Introduction

Our e-commerce platform is designed to provide customers with a seamless online shopping experience for purchasing clothes. With a wide range of clothing options available, including men's, women's, and children's apparel, our platform aims to cater to diverse tastes and preferences. Through intuitive user interfaces and efficient backend systems, we aim to facilitate transactions and ensure customer satisfaction.

Stakeholders:

- User
- Admin

Functional Requirements Document (FRD):

1. Component Architecture:

- **Description:**
  - The project must adhere to a component-based architecture.
- **Functionality:**
  - Components are modularized and reusable, promoting code maintainability and scalability.
- **Requirements:**
  - Logical Structure: Components are organized logically based on their functionality and purpose.
  - Separation of Concerns: Each component has a clear separation of concerns, focusing on specific tasks or features
  - **Dependencies:**

2. Routing:

- **Description:**
  - Implement client-side routing to create a Single Page Application (SPA) experience.
- **Functionality:**
  - Define routes logically to match different sections or pages of the application.
  - Handle nested routes and redirects efficiently.

- **Requirements:**
  - Route Definition: Define routes using React Router with proper nesting and redirection.
  - Navigation: Enable users to navigate seamlessly between different sections of the application without full page reloads

3. Responsive Design:

- **Description:**
  - Ensure the application layout is responsive across various devices and screen sizes.
- **Functionality:**
  - Utilize CSS frameworks like Material-UI or React to facilitate responsiveness.
  - Test and optimize the layout for mobile devices, tablets, and desktops.
- **Requirements:**
  - Responsive Layout: Components adjust dynamically based on the screen size without sacrificing usability or functionality.
  - Compatibility: Ensure compatibility with popular web browsers and devices.

4. Checkout and Order Management:

- **Description:**
  - Users can proceed to checkout to finalize their purchases and place orders.
  - They can review their cart contents, apply discounts or promo codes, and select preferred shipping and payment options.
  - During checkout, users provide shipping and billing details, and they can choose to save this information for future use.
  - Upon successful completion of the checkout process, users receive confirmation of their order placement.
- **User Perspective:**
  - Checkout process is streamlined and user-friendly, minimizing friction and simplifying order placement.
  - Clear and transparent communication throughout the process ensures users are informed and confident in their transactions.

5. Product Management:

- **Description:**
  - Users can browse, search, and view detailed information about available products.
  - They can filter products based on various criteria such as gender
  - Product listings include images, descriptions, prices, sizes, and other relevant details.
  - Users can add products to their cart for purchase or save them for later.
- **User Perspective:**
  - Users can easily find and explore products that match their preferences and requirements.

- They can make informed decisions by accessing comprehensive product information.
- Browsing and filtering options streamline the shopping experience, enabling efficient product discovery.

Technical Requirements Document (TRD):

Architecture

The e-commerce application adopts a client-server architecture. The client-side is developed using React.js, while the server-side is implemented using Node.js with the Express.js framework. Communication between the client and server occurs through HTTP requests and responses.

Components

1. **Server**:
   - Implemented with Node.js and Express.js.
   - Handles HTTP requests, routing, and business logic.
2. **Database**:
   - MongoDB utilized as the database management system.
3. **Frontend**:
   - Client-side application developed using HTML, CSS, and JavaScript, with React.js as the primary JavaScript framework.
   - Utilizes React components for modular and reusable UI elements.
4. **Authentication**:
   - JWT (JSON Web Tokens) employed for user authentication and authorization.
5. **File Uploads**:
   - Multer library used for managing file uploads.
6. **Password Encryption**:
   - Bcrypt library utilized for hashing and salting user passwords to enhance security.
7. **API Documentation**:
   - Endpoints documented using Swagger or similar tools for clear API documentation.

Integrations

1. **MongoDB Atlas**:
   - Cloud-hosted MongoDB utilized for data storage, ensuring scalability and reliability.

Infrastructure Requirements

1. **Node.js Runtime Environment**:

    - Server configured with Node.js runtime environment for executing JavaScript code.
2. **Database**:
    - MongoDB database instance required, preferably hosted on MongoDB Atlas for cloud-based hosting.
3. **Third-party Services**:
    - Integration with external services necessitates appropriate API keys and credentials for seamless communication.

Development Tools

1. **Node.js**:
    - JavaScript runtime environment employed for server-side development.
2. **Express.js**:
    - Web application framework for Node.js utilized for building the server-side application.
3. **MongoDB**:
    - NoSQL database employed for data storage, providing flexibility and scalability.
4. **React.js**:
    - JavaScript framework utilized for building dynamic and interactive user interfaces on the client-side.
5. **Postman**:
    - API development and testing tool utilized for testing endpoints during the development phase.
6. **Visual Studio Code**:
    - Code editor employed for writing, editing, and debugging code efficiently.

APIs and Data Formats

1. **RESTful APIs**:
    - Follows REST principles for designing APIs, ensuring clear endpoints and using standard HTTP methods (GET, POST, PUT, DELETE).
2. **JSON**:
    - Data exchanged between the client and server in JSON format for ease of parsing and manipulation.

Database Schema Documentation:

Users Table

- **Description**: Represents users registered in the system.
    - **Columns:**
        1. `name` (String): Name of the user.
        2. `email` (String, unique): Email address of the user. (Unique constraint)
        3. `password` (String): Encrypted password of the user.
        4. `cartData` (Object): Data structure representing the user's cart.

