Savvas Polydorou – P2526680

Development Project Final Deliverable – CTEC3451

Project Title:

Cinema Ticket Booking System

Supervisor:

Luke Attwood

04 May 2022

Contents

Appendices

Figures

Tables

1. Acknowledgements

Firstly, I would like to thank all the lecturers and lab tutors that have helped me throughout my years at De Montfort University. Without their help and guidance, I would be nowhere near the level I am now, both as a programmer, but also as a person. They were always happy to answer any question about any topic, even during the pandemic, where online learning was established.

I would also like to thank the CTEC3905 (Front-End Web Development) module team, most notably its leader, Dr. Graeme Stuart, who spend extra time to answer any outstanding questions I had about web technologies, such as HTML and CSS, both in the online sessions and labs on campus. His input was extremely beneficial in creating the customer C# side of this project, as I was inexperienced in web development as a whole, but also in aiding in understanding design principles and CSS better.

Additionally, I would like to thank Dr. Martin Stacey, the module leader of CTEC3906 (Interaction Design), and lab tutor of CTEC3905 (Front-End Web Development), for providing excellent material each week so that, by the end of the module, I would be able to apply effective design principles in my projects, but also evaluate existing systems to differentiate the pros and cons of their design choice. This module was eye-opening in terms of taking the role of a non-programmer and designing the friendly graphical interfaces of this project so that users can have a pleasant experience using it.

Lastly, I would like to thank my supervisor, Mr. Luke Attwood, who has been more than happy to help me with all my requests. He changed my way of thinking as a programmer, and also gave me the drive to want to keep improving myself and learn more about the different principles, concepts, and technologies he taught me these past three years. Not only has he taught me everything I know about Java, that quickly became my favourite language to code, but he also motivated me to make my own research and learn new techniques that were not part of the modules he was a part of. Without him, this project would not be a reality, as he kept pushing me to add features I never thought that I would be capable of implementing, but his belief in me was more than enough to keep me going. He is also the person who helped me land a graduate job in software development, and for that, I could not be more grateful.

2. Introduction

2.1 The Software

The Cinema Ticket Booking System is divided into two different systems that talk to each other and communicate via a database. There is a desktop application that is only used by the Cinema Owners and Admins, and a web-based system that is accessible to Customers and Guests that want to book tickets and reserve seats for a particular movie, at a given time and date. The desktop application is written in Java using the JavaFX library, and the web-based system is implemented in C# using the ASP.NET framework, and a Microsoft SQL Server database is used by both systems so they can send and retrieve data to communicate with each other. The Cinema Owners have the ability to add, view, and delete Admins, view any movie that exists in the database regardless of its status (archived, playing now, or coming soon), search for a movie in the system and view its details, bookings, and trailer. Admins are also able to browse movies depending on their status, with the main difference being that only Admins can add, edit, or remove a movie from the system. They can also view all the bookings made for a movie and its trailer, but more importantly, they can delete a booking, if requested. Customers can create an account and view the playing now or coming soon movies only, since the archived ones are of no concern to them. They can view a movie's details and trailer, and can then proceed to book a ticket and reserve seat(s) for a particular date and time. After entering the information requested, they receive an email confirmation that contains the ticket details (movie poster and title, reserved seat(s), ticket holder name etc.). This functionality is also available for Guests (customers without registering an account), but they do not have the ability to change their reserved seats, movie date and show time, or cancel the booking, unlike logged in customers. Cinema Owners, Admins, and registered Customers have the ability to update their account details, reset their password, login, and logout etc. Data validation and attention to detail are vital and key components of this project to ensure that the software can be used as intended, with minimal to no errors. Duplicate usernames, weak passwords, alphabetic characters in phone numbers, incorrect email formatting, future date of birth, reserve already booked seats, book a ticket for a date and time that has passed etc. are **not** allowed. By studying already-established movie theatre systems and being inspired by online streaming services, the Cinema Ticket Booking System mimics the behaviour and functionality of such systems in order to provide a pleasant and smooth customer experience, and be accessible and usable to people without prior experience in using such systems.

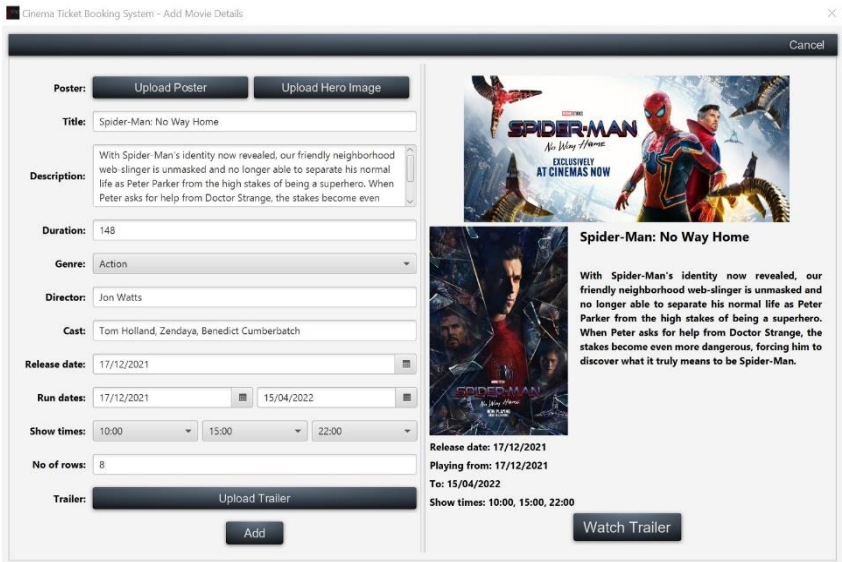

Figure 1: Admin adds a movie to the system

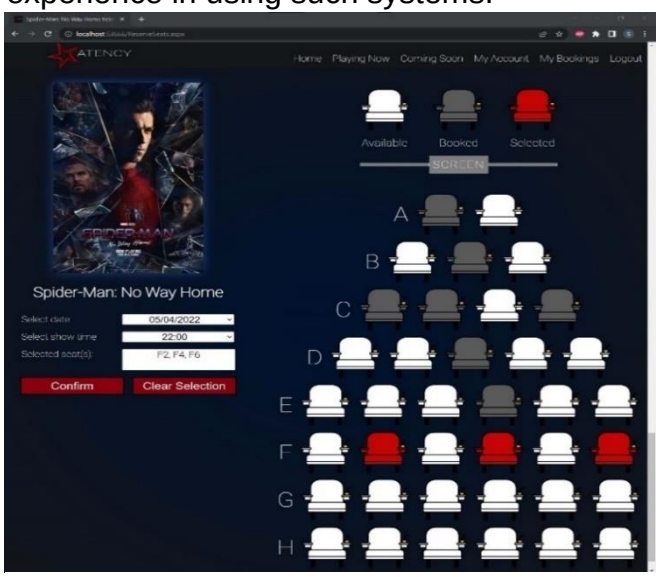

Figure 2: Customer books ticket and reserves seats

2.2 Why is a computerized booking system needed?

This product is needed in our modern times, as almost all businesses prefer to go digital to minimize the use of pen and paper, be more agile and have a vigorous resource management. By using the desktop application, Cinema Owners and Admins can work faster and more efficiently, and most importantly, be more organized [1]. The system takes care of how many bookings are made for a movie and displays each booking information so that the staff members can confirm that the person entering the theatre has indeed booked their ticket. Even if the customer forgets to bring their mobile device with the email confirmation to enter the theatre, the Cinema Owner or Admin in charge can verify that the customer exists in the bookings table in the desktop application, by asking for certain personal information that was provided by the customer whilst in the process of booking the ticket. By using automations, movies are placed in their respective status category; archived, playing now, or coming soon. There is no need for the Admins to check each movie's playing dates every day to determine its status, as it is determined based on the release and playing dates of the movie, and are automatically checked with each successful login. Not only is the whole business accessible at any time of the day, but customers are able to see if a theatre has no available seats for a certain film screening, so they can choose a different date or time. Time is not wasted on having to call the cinema to get access to this information, and disappointment on arrival is completely eliminated. Moreover, long queues are avoided since the booking process can be done online and at the comfort of one's home. Registered customers can also easily cancel their booking or change their seat, date, or time online, by using the web-based system, without the guidance of a staff member. I come from a city where the local cinema uses the traditional booking procedure and offers no online systems to book a ticket, which is frustrating and unacceptable for the era we line in. Long waiting queues, having to call the cinema to ask certain questions about the details of movies that are currently playing or will soon come out, unsure of how full a theatre is etc. are all problems that should be left in the past and be replaced with computerized systems, such as the Cinema Ticket Booking System.

