Backend Development

EP - Course Assignment Resources

Exam Project

Read the course assignment instructions carefully. If you should not understand anything, please contact one of the teachers.
You are not allowed to discuss the course assignment in any class channels.
The course assignment grade is graded.

F - 0-39 - Failed the course and have to do a Resit.
E - 40-49
D - 50-59
C - 60-79
B - 80-89
A - 90-100

! IMPORTANT!: Complete the Course Assignment by following the instructions given below.
Your GIT repository in this Assignment needs to be private.
Make sure to give **'noroff-bed1'** access to your repository in GitHub.

! IMPORTANT: Access to your repository cannot be given AFTER the deadline. If no access is given, this will result in an immediate Not Passed grade.
***No commits or submissions** past the deadline time will be gradable.*

A customer is wanting a Stock-control and Sales system for their warehouse, which sells items direct to customers from an item catalog. They require you to develop **only the back-end system**, to be used with a front-end designed by a separate UX design company to host the front end. The front end will access the back-end system with the API design below. The back-end system will be run on the customers' hardware at this premises.

The back-end system needs to have the following components:

- A database (MySQL)
- API endpoints for all database tables (CRUD)
- Authentication
- Unit testing

For this project, "in-stock" refers to items that are available for purchase, while "out-of-stock" refers to items that cannot be purchased at the moment:

- In-stock means that stock quantity is greater than zero.
- Out-of-stock means that stock quantity is equal to zero.

Customer Requirements

Authentication:
Your back-end_should implement Basic authentication, having multiple User types:

- Guest User (A User that uses the web application without Logging-in)
- Registered User (A User that has registered and Logged-in to the web application)
- Admin User (A User that is registered with the /setup API endpoint. I.e., Cannot be registered through the API /signup)
    - There **can only be 1x Admin User** with the 'Admin' role.

Login/Registration:
Guest Users should be able to register on the application. The required information for site registration includes a unique username, password, and email address. Up to 4 different users can register with the same email address - This is to allow family members to register themselves with a shared email address.

The API endpoint must not allow the registration of duplicate usernames.
If a username exists, an informative error message must be sent from the API.
Login credentials should be stored in the Database.

Items Catalogue:
On the Items Catalogue page, items for sale are shown, with their name, price, and stock levels.
Guest Users can only view in-stock items.
Registered users can view items and add them to their cart to check out once they have all the items they want.

The Admin User should have access to **CRUD** operational functionality, including Adding, Updating or Removing Items from the Items Catalogue.

All User types can **filter/search** the items seen in the catalog, by typing the item's name into a search bar (This would result in only showing items that match the search criteria).

Purchasing Items:
Registered Users can add items to their Cart that they wish to purchase.
A User can only have **one Cart.**
Out-of-stock items cannot be ordered.
Registered Users who have added in-stock items to their Cart, can '**Checkout**' their Cart.

Users with the **same email address** will be given one of the following **Discounts** when Checking-out their Cart:

- If 2 users have the same email address, both will get a 10% discount on orders.
- If 3 users have the same email address, those 3 users will get a 30% discount on orders.
- If 4 users have the same email address, all 4 users will get a 40% discount on orders.

The Discount will be reflected in the Cart total price, shown to the User when Checking-out.
After an order has been Checked-out, the Admin User can either '**Complete**' or '**Cancel**' that order.

Orders:
Registered Users that have Checked-out their Cart and have had their order 'Completed' by an Admin User can **view their Orders**.
Any **previous orders** that have already been fulfilled are also viewable.
All orders should have a **status** that is shown to the User (E.g., In Progress, Completed, Cancelled, etc)

-

Project Specifications

As this is a Backend Program, a **front-end is not required for this Exam Project.** If one is submitted, **it will not be graded**.
GitHub must be used as the version control system during the development of this project.
Jira must be used as the project management software during the development of this project.

Database
A MySQL Database called "**StockSalesDB**" is to be created for this web application.

Use the following **SQL script to** create an "admin" Database User with all database privileges:

CREATE USER 'admin'@'localhost' IDENTIFIED WITH mysql_native_password BY 'P@ssw0rd';
ALTER USER 'admin'@'localhost' IDENTIFIED WITH mysql_native_password BY 'P@ssw0rd';
GRANT ALL PRIVILEGES ON database_name.* TO 'admin'@'localhost';

The database must be designed in the **3rd normalized form** for all tables and relationships. Besides the initial Database creation, all database operations should be done through the **ORM – Sequelize**.

The Database should include, as a minimum, the following tables:

- Users
- Roles
- items
- categories
- carts
- cart items
- orders
- order items

The mentioned tables must include relevant columns and data types and should include `created_at` and `updated_at` timestamps where relevant. By including `created_at` and `updated_at` timestamps, we can track when each record was created and last updated, which can be useful for auditing purposes and for tracking changes over time.

