2025

Ai Finance Project

INTERNSHIP APPLICATION TEST STUDY

ASSIGNMENT - AND PRODUCT MANAGEMENT

Table of Contents

Mission Description:

In this project, we aim to develop real-life scenarios by deploying basic modules such as user management, authentication/authorization, product management, shopping cart operations and purchasing process to two separate microservices.

- **user.service**: Manage user authentication and authorization.
- **product.service**: It will provide e-commerce functions such as product list, purchase, cart operations.

User Service:

What to Do:

1. User management
   - Add, deactivate, change password
2. User address information
   - Add, modify, delete, list addresses (CRUD)
   - Address categorization as Home,
3. User contact information
   - Add, Modify, Delete Communication (CRUD)
   - Categorizing Communication as Home, Work
4. Admin user management panel
   - User deactivation, relay replacement,
   - Password Reset.
5. Authentication
   - Login operations
   - Token generation, Token and Duration control
6. Authorization
   - Pulling authorizations from the database and keeping them in memory or cache. (Keeping them in cache is optional, not mandatory.)
   - Transport of user information between layers during the request.

Endpoints

Authentication Endpoint(Path: / auth)

Method	Path	Description	Role	Permission
*******	*******	Log in	-	*******
*******	*******	Sign out	User,Admin	*******
*******	/checkLogin	Token validity check	User,Admin	*******

Authorization Endpoint(Path: / authz)

Method	Path	Description	Role	Permission
*******	*******	Fetch all permissions of the user	User,Admin	*******
*******	*******	Does the user have this role	User,Admin	*******
*******	*******	the user have specific permission (ops.)?	User,Admin	*******

User Endpoint(Path: / user)

Method	Path	Description	Role	Permission
*******	*******	Bring all users	Admin	*******
*******	*******	User details	Admin	*******
*******	*******	Create new user	Admin	*******
*******	*******	Update user	Admin	*******
*******	*******	Delete user (soft delete)	Admin	*******
*******	*******	Change password (old→new)	User	*******
*******	*******	Reset password	Admin	*******
*******	*******	Logged in user information	User,Admin	*******
*******	*******	Deactivate another user	Admin	*******
*******	*******	Deactivate your own account	User	*******

Address and Contact EndpointPath/address/contact)

Method	Path	Description	Role	Permission
*******	*******	List information	User	*******
*******	*******	Add new address	User	*******
*******	*******	Update address	User	*******
*******	*******	Delete address	User	*******

Explanations:

Language: Go or Python. Db: Must be PostgreSql.
Technologies:

- Pyhton Flask, Fastapi
- Go: Optional.

Authentication: When the first login is made to the screen, the system will whether the user is logged in or expired with checkLogin. If it is expired, it will ask for login again. The token returned after login will be written to the Authorization header in each request. The spelling must comply with the standards. Check this token in each incoming request
must be done.

Authorization: Token must be associated with the user. User information and roles should be accessible across layers throughout the request. Role types will be user and admin. If you wish, you can bind permissions to roles and control authorizations via permission. In such a case, permissions be accessible throughout the request.

Password Change: The user will be able to change his/her own password. If forgotten, the administrator will reset the password and the user will be prompted for a new password even if he/she logs in with the reset password the first time he/she logs in.

Repository layer: Must be orm-based.

Address Information: Can be classified as or Home.

Communication Information: Can be categorized as or Home.

User, Address and Contact Screens: The design of these screens will be your own. Optional additional screens in all panels information can be added. You can use any technology on the front side. Saving user address and contact information can be done separately or all at once with the DTO object. This completely depends on your design. After the user logs in, the user's name, surname and other requested information will be received via the endpoint. This user information panel
will be in the top right corner of the screen. Additional information can be added.

Error management: In case of token expire in any request, the user will be automatically redirected to the login screen. Other errors will be fired as notification.

Validation processes: The values entered on the screens will go through a validation. If desired, this validation can be done in the backend. Or it can be in both ui and backend. This depends on your request.

Product Service:

What to do:

1. Product Management

- Add, update, delete, list products
- Product category support (e.g. electronics, clothing, etc.)
- Product stock and price information

2. Shopping Cart Operations

- Adding/removing products to cart
- View cart
- Cleaning the basket

3. Purchasing (Order)

- The process of purchasing items in the cart
- View order history
- Deduction from stock

4. Administration Panel (for Admin)

- Bulk update products
- Sales reports (optional)
- Product visibility toggle (active/passive)

5. Authentication and Authorization (via user.service)

- Incoming JWT token to be checked
- User ID token→ user_id, role, permissions
- Authorization check will be done via an endpoint control or an endpoint returning a list
- (Optional) Authorizations can be stored via Redis cache

Explanations:

Language: .NET Core 6+ or Go / Python (according to project preferences)
Db: Must be PostgreSql.
Cache: Redis (Optional) Technologies:

- Pyhton Flask, Fastapi
- Go: Optional.

Authorization: Token is pulled from the authorization header. Session information is checked via user.service with checkLogin.

