Design Document

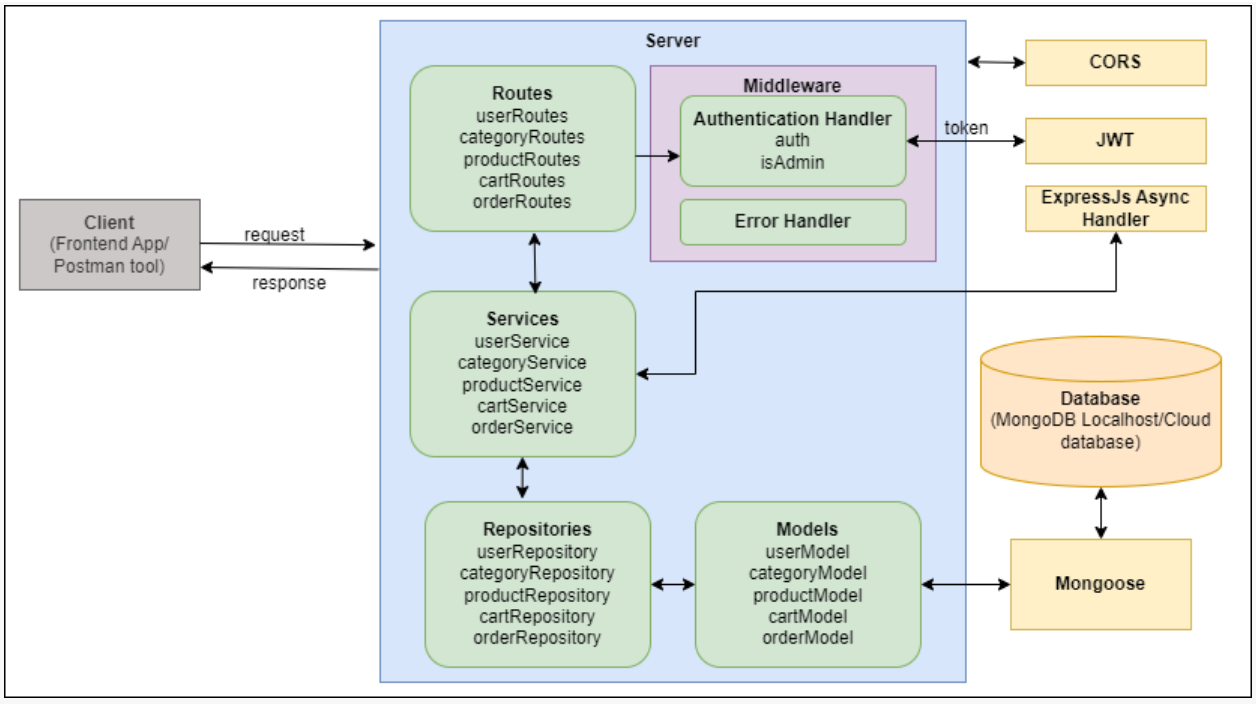
Ecommerce App Backend Server

Table of Contents

Introduction

This document outlines the design and architecture of the backend for an ecommerce application built with ExpressJS. It details the setup, routes, data models, and security measures implemented to support the application's functionality.

Architecture of Ecommerce Backend Server



The server application has various components including middleware for authentication and error handling, and a data handling flow from routes to database as follows :

- **Client**: Represents the external entity that makes requests to and receives responses from the server application.
- **Server Application**: The main block that encapsulates all server-side logic.
    - **Middleware**:
        - **Authentication Handler**: Manages authentication, verifying requests based on JWT tokens.
        - **Error Handler**: Manages error handling across the server application.
    - **Routes**: Defines various endpoints the server exposes to the client.
    - **Services**: Contains business logic that processes data, received from routes, before it's passed to repositories or vice versa.
    - **Repositories**: Acts as a data access layer, where the actual database CRUD (Create, Read, Update, Delete) operations are implemented.
    - **Models**: Represents the structure of the data in the database; this is where you define your data schemas.

- **JWT**: An external library that generates and validates JSON Web Tokens for authentication purposes.
- **CORS**: An external library that handles Cross-Origin Resource Sharing to allow secure communication between client and server.
- **Express-Async-Handler**: An external library that streamlines asynchronous operations and error handling within Express.js route handlers, enhancing code structure and readability.
- **Mongoose:** Facilitates data modeling and interaction with the MongoDB database, enhancing the backend architecture's efficiency and robustness.
- **Database**: The MongoDB database where data is stored. It can be hosted locally or in the cloud.

This architecture supports a clean separation of concerns, with each component focusing on a specific responsibility. The flow of data from routes to the database (and back) is a common pattern in web applications, facilitating maintainability and scalability.

