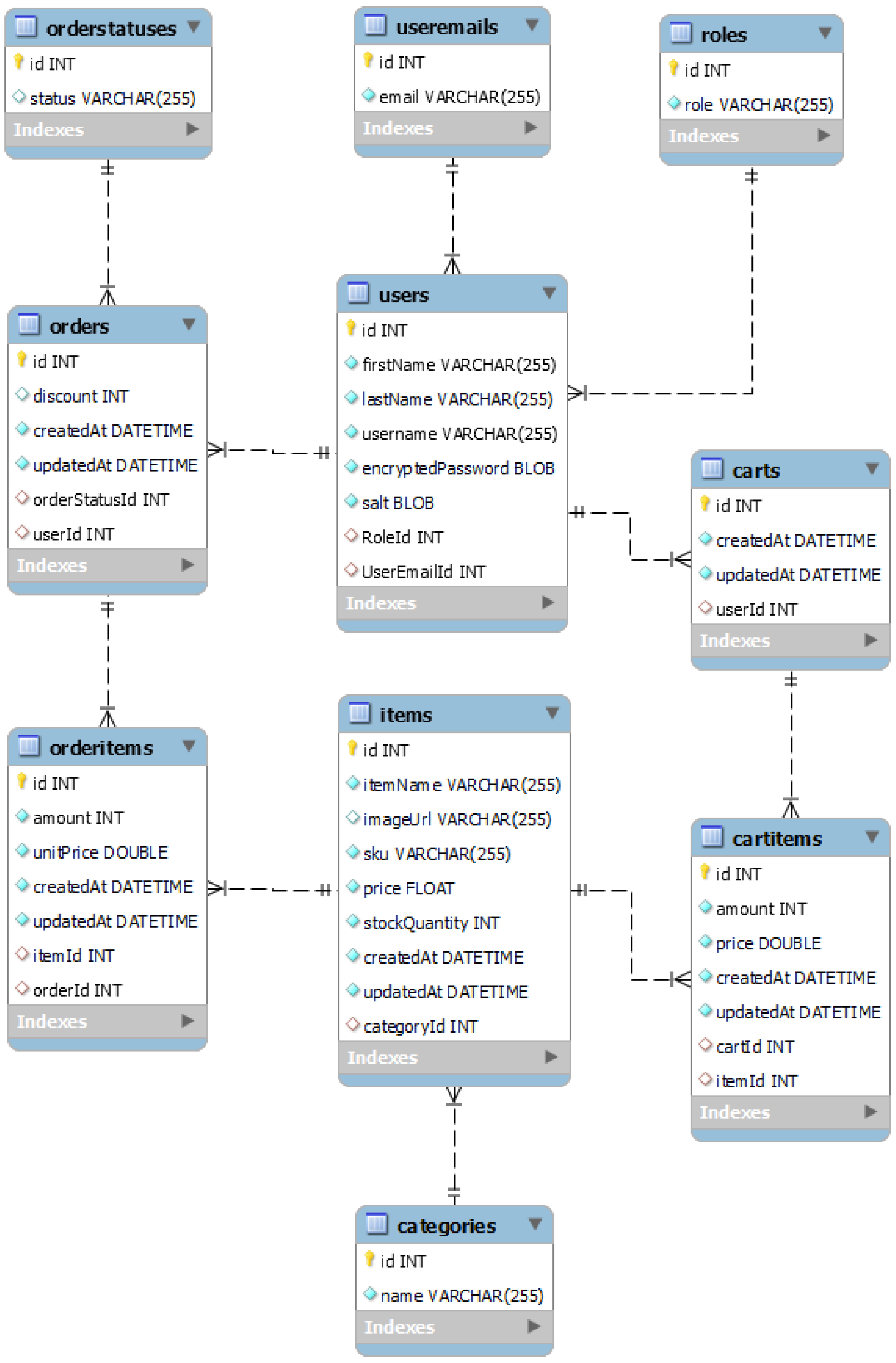
BED Exam Project

Documentation

Retro Perspective Report

1. Database

ERD extracted from MySQL workbench showing tables and relations.



