Theater Application
Analysis and Design Document

Student:Lutas Lucian Ioan
Group:30431

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

1.1 Assignment Specification

Use JAVA/C# API to design and implement an application for the National Theater of Cluj. The application should have two types of users (a cashier user represented and an administrator) which must provide a username and a password to use the application.

1.2 Functional Requirements

- Allow the administrator to add, delete, update and get information about users and shows
- Allow the users to cancel reservations
- Allow the users to sell tickets to a show
- Allow the users to edit the seat
- Allow the users to get notifications when the number of tickets per show has been exceeded
- Allow the administrator to view reports regarding sold tickets

1.3 Non-functional Requirements

The data will be stored in a database. Use the Layers architectural pattern to organize your application. Passwords are encrypted when stored to the database with a one-way encryption algorithm. Provide unit tests for the number of tickets for show exceeded validation and the encryption algorithm.

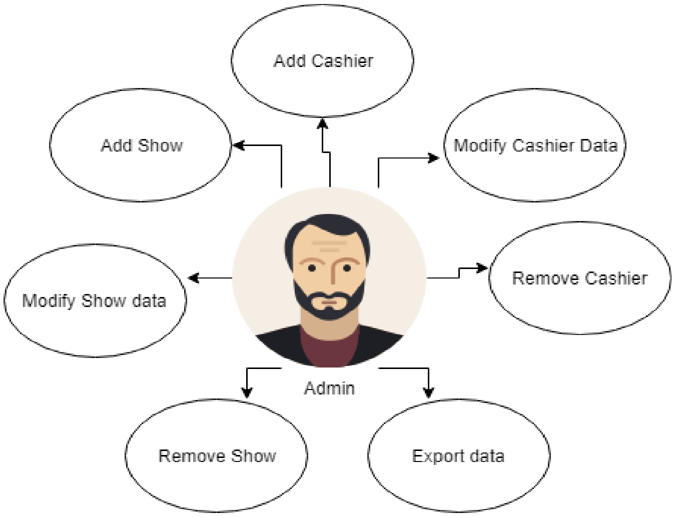
2. Use-Case Model

Use case: Add Cashier
Level: user-goal level
Primary actor: Administrator
Main success scenario:

- Complete username and password fields
- Click login
- Enter username, password and user type
- click the add button

Extensions:

- wrong username or password will not allow you to enter the application
- you cannot create more users with the same username

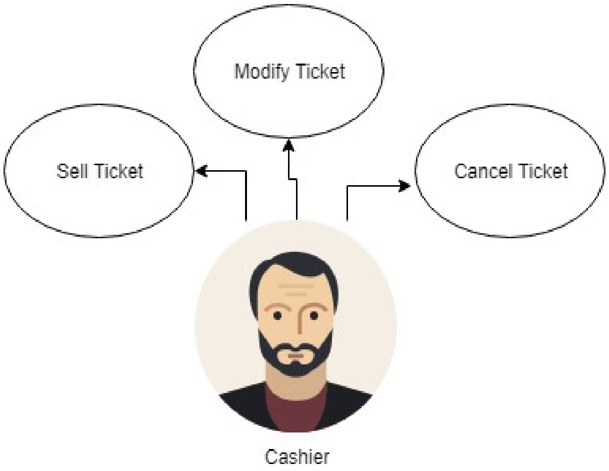


Use case: Sell ticket
Level: user-goal level
Primary actor: Cashier
Main success scenario:

- Complete username and password fields
- Click login
- Filter show
- Select seat from the list
- click the sell button

Extensions:

- wrong username or password will not allow you to enter the application
- you cannot modify a seat to an already sold one



