<Selling Tickets App>

Student: Badareu Victoria

**Group: 30236**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

Use JAVA/C# API to design and implement an application for selling tickets for a three-days conference. The application should have two types of users (a regular user represented by the cashier and an administrator user).

The administrator user can perform the following operations:

* CRUD on cashiers information
* CRUD on tickets information
* Modifying conference capacity
* Raport on the number of tickets sold by a cashier
* Raport on earnings per day
* Total revenue report based on sold tickets

The regular user can perform the following operations:

* Sell tickets ( + create a file.txt on the disk for each ticket with its details)
* Viewing the tickets sold by him ordered by time and date

# Functional Requirements

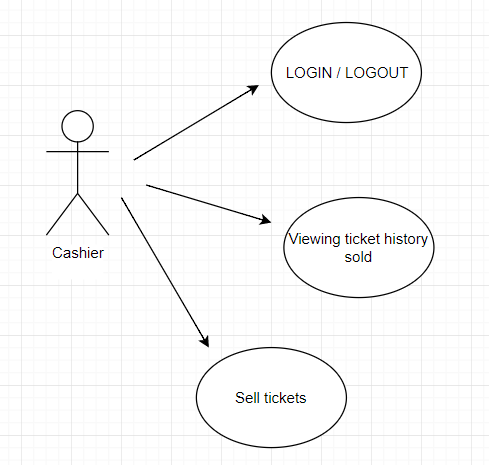
* The data will be stored in a database.
* Use the Layers architectural pattern to organize the application.
* Desktop application using JAVA/C#

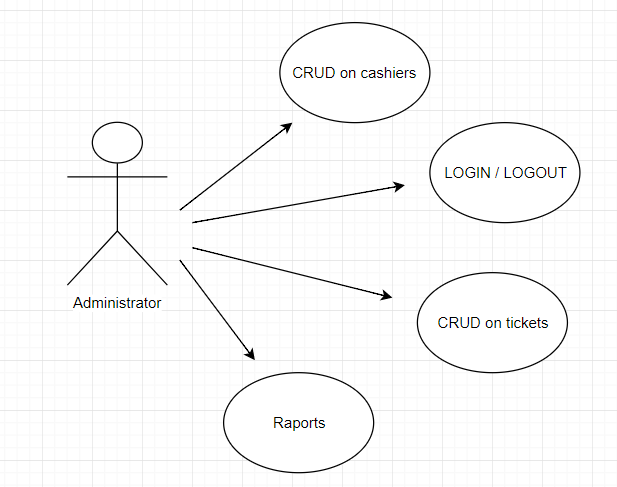
# Non-functional Requirements

Availability: application is a desktop one and it’s always available in offline mode even if computer is running on low resources.

Testability: Cashiers can provide test cases like selling a new ticket and then displaying all sold tickets to see if the sold was made.

2. Use-Case Model





Use case: Log in

Level: user-goal level

Primary actor: Administrator

Main success scenario: the admin logs in and using the interface he can: CRUD on tickets, CRUD on cashiers, and sees raports.

Extensions: an error message is displayed.

Use case: Log in

Level: user-goal level

Primary actor: Cashier

Main success scenario: the cashier logs in and using the interface he can: sell tickets, viewing ticket history sold.

Extensions: an error message is displayed.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering the application appropriately helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

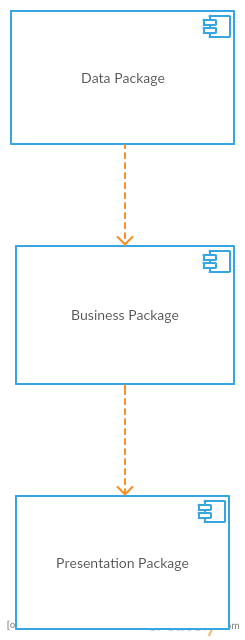
The layered architectural style has been described as an inverted pyramid of reuse where each layer aggregates the responsibilities and abstractions of the layer directly beneath it. With strict layering, components in one layer can interact only with components in the same layer or with components from the layer directly below it. More relaxed layering allows components in a layer to interact with components in the same layer or with components in any lower layer.

Presentation logic is about how to handle the interaction between the user and the software. This can be as simple as a command-line or text-based menu

Data logic is about communicating with other systems that carry out tasks on behalf of the application. These can be transaction monitors, other applications, messaging systems, and so forth. For most enterprise applications the biggest piece of data source logic is a database that is primarily responsible for storing persistent data.

