CANTiLEVER
bridging colleges to corporates

Project title

CUSTOMER SENTIMENT ANALYSIS FOR PRODUCT IMPROVEMENT

Team details

S.NO	Name	Roll no
1	P. Sujith	23R11AO5Y2
2	D. Romith Kumar	23R11A05V4
3	S. Anuroop	23R11A05D0

Abstract:

This project focuses on leveraging customer sentiment analysis to drive continuous product improvement by systematically analyzing diverse forms of user feedback. Utilizing advanced natural language processing (NLP) techniques and state-of-the-art transformer-based machine learning models, the system extracts sentiments, emotions, and recurring themes from sources such as online reviews, social media platforms, and customer support conversations. The models classify sentiments into positive, negative, or neutral categories, while also identifying specific product features or issues mentioned in the feedback.The methodology includes robust data preprocessing steps such as text normalization, language detection for multilingual inputs, and specialized handling of complex language patterns including sarcasm and ambiguity. A hybrid learning approach—combining supervised and unsupervised techniques—ensures the solution is scalable, adaptive, and effective across various product domains.A central feature of the project is the development of a dynamic sentiment analysis dashboard that visualizes sentiment trends, feature-specific feedback, and frequently occurring customer concerns. This enables product and marketing teams to prioritize actionable insights, align developments with user expectations, and track improvements over time. The system incorporates feedback loops to refine model accuracy continuously, adapting to evolving language patterns and emerging topics in customer discourse.Designed to be platform-agnostic, the solution allows seamless integration with existing Customer Relationship Management (CRM) systems and analytics platforms. It also addresses critical challenges such as data privacy, ethical use of consumer data, and bias in training datasets through strict compliance with data protection regulations and bias mitigation strategies.Ultimately, the project aims to bridge the gap between customer expectations and product performance by fostering a culture of data-driven decision-making. Expected outcomes include enhanced customer satisfaction, reduced churn rates, improved user experience, and a stronger product-market fit.

Project Overview and Introduction (Sentiment Shop)

Welcome to the comprehensive documentation for SentimentShop, an innovative e-commerce platform designed with a unique focus on leveraging customer feedback to drive product improvement. In today's competitive online retail landscape, understanding and responding to customer sentiment is paramount. SentimentShop goes beyond traditional e-commerce functionalities by integrating a sophisticated sentiment analysis engine directly into the shopping experience, creating a dynamic feedback loop between consumers and product developers.

This project, developed as an internship initiative, demonstrates the practical application of modern web technologies, database systems, and artificial intelligence/machine learning techniques. SentimentShop functions as a real-time online store where users can browse, search for, and purchase products. Crucially, it provides a robust platform for customers to share their experiences through reviews and suggestions, which are then processed to yield valuable insights for enhancing the product catalog.

The documentation that follows will provide a detailed exploration of SentimentShop, covering its architectural design, implementation details, core functionalities, and instructions for setup and deployment. This introductory section serves to provide a high-level overview of the platform, its purpose, key features, and the underlying technologies, setting the context for the deeper technical discussions presented in subsequent chapters.

Project Goals

The primary objective behind the SentimentShop project is twofold: to build a fully functional e-commerce platform and to integrate an intelligent system for analyzing customer sentiment towards the listed products. Specific goals include:

- Sentiment Analysis Integration: To implement a sentiment analysis module capable of processing customer reviews to determine the overall sentiment (positive, negative, neutral) and extract key themes or opinions.
- Product Improvement Facilitation: To generate actionable product improvement suggestions based on the analyzed customer feedback. This includes aggregating common complaints or requests highlighted in reviews.
- User-Friendly Shopping Experience: To provide a seamless and intuitive online shopping interface, incorporating essential e-commerce features such as product browsing, searching, filtering, and adding items to a cart.

- Customer Interaction Platform: To create a system where users can easily submit reviews and provide their own suggestions for product enhancements.
- Real-time Data Processing: To process reviews and update sentiment analysis results and suggestions in near real-time, providing up-to-date insights.
- Scalable Architecture: To design and implement a system architecture that can potentially handle a growing number of users, products, and reviews.

By achieving these goals, SentimentShop aims to be more than just a retail platform; it endeavors to be a catalyst for product evolution, driven directly by the voice of the customer.

Key Features

SentimentShop is equipped with a range of features designed to meet the demands of both online shoppers and product managers seeking feedback. The core functionalities include:

- User Authentication: A secure login and registration system allows users to create accounts, personalize their experience, and manage their activities on the platform.
- Product Catalog: A comprehensive catalog of products categorized for easy navigation. Products are displayed with essential information such as name, price (in Indian Rupees), and a rating out of 5.
- Browsing and Filtering: Users can browse products by category and apply filters to refine their search based on various criteria, making it easier to find desired items.
- Product Search: A robust search functionality allows users to find products by name or keywords. The system aims for exact matches and also displays related products to enhance discoverability.
- Product Details Pages: Clicking on a product reveals a detailed view including images, in-depth descriptions, and crucially, the customer review section.
- Customer Reviews: On product detail pages, users can view existing reviews submitted by other customers and are also provided with the option to write and submit their own reviews.
- Sentiment Analysis & Suggestions: Integrated into the product pages (or potentially in a backend/admin view), the system presents product improvement suggestions generated from the analysis of customer reviews. Users can also contribute their own specific suggestions.
- Add-to-Cart Functionality: Users can select products and add them to a persistent shopping cart before proceeding to checkout (checkout functionality is outside the primary scope of this internship project but the cart forms the basis).

- Dynamic Product Display: The platform is designed to display a large number of products on the home page, with features like infinite scrolling to allow users to easily explore the catalog.
- Visually Appealing Design: A significant emphasis has been placed on the platform's user interface, utilizing thoughtful designs and background colors to create an engaging and pleasant shopping environment.

These features collectively contribute to a dynamic and interactive platform that prioritizes customer engagement and feedback.

Technology Stack

The development of SentimentShop leverages a diverse set of technologies across different layers of the application. The core technologies employed include:

- Back-end and AI/ML: Python serves as the primary language for the back-end logic and, more importantly, for implementing the AI/ML algorithms required for sentiment analysis and suggestion generation. Frameworks likeorare suitable choices for building the web application's server-side components.
- Web Technologies:
  - Front-end: Standard web languages such asfor structure,for styling, andfor interactive elements are fundamental. A modern JavaScript framework like,, orcan be utilized to build a dynamic and responsive user interface.
  - Server-side (alternative/supplementary): While Python frameworks handle core logic, web servers and related technologies are also part of the stack.
- Database: A robust database system is essential for storing product information, user data, reviews, and sentiment analysis results. Both relational and NoSQL databases are viable options.
  - Relational Databases: SQL-based databases likeorare suitable for structured data.
  - NoSQL Databases: Document-based databases likemight be considered for flexibility in storing varying review structures or sentiment data.
- Other Potential Technologies: Depending on the specific implementation choices, technologies for task queues (e.g., Celery for background sentiment analysis processing), caching, search indexing (e.g., Elasticsearch), and deployment environments (e.g., Docker, cloud platforms) may also be part of the stack.

The combination of these technologies enables the platform to handle complex tasks such as real-time text processing, data persistence, and delivering a modern web experience.

Innovative Aspects

SentimentShop introduces several innovative elements that differentiate it from a standard e-commerce site:

- Real-time Sentiment Integration: The core innovation lies in the immediate or near-immediate analysis of newly submitted reviews, feeding insights directly into the platform's data layer which can then inform product views or administrative dashboards.
- Automated Suggestion Generation: Utilizing AI/ML to automatically generate product improvement suggestions based on analyzed sentiment provides a unique value proposition, directly translating customer feedback into actionable insights.
- Dynamic and Responsive Interface: The focus on a good design, intuitive navigation, and features like infinite scrolling aims to create a highly engaging user experience that feels modern and responsive.
- Feedback Loop Emphasis: The platform isn't just transactional; it actively encourages and utilizes customer feedback as a central component of its operation, creating a valuable feedback loop for vendors or platform administrators.

These aspects highlight SentimentShop's potential to not only facilitate online commerce but also to serve as an intelligent tool for product lifecycle management and customer relationship building.

This introduction has provided a foundational understanding of the SentimentShop project. The subsequent sections of this documentation will elaborate in detail on the system architecture, database design, front-end and back-end implementation specifics, sentiment analysis module design, deployment procedures, and guidelines for future enhancements.

System Architecture and Design

The architecture of the SentimentShop platform is designed to be modular, scalable, and maintainable, separating concerns into distinct layers and services. This approach facilitates development, testing, and future enhancements. The system primarily follows a client-server model with a clear separation between the front-end user interface, the back-end application logic and API, the persistent data storage (database), and a dedicated module for handling computationally intensive sentiment analysis tasks.

Understanding the interactions between these components is crucial to grasping how the SentimentShop processes user requests, manages data, and derives insights from customer feedback. Below is a description of the key components and their relationships.

High-Level Architecture Overview

At a high level, the SentimentShop architecture can be visualized as a multi-tiered system:

1. Client Tier (Front-end): This is the user interface layer that runs in the user's web browser. It's responsible for presenting information to the user, handling user input (clicks, typing), and communicating with the back-end API to fetch or send data.
2. Application Tier (Back-end API Server): This layer hosts the core business logic of the application. It receives requests from the front-end, processes them, interacts with the database, and orchestrates calls to other services like the sentiment analysis module. It exposes a set of API endpoints that the front-end consumes.
3. Data Tier (Database): This layer is responsible for persistent storage of all application data, including product information, user accounts, customer reviews, suggestions, and the results of sentiment analysis.
4. Sentiment Analysis Service/Module: A specialized component dedicated to processing text data (specifically customer reviews) to determine sentiment and extract relevant information for generating suggestions. This can be tightly integrated into the back-end or operate as a separate microservice.

Data flows primarily originate from the client, are processed by the back-end, which retrieves or stores information in the database, and the back-end interacts with the sentiment analysis module as needed. Results are then returned through the back-end to the client for display.

Component Deep Dive

Front-end (Client Interface)

The front-end of SentimentShop is the user's window into the platform. It provides the visual layout, handles navigation, displays product details, allows users to search, filter, add items to the cart, submit reviews, and contribute suggestions. The design prioritizes a user-friendly and visually appealing experience, as outlined in the project goals.

Chosen Framework: For building a dynamic and responsive single-page application (SPA) style front-end, a modern JavaScript framework is highly recommended.

```
React
```

has been chosen for this project.

Reasoning for Choice:

- Component-Based Architecture: React encourages building UIs out of small, reusable components. This is ideal for e-commerce sites where elements like product cards, review forms, shopping cart items, and filters appear repeatedly across different pages. This modularity improves development speed, maintainability, and code reusability.
- Virtual DOM: React uses a virtual DOM, which allows it to efficiently update and render UI changes by minimizing direct manipulation of the browser's DOM. This leads to better performance, especially in applications with frequent data updates like displaying product lists or refreshing review sections.
- Strong Community and Ecosystem: React has a vast ecosystem of libraries, tools, and a large community, providing ample support, pre-built components, and solutions for common development challenges (e.g., state management with Redux or Context API, routing with React Router).
- Declarative Syntax: React's declarative programming style makes the code more predictable and easier to debug. You describe what the UI should look like based on the current state, and React handles the updates efficiently.
- Suitability for Dynamic Interfaces: SentimentShop requires dynamic features like real-time search filtering, infinite scrolling, interactive forms for reviews, and potentially dynamic updates to sentiment data display. React is well-suited to handle these interactive elements effectively.

Key functionalities handled by the front-end include: rendering product lists based on category or search results, displaying detailed product information, presenting existing customer reviews and aggregated sentiment data, providing forms for user login/registration, review submission, and suggestion input, managing the state of the shopping cart (often synchronized with the back-end), and handling client-side routing to navigate between different views without full page reloads.

Back-end (API Server)

The back-end serves as the central hub of the SentimentShop application. It encapsulates the core business logic, interacts with the database, handles user authentication and authorization, processes user requests received from the front-end via API calls, and interfaces with the sentiment analysis module. It acts as an intermediary, ensuring data integrity and security.

Chosen Framework: Given the requirement for a robust, full-featured web application framework in Python (as Python is used for AI/ML),

`Django`

has been chosen for the back-end.

Reasoning for Choice:

- "Batteries Included": Django is a high-level Python web framework that encourages rapid development and pragmatic design. It includes many built-in features that are essential for an application like SentimentShop, such as an Object-Relational Mapper (ORM), an authentication system, URL routing, template engine (though less critical when building an API for a SPA), and a powerful admin panel.
- ORM (Object-Relational Mapping): Django's ORM simplifies database interactions significantly. Instead of writing raw SQL queries, developers interact with database tables using Python objects and methods, which makes data manipulation more intuitive and reduces the risk of SQL injection vulnerabilities. This is particularly useful for managing complex relationships between users, products, reviews, and sentiment data.
- Built-in Authentication System: Django provides a secure and comprehensive user authentication and authorization system out-of-the-box, handling user registration, login, permissions, and session management. This is a critical requirement for the SentimentShop platform.
- Admin Panel: Django automatically generates a professional, ready-to-use administrative interface based on the application's models. This is invaluable for managing products, categories, users, reviews, and potentially viewing sentiment analysis results without needing to build separate administrative tools.
- Scalability Features: While no framework guarantees scalability on its own, Django's design and the ecosystem of supporting libraries provide good foundations. It supports various database backends, caching mechanisms, and is compatible with horizontal scaling patterns.

Key functionalities handled by the back-end include: processing user registration and login, authenticating API requests, managing the product catalog (creating, reading, updating, deleting products and categories, though CUD might be admin-only), handling product search and filtering logic based on requests from the front-end, receiving and validating customer review and suggestion submissions, saving review and suggestion data to the database, triggering the sentiment analysis process (either directly or via a task queue), aggregating sentiment analysis results for display, managing shopping cart state (persisting cart items in the database), and providing structured data (typically in JSON format) to the front-end via RESTful API endpoints.

Database

The database is the backbone of SentimentShop, storing all persistent data required for the platform's operation. A well-designed database schema is essential for efficient data retrieval, storage, and management, especially considering the relationships between different entities like products, users, and reviews.

Chosen Database System: A robust relational database management system (RDBMS) is suitable for the structured nature of e-commerce data.

`PostgreSQL`

has been chosen for this project.

Reasoning for Choice:

- Reliability and Data Integrity: PostgreSQL is known for its strong adherence to ACID properties (Atomicity, Consistency, Isolation, Durability), ensuring data reliability and integrity, which is critical for financial transactions and user data in an e-commerce context.
- Feature Richness: PostgreSQL offers a wide range of advanced features, including support for various data types (including JSONB for semi-structured data if needed), indexing options, full-text search capabilities (useful for product search and review analysis), and extensibility.
- Support for Complex Queries and Relationships: Relational databases excel at managing structured data and complex relationships between tables (e.g., a product having many reviews, a user submitting many reviews). PostgreSQL handles joins and complex queries efficiently.
- Open Source and Community Support: Being open source, PostgreSQL is free to use and benefits from a large, active community that contributes to its development and provides support.
- Scalability: While vertical scaling is common, PostgreSQL also supports various methods for horizontal scaling and high availability through replication, clustering, and sharding techniques.

Database Schema Details:

The database schema is designed to store information about users, products, categories, customer reviews, product suggestions, and the results of sentiment analysis. Key tables include:

- users: Stores user account information.
  - id (Primary Key, Integer)
  - username (Unique, String)

    - password_hash (String)
    - email (Unique, String, Optional)
    - first_name (String, Optional)
    - last_name (String, Optional)
    - is_active (Boolean)
    - is_staff (Boolean)
    - date_joined (Timestamp)
    - ... (other fields provided by Django's auth model)
- categories: Stores product categories.
    - id (Primary Key, Integer)
    - name (Unique, String)
    - slug (Unique, String - for URLs)
    - description (Text, Optional)
- products: Stores product details.
    - id (Primary Key, Integer)
    - name (String)
    - slug (Unique, String - for URLs)
    - description (Text)
    - price_inr (Decimal)
    - category_id (Foreign Key to categories.id)
    - image_url (String - path or URL to product image)
    - average_rating (Decimal - Calculated, e.g., out of 5)
    - created_at (Timestamp)
    - updated_at (Timestamp)
    - stock (Integer, Optional)
    - is_available (Boolean)
- reviews: Stores customer reviews for products.
    - id (Primary Key, Integer)
    - product_id (Foreign Key to products.id)
    - user_id (Foreign Key to users.id)
    - rating (Integer - e.g., 1 to 5)
    - review_text (Text)
    - created_at (Timestamp)
    - updated_at (Timestamp)
- sentiment_results: Stores the results of sentiment analysis for reviews.
    - id (Primary Key, Integer)
    - review_id (Foreign Key to reviews.id, Unique - one sentiment result per review)

    - overall_sentiment (String - e.g., 'Positive', 'Negative', 'Neutral')
    - sentiment_score (Decimal, Optional - e.g., polarity score)
    - extracted_keywords (Text or Array/JSONB - key phrases/aspects)
    - analysis_timestamp (Timestamp)
- suggestions: Stores product improvement suggestions (can be user-submitted or system-generated).
    - id (Primary Key, Integer)
    - product_id (Foreign Key to products.id)
    - user_id (Foreign Key to users.id, Optional - if user-submitted)
    - suggestion_text (Text)
    - is_system_generated (Boolean - distinguish from user suggestions)
    - source_review_id (Foreign Key to reviews.id, Optional - if generated from a specific review)
    - created_at (Timestamp)
- cart_items: Stores items currently in users' shopping carts.
    - id (Primary Key, Integer)
    - user_id (Foreign Key to users.id)
    - product_id (Foreign Key to products.id)
    - quantity (Integer)
    - added_at (Timestamp)

Relationships are established using Foreign Keys, enforcing referential integrity. For instance, a review record must be linked to a specific product and user. The average_rating in the products table would typically be a calculated field or updated periodically based on new reviews to optimize read performance.

Sentiment Analysis Module/Service

This component is responsible for the core AI/ML task of the platform. When a new review is submitted, this module processes the review_text to determine its sentiment (positive, negative, neutral) and potentially identify common themes, features discussed, or specific issues raised. This process can be computationally intensive, especially if using complex machine learning models.

Integration Strategy: The sentiment analysis can be implemented within the Django back-end itself, perhaps using Python libraries like NLTK, SpaCy, TextBlob, or more sophisticated techniques involving machine learning models (e.g., trained on product review data). Alternatively, for scalability, it could be implemented as a separate microservice that the Django back-end communicates with (e.g., via REST API or message queue).

Triggering Analysis: Sentiment analysis should ideally be triggered asynchronously whenever a new review is successfully added to the database. This prevents the user submitting the review from waiting for the analysis to complete, which could take time. A task queue system (like Celery with a message broker like Redis or RabbitMQ) is a good choice for handling these background tasks.

Data Flow: The back-end receives a review from the front-end, saves it to the reviews table, and then sends the review text (and perhaps the review ID) to the Sentiment Analysis Module. The module processes the text and returns the results (sentiment label, score, keywords). The back-end then saves these results into the sentiment_results table, linking them to the original review. Aggregated sentiment for a product (e.g., overall positive percentage, average score) can be calculated and stored in the products table or generated on the fly or cached.

Suggestion Generation: The sentiment analysis module can also be responsible for generating product improvement suggestions. This could involve analyzing negative reviews to find common complaints or analyzing positive reviews for praised features. These system-generated suggestions can then be stored in the suggestions table.

Component Interaction and Data Flow Examples

Understanding how the components communicate is key. Here are a few common user flows:

1. User Browsing Products:

- User navigates to the homepage or a category page in the Front-end.
- The Front-end makes an API call to the Back-end (e.g., GET /api/products/ or GET /api/products/category//).
- The Back-end receives the request, uses the Django ORM to query the products and potentially categories tables in the Database.
- The Database returns the requested product data (name, price, image_url, average_rating, etc.).
- The Back-end formats the data into a JSON response and sends it back to the Front-end.
- The Front-end receives the JSON data and renders the product list on the page.

2. User Viewing Product Details and Reviews:

- User clicks on a product in the Front-end.
- The Front-end navigates to the product detail page and makes one or more API calls to the Back-end (e.g., GET /api/products//, GET /api/reviews/product//, GET /api/suggestions/product//, GET /api/sentiment/product//).

- The Back-end receives these requests. For product details, it queries the products table. For reviews, it queries the reviews table, joining with users to get reviewer names, and potentially joining with sentiment_results. For suggestions, it queries the suggestions table. For overall sentiment, it might query/calculate aggregated data from sentiment_results related to that product's reviews.
- The Database returns the relevant data.
- The Back-end formats the data into JSON responses and sends them to the Front-end.
- The Front-end receives the data and renders the product description, images, lists of reviews (with ratings and sentiment indicators), and product improvement suggestions.

3. User Submitting a Review:

- User fills out the review form (rating, text) on the product detail page in the Front-end.
- The Front-end sends a POST request with the review data (including product ID and user ID) to the Back-end API (e.g., POST /api/reviews/).
- The Back-end receives the request, validates the data (e.g., user is logged in, rating is valid), and saves the new review record to the reviews table in the Database using the Django ORM.
- After successfully saving the review, the Back-end (or a signal triggered by the save) sends the review text to the Sentiment Analysis Module for processing. This is ideally done asynchronously via a task queue.
- The Sentiment Analysis Module processes the text and returns the sentiment results (label, score, keywords).
- The Back-end receives the sentiment results and saves them to the sentiment_results table, linked to the new review ID. It might also trigger an update to the product's average_rating and generate/update suggestions.
- The Back-end sends a success response back to the Front-end.
- The Front-end updates the UI, potentially adding the new review to the list and refreshing the displayed sentiment/suggestions.

API Endpoints

The interaction between the Front-end and Back-end is facilitated through a set of RESTful API endpoints. These endpoints define how the client can request and manipulate data.

Key API Endpoint Examples:

- GET /api/categories/: Retrieve a list of all product categories.

- GET /api/products/: Retrieve a list of all products (can support pagination, sorting).
- GET /api/products/?category=: Retrieve products belonging to a specific category.
- GET /api/products/search/?q=: Search for products by name or keywords.
- GET /api/products//: Retrieve details for a single product. This endpoint might also embed related data or separate endpoints might be used for:
  - GET /api/products//reviews/: Retrieve reviews for a specific product.
  - GET /api/products//suggestions/: Retrieve suggestions for a specific product.
  - GET /api/products//sentiment/: Retrieve aggregated sentiment analysis results for a product's reviews.
- POST /api/auth/register/: Register a new user account.
- POST /api/auth/login/: Authenticate a user and provide an authentication token (e.g., JWT).
- GET /api/user/me/: Retrieve information about the currently authenticated user.
- POST /api/reviews/: Submit a new review for a product (requires authentication). Request body includes product_id, rating, review_text.
- POST /api/suggestions/: Submit a new user suggestion for a product (requires authentication). Request body includes product_id, suggestion_text.
- GET /api/cart/: Retrieve items in the current user's cart (requires authentication).
- POST /api/cart/items/: Add an item to the current user's cart (requires authentication). Request body includes product_id, quantity.
- DELETE /api/cart/items//: Remove an item from the cart (requires authentication).
- PUT /api/cart/items//: Update item quantity in the cart (requires authentication).

Data exchanged through these APIs is typically formatted as JSON (JavaScript Object Notation), a lightweight and widely compatible data interchange format.

Scalability and Performance Considerations

Designing for scalability and performance from the outset is crucial for handling potentially large numbers of users, products, and reviews. Sentiment analysis, in particular, can be resource-intensive.

Database Performance:

- Indexing: Proper indexing of frequently queried columns (e.g., product_id, user_id, category_id, creation timestamps) is essential to speed up read operations.

