EKINOKS NODEJS CODING CHALLENGE

PROJECT DOCUMENTATION

Sadık Mahmut

October 4, 2023

Contents

1 Getting Started

1.1 Cloning the Repository

To begin working with the ShoppingApp project, you should first clone the repository from GitHub. Open your terminal or command prompt and execute the following command:

```
git clone https://github.com/sadikmahmut/ShoppingApp.git
```

This will create a local copy of the project on your machine.

1.2 Running the Application

Once you have cloned the repository, navigate to the project directory using your terminal or command prompt. To start the application and its associated services, such as the database, backend and frontend, run the following command:

```
docker-compose up
```

This command will orchestrate the deployment of the application and its dependencies using Docker Compose. Ensure that Docker and Docker Compose are installed on your system before running this command.

1.2.1 Accessing the Application

After the application is successfully started, you can access it through your web browser by navigating to the following URL:

```
http://localhost:4200
```

This will take you to the ShoppingApp frontend. You can also access the ShoppingApp backend by navigating to the following URL:

```
http://localhost:3000
```

1.2.2 Accessing the Database

If you need to interact with the PostgreSQL database directly, you can use a database client or tool of your choice. The database is accessible at the following connection details:

```
User: 'postgres',
Uost: 'postgres',
Database: 'ShoppingAppDb',
Password: '12345',
Port: 5432,
```

You can use these credentials to connect to the database and perform administrative tasks if necessary.

2 Tools & Technologies

These are the tools, technologies and libraries used throughout the project:

- **Backend:** Node.js, Express.js with TypeScript
- **Frontend:** Angular
- **Database:** PostgreSQL
- **Backend Testing:** Mocha, Chai
- **Containerization:** Docker
- **Version Control:** GitHub
- **API Testing:** Postman
- **Security:** bcrypt, jsonwebtoken
- **Utility Libraries:** uuidv4
- **Database Connection:** pg (PostgreSQL driver)

3 Postman Collection

You can find the Postman Collection file in the root directory by the name:
ShoppingApp.postman_collection

4 Automated Testing

Using Mocha and Chai unit testing frameworks, endpoints on Login, Order and User controller are tested. Here are the tested scenarios:

4.1 User Login

4.1.1 Scenario: Successful User Login

- **Test Case:** Verify that a user can successfully log in with valid credentials.
- **Test Steps:**
  1. Send a POST request to the '/api/Login/login' endpoint with valid login credentials.
  2. Expect a response with a 200 status code.
  3. Verify that the response contains a valid JWT token.
- **Expected Result:** The user should be able to log in successfully and receive a JWT token.

4.1.2 Scenario: User Login Failure

- **Test Case:** Verify that user login fails with invalid credentials.
- **Test Steps:**
  1. Send a POST request to the '/api/Login/login' endpoint with invalid login credentials (e.g., incorrect password).
  2. Expect a response with a 401 status code.
  3. Verify that the response contains an error message.
- **Expected Result:** User login should fail with an appropriate error message.

4.2 Order Management

4.2.1 Scenario: Retrieve All Orders

- **Test Case:** Verify that all orders can be retrieved.
- **Test Steps:**
  1. Send a GET request to the '/api/Order/getAllOrders' endpoint.
  2. Expect a response with a 200 status code.
  3. Verify that the response contains a JSON array of orders.
- **Expected Result:** All orders should be retrieved successfully.

4.2.2 Scenario: Retrieve Orders by User ID

- **Test Case:** Verify that orders can be retrieved by user ID.
- **Test Steps:**
  1. Send a GET request to the '/api/Order/getOrdersByUserId/:userId' endpoint with a valid user ID.
  2. Expect a response with a 200 status code.
  3. Verify that the response contains a JSON array of orders belonging to the user.
- **Expected Result:** Orders for the specified user should be retrieved successfully.

