Redux 2.4_CW Exercises

ex01: set up redux store

challenge

1. Create a file named `store.js`.
2. Import necessary dependencies: `applyMiddleware`, `createStore`, `redux-thunk`, and your `financeReducer`.
3. Create the Redux store using `createStore` and apply `redux-thunk` middleware.
4. Export the configured store.

understanding

In a typical Redux application, actions are synchronous, meaning they happen immediately when you dispatch them. For instance, increasing a counter value happens right away.

However, many real-world tasks take time, like fetching data from a server or saving data to a database. You can't do these tasks synchronously because they may take a while.

Redux Thunk is like a helper that extends Redux's abilities. It lets you dispatch actions that are functions instead of plain objects.

solution

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

const store = createStore(financeReducer, applyMiddleware(thunk))

export default store
```

COPY

ex02: create redux reducers

challenge

1. Create `reducers.js` File: Start by creating a new file named in your project.

2. Set Up Initial State:

   - In `reducers.js`, define an initial state object called `initialState`. This object should have the following properties:

     - `income`: An empty array to hold income data.
     - `expenses`: An empty array to hold expenses data.
     - `savings`: An empty array to hold savings data.
     - `loading`: A boolean set to `false` initially to indicate that no data is being loaded.
     - `error`: Initially set to `null` to indicate that there are no errors.

3. Create `financeReducer` Function:

   - Define a reducer function named `financeReducer`
   - You should create cases for action types like `**FETCH_INCOME_SUCCESS**` , `FETCH_DATA_LOADING`, and handle failure with `**'FETCH_INCOME_FAILURE'**`

solution

```
const initialState = {
  income: [],
  expenses: [],
  savings: [],
  loading: false,
  error: null,
}

const financeReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'FETCH_INCOME_SUCCESS':
      return {
        ...state,
        income: action.payload,
        loading: false,
        error: null,
      }

    case 'FETCH_INCOME_FAILURE':
      return {
        ...state,
        loading: false,
        error: 'Error fetching income data',
      }
    case 'FETCH_DATA_LOADING':
      return {
        ...state,
        loading: true,
      }
    default:
      return state
  }
}

export default financeReducer
```

COPY

ex02.1: define redux actions - income

challenge

1. Create an action creator function named `fetchIncome`. It will be an asynchronous function.
2. Inside `fetchIncome`, use a try-catch block to handle errors.
3. Inside the try block, fetch income data from your backend API. Replace `'/api/income'` with the actual API endpoint to fetch income data.
4. Use `await` to get the response and parse it as JSON.
5. Dispatch an action of type `'FETCH_INCOME_SUCCESS'` with the fetched data as the payload if the request is successful.
6. If there's an error, catch it and dispatch an action of type `'FETCH_INCOME_FAILURE'`.

solution

```
// actions.js
export const fetchIncome = () => async (dispatch) => {
  try {
    dispatch({ type: 'FETCH_DATA_LOADING' })
    const response = await fetch(
      'https://redux-example.tanaypratap.repl.co/income',
    )
    const data = await response.json()
    dispatch({ type: 'FETCH_INCOME_SUCCESS', payload: data })
  } catch (error) {
    console.error('Error fetching income data:', error)
    dispatch({ type: 'FETCH_INCOME_FAILURE' })
  }
}
```

COPY

ex02.2: create react component - income page

challenge

Create Income Component (`Income.js`):

- Create a new file named `Income.js` in your project's pages folder.
- Inside the `Income` component:
  - Use `useDispatch` to get the dispatch function.
  - Use `useSelector` to access the `income` data from the Redux store.
  - Calculate the `totalIncome` by reducing the `income` array.
  - Implement a `useEffect` to dispatch the `fetchIncome` action when the component mounts.
  - Render the income data as a list (e.g., a `<ul>` with `<li>` elements).
  - Display the `totalIncome` as a summary.

solution