- Optimized Queries: Writing efficient SQL queries (handled by the ORM) and minimizing the number of database calls per request are important.
- Connection Pooling: Using a database connection pool helps manage database connections efficiently, reducing overhead.
- Read Replicas: As read traffic increases, setting up read-only replicas of the database can distribute the load.
- Caching: Caching frequently accessed data (e.g., popular products, category lists) in a fast in-memory store (like Redis or Memcached) reduces database load.
- Database Sharding/Partitioning: For extremely large datasets, splitting data across multiple database instances might be necessary in the future.

Back-end Performance and Scalability:

- Horizontal Scaling: Deploying multiple instances of the Django application server behind a load balancer allows the system to handle more concurrent requests.
- Caching: Implementing caching at the API layer (e.g., caching responses for product lists or popular items) reduces the need to hit the database or process logic for every request.
- Asynchronous Tasks: Offloading resource-intensive or time-consuming tasks, such as sending emails, processing image uploads, or critically, running sentiment analysis, to background worker processes using a task queue (like Celery) frees up the web server to handle more incoming requests quickly.
- Optimizing Code: Profiling and optimizing hot code paths in the Django application to reduce CPU usage and latency.
- Efficient Serialization: Ensuring data is serialized to JSON efficiently for API responses.

Sentiment Analysis Module Performance:

- Asynchronous Processing: As mentioned, running sentiment analysis asynchronously is vital.
- Dedicated Resources: The sentiment analysis module could be deployed on servers with more CPU or GPU resources if needed, especially if using deep learning models.
- Scaling the Analysis Service: If implemented as a separate service, this service can be scaled independently based on the volume of incoming reviews requiring analysis.
- Optimized Models: Using efficient algorithms and pre-trained models where possible can reduce processing time.

Front-end Performance:

- Code Splitting and Lazy Loading: Loading only the necessary JavaScript code for a specific page or component improves initial load time.
- Asset Optimization: Optimizing images and other static assets.
- State Management: Efficiently managing application state to minimize unnecessary re-renders.
- Client-Side Caching: Utilizing browser caching for static assets and potentially caching some API responses on the client.

Overall system performance is also monitored through metrics like request latency, error rates, and resource utilization. Implementing robust logging and monitoring helps identify bottlenecks and performance issues proactively.

Front-End Development

The front-end of the SentimentShop platform is the primary interface through which users interact with the application. It is responsible for rendering the user interface, handling user input, displaying product information, reviews, and suggestions, and facilitating navigation through the site. Built with a focus on usability, responsiveness, and visual appeal, the front-end provides a seamless and engaging online shopping experience. This section details the technologies, structure, key features, and design principles employed in developing the SentimentShop front-end.

Core Technologies and Framework

The SentimentShop front-end is built using standard, modern web development technologies, primarily leveraging a popular JavaScript framework to create a dynamic and maintainable application. The core technologies are:

- HTML (HyperText Markup Language): Provides the fundamental structure and content of the web pages. It is used semantically to define different sections of the interface, such as headers, navigation, product listings, forms, and footers. Within the chosen framework, HTML is often written using JSX (JavaScript XML), a syntax extension that allows writing HTML-like code directly within JavaScript, making component structure intuitive.
- CSS (Cascading Style Sheets): Controls the visual presentation and layout of the HTML elements. CSS is used extensively to implement the required designs, including background colors, typography, spacing, responsive layouts, and visual styling for interactive elements like buttons and forms. Methodologies like CSS Modules, styled-components, or BEM could be adopted within the framework context for better style organization and scope management.
- JavaScript: The core programming language that powers the dynamic and interactive aspects of the application. JavaScript handles client-side logic,

including responding to user events (clicks, scrolls, form submissions), fetching data from the back-end API, manipulating the DOM (indirectly via the framework), and managing application state.

- React: As established in the system architecture section, React was chosen as the primary JavaScript framework for building the user interface. React is a declarative, component-based library that simplifies the creation of complex, interactive UIs.

The choice of React is foundational to the front-end's architecture and development process. Its component-based paradigm encourages breaking down the UI into isolated, reusable pieces (e.g., a ProductCard component, a ReviewForm component, a CategoryFilter component). This modularity enhances code reusability, simplifies debugging, and makes the application easier to scale and maintain as features are added. React's use of a Virtual DOM optimizes rendering performance by minimizing direct manipulations of the browser's DOM, which is particularly beneficial for data-intensive views like product listings or live search results.

Application Structure and Pages

The SentimentShop front-end is structured as a Single Page Application (SPA), where navigation between different views typically happens without a full page reload, providing a faster and smoother user experience. The application is organized into various components and pages (views) corresponding to different functionalities. The main pages of the application include:

- Homepage: The initial landing page that provides an overview of the platform, often featuring a selection of products, categories, or promotional banners. It serves as a starting point for users to begin browsing or searching.
- Product Listing Page (Category/Browse View): Displays a grid or list of products belonging to a specific category or resulting from a general browse. It includes options for filtering and sorting products.
- Product Details Page: A dedicated view for a single product, showing comprehensive information including multiple images, detailed description, price, average rating, and sections for viewing existing customer reviews, submitting new reviews, and viewing/contributing product improvement suggestions.
- Search Results Page: Displays products that match the user's search query. This page needs to handle the specific requirements of showing exact matches prominently, followed by related products, and ensuring no empty displays even if exact matches are few.
- Login Page: A dedicated page for existing users to log in to their accounts.
- Registration Page: A page for new users to create a new account.
- Shopping Cart Page: Displays the list of items the user has added to their cart, allowing them to review quantities, remove items, and proceed towards checkout (though checkout itself might be outside the primary scope).
- User Profile/Dashboard (Optional but desirable): A page where authenticated users can view their order history (if implemented), manage their profile information, or view their past reviews/suggestions.

Each of these pages is typically represented by one or more high-level React components that compose smaller, reusable components. Routing between these pages is managed using a client-side routing library (like React Router), which maps URL paths to specific components.

User Interface Design and Wireframes

A significant emphasis was placed on creating a visually appealing and intuitive user interface. The design incorporates suitable background colors and layouts to enhance user experience and align with a modern e-commerce aesthetic. While actual images of wireframes or mockups cannot be displayed here, they serve as crucial planning tools during the design phase. The wireframes would typically illustrate:

- Layout Structure: The placement of key elements on each page, such as the header (including logo, search bar, navigation links, cart icon), sidebar (for categories and filters), main content area (for product listings, details), and footer.
- Content Hierarchy: How information is organized and presented to guide the user's eye and make essential details (like product names, prices, ratings) easily discoverable.
- Interactive Elements: The appearance and placement of buttons (e.g., "Add to Cart", "Submit Review"), forms, sliders, dropdowns, and other interactive components.
- Page Flows: How users navigate from one page to another (e.g., clicking a product on the homepage leads to the product details page, clicking a category filter updates the product listing).
- Responsive Breakpoints: How the layout adapts to different screen sizes (mobile, tablet, desktop).

For instance, a wireframe for the Homepage might show a prominent search bar at the top, a left sidebar listing categories, and a large central area displaying product cards in a grid. The Product Details page wireframe would show a main image area, a section for product title/price/rating, a description area, and dedicated expandable sections or tabs for Reviews and Suggestions below the main product info. These wireframes, transitioning into higher-fidelity mockups incorporating colors and specific styling, guide the front-end development process to ensure the final implementation matches the intended user experience.

Implementation of Key Features

Product Browsing with Categories and Filters

The primary way users discover products is through browsing. The homepage might display a curated selection or recently added items, while dedicated listing pages allow browsing by category.

Implementation details:

- Fetching Categories: Upon loading the application or relevant pages, the front-end makes an API call toto fetch the list of available product categories. This data is stored in the application's state.

```
GET /api/categories/
```

- Displaying Categories: The fetched categories are rendered, typically in a sidebar or a navigation menu, as interactive links or buttons.
- Fetching Products by Category/Browse: When a user clicks on a category or navigates to a general browsing page, the front-end triggers an API call to fetch the relevant products. For a specific category, this would be something like. For the initial homepage or general browse, it might bewith potential query parameters for pagination and initial sorting.

```
GET /api/products/

GET /api/products/?category=
```

- Rendering Product List: The fetched product data (which includes name, price, image URL, rating) is stored in the component's state. This state is then used to render a list or grid of ProductCard components. Each ProductCard displays essential information and is typically wrapped in a link that navigates to the Product Details page.

Filtering refines the product list based on attributes other than category. While the initial prompt didn't specify exact filters, common e-commerce filters include price range, brand, availability, or specific product attributes (e.g., size, color). SentimentShop could potentially add filters based on aggregated sentiment or rating ranges.

Implementation details for Filtering:

- Filter Controls: UI elements (checkboxes, sliders, dropdowns) are rendered based on available filter options (which might be fetched from the back-end or hardcoded if static).
- Applying Filters: When a user interacts with a filter control (e.g., selects a price range), the front-end updates the query parameters for the API call. For example, filtering by a price range might change the request to.

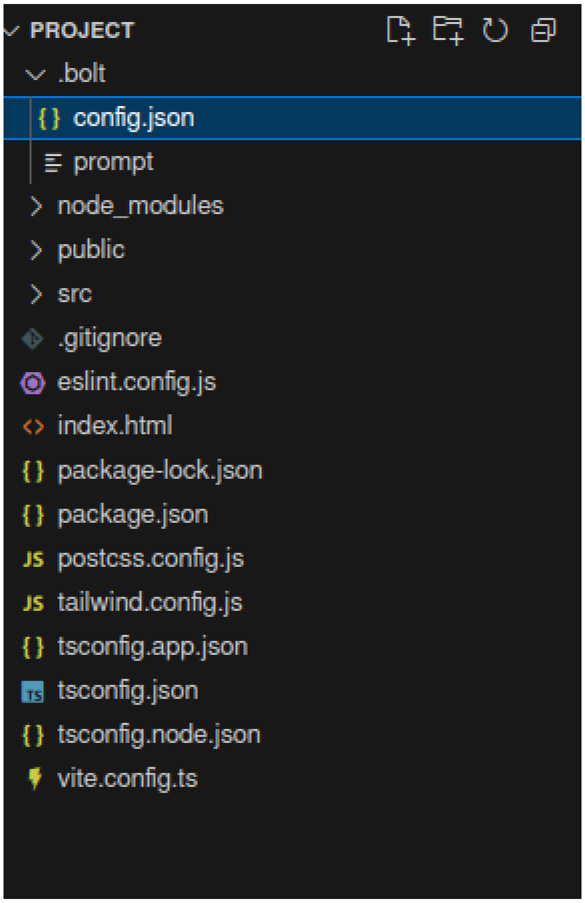
```
GET /api/products/?category=&min_price=1000&max_price=5000
```

- Re-fetching and Re-rendering: The front-end makes a new API call with the updated query parameters. Upon receiving the new, filtered data, the product list component's state is updated, triggering a re-render of the displayed products to show only those matching the filter criteria.

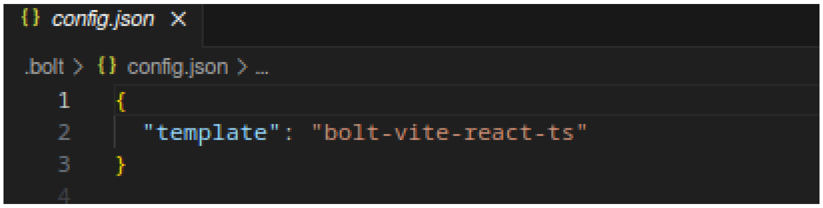
Product Search Functionality

The search functionality is a critical feature allowing users to quickly find specific items. The requirements include displaying exact matches prominently, showing related products, and ensuring the display is never empty if any products match the query, even loosely.

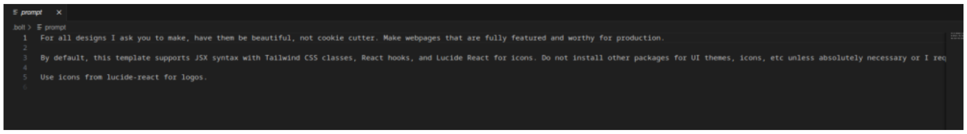
CODES



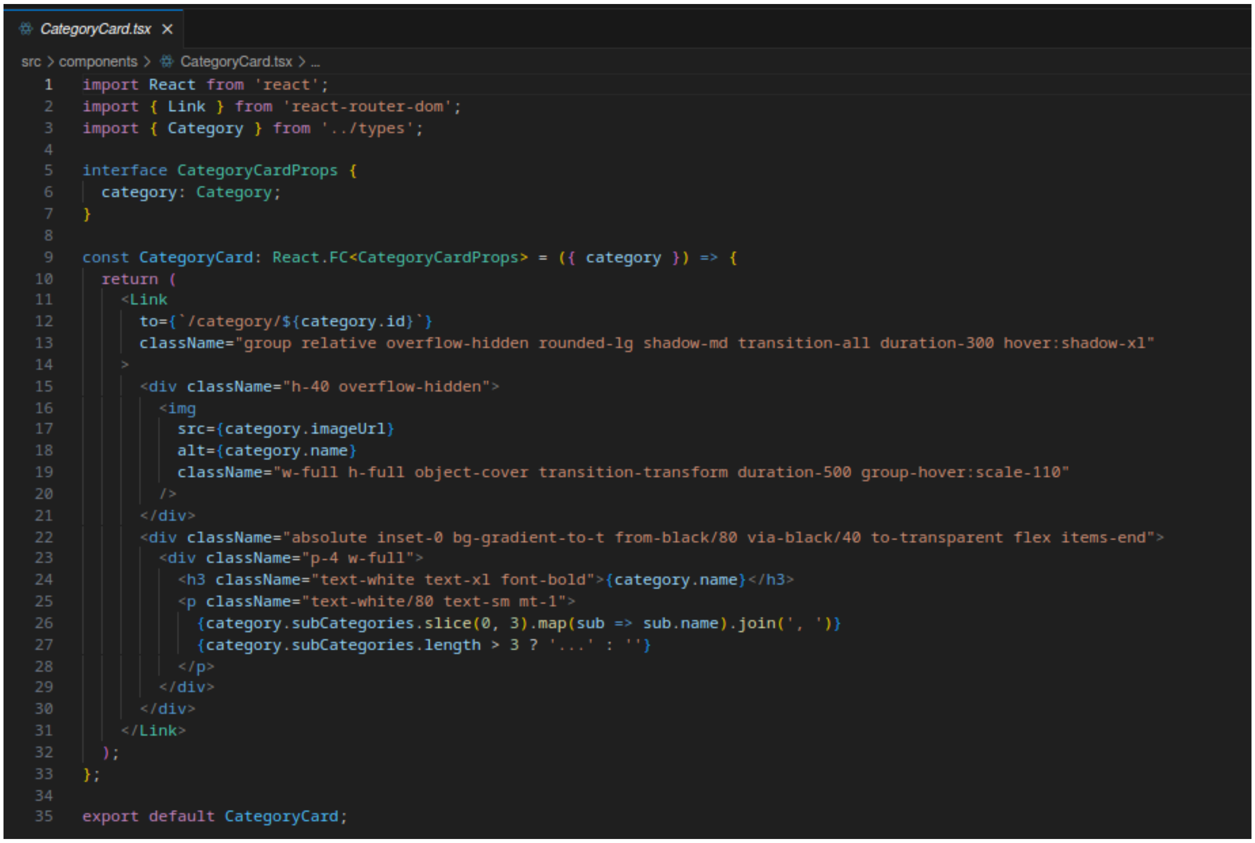
```
PROJECT
  .bolt
    config.json
    prompt
  node_modules
  public
  src
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  postcss.config.js
  tailwind.config.js
  tsconfig.app.json
  tsconfig.json
  tsconfig.node.json
  vite.config.ts
```



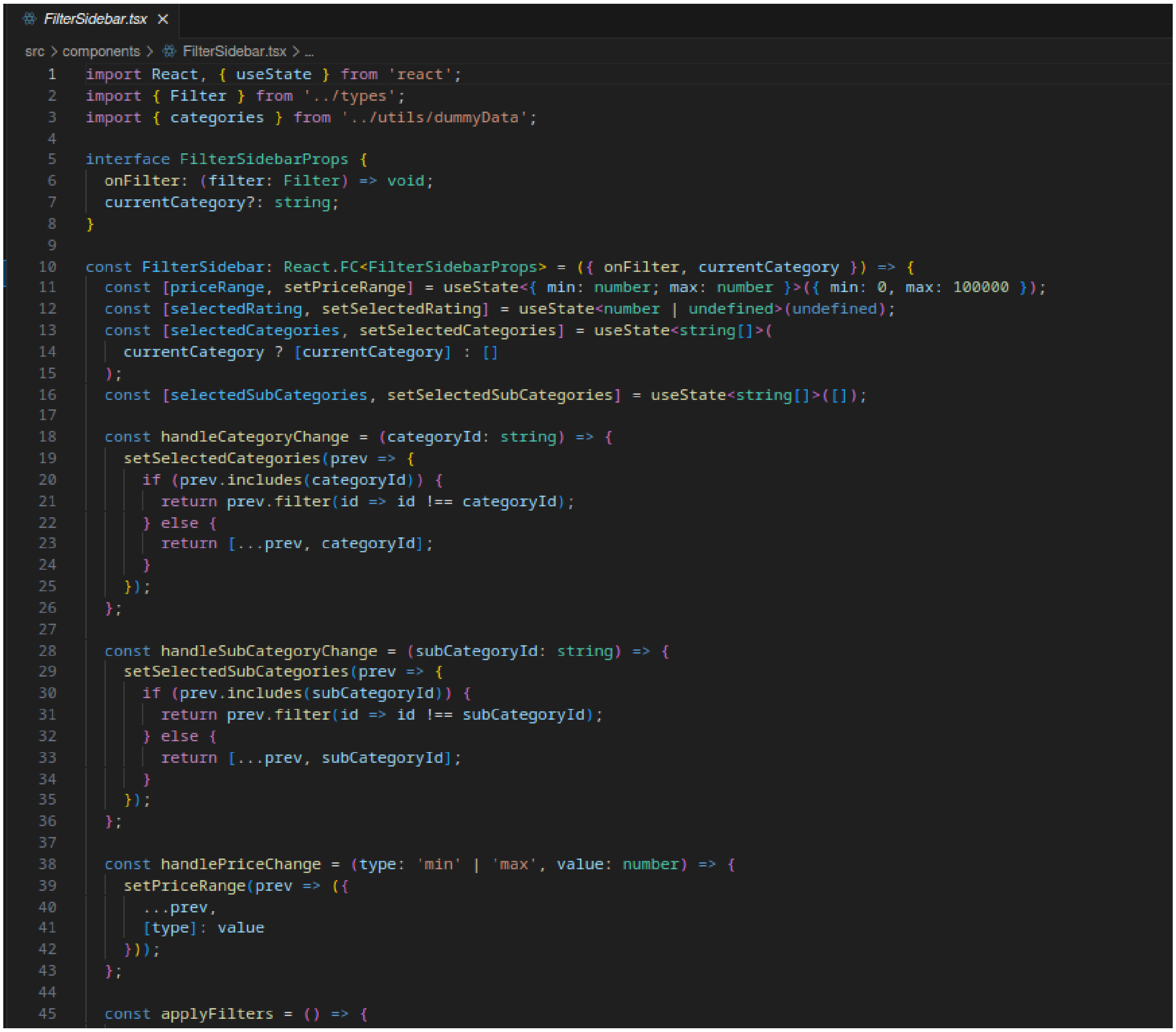
```json
{
  "template": "bolt-vite-react-ts"
}
```



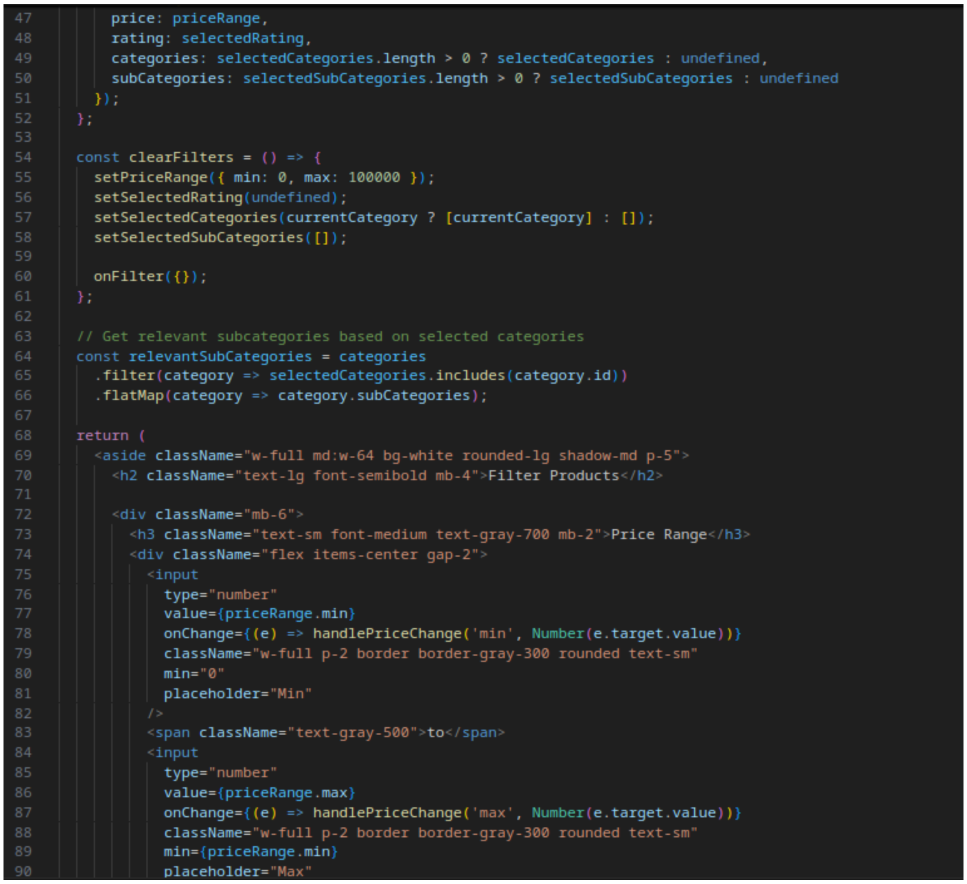
```
For all designs I ask you to make, have them be beautiful, not cookie cutter. Make webpages that are fully featured and worthy for production.

By default, this template supports JSX syntax with Tailwind CSS classes, React hooks, and Lucide React for icons. Do not install other packages for UI themes, icons, etc unless absolutely necessary or I req

Use icons from lucide-react for logos.
```