4.3 User Management

4.3.1 Scenario: Retrieve User Information

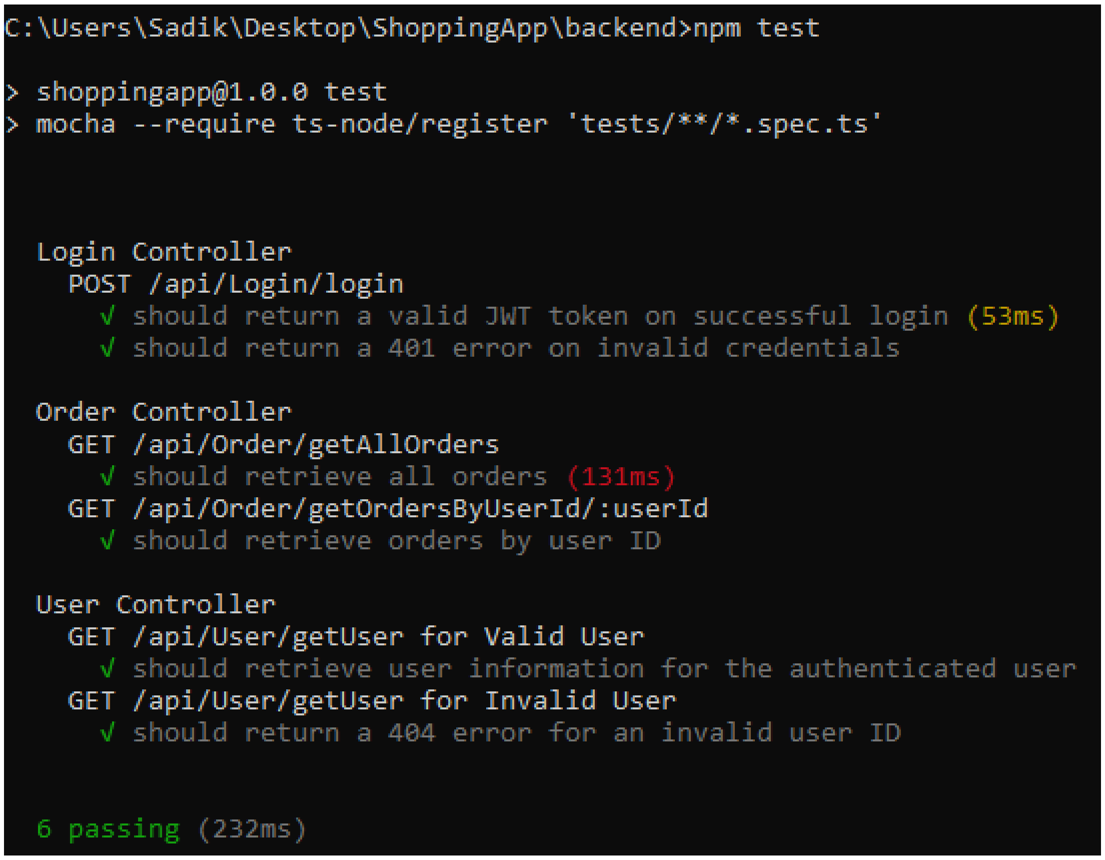
- **Test Case:** Verify that user information can be retrieved for the authenticated user.
- **Test Steps:**
  1. Send a GET request to the '/api/User/getUser' endpoint with a valid authentication token.
  2. Expect a response with a 200 status code.
  3. Verify that the response contains the user's information.
- **Expected Result:** The user's information should be retrieved successfully.

4.3.2 Scenario: User Information Retrieval Failure

- **Test Case:** Verify that user information retrieval fails without a valid authentication token.
- **Test Steps:**
  1. Send a GET request to the '/api/User/getUser' endpoint without a valid authentication token.
  2. Expect a response with a 401 status code.
  3. Verify that the response contains an error message.
- **Expected Result:** User information retrieval should fail with an appropriate error message.

Here are the test result after running the following command:

```
npm test
```



```
C:\Users\Sadik\Desktop\ShoppingApp\backend>npm test

> shoppingapp@1.0.0 test
> mocha --require ts-node/register 'tests/**/*.spec.ts'



  Login Controller
    POST /api/Login/login
      √ should return a valid JWT token on successful login (53ms)
      √ should return a 401 error on invalid credentials

  Order Controller
    GET /api/Order/getAllOrders
      √ should retrieve all orders (131ms)
    GET /api/Order/getOrdersByUserId/:userId
      √ should retrieve orders by user ID

  User Controller
    GET /api/User/getUser for Valid User
      √ should retrieve user information for the authenticated user
    GET /api/User/getUser for Invalid User
      √ should return a 404 error for an invalid user ID


  6 passing (232ms)
```

5 Endpoints

Here is a detailed description of the various endpoints available in the ShoppingApp project:

5.1 Register a New User

- **Endpoint URL:** http://localhost:3000/api/Register/registerUser
- **Description:** This endpoint allows users to register a new account by providing their personal information, including username, email, first name, last name, phone number, address, and password.
- **Authentication:** No authentication required.
- **Request Body:**
  - 'username' (string): The user's desired username.
  - 'email' (string): The user's email address.
  - 'firstName' (string): The user's first name.
  - 'lastName' (string): The user's last name.
  - 'phone' (string): The user's phone number.
  - 'address' (string): The user's address.
  - 'password' (string): The user's chosen password.
- **Response:** The newly created user object.
- **Possible Responses:**
  - 201 Created: Successful user registration.
  - 400 Bad Request: Invalid or missing user registration data.
  - 500 Internal Server Error: An error occurred during user registration.