```
function Income() {
  const dispatch = useDispatch()
  const income = useSelector((state) => state.income)

  const totalIncome = income.reduce((acc, value) => value.amount + acc, 0)

  useEffect(() => {
    dispatch(fetchIncome())
  }, [dispatch])

  return (
    <div>
      <h1>Income Page</h1>
      <ul>
        {income.map((transaction, index) => (
          <li key={index}>
            {transaction.description}: ${transaction.amount}
          </li>
        ))}
      </ul>
      <h2>Summary</h2>
      <div>Total Income: ${totalIncome}</div>
    </div>
  )
}
```

COPY

ex03: connect app component

challenge

In your main `App.js` component, wrap your components with the `Provider` and render them.

solution

```
import { StrictMode } from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'
import store from './store'
import App from './App'

const rootElement = document.getElementById('root')
ReactDOM.render(
  <StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </StrictMode>,
  rootElement,
)
```

COPY

ex04: create redux reducers - expenses & savings

challenge

1. Create `financeReducer` Function:
    - Define a reducer function named `financeReducer`
    - You should create cases for action types like `**FETCH_INCOME_SUCCESS**`, and `'FETCH_INCOME_FAILURE'` for expenses and savings.
    - `**'FETCH_EXPENSES_SUCCESS'**`, `**'FETCH_SAVINGS_SUCCESS'**`, and handle failures like `**'FETCH_INCOME_FAILURE'**`, `**'FETCH_EXPENSES_FAILURE'**`, `**FETCH_SAVINGS_FAILURE.\*\*`

solution

```
const financeReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'FETCH_EXPENSES_SUCCESS':
      return {
        ...state,
        expenses: action.payload,
        loading: false,
        error: null,
      }
    case 'FETCH_SAVINGS_SUCCESS':
      return {
        ...state,
        savings: action.payload,
        loading: false,
        error: null,
      }

    case 'FETCH_EXPENSES_FAILURE':
      return {
        ...state,
        loading: false,
        error: 'Error fetching expense data',
      }
    case 'FETCH_SAVINGS_FAILURE':
      return {
        ...state,
        loading: false,
        error: 'Error fetching savings data',
      }
    default:
      return state
  }
}

export default financeReducer
```

COPY

ex04.1: create redux actions for expenses & savings

challenge

- Create similar action creators for fetching expenses and savings data. You can copy the structure from the `fetchIncome` action.
- Use different action types
  like `'FETCH_EXPENSES_SUCCESS'`, `'FETCH_EXPENSES_FAILURE'`, `'FETCH_SAVINGS_SUCCESS'`, and `'FETCH_SAVINGS_FAILURE'`.

solution

```
export const fetchSavings = () => async (dispatch) => {
  try {
    dispatch({ type: 'FETCH_DATA_LOADING' })
    const response = await fetch(
      'https://redux-example.tanaypratap.repl.co/savings',
    )
    const data = await response.json()
    dispatch({ type: 'FETCH_SAVINGS_SUCCESS', payload: data })
  } catch (error) {
    console.error('Error fetching savings data:', error)
    dispatch({ type: 'FETCH_SAVINGS_FAILURE' })
  }
}

export const fetchExpenses = () => async (dispatch) => {
  try {
    dispatch({ type: 'FETCH_DATA_LOADING' })
    const response = await fetch(
      'https://redux-example.tanaypratap.repl.co/expenses',
    )
    const data = await response.json()
    console.log({ data })
    dispatch({ type: 'FETCH_EXPENSES_SUCCESS', payload: data })
  } catch (error) {
    console.error('Error fetching expense data:', error)
    dispatch({ type: 'FETCH_EXPENSES_FAILURE' })
  }
}
```

COPY

ex04.2: create react component - expense page

challenge

- Create a new file named `Expense.js`.
- Inside the `Expense` component:
  - Use `useDispatch` to get the dispatch function.
  - Use `useSelector` to access the `expenses` data from the Redux store.
  - Calculate the `totalExpenses` by reducing the `expenses` array.
  - Implement a `useEffect` to dispatch the `fetchExpenses` action when the component mounts.

- Render the expense data as a list (e.g., a `<ul>` with `<li>` elements).
- Display the `totalExpenses` as a summary.

solution