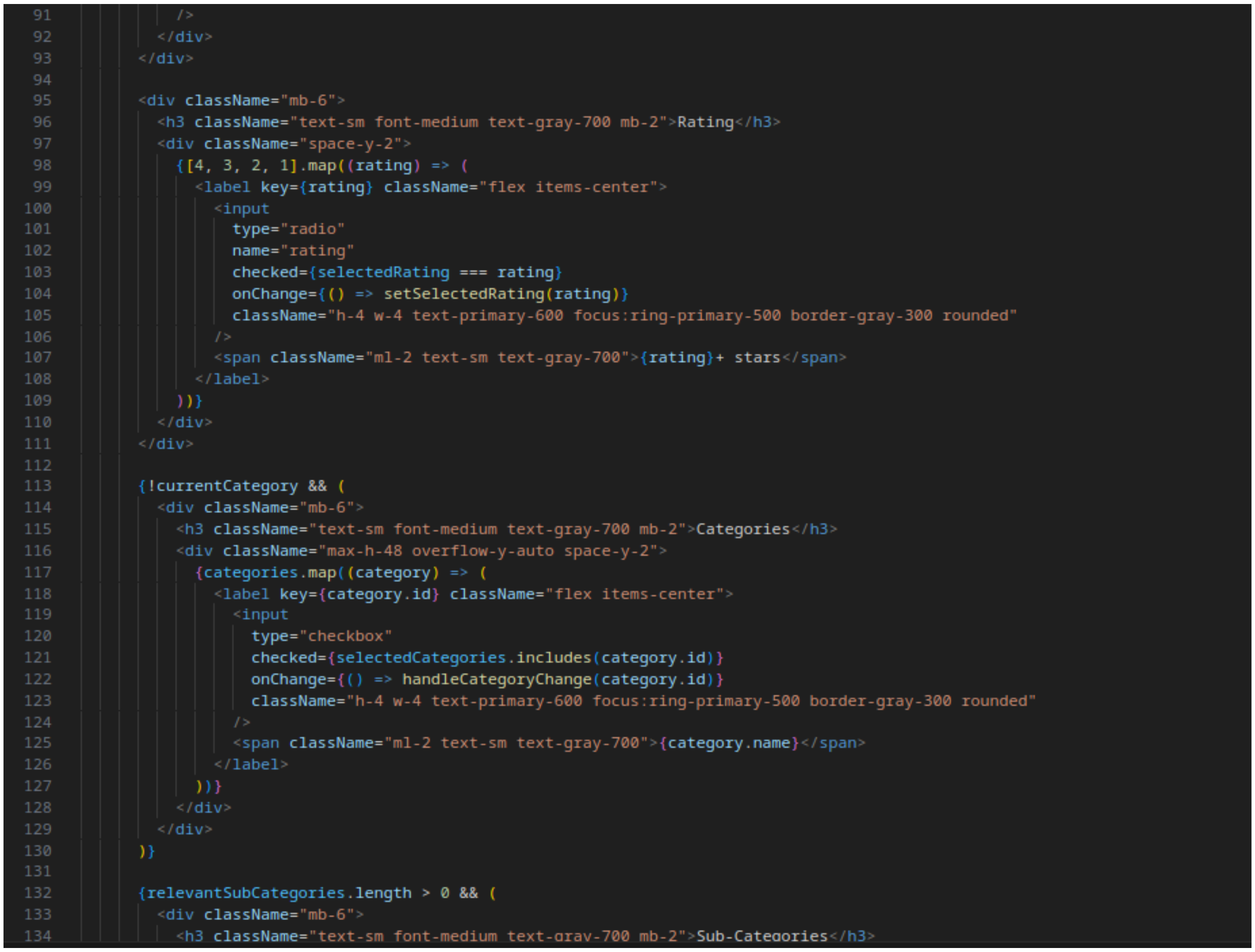
```tsx
import React from 'react';
import { Link } from 'react-router-dom';
import { Category } from '../types';

interface CategoryCardProps {
  category: Category;
}

const CategoryCard: React.FC<CategoryCardProps> = ({ category }) => {
  return (
    <Link
      to={`/category/${category.id}`}
      className="group relative overflow-hidden rounded-lg shadow-md transition-all duration-300 hover:shadow-xl"
    >
      <div className="h-40 overflow-hidden">
        <img
          src={category.imageUrl}
          alt={category.name}
          className="w-full h-full object-cover transition-transform duration-500 group-hover:scale-110"
        />
      </div>
      <div className="absolute inset-0 bg-gradient-to-t from-black/80 via-black/40 to-transparent flex items-end">
        <div className="p-4 w-full">
          <h3 className="text-white text-xl font-bold">{category.name}</h3>
          <p className="text-white/80 text-sm mt-1">
            {category.subCategories.slice(0, 3).map(sub => sub.name).join(', ')}
            {category.subCategories.length > 3 ? '...' : ''}
          </p>
        </div>
      </div>
    </Link>
  );
};

export default CategoryCard;
```



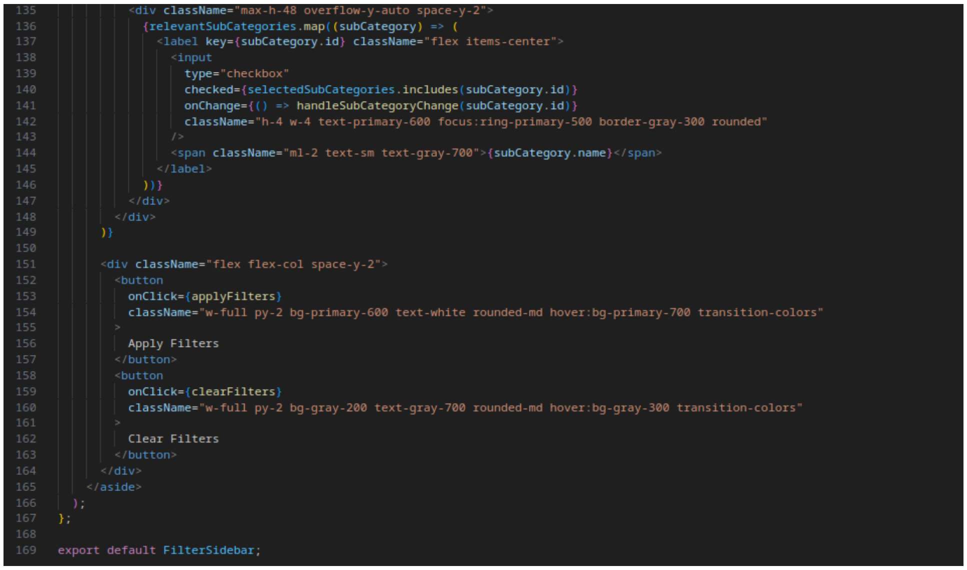
```tsx
import React, { useState } from 'react';
import { Filter } from '../types';
import { categories } from '../utils/dummyData';

interface FilterSidebarProps {
  onFilter: (filter: Filter) => void;
  currentCategory?: string;
}

const FilterSidebar: React.FC<FilterSidebarProps> = ({ onFilter, currentCategory }) => {
  const [priceRange, setPriceRange] = useState<{ min: number; max: number }>({ min: 0, max: 100000 });
  const [selectedRating, setSelectedRating] = useState<number | undefined>(undefined);
  const [selectedCategories, setSelectedCategories] = useState<string[]>(
    currentCategory ? [currentCategory] : []
  );
  const [selectedSubCategories, setSelectedSubCategories] = useState<string[]>([]);

  const handleCategoryChange = (categoryId: string) => {
    setSelectedCategories(prev => {
      if (prev.includes(categoryId)) {
        return prev.filter(id => id !== categoryId);
      } else {
        return [...prev, categoryId];
      }
    });
  };

  const handleSubCategoryChange = (subCategoryId: string) => {
    setSelectedSubCategories(prev => {
      if (prev.includes(subCategoryId)) {
        return prev.filter(id => id !== subCategoryId);
      } else {
        return [...prev, subCategoryId];
      }
    });
  };

  const handlePriceChange = (type: 'min' | 'max', value: number) => {
    setPriceRange(prev => ({
      ...prev,
      [type]: value
    }));
  };

  const applyFilters = () => {
```



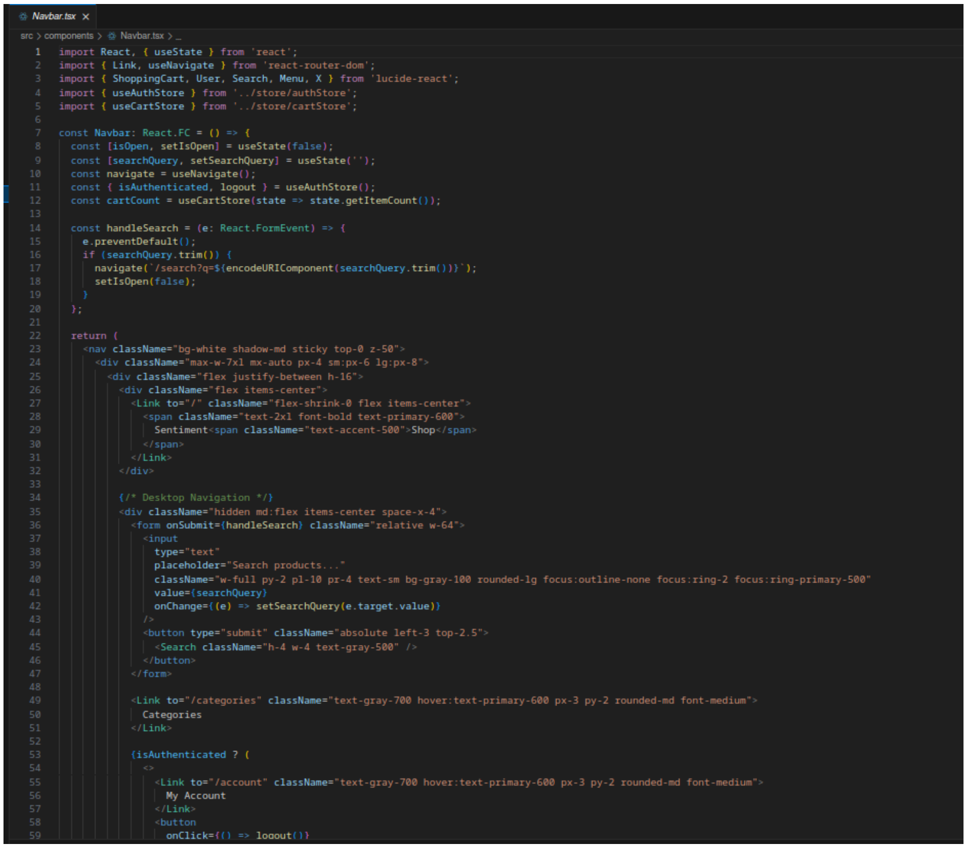
```
      price: priceRange,
      rating: selectedRating,
      categories: selectedCategories.length > 0 ? selectedCategories : undefined,
      subCategories: selectedSubCategories.length > 0 ? selectedSubCategories : undefined
    });
  };

  const clearFilters = () => {
    setPriceRange({ min: 0, max: 100000 });
    setSelectedRating(undefined);
    setSelectedCategories(currentCategory ? [currentCategory] : []);
    setSelectedSubCategories([]);

    onFilter({});
  };

  // Get relevant subcategories based on selected categories
  const relevantSubCategories = categories
    .filter(category => selectedCategories.includes(category.id))
    .flatMap(category => category.subCategories);

  return (
    <aside className="w-full md:w-64 bg-white rounded-lg shadow-md p-5">
      <h2 className="text-lg font-semibold mb-4">Filter Products</h2>

      <div className="mb-6">
        <h3 className="text-sm font-medium text-gray-700 mb-2">Price Range</h3>
        <div className="flex items-center gap-2">
          <input
            type="number"
            value={priceRange.min}
            onChange={(e) => handlePriceChange('min', Number(e.target.value))}
            className="w-full p-2 border border-gray-300 rounded text-sm"
            min="0"
            placeholder="Min"
          />
          <span className="text-gray-500">to</span>
          <input
            type="number"
            value={priceRange.max}
            onChange={(e) => handlePriceChange('max', Number(e.target.value))}
            className="w-full p-2 border border-gray-300 rounded text-sm"
            min={priceRange.min}
            placeholder="Max"
```



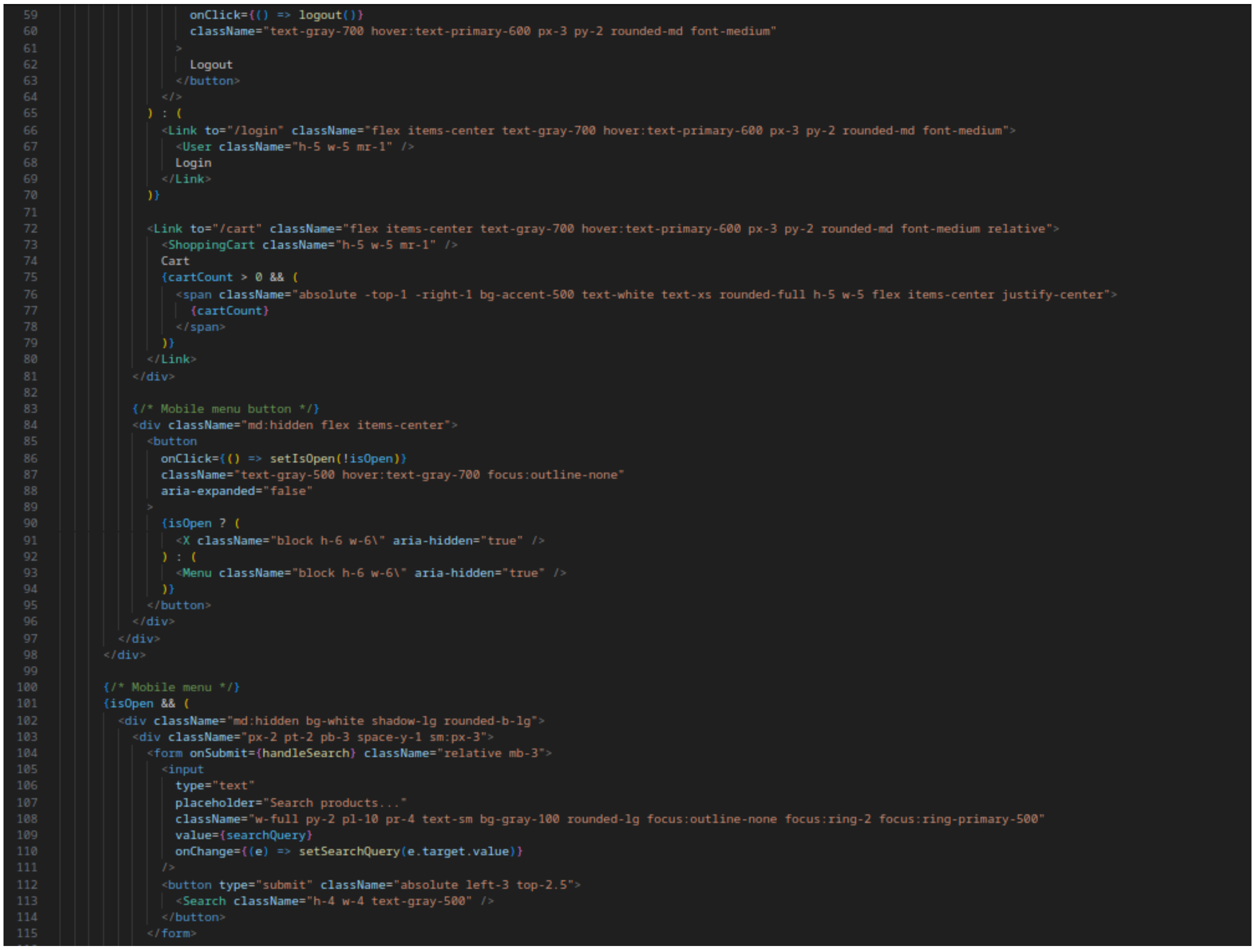
```
          />
        </div>
      </div>

      <div className="mb-6">
        <h3 className="text-sm font-medium text-gray-700 mb-2">Rating</h3>
        <div className="space-y-2">
          {[4, 3, 2, 1].map((rating) => (
            <label key={rating} className="flex items-center">
              <input
                type="radio"
                name="rating"
                checked={selectedRating === rating}
                onChange={() => setSelectedRating(rating)}
                className="h-4 w-4 text-primary-600 focus:ring-primary-500 border-gray-300 rounded"
              />
              <span className="ml-2 text-sm text-gray-700">{rating}+ stars</span>
            </label>
          ))}
        </div>
      </div>

      {!currentCategory && (
        <div className="mb-6">
          <h3 className="text-sm font-medium text-gray-700 mb-2">Categories</h3>
          <div className="max-h-48 overflow-y-auto space-y-2">
            {categories.map((category) => (
              <label key={category.id} className="flex items-center">
                <input
                  type="checkbox"
                  checked={selectedCategories.includes(category.id)}
                  onChange={() => handleCategoryChange(category.id)}
                  className="h-4 w-4 text-primary-600 focus:ring-primary-500 border-gray-300 rounded"
                />
                <span className="ml-2 text-sm text-gray-700">{category.name}</span>
              </label>
            ))}
          </div>
        </div>
      )}

      {relevantSubCategories.length > 0 && (
        <div className="mb-6">
          <h3 className="text-sm font-medium text-gray-700 mb-2">Sub-Categories</h3>
```



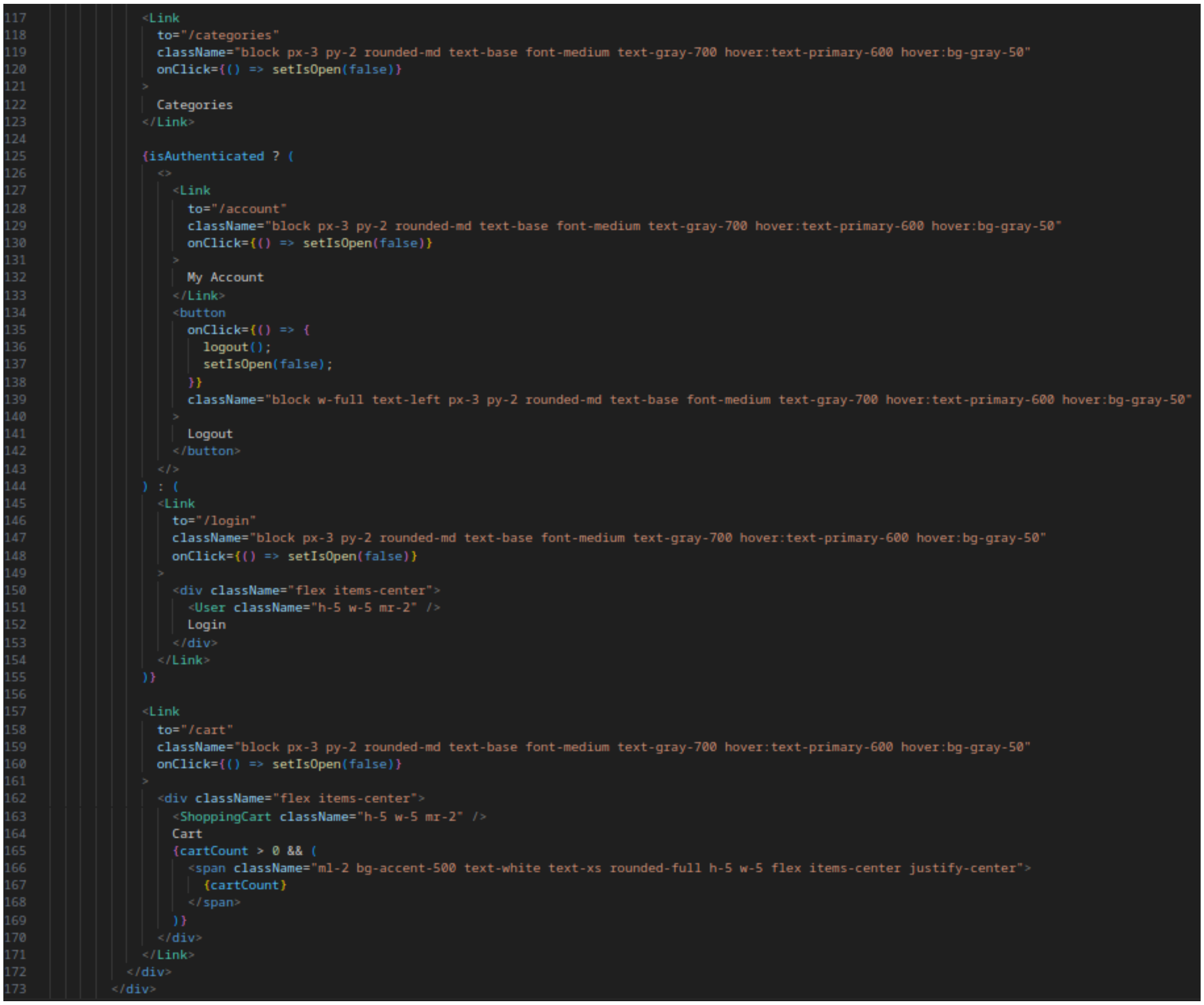
```
          <div className="max-h-48 overflow-y-auto space-y-2">
            {relevantSubCategories.map((subCategory) => (
              <label key={subCategory.id} className="flex items-center">
                <input
                  type="checkbox"
                  checked={selectedSubCategories.includes(subCategory.id)}
                  onChange={() => handleSubCategoryChange(subCategory.id)}
                  className="h-4 w-4 text-primary-600 focus:ring-primary-500 border-gray-300 rounded"
                />
                <span className="ml-2 text-sm text-gray-700">{subCategory.name}</span>
              </label>
            ))}
          </div>
        </div>
      )}

      <div className="flex flex-col space-y-2">
        <button
          onClick={applyFilters}
          className="w-full py-2 bg-primary-600 text-white rounded-md hover:bg-primary-700 transition-colors"
        >
          Apply Filters
        </button>
        <button
          onClick={clearFilters}
          className="w-full py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300 transition-colors"
        >
          Clear Filters
        </button>
      </div>
    </aside>
  );
};

export default FilterSidebar;
```



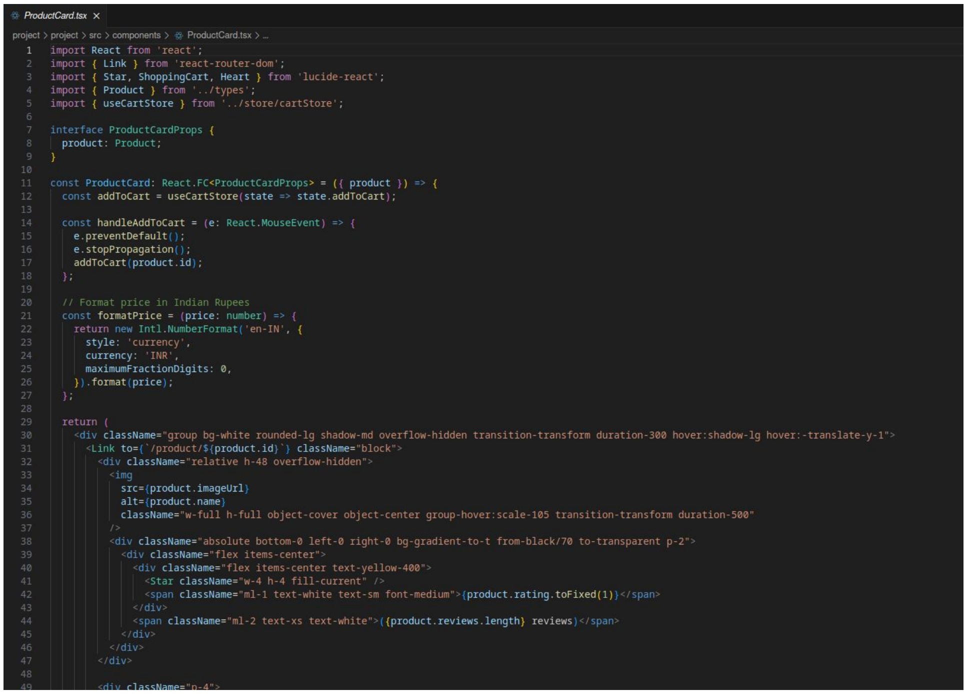
```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { ShoppingCart, User, Search, Menu, X } from 'lucide-react';
import { useAuthStore } from '../store/authStore';
import { useCartStore } from '../store/cartStore';

const Navbar: React.FC = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [searchQuery, setSearchQuery] = useState('');
  const navigate = useNavigate();
  const { isAuthenticated, logout } = useAuthStore();
  const cartCount = useCartStore(state => state.getItemCount());

  const handleSearch = (e: React.FormEvent) => {
    e.preventDefault();
    if (searchQuery.trim()) {
      navigate(`/search?q=${encodeURIComponent(searchQuery.trim())}`);
      setIsOpen(false);
    }
  };

  return (
    <nav className="bg-white shadow-md sticky top-0 z-50">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="flex justify-between h-16">
          <div className="flex items-center">
            <Link to="/" className="flex-shrink-0 flex items-center">
              <span className="text-2xl font-bold text-primary-600">
                Sentiment<span className="text-accent-500">Shop</span>
              </span>
            </Link>
          </div>

          {/* Desktop Navigation */}
          <div className="hidden md:flex items-center space-x-4">
            <form onSubmit={handleSearch} className="relative w-64">
              <input
                type="text"
                placeholder="Search products..."
                className="w-full py-2 pl-10 pr-4 text-sm bg-gray-100 rounded-lg focus:outline-none focus:ring-2 focus:ring-primary-500"
                value={searchQuery}
                onChange={(e) => setSearchQuery(e.target.value)}
              />
              <button type="submit" className="absolute left-3 top-2.5">
                <Search className="h-4 w-4 text-gray-500" />
              </button>
            </form>

            <Link to="/categories" className="text-gray-700 hover:text-primary-600 px-3 py-2 rounded-md font-medium">
              Categories
            </Link>

            {isAuthenticated ? (
              <>
                <Link to="/account" className="text-gray-700 hover:text-primary-600 px-3 py-2 rounded-md font-medium">
                  My Account
                </Link>
                <button
                  onClick={() => logout()}
```