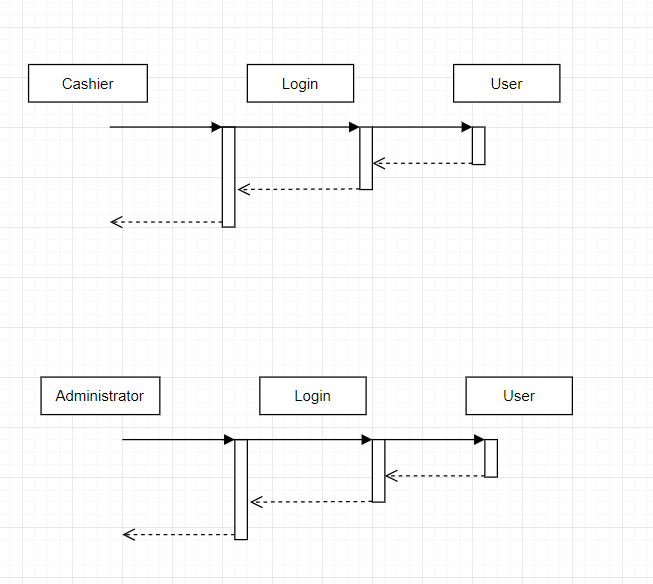
The business logic is about the work that this application needs to do for the domain you’re working with. It involves calculations based on inputs and stored data, validation of any data that comes in from the presentation, and figuring out exactly what data source logic to dispatch, depending on commands received from the presentation.

**3.2 Diagrams**

**

4. UML Sequence Diagrams

I will present a sequence diagram for a log in scenario. User must be an cashier or an administrator to have a successful log in.

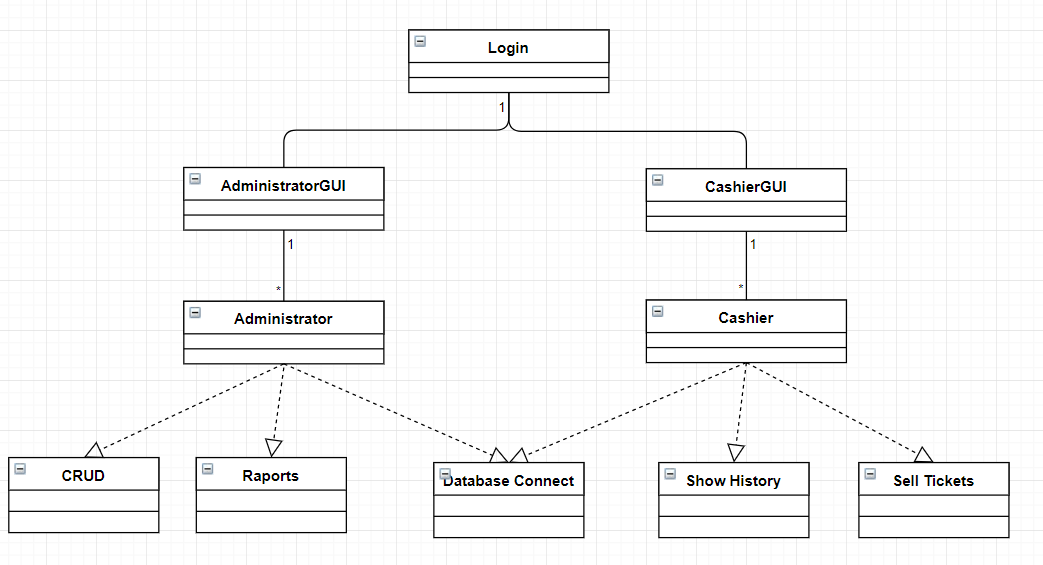


5. Class Design

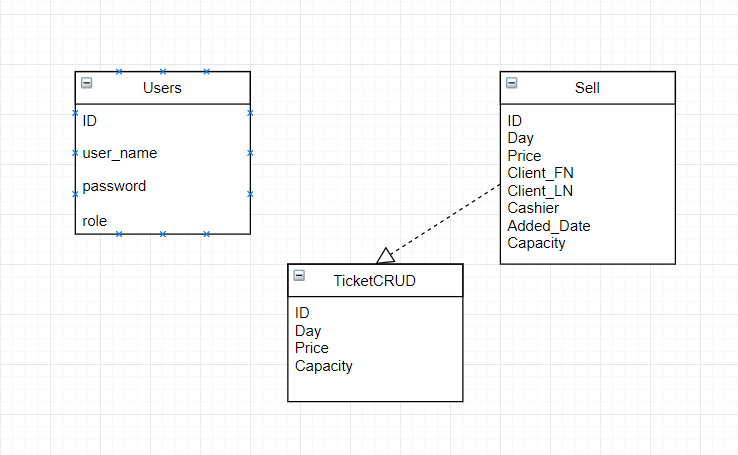
**5.1 Design Patterns Description**

Most business applications can be thought of as a series of transactions. A transaction may view some information as organized in a particular way, another will make changes to it. Each interaction between a client system and a server system contains a certain amount of logic.

**5.2 UML Class Diagram**



6. Data Model



Mixing SQL in application logic can cause several problems. Many developers aren't comfortable with SQL, and many who are comfortable may not write it well. Database administrators need to be able to find SQL easily so they can figure out how to tune and evolve the database.

7. System Testing

For the main operations the system supports tests: delete, update, create, retrieve etc. If something is going wrong the application send an error message to inform the user. Also, the information is tested and validated before register it into the database

8. Bibliography

<https://github.com/vijaythapa333/anystore>

<https://www.youtube.com/watch?v=HEVgdyYSzEE>

https://www.martinfowler.com/eaaCatalog/activeRecord.html

https://www.martinfowler.com/eaaCatalog/tableDataGateway.html

http://tutorialspoint.com/

<http://www.w3schools.com/>