Exam Project 1

Read

Please read the following instructions carefully. If any part is unclear or you have questions, contact the designated teachers through Microsoft Teams direct message. **Discussing the course assignment in class channels is strictly prohibited.** Additionally, any discussions about the Exam Project outside of official teacher-student communications, including private messages or in-person conversations with classmates, are not allowed.

If you struggle to understand the Exam Project instructions, contacting your teachers for help is perfectly acceptable. Please understand that the teachers cannot give students the answers or step-by-step instructions on implementing course assignment requirements, as the Course Assignments, Projects, and Exams are graded assessments. Teachers can, however, clarify **Exam Project instructions** if needed.

The Exam Project will be **graded "A to F".**

- 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

The following is tested in the Exam Project:

- Database design/queries
- API design / Testing / Documentation
- Documentation / Planning
- Logical thinking and problem-solving skills
- Basic front-end skills

Repository

The **Git Classroom link** for this assignment will be posted in your **class channel on Microsoft Teams** on the day the assignment opens. Expect the link by 09H00 when the assignments open.
Once you accept this assignment, your private GIT repository will be created, accessible only by you and the course instructors and graders.

Only commits made in the Git Classroom will be considered for grading. Any repositories you create outside the provided Git Classroom link will not be graded. Ensure all your work is committed **before the course assignment deadline. After the deadline,** you cannot commit any more code to your repository or Git classroom. Any attempts to commit after the deadline will be automatically rejected.

Only the **main** branch will be graded. When you have completed the project, ensure your working code is in the main branch and that you submit it before the deadline.

VIDEO LINK: Git classroom informative video

Internships Projects Only

Create your GitHub account and give access to **"noroff-bed1"** before the deadline for the exam project.

Internship projects are the ONLY ones that do not use Git Classroom.

Internship project students must submit their text file as explained in the submission section

Access to your repository after the deadline will not be graded and will result in an F grade.

Submission

Remember to create a .gitignore file and exclude the relevant files or folders.
Your README file must be committed to the GitHub Classroom repository and must include the following:

- An example of the configuration of your .env file
- Detailed instructions on how to run your application
- The version of NodeJS and any other plugins used for your application

Your application must be pushed to your GitHub Classroom repository **BEFORE** the course assignment deadline.
Moodle submission:

- The text file must include the link to your **repository and your GitHub username**
- This .txt file must be submitted on Moodle
- The file must be named as follows **"FName_LName_EP1_CA_ClassXXYY.txt"**

Your reflection **PDF** report must be in your repository in the folder **Documentation**.

(Replace 'Class' with your class, e.g. 'Aug', 'Oct', etc)
(Replace 'XX' with your class year e.g. 22, 23)
(Replace 'YY' with either FT for full-time, or PT for part-time)
EXAMPLE: JOHN_DOE_EP1_CA_JAN23FT.txt

Commits or submissions past the deadline will not be considered for grading.
Late submissions will not be accepted, and there will be no exceptions to this rule.

Failure to submit this file to Moodle OR push (commit) your code to the git classroom will result in a not passed grade

Noroff EP e-commerce

A new customer with an existing e-commerce site using static data wants to convert their website to instead use a back-end system that includes APIs and a database to showcase their products for customers to purchase. They also require an Admin front-end, which uses the back-end system to manage the data .

Front-end developers have started to change the website, but the process is still ongoing. The front-end developers created a database with mock data for them to test with. However, the current database design is not in the correct format. They created one table for all the product information. The database is unavailable, but the Noroff API can access the data. (see Initialization endpoint)

NOTE: The back-end and front-end applications must run on separate ports. The back-end and front-end are separate applications.

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

- A database (MySQL) designed in third normal form (3NF)
- API endpoints for all relevant database tables (CRUD) with authentication/registration for users, and protected routes for adding/editing data for the admin user role
- API Swagger Documentation
- A separate Admin front-end interface. The API endpoints of your back-end application must be used, no service files must be used in the front-end. Only the back-end API endpoints must be used in the front-end
- Testing - Documentation - Good code practice (DRY - Don't Repeat Yourself)

Think about:

When a product is added to a user's cart and checked out, if the admin user changes the price of the item, the price of the item in the order should not be changed. The original price of the product should be included in the order.