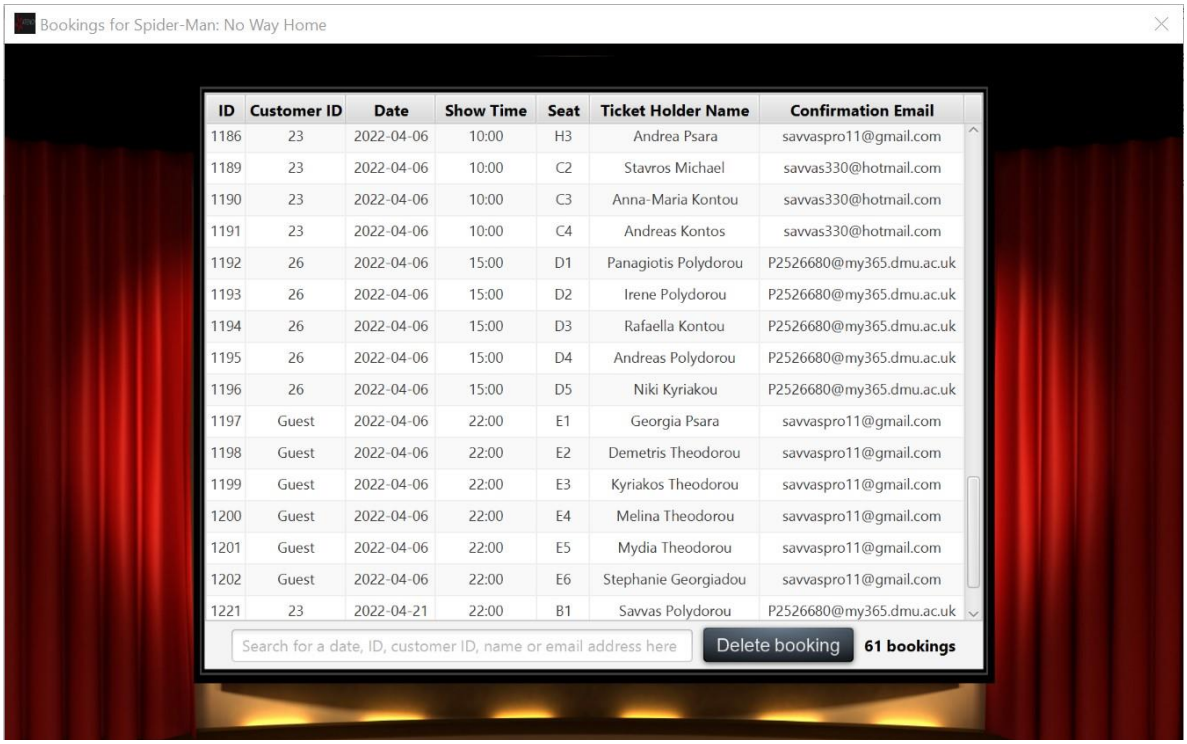

Figure 3: Cinema Owner/Admin viewing all the bookings for a movie

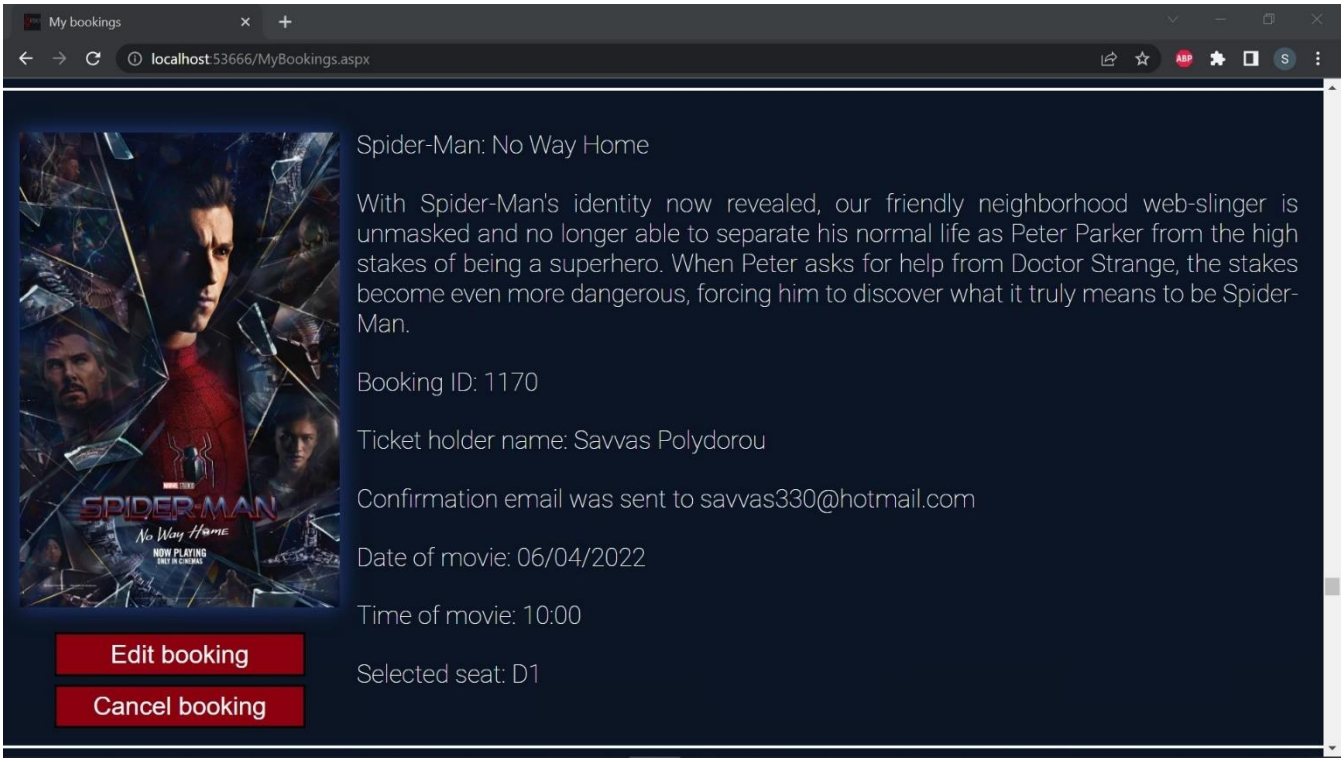

Figure 4: Customer being able to edit or cancel a booking

2.3 Objectives Recap

- To produce a literature review based on the area of interest
- To produce functional requirements that facilitate several different user types e.g. administrator, staff member, customer, guest.
- To produce appropriate design documentation to assist the smooth and reliable use of the software product e.g. using interaction design techniques, to ensure the software is fairly simple and easy to use to an untrained eye
- To write test code/plans that reflect the use cases
- To develop the cinema booking system software with a focus on maintainability
- To produce a final report and critical evaluation of the project

2.4 Objectives Achieved

All objectives found in the updated list of functional requirements (see Appendix A) have been implemented and all work as expected, with no errors. Both systems have been fully developed by scratch, without the use of external APIs or libraries. Test code has been written for the Java application, focusing on the user's inputs to ensure that the system cannot be misused. Appropriate database tests, such as adding and removing movies was also considered (see Appendix B). For the web-based system, writing test code would be redundant, as ASP.NET's TextBoxes (HTML input equivalent) have a min, max, minlength, maxlength, pattern, text mode etc. properties that takes care of the desired input format.

3. Research

3.1 Focused areas

Briefly touched upon subsection 2.2, the major area of focus that the research was based on was to offer insight on the benefits of cinema ticket booking systems and explain how inconvenient traditional theatre booking procedures were, compared to the more modern procedure; by using a computerized system to manage business needs and offer accessibility to customers at any given time of the day. Prior to the use of a digital system, the preferred method to obtain a ticket was to visit a theatre and purchase them physically, during business hours at a box office [2]. This proved to be very inconsistent and inconvenient, as there could be a long waiting queue and tickets could be sold out, or one's preferred seats could be booked by someone else, if the customer did not arrive early to stand in the queue. Moreover, there was no way of knowing which movies were playing at the time or were soon to be released and their showtimes without reading a newspaper, or how full a theatre is and which seats are available, resulting in the absolute necessity of contacting a staff member during working hours or visiting a theatre in person [3]. Not only did the customers had an unpleasant time going through this process, but staff members had to present a paper plan of the theatre to the person being treated for them to select their seats, which proved to be messy due to seat change requests and booking cancellations. Furthermore, it was expected that the employees would have advanced knowledge of the context of the movies that were being advertised at a theatre, to be able to solve any query a customer may have. Finally, business analytics such as how many people booked a ticket for a particular movie at a given date and time would need to be manually evaluated, as there were no means of automated calculations without the use of software. Evidently enough, with the introduction of digital systems, such as the Cinema Ticket Booking System, customers can access the content of a company online whenever they please, and are presented with a digital seat selection plan and are able to view all the details of a movie, including a video trailer that helps the viewer understand the plot better. Additionally, the management of bookings has become easier, as registered customers have the ability to alter their ticket information or cancel it entirely, without the need of contacting a technical person or visiting a theatre in person. Use of paper is eliminated entirely, as the treasurers interact with a graphical interface and are able to continue performing their tasks faster, more efficiently and be more organized.

3.2 Identification of clientele and functional requirements

In addition as to why this project was chosen, further research was carried out to identify the targeted audience and their functional requirements. As stated earlier, an important benefit of shifting into a digital model is that the business is accessible anytime, and so several ticket booking web systems were thoroughly inspected and used in order to identify the most common and major functionalities a customer should be able to perform. Whilst inspecting, it became clear that all of the web systems shared two different user types for their clients; a registered customer, and a guest that has not created an account. By not having to register an account, the process of using the system's main purpose, which is to book a ticket and reserve seats, could be performed faster. Speed has a significant impact on customer experience and the success of a company, as customers tend to be unwilling to keep using a system if it

is not responsive enough [4]. This led to the inclusion of the Guest user type in the Cinema Ticket Booking System and the addition of several use cases to differentiate Customers from Guests, as initially, this user type was not part of the system. It became apparent that registered customers have the ability to view, manage or cancel their bookings, which eliminated the need of contacting an administrator to perform these actions for them and was the inspiration for adding these functionalities and the Guest user type in this project [5]. Furthermore, a seat legend was also shared amongst the different systems that were studied, in order to provide a better understanding of the theatre's structure and help the user decide where they would like to sit for a smoother viewing experience. Additionally, the legend provided a visual guide representation of the available, selected and booked seats to aid the user in identifying the status of a seat, which was later included in this project as well. While the requirements of the customers expanded, the need to implement more advanced features in the desktop application also increased. For instance, a video player needed to be implemented for the staff members to be able to watch the trailer they uploaded to minimize room for error before adding the movie to the system. As previously mentioned, a trailer is essential to provide the customer with some context for a film. The ability to cancel a booking with the aid of an Admin was also inspired by these systems, due to the Guests not being able to view or manage them as they do not have an account.

3.3 Key findings

- The shift into a digital model is essential
- Opening up data to the public allows the business to be accessible anytime
- Staff members need to be able to view all the bookings made for a movie digitally for a more organized workspace
- Registered customers should be able to view/update/cancel their bookings online
- The allowance of certain major functionalities to unregistered customers is vital
- Guidance through appropriate layout and design greatly improves customer satisfaction and experience

4. Analysis

4.1 Types of users

The desktop application is run by the Cinema Owners/Admins, and the web-based system is meant to be used by the Customers/Guests, amounting to 4 different user types for both systems. Brief descriptions of the role each user type has in the system can be found below.

4.1.1 Cinema Owner

The Cinema Owners are responsible for adding the Admins, so that they can run the business through the desktop software. They are also able to view all the Admins of the system, and optionally delete them, as this user type is equivalent to a business owner hiring, viewing, or firing employees. Business analytics, such as all the details of the movies that have been, are currently, or are coming to the cinema are accessible to them, along with all the bookings made for any movie. Cinema Owners have a total of 13 use cases.

4.1.2 Admin

The Admins represent the personnel of the company, as the task of managing the system's content, such as adding, editing, or removing movies is handled by them. They cannot register to the system by themselves, as they have to be "hired" by a Cinema Owner. They also have access to the previously mentioned business analytics, and just like Cinema Owners, they have the ability to search for and verify a booking, as a customer is very likely to interact with a staff member to show their ticket before entering a movie theatre. Unlike Cinema Owners, they can delete a booking, should the customer request so. A total of 16 use cases were identified for the Admins.

4.1.3 Customer

The Customers are considered as the clients of the company, as they are able to book a ticket and reserve one or multiple seats, according to their preference and availability. They only have the ability to view the currently playing and coming soon movies of the system, as the no longer playing movies are of no concern to them. The main difference between Customers and Guests is that Customers are registered and have a personal account, in which they are able to update their details and delete it, view all the bookings they made in the system, and can optionally edit a booking or cancel it (if the movie has not been shown yet). 14 use cases were identified for this user.

