Cinema Booking&Payment Application

Analysis and Design Document

Student: Has Darius

**Group: 30433**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

# This is a Cinema Booking&Payment application that is meant for users in order to book tickets at the cinema, as well as pay them in a safe environment during this pandemic context. The application should be able to let the user decide the time and the place, provide a UI to show the available and unavailable seats, let the user decide the seats that he wants to use and let him book those seats. It also should provide a UI for the admin in order to perform CRUD on movies, bookings and Users, as well as seeing report on different movies and the audiences that the movies had.

# Elaboration – Iteration 1.1

# Domain Model

From the domain model point of view we have the following models: the User, the Movie, the Seat, the CinemaRoom and the Booking and below it is presented a UML class diagram.

Text

Description automatically generated

# Architectural Design

## Conceptual Architecture

*The System will have a layered architecture, having different layers such as: the database access layer, the persistence layer, the business logic layer and the presentation layer. Each classes that belong to a layer will only be able to communicate to the classes from the immediate lower layer. In the database access layer we will have the database connection class, in the persistence layer we will have the classes that correspond to CRUD operations on the database, actually facilitating these operations. Then, in the business logic layer we will have the logic classes for the application, the validation classes and the model classes for the Cinema Booking&Payment application. In the Presentation layer we will have the UI classes that will facilitate the interaction between the user and the application itself. The mediation between the UI classes and the business logic classes and the models will be done through a controller, that is meant for collecting the data provided by the user, fetching it, pre-processing it and creating models from it in order to be passed to the lower level layer classes that will use these models in order to implement the logic of this application.*

*Also, as a Domain Logic pattern, the Table Module pattern will be used as for each table, only one instance of a business logic class will be used to manipulate all the rows, so if we have multiple bookings, one instance will be responsible for performing CRUD operations on them.*

## Package Design

*The Package Diagram is:*

*Text

Description automatically generated*

## Component and Deployment Diagrams

*Deployment Diagram:*

Diagram

Description automatically generated

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

*A first relevant scenario is the signUp/sigIn scenario.*

*This is the sequence diagram:*

*Diagram

Description automatically generated*

*And the communication diagram is:*

User

Service(User Logic from above)

userRepository

signUp

find insert/find

Controller

Sign Up

create

User

*The second relevant scenario is the one in which a reservation is made. A concludent sequence diagram would be:*

*![A picture containing graphical user interface

Description automatically generated]()*

*And the communication diagram is:*

Booking

Service

bookingRepository

send

Create booking validate

Select movie, timeslot, seats

Controller

booking booking

create

Booking

1.2**. Class Design**

**A picture containing text, scoreboard

Description automatically generated**

*From the class diagram point of view, we can see that we have a layered architecture, we can see that we have at least 3 layers (The controller, the Service and the Repo layers), each communicating only with the ones directly below them.*

*From the GoF patterns point of view, a pattern that I can identify is the Observable pattern that should facilitate the communication with the user, specifically, the user should be notified when he/she makes a reservation and if it is or not successful.*

# Data Model

*The data model for this final project would be the User, the Movie and the Seat and can be further extended by containing also a CinemaRoom, in order to extend the application to multiple CinemaRooms, multiple timeSlots, multiple movies. One UML diagram representative of these aspects would be:*

Text

Description automatically generated

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

*Talking about the Unit tests that can be performed in order to showcase the functionality of the application, there are several that can be made and can be grouped into different “user stories” because they can work together to skow that a feature actually works. Some of them would be:*

1. *For sign-in*

*When a user wants to sign in, he must have already signed-up. So, if user is existent and same user wants to sign up, he shouldn’t be able to, same goes with a user that wants to sign in and did not sign up.*

1. *For adding movie*

*Same goes with adding a movie, a movie that already exists shouldn’t be able to be added once more and a movie that does not exist cannot be booked (reservations to that movie cannot be made).*

1. *For selecting movie, selecting seats, selecting time slot*

*When a reservation is made, the movie, timeslot and seats chosen should all be verified, as a reservation cannot be made on already booked seats for a movie at a certain time slots.*

*All these aspects from above can be unit tests and serve as a confirmation of functionality of the features implemented.*

# Elaboration – Iteration 2

# Architectural Design Refinement

*Talking about the package design point of view, the architecture remains the same (We have a layered architecture). The picture presented below is an indicator in this sense:*