Setup and Configuration

Prerequisites

Node.js installed
MongoDB for the database
Environment variables for database URI, secret keys, etc.

Installation

1. Clone the repository
2. Run **npm install** to install dependencies
3. Start the server with **npm start**
4. Access the server at **http://localhost:8080**

User Flow and Access Control

1. Create a user account (admin or non-admin).
2. Log in with the created user credentials.
3. Admin users have access to CRUD operations on users, categories, and products.
4. Non-admin users have restricted access and can perform cart and order related operations and fetch categories and products.
5. Once logged out, only operations that do not require authentication can be performed.

Sample Flow

1. Create a user using the HTTP POST request http://localhost:8080/api/v1/users with http body -

```
{
    "username":"john_d",
    "fullname" : "John Doe",
    "email" : "johnd@gmail.com",
```

```
    "isAdmin" : true,
    "password" : "johnd"
}
```

2. On successful creation of user, now login with user credentials using HTTP POST request http://localhost:8080/api/v1/users/login with http body -

```
{
    "username":"john_d",
    "password" : "johnd"
}
```

3. On successful login, copy the session token generated and paste it in the Authorization tab (Authorization Type should be Bearer Token) in the Token input field.
4. Now since you have successfully logged in the user credentials who is Admin so you should be able to get all users list or generate a new category or new product etc.
5. Refer to the endpoints mentioned below to perform all such operations and then you can give a logout request.

API Endpoints

User Routes

- **Register**: **POST /api/v1/users** - Creates a new user account.
- **Login**: **POST /api/v1/users/login** - Authenticates a user and returns a token.
- **Logout**: **GET /api/v1/users/logout** - Logs out a user, requiring token authentication.
- **Fetch User**: **GET /api/v1/users/:id** - Retrieves user details, requires token authentication.
- **Delete User**: **DELETE /api/v1/users/:id** - Removes a user account, requires token authentication and admin rights.
- **Edit User**: **PUT /api/v1/users/:id** - Updates user details, requires token authentication.
- **Fetch All Users**: **GET /api/v1/users** - Retrieves all users, requires token authentication and admin rights.

Category Routes

- **Create Category**: **POST /api/v1/categories** - Adds a new category, requires token authentication and admin rights.
- **Edit Category**: **PUT /api/v1/categories/:id** - Updates category details, requires token authentication and admin rights.
- **Delete Category**: **DELETE /api/v1/categories/:id** - Removes a category, requires token authentication and admin rights.
- **Fetch Category**: **GET /api/v1/categories/:id** - Retrieves category details.
- **Fetch All Categories**: **GET /api/v1/categories** - Lists all categories.

Product Routes

- **Create Product**: **POST /api/v1/products** - Adds a new product, requires token authentication and admin rights.
- **Edit Product**: **PUT /api/v1/products/:id** - Updates product details, requires token authentication and admin rights.

- **Delete Product**: **DELETE /api/v1/products/:id** - Removes a product, requires token authentication and admin rights.
- **Fetch Product**: **GET /api/v1/products/:id** - Retrieves product details.
- **Fetch All Products**: **GET /api/v1/products** - Lists all products.

Cart Routes

- **Add to Cart**: **POST /api/v1/carts** - Saves a product in the user's cart, requires token authentication.
- **View Cart**: **GET /api/v1/carts** - Shows user's cart items, requires token authentication.
- **Remove from Cart**: **DELETE /api/v1/carts/:id** - Deletes a product from the cart, requires token authentication.

Order Routes

- **Place Order**: **POST /api/v1/orders** - Submits a new order, requires token authentication.
- **User Orders**: **GET /api/v1/orders/user/:userId** - Retrieves a user's order history, requires token authentication.
- **All Orders**: **GET /api/v1/orders** - Lists order history of all users, requires token authentication and admin rights.
- **Change Order Status**: **PUT /api/v1/orders/:orderId** - Modifies the status of an order, requires token authentication and admin rights.

Data Models

User

- ID
- Name
- Email
- Password (hashed)
- Admin (boolean)

Category

- ID
- Name
- Description

Product

- ID
- Name
- Description
- Price
- CategoryID
- Stock

Cart

- ID
- UserID
- Products (Array of ProductID and Quantity)

Order

- ID
- UserID
- Products (Array of ProductID and Quantity)
- Status (Pending, Completed, Cancelled)
- TotalPrice

Security and Authentication

- Implement JWT for user authentication and authorization.
- Admin routes require token verification and an admin flag check.

Error Handling

- Implement global error handling middleware for catching and responding to errors.

Conclusion

This design document outlines the key components of the backend for an ExpressJS-based ecommerce application, including the setup, API endpoints, data models, and security considerations. This structure aims to provide a scalable, secure, and efficient backend system for managing ecommerce operations.