The **users** data entity has one **role** and one **email**. Both role and email can be shared with several users and are stored in separate tables with a one to many relationship.

The **item** entities has one **category**, and one category may belong to several items, having a one to may relationship.

The **cart** entity can only belong to one **user** but the user can in principle have many carts, having a one to many relationships.

NOTE: For solution no user can have more than one cart. This is not constrained by database, but it is constrained by application.

The **cart item** entity can only belong to one **cart**, and only have one **item**, though the cart can hold several cart items, and items can belong to many cart items. These are one to many relationships.

The **order item** entities are similar to cart item entities have one to many relationships with **items** and **orders.**

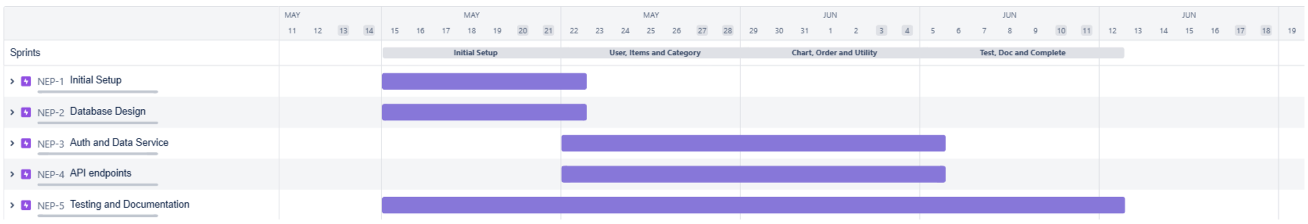
The **users** can have many **orders**, and one order can only belong to a single user, having a one to many relationship.

The **order** can have one predefined **order status**, and any order status can have several orders. Tables have a one to many relationship.

Timestamps are included for all entities where amount, quantity or similar values are expected to change.

Values that can be calculated for other data are not included in any tables as they are or at least should be calculated on query or processing.

Jira Roadmap



The Exam project was divided into 5 Epics and 4 Sprints. The defined epics are not kosher in accordance with agile, but represent a completed set of functionality inside the application. Each sprint are adding a set of features resulting in a functional application with extended features on completion. Tasks from two or more epics are concurrent in each sprint, including continuous updating of documentation.

Challenges / Retrospect

I wanted to keep the routes as clean as possible, so I chose to create an architecture where each each route was served by it's own designated data service. Any interaction with the data service not resulting in the desired outcome or dataset, would throw an error that would be handled by the route handler. The router handler would then relay the result with JSend as a success, fail or error depending on the request outcome.

Having JSend relaying error messages on **500 internal server error** is not a good solution, but left as is for development purposes.

I also created middleware for route handlers where validation of data in request body would require more than a few lines of code. Custom middleware was also created for securing endpoints.

Having one singular data service for each group of endpoints would result in some duplicate code, but seems to be limited to query methods only.

Not sure what should be strictly required in each entity, when posting data, I made everything strictly required, with the exception of updating items where only provided values are updated. I did not make item name required to be a unique value, but a warning is provided if an item with a similar name exists in the database.

When importing from Noroff API, the Sequelize **bulkCreate()** method was never a good option for a range of reasons. Instead I used loops to remap and insert the imported data. My first version of this method was created to insert any values existing in import missing in the database. But when reviewing the course assignment, I noted importing data should only execute if no data exists. I did a minor refactor to meet the requirement, but kept most of the code in the method as before for easier reversion to previous feature.

I never really been a huge fan of Object Relation Mappers, and honestly enjoy writing SQL code. In this solution I have used both Sequelize and raw queries through Sequelize for CRUD operations. Especially minor queries regarding a single value like ID have been my prime target for raw queries. There is a minor discrepancy in the data format when queried either by admin or user account. Since having clear definition of what data and in what format it is expected as return value, I left it as is.

The test part of this application had a real Catch 22:

post /search – search for all items with the category name "Laptop" (one item should be returned from the initial data).

