UTCN
Computer Science
DepartmentSoftware Design
2024

ASSIGNMENT A1

==

1. Objective

The objective of this assignment is to allow students to become familiar with architectural patterns and basic architectural techniques implemented from scratch.

2. Application Description

Use JAVA/C# API to design and implement an application for the employees of a musical instruments shop. The application should have three types of users (a customer, a regular user represented by the cashier and an administrator user) which have to provide a username and a password in order to use the application.

The customer type user can perform the following operations:

- Access the app
    - As a guest (no login)
    - As a fidelity-user (with login)
- View items and add them to a shopping cart
    - If not logged in, there will be just 1 "floating" shopping cart (not associated to any user) that gets emptied each time the app restarts or another guest user access it
    - If logged in, the contents of the shopping cart should be retrieved from the DB and can be viewed by the logged in user
    - Checkout the shopping cart
        - Once the "checkout" button is pressed, a cashier will need to login and take over the shopping cart (see next section)
        - The shopping cart does not get emptied at this point unless the user is not logged in or the purchase is confirmed/completed by the cashier

The cashier type user can perform the following operations:

- Create fidelity-users by adding/updating/viewing client information (name, identity card number, personal numericalcode, address, etc.). Fidelity-users need to have a unique email attached to them and they accumulate points based on a percentage of the total cost of a buyer's list once a purchase is completed. The percentage is defined by the administrator. At subsequent purchases, these points can be redeemed and users can pay less based on how many points they have/how many they want to spend

- Process a payment (a simple button)
    - Make sure that the cashier can see how many points the user has, in case it is a logged in user (hide field if not) and if operation is complete, that the points are subtracted from the point balance currently associated to the user
    - The payment processing should work only in 80% of the cases, in the rest of 20%, a random error should be thrown (either “not enough funds”, or “database error, please try again later.” [ignore how illogical this might seem]). In either case, the processing will not finish and the shopping cart will not be emptied, neither any fidelity points used. The user can try again without any repercussions.

The administrator type user can perform the following operations:

- Create/View/Update/Delete (CRUD) on cashier accounts and information (define whichever fields you deem necessary)
- Generate reports for a particular period containing the activities performed by a cashier (how many items they sold, how much money was that all was, how many unique customers they served)

3. Application Constraints

- The data will be stored in a database. Use the Layers architectural pattern to organise your application

- All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

4. Requirements

- Implement and test the application
    - All repository methods must have an associated test
    - All service methods must have an associated test
    - Persistence constraints (unique fields, etc.) must be tested
- Everything through GitHub!

5. Deliverables

- Implementation source files
- repository

6. References

Google.