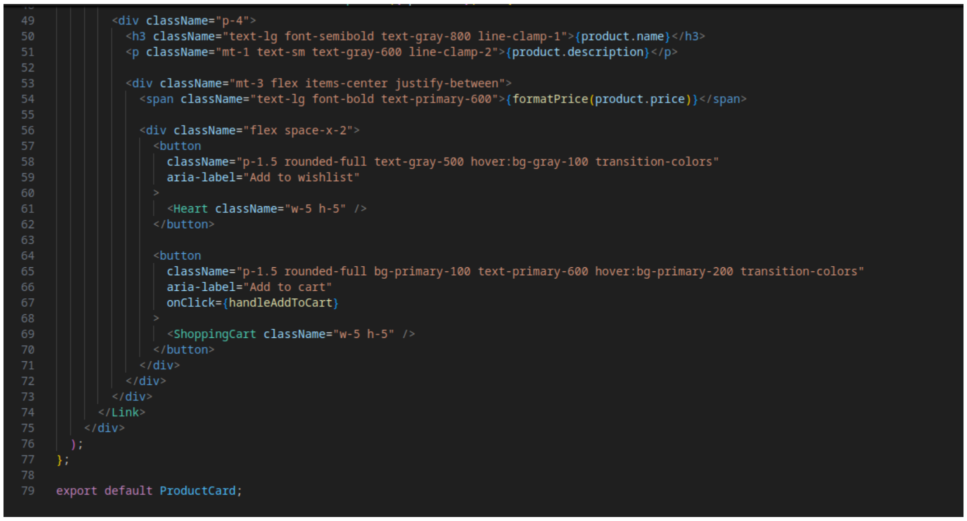
```
                    onClick={() => logout()}
                    className="text-gray-700 hover:text-primary-600 px-3 py-2 rounded-md font-medium"
                  >
                    Logout
                  </button>
                </>
              ) : (
                <Link to="/login" className="flex items-center text-gray-700 hover:text-primary-600 px-3 py-2 rounded-md font-medium">
                  <User className="h-5 w-5 mr-1" />
                  Login
                </Link>
              )}

              <Link to="/cart" className="flex items-center text-gray-700 hover:text-primary-600 px-3 py-2 rounded-md font-medium relative">
                <ShoppingCart className="h-5 w-5 mr-1" />
                Cart
                {cartCount > 0 && (
                  <span className="absolute -top-1 -right-1 bg-accent-500 text-white text-xs rounded-full h-5 w-5 flex items-center justify-center">
                    {cartCount}
                  </span>
                )}
              </Link>
            </div>

            {/* Mobile menu button */}
            <div className="md:hidden flex items-center">
              <button
                onClick={() => setIsOpen(!isOpen)}
                className="text-gray-500 hover:text-gray-700 focus:outline-none"
                aria-expanded="false"
              >
                {isOpen ? (
                  <X className="block h-6 w-6\" aria-hidden="true" />
                ) : (
                  <Menu className="block h-6 w-6\" aria-hidden="true" />
                )}
              </button>
            </div>
          </div>
        </div>

        {/* Mobile menu */}
        {isOpen && (
          <div className="md:hidden bg-white shadow-lg rounded-b-lg">
            <div className="px-2 pt-2 pb-3 space-y-1 sm:px-3">
              <form onSubmit={handleSearch} className="relative mb-3">
                <input
                  type="text"
                  placeholder="Search products..."
                  className="w-full py-2 pl-10 pr-4 text-sm bg-gray-100 rounded-lg focus:outline-none focus:ring-2 focus:ring-primary-500"
                  value={searchQuery}
                  onChange={(e) => setSearchQuery(e.target.value)}
                />
                <button type="submit" className="absolute left-3 top-2.5">
                  <Search className="h-4 w-4 text-gray-500" />
                </button>
              </form>
```



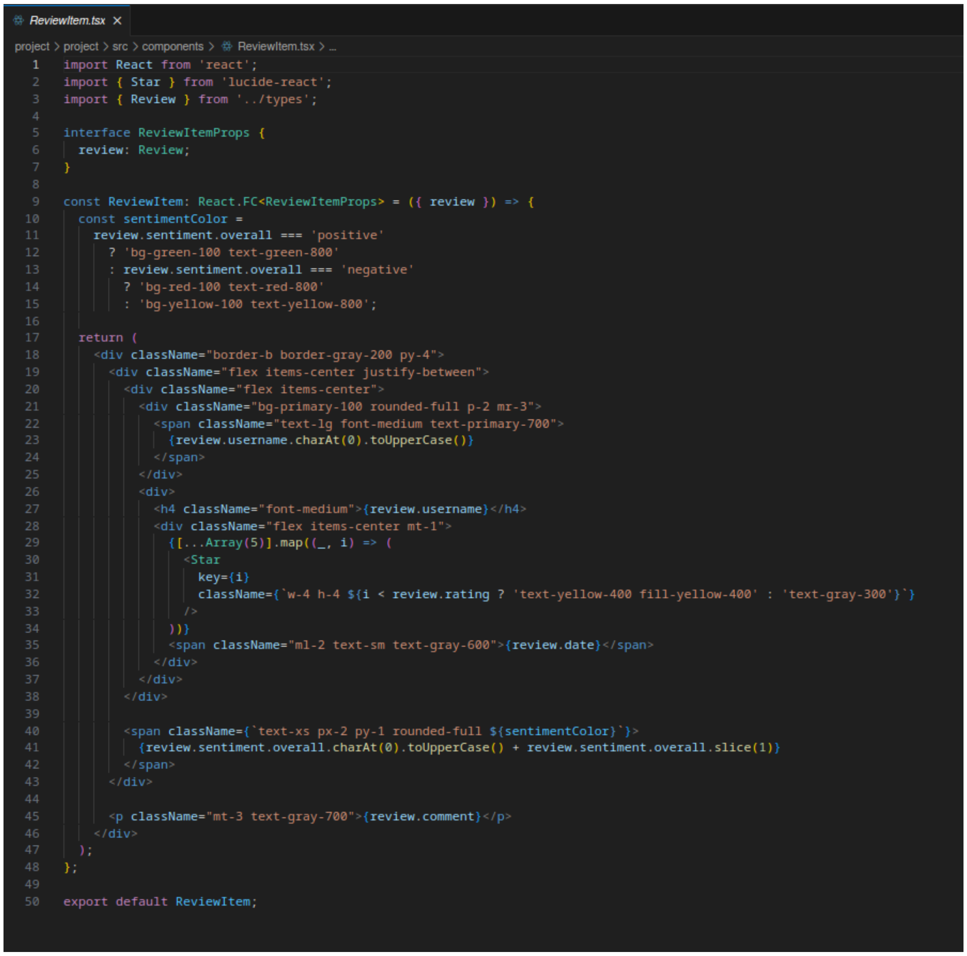
```
              <Link
                to="/categories"
                className="block px-3 py-2 rounded-md text-base font-medium text-gray-700 hover:text-primary-600 hover:bg-gray-50"
                onClick={() => setIsOpen(false)}
              >
                Categories
              </Link>

              {isAuthenticated ? (
                <>
                  <Link
                    to="/account"
                    className="block px-3 py-2 rounded-md text-base font-medium text-gray-700 hover:text-primary-600 hover:bg-gray-50"
                    onClick={() => setIsOpen(false)}
                  >
                    My Account
                  </Link>
                  <button
                    onClick={() => {
                      logout();
                      setIsOpen(false);
                    }}
                    className="block w-full text-left px-3 py-2 rounded-md text-base font-medium text-gray-700 hover:text-primary-600 hover:bg-gray-50"
                  >
                    Logout
                  </button>
                </>
              ) : (
                <Link
                  to="/login"
                  className="block px-3 py-2 rounded-md text-base font-medium text-gray-700 hover:text-primary-600 hover:bg-gray-50"
                  onClick={() => setIsOpen(false)}
                >
                  <div className="flex items-center">
                    <User className="h-5 w-5 mr-2" />
                    Login
                  </div>
                </Link>
              )}

              <Link
                to="/cart"
                className="block px-3 py-2 rounded-md text-base font-medium text-gray-700 hover:text-primary-600 hover:bg-gray-50"
                onClick={() => setIsOpen(false)}
              >
                <div className="flex items-center">
                  <ShoppingCart className="h-5 w-5 mr-2" />
                  Cart
                  {cartCount > 0 && (
                    <span className="ml-2 bg-accent-500 text-white text-xs rounded-full h-5 w-5 flex items-center justify-center">
                      {cartCount}
                    </span>
                  )}
                </div>
              </Link>
            </div>
          </div>
```


ProductCard.tsx ×
project > project > src > components > ProductCard.tsx > ...

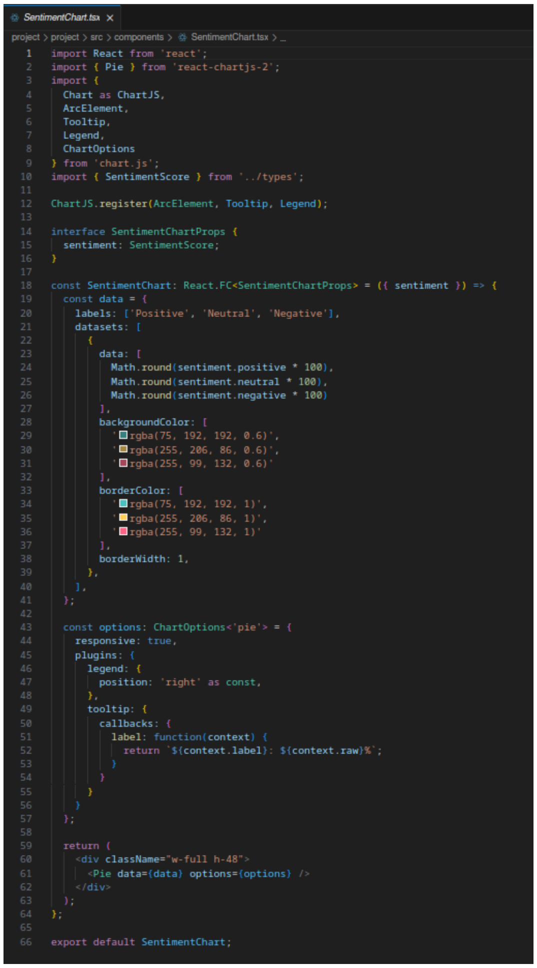
```tsx
import React from 'react';
import { Link } from 'react-router-dom';
import { Star, ShoppingCart, Heart } from 'lucide-react';
import { Product } from '../types';
import { useCartStore } from '../store/cartStore';

interface ProductCardProps {
  product: Product;
}

const ProductCard: React.FC<ProductCardProps> = ({ product }) => {
  const addToCart = useCartStore(state => state.addToCart);

  const handleAddToCart = (e: React.MouseEvent) => {
    e.preventDefault();
    e.stopPropagation();
    addToCart(product.id);
  };

  // Format price in Indian Rupees
  const formatPrice = (price: number) => {
    return new Intl.NumberFormat('en-IN', {
      style: 'currency',
      currency: 'INR',
      maximumFractionDigits: 0,
    }).format(price);
  };

  return (
    <div className="group bg-white rounded-lg shadow-md overflow-hidden transition-transform duration-300 hover:shadow-lg hover:-translate-y-1">
      <Link to={`/product/${product.id}`} className="block">
        <div className="relative h-48 overflow-hidden">
          <img
            src={product.imageUrl}
            alt={product.name}
            className="w-full h-full object-cover object-center group-hover:scale-105 transition-transform duration-500"
          />
          <div className="absolute bottom-0 left-0 right-0 bg-gradient-to-t from-black/70 to-transparent p-2">
            <div className="flex items-center">
              <div className="flex items-center text-yellow-400">
                <Star className="w-4 h-4 fill-current" />
                <span className="ml-1 text-white text-sm font-medium">{product.rating.toFixed(1)}</span>
              </div>
              <span className="ml-2 text-xs text-white">({product.reviews.length} reviews)</span>
            </div>
          </div>
        </div>

```



```tsx
        <div className="p-4">
          <h3 className="text-lg font-semibold text-gray-800 line-clamp-1">{product.name}</h3>
          <p className="mt-1 text-sm text-gray-600 line-clamp-2">{product.description}</p>

          <div className="mt-3 flex items-center justify-between">
            <span className="text-lg font-bold text-primary-600">{formatPrice(product.price)}</span>

            <div className="flex space-x-2">
              <button
                className="p-1.5 rounded-full text-gray-500 hover:bg-gray-100 transition-colors"
                aria-label="Add to wishlist"
              >
                <Heart className="w-5 h-5" />
              </button>

              <button
                className="p-1.5 rounded-full bg-primary-100 text-primary-600 hover:bg-primary-200 transition-colors"
                aria-label="Add to cart"
                onClick={handleAddToCart}
              >
                <ShoppingCart className="w-5 h-5" />
              </button>
            </div>
          </div>
        </div>
      </Link>
    </div>
  );
};

export default ProductCard;
```

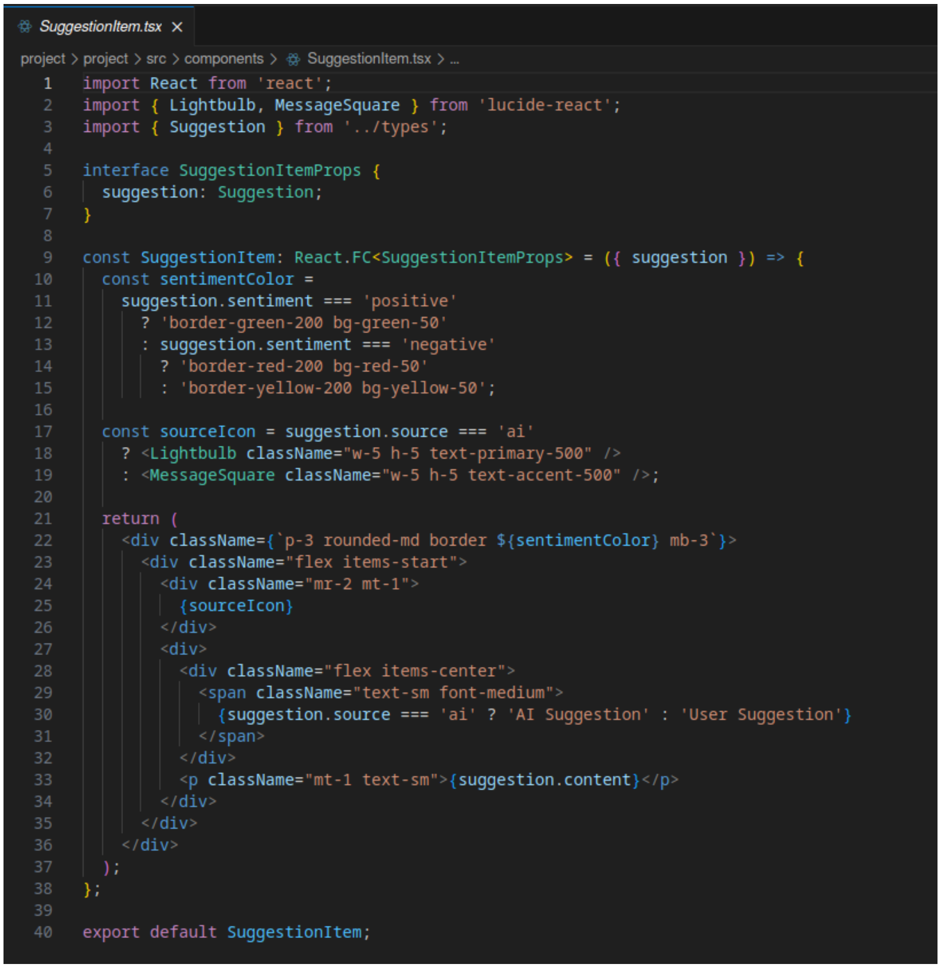


```tsx
import React from 'react';
import { Star } from 'lucide-react';
import { Review } from '../types';

interface ReviewItemProps {
  review: Review;
}

const ReviewItem: React.FC<ReviewItemProps> = ({ review }) => {
  const sentimentColor =
    review.sentiment.overall === 'positive'
      ? 'bg-green-100 text-green-800'
      : review.sentiment.overall === 'negative'
        ? 'bg-red-100 text-red-800'
        : 'bg-yellow-100 text-yellow-800';

  return (
    <div className="border-b border-gray-200 py-4">
      <div className="flex items-center justify-between">
        <div className="flex items-center">
          <div className="bg-primary-100 rounded-full p-2 mr-3">
            <span className="text-lg font-medium text-primary-700">
              {review.username.charAt(0).toUpperCase()}
            </span>
          </div>
          <div>
            <h4 className="font-medium">{review.username}</h4>
            <div className="flex items-center mt-1">
              {[...Array(5)].map((_, i) => (
                <Star
                  key={i}
                  className={`w-4 h-4 ${i < review.rating ? 'text-yellow-400 fill-yellow-400' : 'text-gray-300'}`}
                />
              ))}
              <span className="ml-2 text-sm text-gray-600">{review.date}</span>
            </div>
          </div>
        </div>

        <span className={`text-xs px-2 py-1 rounded-full ${sentimentColor}`}>
          {review.sentiment.overall.charAt(0).toUpperCase() + review.sentiment.overall.slice(1)}
        </span>
      </div>

      <p className="mt-3 text-gray-700">{review.comment}</p>
    </div>
  );
};

export default ReviewItem;
```



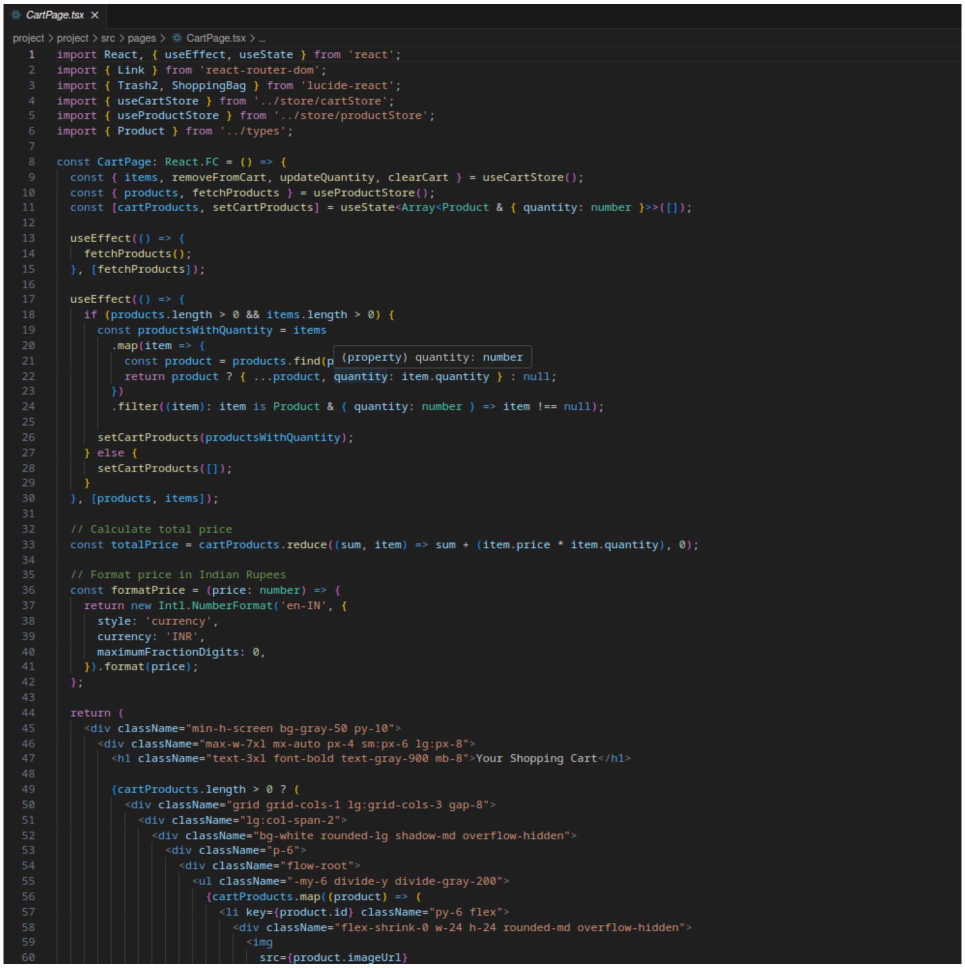
SentimentChart.tsx ×
project > project > src > components > SentimentChart.tsx > ...

```tsx
import React from 'react';
import { Pie } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  ArcElement,
  Tooltip,
  Legend,
  ChartOptions
} from 'chart.js';
import { SentimentScore } from '../types';

ChartJS.register(ArcElement, Tooltip, Legend);

interface SentimentChartProps {
  sentiment: SentimentScore;
}

const SentimentChart: React.FC<SentimentChartProps> = ({ sentiment }) => {
  const data = {
    labels: ['Positive', 'Neutral', 'Negative'],
    datasets: [
      {
        data: [
          Math.round(sentiment.positive * 100),
          Math.round(sentiment.neutral * 100),
          Math.round(sentiment.negative * 100)
        ],
        backgroundColor: [
          'rgba(75, 192, 192, 0.6)',
          'rgba(255, 206, 86, 0.6)',
          'rgba(255, 99, 132, 0.6)'
        ],
        borderColor: [
          'rgba(75, 192, 192, 1)',
          'rgba(255, 206, 86, 1)',
          'rgba(255, 99, 132, 1)'
        ],
        borderWidth: 1,
      },
    ],
  };

  const options: ChartOptions<'pie'> = {
    responsive: true,
    plugins: {
      legend: {
        position: 'right' as const,
      },
      tooltip: {
        callbacks: {
          label: function(context) {
            return `${context.label}: ${context.raw}%`;
          }
        }
      }
    }
  };

  return (
    <div className="w-full h-48">
      <Pie data={data} options={options} />
    </div>
  );
};

export default SentimentChart;
```

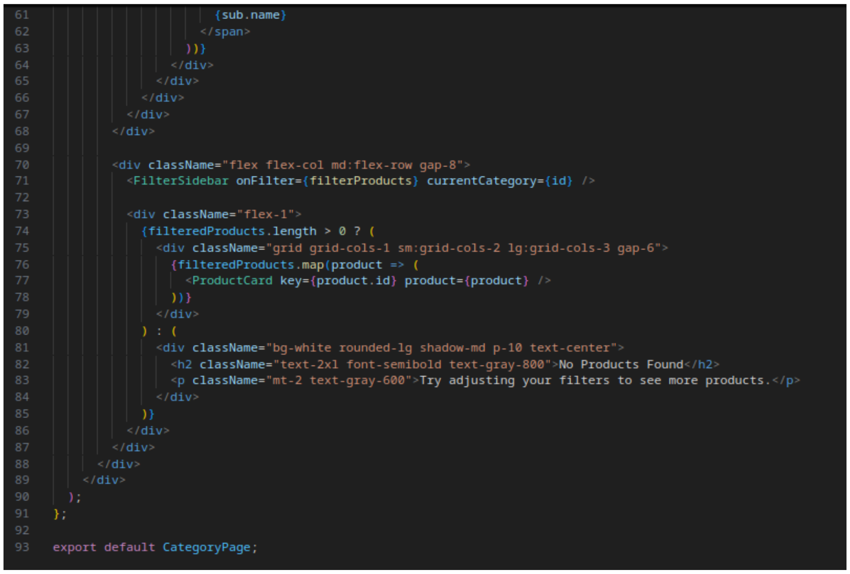


SuggestionItem.tsx ×
project > project > src > components > SuggestionItem.tsx > ...

