**Design Decisions**

1. **Technology Stack Selection**:
   * **Frontend**: I chose React.js for its component-based architecture, which allows for reusable UI components, making it easier to manage the product display and details. React's ecosystem also provides a variety of libraries (like Axios for API calls) that enhance development speed and efficiency.
   * **Backend**: Node.js with Express was selected for its non-blocking, event-driven architecture, which is well-suited for building RESTful APIs. This choice allows for handling multiple requests efficiently, which is crucial for a product display application.
   * **Database**: MongoDB was utilized for its flexibility in handling unstructured data. This is particularly useful for product listings, which may vary significantly in fields (e.g., some products may have different attributes). MongoDB's document-oriented structure simplifies storing and retrieving product data.
   * **Barcode Scanner**: I opted for QuaggaJS as it is a well-documented and widely used library for barcode scanning in web applications. It can easily integrate with React, providing a straightforward way to implement the bonus feature.
2. **Component Structure**:
   * The application was designed with a clear separation of concerns. The frontend is divided into components like **ProductList** and **ProductDetails**, ensuring that each component has a single responsibility. This modular approach enhances maintainability and scalability.
3. **RESTful API Design**:
   * The API was designed to follow REST principles, providing clear and predictable endpoints for fetching products and their details. This decision makes it easier for frontend developers to understand how to interact with the backend.

**Challenges Faced**

1. **Data Fetching and State Management**:
   * **Challenge**: Initially, managing the state for product data and handling asynchronous API calls was complex, especially when ensuring that the UI updated correctly after data fetching.
   * **Solution**: I implemented the **useEffect** hook in React to handle data fetching on component mount. This approach ensures that the product list is fetched once when the component loads, and I used **useState** to manage the product data efficiently.
2. **Error Handling**:
   * **Challenge**: Implementing error handling for API calls was initially overlooked, which led to unhandled promise rejections when the API was unreachable or returned an error.
   * **Solution**: I added try-catch blocks in the API call functions to catch errors and set appropriate error messages in the state, which can then be displayed in the UI to inform users of any issues.
3. **Database Population**:
   * **Challenge**: Pre-populating the database with sample products was challenging, especially in ensuring that the data format matched the expected schema.
   * **Solution**: I created a seed script that inserts sample product data into the MongoDB database. This script is run once to populate the database, ensuring that the application has data to display during development.

#### Conclusion

This project was an excellent opportunity to apply full-stack development skills, from designing a responsive UI to building a robust backend API. The challenges faced during development were valuable learning experiences that enhanced my problem-solving abilities. Overall, the design decisions made throughout the project contributed to a functional and user-friendly application that meets the assignment's objectives.