4.1.4 Guest

The Guests are also considered to be the clients of the company and can perform main tasks such as booking a ticket and reserving seats, but once a booking has been made, they cannot manage or cancel it by themselves, as they do not own an account. They need to contact an Admin in order to have their booking cancelled, but changing the details of an existing booking is not allowed. Just like the Customers, they can view the playing now and coming soon movies of the system only, making a total of 5 use cases.

4.2 Core functional requirements descriptions

The functional requirements below were the inspiration in identifying the 4 different types of users the Cinema Ticket Booking System aims to target. The tables below include only the core use cases and offer a description and the pathways the feature can be achieved or failed, but most importantly, they justify the reasons behind creating these users and are essential to run the business. The descriptions of the less important functionalities can be found in Appendix C (parts 1-4).

4.2.1 Cinema Owner

Table 1: Use case for adding Admins

UC001	
Title	Add Admins
Description	The Cinema Owners are able to add Admins to the system
Primary pathway	1. Cinema Owner enters the system after registering 2. Cinema Owner is presented with a menu where the add Admin functionality exists 3. Cinema Owner fills in details and registers an Admin
Exception pathway	1. Cinema Owner authentication failed due to incorrect credentials 2. Admin cannot be added due to invalid data formatting or the username is taken; Cinema Owner is presented with a set of instructions

Table 2: Use case for removing Admins

UC002	
Title	Remove Admins
Description	The Cinema Owners are able to remove Admins from the system
Primary pathway	1. Cinema Owner enters the system after registering 2. Cinema Owner is presented with a menu where the remove Admin functionality exists 3. Cinema Owner selects an Admin and presses the Remove button 4. An alert pops up, the Cinema Owner presses the “Yes” button to confirm the action
Exception pathway	1. Cinema Owner authentication failed due to incorrect credentials 2. Cinema Owner does not select an Admin and presses the Remove button, an alert with an appropriate message is displayed 3. Cinema Owner selects an Admin and presses the Remove button, an alert pops up, the Cinema Owner presses the “No” button to cancel the action

4.2.2 Admin

Table 3: Use case for adding movies

UC052	
Title	Add movie to system
Description	The Admins are able to add a movie to the system
Primary pathway	1. Admin enters the system 2. Admin is presented with a menu where the add movie functionality exists 3. Admin proceeds to fill in details and add a movie
Exception pathway	1. Admin authentication failed due to incorrect credentials 2. Admin enters invalid data or does not upload a poster, hero image and trailer; an alert pops up to offer further advise

Table 4: Use cases for removing movies and their bookings

UC053, UC066	
Title	Remove movie and all its bookings from system
Description	The Admins are able to remove a movie and all its bookings from the system
Primary pathway	1. Admin enters the system 2. Admin is presented with a menu where the browse movies functionalities exist and clicks on one 3. Admin selects a movie and presses the Remove button
Exception pathway	1. Admin authentication failed due to incorrect credentials 2. Admin does not select a movie and presses the Remove button; an alert pops up to offer further advise

Table 5: Use case for editing movie details

UC054	
Title	Edit movie details
Description	The Admins are able to edit a movie's details
Primary pathway	1. Admin enters the system 2. Admin is presented with a menu where the browse <*movie status*> movies functionalities exist and clicks on one 3. Admin selects a movie and presses the Edit button
Exception pathway	1. Admin authentication failed due to incorrect credentials 2. Admin does not select a movie and presses the Edit button; an alert pops up to offer further advise

Table 6: Use case for deleting a booking

UC065	
Title	Delete a booking
Description	The Admins are able to delete a booking
Primary pathway	1. Admin enters the system 2. Admin is presented with a menu where the browse <*movie status*> movies functionalities exist and clicks on one. Alternatively, the Admin clicks on the search for a movie control and skips to step 4 3. Admin selects a movie and presses the View Details button 4. Admin presses the View Bookings button and then selects a booking and presses the Delete button 5. An alert pops up, the Admin presses the "Yes" button to confirm the action
Exception pathway	1. Admin authentication failed due to incorrect credentials 2. Admin does not select a movie and presses the View Details button; an alert pops up to offer further advise 3. Admin does not select a booking and presses the Delete button; an alert pops up to offer further advise 4. Admin selects a booking and presses the Delete button, an alert pops up, the Admin presses the "No" button to cancel the action

4.2.3 Customer/Guest

Table 7: Use cases for registering and logging in Customers

UC101, UC102	
Title	Register/Login Customers **only**
Description	The Customers are able to create an account and login
Primary pathway	1. Customer enters the system 2. Customer clicks on the Register button 3. Customer fills in details to create an account and then logs in
Exception pathway	1. Customer enters invalid data formatting or the username is already taken; an error appears 2. Customer authentication failed due to incorrect credentials

Table 8: Use case for finding movies based on status

UC103, UC151	
Title	Find movie based on status
Description	Clients are able to find movies based on status
Primary pathway	1. Client enters the system 2. Client is presented with 6 playing now and 6 coming soon movies on the Home page 3. Alternatively, the client can click on the playing now or coming soon buttons to find all the movies of that status and reveal more information
Exception pathway	N/A

Table 9: Use cases for viewing movie details,trailer,booking ticket/reserving seats

UC104, UC107, UC109, UC110, UC152, UC153, UC154, UC155	
Title	View movie details Watch a movie's trailer Book ticket Reserve seat(s)
Description	Clients are able to view the details of a movie and its trailer, book a ticket and reserve seats
Primary pathway	1. Client enters the system 2. Client finds a movie of preference and clicks on the "More Info" button and to view its details and trailer, and clicks the Book ticket button. If the client is on the home page they can click on the poster of the movie instead. 3. Client is asked to select a date, time, and an available seat
Exception pathway	1. Client tries to select an already reserved seat and is unable to click it

Table 10: Use cases for viewing account's bookings, updating, or cancelling them

UC112, UC113, UC114	
Title	View account's bookings Update booking date, time, or seat Cancel booking Customers **only**
Description	Customers are able to view all of their bookings and if the movie has not played yet, they can update or cancel their booking
Primary pathway	1. Customer enters the system 2. Customer clicks the My Bookings button 3. Customer finds a booking of preference and can edit or cancel it 4. To edit a booking they are asked to select a new date, time, or an available seat 5. To cancel a booking they need to confirm their action first
Exception pathway	1. Customer authentication failed due to incorrect credentials 2. Customer tries to edit or cancel a booking but is unable to because the movie has already played in the theatre 3. Customer tries to select an already booked seat and cannot proceed 4. Customer does not confirm that they wish to delete their booking

5. Design

Like every other project, the two systems need to be as easy and straight-forward to use as possible, thus, design consideration, both presentation-wise and structure-wise was a key factor for this project. Aided by CSS, the presentation is project-appropriate and has a professional look and feeling, as both systems were inspired by multiple existing systems, with a touch of personal preference included. By using the MVC pattern for the desktop application, low coupling and high cohesion is achieved, hence presentation changes can be easily made without breaking any parts of the software logic. It is worth noting that no GUI builder, such as JavaFX FXML or drag-and-drop features were used for the creation of these systems, but were created using pure code instead.

5.1 Front – End

As more people are expected to use the web-based system to book a ticket and reserve seats, it has a plethora of animations and eye-catching pictures, a friendly and welcoming font style, on mouse hover styles that indicate whether a user is allowed to perform an action or not, and a seat legend that clearly showcases how an available, booked, or selected seat is supposed to look like to assist the user in identifying the status of a seat (see Appendix D). In terms of the desktop application, JavaFX also allows the styling of components, but have slightly different names and syntax than their typical web counterparts, and the application was styled in a way that it looks professional and put together, rather than aiming for a friendlier and more welcoming feeling.

5.2 Back – End

The two systems were developed in Java and C# mostly due to their syntax similarity and the fact that they are both object-oriented. By combining the JavaFX library with the Model-View-Controller pattern, the code is more reusable and code repetition is significantly reduced. This was extremely beneficial as screens such as the registration form and the overall user menu are both written in one file respectively, and it is only a matter of changing the controls depending on the user type; Cinema Owner or Admin, via the controller. The trailer player is also reused, either as a stand-alone screen with controls and sound, or as a muted and dimmed background trailer with the movie's information stacked on top of it, acting as a preview of the movie. Screenshots of said screens and more can be found in Appendix E (parts 1-6). For the web-based system, by applying the Three Tier Architecture pattern the interface is unaware of the structure of the database, so that the communication between them remains unaffected in the case of a UI redesign. Thus, the business logic continues to perform as it is, even if the look and structure of the websites are changed [6]. The communication between the two systems is achieved via the JDBC drivers and the SqlConnection class (Java and C# respectively), by sharing the same database and performing actions such as writing to and reading from the same database table (see Appendices F and G).

5.3 Database

A localhost Microsoft SQL Server database with SQL Server Authentication was created for this project and there is a table for the Cinema Owners/Admins, Customers, bookings, and movies (see Appendix H). As the Cinema Owners/Admins need to be able to view the bookings for a particular movie, find a customer's booking using their unique ID, or use the ticket ID, the bookings table is constructed in a way that it references this information. Most importantly, it is designed using Single Association Referencing, which means that it references the Customer and movie ID, so they can be cross-referenced to grab the required information from the other tables respectively. Thus, by using the booking table's CustomerID and MovieID columns, it is possible to get all the bookings for a specific movie, all the bookings that a customer made, and the Cinema Owners/Admins can use both of these columns to find the ticket of a customer for a particular movie. Furthermore, the username column is constructed in a way that it does not allow duplicate values, by using SQL's UNIQUE constraint, due to the fact that all usernames in a system must be linked to only one account. By default, SQL columns are not case sensitive, so the password column is forced to respect case sensitivity, so the value that the user entered is an exact match of the stored password [7]. This is achieved by modifying the Collation property of the column, or in this project's case, this restriction is achieved by adding "*COLLATE SQL_Latin1_General_CP1_CS_AS*" in the SQL query via the back-end data connection classes. Finally, to store the images and video files for the users and movies, their local paths are stored in the database tables so that the database remains light, reliable, and fast. The paths are read from the database tables and are used to load the files from the local drive in both systems.