Planning:

Use Jira to plan your project. In the documentation section, you will need to provide screenshots of your planning.

At the end of the project, documentation must be provided. This will include a reflection report. Keep this in mind while planning and creating your project.

Back-end Requirements

Database

You must design a new MySQL database that you will use for your back-end API. The database must be designed in the third normal form. The 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 tracking of changes over time.

Besides the initial database creation, all database operations and queries must be done through the ORM **Sequelize.**

A service file must be created per table that requires database interaction.

API Specification

Your project must include all required endpoints specified in each section. If your project requires additional endpoints, you may add them.
All API endpoints must return valid status codes to JSON Objects for the back-end. Error handling and validation must be implemented in each endpoint.
Route protection/authentication/admin checks must be **implemented with middleware** for all required routes.

Authentication should be implemented with **JWT**. The JWT token should expire in 2 hours. All relevant logged-in user information must be extracted from the JWT.
Other than when Logging in or Registering, **User credentials or information must not be sent in the URL Path or body as parameters** of an API endpoint.

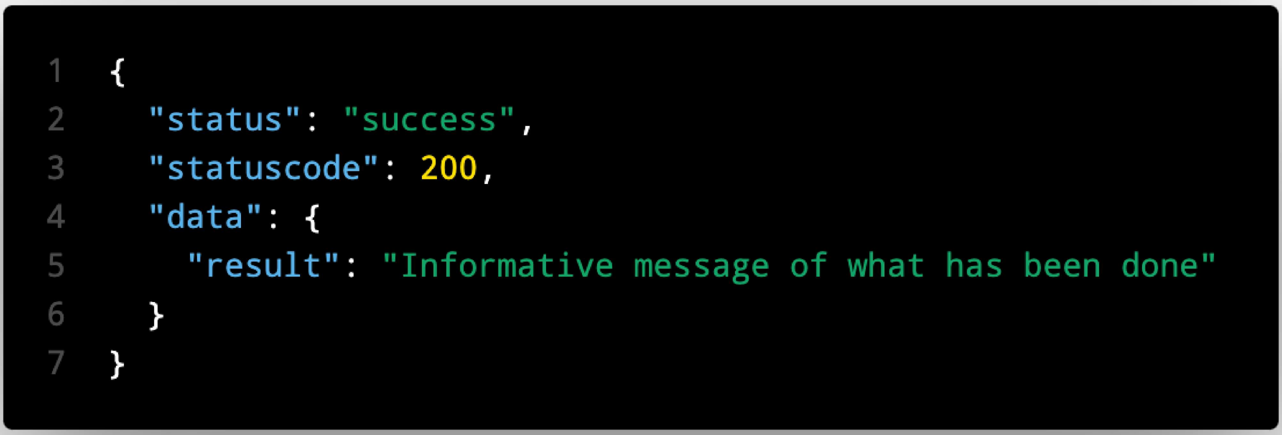
an API endpoint.

Error handling must be implemented for all endpoints. (The back-end must not crash if something is not correct.)

NOTE: More properties can be added if required

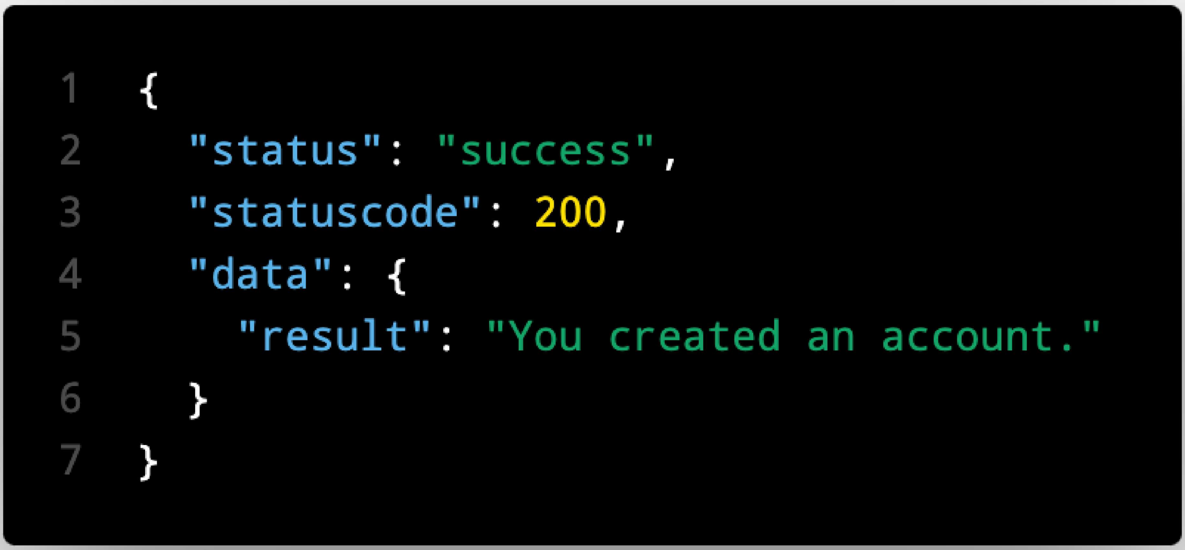
The base JSON response structure must be used for each endpoint