Diagram

Description automatically generated

*What has changed from the packeage design point of view are the packages themselves. I have added a package for security, in order for the users to log in and in order for the application to restrict certain users to access certain features (a regular user should not have access to the movies and should not be able to manipulate them).*

*I have also added a WebSocket package that is used when new movies/screenings of a movie are added to notify the pages that display the movies and the screenings that a new entity is added and they should refresh.*

*The Upload package is used for the movies, when it is added in order to upload an image for the movie.*

*The rest of the packages are the same as before, but the naming could have been changed:*

*The entity package contains the model classes.*

*The dao package contains the repository classes (classes used for persisting the entity objects in the database).*

*The dto package contains the data transfer object classes (dtos come from controller and in the service classes objects are mapped from the dtos and these objects are persisted in the database).*

*The service package remains the same.*

*The controller package is the bridge between the html files and the service, serving as a link between the two of them (we do that by using annotations such as @PostMapping and @GetMapping so for each method we know which html file it corresponds to).*

*The report generator class is used for generating pdf report files for the admin such as:*

*A User pdf files containing all the users and their number of bookings and number of seats booked.*

*A Screening pdf file containing screenings sorted by genre and the number of bookings made for each genre of a movie.*

*These extra packages that were added were used to add extra functionality: security for adding a security layer, WebSocket for implementing the Observer DP so that whener a movie or a screening are added the html pages that display them are notified, Upload for facilitating the uploading of image files and reportGenerator for helping the admin user have an extra overview upon the application and its benefits (number of booking can be seen, admin can jump to conclusions as to which genre appeals the most to the public, admin can view the users of the page and their interaction with the application by means of the number of bookings made by each user.*

*From the deployment diagram point of view, nothing has changed as well as for the communication diagrams. For the last ones the common flow is: Information comes from the html files (frontend). It goes to Controller -> Service->Repository.*

*The interaction between the controller and service are done by using the data transfer objects.*

# Design Model Refinement

## *The refined UML class diagram for the entity objects are the following:*

Diagram

Description automatically generated with medium confidence

*The primary change consists of removing the Cinema Room class and replacing it with a Screening class. The meaning of this Screening class is the following:*

*A Screening has a movie, an hour and a map of the seats that are available/booked. So , when a booking is made, it is made for a certain screening. In this way, I could manage easier the booking/availability of the seats so that they are completely disjunct when talking about different screening (If some seats are booked for a certain screening they should not be booked for every screening, but if I had the Cinema Room class this could have been much harder to do because the Cinema Room class would have certain movies, each in a different timeframe so if seats were booked for a movie they would have been booked of the Cinema room and when a booking for these seats would have been made for another movie, this could not have been done, although it should be possible.*

*Talking about the design principle and GRASP, the high cohesion low coupling principle were kept in mind when the classes were designed.*

# Construction and Transition

# System Testing

*From the system testing point of view, the images below will present some of the functionalities:*

Graphical user interface, application

Description automatically generated

*Crud on movies*

Graphical user interface, website

Description automatically generated

Graphical user interface, application

Description automatically generated

*Addition of a screening*

Graphical user interface, text

Description automatically generated

*Incomplete search*

Graphical user interface, website

Description automatically generated

*The seats page view*

Graphical user interface, website

Description automatically generated

*Addition of new movies*

Graphical user interface, application

Description automatically generated

*Selecting your seats for a booking*

Graphical user interface, application

Description automatically generated

*Booking made successfully*

Graphical user interface, website

Description automatically generated

*Booking history for user:*

Graphical user interface, application

Description automatically generated

# Future improvements

*Further improvements for the system can always be made, but some that I think of right now are regarding the user experience such as: the system can recommend to the user a new movie from a certain genre which the user seens to like (several bookings for movies belonging to the genre were made by user). Also, a payment extension could be added for the user in order to not only book tickets to the movie, but also to pay for them.*

*Another improvement would be to extend the application to add multiple cinema rooms, each having a set of screenings and these screenings could be allocated in such a manner that each cinema room is used to its maximum and the number of movies displayed are the highest possible.*

# Bibliography

<https://www.w3schools.com/html/default.asp>

<https://www.w3schools.com/css/default.asp>

<https://www.w3schools.com/js/default.asp>