```
import React, { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { fetchExpenses } from '../actions'

function Expense() {
  const dispatch = useDispatch()
  const expenses = useSelector((state) => state.expenses)

  const totalExpenses = expenses.reduce((acc, value) => value.amount + acc, 0)

  useEffect(() => {
    dispatch(fetchExpenses())
  }, [dispatch])

  return (
    <div>
      <h1>Expense Page</h1>
      <ul>
        {expenses.map((transaction, index) => (
          <li key={index}>
            {transaction.description}: ${transaction.amount}
          </li>
        ))}
      </ul>
      <h2>Summary</h2>
      <div>Total Expenses: ${totalExpenses}</div>
    </div>
  )
}

export default Expense
```

COPY

ex04.3: create react component - savings page

challenge

- Create a new file named `Savings.js`
- Inside the `Savings` component:
  - Use `useDispatch` to get the dispatch function.
  - Use `useSelector` to access the `savings` data from the Redux store.
  - Calculate the `totalSavings` by reducing the `savings` array.
  - Implement a `useEffect` to dispatch the `fetchSavings` action when the component mounts.
  - Render the savings data as a list (e.g., a `<ul>` with `<li>` elements).
  - Display the `totalSavings` as a summary.

solution

```
import React, { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { fetchSavings } from '../actions'

function Savings() {
  const dispatch = useDispatch()
  const savings = useSelector((state) => state.savings)

  const totalSavings = savings.reduce((acc, value) => value.amount + acc, 0)

  useEffect(() => {
    dispatch(fetchSavings())
  }, [dispatch])

  return (
    <div>
      <h1>Savings Page</h1>
      <ul>
        {savings.map((transaction, index) => (
          <li key={index}>
            {transaction.description}: ${transaction.amount}
          </li>
        ))}
      </ul>
      <h2>Summary</h2>
      <div>Total Savings: ${totalSavings}</div>
    </div>
  )
}

export default Savings
```

COPY

ex05: create reducer cases

challenge

1. Create `financeReducer` Function:
    - Define a reducer function named `financeReducer`
    - You should create cases for action types `**ADD_ENTRY_FAILURE**` , `**ADD_INCOME_SUCCESS**` and `**ADD_EXPENSE_SUCCESS**`

solution

```
const financeReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'ADD_ENTRY_FAILURE':
      return {
        ...state,
        loading: false,
        error: 'Error fetching or adding data',
      }
```

```
    case 'ADD_INCOME_SUCCESS':
      return {
        ...state,
        income: [...state.income, action.payload],
        loading: false,
        error: null,
      }
    case 'ADD_EXPENSE_SUCCESS':
      return {
        ...state,
        expenses: [...state.expenses, action.payload],
        loading: false,
        error: null,
      }
    default:
      return state
  }
}

export default financeReducer
```

COPY

ex05.1: create redux actions to add entries

challenge

- Create an action creator function named `addEntry`. It will take an entry object as a parameter.
- Inside `addEntry`, send a POST request to your backend API to add a new entry. Make sure to set the appropriate headers and include the entry data in the request body as JSON.
- Use `await` to get the response and parse it as JSON.
- Check if the response indicates success (you can define your own success criteria).
- If the addition is successful, dispatch an action of type `'ADD_ENTRY_SUCCESS'` with the new entry data as the payload.
- If there's an error, catch it and dispatch an action of type `'ADD_ENTRY_FAILURE'`.

solution