My honest interpretation is that this test is supposed to query all times with the category name "Laptop", and expecting at least one item. But "Laptop" do not exist as a category in the dataset provided for the test. Not sure if I was supposed to write the right test to fail, or the wrong test to pass, I decided on the first alternative. So, this test will fail.

Noroff Backend Exam Project

Backend solution for RestAPI for Exam Project Assignment.

Created with ExpressJS. No views or view engine are included, RestAPI endpoints only.

Installation and Usage

Application require NodeJS, and a MySQL database for persistence.

1. Clone or fork this repo.
2. Run npm install to install packages.
3. Set up database and set required environment variables *(see example below)*
4. Run application least once to migrate database models from sequelize to database.
5. SEND a POST **/setup** to fetch or create initial data.

Setting db logging should be set to false in production!

Credentials for admin is currently hardcoded in **AuthService.createAdmin()** and meet expectations found in the Course Assignment Resources.

Important:

To successfully run tests, a database must be connected and the application must run at least once to migrate datamodels to database.

Run Scripts:

- **npm start** : Run application in production
- **npm run dev** : Development mode with nodemon
- **npm test** : Run tests

Known issues:

- Category *"Laptop"* **was not** provided as category name from the initial data. Thereby searching for for all items with category *"Laptop"* will not produce one item as indicated in the exam instruction. To meet the exam instructions, the test is written to expect at least one result. Unless provided data is changed: THIS TEST WILL FAIL!
  - *instruction reference:* 7. post /search – search for all items with the category name "Laptop" (one item should be returned from the initial data).
  - *source data* as json file with the content and format of data available when this solution was created.

Environment

Application using environment variables. In development, variables are extracted from a local .env file. Example below lists environment variables.

```
HOST = "localhost"                    # Database host
ADMIN_USERNAME = "admin"              # Db admin account
ADMIN_PASSWORD = "admin_password"     # Db admin password
DATABASE_NAME = "db_name"             # DB name
DIALECT = "mysql"                     # SQL Dialect
PORT = "3000"                         # Application port, default = 3000
DB_LOGGING = false                    # Sequelize SQL Logging enable/disabled,
                                      #   default = False
TOKEN_SECRET = secret token           # Token Secret
```

Packages

Packages used in this application are listed here for easy documentation lookup.

- ExpressJS : Application Framework
- Sequelize : DB Orm
- MySql2 : MySql library
- Jsend : Middleware for response.
- JsonWebToken : Implementing JWT for Auth.
- dotenv : loading environment variables-
- validator : used for validating data.

Dev packages:

- Nodemon : Restart on change in dev mode.
- Jest : Test library
- Supertest : Test library

Acknowledgment

Solution code is based on documentation listed in packages above or by previous assignment solutions defined by course material. Other sources are listed below.

Documentation

The **Documentation** has been compiled from three PDFs as required by the course assignment.

The PDFs have also been included for this solution under *./documentaion/parts/* and include - This README.md converted to PDF. - Standalone Postman Documentation.pdf - The Retrospective Report - Postman Documentation Online

This solution was created in accordance with Course Assignment Resources given for this exam project. PDF printout contain timestamp. Changes made to the assignment text after this time and date, have been ignored, with the exception where I have received direct and explicit notification from those responsible about changes that have been made.

_Noroff Exam Project

Documentation for API as described in the final exam assignment.

API is secured with Json Web Token. Some endpoint allow anonymous users, while other require authenticated user or admin account. Auth details are provided for each endpoint.

AUTHORIZATION Bearer Token

Token	<token>

Auth

Auth endpoints allow signup of new users and login for both users and admin.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

POST Signup

http://localhost:3000/signup

Create a new user account.

Endpoint open for anonymous users.

Requred fields are shown in examle below. Will respond 400 Bad Request and report if any required fields are missing for request.

Username must to be unique for each user.

Up to 4 user accounts can be registered to one single email address.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

json

```
{
    "firstName": "John",
    "lastName": "Smith",
    "username": "johnsmith",
    "email": "john.smith@app.com",
    "password": "s3cr3tP@ssw0rd"
}
```

POST Login

http://localhost:3000/login

Login for registed users.