5. `date` (Date): Date when the user was created. (Default: Current date)

Promotions Table

- **Description:** Stores information about promotions available in the system.
  - **Columns:**
    1. `code` (Number, required): Unique code assigned to the promotion. (Required constraint)
    2. `value` (Number, required): Value of the promotion.

Products Table

- **Description:** Contains details of products available for sale.
  - **Columns:**
    1. `id` (Number, required): Unique identifier for the product. (Required constraint)
    2. `name` (String, required): Name of the product. (Required constraint)
    3. `image` (String, required): URL of the product image.
    4. `category` (String, required): Category to which the product belongs.
    5. `new_price` (Number): Price of the product when it's new.
    6. `old_price` (Number): Previous price of the product.
    7. `date` (Date): Date when the product was added to the database. (Default: Current date)
    8. `avilable` (Boolean): Availability status of the product. (Default: true)
    9. `rating` (Number): Rating of the product.

API Documentation:

1. **`/upload` (POST):**
   - Description: Uploads an image file to the server.
   - Parameters: `product` (file) - Image file to upload.
   - Response: JSON object containing the success status and the URL of the uploaded image.
2. **`/login` (POST):**
   - Description: Logs in a user and generates an authentication token.
   - Parameters: `email` (string) - User's email, `password` (string) - User's password.
   - Response: JSON object containing the success status and the authentication token.
   - Sample Request:

```
{
  "email": "example@example.com",
  "password": "password"
}
```

- Sample Response:

```
{
  "success": true,
  "token": "<authentication token>"
}
```

3. **`/signup` (POST):**
   - Description: Registers a new user and generates an authentication token.
   - Parameters: `username` (string) - User's username, `email` (string) - User's email, `password` (string) - User's password.
   - Response: JSON object containing the success status and the authentication token.
   - Sample Request

```
{
  "username": "example",
  "email": "example@example.com",
  "password": "password"
}
```

   - Sample Response

```
{
  "success": true,
  "token": "<authentication token>"
}
```

4. **`addproduct` (POST):**
   - Description: Adds a new product to the database.
   - Parameters: Product details including `name`, `image`, `category`, `new_price`, `old_price`, `rating`.
   - Response: JSON object indicating success status and name of the added product.
   - Sample Request:

```
{
  "name": "New Product",
  "image": "image_url",
  "category": "Category",
  "new_price": 100,
  "old_price": 90,
  "rating": 4.5
}
```

- Sample response:

```
{
  "success": true,
  "name": "New Product"
}
```

5. **`/allproducts` (GET):**
   - Description: Retrieves all products from the database.
   - Response: JSON array containing all product objects.
6. **`/newcollections` (GET):**
   - Description: Retrieves a selection of new collection products.
   - Response: JSON array containing new collection product objects.
7. **`/popularinwomen` (GET):**
   - Description: Retrieves a selection of popular products in the women's category.
   - Response: JSON array containing popular women's products.
8. **`/addtocart` (POST):**
   - Description: Adds a product to the user's cart.
   - Parameters: `itemId` (number) - ID of the product to add to the cart.
   - Middleware: `fetchuser` - Fetches user data from the database using authentication token.
   - Response: Success message indicating the product was added to the cart.
9. **`/removefromcart` (DELETE):**
   - Description: Removes a product from the user's cart.
   - Parameters: `itemId` (number) - ID of the product to remove from the cart.
   - Middleware: `fetchuser` - Fetches user data from the database using authentication token.
   - Response: Success message indicating the product was removed from the cart.
10. **`/getcart` (POST):**
    - Description: Retrieves the user's cart data.

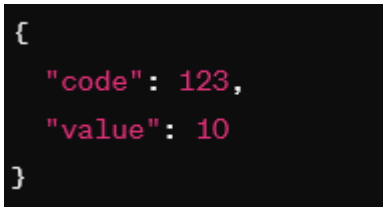
- Middleware: `fetchuser` - Fetches user data from the database using authentication token.
- Response: JSON object containing the user's cart data.

11. `/removeproduct` (DELETE):

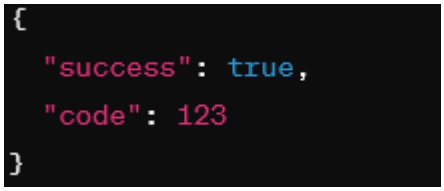
- Description: Removes a product from the database.
- Parameters: `id` (number) - ID of the product to remove.
- Response: JSON object indicating success status and name of the removed product.

12. `/addpromution` (POST):

- Description: Adds a new promotion to the database.
- Parameters: Promotion details including `code` and `value`.
- Response: JSON object indicating success status and code of the added promotion.
- Sample Request



- Sample Response



13. `/removepromution` (DELETE):

- Description: Removes a promotion from the database.
- Parameters: `code` (number) - Code of the promotion to remove.
- Response: JSON object indicating success status and code of the removed promotion.

14. `/allpromution` (GET):

- Description: Retrieves all promotions from the database.
- Response: JSON array containing all promotion objects.

15. `/updatepromution` (PUT):

- Description: Updates an existing promotion in the database.
- Parameters: Updated promotion details including `code` and `value`.
- Response: JSON object indicating success status and updated promotion details.