- The base JSON return structure



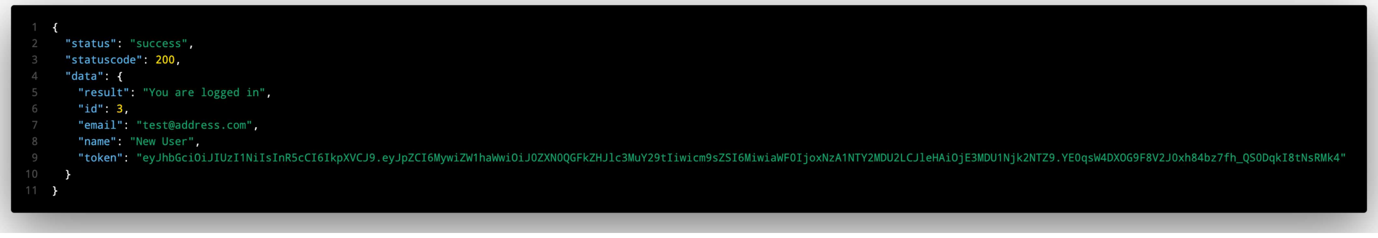
```json
{
  "status": "success",
  "statuscode": 200,
  "data": {
    "result": "Informative message of what has been done"
  }
}
```

