

# Advantages of RTK

### Standardized best practices

Provides a set of standardized tools and methods to guide developers towards a consistent pattern

### Enhanced developer experience

Enhances the overall developer experience with its simplified API and reduction in boilerplate

### Organized state logic

Makes scaling applications manageable and promotes maintainable code structure

### Improved debugging

Simplifies the process of tracing issues and writing tests

# Setting Up Redux Toolkit in a Project

The process is as follows:

1. Create a React application

```
npm create vite@latest redux-app --
--template react
```

2. Move to the directory

```
cd redux-app
```

# Setting Up Redux Toolkit in a Project

The process is as follows:

3. Install preprovided modules

```
npm install
```

4. Install React Redux and Redux Toolkit

```
npm install react-redux
@reduxjs/toolkit
```

# Simplifying Redux Store Setup with Redux Toolkit

# Understanding the configureStore Function

This function replaces the traditional process of manually combining reducers, applying middleware, and creating the store.

The function takes an object as its argument, allowing users to define various aspects of the store.

# Integrating Slices Using configureStore

Slices are small reducers that can be combined to form the root reducer.

Slices encapsulate the logic for a specific piece of the state, making the codebase more modular and maintainable.

Once the slices are defined, integrating them into the Redux store is straightforward using **configureStore**.

# Integrating Middleware

Redux middleware provides a way to extend the store's capabilities.

Common use cases for middleware include:

- Logging data
- Performing asynchronous actions
- Handling side effects

RTK allows the inclusion of middleware directly in the **configureStore** function.

# Simplified Store Configuration

The example showcases a comprehensive store setup with multiple slices, asynchronous logic handling, and custom middleware.

```
import { configureStore, getDefaultMiddleware } from '@reduxjs/toolkit';
import thunk from 'redux-thunk';
import logger from 'redux-logger';
import rootReducer from './rootReducer';

const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(thunk, logger),
  devTools: process.env.NODE_ENV !== 'production',
});

export default store;
```

# Creating the React Application with Redux Toolkit

**Duration: 20 min**

**Problem Statement:**

You have been assigned a task to perform the steps to implement the React application with the Redux toolkit to store, delete, edit, and display course details.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to be followed:

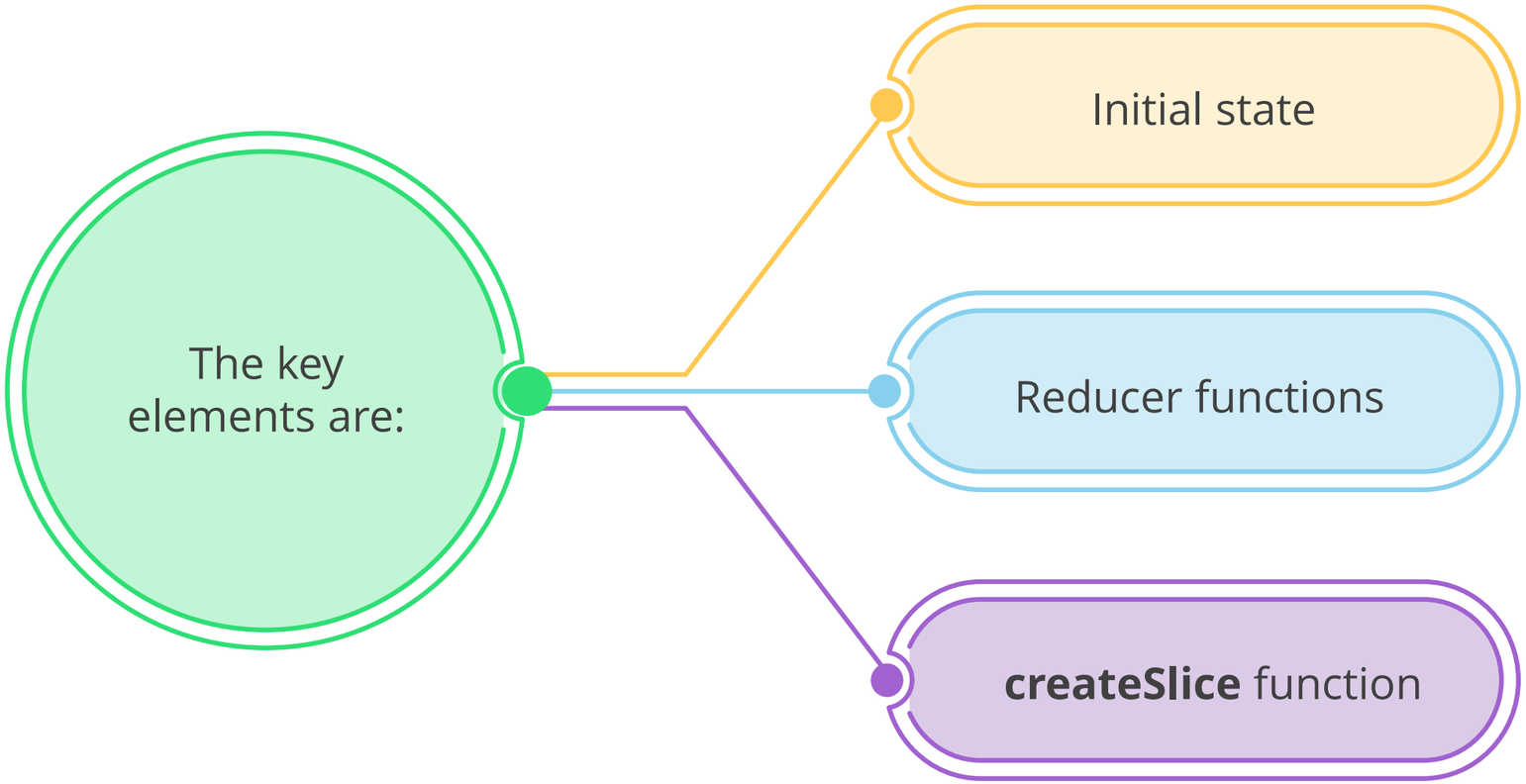
1. Create and set up the React project
2. Create the **courseSlice.js** file
3. Create the store
4. Configure store in **index.js** file
5. Create **AddCourse** file
6. Modify the **App.js** file
7. Test the application