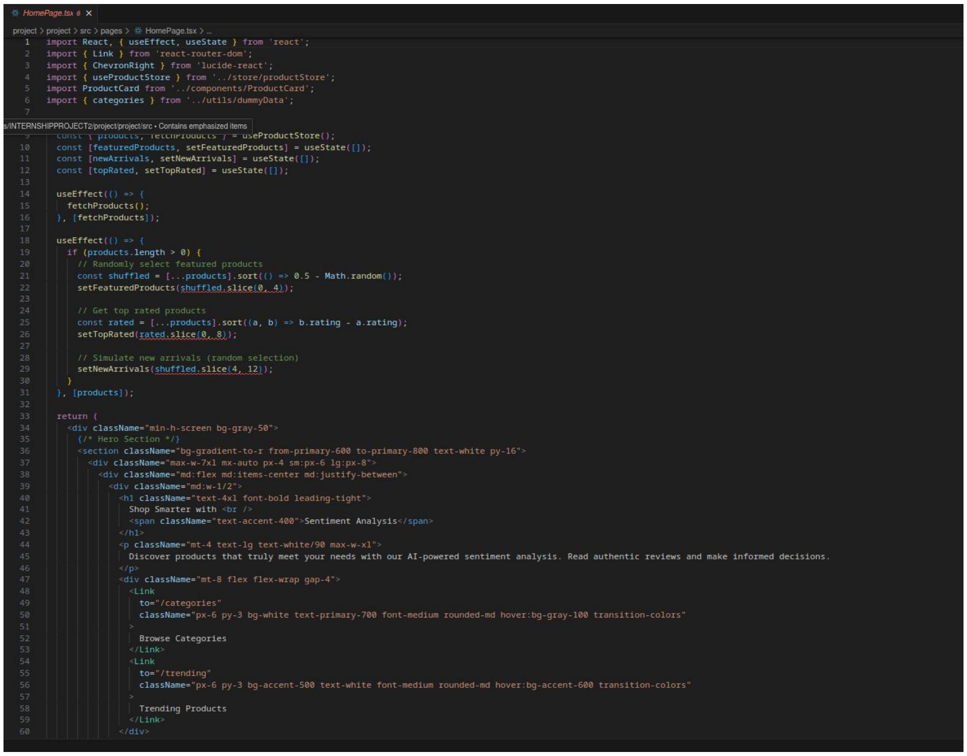
```tsx
import React from 'react';
import { Lightbulb, MessageSquare } from 'lucide-react';
import { Suggestion } from '../types';

interface SuggestionItemProps {
  suggestion: Suggestion;
}

const SuggestionItem: React.FC<SuggestionItemProps> = ({ suggestion }) => {
  const sentimentColor =
    suggestion.sentiment === 'positive'
      ? 'border-green-200 bg-green-50'
      : suggestion.sentiment === 'negative'
        ? 'border-red-200 bg-red-50'
        : 'border-yellow-200 bg-yellow-50';

  const sourceIcon = suggestion.source === 'ai'
    ? <Lightbulb className="w-5 h-5 text-primary-500" />
    : <MessageSquare className="w-5 h-5 text-accent-500" />;

  return (
    <div className={`p-3 rounded-md border ${sentimentColor} mb-3`}>
      <div className="flex items-start">
        <div className="mr-2 mt-1">
          {sourceIcon}
        </div>
        <div>
          <div className="flex items-center">
            <span className="text-sm font-medium">
              {suggestion.source === 'ai' ? 'AI Suggestion' : 'User Suggestion'}
            </span>
          </div>
          <p className="mt-1 text-sm">{suggestion.content}</p>
        </div>
      </div>
    </div>
  );
};

export default SuggestionItem;
```


CartPage.tsx
project > project > src > pages > CartPage.tsx > ...
(property) quantity: number

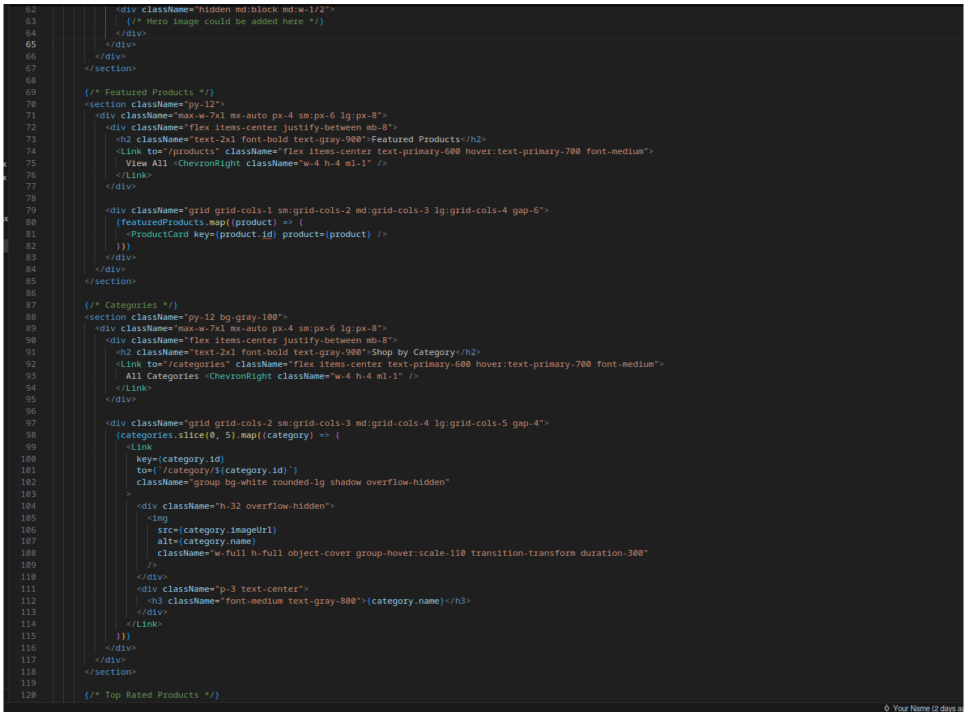
```tsx
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { Trash2, ShoppingBag } from 'lucide-react';
import { useCartStore } from '../store/cartStore';
import { useProductStore } from '../store/productStore';
import { Product } from '../types';

const CartPage: React.FC = () => {
  const { items, removeFromCart, updateQuantity, clearCart } = useCartStore();
  const { products, fetchProducts } = useProductStore();
  const [cartProducts, setCartProducts] = useState<Array<Product & { quantity: number }>>([]);

  useEffect(() => {
    fetchProducts();
  }, [fetchProducts]);

  useEffect(() => {
    if (products.length > 0 && items.length > 0) {
      const productsWithQuantity = items
        .map(item => {
          const product = products.find(p
          return product ? { ...product, quantity: item.quantity } : null;
        })
        .filter((item): item is Product & { quantity: number } => item !== null);

      setCartProducts(productsWithQuantity);
    } else {
      setCartProducts([]);
    }
  }, [products, items]);

  // Calculate total price
  const totalPrice = cartProducts.reduce((sum, item) => sum + (item.price * item.quantity), 0);

  // Format price in Indian Rupees
  const formatPrice = (price: number) => {
    return new Intl.NumberFormat('en-IN', {
      style: 'currency',
      currency: 'INR',
      maximumFractionDigits: 0,
    }).format(price);
  };

  return (
    <div className="min-h-screen bg-gray-50 py-10">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <h1 className="text-3xl font-bold text-gray-900 mb-8">Your Shopping Cart</h1>

        {cartProducts.length > 0 ? (
          <div className="grid grid-cols-1 lg:grid-cols-3 gap-8">
            <div className="lg:col-span-2">
              <div className="bg-white rounded-lg shadow-md overflow-hidden">
                <div className="p-6">
                  <div className="flow-root">
                    <ul className="-my-6 divide-y divide-gray-200">
                      {cartProducts.map((product) => (
                        <li key={product.id} className="py-6 flex">
                          <div className="flex-shrink-0 w-24 h-24 rounded-md overflow-hidden">
                            <img
                              src={product.imageUrl}
```



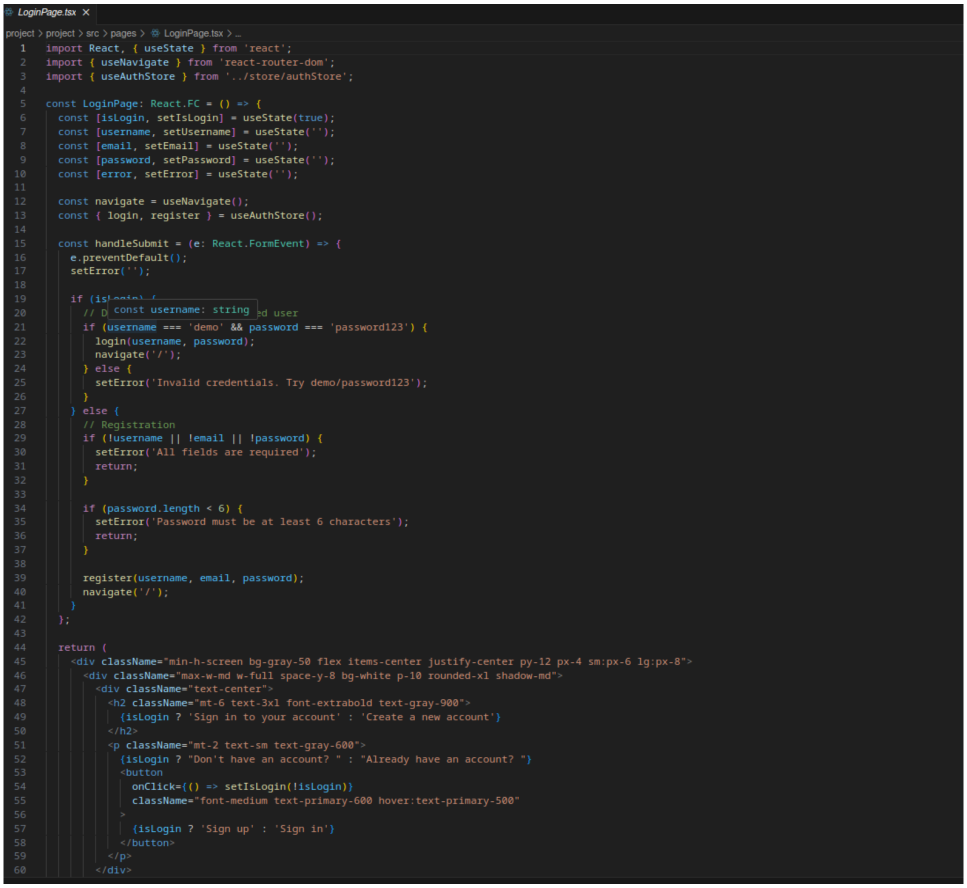
```
                      {sub.name}
                    </span>
                  ))}
                </div>
              </div>
            </div>
          </div>
        </div>

        <div className="flex flex-col md:flex-row gap-8">
          <FilterSidebar onFilter={filterProducts} currentCategory={id} />

          <div className="flex-1">
            {filteredProducts.length > 0 ? (
              <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
                {filteredProducts.map(product => (
                  <ProductCard key={product.id} product={product} />
                ))}
              </div>
            ) : (
              <div className="bg-white rounded-lg shadow-md p-10 text-center">
                <h2 className="text-2xl font-semibold text-gray-800">No Products Found</h2>
                <p className="mt-2 text-gray-600">Try adjusting your filters to see more products.</p>
              </div>
            )}
          </div>
        </div>
      </div>
    </div>
  );
};

export default CategoryPage;
```


HomePage.tsx 6 ×
project > project > src > pages > HomePage.tsx > ...
s/INTERNSHIPPROJECT2/project/project/src • Contains emphasized items

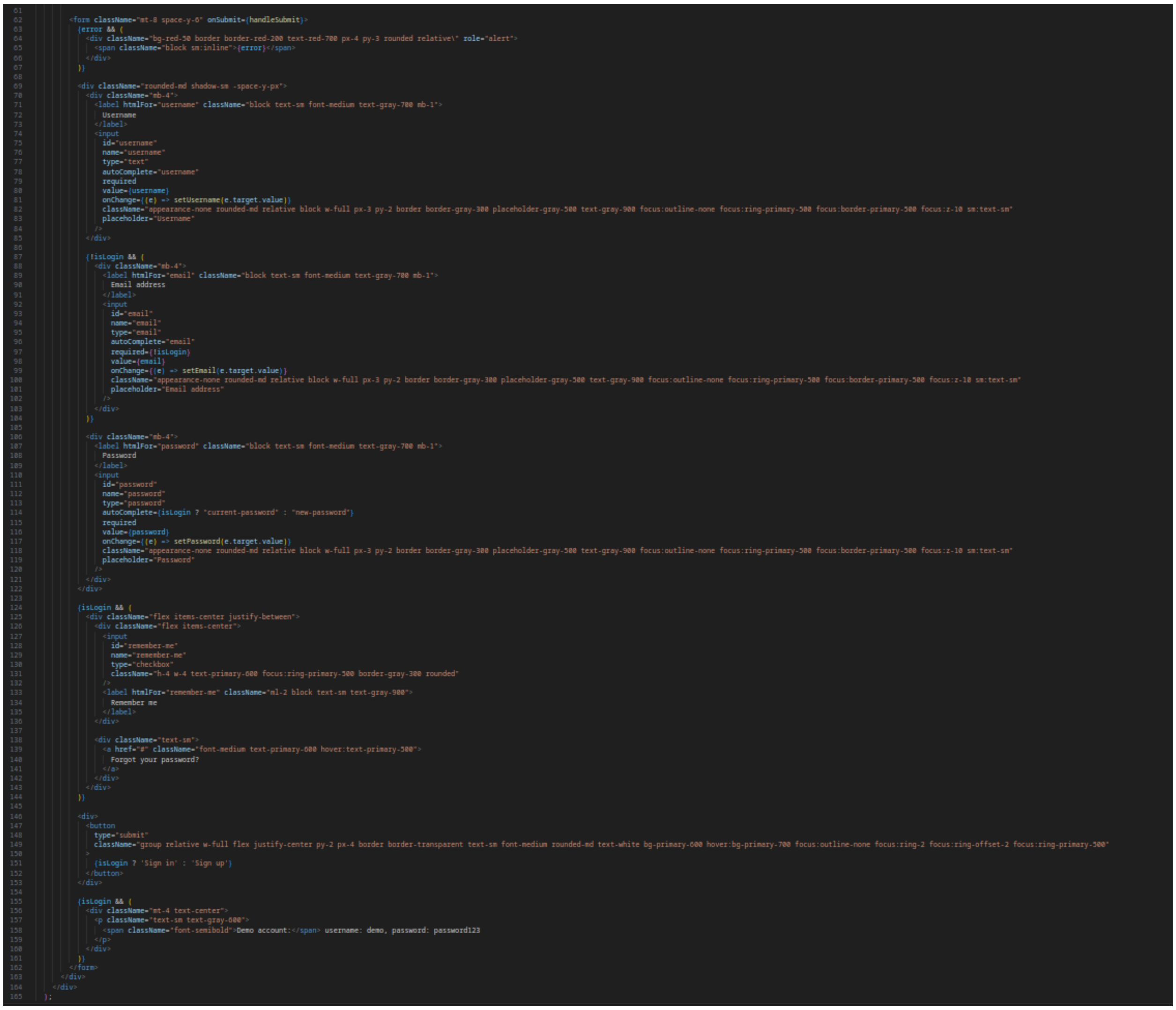
```
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { ChevronRight } from 'lucide-react';
import { useProductStore } from '../store/productStore';
import ProductCard from '../components/ProductCard';
import { categories } from '../utils/dummyData';

  const { products, fetchProducts } = useProductStore();
  const [featuredProducts, setFeaturedProducts] = useState([]);
  const [newArrivals, setNewArrivals] = useState([]);
  const [topRated, setTopRated] = useState([]);

  useEffect(() => {
    fetchProducts();
  }, [fetchProducts]);

  useEffect(() => {
    if (products.length > 0) {
      // Randomly select featured products
      const shuffled = [...products].sort(() => 0.5 - Math.random());
      setFeaturedProducts(shuffled.slice(0, 4));

      // Get top rated products
      const rated = [...products].sort((a, b) => b.rating - a.rating);
      setTopRated(rated.slice(0, 8));

      // Simulate new arrivals (random selection)
      setNewArrivals(shuffled.slice(4, 12));
    }
  }, [products]);

  return (
    <div className="min-h-screen bg-gray-50">
      {/* Hero Section */}
      <section className="bg-gradient-to-r from-primary-600 to-primary-800 text-white py-16">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <div className="md:flex md:items-center md:justify-between">
            <div className="md:w-1/2">
              <h1 className="text-4xl font-bold leading-tight">
                Shop Smarter with <br />
                <span className="text-accent-400">Sentiment Analysis</span>
              </h1>
              <p className="mt-4 text-lg text-white/90 max-w-xl">
                Discover products that truly meet your needs with our AI-powered sentiment analysis. Read authentic reviews and make informed decisions.
              </p>
              <div className="mt-8 flex flex-wrap gap-4">
                <Link
                  to="/categories"
                  className="px-6 py-3 bg-white text-primary-700 font-medium rounded-md hover:bg-gray-100 transition-colors"
                >
                  Browse Categories
                </Link>
                <Link
                  to="/trending"
                  className="px-6 py-3 bg-accent-500 text-white font-medium rounded-md hover:bg-accent-600 transition-colors"
                >
                  Trending Products
                </Link>
              </div>
```



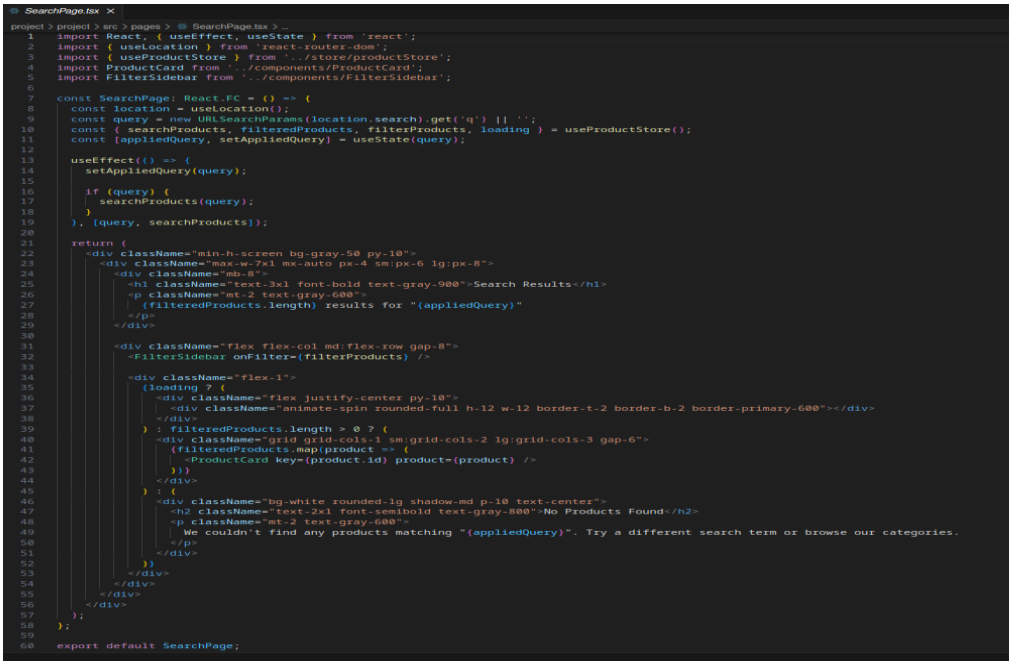
```jsx
      <div className="hidden md:block md:w-1/2">
        {/* Hero image could be added here */}
      </div>
    </div>
  </div>
</section>

{/* Featured Products */}
<section className="py-12">
  <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
    <div className="flex items-center justify-between mb-8">
      <h2 className="text-2xl font-bold text-gray-900">Featured Products</h2>
      <Link to="/products" className="flex items-center text-primary-600 hover:text-primary-700 font-medium">
        View All <ChevronRight className="w-4 h-4 ml-1" />
      </Link>
    </div>

    <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6">
      {featuredProducts.map((product) => (
        <ProductCard key={product.id} product={product} />
      ))}
    </div>
  </div>
</section>

{/* Categories */}
<section className="py-12 bg-gray-100">
  <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
    <div className="flex items-center justify-between mb-8">
      <h2 className="text-2xl font-bold text-gray-900">Shop by Category</h2>
      <Link to="/categories" className="flex items-center text-primary-600 hover:text-primary-700 font-medium">
        All Categories <ChevronRight className="w-4 h-4 ml-1" />
      </Link>
    </div>

    <div className="grid grid-cols-2 sm:grid-cols-3 md:grid-cols-4 lg:grid-cols-5 gap-4">
      {categories.slice(0, 5).map((category) => (
        <Link
          key={category.id}
          to={`/category/${category.id}`}
          className="group bg-white rounded-lg shadow overflow-hidden"
        >
          <div className="h-32 overflow-hidden">
            <img
              src={category.imageUrl}
              alt={category.name}
              className="w-full h-full object-cover group-hover:scale-110 transition-transform duration-300"
            />
          </div>
          <div className="p-3 text-center">
            <h3 className="font-medium text-gray-800">{category.name}</h3>
          </div>
        </Link>
      ))}
    </div>
  </div>
</section>

{/* Top Rated Products */}
```