Endpoint open for anonymous users.

Successful login will provide a JWT token valid for 2 hours.

Requred fields are shown in examle below. Will respond 400 Bad Request and report if any required fields are missing for request.

Invalid username, password or both will result the same 400 bad request response.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

json

```json
{
    "username" : "johnsmith",
    "password" : "s3cr3tP@ssw0rd"
}
```

Items

Endpoints for items.

Anonymous users are allowed to get all items on request, while create, update and deleting items require authenticated user with admin role.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

GET Get All Items

http://localhost:3000/items

Get all item entities include nested category.

Anonymous users allowed.

NOTE: Guest user can only see in stock items.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

POST Post new Item

http://localhost:3000/item

Creates a new item entity.

Autheticated admin required.

Requred fields are shown in examle below. Will respond 400 Bad Request and report if any required fields are missing for request.

SKU (Product Code) must be unique for the item. Will result a 400 response if existing SKU is provided.

Item name is not required to be unique, but a warning on creating with existing name is returned with the response.

Note that a valid category ID is required for creating a new item entity.

Price and Stock Quantity can be set to 0, but can not be negative and will throw a bad request if provided.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

json

```json
{
    "itemName": "Cellphone",
    "categoryId": 3,
    "imageUrl": "none",
    "sku": "EXMP_5",
    "price": 100,
    "stockQuantity": 10
}
```

PUT Update Item

http://localhost:3000/item/:id

Update an item entity.

Autheticated admin required.

Follows the same pattern as a post request, but no fields are required in the request body. Will only update data from provided fields.

Reponds 404 not found if Item ID not valid.

SKU (Product Code) must be unique for the item. Will result a 400 response if new existing SKU is provided.

Item name is not required to be unique, but a warning on replacing with existing name is returned with the response.

Note that a valid category ID is required for updating the item entity.

Price and Stock Quantity can be set to 0, but can not be negative and will throw a bad request if provided.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	167 Integer - Item ID

Body raw (json)

json

```json
{
    "itemName": "Cellphone",
    "categoryId": 3,
    "imageUrl": "none",
    "sku": "EXMP_5",
    "price": 250,
    "stockQuantity": 10
}
```

DELETE Delete Item

http://localhost:3000/item/:id

Deletes a item entity by provided ID in path variable.

Autheticated admin required.

Will result a 404 response if invalid ID was provided.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	167 Integer - Item ID

Category

Endpoints for Category entities.

Anonymous users are allowed GET requests retriving all category items. Other request types require admin role for authenticated user.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

GET Get All Categories

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

Get all category items.

Anonymoyus users allowed.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

POST Post new Category

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

Creates a new Category

Autheticated admin required

Requred fields are shown in examle below. Will respond 400 Bad Request and report if any required fields are missing for request.

Category names are unique values. Providing an existing category name will result a 400 response.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

json

```json
{
    "name": "New Category"
}
```

PUT Update Category

http://localhost:3000/category/:id

Update a category with entity id set in the path variable.

Autheticated admin required

Requred fields are shown in examle below. Will respond 400 Bad Request and report if any required fields are missing for request.

Respond a 404 not found if provided id not valid.

Do not allow updating to an existing category name, request will result a 400 response.

API do not accept the nonsense of updating the existing name with the existing name. Request will result the exotic reponse 406 - Not Acceptable.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	2 Integer - Category Id

Body raw (json)

json

```
{
    "name" : "Category 1"
}
```

DELETE Delete Category

http://localhost:3000/category/:id

Deletes a category item from provided ID in path variable.

Require autheticated user with admin role.

Will respons a 404 not found if invalid id was provided.

Cannot delete a category entity if it has a dependent item entity, and will result a 400 response.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	36 Integer - Category ID

Cart

Endpoint for managing carts.

All endpoint required autheticated user. Only admin user can extract data for all user carts.

Cart does not exist for new users, but will be created when the first item gets added.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

GET User Cart

http://localhost:3000/cart

Get cart registered to logged in user.