ASSISTED PRACTICE

# Reducing Boilerplate with createSlice

# Fundamentals of createSlice

It is a utility function designed to minimize boilerplate and streamline the process of creating Redux actions and reducers.



# Elements of createSlice

**Initial state:**

This represents the starting point of the state tree.

```
// Example: Initial state for a counter slice
const initialState = {
  value: 0,
};
```

# Elements of createSlice

**Reducer functions:**

Each key-value pair corresponds to a specific action type and the logic to update the state in response to that action.

```
// Example: Reducer functions for a
counter slice
const reducers = {
  increment: (state) => {
    state.value += 1;
  },
  decrement: (state) => {
    state.value -= 1;
  },
};
```

# Elements of createSlice

**createSlice function:**

It is an object that includes automatically generated actions and a reducer based on the provided initial state and reducer functions.

```
// Example: Creating a counter slice with createSlice
import { createSlice } from '@reduxjs/toolkit';


const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers,
});
```

# Automating Action Creators and Reducers

**createSlice** automates the generation of action creators and reducers, reducing the need for manual coding and minimizing the boilerplate.

**Action creators**

```
// Example: Automatically generated action creators
const { increment, decrement } = counterSlice.actions;
```

**Increment** and **decrement** are action creators that users can use directly in their components.

# Automating Action Creators and Reducers

**Reducers**

```
// Example: Automatically generated reducer
const counterReducer = counterSlice.reducer;
```

**counterReducer** is a reducer function that can be included in the Redux store configuration.

# Managing Asynchronous Actions in Redux Toolkit

# Introduction to Async Actions

Redux uses asynchronous actions to retrieve data from external APIs, databases, or any other external data source.

These actions involve three different action types:

**Request**

It involves dispatching an initial action to signal the start of the data retrieval process.

**Success**

Success action is dispatched with the payload containing the data once the data is retrieved.

**Failure**

If there is an error during the data retrieval process, an error action is dispatched with an error message.