- Success



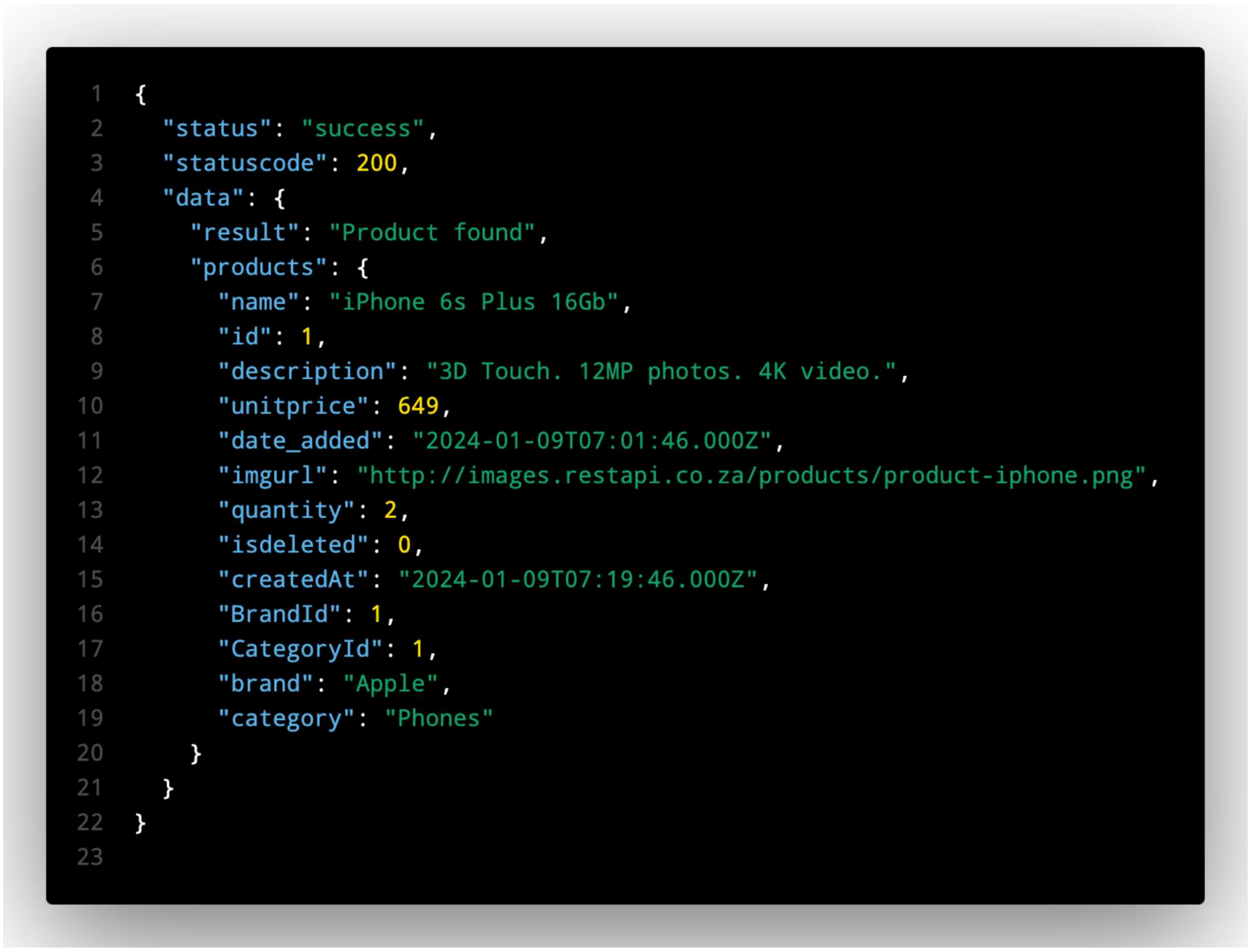
```json
{
  "status": "success",
  "statuscode": 200,
  "data": {
    "result": "You created an account."
  }
}
```

- Example of a user that logged in with extra information



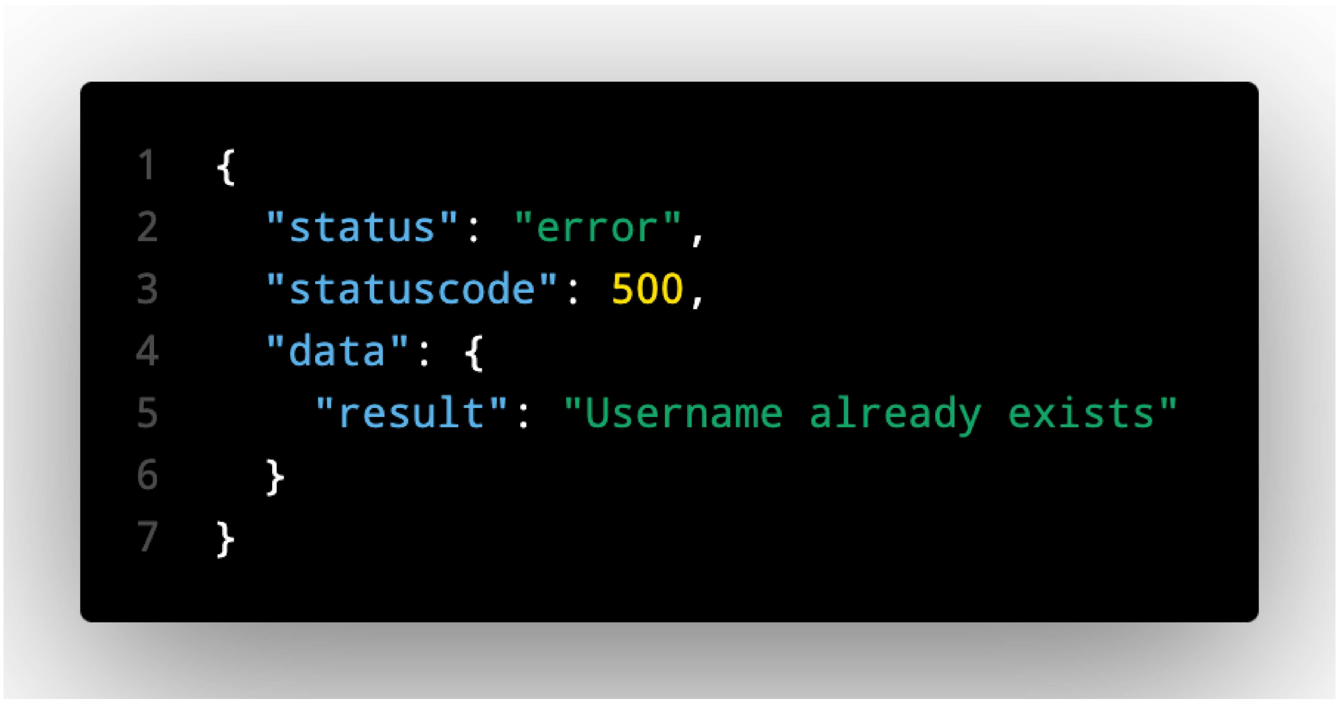
```json
{
  "status": "success",
  "statuscode": 200,
  "data": {
    "result": "You are logged in",
    "id": 3,
    "email": "test@address.com",
    "name": "New User",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiZW1haWwiOiJ0ZXN0QGFkZHJlc3MuY29tIiwicm9sZSI6MiwiaWF0IjoxNzA1NTY2MDU2LCJleHAiOjE3MDU1Njk2NTZ9.YE0qsW4DXOG9F8V2J0xh84bz7fh_QS0DqkI8tNsRMk4"
  }
}
```