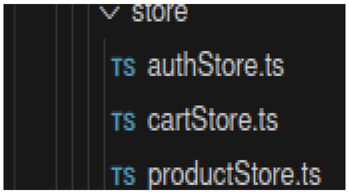
```tsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuthStore } from '../store/authStore';

const LoginPage: React.FC = () => {
  const [isLogin, setIsLogin] = useState(true);
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const navigate = useNavigate();
  const { login, register } = useAuthStore();

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    setError('');

    if (isLogin) {
      // D[illegible]ed user
      if (username === 'demo' && password === 'password123') {
        login(username, password);
        navigate('/');
      } else {
        setError('Invalid credentials. Try demo/password123');
      }
    } else {
      // Registration
      if (!username || !email || !password) {
        setError('All fields are required');
        return;
      }

      if (password.length < 6) {
        setError('Password must be at least 6 characters');
        return;
      }

      register(username, email, password);
      navigate('/');
    }
  };

  return (
    <div className="min-h-screen bg-gray-50 flex items-center justify-center py-12 px-4 sm:px-6 lg:px-8">
      <div className="max-w-md w-full space-y-8 bg-white p-10 rounded-xl shadow-md">
        <div className="text-center">
          <h2 className="mt-6 text-3xl font-extrabold text-gray-900">
            {isLogin ? 'Sign in to your account' : 'Create a new account'}
          </h2>
          <p className="mt-2 text-sm text-gray-600">
            {isLogin ? "Don't have an account? " : "Already have an account? "}
            <button
              onClick={() => setIsLogin(!isLogin)}
              className="font-medium text-primary-600 hover:text-primary-500"
            >
              {isLogin ? 'Sign up' : 'Sign in'}
            </button>
          </p>
        </div>
```



```
        <form className="mt-8 space-y-6" onSubmit={handleSubmit}>
          {error && (
            <div className="bg-red-50 border border-red-200 text-red-700 px-4 py-3 rounded relative\" role="alert">
              <span className="block sm:inline">{error}</span>
            </div>
          )}

          <div className="rounded-md shadow-sm -space-y-px">
            <div className="mb-4">
              <label htmlFor="username" className="block text-sm font-medium text-gray-700 mb-1">
                Username
              </label>
              <input
                id="username"
                name="username"
                type="text"
                autoComplete="username"
                required
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                className="appearance-none rounded-md relative block w-full px-3 py-2 border border-gray-300 placeholder-gray-500 text-gray-900 focus:outline-none focus:ring-primary-500 focus:border-primary-500 focus:z-10 sm:text-sm"
                placeholder="Username"
              />
            </div>

            {!isLogin && (
              <div className="mb-4">
                <label htmlFor="email" className="block text-sm font-medium text-gray-700 mb-1">
                  Email address
                </label>
                <input
                  id="email"
                  name="email"
                  type="email"
                  autoComplete="email"
                  required={!isLogin}
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  className="appearance-none rounded-md relative block w-full px-3 py-2 border border-gray-300 placeholder-gray-500 text-gray-900 focus:outline-none focus:ring-primary-500 focus:border-primary-500 focus:z-10 sm:text-sm"
                  placeholder="Email address"
                />
              </div>
            )}

            <div className="mb-4">
              <label htmlFor="password" className="block text-sm font-medium text-gray-700 mb-1">
                Password
              </label>
              <input
                id="password"
                name="password"
                type="password"
                autoComplete={isLogin ? "current-password" : "new-password"}
                required
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="appearance-none rounded-md relative block w-full px-3 py-2 border border-gray-300 placeholder-gray-500 text-gray-900 focus:outline-none focus:ring-primary-500 focus:border-primary-500 focus:z-10 sm:text-sm"
                placeholder="Password"
              />
            </div>
          </div>

          {isLogin && (
            <div className="flex items-center justify-between">
              <div className="flex items-center">
                <input
                  id="remember-me"
                  name="remember-me"
                  type="checkbox"
                  className="h-4 w-4 text-primary-600 focus:ring-primary-500 border-gray-300 rounded"
                />
                <label htmlFor="remember-me" className="ml-2 block text-sm text-gray-900">
                  Remember me
                </label>
              </div>

              <div className="text-sm">
                <a href="#" className="font-medium text-primary-600 hover:text-primary-500">
                  Forgot your password?
                </a>
              </div>
            </div>
          )}

          <div>
            <button
              type="submit"
              className="group relative w-full flex justify-center py-2 px-4 border border-transparent text-sm font-medium rounded-md text-white bg-primary-600 hover:bg-primary-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary-500"
            >
              {isLogin ? 'Sign in' : 'Sign up'}
            </button>
          </div>

          {isLogin && (
            <div className="mt-4 text-center">
              <p className="text-sm text-gray-600">
                <span className="font-semibold">Demo account:</span> username: demo, password: password123
              </p>
            </div>
          )}
        </form>
      </div>
    </div>
  );
```



```
import React, { useEffect, useState } from 'react';
import { useLocation } from 'react-router-dom';
import { useProductStore } from '../store/productStore';
import ProductCard from '../components/ProductCard';
import FilterSidebar from '../components/FilterSidebar';

const SearchPage: React.FC = () => {
  const location = useLocation();
  const query = new URLSearchParams(location.search).get('q') || '';
  const { searchProducts, filteredProducts, filterProducts, loading } = useProductStore();
  const [appliedQuery, setAppliedQuery] = useState(query);

  useEffect(() => {
    setAppliedQuery(query);

    if (query) {
      searchProducts(query);
    }
  }, [query, searchProducts]);

  return (
    <div className="min-h-screen bg-gray-50 py-10">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="mb-8">
          <h1 className="text-3xl font-bold text-gray-900">Search Results</h1>
          <p className="mt-2 text-gray-600">
            {filteredProducts.length} results for "{appliedQuery}"
          </p>
        </div>

        <div className="flex flex-col md:flex-row gap-8">
          <FilterSidebar onFilter={filterProducts} />

          <div className="flex-1">
            {loading ? (
              <div className="flex justify-center py-10">
                <div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-primary-600"></div>
              </div>
            ) : filteredProducts.length > 0 ? (
              <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
                {filteredProducts.map(product => (
                  <ProductCard key={product.id} product={product} />
                ))}
              </div>
            ) : (
              <div className="bg-white rounded-lg shadow-md p-10 text-center">
                <h2 className="text-2xl font-semibold text-gray-800">No Products Found</h2>
                <p className="mt-2 text-gray-600">
                  We couldn't find any products matching "{appliedQuery}". Try a different search term or browse our categories.
                </p>
              </div>
            )}
          </div>
        </div>
      </div>
    </div>
  );
};

export default SearchPage;
```

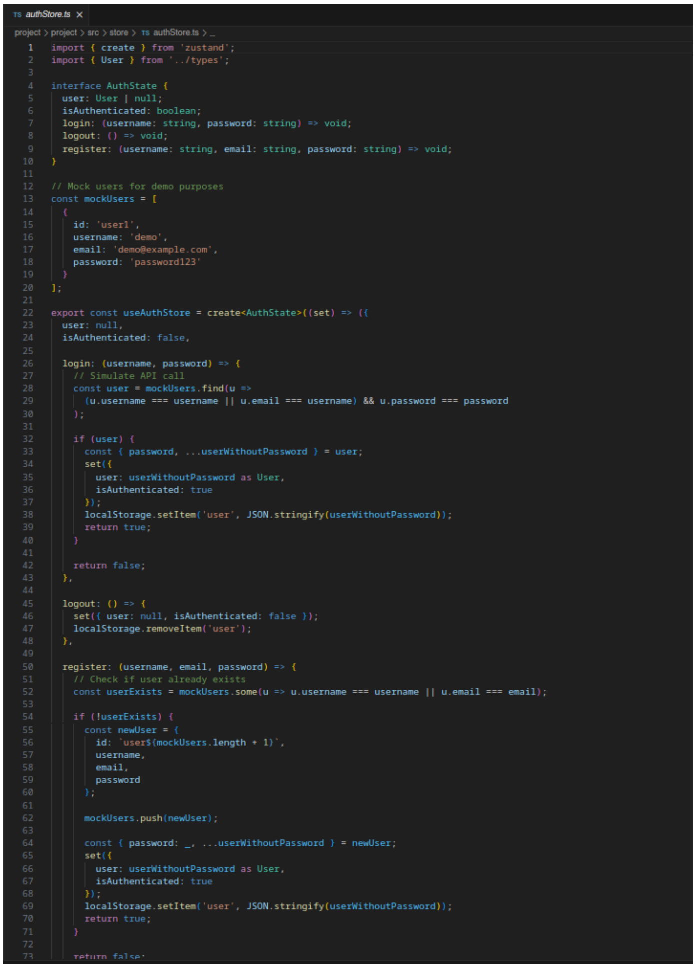


store

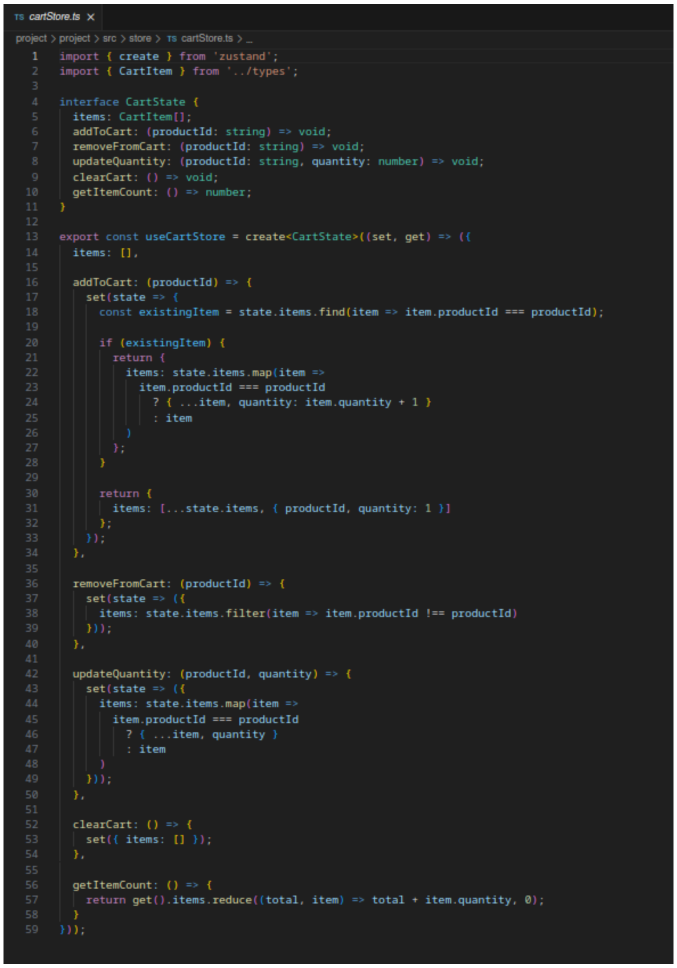
authStore.ts

cartStore.ts

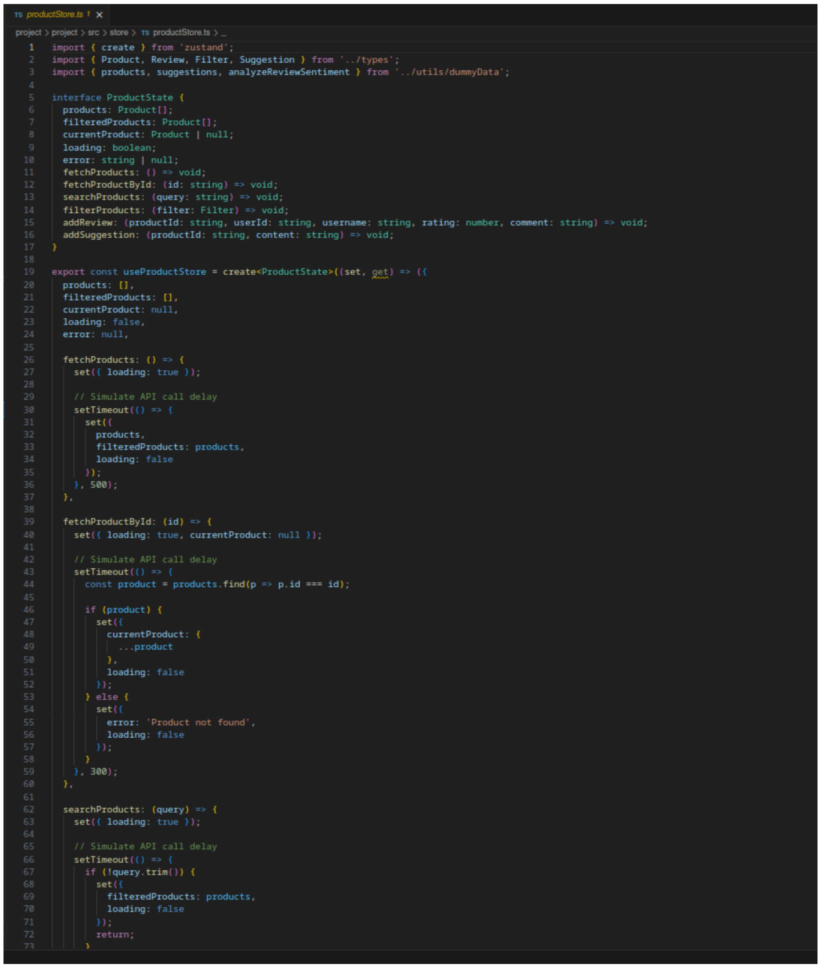
productStore.ts



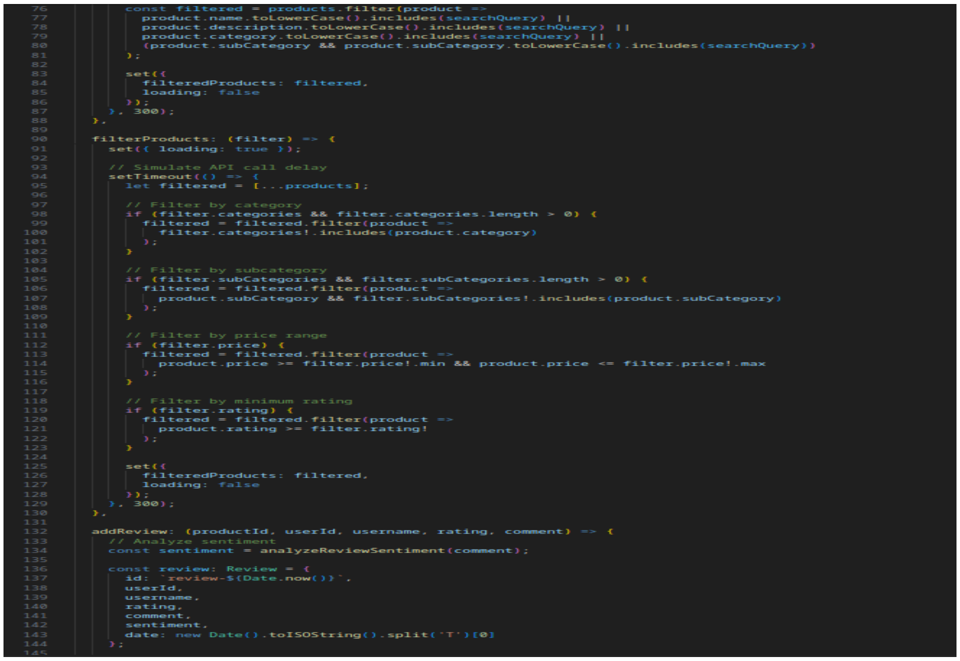
```typescript
import { create } from 'zustand';
import { User } from '../types';

interface AuthState {
  user: User | null;
  isAuthenticated: boolean;
  login: (username: string, password: string) => void;
  logout: () => void;
  register: (username: string, email: string, password: string) => void;
}

// Mock users for demo purposes
const mockUsers = [
  {
    id: 'user1',
    username: 'demo',
    email: 'demo@example.com',
    password: 'password123'
  }
];

export const useAuthStore = create<AuthState>((set) => ({
  user: null,
  isAuthenticated: false,

  login: (username, password) => {
    // Simulate API call
    const user = mockUsers.find(u =>
      (u.username === username || u.email === username) && u.password === password
    );

    if (user) {
      const { password, ...userWithoutPassword } = user;
      set({
        user: userWithoutPassword as User,
        isAuthenticated: true
      });
      localStorage.setItem('user', JSON.stringify(userWithoutPassword));
      return true;
    }

    return false;
  },

  logout: () => {
    set({ user: null, isAuthenticated: false });
    localStorage.removeItem('user');
  },

  register: (username, email, password) => {
    // Check if user already exists
    const userExists = mockUsers.some(u => u.username === username || u.email === email);

    if (!userExists) {
      const newUser = {
        id: `user${mockUsers.length + 1}`,
        username,
        email,
        password
      };

      mockUsers.push(newUser);

      const { password: _, ...userWithoutPassword } = newUser;
      set({
        user: userWithoutPassword as User,
        isAuthenticated: true
      });
      localStorage.setItem('user', JSON.stringify(userWithoutPassword));
      return true;
    }

    return false;
```



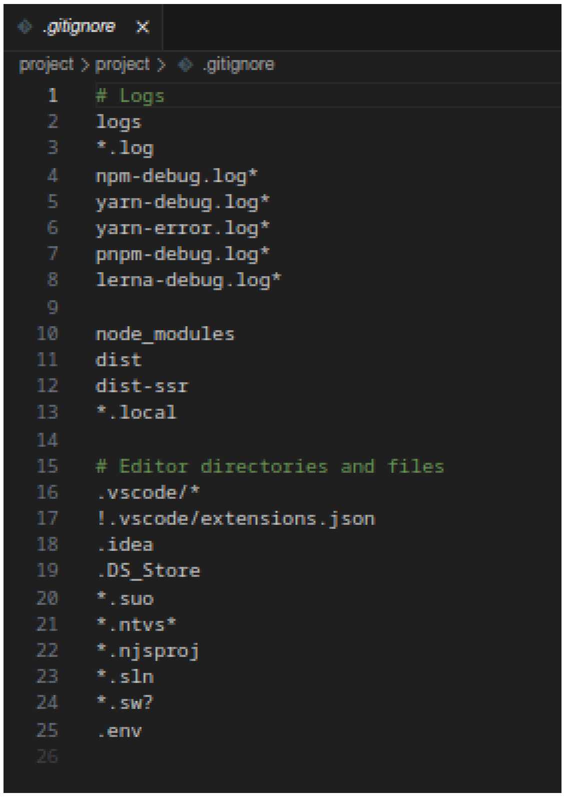
```typescript
import { create } from 'zustand';
import { CartItem } from '../types';

interface CartState {
  items: CartItem[];
  addToCart: (productId: string) => void;
  removeFromCart: (productId: string) => void;
  updateQuantity: (productId: string, quantity: number) => void;
  clearCart: () => void;
  getItemCount: () => number;
}

export const useCartStore = create<CartState>((set, get) => ({
  items: [],

  addToCart: (productId) => {
    set(state => {
      const existingItem = state.items.find(item => item.productId === productId);

      if (existingItem) {
        return {
          items: state.items.map(item =>
            item.productId === productId
              ? { ...item, quantity: item.quantity + 1 }
              : item
          )
        };
      }

      return {
        items: [...state.items, { productId, quantity: 1 }]
      };
    });
  },

  removeFromCart: (productId) => {
    set(state => ({
      items: state.items.filter(item => item.productId !== productId)
    }));
  },

  updateQuantity: (productId, quantity) => {
    set(state => ({
      items: state.items.map(item =>
        item.productId === productId
          ? { ...item, quantity }
          : item
      )
    }));
  },

  clearCart: () => {
    set({ items: [] });
  },

  getItemCount: () => {
    return get().items.reduce((total, item) => total + item.quantity, 0);
  }
}));
```



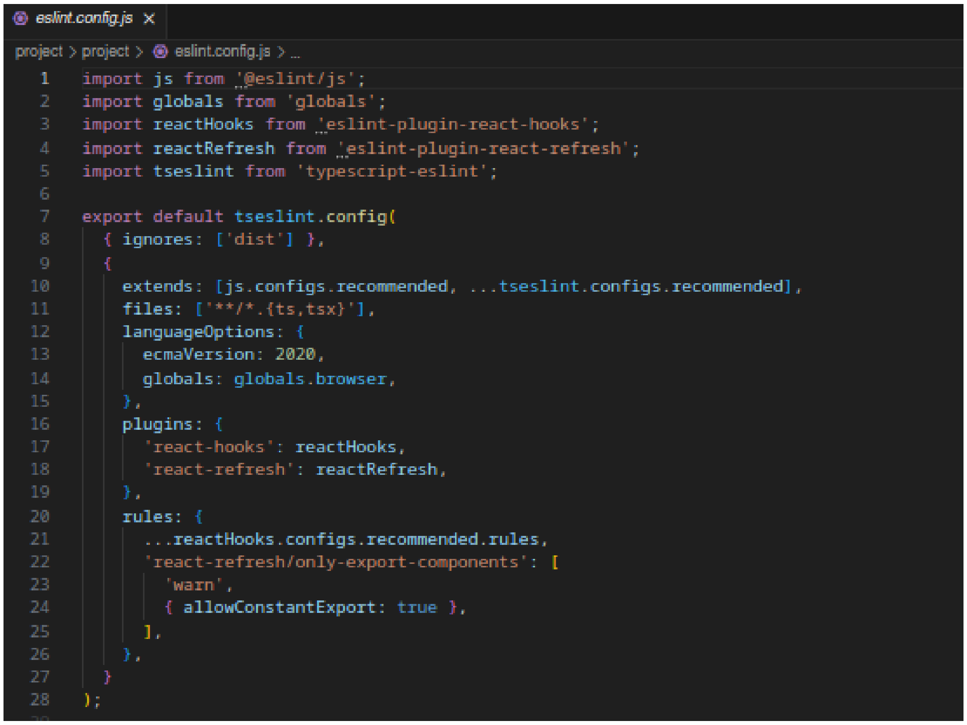
```typescript
import { create } from 'zustand';
import { Product, Review, Filter, Suggestion } from '../types';
import { products, suggestions, analyzeReviewSentiment } from '../utils/dummyData';

interface ProductState {
  products: Product[];
  filteredProducts: Product[];
  currentProduct: Product | null;
  loading: boolean;
  error: string | null;
  fetchProducts: () => void;
  fetchProductById: (id: string) => void;
  searchProducts: (query: string) => void;
  filterProducts: (filter: Filter) => void;
  addReview: (productId: string, userId: string, username: string, rating: number, comment: string) => void;
  addSuggestion: (productId: string, content: string) => void;
}

export const useProductStore = create<ProductState>((set, get) => ({
  products: [],
  filteredProducts: [],
  currentProduct: null,
  loading: false,
  error: null,

  fetchProducts: () => {
    set({ loading: true });

    // Simulate API call delay
    setTimeout(() => {
      set({
        products,
        filteredProducts: products,
        loading: false
      });
    }, 500);
  },

  fetchProductById: (id) => {
    set({ loading: true, currentProduct: null });

    // Simulate API call delay
    setTimeout(() => {
      const product = products.find(p => p.id === id);

      if (product) {
        set({
          currentProduct: {
            ...product
          },
          loading: false
        });
      } else {
        set({
          error: 'Product not found',
          loading: false
        });
      }
    }, 300);
  },

  searchProducts: (query) => {
    set({ loading: true });

    // Simulate API call delay
    setTimeout(() => {
      if (!query.trim()) {
        set({
          filteredProducts: products,
          loading: false
        });
        return;
      }
```



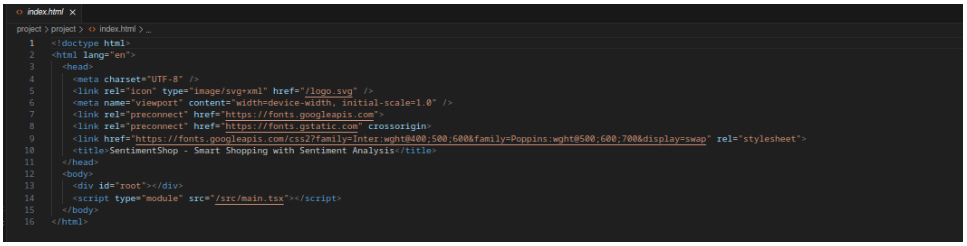
```typescript
      const filtered = products.filter(product =>
        product.name.toLowerCase().includes(searchQuery) ||
        product.description.toLowerCase().includes(searchQuery) ||
        product.category.toLowerCase().includes(searchQuery) ||
        (product.subCategory && product.subCategory.toLowerCase().includes(searchQuery))
      );

      set({
        filteredProducts: filtered,
        loading: false
      });
    }, 300);
  },

  filterProducts: (filter) => {
    set({ loading: true });

    // Simulate API call delay
    setTimeout(() => {
      let filtered = [...products];

      // Filter by category
      if (filter.categories && filter.categories.length > 0) {
        filtered = filtered.filter(product =>
          filter.categories!.includes(product.category)
        );
      }

      // Filter by subcategory
      if (filter.subCategories && filter.subCategories.length > 0) {
        filtered = filtered.filter(product =>
          product.subCategory && filter.subCategories!.includes(product.subCategory)
        );
      }

      // Filter by price range
      if (filter.price) {
        filtered = filtered.filter(product =>
          product.price >= filter.price!.min && product.price <= filter.price!.max
        );
      }

      // Filter by minimum rating
      if (filter.rating) {
        filtered = filtered.filter(product =>
          product.rating >= filter.rating!
        );
      }

      set({
        filteredProducts: filtered,
        loading: false
      });
    }, 300);
  },

  addReview: (productId, userId, username, rating, comment) => {
    // Analyze sentiment
    const sentiment = analyzeReviewSentiment(comment);

    const review: Review = {
      id: `review-${Date.now()}`,
      userId,
      username,
      rating,
      comment,
      sentiment,
      date: new Date().toISOString().split('T')[0]
    };
```