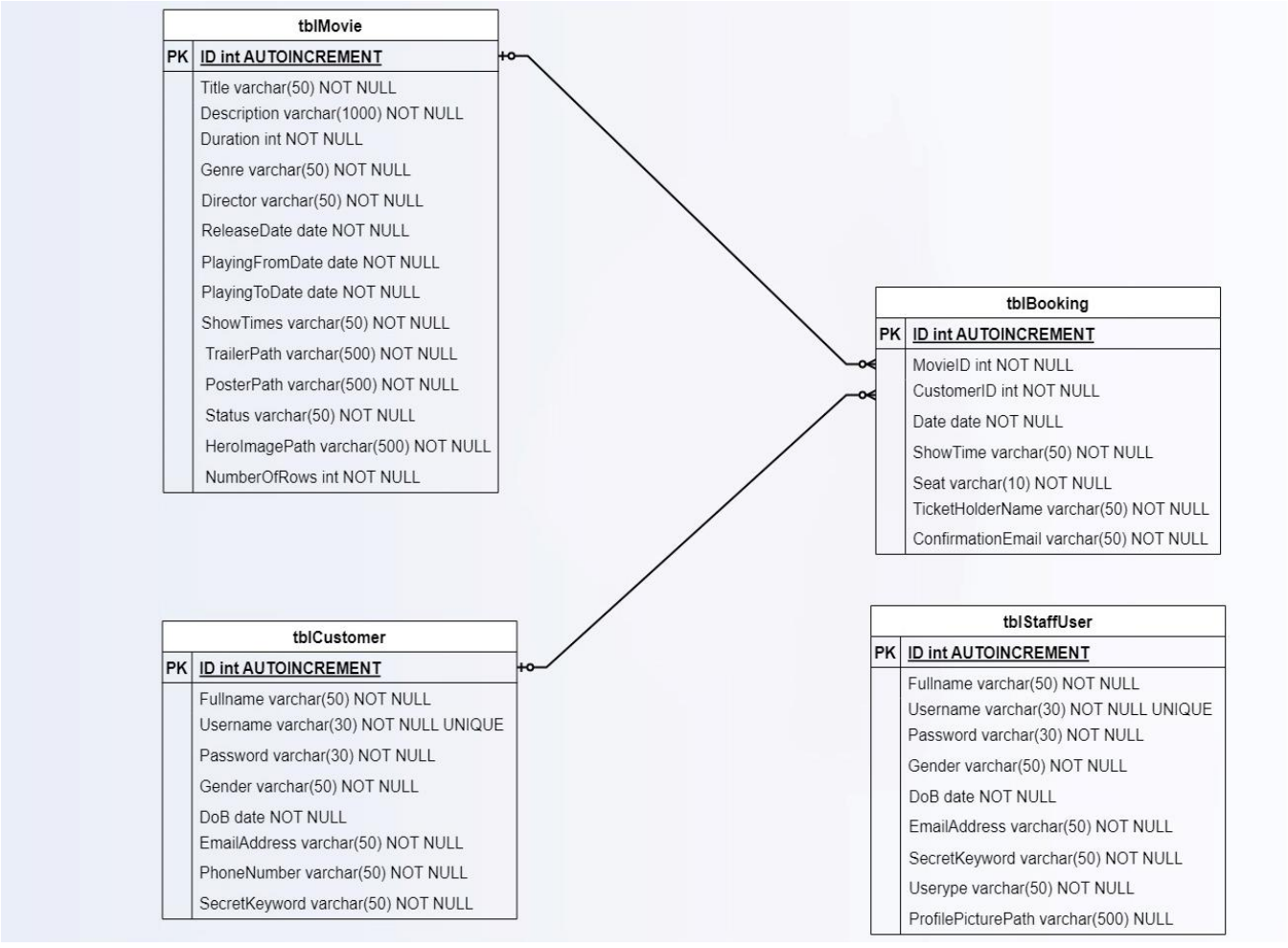

Figure 5: Entity Relationship Diagram

6. Implementation

6.1 Development Lifecycle

As previously mentioned in the January deliverable, this project was completed with the adaptation of the principles of Agile Methodology in order to meet the project's deadline. The Scrum framework was picked, and most specifically, Personal Scrum as there is only one developer, due to the fact that every two weeks a meeting was held to discuss the project's successes and problems faced at each stage, and its future functionalities. The supervisor acted as the customer of the product, which this framework highly involves in order to receive feedback during development [8]. As this framework is iterative and incremental, it allowed room for more growth in terms of how functional the two systems were going to be. The desktop application was built first, and then the web-based system, hence the adaptation of this model allowed the desktop software to be revisited while the web-based system was still being developed, to add more functionality and increase the level of communication between them. Features such as adding a hero image for each movie so that the home page of the web-based system would look more professional and welcoming, adding the number of rows of seats for each movie to mimic the behaviour of multiple cinema locations or screens (since this falls outside the project's scope) etc. were thought and considered only after the development of the web-based system begun. Therefore, it was appropriate to adapt this framework so that there was better control of the project, increased predictability over the overall performance and how functional it actually is, and most importantly, whether the project would be finished by the agreed deadline or not.

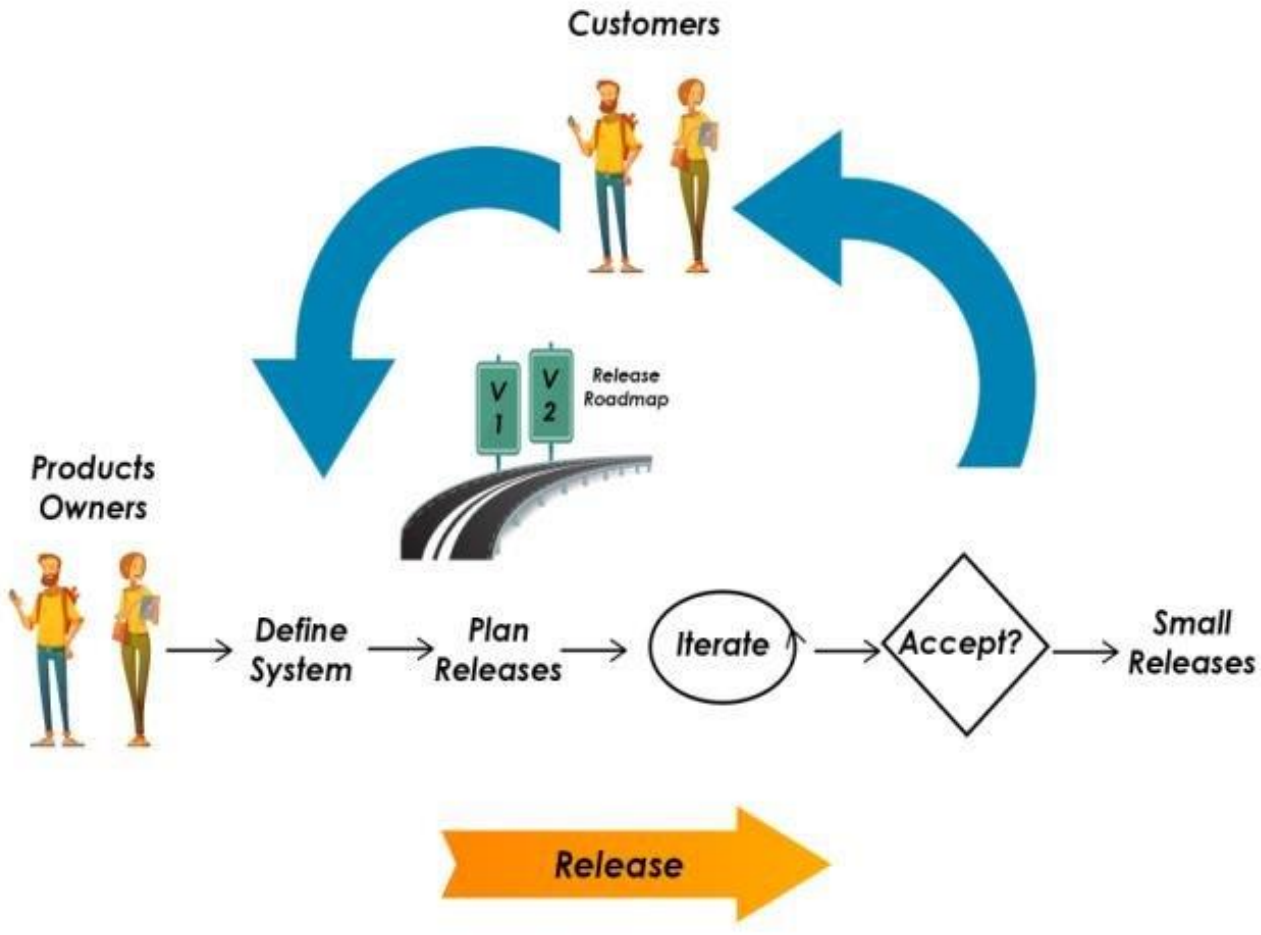

Figure 6: Example of Agile methodology with customer involvement [8]

6.2 Planning

Generally speaking, a project takes about 4 months minimum to be deemed as completed by its developers [9], and so, the plan was to devote almost 3 months for each of the systems to be developed. By doing so, there was increased predictability over the performance of the systems and how functional they are, and there is a feeling of balance, so that one system does not overpower the other. To decide upon the future of the project, the meeting notes acted as a sprint meeting, where the project's current goals, problems that arose during the development of those goals, what was actually achieved by the time of the meeting and where the project should go next were discussed every two weeks with the supervisor's input. This helped the project stay active until the next meeting, where it was decided that the systems should keep expanding as time constraints were no problem, as there was still enough time to develop new features before the agreed deadline.