- Product success return JSON



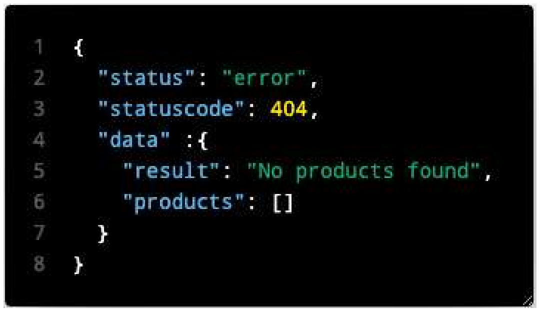
```json
{
  "status": "success",
  "statuscode": 200,
  "data": {
    "result": "Product found",
    "products": {
      "name": "iPhone 6s Plus 16Gb",
      "id": 1,
      "description": "3D Touch. 12MP photos. 4K video.",
      "unitprice": 649,
      "date_added": "2024-01-09T07:01:46.000Z",
      "imgurl": "http://images.restapi.co.za/products/product-iphone.png",
      "quantity": 2,
      "isdeleted": 0,
      "createdAt": "2024-01-09T07:19:46.000Z",
      "BrandId": 1,
      "CategoryId": 1,
      "brand": "Apple",
      "category": "Phones"
    }
  }
}

```

- Error example



```json
{
  "status": "error",
  "statuscode": 500,
  "data": {
    "result": "Username already exists"
  }
}
```

- Product error JSON example



```
{
  "status": "error",
  "statuscode": 404,
  "data" :{
    "result": "No products found",
    "products": []
  }
}
```

Login and Registration

General information

- Guest Users are any users that have not registered.
- Guest Users should be able to register.
- A Guest user is anonymous.
- The "Guest" user role should not be specified in the database (As the user would not be registered)
- Guest users cannot add anything to their cart or buy any items.
- Guest users can view and search for products
- Registered users will have carts and orders.
- Login credentials should be stored in the Database, and the password must be **hashed**.
- **Validation** must be done for the email address (Ensure it has the correct email format)
- **Error handling / Validation** must be done to ensure all required properties have been included and are valid when a user registers

Each user has a membership status based on how many items have been purchased - the total quantity of items purchased. A newly registered member has a default Bronze membership status. The membership status will give users a discount based on the number of items purchased (the total quantity).

The following specified user roles must exist:

- User - Registered User (A user that has registered and Logged-in to the web application)
- Admin - Admin User (A User that is created with the /init API endpoint. i.e., A user cannot be registered as the Admin role through the API /auth/register endpoint)

Roles have the following permissions:

- Admin user roles can change registered/signed-up user roles to admin from the back-end with the admin user interface.
- Admin user roles can add/edit/delete records. (The admin role cannot change Orders, except the status of the order)
- When a user registers, their default role must be User.