All passwords in the database should be **hash encrypted.**

Authentication:
Authentication should be implemented with JWT. The JWT token should expire in 2 hours. Other than Logging-in or Registering, **no User credentials should be sent in the URL Path of any API endpoint.**

API Requirements

All the following API endpoints should be present in the project and should return JSON objects with valid status codes.

Other than the few endpoints that specify an :id is sent through the URL Path, ALL parameters must be sent through the Request Body.

Authentication endpoints

POST /login This endpoint will be used so that Registered Users can login. Users with a valid username and password should receive a JWT token. For invalid logins, a specific error should be returned. With a valid login where the username and password match, a valid JSON object with a token should be returned from the API.

POST /signup This endpoint will be used to register new users. A specific error message should be returned if invalid information is posted to the endpoint.

Items endpoints

GET /items This endpoint should return all items in the database. This endpoint should not be protected with authentication so Guest Users can view the items. This endpoint should return all items with their specified categories.

POST /item This endpoint should only be accessible as an Admin User. This endpoint will be used to add a new item to the database. This endpoint should receive the item information with the specific category id. The API should update the items table if a valid object has been received.

PUT /item/:id This endpoint should only be accessible as an Admin User. This endpoint will update an existing item in the database when a valid id has been provided. When an invalid id has been provided, a specific error message should be returned. This endpoint should be able to update an item's properties, including the category. NOTE that new categories cannot be created when updating catalog items.

DELETE /item/:id This endpoint should only be accessible as an Admin User. This endpoint will delete an item from the database if a valid id has been provided. When an invalid id has been provided, a specific error message should be returned.

Category endpoints

GET /categories This endpoint should return all categories in the database. This endpoint should not be protected with authentication so Guest users can view the categories. This endpoint should return all categories in the database.

POST /category This endpoint should only be accessible as an Admin User. This endpoint will be used to add a new category to the database. If any error occurs, a specific error message should be returned.

PUT /category/:id This endpoint should only be accessible as an Admin User. This endpoint will update the category name with a new name. The endpoint should only accept a new name along with the id that is provided in the URL. If any error occurs, a specific error should be returned.

DELETE /category/:id This endpoint should only be accessible as an Admin User. This endpoint will delete a category from the database if a valid id has been provided. When an invalid id has been provided, a specific error message should be returned. A Category can only be deleted if it contains no items in the database.

Cart endpoints

GET /cart This endpoint should be accessible when a Registered User has logged in. This endpoint should only return cart for the logged-in user. The user information should be extracted from the JWT. No user information should be sent with the API request.

GET /allcarts This endpoint should only be accessible by the Admin user. This endpoint should return all carts that exist, including the items in those carts and the full names of the users to whom those carts belong. **The Sequelize.query() and a raw SQL query MUST be used for this endpoint** (Pay attention to the necessary JOINS needed).

POST /cart_item This endpoint should be accessible when a Registered User has logged in. This endpoint allows Users to add items they want to purchase to their cart. The item placed in the cart should reference the item in the items table (relationship); however, the purchase price should be placed in the cart item table.

PUT /cart_item/:id This endpoint should be accessible when a Registered User has logged in. This endpoint changes the desired quantity of a specific item in for the logged in user’s cart. The item's id should be provided with a new desired purchase quantity. When a cart item's desired purchase quantity is increased, the back-end must ensure enough stock of the item in the items table. The item’s stock quantity should only be updated once the order has been placed.

DELETE /cart_item/:id This endpoint should be accessible when a Registered User has logged in. This endpoint should delete an item from the cart of the specific user only. The item id should be used.

DELETE /cart/:id This endpoint should be accessible when a Registered User has logged in. This endpoint should delete all items from the cart of the specific user. The cart id should be used.

Orders endpoint

GET /orders This endpoint should be accessible when a Registered User has logged in. This endpoint should only return orders for the logged in user. The user information should be extracted from the JWT. For the Admin User, this endpoint should return ALL orders.

GET /allorders This endpoint should only be accessible by the Admin user. This endpoint should return all orders that exist, including the items in those orders and the full names of the users to whom those orders belong, regardless of order status. **The Sequelize.query() and a raw SQL query MUST be used for this endpoint** (Pay attention to the necessary JOINS needed).
POST /order/:id This endpoint should be accessible when a Registered User has logged in. The order item in the order table should reference the item in the items table, however the price of the item should be in the order items table. Once this has been done, the item's stock level should be adjusted in the items table. If one item has been ordered, the item in the item table should be deducted by one. If a quantity of two has been ordered of the same item SKU, then the item in the items table should be deducted with two and so on. Checks should be made to ensure there is enough stock i.e. the stock quantity of the item is enough to honor the order.