Require autheticated user with User role.

Will return not found if cart does not exist. A cart item must be added to create a new cart if it does not exist

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

GET All Carts (Admin)

http://localhost:3000/allcarts

Return all carts to all user including cart items and basic user data.

Require autheticated user with Admin role.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

POST Add Cart Item

http://localhost:3000/cart_item

Add a new cart item to cart.

Items can be added by providing a valid item id. Amout is not required as missing valie will set 1 as default.

Only new items can be added. Use PUT request to edit existing cart items.

Autenticated user required!

See examples below for required fields in request body. If required values are missing, API will return a 400 response reporting the missing values.

Will report 404 if a non existing item id is provided.

Will report 400 if item exists in cart.

Will report 400 if item is out of stock or requested amout exceed items in stock.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

```json
{
    "itemId": 157,
    "amount": 1
}
```

PUT Update Cart Item

http://localhost:3000/cart_item/:id

Updates the amout of items in a cart item

NOTE: The path variable must reference the Item ID, and **NOT** the Cart Item ID

Authenticated user required!

See examples below for required fields in request body. If required values are missing, API will return a 400 response reporting the missing values.

Will return 404 if provided item ID does not exist among cart items in user cart.

Will return 400 if new amout requested exceed items in stock.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	132 Integer - Item ID

Body raw (json)

```json
{
    "newAmount": 1000
}
```

DELETE Delete Cart item

```
http://localhost:3000/cart_item/:id
```

Removes an item from cart.

NOTE: The path variable must reference the Item ID, and **NOT** the Cart Item ID

Require authenticated user.

Will respond 404 if invalid cart item ID is provided

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	135 Integer - Cart Item ID

DELETE Delete All Cart Items

```
http://localhost:3000/cart/:id
```

Deletes all items from cart.

Require autheticated user with User role.

Will respond 404 if provided cart ID is invalid.

NOTE that this method only removes all existing items from cart entity. The cart entity will remain thought wihtout any related entities in database.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	1 Integer - Cart ID

Orders

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

GET Get Order (User)

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

Get all orders for user if user signed in.

Logged in admin will get same result as from **GET /allorders.**

Endpoint require signed in user with Admin or User role.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

GET Get All Orders (Admin)

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

Get all orders for all users

Required logged in user with Admin role

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

POST Post new Order (User)

```
http://localhost:3000/order/:id
```

Check out cart and place it as an order. Will remove all cart items as they are transformed to order item. Will also subtract requested amout in cart from items in stock.

Require logged in user with User role.

Will respond with 404 if cart Id is invalid or doesn't belong to logged in user.

Will respond with 404 if not items are added to cart.

Will respond 400 if requested cart item amout exceed items in stock, or if item is out of stock.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	1 Integer - Cart ID

PUT Update Order (Admin)

http://localhost:3000/order/:id

StartFragment

Update a category with entity id set in the path variable.

Change status for order with entity id set in the path variable.

Valid statues are COMPLETE, IN PROCESS and CANCELLED.

Require autheticated user with admin role.

See examples below for required fields in request body. If required values are missing, API will return a 400 response reporting the missing values.

Respond a 404 not found if provided order does not exist.

Respond 400 if invalid status is provided

EndFragment

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

PATH VARIABLES

id	1 Integer - Order ID

Body raw (json)

json

```json
{
    "orderStatus": "complete"
}
```

Utils

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection _Noroff Exam Project

POST Setup endpoint

http://localhost:3000/setup

This endpoint inserts a collection of values to database, as well as importing data from external sources.

Will always respond a report on what have been created.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

POST Search

http://localhost:3000/search

Search by SKU, category or partial item name.

If sku is provded, other values are ignored and item is searched by SKU only.

Categories require full name. Item name can be partial.

SKU search will provide a single item, while search by partial item name and / or category will return a list.

A 404 response is provided if no items are found.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project

Body raw (json)

json

```json
{
    "itemName": "mar"
}
```

GET Alive@Index

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

Base url will respond 200 I am alive if reciving a get request.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection _Noroff Exam Project