NOTE: Some routes must only be accessible to an Admin role.

The specified endpoints will need CRUD. You are free to add CRUD for tables that require it but are not specified. Endpoints not described here but that are required can be added to the course assignment.

Registration endpoint

Minimum required endpoint

- POST /auth/register

This endpoint will register new users. Each new user should provide a unique username and email address. Users with a valid username or email and password should receive a JWT token. This means that if the user's email and password OR username and password are correct, they can log in.

For invalid logins, a specific error should be returned. With a valid login where the username, email, and password match, a valid JSON object with a token should be returned from the API.

A new user must provide the following information to register on the system successfully

{ "firstname" : "John", "lastname" : "Doe", "username" : "user123" (unique), "email" : "johndoe@email.com" (unique), "password" : "password", "address" : "123 Street, City", "phone" : "123456789" }

Login endpoint

The minimum required endpoint for registering a new user

- POST /auth/login

This endpoint will be used for users who have already registered to log in to the system.

Products

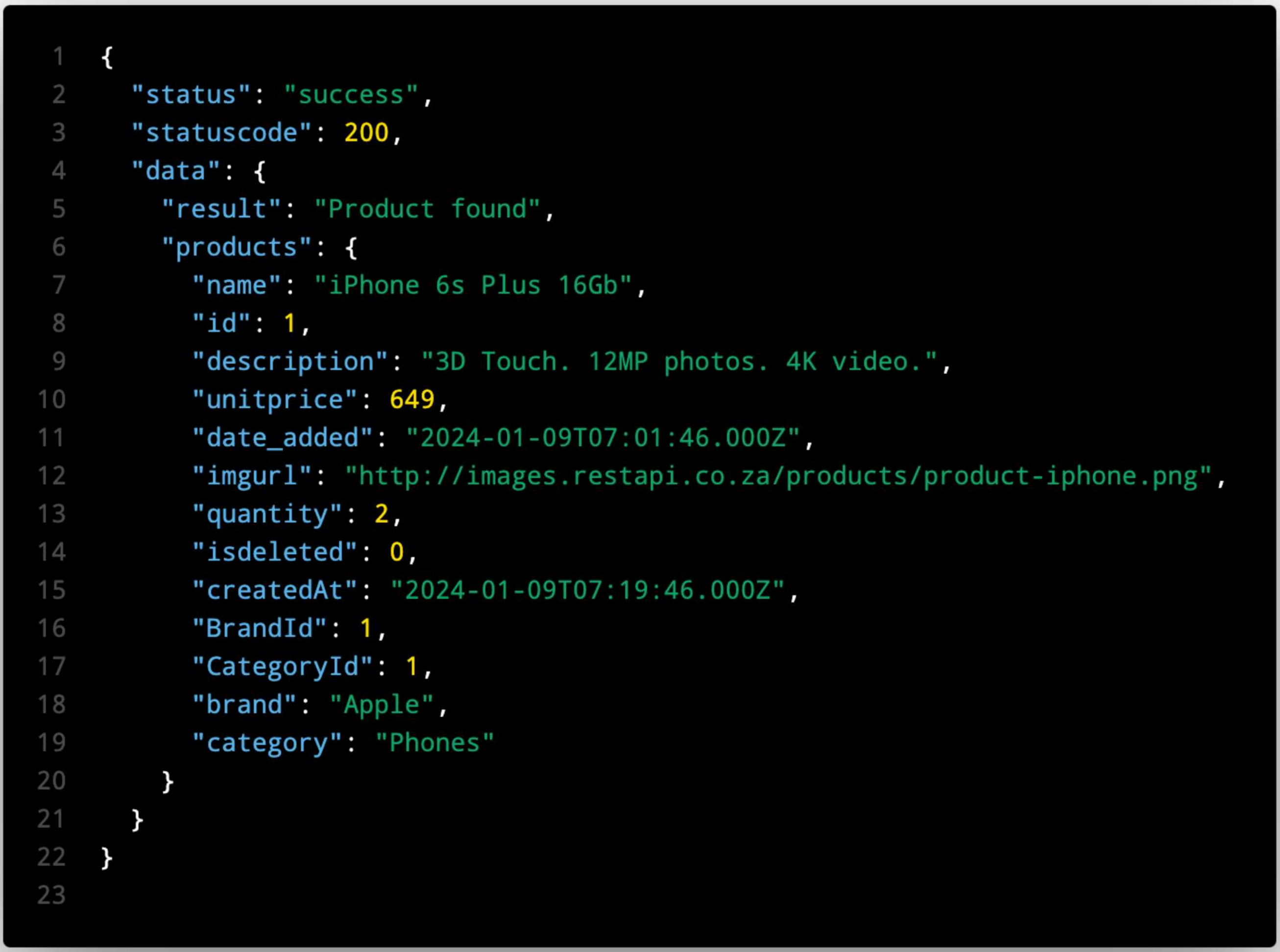
Minimum base endpoint required

- /products

All CRUD operations for products must be implemented:

- Only Admins can change/edit product information
- Deleting a product must be done with a soft delete (The product must not be permanently removed from the database).
- The admin roles should see all products deleted and not deleted from the GET /products endpoint
- The user role should only see the products that are not deleted from the GET /products endpoint

Requirement: A raw SQL query must be used to return the category and brand names in one object when a user is to view all the products. An example of the product object that should be returned



```json
{
  "status": "success",
  "statuscode": 200,
  "data": {
    "result": "Product found",
    "products": {
      "name": "iPhone 6s Plus 16Gb",
      "id": 1,
      "description": "3D Touch. 12MP photos. 4K video.",
      "unitprice": 649,
      "date_added": "2024-01-09T07:01:46.000Z",
      "imgurl": "http://images.restapi.co.za/products/product-iphone.png",
      "quantity": 2,
      "isdeleted": 0,
      "createdAt": "2024-01-09T07:19:46.000Z",
      "BrandId": 1,
      "CategoryId": 1,
      "brand": "Apple",
      "category": "Phones"
    }
  }
}

```

Categories

Minimum base endpoint required

- /categories

All CRUD operations for categories must be implemented:

- Only Admins can edit/change category information.

Brands

Minimum base endpoint required

- /brands

All CRUD operations for brands must be implemented:

- Only Admins can edit/change brand information

Membership

Minimum base endpoint required

- Membership

A member can only have **one membership status** at a time.
For example, once a Bronze user has made the correct quantity of purchases to become Silver, their status should change to Silver.

The membership statuses are as follows:

- Bronze (New member - No discounts on items purchased)
- Silver (More than 15 and less than 30 items purchased - 15% discount on their current purchase)
- Gold (More than 30 items purchased 30% discount in their current purchase)

Cart

Minimum base endpoints required

- /cart
- /cart/checkout/now

Only registered users can add items they wish to purchase to their cart. **Out-of-stock items cannot be placed in a cart.**

When a user adds a product item to their cart, the quantity of the product must be calculated to ensure there is enough of the specific product for the user to purchase. For example, if a user wants 10 product items, and the product quantity is only 2, the product cannot be added to their cart or checked out. Registered users who have added in-stock product items to their cart and there is enough quantity of the product item, can 'Checkout' their cart.

If a user adds the same product to their cart (if the cart has not been checked out), the quantity of the product item in their cart must be increased with one. A duplicate item for the same cart must not be added.

The current discount should be calculated based on the total number of items in the cart (i.e., discount based on the current membership status and total quantity) when the cart is checked out.

When a user Checks out their cart, an **order must be created** with an "In progress" status and a unique order number. Once a user has checked out their cart, it becomes an order.

The user **membership discount** must be recalculated once the cart is checked out and the order has been created (I.e., in case the membership status has changed due to the total quantity of items purchased to advance to the next membership level. This updated membership discount will be used for the next purchase).

Orders

Minimum base endpoint required

- /orders

Only registered users can view their own orders.
As mentioned, Orders are created once a user checks out their cart.
An order must contain order items (I.e., the products in that order).

Any previous orders that have already been fulfilled are also available to be viewed.
All orders should have one of the following statuses that is shown to the user:

- In Progress
- Ordered
- Completed

Only an admin user must be able to change the order status.
A user must be able to view their orders and the items on their order

The user's membership status for the current purchase must be captured on the order. This has to be done for auditing purposes.

