**Node js: Graduated Project**

Deadline: 1 Week (Starting from Tuz, May 18, 2023)

\*\* Group work is not allowed.

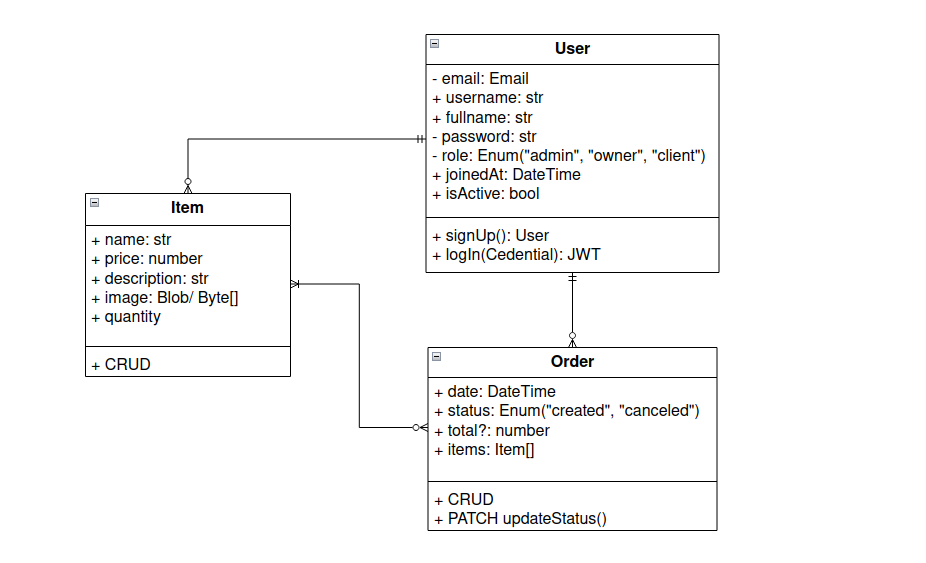
Use case: E-Store/Shop API with Express

Using Node JS, Express, and MongoDB build a RESTful API that provide endpoints:

* Manage CRUD (Create, Read, Update, and Delete) items.
* Users **auth** signup and sign-in
* Searching and Filtering items by fields
* Querying orders by items, owner, or clients

**Actors** (roles): Admin, Owner, Client

**Classes diagram**



**Project setup**

\*\* Please make sure to check your project before submitting to get a full complete review and marks as well.

**Validation plan**

**1. Database setup (**2 Marks**)**

**1.1**. Setup a local MongoDB database server or use Atlas database as a service (free cluster of 512 Mo storage capacity).

**1.2**. Once you have got a connection string to your cluster/database, use mongoose to connect to the database.

**2. Defining Models schemes (**4 Marks**)**

Following what we have done in the w4 persisting data with mongodb; using mongoose define your API models schemes.

**2.1**. **User**(email, username, fullname, password, role, joinedAt, isActive)

**2.2**. **Item**(name, price, description, image, #owner, quantity?)

**2.3**. Add some validation for User and Item models schemes.

**2.4**. **Order**(date, status, #client, total, Item [] or ItemOrder[], where

ItemOrder = { item: objectId, quantity: number }

It's for you to judge, suggest or propose another data model regarding the order model.

**3. Routers & CRUD operations (**9 Marks**)**

Implement the auth, and items routers

Handling HTTP requests, producing responses, works with HTTP status code standard and all RESTful best practices

**3.1**. **The auth router** (x2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **URI** | **Req body** | **Res body /Code** | **Usage** |
| POST | /api/v1/auth/signup | User | User /201  Error /400 | Add new user |
| POST | /api/v1/auth/login | Credentials | JWT /200  Error /400 | Authenticate user |

**3.2**. **The items router** (x4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **URI** | **Req body** | **Res body / Code** | **Usage** |
| GET | /api/v1/items?q |  | Item [] /200 | Querying items |
| POST | /api/v1/items | Item | Item /201 | Add item |
| GET | /api/v1/items/:id |  | Item /200  NA /404 | Get item by id |
| PUT | /api/v1/items/:id | Item | Item /203  Errors /400/404 | Update item by id |
| DELETE | /api/v1/items/:id |  | NA /204  Error /404 | Delete item by id |

**3.3**. **The orders router** (x4)

Implement CRUD for orders

Add PATCH /api/v1/orders/:id that changes the status of given order information, quantity update in the store collection.

**4. Securing your API (**3 Marks**)**

**4.1**. Encrypt users passwords before persisting them to the database. (x1)

**4.2**. Add an authentication layer for your API endpoints that requires an authenticated user for accessing or managing data by providing a JWT on each request headers. (x1.5)

Use authentication for all API routers except the auth endpoint.

**4.3**. Add two authorizations middlewares; **hasRole**(string role) and **isOwner**(string objectId, string userId) that ensure authorization policy for your API. (x0.5)

**Testing and best practices (**2 Marks**)**

Demonstrating testing (Manual) and best practices knowledge and understanding.