PUT /order/:id This endpoint should only be accessible as an Admin User. This endpoint is used to update the order status. The Admin User should be able to update ANY user's order status. The valid status values are In Process, Complete, Cancelled

Utility endpoints
POST /setup This endpoint is used for the initial database population. This endpoint should only populate the database if no records exist in the items table.
The initial data can be obtained with an API GET call to the Noroff API (http://143.42.108.232:8888/items/stock). Pay attention to the categories and populate the items and categories tables. When this is done with no errors, a JSON success should be returned. If any error occurs, a specific error message should be returned. If there are items in the items table, then this API call should return a message stating that it did not populate the database and the reason for not doing so.

This API endpoint should do the following only if there is no data in the items table:

- Populate the items table with the data from the Noroff API
- Populate the category table with the data from the Noroff API (Pay attention to the relationship)
- Populate the roles table with two roles:
  - 1 – Admin
  - 2 – User
- Create an Admin user in the Users table with:
  - Username: Admin
  - Password: P@ssword2023

POST /search This endpoint **should not be protected** with authentication. This endpoint is used to search for items in the database. Depending on the search criteria, items or categories should be searched from the database and returned as a JSON object. The search endpoint should, as a minimum, be able to

- Search a partial item_name (it does not need to be the full item name)
- Search for a specific category name
- Search for a specific SKU (product code)
- Search for a partial item_name for a specific category name

The search must be done from the database, either with a SQL query or Sequelize's search functionality. If any error occurs, a specific message should be returned.

Unit Testing

The Jest testing framework and Supertest library should be used to create and run **the following CRUD tests** on the created API endpoints:

1. post /setup endpoint - Checks if database is populated, makes API call to Noroff API, and populates empty database.
2. post /signup endpoint – registers a new user.
3. post /login - use the token from the user created in test 1.
4. post /category - create a new category with the name CAT_TEST.
5. post /item - create a new item with the CAT_TEST category and the ITEM_TEST item name.
6. post /search - search for items with the text "mart "in the item name (three items should be returned from the initial data).
7. post /search – search for all items with the category name "Laptop" (one item should be returned from the initial data).
8. Test the Admin user endpoints with the user created in test 1 (at least 3 endpoints should be tested).
9. Delete all the values added to the database in the previous tests (CAT_TEST, ITEM_TEST, and the user created)**.**
10. post /setup endpoint - Test must be run again, but should not make the API call or populate the database. Should return relevant message.

Acknowledgment

Students must indicate where they received help or used outside knowledge for their Exam Project. Only students working together in a mentor group can help each other. **This must be indicated in the README file of the project**.
Each project is checked for plagiarism using various tools and software, so this step is VERY IMPORTANT!

This would include:

- Acknowledgments of any help received from other students (If the student is working in a mentor group)
- Any code or knowledge sourced from internet forums, textbooks, AI-generated code, etc.

Documentation

Once the project is complete, all Documentation for this project must be bundled into a **single PDF file** in the folder documentation containing the following:

- Detailed instructions on how to install and run the back end must be included in the Readme file
- Postman Documentation
  - Including all API Endpoints (Endpoint path, description, JSON objects)
- Retrospective Report:
  - Screenshot of the complete Database ERD (Showing all tables, relationships, properties, etc.)
  - An explanation of the relationships between tables (e.g., An X has many Y's therefore, this is a one-to-many relationship)
  - Screenshot of only the Jira Roadmap (Showing Epics and Sprints)
  - Project Retrospective write-up including the following sections:
    - Discussion of the progression of the project.
    - Challenges faced during the development of this project.

Submission

The README file must be committed to your GitHub repository and must include:

- A copy of the configuration of your .env file
- Access to your GIT repository must only be given to "**noroff-bed1**" BEFORE the deadline.

A .txt file needs to be submitted on Moodle - This file must include the link to your GitHub repository.
This .txt file should be named the following:
"FName_LName_EP_CA_ClassXXFT.txt" - if you are a Full-Time student,
"FName_LName_EP_CA_ClassXXPT.txt" - if you are a Part-Time student

(Replace 'FName' with your First Name, and 'LName' with your Last Name).
(Replace 'Class' with your class, e.g. 'Aug', 'Oct', etc).
(Replace 'XX' with your class year e.g. 22, 23).

EXAMPLE: John_Smith_EP_CA_AUG22FT.txt

NOTE: This .txt file is the ONLY Moodle submission for this Course Assignment.

Previous Activity

Fortsett til...

Next Activity