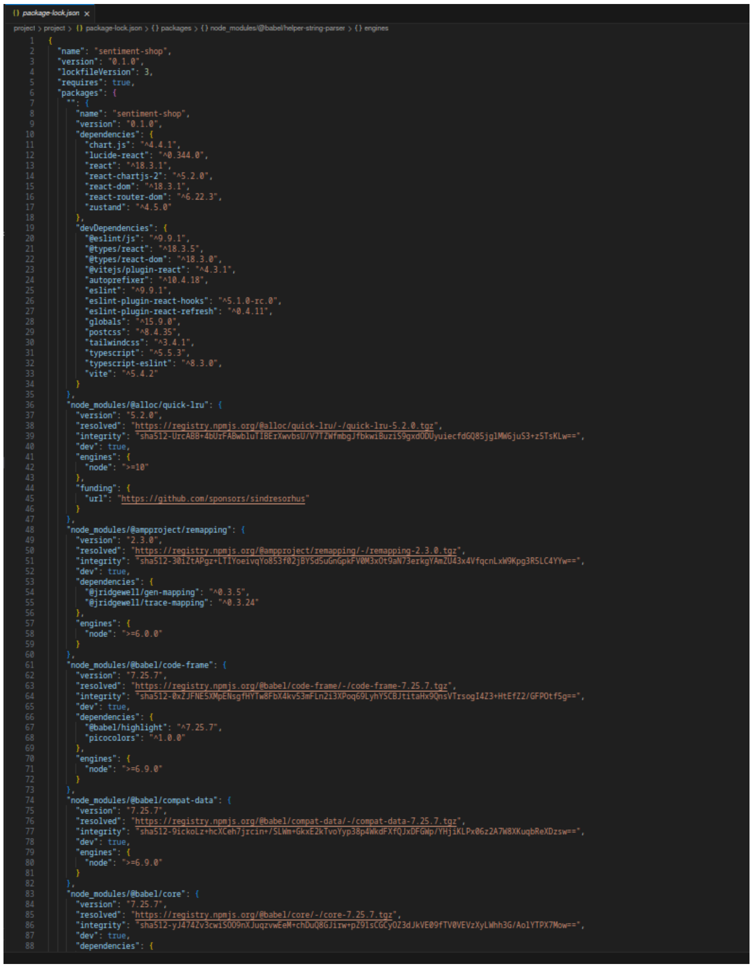
```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
.env

```



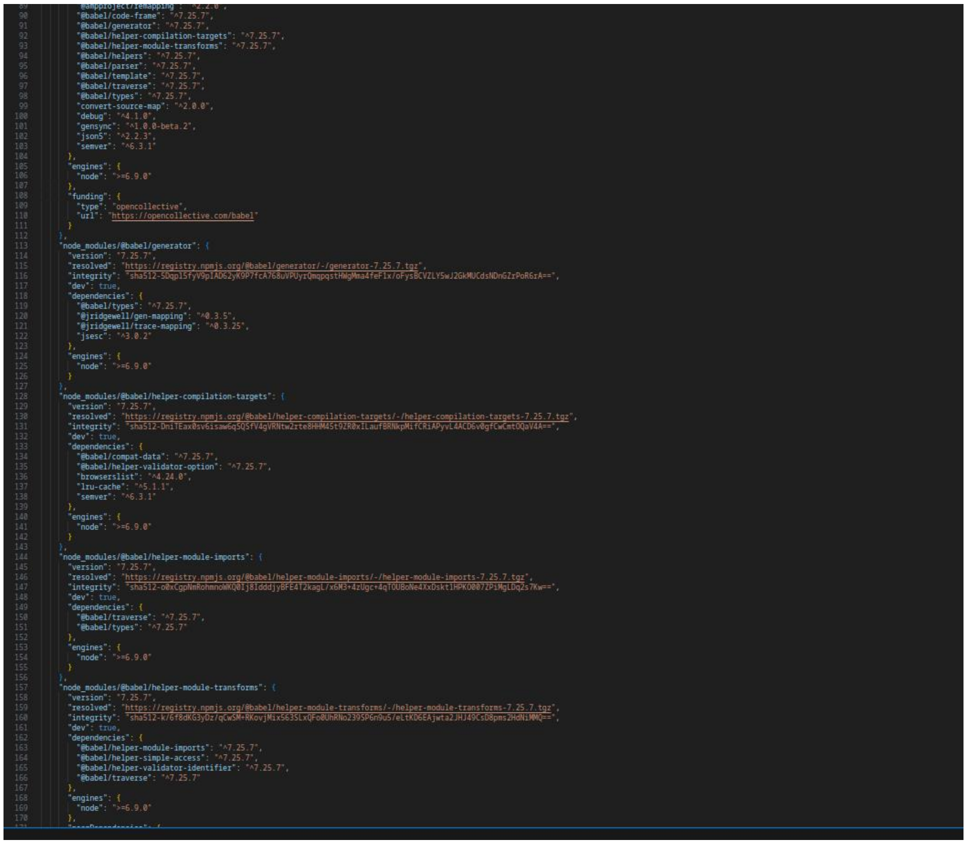
```js
import js from '@eslint/js';
import globals from 'globals';
import reactHooks from 'eslint-plugin-react-hooks';
import reactRefresh from 'eslint-plugin-react-refresh';
import tseslint from 'typescript-eslint';

export default tseslint.config(
  { ignores: ['dist'] },
  {
    extends: [js.configs.recommended, ...tseslint.configs.recommended],
    files: ['**/*.{ts,tsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...reactHooks.configs.recommended.rules,
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  }
);
```



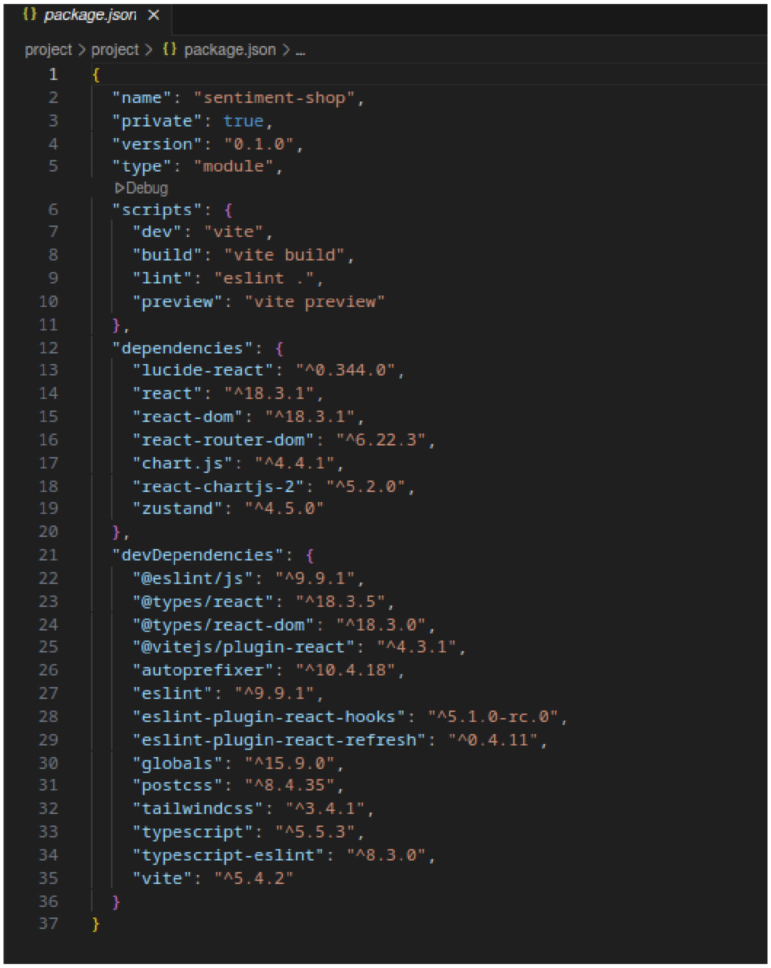
```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/logo.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600&family=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
    <title>SentimentShop - Smart Shopping with Sentiment Analysis</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```


package-lock.json
project > project > package-lock.json > packages > node_modules/@babel/helper-string-parser > engines

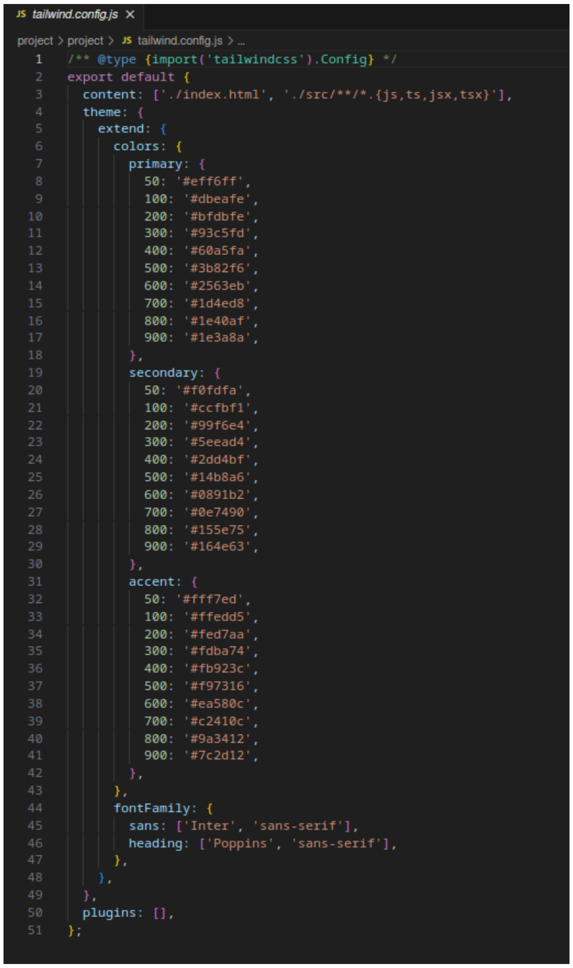
```json
{
  "name": "sentiment-shop",
  "version": "0.1.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "sentiment-shop",
      "version": "0.1.0",
      "dependencies": {
        "chart.js": "^4.4.1",
        "lucide-react": "^0.344.0",
        "react": "^18.3.1",
        "react-chartjs-2": "^5.2.0",
        "react-dom": "^18.3.1",
        "react-router-dom": "^6.22.3",
        "zustand": "^4.5.0"
      },
      "devDependencies": {
        "@eslint/js": "^9.9.1",
        "@types/react": "^18.3.5",
        "@types/react-dom": "^18.3.0",
        "@vitejs/plugin-react": "^4.3.1",
        "autoprefixer": "^10.4.18",
        "eslint": "^9.9.1",
        "eslint-plugin-react-hooks": "^5.1.0-rc.0",
        "eslint-plugin-react-refresh": "^0.4.11",
        "globals": "^15.9.0",
        "postcss": "^8.4.35",
        "tailwindcss": "^3.4.1",
        "typescript": "^5.5.3",
        "typescript-eslint": "^8.3.0",
        "vite": "^5.4.2"
      }
    },
    "node_modules/@alloc/quick-lru": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@alloc/quick-lru/-/quick-lru-5.2.0.tgz",
      "integrity": "sha512-UrcABB+4bUrFABwbluTIBErXwvbsU/V7TZWfmbgJfbkwiBuziS9gxdODUyuiecfdGQ85jglMW6juS3+z5TsKLw==",
      "dev": true,
      "engines": {
        "node": ">=10"
      },
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/@ampproject/remapping": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/@ampproject/remapping/-/remapping-2.3.0.tgz",
      "integrity": "sha512-30iZtAPgz+LTIYoeivqYo853f02jBYSd5uGnGpkFV0M3xOt9aN73erkgYAmZU43x4VfqcnLxW9Kpg3R5LC4YYw==",
      "dev": true,
      "dependencies": {
        "@jridgewell/gen-mapping": "^0.3.5",
        "@jridgewell/trace-mapping": "^0.3.24"
      },
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/@babel/code-frame": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.25.7.tgz",
      "integrity": "sha512-0xZJFNE5XMpENsgfHYTw8FbX4kv53mFLn2i3XPoq69LyhYSCBJtitaHx9QnsVTrsogI4Z3+HtEfZ2/GFPOtf5g==",
      "dev": true,
      "dependencies": {
        "@babel/highlight": "^7.25.7",
        "picocolors": "^1.0.0"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/compat-data": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/compat-data/-/compat-data-7.25.7.tgz",
      "integrity": "sha512-9ickoLz+hcXCeh7jrcin+/SLWm+GkxE2kTvoYyp38p4WkdFXfQJxDFGWp/YHjiKLPx06z2A7W8XKuqbReXDzsw==",
      "dev": true,
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/core": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/core/-/core-7.25.7.tgz",
      "integrity": "sha512-yJ474Zv3cwiSOO9nXJuqzvwEeM+chDuQ8GJirw+pZ91sCGCyOZ3dJkVE09fTV0VEVzXyLWhh3G/AolYTPX7Mow==",
      "dev": true,
      "dependencies": {
```



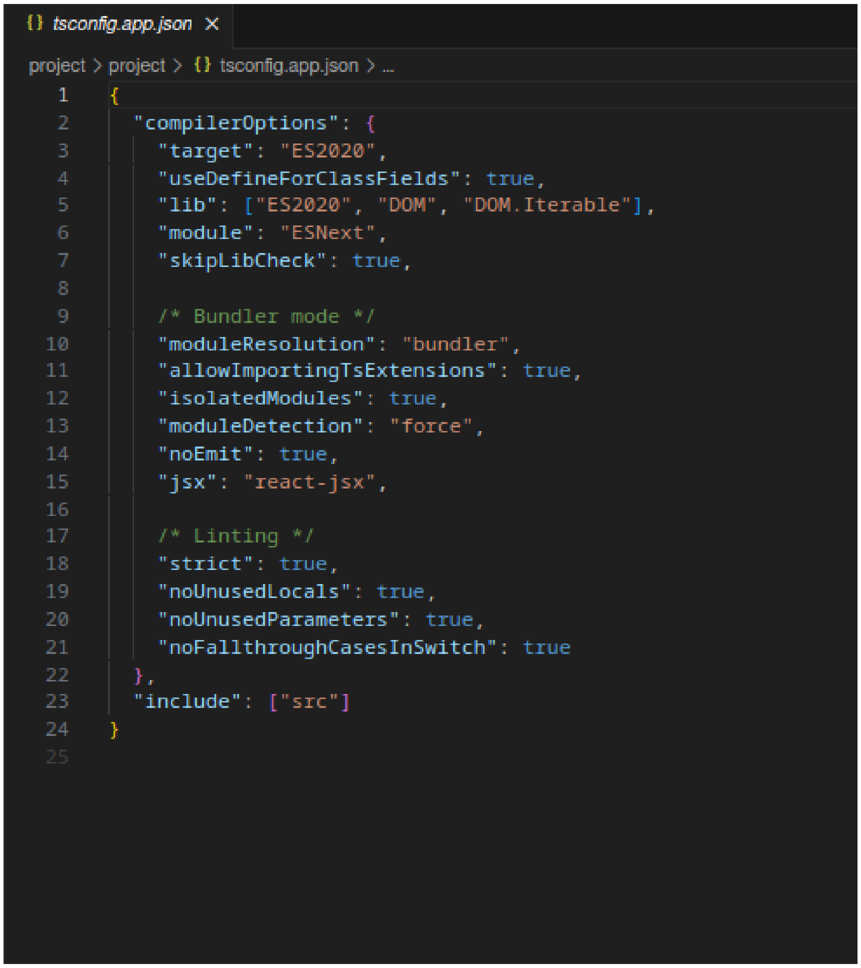
```
        "@ampproject/remapping": "^2.2.0",
        "@babel/code-frame": "^7.25.7",
        "@babel/generator": "^7.25.7",
        "@babel/helper-compilation-targets": "^7.25.7",
        "@babel/helper-module-transforms": "^7.25.7",
        "@babel/helpers": "^7.25.7",
        "@babel/parser": "^7.25.7",
        "@babel/template": "^7.25.7",
        "@babel/traverse": "^7.25.7",
        "@babel/types": "^7.25.7",
        "convert-source-map": "^2.0.0",
        "debug": "^4.1.0",
        "gensync": "^1.0.0-beta.2",
        "json5": "^2.2.3",
        "semver": "^6.3.1"
      },
      "engines": {
        "node": ">=6.9.0"
      },
      "funding": {
        "type": "opencollective",
        "url": "https://opencollective.com/babel"
      }
    },
    "node_modules/@babel/generator": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/generator/-/generator-7.25.7.tgz",
      "integrity": "sha512-5Dqpl5fyV9pIAD62yK9P7fcA768uVPUyrQmqpqstHWgMma4feF1x/oFysBCVZLY5wJ2GkMUCdsNDnGZrPoR6rA==",
      "dev": true,
      "dependencies": {
        "@babel/types": "^7.25.7",
        "@jridgewell/gen-mapping": "^0.3.5",
        "@jridgewell/trace-mapping": "^0.3.25",
        "jsesc": "^3.0.2"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/helper-compilation-targets": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/helper-compilation-targets/-/helper-compilation-targets-7.25.7.tgz",
      "integrity": "sha512-DniTEax0sv6isaw6qSQSfV4gVRNtw2rte8HHM45t9ZR0xILaufBRNkpMifCRiAPyvL4ACD6v0gfCwCmtOQaV4A==",
      "dev": true,
      "dependencies": {
        "@babel/compat-data": "^7.25.7",
        "@babel/helper-validator-option": "^7.25.7",
        "browserslist": "^4.24.0",
        "lru-cache": "^5.1.1",
        "semver": "^6.3.1"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/helper-module-imports": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/helper-module-imports/-/helper-module-imports-7.25.7.tgz",
      "integrity": "sha512-o0xCgpNmRohmnoWKQ0Ij8IdddjyBFE4T2kagL/x6M3+4zUgc+4qTOUBoNe4XxDskt1HPKO007ZPiMgLDq2s7Kw==",
      "dev": true,
      "dependencies": {
        "@babel/traverse": "^7.25.7",
        "@babel/types": "^7.25.7"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/helper-module-transforms": {
      "version": "7.25.7",
      "resolved": "https://registry.npmjs.org/@babel/helper-module-transforms/-/helper-module-transforms-7.25.7.tgz",
      "integrity": "sha512-k/6f8dKG3yDz/qCwSM+RKovjMix563SLxQFo0UhRNo239SP6n9u5/eLtKD6EAjwta2JHJ49CsD8pms2HdNiMMQ==",
      "dev": true,
      "dependencies": {
        "@babel/helper-module-imports": "^7.25.7",
        "@babel/helper-simple-access": "^7.25.7",
        "@babel/helper-validator-identifier": "^7.25.7",
        "@babel/traverse": "^7.25.7"
      },
      "engines": {
        "node": ">=6.9.0"
      },
      "peerDependencies": {
```



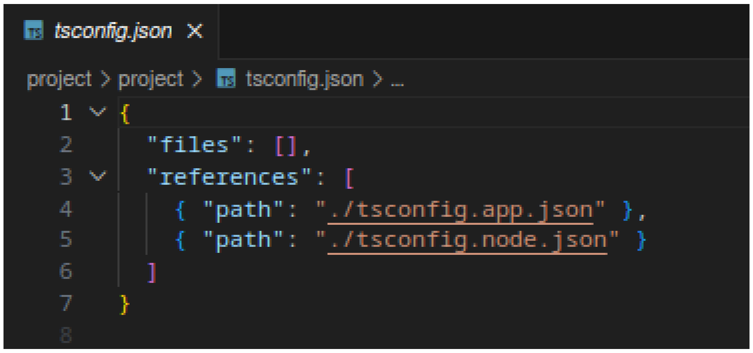
```json
{
  "name": "sentiment-shop",
  "private": true,
  "version": "0.1.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "lucide-react": "^0.344.0",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.22.3",
    "chart.js": "^4.4.1",
    "react-chartjs-2": "^5.2.0",
    "zustand": "^4.5.0"
  },
  "devDependencies": {
    "@eslint/js": "^9.9.1",
    "@types/react": "^18.3.5",
    "@types/react-dom": "^18.3.0",
    "@vitejs/plugin-react": "^4.3.1",
    "autoprefixer": "^10.4.18",
    "eslint": "^9.9.1",
    "eslint-plugin-react-hooks": "^5.1.0-rc.0",
    "eslint-plugin-react-refresh": "^0.4.11",
    "globals": "^15.9.0",
    "postcss": "^8.4.35",
    "tailwindcss": "^3.4.1",
    "typescript": "^5.5.3",
    "typescript-eslint": "^8.3.0",
    "vite": "^5.4.2"
  }
}
```



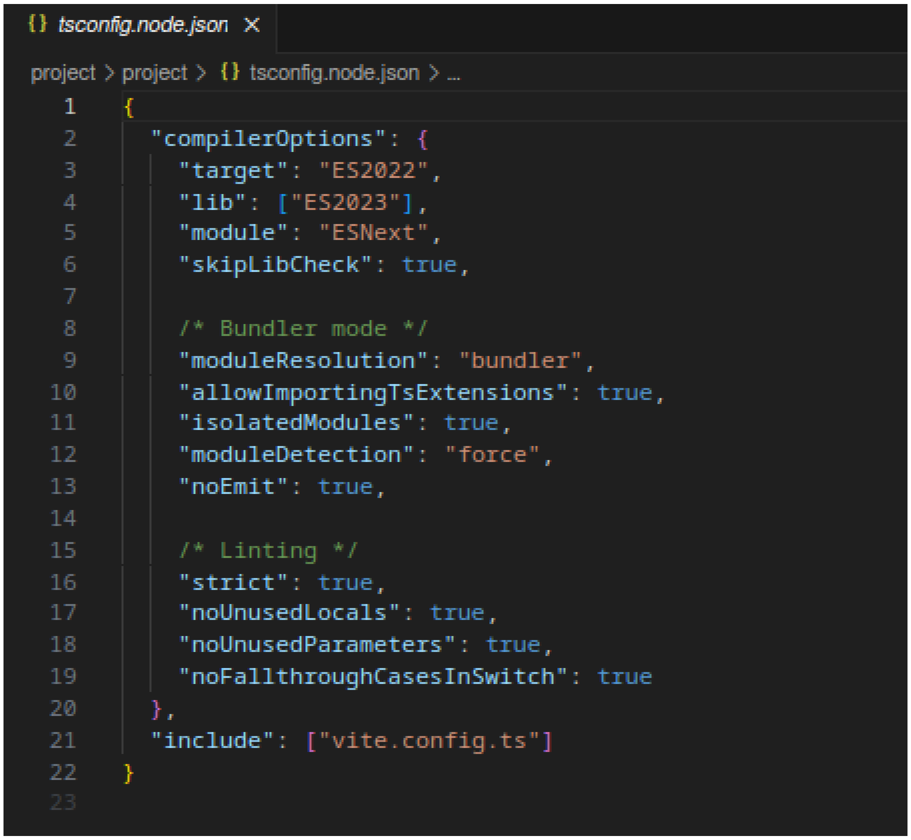
```js
/** @type {import('tailwindcss').Config} */
export default {
  content: ['./index.html', './src/**/*.{js,ts,jsx,tsx}'],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#eff6ff',
          100: '#dbeafe',
          200: '#bfdbfe',
          300: '#93c5fd',
          400: '#60a5fa',
          500: '#3b82f6',
          600: '#2563eb',
          700: '#1d4ed8',
          800: '#1e40af',
          900: '#1e3a8a',
        },
        secondary: {
          50: '#f0fdfa',
          100: '#ccfbf1',
          200: '#99f6e4',
          300: '#5eead4',
          400: '#2dd4bf',
          500: '#14b8a6',
          600: '#0891b2',
          700: '#0e7490',
          800: '#155e75',
          900: '#164e63',
        },
        accent: {
          50: '#fff7ed',
          100: '#ffedd5',
          200: '#fed7aa',
          300: '#fdba74',
          400: '#fb923c',
          500: '#f97316',
          600: '#ea580c',
          700: '#c2410c',
          800: '#9a3412',
          900: '#7c2d12',
        },
      },
      fontFamily: {
        sans: ['Inter', 'sans-serif'],
        heading: ['Poppins', 'sans-serif'],
      },
    },
  },
  plugins: [],
};
```



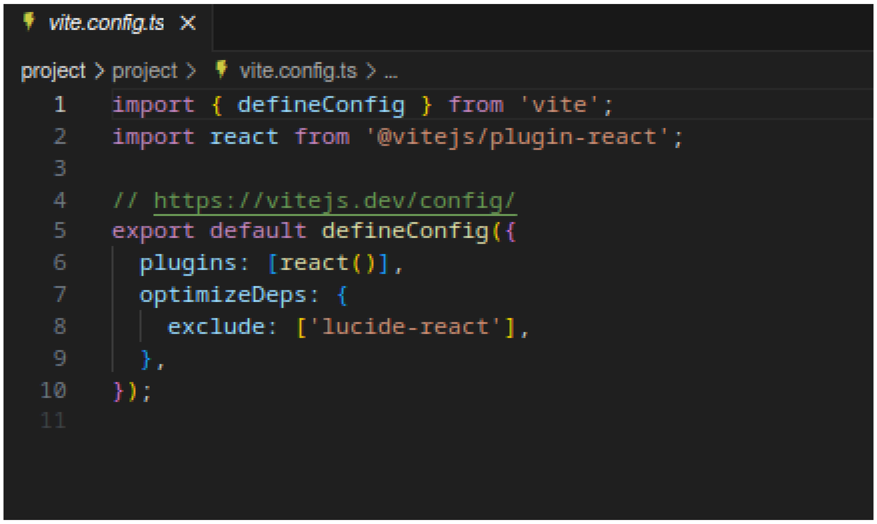
```json
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src"]
}
```



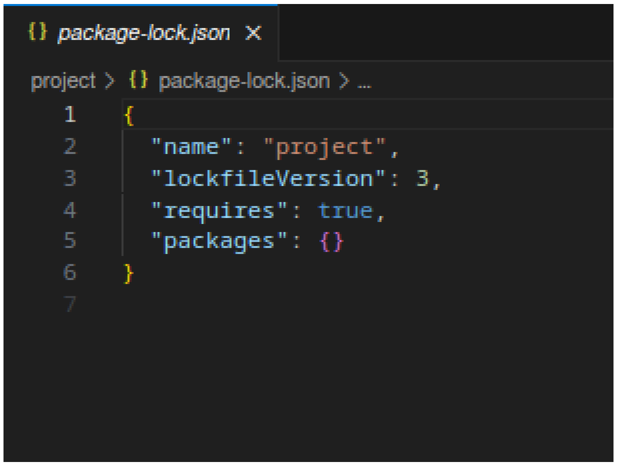
```json
{
  "files": [],
  "references": [
    { "path": "./tsconfig.app.json" },
    { "path": "./tsconfig.node.json" }
  ]
}
```



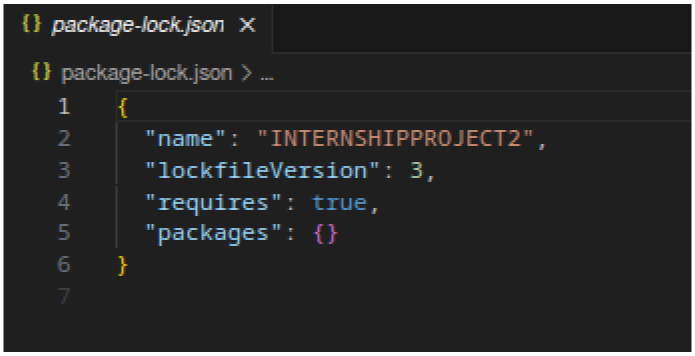
```json
{
  "compilerOptions": {
    "target": "ES2022",
    "lib": ["ES2023"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["vite.config.ts"]
}
```