An order has a unique generated 8-character (Example 6ad4JHid) order number. This number must not be used as the primary key. The order number must be generated when the order is created when the user checks out their cart. The user's current membership should be captured and stored in the database for the current order.

Initialize the database

Minimum endpoint required

- POST /init (required)

This endpoint is used for the **initial database population**. It should only populate the database once. All the relevant data from the Noroff API must be used to populate the database with the relationships.

The initial data can be obtained with an API GET call to the Noroff API **(http://backend.restapi.co.za/items/products)**
Pay attention to the relationships.

A JSON success should be returned without errors when this population is completed correctly. If an error occurs, a specific error message should be returned.

This API endpoint additionally should:

- Populate the roles table with two roles:

1. Id of 1 for Admin
2. Id of 2 for User

- Create an initial Admin user in the Users table:
  - Username: Admin
  - Password: P@ssword2023
  - Email: admin@noroff.no
  - First Name: Admin
  - Last Name: Support
  - Address: Online
  - Telephone: 911

- Populate the membership table with the following data:
  - Bronze with 0% discount
  - Silver with min 15 and max 30 for a 15% discount
  - Gold with a min of 30 for a 30% discount

Search endpoint

Minimum endpoint required

- POST /search

This endpoint searches for products in the database. Results should be returned as a JSON object.

Any user can access the /search endpoint. The /search endpoint must be done with a POST handler and done in the back-end. (via the /search route) The search endpoint must be added to the admin front-end products page.

The search endpoint should, as a minimum, be able to:

- Search for a partial product name (it does not need to be the full product name)
- Search for a product with a specific category name. The result should be all the products in the searched category.
- Search for a product with a specific brand name - The result should be all the products for the searched brand.

The searches must be done with **raw SQL queries**. No ORM search functionality can be used for search queries other than the query method.

The search results should return the items found in the result object and the number of records found.

Admin Front-end

Your Exam Project must include a separate **Admin Front-end Interface** that you design and implement. The admin front-end should be done with ExpressJS and the EJS template engine. This system should only allow **admin users** to log in.

You must design the front-end. Bootstrap should be used for the front-end CSS design.

The front-end system must only **use the API endpoints** that were created on your back end. (No sequelize service files must be used for the front-end, only the back-end API endpoints. The back-end API uses service files, and this is allowed.)

The Admin front-end should include the following:

- Product management: add, edit, delete, and search
- Category and brand management: add, edit, and delete
- Order management
- User management
- Membership management

For example (Not all routes have been shown):

- http://localhost:3001/products (To show/edit/add all products)
- http://localhost:3001/brands (To show/edit/add all brands)

Attention should be given to user experience and functionality.

An example front-end for the back-end API

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. Add a category with the name TEST_CATEGORY
2. Add a brand with the name TEST_BRAND
3. Add a product with the name TEST_PRODUCT, brand must be TEST_BRAND, and category must be TEST_CATEGORY, quantity 10, price 99.99
4. Get the newly created TEST_PRODUCT with all the information, including category and brand name.
5. Change the category name TEST_CATEGORY to TEST_CATEGORY2
6. Change the brand name TEST_BRAND to TEST_BRAND2
7. Get the product TEST_PRODUCT with all the information, including the category and brand name.
8. Delete the TEST_PRODUCT

Documentation

Student must indicate where they have received help or used outside knowledge for their Exam Project. This must be indicated in the **project's README** file under the heading "REFERENCES."

Each project is checked for plagiarism using various tools and software, so this step is **VERY IMPORTANT!** Artificial Intelligence (AI) can be used, but not as your own code. This can be used to help you, but the assignment must be your own work.

This would include:

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

Once the project is complete, all Documentation for this project must be bundled into a **single PDF file** and added to your Git repository Documentation folder and must contain the following:

- A Reflection 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 reflection write-up including the following sections:
  - Discussion of the progression of the project.
  - Challenges faced during the development of this project.

- Include an example of your .env file

The API documentation (Swagger), including methods, description, and JSON objects, must be accessible from the endpoint /doc from the API URL example:**http://localhost:3000/doc**

(The reflection report must be between 500 to 1000 words)

Good luck, everyone. We hope you enjoy the exam project!