VIA UNIVERSITY COLLEGE

Project Report

ICT
Engineering

Ovidiu Muresan 254119
Stefan-Daniel Horvath 253724

Course: RE-IT-SEP2

Date: 19 August 2017

VIA UNIVERSITY COLLEGE

Table of contents

Table of Figures

Abstract:

VIA Cinema is an entertainment company, providing customers with auditoriums in which they can see the latest movies, as long as they reserve a seat. They require a software system that will allow customers to make reservations ahead of time, from their own homes, as to reduce crowding in the theaters. The program needs to be able to store data about reservations, customers, schedules, movies, and the cinemas themselves. The data should be stored in an sql database.

In this report is outlined the process of creating the project from an early stage of analysis, through design, to the implementation and testing. Results of the project are stated in its own chapter and the project is summarized in the conclusion.

The software which was developed matches the requirements, however, it cannot be used by the customers clients, as it is incomplete and not fail-proof.

1. Introduction:

VIA Cinema is a local company offering cinematographic services to its clients. The company currently has one cinema in Horsens, but is making plans to expand in the future.

VIA Cinema requires a system that can help customers book reservations online, from the comfort of their homes, which will also help lower the man-power required to serve all the clients wanting to buy tickets directly at the cinema.

The focus of this project is to make an application that allows users to create and manage their own reservations, online, and also be allow them to view cinema schedules.

2.Use Case Diagram

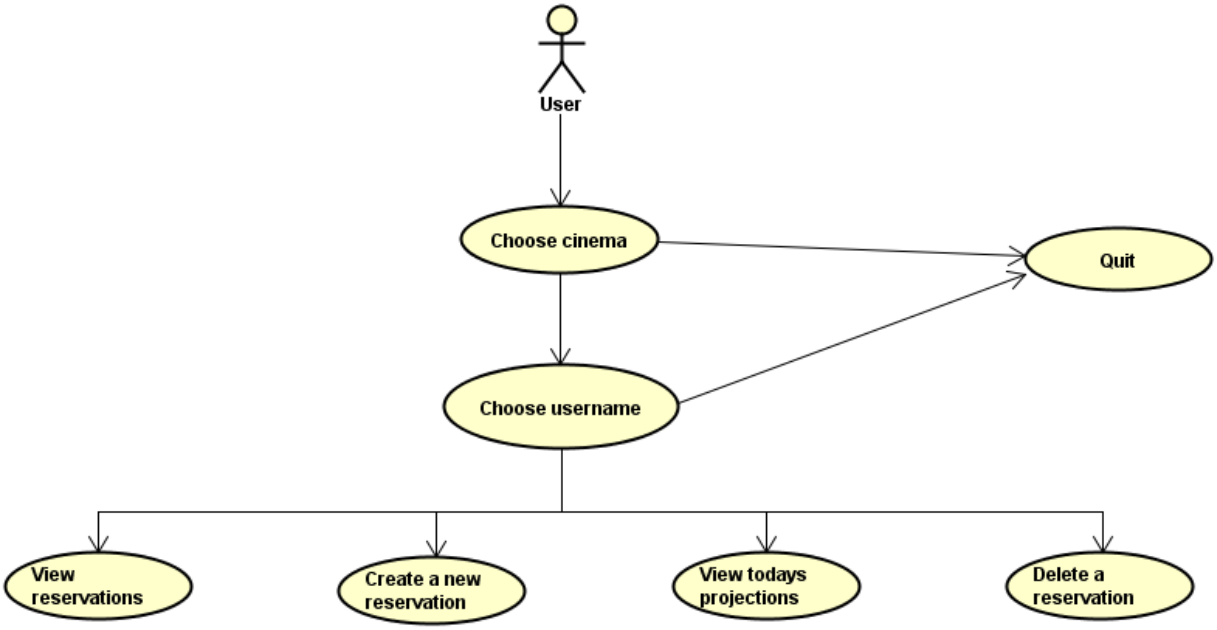

Figure 1 Use case diagram

Choose cinema: Choose a cinema on which to operate.

Choose username: Choose a username.

View reservations: View your reservations.

Create a new reservation: Create a new reservation for the chosen cinema and username.

View todays projections: Displays a list of projections starting within the next 24h for the chosen cinema.

Delete a reservation: Delete a reservation for the current user.

Quit: Stops the program.

3.Use Case Description.