Endpoints

Product Endpoint(Path: /product)

Method	Path	Description	Role	Permission
****	******	List all products	User,Admin	*******
****	*****	Product detail	User,Admin	*******
****	*****	Add new product	Admin	*******
****	****	Add bulk product	Admin	*******
****	*******	Update product	Admin	*******
****	****	Product delete	Admin	*******

Card Endpoint(Path: /card)

Method	Path	Description	Role	Permission
****	*****	Fetch user's cart	User	*******
****	****	Bring up the product detail in the cart	User	*******
****	****	Add product to cart	User	*******
****	*****	Update product quantity in cart	User	*******
***	****	Remove items from the cart	User	*******
****	****	Clear basket	User	*******

Order Endpoint(Path: /order)

Method	Path	Description	Role	Permission
*****	****	Buy the cart	User	*******
****	****	Order history	User	*******

Repository Layer (ORM Based)

- Entity: Product, Cart, Order
- ORM: Entity Framework Core (C#), GORM (Go), SQLAlchemy (Python)
- Repository Interface and Concrete folder structure is recommended.

User Information

- In each request, the user_id will be extracted from the token and linked to orders or cart transactions.
- Users with admin role can perform product operations.

Test Scenarios

Authentication & Authorization Test Scenarios

Scenario No	Description
A1	Login with valid user credentials→ 200+ JWT token returns
A2	Login with invalid password→ 401 Unauthorized
A3	Access to a protected endpoint without login→ 401 Unauthorized
A4	Token expired user→ /CheckLogin→ 401
A5	After logout the token should be invalid→ should return 401
A6	Test of service that pulls user information with JWT token→ User information should be returned
A7	User authorization and relay controls→ If there is authorization or relay 200

User Test Scenarios

Scenario No	Description
U1	Admin user creates new user→ 201 Created
U2	Re-register with the same username→ 409 Conflict
U3	User changes their password to the correct old password→ 200 OK
U4	If old password is incorrect when changing user password→ 400 Bad Request
U5	Admin resets another user's password→ 200 OK
U6	Updates user status→ 200 OK
U7	The user their own user. → 200 OK
U8	User information is retrieved with the correct id→ 200 OK
U9	Queried with a non-user ID→ 404 Not Found

Address and Contact Information Scenarios

Scenario No	Description
C1	User adds a new home and work address→ 201
C2	Updates user address→ 200
C3	Deletion with invalid address ID→ 404
C4	User adds work and home phone→ 201
C5	If there is a format error when updating user contact information→ 400

Role & Permission Test Scenarios

Scenario No	Description
R1	Admin defines new role→ 201
R2	Admin updates role details→ 200
R3	User role is changed by admin→ 200
R4	If a non-user is assigned to a role→ 404
R5	Admin role permission assigns→ 200

Product Service Test Scenarios

Scenario No	Description
P1	User role pulls product list→ 200+ product list
P2	Admin adds new product→ 201
P3	Admin adds product with same name→ 409
P4	Admin updates product→ 200
P5	Non-admin user tries to delete product→ 403
P6	Product detail information retractable→ product detail returns
P7	Invalid ID→ 404
P8	Bulk product addition request→ 201 and product list returns

Shopping Cart Test Scenarios

Scenario No	Description
S1	User adds items to cart→ 200 OK
S2	Adds the same product to cart again→ quantity is updated
S3	User lists cart→ returns with product information
S4	Remove items from cart→ 200 OK
S5	Cleans the basket→ basket returns empty
S6	If stock is insufficient at the time of product addition→ 400 Bad Request
S7	If a non-logged in user adds a cart→ 401

Order Test Scenarios

Scenario No	Description
O1	User orders items in the cart→ 201+ order ID
O2	Order process if cart is empty→ 400
O3	After the order is deducted from the stock→ stock is updated
O4	Displays user order history→ 200+ list
O5	Unauthorized user view order history→ 403

Validation Test Scenarios (General)

Scenario No	Description
V1	Password not less than 8 characters→ 400
V2	Email format incorrect→ 400
V3	If the address type is something other than "work/home"→ 400
V4	→400 if phone number is not valid with regex
V5	If JSON is sent with missing field→ 422 Unprocessable Entity

Interface Design

- UI framework preference is free (example: React, Vue, Angular)
- After logging in, the user's name, surname and other information will be displayed in the upper right corner
- There will be user information (address, contact) login screens.
- Product listing screen will be made.
- Basket automation and display will be made.
- Payment screen will be made.
- Admin panel (User management) will be made.
- Product management panel will be made.
- Stock status information screen will be made.
- Changing the user password and logout will also be shown in the top right corner.

Delivery Expectation

- separate repo or project folder for user.service and product.service
- API documentation (Swagger/OpenAPI or Postman Collection)
- Dockerfile and docker-compose.yml file
- Positive and negative test scenarios should be prepared for all endpoints except sample tests.
- Readme file:
  - Installation and operation steps
  - Technologies used
  - API endpoint list
  - Default admin user information
- Clean, readable and layered code (service, repository, model, controller separation)