	Meeting number	Academic week	Objectives for period	Objectives for next period
Term 1	1	2	1. Decide on my project idea 2. Figure out specification, objectives, and relevant information 3. Start thinking about possible libraries, frameworks and what languages will be used to develop my project	1. Have a project proposal along with an official title 2. Figure out the goals and aims of my project 3. Figure out what languages will be used to develop the project 4. Produce draft copies of the contract, ethics form and global checklist
	2	4	1. Have a project proposal along with an official title 2. Figure out the goals and aims of my project 3. Figure out what languages will be used to develop the project 4. Produce draft copies of the contract, ethics form and global checklist	1. Have a draft of my literature review for my next meeting in week 6 2. Produce a draft of the functional requirements of my project 3. Ensure the project contract, ethics form, global checklist are checked and ready to be submitted when the time comes
	3	6	1. Have a draft of my literature review for my next meeting in week 6 2. Produce a draft of the functional requirements of my project 3. Ensure the project contract, ethics form, global checklist are checked and ready to be submitted when the time comes 4. Create a database connection between Java, C# and SQL 5. Start implementing core functionalities of the Java app	1. Have a semi-complete draft of my literature review for my next meeting 2. Produce a semi-complete draft of the functional req of my project 3. Start working on design documentation
	4	8	1. Have a semi-complete draft of my literature review for my next meeting 2. Produce a semi-complete draft of the functional req of my project 3. Start working on design documentation 4. Apply some CSS to the Java app for the design documentation	1. Complete Literature Review / Functional Requirements Specifications 2. Further progress system design documentation 3. Make a start with the test plan and strategy
	5	10	1. Complete Literature Review / Functional Requirements Specifications 2. Further progress system design documentation 3. Make a start with the test plan and strategy	1. Finish all the documents for the submission of the first deliverable
Christmas break			1. All documents are finished, keep working on the Java software until the university opens after the Christmas break	1. Present the current Java application to my supervisor when we agree on a meeting date (after the January deliverable)
Term 2	6	18	1. Finish all the documents for the submission of the first deliverable	1. Keep working on the presentation layer of the C# system. Try to get the view side of things ready before the next meeting and then start implementing the core functionality of the customers.
	7	20	1. Keep working on the presentation layer of the C# system. Try to get the view side of things ready before the next meeting and then start implementing the core functionality of the customers.	1. Design login screen 2. Create the My Account page for the user to be able to edit their details 3. Functionality for the user to be able to delete their account. 4. Implement the core functionality of the C# system; the ability to book tickets for a movie
	8	22	1. Design login screen 2. Create the My Account page for the user to be able to edit their details 3. Functionality for the user to be able to delete their account. 4. Implement the core functionality of the C# system; the ability to book tickets for a movie	1. Implement a new feature where customers/guests will be able to book more than one seat 2. Allow customers to cancel a booking 3. Add a search filter on the table view in the Java app
	9	24	1. Implement a new feature where customers/guests will be able to book more than one seat 2. Allow customers to cancel a booking 3. Add a search filter on the table view in the Java app	1. Allow the admin/cinema owner to search for a particular date for a booking 2. Implement a new feature where each movie can have different number of rows of seats
	10	26	1. Allow the admin/cinema owner to search for a particular date for a booking 2. Implement a new feature where each movie can have different number of rows of seats	1. Start working on the final report document
Easter break			Both systems are finished, keep working on the final report document	
Submission deadline		31	The submission deadline is 06/05/2022 (Week 31). Submit final deliverable report	
Viva		32 - 33	Arrange the Viva. Meeting date TBA	

Figure 7: Final year project plan

6.3 Early stages

Before designing the UI for the desktop application, it was crucial that some of the main core features were implemented before applying styles through CSS, displaying images, trailers etc. A good-looking software is pointless without being able to perform as envisioned. Because the web-based system cannot function properly without the functionality of the desktop software (as there would be no data to pull from the movies' database table), the completion of the main features, such as adding/editing/removing movies from the system had to be implemented and confirmed to be working as expected, before even attempting to start working on the customer system. This being said, the Cinema Owner/Admin product was first structured in the shape of Forms, so that it would be easier to get the main functionality right first, and then add more elements and styles to them. Most notably, the add/remove movies and view their details, register/login etc. functionalities were the first ones to be coded, as they have a high priority in the functional requirements list (see Appendix A). Once some of the high priority features were confirmed to be working as expected, a design phase took place and the software started to take its final shape, and more information fields kept being added and changes were made to the layout/screens, as the functionality kept increasing. This was not as big of a concern for the web-based system, as most businesses nowadays offer online services for their customers, in which their UI design was inspirational. Influenced by such systems, and with a touch of personal preference and imagination, the web-based system was more straight-forward to finalize in terms of its UI design phase. In regard to its functionalities, features that are common to both systems, such as registering/logging in, editing account details, viewing a movie's information etc. was only a matter of matching certain controls to their web counterpart, and turning Java code into C# code, which was not a big problem as both languages have similar syntax.

