



CSSE4103-FULL STACK WEB DEVELOPMENT

SCL Activity 1 – Shopping Cart (Redux Toolkit CRUD)

The objectives of this activity are to:

- Design the front-end of a React application using Reactstrap for responsive and modern UI components.
- 2. Implement interactive elements and forms to enhance user experience.
- 3. Use Reactstrap components like Button, Navbar, Form, Card, Collapse, and Input to build a structured, functional layout.
- 4. Apply form validation using Yup and react-hook-form for robust input handling and validation.
- 5. Dispatch actions to modify the store.
- 6. Design and implement reducers.
- 7. Create and integrate action creators to handle CRUD operations

Instructions:

- 1. Download the activity src files from E-learning, unzip the folder in your Desktop.
- 2. Open the folder in Visual Studio Code.
- 3. Delete the package-lock.json from the client folder.
- 4. Open a terminal, change directory to the client folder and execute the command: **npm** install.
- 5. Start the server.

Section 1 - Front-End Development

- 1. In the **App** component:
 - a. Add the following routes:
 - i. /-render the Home component
 - ii. /manage render the ManageProducts component
 - iii. /update render the the UpdateProduct component
 - iv. /cart render the Cart component
- 2. In the **Header** component:
 - a. Create the corresponding link to the created routes.
- 3. In the **Home** component, use the ReactStrap layout component for the given below:

Banner Image	
Product Component	

- a. In the first row, display the banner image.
- b. Second row, render the Product Component.





Section 2 - Redux Toolkit Setup

- 1. In src/Features/ProductSlice.js:
 - a. Import the ProductsData from the ProductsData.js file.
 - b. Create a variable for the initialState and assign the data from the ProductsData.
 - c. Create a slice with the following details:
 - i. The name is "products".
 - ii. Set the initialState to the initialState variable
 - iii. Create the reducers: **addProduct**, **deleteProduct**, **updateProduct**. Write the corresponding codes to implement the respective reducers.
 - d. Export the all the reducer functions.

2. In src/Store/store.js:

- a. Import the productReducer.
- b. Write the reducer configuration for the Redux store to manage the product state slice using the productReducer.

3. In index.js:

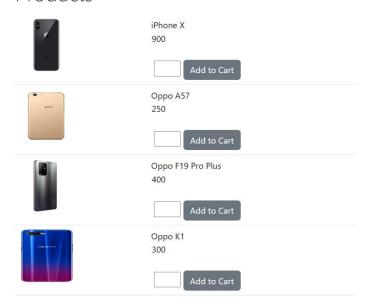
- a. Import the Redux store.
- b. Import the Provider component from react-redux.
- c. Integrate the Redux store into your React application.

Section 3 – Retrieving/Reading value from the Redux Store –(C-R-U-D)

4. In **Products** component:

- a. Import the useSelector from react-readux.
- b. Declare a variable to store the current value of the producs state from the Redux store.
- c. Use the array map method to display the values from the state. Refer to the expected UI design below.

Products







Section 4 - Validation

- 5. In /Validation/ ProductValidation.js, write the code to implement the following validations:
 - a. id must be a number and required.
 - b. **title** must be a string and required, minimum of 4 characters and maximum of 20 characters.
 - c. **price** must be a number and required.
 - d. Import this schema in the **ManageProducts** component.
- 6. In **ManageProducts** component:
 - a. Create the state variables required for all the data from the form.
 - b. Use the register function to register all form elements in the react-hook-form and implement the onChange handler with the register function.
 - c. Display an error message display for each validation error encountered.

Section 5 – Create/Adding new value to the Redux Store (C-R-U-D)

- 7. In ManageProducts component:
 - a. Import the useSelector and useDispatch hook from the react-redux.
 - a. Declare a variable for the useDispatch.
 - b. Import the addProduct reducer from the ProductSlice file.
 - c. Update the function **onSubmit()** to perform the following:
 - i. Use a try...catch() block.
 - ii. Declare a variable **productData** which is an object with properties that will contain the data from the submitted form.
 - iii. Compute the price to be saved on the state based on the following:

Price (from the form)	Tax
Less than 200	0
200-500	10% of price
501-1000	20% of price
More than 1000	25% of price

price = price + tax

- iv. Use the dispatch hook to call the function **addProduct** and pass as argument the productData object.
- d. In the <form> tag, write the code to trigger the **onSubmit** function and execute the required validation.

Section 7 – Retrieving/Reading value from the Redux Store –(C-R-U-D)

- 8. In **ManageProducts** component:
 - a. Declare a variable and retrieve the value of the products from the store.
 - b. In the table, use the map function to display the values from the Redux Store in the browser as shown below.

ID	Image	Title	Price		
124		iPhone X	900	Delete	Update
125	-	Oppo A57	250	Delete	Update
126	•	Oppo F19 Pro Plus	400	Delete	Update
127		Орро К1	300	Delete	Update
128	3-	Realme C35	150	Delete	Update





Section 4 - Deleting values from the Redux Store -(C- R -U-D)

- 9. In ManageProducts component:
 - a. Import the **deleteProduct** reducer from the ProductSlice file.
 - b. Create a new function **handleDelete** and dispatch the **deleteProduct** reducer with the id as the parameter.
 - c. In the table displaying the values from the state, add a delete button.
 - d. In the delete button, call the **handleDelete** function and pass the product id as the argument.

Section 5 – Updating values from the Redux Store – (C - R – U – D)

- 10. In **App.js**, create a new route "/update" to render the UpdateProduct component. Add the route parameter **prod_id**.
- 11. In **ManageProducts** component:
 - a. Import the Link component from react-router-dom.
 - b. In the table displaying the values from the state, add an update button.
 - c. In the update button, use the Link component to redirect to the /update route and pass the product id in as route argument.

12. In **UpdateProduct** component:

- a. Import the useSelector and useDispatch from react-redux
- b. Import the **useParams** hook from react-router-dom.
- c. Import the **useNavigate** from react-router-dom.
- d. Import the **updateProduct** reducer from the ProductSlice
- e. Retrieve the route parameter using useParams.
- f. Store in a variable the value of the products state.
- g. Create variable for the dispatch.
- h. Create variable for the useNavigate.
- Create the function findbyProductId() to search the product id from the variable containing the value of the products from the redux store. Function should return the found product.
- j. Invoke the **findbyProductId()** and pass the variable containing the product id from the route parameter. Save the return value in a variable.
- k. Set the values from the product to be updated object as initial value of the state.
- I. Complete the **handleUpdate** function by copying the code from the onSubmit() function in the **ManageProducts** component, as they share the same logic..
- m. In the **handleUpdate**() function, change the dispatch call to **udpateProduct** instead of **addProduct** reducer.
- n. Change the alert message to "Product updated".
- o. Use the **usenavigate** hook to redirect to the ManageProduct component after the dispatch call.
- p. Copy the form from the **ManageProduct** for adding a new Product and paste it in the second row.
- q. In each of the input form elements, add the attribute **value** and assign the value of the corresponding state variables.

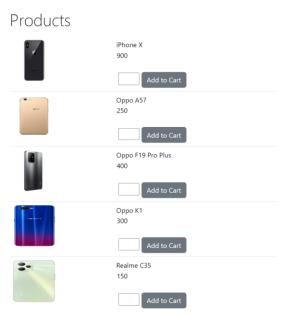






Example Application Output:





© 2024 Jasmine Shop. All rights reserved.

Salalah, Oman | Phone: (123) 456-7890 | Email: jasmine@jasmineshop.com

<u>Privacy. Policy | Terms of Service</u>