```
export const addEntry = (entry) => async (dispatch) => {
  console.log(`https://redux-example.tanaypratap.repl.co/add-${entry.type}`)
  try {
    const response = await fetch(
      `https://redux-example.tanaypratap.repl.co/add-${entry.type}`,
      {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(entry),
      },
    )

    const data = await response.json()
```

```
    if (data.success === true) {
      if (entry.type === 'income') {
        dispatch({ type: 'ADD_INCOME_SUCCESS', payload: data.data })
      } else {
        dispatch({ type: 'ADD_EXPENSE_SUCCESS', payload: data.data })
      }
    }
  } catch (error) {
    console.error('Error adding entry:', error)
    dispatch({ type: 'ADD_ENTRY_FAILURE' })
  }
}
```

COPY

ex05.2: create react component - IncomeExpenseForm

challenge

- Create a new file named `IncomeExpenseForm.js`
- Inside the `IncomeExpenseForm` component:
  - Use `useState` to manage the state for `description`, `amount`, and `entryType`.
  - Implement a form with input fields for description, amount, and a dropdown/select for choosing the entry type (income or expense).
  - Handle form submissions by dispatching the `addEntry` action with the form data.
  - Clear the form fields after submission.

solution

```
import React, { useState } from 'react'
import { useDispatch } from 'react-redux'
import { addEntry } from '../actions'

function IncomeExpenseForm() {
  const dispatch = useDispatch()

  const [description, setDescription] = useState('')
  const [amount, setAmount] = useState('')
  const [entryType, setEntryType] = useState('income')

  const handleAddEntry = (e) => {
    e.preventDefault()

    dispatch(addEntry({ description, amount: parseFloat(amount), entryType }))
    setDescription('')
    setAmount('')
    setEntryType('income')
  }

  return (
    <div>
      <h1>New Entry Page</h1>
```

```
      <form>
        <div>
          <label>Description:</label>
          <input
            type='text'
            value={description}
            onChange={(e) => setDescription(e.target.value)}
          />
        </div>
        <div>
          <label>Amount:</label>
          <input
            type='number'
            value={amount}
            onChange={(e) => setAmount(e.target.value)}
          />
        </div>
        <div>
          <label>Entry Type:</label>
          <select
            value={entryType}
            onChange={(e) => setEntryType(e.target.value)}
          >
            <option value='income'>Income</option>
            <option value='expense'>Expense</option>
          </select>
        </div>
        <button onClick={handleAddEntry}>Add Entry</button>
      </form>
    </div>
  )
}

export default IncomeExpenseForm
```

COPY

ex05: set up react router

challenge

1. Install `react-router-dom` if you haven't already.
2. Configure routes for Income, Expense, Savings, and New Entry pages.
3. Use the components you created earlier as page components for each route.

solution

```
import IncomeExpenseForm from './pages/IncomeExpenseForm'
import { BrowserRouter as Router, Route, Link, Routes } from 'react-router-dom'
import Income from './pages/Income'
import Expense from './pages/Expense'
import Savings from './pages/Savings'

export default function App() {
  return (
```

```
    <div className='App'>
      <Router>
        <div>
          <nav>
            <ul>
              <li>
                <Link to='/income'>Income</Link>
              </li>
              <li>
                <Link to='/expenses'>Expense</Link>
              </li>
              <li>
                <Link to='/savings'>Savings</Link>
              </li>
              <li>
                <Link to='/'>New Entries</Link>
              </li>
            </ul>
          </nav>