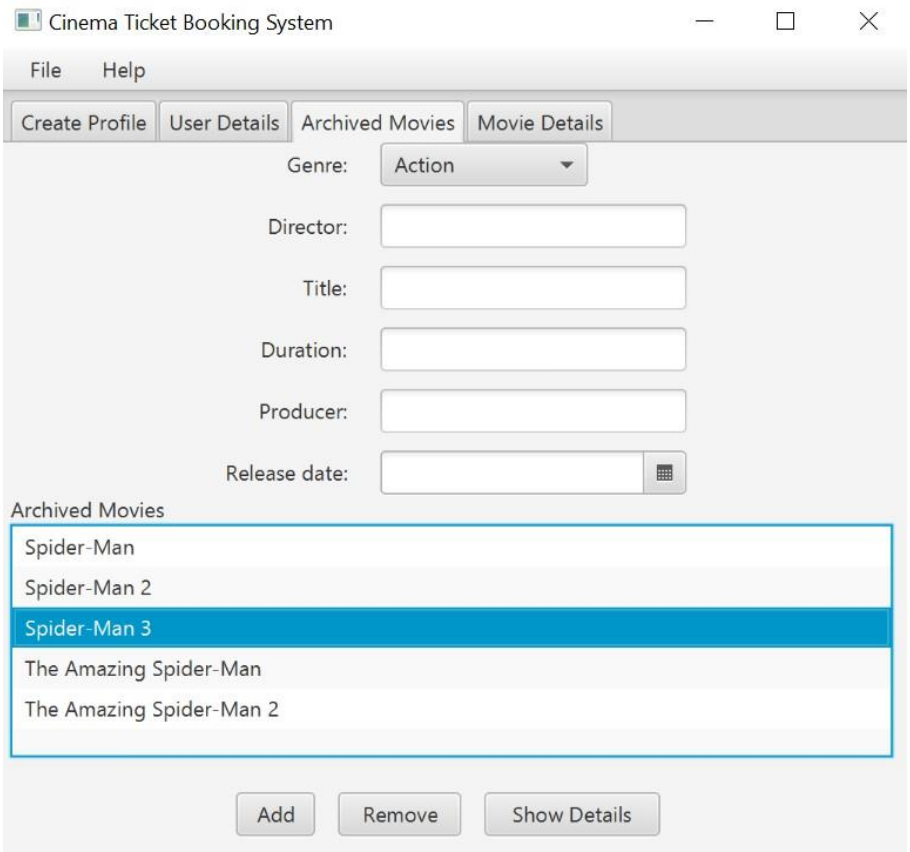

Figure 8: Archived movies list with controls prototype

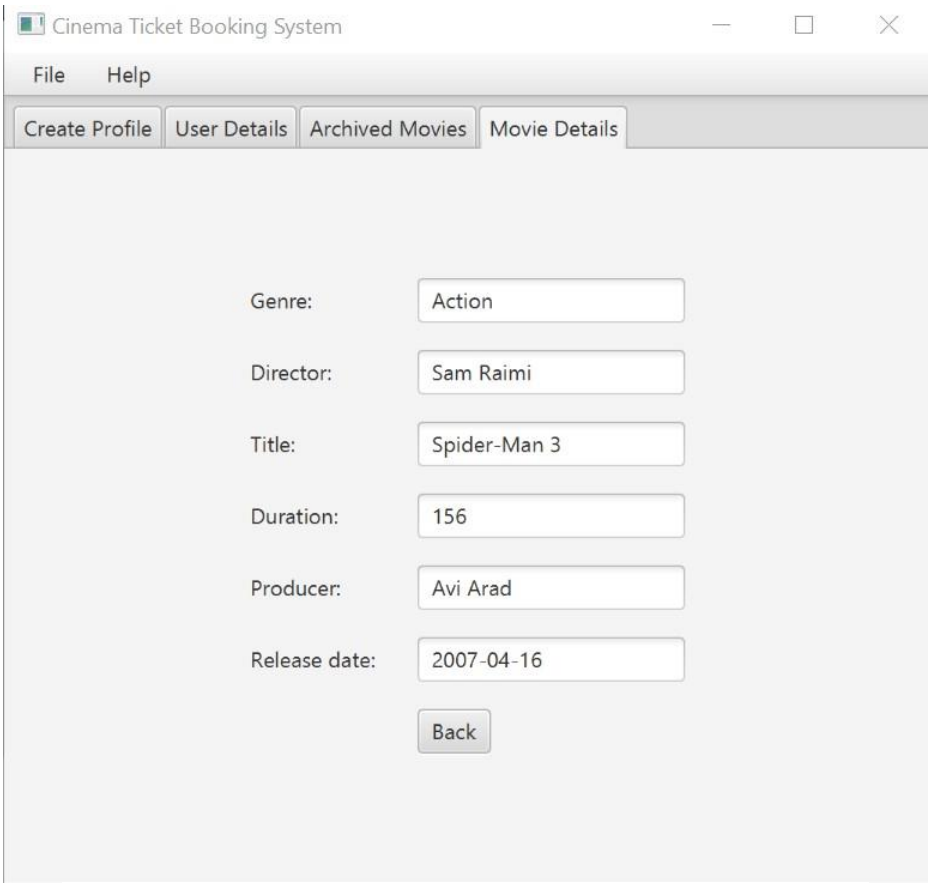

Figure 9: View movie details prototype

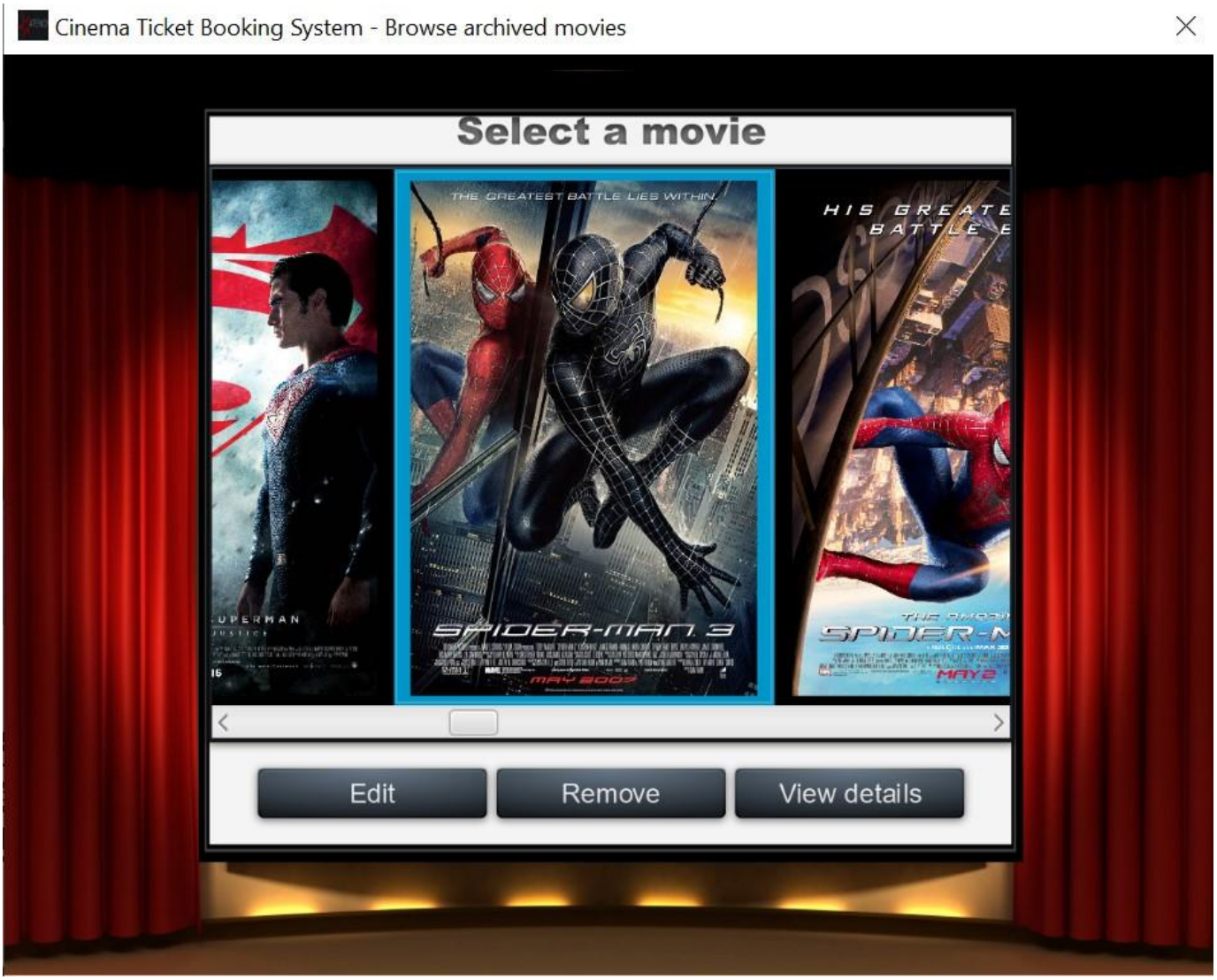

Figure 10: View archived movies final design

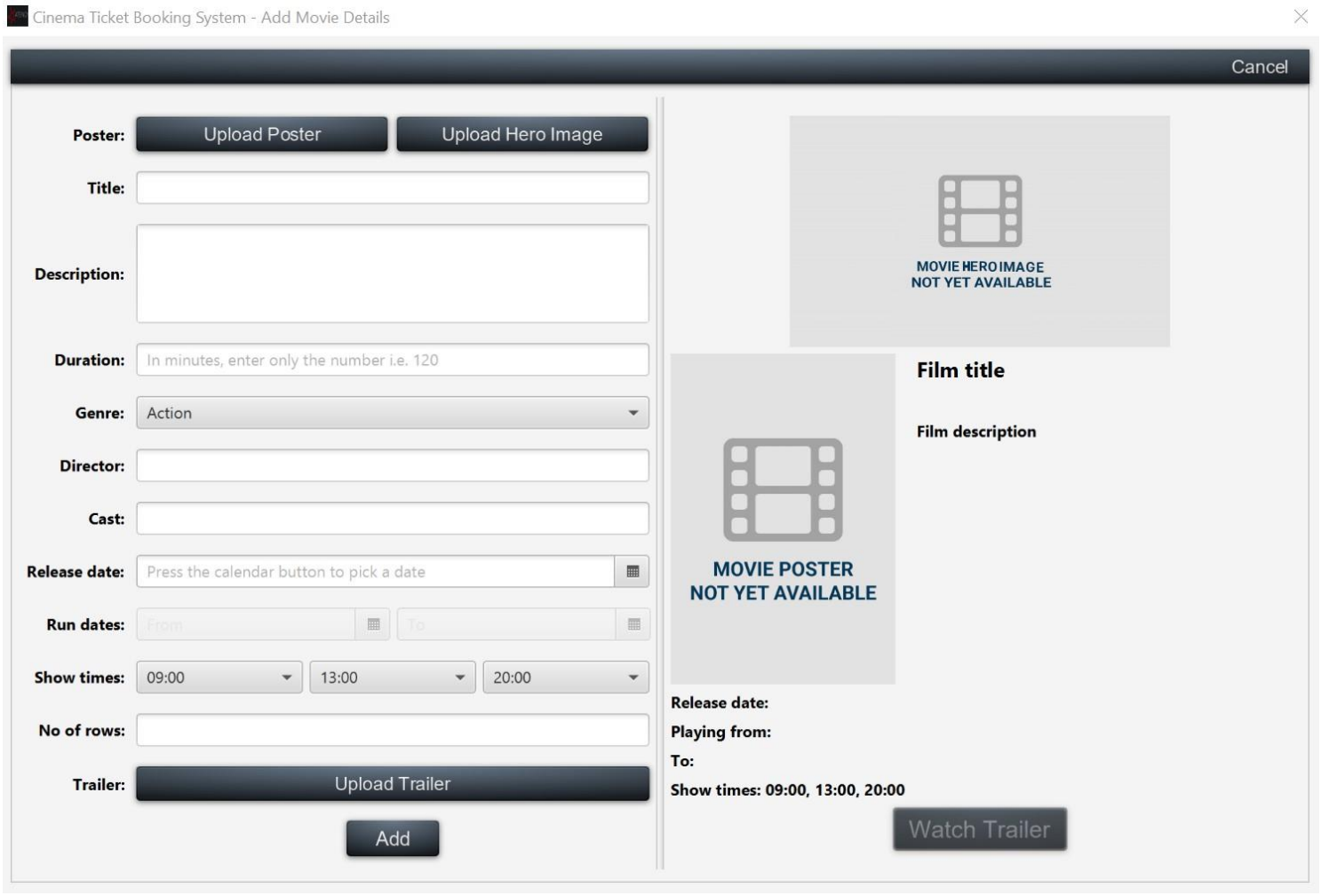

Figure 11: Add movie details final design

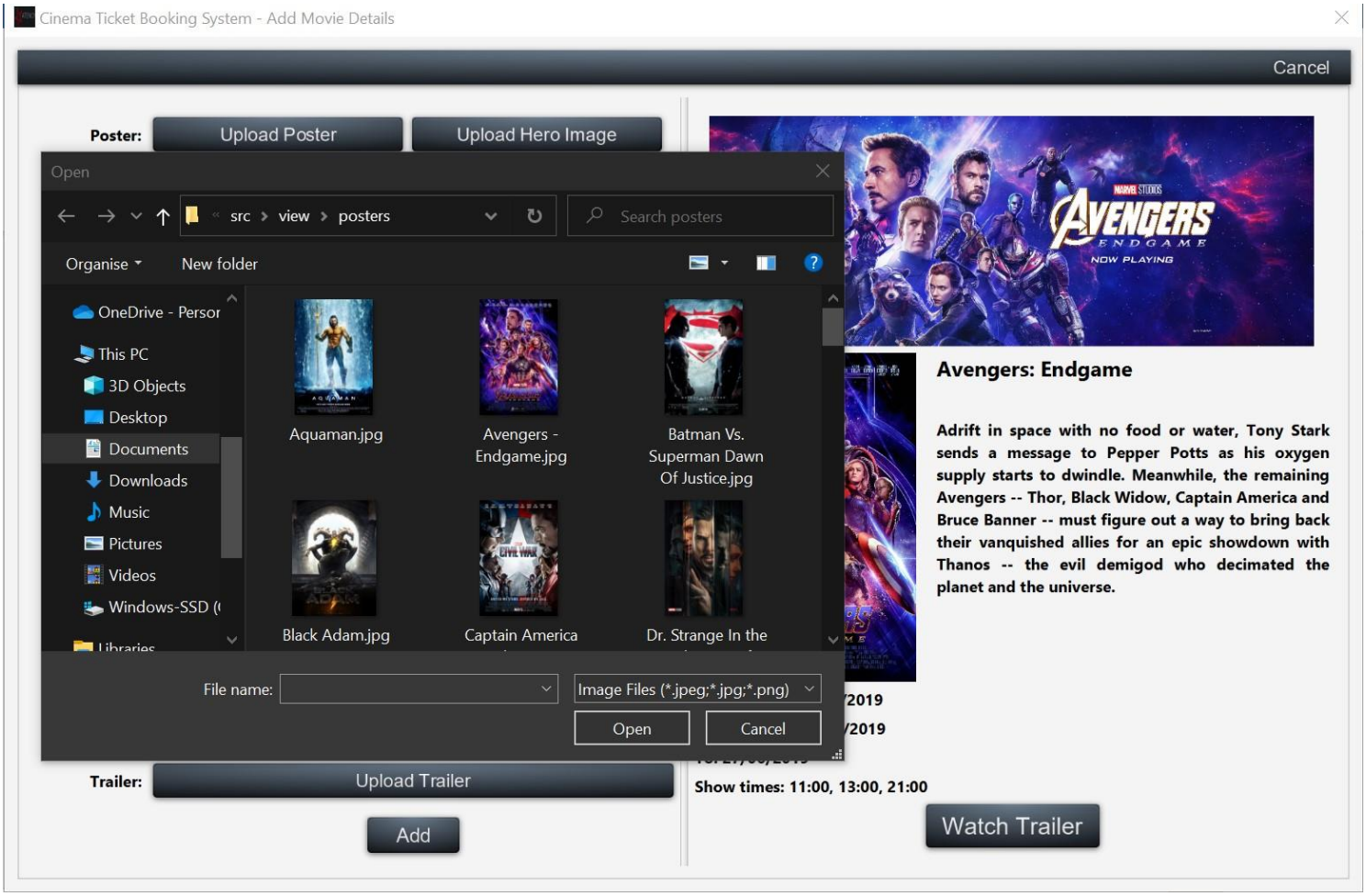

Figure 12: Add movie poster, hero image and trailer file navigation dialog

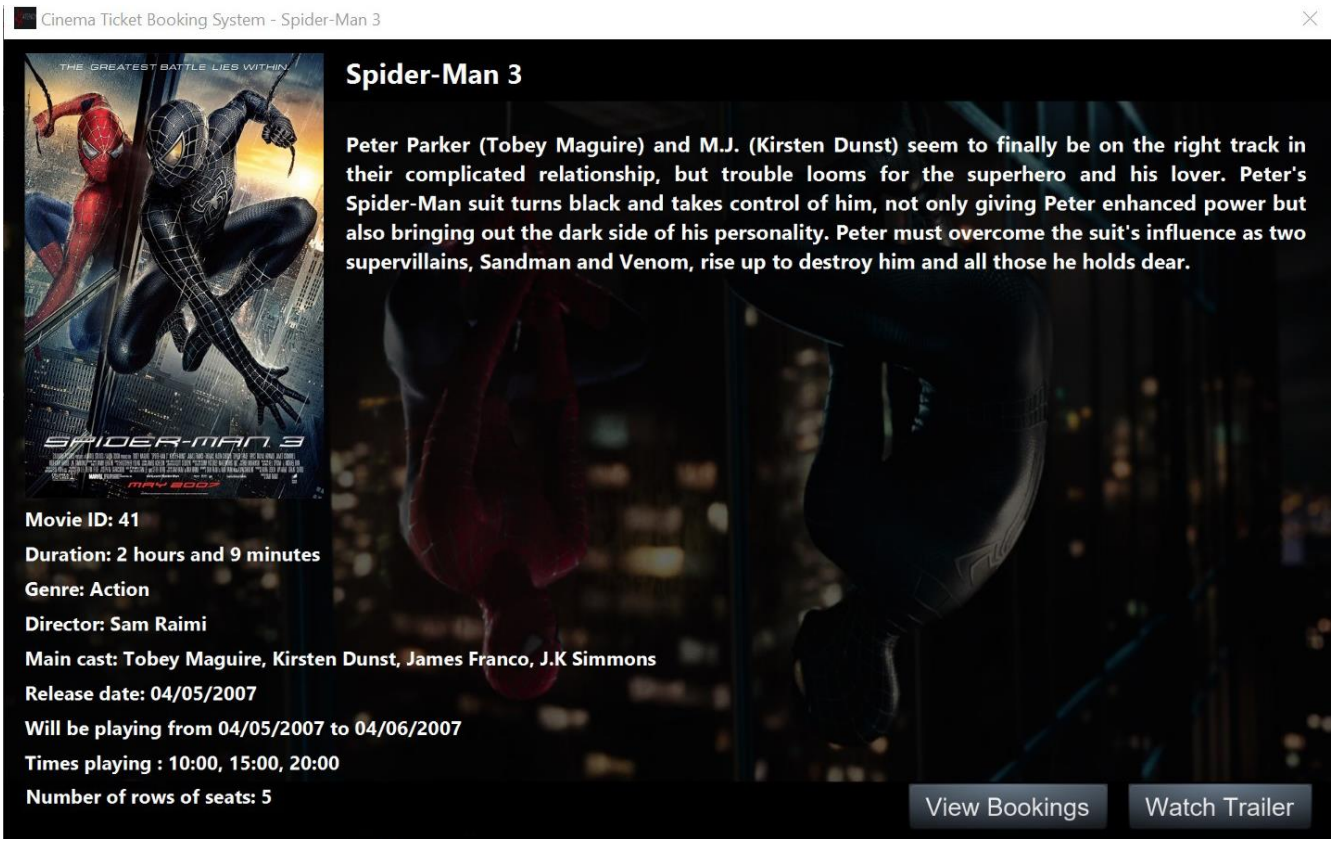

Figure 13: View movie details final design admin system

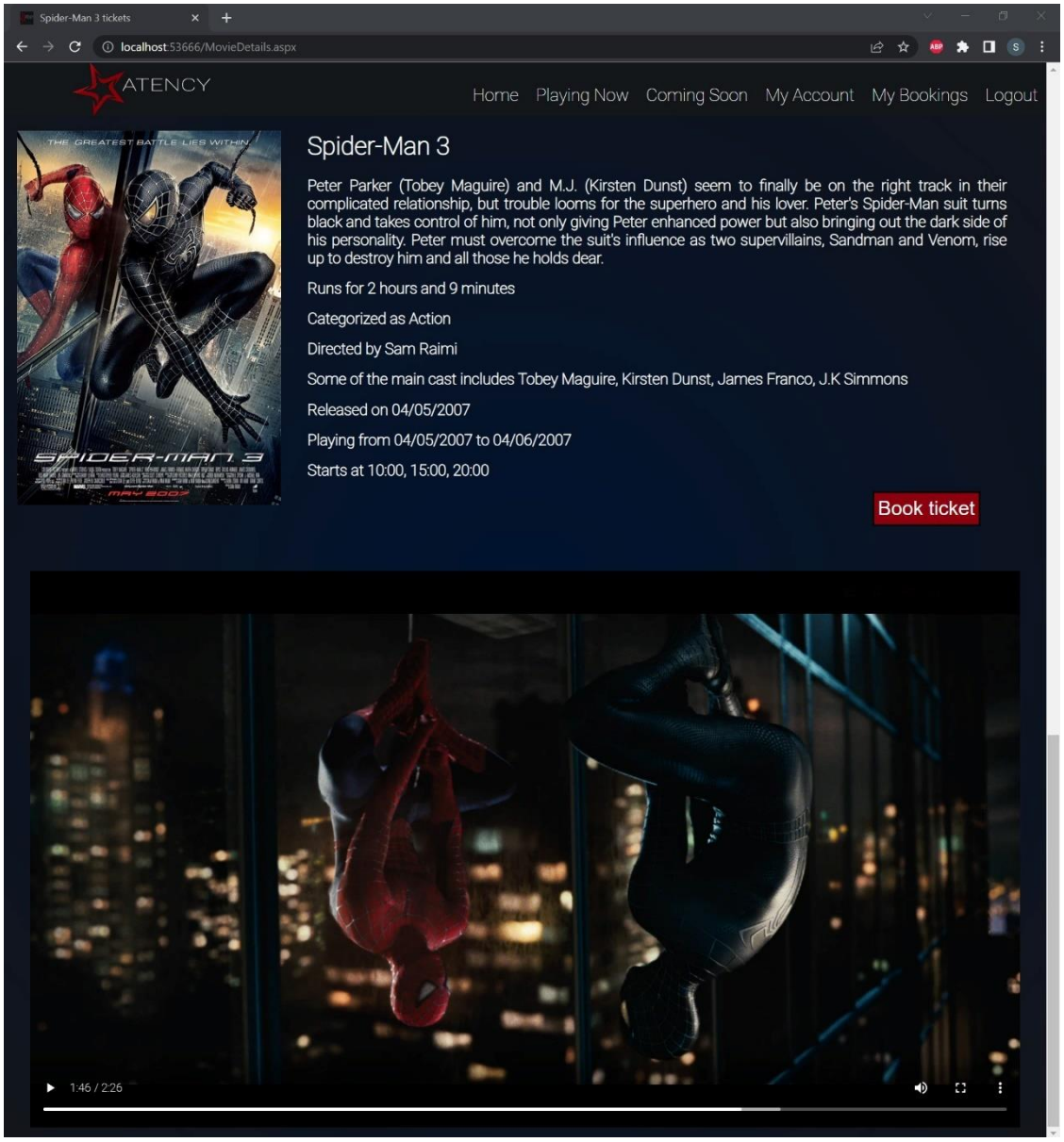

Figure 14: View movie details web system

6.4 Major problems encountered along the way and their solutions

The desktop software was less complex to implement, as experience in using the JavaFX library was already high enough to design it as envisioned. The most troublesome feature was the video player (trailer of the movie) because it was the first attempt at implementing such functionality. To create a video player, three JavaFX controls must be used together; a MediaView, MediaPlayer, and a Media. Together, they output a video in a JavaFX Stage (window). The MediaView control is essentially the container that displays the Media being played by a MediaPlayer [10]. The Media control holds the video's information such as its duration, metadata, video resolution and-so-on, of a source URI, and acts as a media resource [11]. The MediaPlayer provides the controls for playing, stopping, pausing, seeking etc. the media, but does not provide any visual elements, hence it must be used with the MediaView control [12]. The MediaView, combined with the Media and MediaPlayer objects, provide a visual representation