# Product Management System Documentation

## **Architecture Design**

### Backend Architecture

#### Microservices Approach

* **Microservices Architecture:** Implement a microservices architecture to achieve scalability, modularity, and independent deployment of product management functionalities.

#### Components

* **Product Service:** Manages CRUD operations for products.

#### Communication

* **RESTful APIs:** Utilize RESTful APIs for communication between services to ensure loose coupling and flexibility.

### Frontend Architecture

#### Single Page Application (SPA)

* **React.js:** Develop the frontend using React.js to create dynamic and responsive user interfaces.

#### Component-Based Architecture

* **Reusable Components:** Design and implement reusable components for product listing, product detail views, etc.

#### State Management

* **State Management Libraries:** Use state management libraries such as Redux for managing the application state.

## **Database Schema Design**

### Product Entity

#### Attributes

* **id:** Unique identifier for the product.
* **name:** Name of the product.
* **description:** Description of the product.
* **price:** Price of the product.
* **category:** Category to which the product belongs.
* **stockQuantity:** Quantity of the product in stock.
* **createdAt:** Timestamp indicating when the product was created.
* **updatedAt:** Timestamp indicating the last update time of the product.

#### Relationships

* **NoSQL Database (MongoDB):**
  + **Collections:** Use a single collection for storing products.
  + **Indexes:** Create indexes on id for quick lookup and createdAt for sorting.
  + **Schema Flexibility:** Leverage MongoDB's flexibility to accommodate changes in product attributes or schema evolution.

## **Technical Flow Diagram**

### Frontend (React)

1. **User Interface (UI) Components**
   * ProductList
   * ProductDetails
   * AddProduct
   * EditProduct
   * DeleteProduct
2. **React Router**
   * Handles routing between different pages (ProductList, ProductDetails, AddProduct, EditProduct).
3. **State Management**
   * Use useState and useEffect hooks.
   * Local component state for managing products, search terms, sorting, and pagination.
4. **Services**
   * productService: Manages API calls to the backend.
5. **Material-UI**
   * Utilizes UI components like Table, Button, TextField, etc.

### Backend (Spring Boot)

1. **Controller Layer**
   * **ProductController:** Manages incoming HTTP requests and maps them to the appropriate service methods.
2. **Service Layer**
   * **ProductService:** Contains business logic.
   * **ProductServiceImpl:** Implementation of ProductService.
3. **Repository Layer**
   * **ProductRepository:** Interface for CRUD operations with the MongoDB database.
4. **Model Layer**
   * **Product:** Entity representing a product.
5. **Database**
   * **MongoDB:** NoSQL database to store product data.

### Diagram Components

#### Frontend

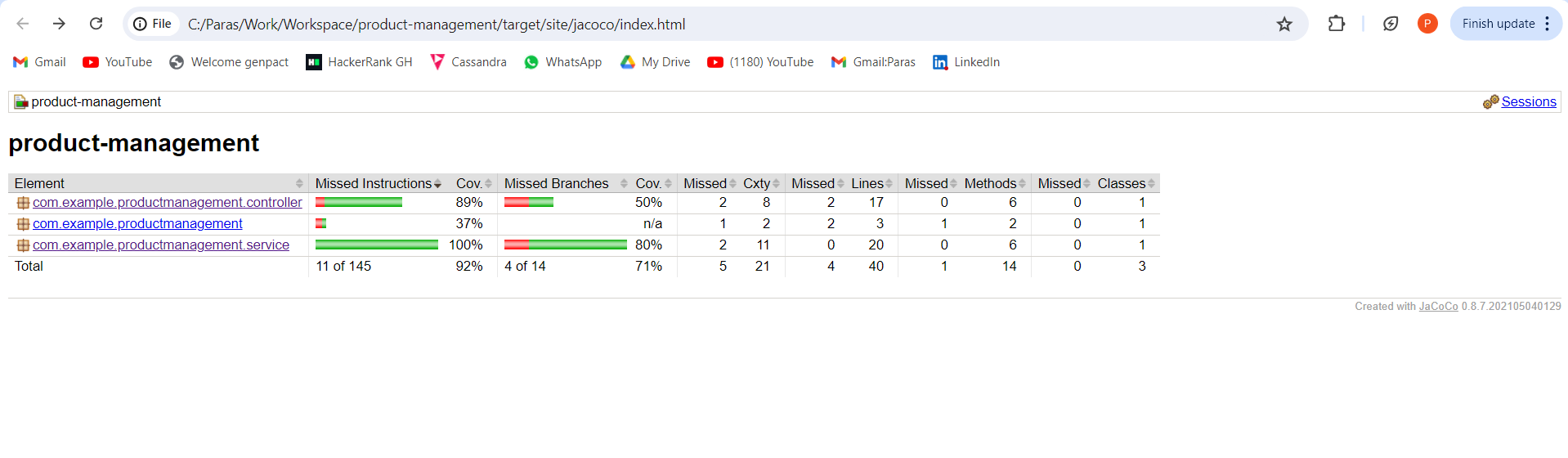
* **User Interaction** -> **UI Components** -> **React Router** -> **State Management** -> **Services** -> **API Calls to Backend**

#### Backend

* **API Requests** -> **Controller Layer** -> **Service Layer** -> **Repository Layer** -> **Database**

## **Testing**

* **Unit Tests:** Ensure comprehensive unit tests are written to cover all functionalities and branches, including null and non-null scenarios.
* **Integration Tests:** Implement integration tests to verify the correct interaction between different layers and components.
* **Test Coverage**



## **Integrations**

* **Product Analytics:** Integrate with open-source APIs or SDKs to add functionalities such as product analytics for better insights and reporting.