ITEM	VALUE
UseCase	Choose username
Summary	The client chooses a username.
Actor	
Precondition	The user has already chosen a cinema.
Postcondition	The program is going to use the chosen username.
Base Sequence	1. Choose "Choose username". 2. Type username. 3. Confirm if the chosen username is correct.
Branch Sequence	If the user didn't confirm that the username is correct, he can continue from step 2.
Exception Sequence	
Sub UseCase	
Note	

Figure 2 Use case description

4.Activity diagram

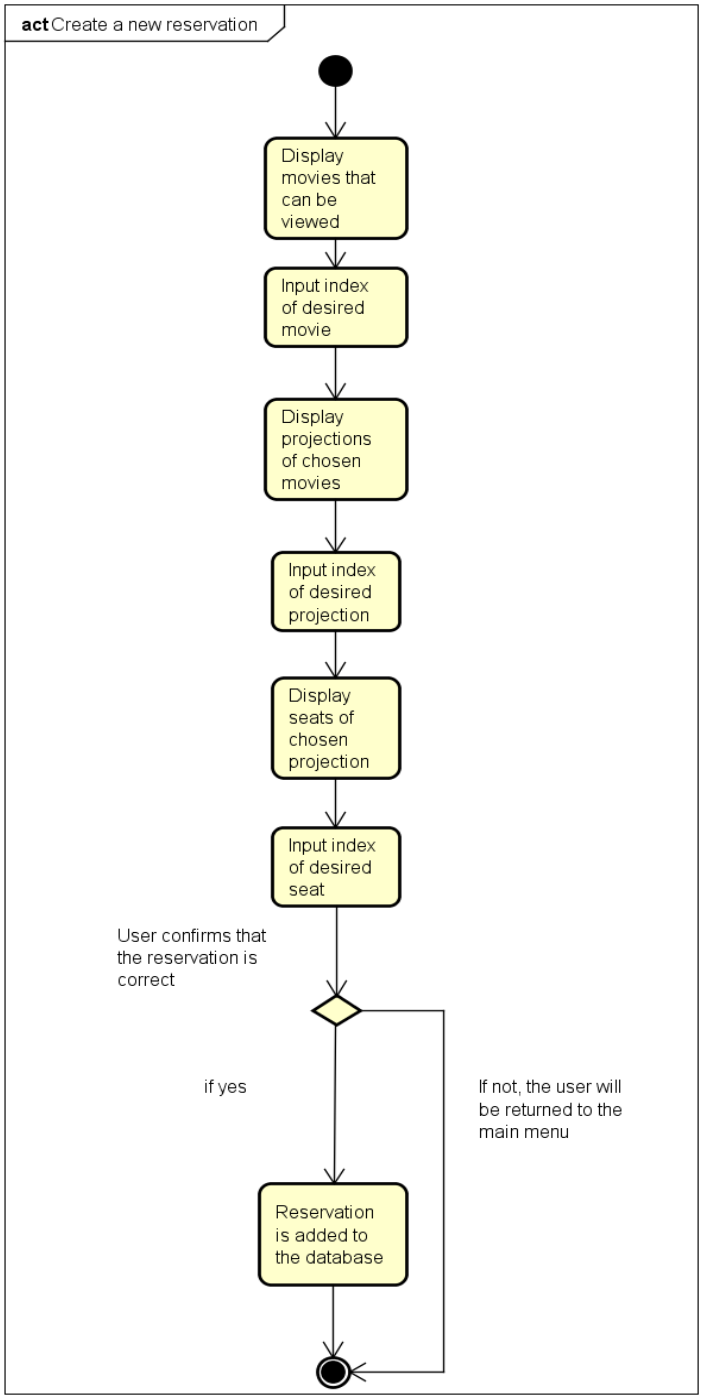

Figure 3Activity diagram

5.UML Class Diagram

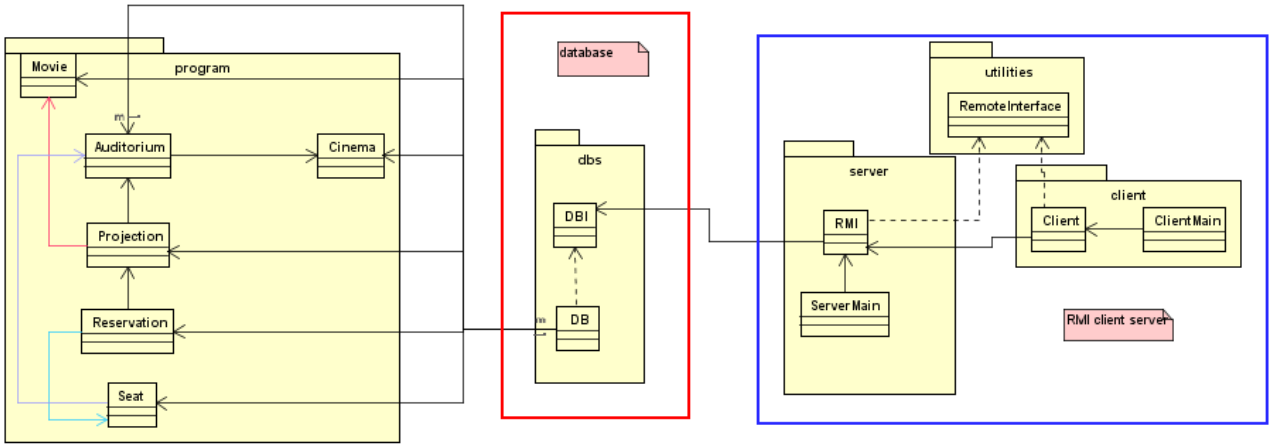

Figure 4 UML class diagram

ViaCinema represents a model package that has variables for all sql tables. The objects from the model are stored in database by taking each element from an object. Then the element are used to create objects with the date stored earlier.

6.Sequence diagram.