5.2 Login User

- **Endpoint URL:** http://localhost:3000/api/Login/login
- **Description:** This endpoint is used for user login. Users provide their username and password, and if they match a user in the database, a JWT token is generated for authentication.
- **Authentication:** No authentication required.
- **Request Body:**
  - 'username' (string): The user's username.
  - 'password' (string): The user's password.

- **Response:** The JWT token for authentication.
- **Possible Responses:**
  - 200 OK: Successful login, returns a JWT token.
  - 401 Unauthorized: Invalid username or password.
  - 500 Internal Server Error: An error occurred during login.

5.3 Get All Orders

- **Endpoint URL:** http://localhost:3000/api/Order/getAllOrders
- **Description:** This endpoint retrieves all orders from the database, including details about the associated user and product.
- **Authentication:** Requires JWT token authentication.
- **Response:** An array of order objects.
- **Possible Responses:**
  - 200 OK: Successful retrieval of orders.
  - 500 Internal Server Error: An error occurred during the database query.

5.4 Get Orders by User ID

- **Endpoint URL:** http://localhost:3000/api/Order/getOrdersByUserId/:userId
- **Description:** This endpoint retrieves orders for a specific user based on their user ID.
- **Authentication:** Requires JWT token authentication.
- **Request Parameter:**
  - ‘userId‘ (string): The user’s ID.
- **Response:** An array of order objects.
- **Possible Responses:**
  - 200 OK: Successful retrieval of user’s orders.
  - 500 Internal Server Error: An error occurred during the database query.

5.5 Get User Information

- **Endpoint URL:** http://localhost:3000/api/User/getUser
- **Description:** This endpoint retrieves user information for the authenticated user.
- **Authentication:** Requires JWT token authentication.

- **Response:** User information including ID, username, email, first name, last name, phone, address, and role.
- **Possible Responses:**
  - 200 OK: Successful retrieval of user information.
  - 404 Not Found: User not found.
  - 500 Internal Server Error: An error occurred during the database query.

5.6 Get All Active Products

- **Endpoint URL:** http://localhost:3000/api/Product/getAllProducts
- **Description:** This endpoint retrieves all active products from the database.
- **Authentication:** Requires JWT token authentication.
- **Response:** An array of active product objects.
- **Possible Responses:**
  - 200 OK: Successful retrieval of products.
  - 500 Internal Server Error: An error occurred during the database query.

5.7 Get Product by ID

- **Endpoint URL:** http://localhost:3000/api/Product/getProduct/:id
- **Description:** This endpoint retrieves an active product by its ID.
- **Authentication:** Requires JWT token authentication.
- **Request Parameter:**
  - ‘id‘ (string): The product’s ID.
- **Response:** The product object.
- **Possible Responses:**
  - 200 OK: Successful retrieval of the product.
  - 404 Not Found: Product not found.
  - 500 Internal Server Error: An error occurred during the database query.

5.8 Create a New Product

- **Endpoint URL:** http://localhost:3000/api/Product/createProduct
- **Description:** This endpoint allows the creation of a new product.
- **Authentication:** Requires JWT token authentication.

- **Request Body:**
  - ‘Name‘ (string): The name of the product.
  - ‘Description‘ (string): The description of the product.
  - ‘Price‘ (number): The price of the product.
- **Response:** The newly created product object.
- **Possible Responses:**
  - 201 Created: Successful product creation.
  - 400 Bad Request: Invalid or missing product data.
  - 500 Internal Server Error: An error occurred during product creation.

5.9 Update Product Information

- **Endpoint URL:** http://localhost:3000/api/Product/updateProduct/:id
- **Description:** This endpoint allows updating product information based on its ID.
- **Authentication:** Requires JWT token authentication.
- **Request Parameter:**
  - ‘id‘ (string): The product’s ID.
- **Request Body:** Fields to be updated, including ‘Name‘, ‘Description‘, and ‘Price‘.
- **Response:** The updated product object.
- **Possible Responses:**
  - 200 OK: Successful product update.
  - 400 Bad Request: Invalid or missing product data.
  - 404 Not Found: Product not found.
  - 500 Internal Server Error: An error occurred during product update.

5.10 Delete Product

- **Endpoint URL:** http://localhost:3000/api/Product/deleteProduct/:id
- **Description:** This endpoint allows the deletion of a product based on its ID.
- **Authentication:** Requires JWT token authentication.
- **Request Parameter:**
  - ‘id‘ (string): The product’s ID.
- **Response:** A success message.

- **Possible Responses:**
  - 204 No Content: Successful product deletion.
  - 404 Not Found: Product not found.
  - 500 Internal Server Error: An error occurred during product deletion.