E-Commerce Web App — AI-Assisted Development (Report-7)

Tools Used

- **ChatGPT** – Guided Redux setup for global state management; implemented product fetching (all products and single product details) through Redux Toolkit actions; assisted in debugging asynchronous dispatch logic and backend endpoint errors; helped structure the `Loader` and conditional rendering pattern.

- **Claude AI** – Suggested modular separation between reducers, constants, and actions; advised clean API endpoint handling and proper async error catching; helped refine pagination flow and category-based filtering logic.

- **GitHub Copilot** – Auto-completed repetitive Redux boilerplate (switch statements, action types, payload structures); assisted in mapping frontend states to backend API responses.

Project Overview

This phase focused on **integrating Redux** for centralized state management and ensuring smooth data flow between frontend and backend. Core progress includes:

- Setting up the **Redux store**, reducers, and actions for fetching all products and individual product details.

- Implementing **loading states** with a responsive loader component for better UX.

- Completing the **Product Details page** with dynamic routing and star ratings.

- Finalizing **pagination** on the Products page.

- Debugging frontend and backend inconsistencies, particularly with API endpoints and product count responses.

- Initiating **filtering and category selection** logic on the product listing page.

AI Interaction & Prompts

Redux Store & Reducers

- **ChatGPT** provided the structure for `configureStore` and `combineReducers`, integrating `productReducer` and `productDetailsReducer`.
- It suggested using `process.env.NODE_ENV` to enable Redux DevTools only in development mode.
- Assisted in separating action constants (`ALL_PRODUCT_REQUEST`, `ALL_PRODUCT_SUCCESS`, etc.) and reducers for cleaner scalability.

Redux Actions

- **ChatGPT** helped craft async `getProduct` and `getProductDetails` actions using `fetch` with error handling and conditional URL parameters.

Guided implementation of dynamic API routes for filtering:

```
/api/v1/pro/products?keyword=phone&page=2&price[gte]=0&price[lte]=25
000&category=Electronics
```

- Added `clearErrors` for resetting state without page reload.

Claude's Suggestions

- Recommended abstracting filters (price, category, ratings) into controlled state hooks.
- Advised implementing defensive parsing (`safeProductsCount` and `safeResultPerPage`) to prevent rendering issues.
- Suggested maintaining consistent naming conventions between backend payload (`productCount` → `productsCount`).

Copilot's Role

- Completed switch-case statements for reducers and action type imports.
- Auto-generated pagination structure and JSX mapping for product cards.
- Filled placeholder error-handling logic and loader return blocks.

My Implementation

- Implemented **Redux store** using `@reduxjs/toolkit` and modularized reducers (`productReducer`, `productDetailsReducer`).
- Built **actions** for fetching all products with filters (keyword, price, category, rating) and product details by ID.
- Created **Loader** component with simple rotating animation for global loading state.
- Integrated **ProductDetails** page using `useParams` and `react-simple-star-rating`.
- Built **Products** listing page with:
  - **Price slider filter** using MUI Slider.
  - **Category filter** with clickable category links.
  - **Ratings filter** and pagination integrated with Redux.
- Added **MetaData** component for dynamic document titles.
- Debugged backend–frontend mismatch (e.g., `/products` vs `/product` routes, resultPerPage edge cases).

Component Working

- **Redux Store** – Combines product list and details reducers for shared state access across pages.
- **Products Page** – Fetches product list with filters and renders paginated cards; uses sliders and category filters for dynamic updates.
- **Product Details Page** – Fetches individual product data from Redux store and displays details, images, ratings, and reviews.
- **Loader Component** – Displays animated loading spinner during API fetch operations.

Advantages

- Centralized state management simplifies data sharing across components.

- Responsive Loader improves UX during API calls.
- Dynamic filters (price, category, rating) ready for enhancement.
- Scalable Redux structure supports additional modules like cart and user authentication.
- AI-assisted coding reduced repetitive setup and debug time.

Limitations

- Filters and pagination need refinement for edge cases (e.g., empty category, missing products).
- Some backend responses require normalization (e.g., `productCount` → `productsCount`).
- Error handling can be improved with global toasts or fallback UI.
- Cart and review submission features still pending.

Reflection

- **ChatGPT** streamlined Redux setup, async actions, and error handling logic.
- **Claude AI** provided architectural guidance for filter state and reducer separation.
- **Copilot** accelerated repetitive Redux and JSX writing.
- Manual effort focused on debugging API integrations, pagination math, and UI consistency.
- The AI-assisted workflow enabled faster development while maintaining structure and modularity.