3. System Architectural Design

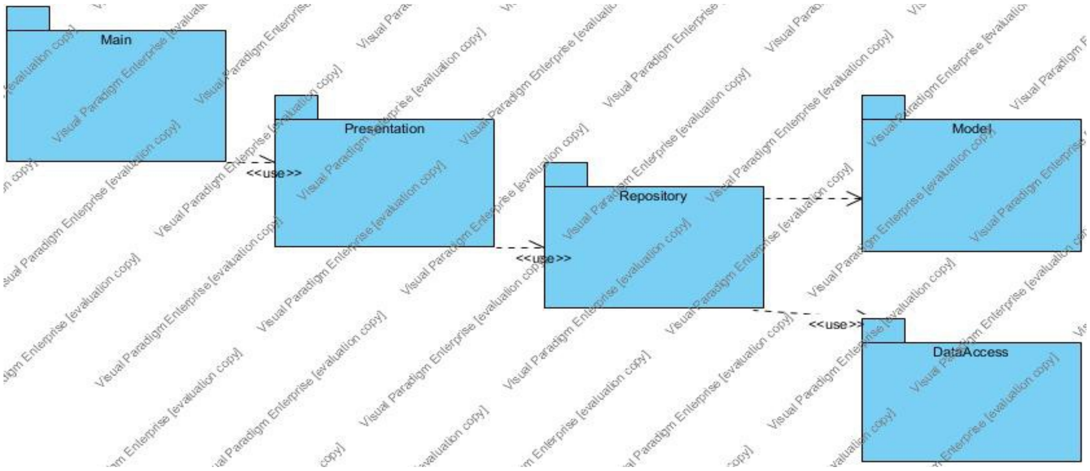
3.1 Architectural Pattern Description

Layered Architecture Design Pattern

There are 3 layers: PresentationLayer, ServicesLayer, DataLayer.
The presentation layer contains 3 forms, one for login, one for admin and one for cashier.
The services layer is split into contracts (containing interfaces), models (containing data models) and services (implementations of the interfaces).
The data layer is split into contracts (containing interfaces), model (containing data transfer object classes) and services (implementations of the interfaces).

We can only access the layer which is one level below the other one, so in order to get from the presentation layer to the data layer we must pass through the services layer. The data layers uses data in DTOs, the services layers uses data in DTOs and Models and the presentation layer uses data only in Models form.