of the video file, and with some buttons with appropriate event handlers, the functionality that the MediaPlayer provides, can be visibly interacted with. To successfully create a Media control, an absolute path to the video file (as a String) is used as a parameter to its constructor, and that Media object is then passed in as a constructor parameter to the MediaPlayer. Then, the MediaView is instantiated from the MediaPlayer object to play and show the video. Additionally, functionalities such as displaying the current time of the video, its total duration and being able to seek to a certain point are a necessity and a vital part of a video player. To create these, a Slider and two Label controls were used. The current time Label's text property is attached to the current time property of the MediaPlayer and so, there is a visual representation of the current time of the video. The total duration Label's text is simply the total duration of the MediaPlayer. Both Labels are formatted in minutes and seconds, as the durations are read as milliseconds, to make them more human-readable (see Appendix I). The Slider, however, has a change listener applied to its value property, which when dragged, the video seeks to the point that the Slider was dropped, and the current time Label is updated accordingly. Furthermore, a maximum point has to be acknowledged, and this is achieved by getting the total duration of the video in seconds. In contrast to Java 8 not providing an official API that does the heavy lifting to create a video player, HTML5 provides a very elegant, straight-forward, more advanced, and easy to use tag that embeds a video in a web page, with an optional controls attribute (see Appendix J).