          <Routes>
            <Route path='/income' element={<Income />} />
            <Route path='/expenses' element={<Expense />} />
            <Route path='/savings' element={<Savings />} />
            <Route path='/' element={<IncomeExpenseForm />} />
          </Routes>
        </div>
      </Router>
    </div>
  )
}
```

COPY

ex06: create a dashboard

challenge

Create a financial reports dashboard page where users can generate and view financial reports based on their income and expenses. Implement the following features:

1. Create a functional component `Dashboard` in the `Dashboard.js` file.
2. Inside the `Dashboard` component:
    - Set up state variables using the `useState` hook to manage the following:
        - `reportType`: A string to store the selected report type, initially set to an empty string.
        - `report`: An object with the following properties, initially set to default values:
            - `totalIncome`: 0
            - `totalExpenses`: 0
            - `savings`: 0

        - `expenseBreakdown`: An empty object.
    - Use the `useSelector` hook to get the `income` and `expenses` data from the Redux store.
    - Implement a function `generateReport`:
        - If the `reportType` is "incomeVsExpenses":
            - Calculate the `totalIncome` by reducing the `income` data.
            - Calculate the `totalExpenses` by reducing the `expenses` data.
            - Calculate `savings` as the difference between `totalIncome` and `totalExpenses`.
            - Update the `report` state with these values.
        - If the `reportType` is "expenseBreakdown":
            - Initialize an empty object `expenseBreakdown`.
            - Iterate through the `expenses` data and group expenses by their categories, summing up the amounts.
            - Update the `report` state with the `expenseBreakdown` object.
    - Create a dropdown menu (`<select>`) to allow the user to select the report type ("Income vs. Expenses" or "Expense Breakdown").
    - Create a button that, when clicked, calls the `generateReport` function to generate the selected report.
    - Display the generated report based on the `reportType`:
        - If the `reportType` is "incomeVsExpenses," display the total income, total expenses, and savings.
        - If the `reportType` is "expenseBreakdown," display an itemized list of expenses grouped by category.

solution

```
import React, { useState } from 'react'
import { useSelector } from 'react-redux'

function Dashboard() {
  const [reportType, setReportType] = useState('')
  const [report, setReport] = useState({
    totalIncome: 0,
    totalExpenses: 0,
    savings: 0,
    expenseBreakdown: {},
  })
  const income = useSelector((state) => state.income)
  const expenses = useSelector((state) => state.expenses)

  const generateReport = () => {
    if (reportType === 'incomeVsExpenses') {
      const totalIncome = income.reduce(
        (acc, transaction) => acc + transaction.amount,
        0,
      )
```

```
    const totalExpenses = expenses.reduce(
      (acc, transaction) => acc + transaction.amount,
      0,
    )

    const savings = totalIncome - totalExpenses
    setReport((oldReport) => ({
      ...oldReport,
      totalIncome,
      totalExpenses,
      savings,
    }))
  } else {
    const expenseBreakdown = {}
    expenses.forEach((transaction) => {
      const { category, amount } = transaction
      if (expenseBreakdown[category]) {
        expenseBreakdown[category] += amount
      } else {
        expenseBreakdown[category] = amount
      }
    })

    setReport((oldReport) => ({
      ...oldReport,
      expenseBreakdown,
    }))
  }
}

return (
  <div className='report'>
    <h2>Financial Reports</h2>
    <div>
      <label>Select Report Type:</label>
      <select
        value={reportType}
        onChange={(e) => setReportType(e.target.value)}
      >
        <option value=''>Select a report type</option>
        <option value='incomeVsExpenses'>Income vs. Expenses</option>
        <option value='expenseBreakdown'>Expense Breakdown</option>
      </select>
    </div>

    <button onClick={generateReport}>Generate Report</button>

    {report.totalIncome > 0 && reportType === 'incomeVsExpenses' && (
      <div>
        <h3> Report</h3>
        <div>
          <p>Total Income: ${report.totalIncome}</p>
          <p>Total Expenses: ${report.totalExpenses}</p>
          <p>Savings: ${report.savings}</p>
        </div>
      </div>
    )}

    {Object.keys(report.expenseBreakdown).length > 0 &&
      reportType === 'expenseBreakdown' && (
```

```
          <div>
            <h4>Expense Breakdown:</h4>
            <ul>
              {Object.keys(report.expenseBreakdown).map((category, index) => (
                <li key={index}>
                  {category}: ${report.expenseBreakdown[category]}
                </li>
              ))}
            </ul>
          </div>
        )}
    </div>
  )
}

export default Dashboard
```

COPY

entire solution

https://codesandbox.io/s/redux-without-toolkit-with-react-rx2-4-forked-h5pyz8

express app

https://replit.com/@tanaypratap/redux-example