3.2 Diagrams

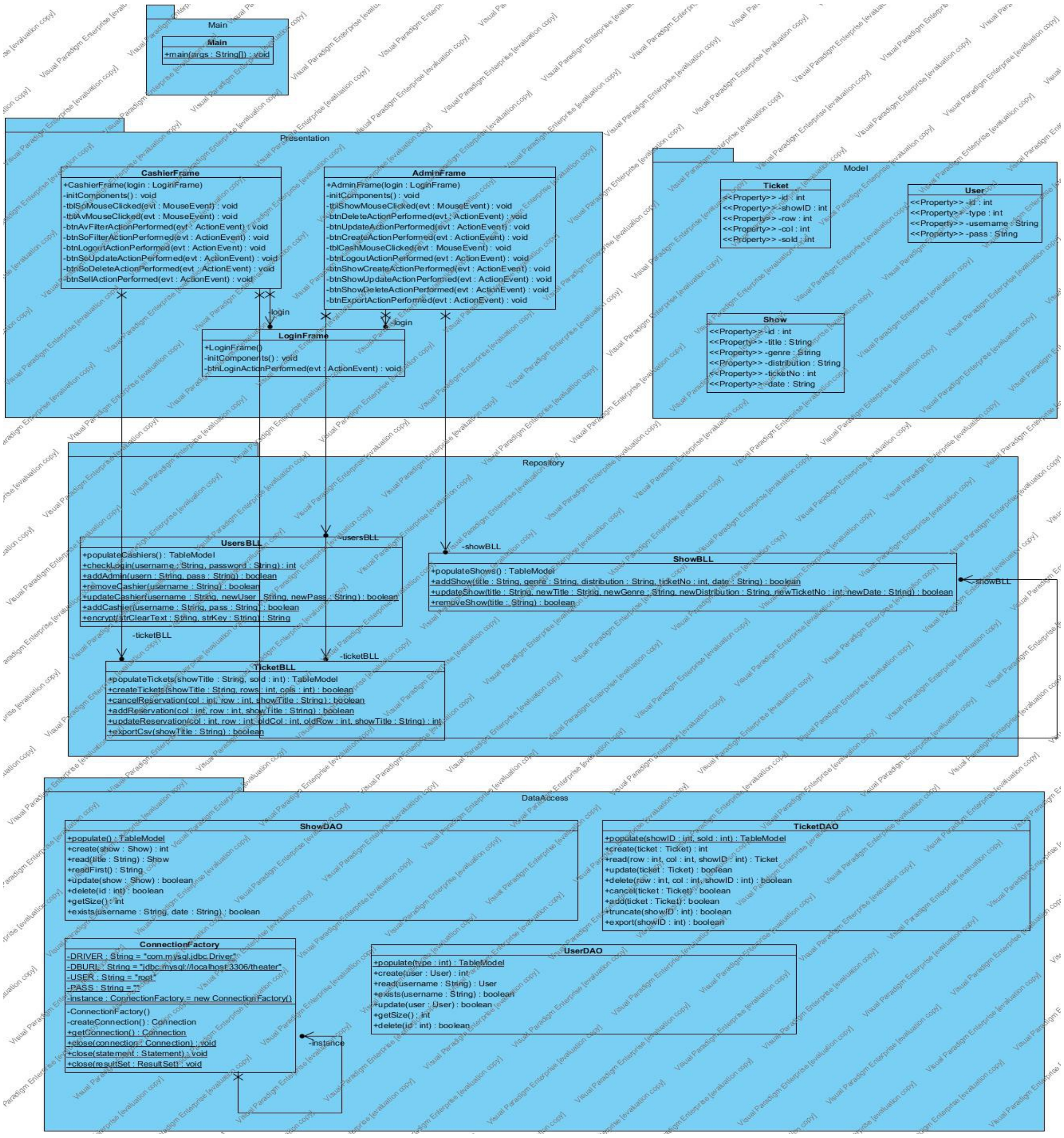


4. UML Sequence Diagrams

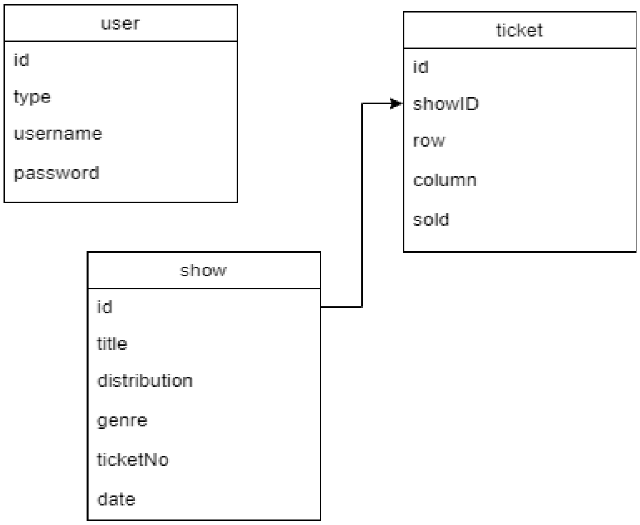
5. Class Design

5.1 Design Patterns Description

5.2 UML Class Diagram


Main
Main
+main(args : String[]) : void
Presentation
CashierFrame
+CashierFrame(login : LoginFrame)
-initComponents() : void
-tblSoMouseClicked(evt : MouseEvent) : void
-tblAvMouseClicked(evt : MouseEvent) : void
-btnAvFilterActionPerformed(evt : ActionEvent) : void
-btnSoFilterActionPerformed(evt : ActionEvent) : void
-btnLogoutActionPerformed(evt : ActionEvent) : void
-btnSoUpdateActionPerformed(evt : ActionEvent) : void
-btnSoDeleteActionPerformed(evt : ActionEvent) : void
-btnSellActionPerformed(evt : ActionEvent) : void
AdminFrame
+AdminFrame(login : LoginFrame)
-initComponents() : void
-tblShowMouseClicked(evt : MouseEvent) : void
-btnDeleteActionPerformed(evt : ActionEvent) : void
-btnUpdateActionPerformed(evt : ActionEvent) : void
-btnCreateActionPerformed(evt : ActionEvent) : void
-tblCashMouseClicked(evt : MouseEvent) : void
-btnLogoutActionPerformed(evt : ActionEvent) : void
-btnShowCreateActionPerformed(evt : ActionEvent) : void
-btnShowUpdateActionPerformed(evt : ActionEvent) : void
-btnShowDeleteActionPerformed(evt : ActionEvent) : void
-btnExportActionPerformed(evt : ActionEvent) : void
-login
-login
LoginFrame
+LoginFrame()
-initComponents() : void
-btnLoginActionPerformed(evt : ActionEvent) : void
Model
Ticket
<<Property>> -id : int
<<Property>> -showID : int
<<Property>> -row : int
<<Property>> -col : int
<<Property>> -sold : int
User
<<Property>> -id : int
<<Property>> -type : int
<<Property>> -username : String
<<Property>> -pass : String
Show
<<Property>> -id : int
<<Property>> -title : String
<<Property>> -genre : String
<<Property>> -distribution : String
<<Property>> -ticketNo : int
<<Property>> -date : String
Repository
-usersBLL
UsersBLL
+populateCashiers() : TableModel
+checkLogin(username : String, password : String) : int
+addAdmin(usern : String, pass : String) : boolean
+removeCashier(username : String) : boolean
+updateCashier(username : String, newUser : String, newPass : String) : boolean