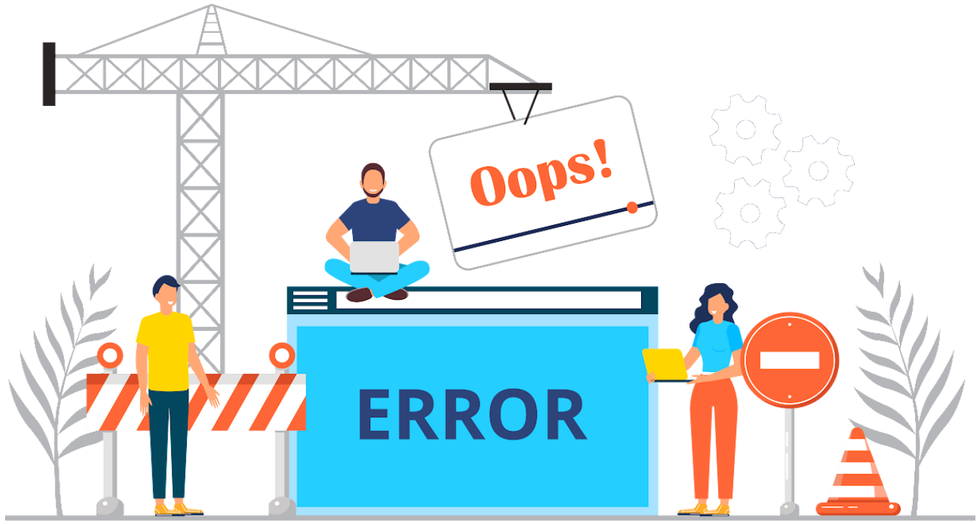
# Handling Loading States

Asynchronous operations often involve loading states to inform the happenings in the background.

The **createAsyncThunk** function automatically generates action types for pending, fulfilled, and rejected states, making it easy to handle loading states in the reducers.

# Accessing Loading States in Components

Users can check the status in the Redux store to conditionally render content based on the loading state.




This ensures a smooth user experience while displaying loading indicators or error messages as needed.

# Handling Side Effects with createAsyncThunk

# Understanding createAsyncThunk

**createAsyncThunk** is a function that streamlines the process of managing asynchronous logic in Redux applications.

A **thunk** function performs asynchronous operations and dispatches actions based on the results.

# Error Handling and Dispatching Actions from Thunks

**createAsyncThunk** provides a convenient way to handle errors within the thunk and dispatch actions accordingly.

When an error occurs during the API calls:


In some cases, users might want to catch and handle the error within the thunk without rethrowing it.

# Creating a React Redux Toolkit Rest API Cart Application

**Duration: 20 min**

**Problem Statement:**

You have been assigned a task to implement the React application with Redux using the Thunk module to retrieve mobile details from a JSON file.

# Assisted Practice: Guidelines

Steps to be followed:

1. Create and set up the React project
2. Create the **mobiles.json** file
3. Create a features folder
4. Create a components folder
5. Add user-defined components in the **App.js** file
6. Configure the **index.js** file with store details
7. Test the application

# Key Takeaways

- Redux Toolkit aims to address the common complaints about Redux being too verbose and complicated by providing a set of tools that streamline Redux development.
- configureStore function replaces the traditional process of manually combining reducers, applying middleware, and creating the store.
- Slices encapsulate the logic for a specific piece of the state, making the codebase more modular and maintainable.
- The createAsyncThunk function automatically generates action types for pending, fulfilled, and rejected states, making it easy to handle loading states in the reducers.


# Knowledge Check

**Knowledge Check**

**1**

**Which function abstracts the process of writing action creators and reducers?**

A. createAsyncThunk

B. createSlice

C. configureStore

D. Immer

**Knowledge Check 1**

**Which function abstracts the process of writing action creators and reducers?**

A. createAsyncThunk

B. createSlice

C. configureStore

D. Immer

The correct answer is **B**

**createSlice is a function that abstracts the process of writing action creators and reducers.**

**Knowledge Check 2**

**Which of the following functions in Redux simplifies the process of integrating defined slices into the store?**

A. createSlice

B. Immer

C. createAsyncThunk

D. configureStore

**Knowledge Check 2**

**Which of the following functions in Redux simplifies the process of integrating defined slices into the store?**

A. createSlice

B. Immer

C. createAsyncThunk

D. configureStore

The correct answer is **D**

**Once the slices are defined, integrating them into the Redux store is straightforward using configureStore.**