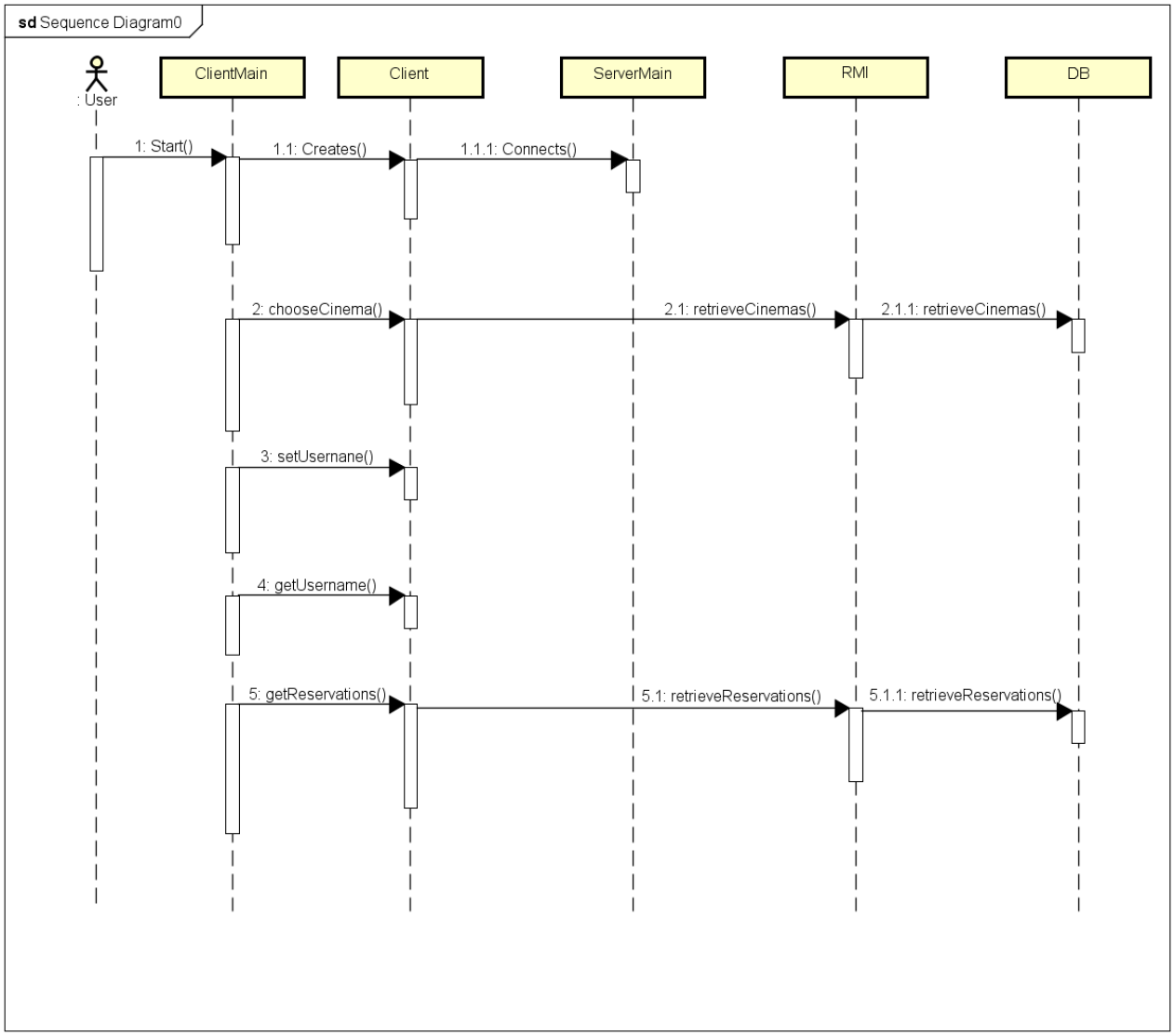

Figure 5Sequence diagram

7.Implementation.

```
@Override
public void addCinema(Cinema cinema) throws SQLException {

    Connection connection = establishConnection();
    PreparedStatement insertStatement;
    try
    {
        insertStatement = connection.prepareStatement("INSERT INTO sep2.cinemas(id,location) "
                + "VALUES (?,?)");
        insertStatement.setInt(1, cinema.getId());
        insertStatement.setString(2, cinema.getLocation());

        insertStatement.executeUpdate();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }


}
```

Figure 6 Adding a cinema in database

The program stores a Cinema object into the database.

```
@Override
public void addMovie(Movie movie) throws SQLException {
    Connection connection = establishConnection();
    PreparedStatement insertStatement;
    try
    {
        insertStatement = connection.prepareStatement("INSERT INTO sep2.movies(title,length) "
                + "VALUES (?,?,?)");
        insertStatement.setString(1, movie.getTitle());
        insertStatement.setInt(2, movie.getLength());

        insertStatement.executeUpdate();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
```

Figure 7Adding a movie in database

The program stores a Movie object into the database.

```
@Override
public Cinema getCinema(int id) throws SQLException{
    cinemas=retrieveCinemas();
    for(int i=0;i<cinemas.size();i++)
        if(cinemas.get(i).getId()==id)
            return cinemas.get(i);
    return null;
}
```

Figure 8Getting Cinemas out of database

The program reads from database the cinemas.

```
@Override
public ArrayList<Reservation> retrieveReservations() throws SQLException {
    ArrayList<Reservation> reservations = new ArrayList<Reservation>();
    PreparedStatement preparedStatement;
    try
    {
        preparedStatement = connection.prepareStatement("SELECT * FROM sep2.reservations");
        ResultSet result = preparedStatement.executeQuery();
        while (result.next())
        {

            Timestamp ts = result.getTimestamp("time");
            LocalDateTime start = null;
            if( ts != null )
                start =  LocalDateTime.ofInstant(Instant.ofEpochMilli(ts.getTime()), ZoneOffset.UTC);
            int auditorium_id=result.getInt("auditorium_id");
            int cinema_id=result.getInt("cinema_id");
            int row=result.getInt("row");
            int col=result.getInt("col");