+addCashier(username : String, pass : String) : boolean
+encrypt(strClearText : String, strKey : String) : String
-showBLL
ShowBLL
+populateShows() : TableModel
+addShow(title : String, genre : String, distribution : String, ticketNo : int, date : String) : boolean
+updateShow(title : String, newTitle : String, newGenre : String, newDistribution : String, newTicketNo : int, newDate : String) : boolean
+removeShow(title : String) : boolean
-showBLL
-ticketBLL
-ticketBLL
TicketBLL
+populateTickets(showTitle : String, sold : int) : TableModel
+createTickets(showTitle : String, rows : int, cols : int) : boolean
+cancelReservation(col : int, row : int, showTitle : String) : boolean
+addReservation(col : int, row : int, showTitle : String) : boolean
+updateReservation(col : int, row : int, oldCol : int, oldRow : int, showTitle : String) : int
+exportCsv(showTitle : String) : boolean
DataAccess
ShowDAO
+populate() : TableModel
+create(show : Show) : int
+read(title : String) : Show
+readFirst() : String
+update(show : Show) : boolean
+delete(id : int) : boolean
+getSize() : int
+exists(username : String, date : String) : boolean
TicketDAO
+populate(showID : int, sold : int) : TableModel
+create(ticket : Ticket) : int
+read(row : int, col : int, showID : int) : Ticket
+update(ticket : Ticket) : boolean
+delete(row : int, col : int, showID : int) : boolean
+cancel(ticket : Ticket) : boolean
+add(ticket : Ticket) : boolean
+truncate(showID : int) : boolean
+export(showID : int) : boolean
ConnectionFactory
-DRIVER : String = "com.mysql.jdbc.Driver"
-DBURL : String = "jdbc:mysql://localhost:3306/theater"
-USER : String = "root"
-PASS : String = ""
-instance : ConnectionFactory = new ConnectionFactory()
-ConnectionFactory()
-createConnection() : Connection
+getConnection() : Connection
+close(connection : Connection) : void
+close(statement : Statement) : void
+close(resultSet : ResultSet) : void
-instance
UserDAO
+populate(type : int) : TableModel
+create(user : User) : int
+read(username : String) : User
+exists(username : String) : boolean
+update(user : User) : boolean
+getSize() : int
+delete(id : int) : boolean

6. Data Model



7. System Testing

8. Bibliography