```ts
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  optimizeDeps: {
    exclude: ['lucide-react'],
  },
});
```

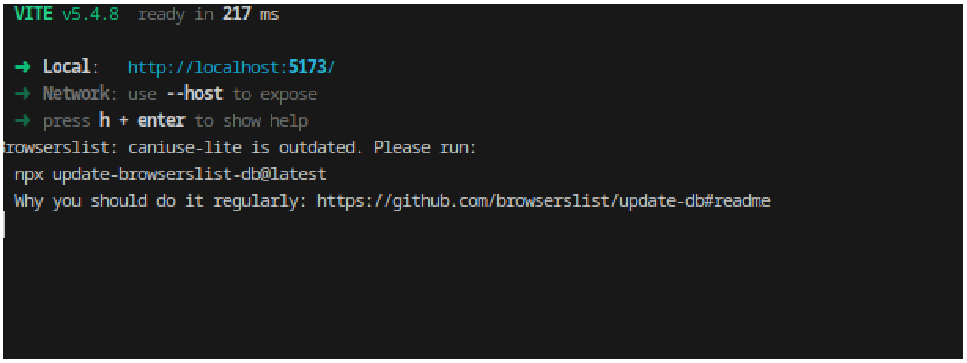


```json
{
  "name": "project",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {}
}
```

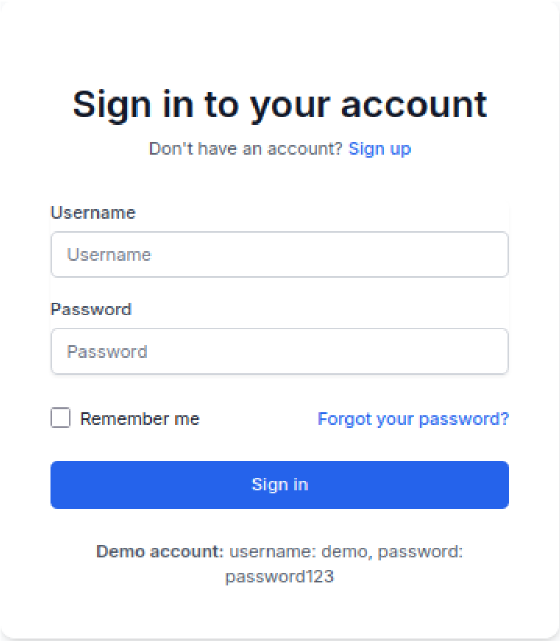


```json
{
  "name": "INTERNSHIPPROJECT2",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {}
}
```

Output images



```
VITE v5.4.8  ready in 217 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
Browserslist: caniuse-lite is outdated. Please run:
 npx update-browserslist-db@latest
 Why you should do it regularly: https://github.com/browserslist/update-db#readme
```



Sign in to your account

Don't have an account? Sign up

Username

Username

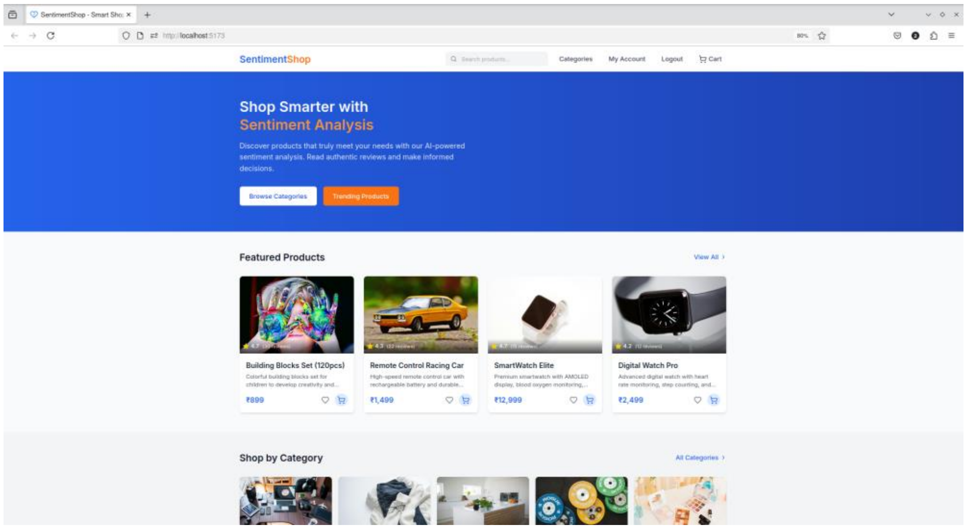
Password

Password

□ Remember me Forgot your password?

Sign in

Demo account: username: demo, password: password123



SentimentShop - Smart Sho ×

http://localhost:5173

Shop Smarter with
Sentiment Analysis

Discover products that truly meet your needs with our AI-powered sentiment analysis. Read authentic reviews and make informed decisions.

Featured Products

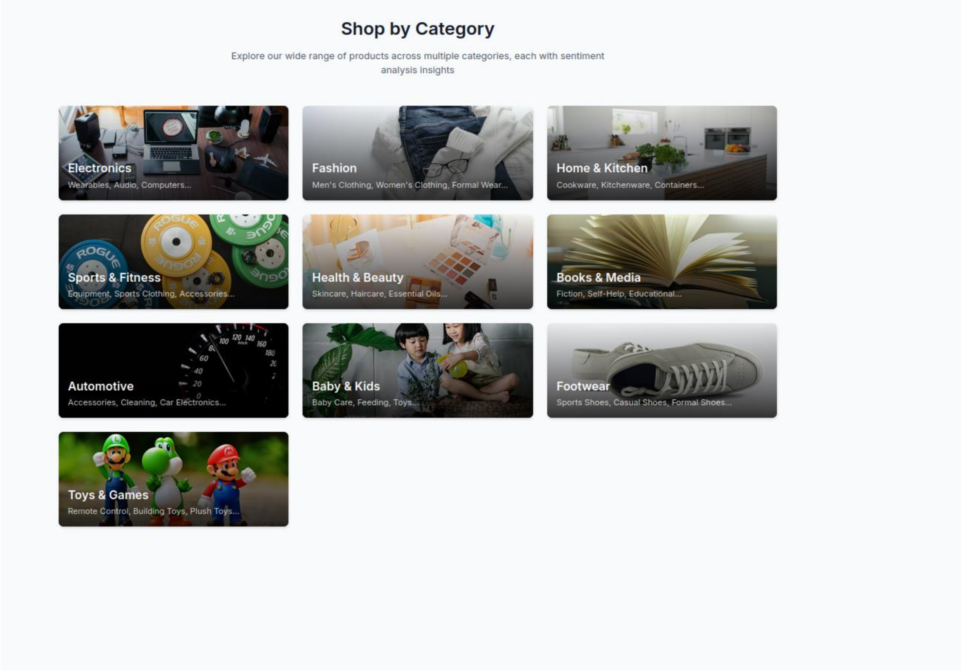
Building Blocks Set (120pcs)
Colorful building blocks set for children to develop creativity and...
₹899

Remote Control Racing Car
High-speed remote control car with rechargeable battery and durable...
₹1,499

SmartWatch Elite
Premium smartwatch with AMOLED display, blood oxygen monitoring,...
₹12,999

Digital Watch Pro
Advanced digital watch with heart rate monitoring, step counting, and...
₹2,499

Shop by Category



Shop by Category

Explore our wide range of products across multiple categories, each with sentiment analysis insights

Electronics
Wearables, Audio, Computers...

Fashion
Men's Clothing, Women's Clothing, Formal Wear...

Home & Kitchen
Cookware, Kitchenware, Containers...

Sports & Fitness
Equipment, Sports Clothing, Accessories...

Health & Beauty
Skincare, Haircare, Essential Oils...

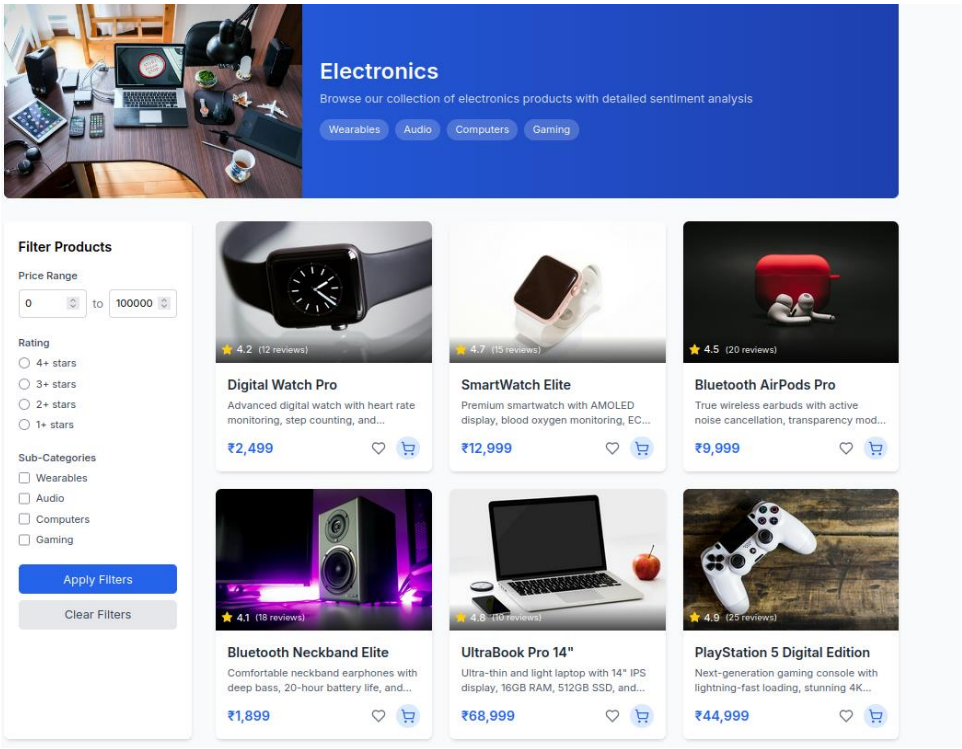
Books & Media
Fiction, Self-Help, Educational...

Automotive
Accessories, Cleaning, Car Electronics...

Baby & Kids
Baby Care, Feeding, Toys...

Footwear
Sports Shoes, Casual Shoes, Formal Shoes...

Toys & Games
Remote Control, Building Toys, Plush Toys...



Electronics

Browse our collection of electronics products with detailed sentiment analysis

Filter Products

Price Range: 0 to 100000

Rating
○ 4+ stars
○ 3+ stars
○ 2+ stars
○ 1+ stars

Sub-Categories
□ Wearables
□ Audio
□ Computers
□ Gaming

★ 4.2 (12 reviews)
Digital Watch Pro
Advanced digital watch with heart rate monitoring, step counting, and...
₹2,499

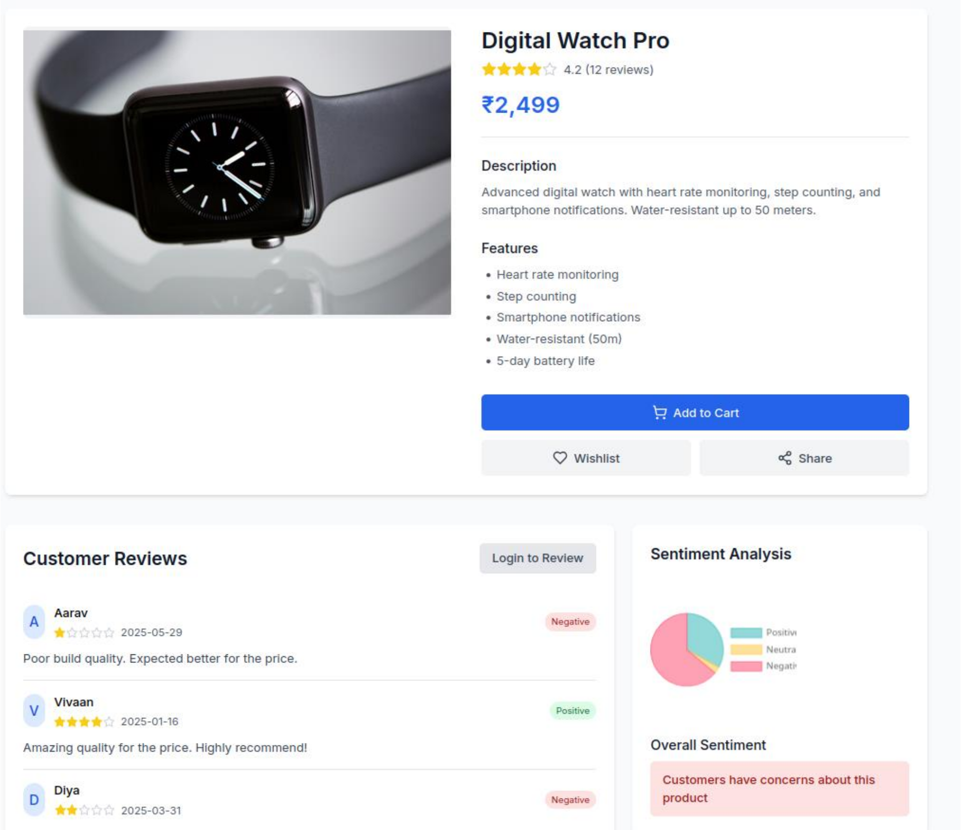
★ 4.7 (15 reviews)
SmartWatch Elite
Premium smartwatch with AMOLED display, blood oxygen monitoring, EC...
₹12,999

★ 4.5 (20 reviews)
Bluetooth AirPods Pro
True wireless earbuds with active noise cancellation, transparency mod...
₹9,999

★ 4.1 (18 reviews)
Bluetooth Neckband Elite
Comfortable neckband earphones with deep bass, 20-hour battery life, and...
₹1,899

★ 4.8 (10 reviews)
UltraBook Pro 14"
Ultra-thin and light laptop with 14" IPS display, 16GB RAM, 512GB SSD, and...
₹68,999

★ 4.9 (25 reviews)
PlayStation 5 Digital Edition
Next-generation gaming console with lightning-fast loading, stunning 4K...
₹44,999



Digital Watch Pro

★★★★☆ 4.2 (12 reviews)

₹2,499

Description

Advanced digital watch with heart rate monitoring, step counting, and smartphone notifications. Water-resistant up to 50 meters.

Features

- Heart rate monitoring
- Step counting
- Smartphone notifications
- Water-resistant (50m)
- 5-day battery life

Customer Reviews

A Aarav
★☆☆☆☆ 2025-05-29 Negative
Poor build quality. Expected better for the price.

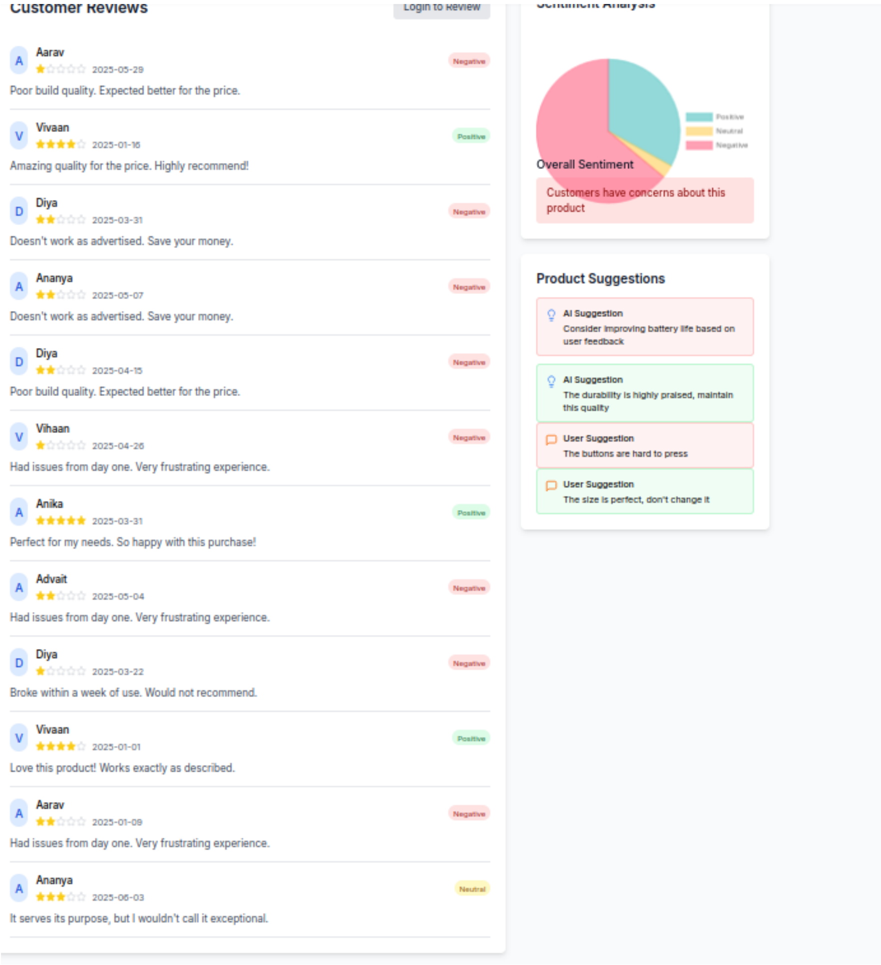
V Vivaan
★★★★☆ 2025-01-16 Positive
Amazing quality for the price. Highly recommend!

D Diya
★★☆☆☆ 2025-03-31 Negative

Sentiment Analysis

Overall Sentiment

Customers have concerns about this product



Customer Reviews

A Aarav
★☆☆☆☆ 2025-05-29 Negative
Poor build quality. Expected better for the price.

V Vivaan
★★★★☆ 2025-01-16 Positive
Amazing quality for the price. Highly recommend!

D Diya
★★☆☆☆ 2025-03-31 Negative
Doesn't work as advertised. Save your money.

A Ananya
★★☆☆☆ 2025-05-07 Negative
Doesn't work as advertised. Save your money.

D Diya
★★☆☆☆ 2025-04-15 Negative
Poor build quality. Expected better for the price.

V Vihaan
★☆☆☆☆ 2025-04-26 Negative
Had issues from day one. Very frustrating experience.

A Anika
★★★★★ 2025-03-31 Positive
Perfect for my needs. So happy with this purchase!

A Advait
★★☆☆☆ 2025-05-04 Negative
Had issues from day one. Very frustrating experience.

D Diya
★☆☆☆☆ 2025-03-22 Negative
Broke within a week of use. Would not recommend.

V Vivaan
★★★★☆ 2025-01-01 Positive
Love this product! Works exactly as described.

A Aarav
★★☆☆☆ 2025-01-09 Negative
Had issues from day one. Very frustrating experience.

A Ananya
★★★☆☆ 2025-06-03 Neutral
It serves its purpose, but I wouldn't call it exceptional.

Sentiment Analysis

Overall Sentiment

Customers have concerns about this product

Product Suggestions

AI Suggestion
Consider improving battery life based on user feedback

AI Suggestion
The durability is highly praised, maintain this quality

User Suggestion
The buttons are hard to press

User Suggestion
The size is perfect, don't change it