            String client=result.getString("username");
            Projection proj=getProjection(start, auditorium_id, cinema_id);
            Seat seat = new Seat(row, col, getAuditorium(auditorium_id,cinema_id));
            reservations.add(new Reservation(proj,seat,client));


        }
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
    return reservations;
}
```

Figure 9 Getting a list of reservations out of database

The program creates objects with the elements from database.

```
@Override
public void deleteProjection(Projection projection) throws SQLException {
    try
    {
        int auditorium_id=projection.getAuditorium().getId();
        int cinema_id=projection.getAuditorium().getCinema().getId();
        Connection connection = establishConnection();
        PreparedStatement deleteStatement = connection.prepareStatement("DELETE FROM sep2.projections "
                + "WHERE time=? AND auditorium_id ="+auditorium_id+" AND cinema_id ="+cinema_id);

        Timestamp ts = null;
        if( projection.getStartTime() != null)
            ts = new Timestamp(projection.getStartTime().toInstant(ZoneOffset.UTC).toEpochMilli());
        deleteStatement.setTimestamp(1,ts);

        deleteStatement.executeUpdate();
    }catch (SQLException e)
    {
        e.printStackTrace();
    }

}
```

Figure 10 Delete projection

Delete a specific projection.

8.Test.

The testing was done only for the model. All the problems, that have been discovered by testing the model, have been completely fixed or only provisionally fixed.

Program was tested only with a few fictional entries. Then all methods from DB class were tested.

9.Results.

The program was supposed to facilitate the work and keeping track of the relevant data for the VIA Cinema company. At the time of the deadline, program was still in development.

The model is working as aforementioned in the testing. We miss almost entirely the error handling. At the time of the deadline, the program can be used but if a mistake is made, the program stops.

10.Conclusion.

Main goal of the project was to create a software for VIA Cinema company providing an easy to use system, allowing a customer to work with reservations and keep a schedule for movies.

Analysis and design of the project was done very carefully. However, implementation has only been partly made and includes the model, TUI layout and database. The software can be used as long as the user follows the instructions displayed preventing a crash