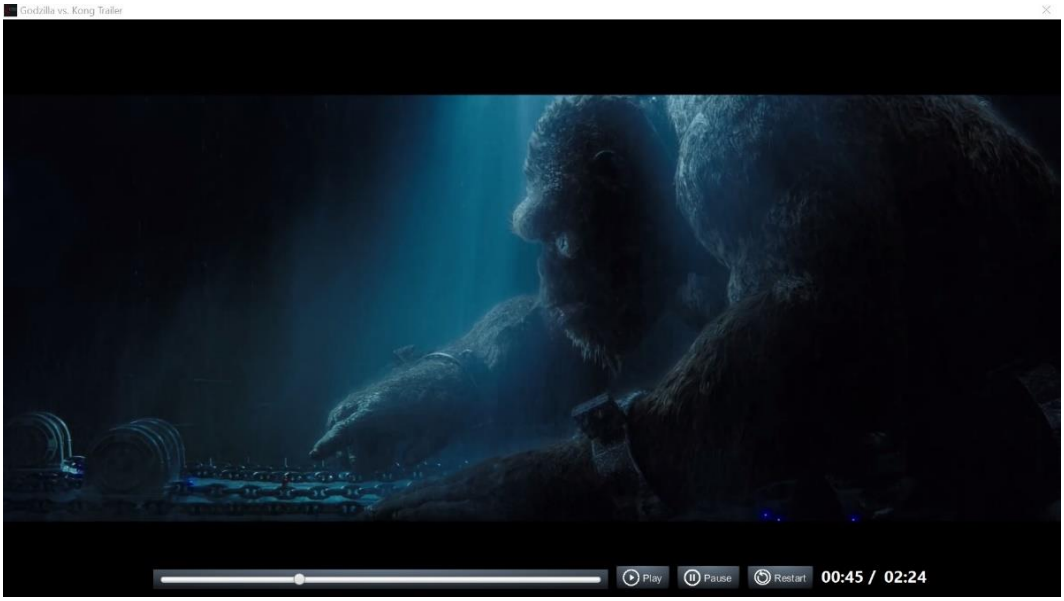

Figure 15: JavaFX video player with controls

Generally speaking, the desktop software was easier to implement and control, as it is an offline application, unlike the web-based system that deals with postbacks between servers. A postback occurs when the user submits information to a web page (such as login credentials, changes in a drop down list etc.) and then a response is returned to the user, refreshing the web page [13]. If not done correctly, the information that was entered might be lost, the user might not be able to change the selected item of a list, the web page might be reloaded before the user can finish their action, and more, and this results in a non-working, frustrating software. Coming from a non-web based background and being new to the field, managing postbacks was one of the challenges that had to be overcome in order to deliver a robust piece of working software, as the web pages offer a substantial amount of ImageButtons (buttons that are represented with an image, see Appendix D). By default, buttons automatically trigger a postback event as they are generally used to submit information, and so, certain controls had to be adjusted to not trigger such an event, where appropriate. This was achieved by using ASP.NET's UpdatePanel class, where sections of a page are partially rendered without a postback or refreshing the entire page [14]. For instance, when the user selects a seat, postback events need to be handled in order to prevent the page from reloading, and having the user be directed to the top of the page. If this was to happen, it would result in poor customer experience, and the likeliness of using this system frequently could be lower (see Table 12). Additionally, one of the most important features of the Cinema Ticket Booking System is to allow the user to reserve seats upon booking a ticket, so extra attention to detail was dedicated to this functionality. The most challenging part of the web-based system was to ensure that the booked seats ImageButtons for a particular movie, date, and time, are disabled and represented with the grey chair sprite, as per the seat legend. Each seat has a different ID, so they can be individually identified, in the format of the row's alphabetic character followed by the number of the seat in terms of its order in the row (see Appendix D).To achieve this, whenever the date or time is changed, a partial render occurs and updates the map of the seats according to the bookings database table seats column for that selected movie, date, and time. Firstly, all the bookings for a certain movie are stored in a variable and then all relevant information, such as the ID of the seat, the selected date and time, have to be checked one by one to see whether they match the information stored in the variable. If so, the ImageButton seat is disabled, the image is changed to the booked chair sprite to indicate its status, and the alternate text is set for screen readers (see Appendix K). Finally, another problem that was encountered was the fact that the available dates and times a movie would play had to be adjusted depending on the current date and time that the action took place. A booking cannot be made for a past date and time, and so each time a customer would be in the seat selection page, the computer's local date and time were used to determine which dates and times are available to be booked. The same logic applies when a customer would edit or cancel a booking, as ticket updates or cancellations cannot occur after the movie has played in the theatre (see Appendix L parts 1-2).

6.5 Data Validation

One key aspect of the Cinema Ticket Booking System is that data validation exists for both systems, to eliminate product misuse and reduce bugs so that the systems are used as intended. The most important aspect is that any information that is entered by any user is able to be stored in a database table accordingly to avoid data loss, as a maximum limit is set for all data types, most notably varchar/Strings; either explicitly or implicitly (see Appendix H). If that limit was to be exceeded, the whole software could crush or data loss would occur, resulting in an incomplete data entry. This being said, it is important to let the user know what they did wrong and how to fix the error they produced, in order to continue their action. For the desktop software, JavaFX Alerts are used to indicate that the submission of information is either accepted or not, with advice on how to proceed if an error indeed exists. This is achieved by first validating all inputs that are entered by the Cinema Owner/Admin, right before submitting the information to a database table (see Appendices M (parts 1-2) and N). Making sure that the information entered is in the correct format and data type, and most importantly, their length is lower than the specified maximum limit of the field's data type is crucial in order for the database to accept that information. To reduce code duplication, two private functions were created in order to enhance reusability; a function that creates an error log Alert, and one that indicates that the information was submitted successfully, to guide the user (see Appendix O). For the web-based system, HTML5 input controls have built-in attributes that do the heavy lifting and check if a field is mandatory or not, whether the correct format was entered for a field as per the *type* attribute, and a maximum/minimum length limit for the text box can be specified in the control's declaration. To assist the user in fixing the errors they produced, an error message pops up that gives clear advice on how to proceed. Aided by CSS, each text box has a red/green left border to indicate form validity. Colour representation can assist the user in spotting mistakes earlier or avoiding them, and provide a more pleasant customer experience, as red is mostly used to demonstrate that something went wrong, and green illustrates acceptance and correctness [15]. Logic errors also need to be addressed, even if the information entered is syntactically correct. Errors such as duplicate usernames, login errors due to the username/password not being present in the system or matching etc. need to be handled via the code-behind, as a pre-built error message alert cannot be produced for them. Those types of errors are shown in the form of a label, coloured in red with an appropriate message (see Appendix P).

6.6 Email functionalities

Just like any other booking web-system, the Cinema Ticket Booking System has the ability to send an email confirmation, cancellation, or ticket update, depending on the user's course of action. Upon booking a ticket and reserving seat(s), the user is asked to provide an email address, so that the ticket information can be available to both parties. Customers/Guests will need proof of their booking so that the Cinema Owners/Admins are able to confirm their reservation and be allowed to enter the theatre to watch a movie, and having the confirmation email with the ticket information at hand is essential for a smooth experience. However, if that email was to be lost, the staff members are capable of searching for a booking in the desktop software, by asking the customer personal questions, such as the ticket holder name, the ticket ID

(if not forgotten), or the address that the email was sent to. By registering, Customers are able to cancel or change their preferred date, time, and seat and receive a cancellation/ticket update email, unlike Guests, who can only receive a confirmation email and are only able to cancel their booking via the staff members without receiving an email. A business Gmail account was set up for this purpose, and was configured to be able to be accessed by the web-based software, so that the content of the email can be automated (see Appendix Q).

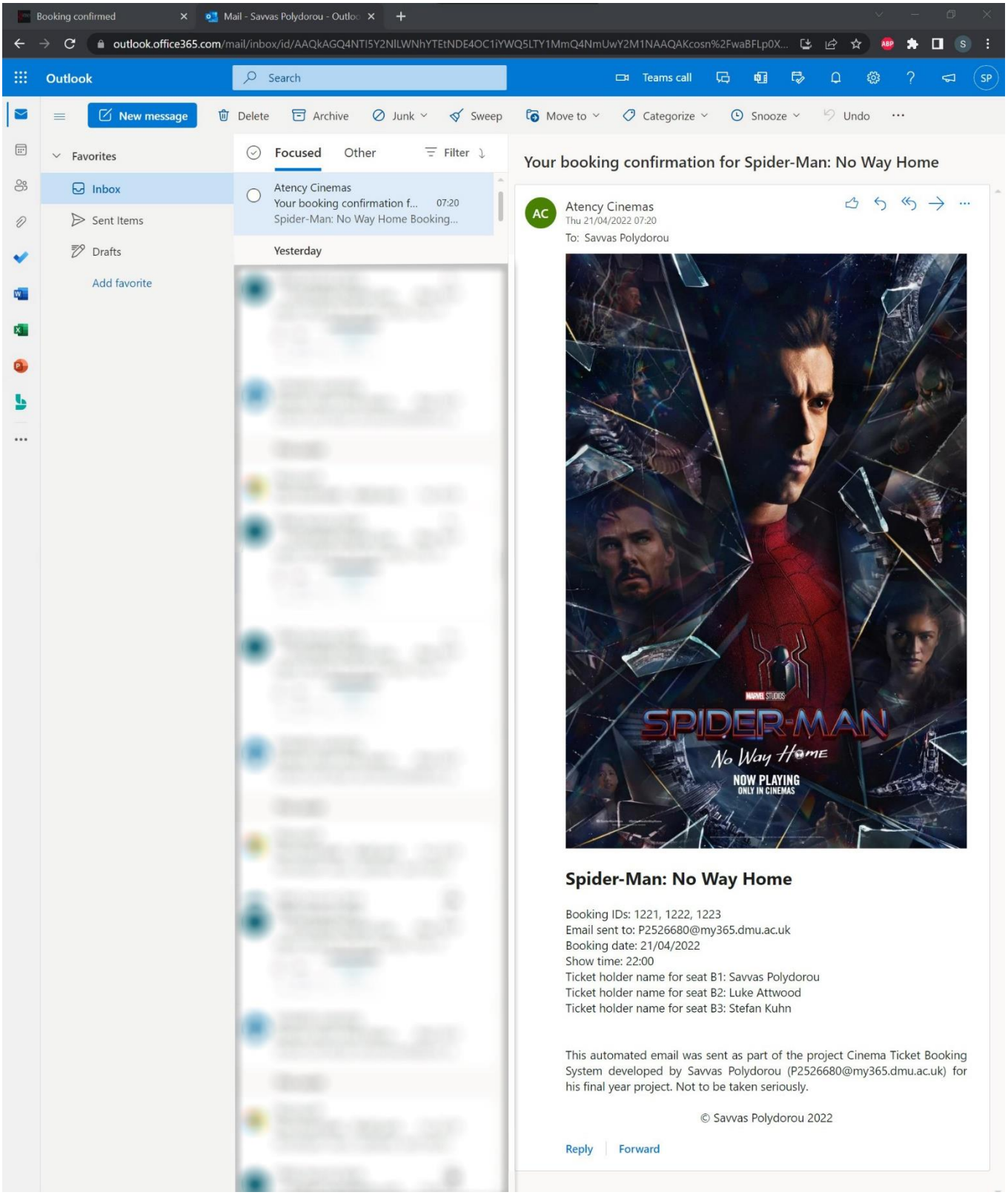

Figure 16: Automated email confirmation with booking information

7. Testing

7.1 Methodology and plans

As noted above, data validation is a key feature of both systems, and so the Unit testing process was applied on both systems to test individual components, such as model data and database functionalities. Dummy data were used for testing purposes and were passed as parameters to the data validation methods of each class that implements such functionality, each respecting the specified minimum and maximum boundaries, data type and formatting of the component being tested (see Appendix M for the movie data validation function). Database functionalities such as registering users and removing them, adding, and removing movies etc. were also considered, to test if data can be added to their respective database table, after validating their data (see Appendix B). An example of how the data was tested can be found below and refer to Appendix R for the component's test code. The rest of the test cases are specified in Appendix S (parts 1-7).

Description of item to be tested:

Full name testing for Cinema Owners/Admins/Customers. The full name validation can be found when the user registers or wants to edit their details, and when booking a ticket **(UC001, UC003, UC004, UC051, UC101, UC104, UC105) (see Appendix A). Field cannot be empty.** Full name must be between 1 and 50 characters. Data type accepted is varchar(50)/String.

Table 11: Test data for full name/ticket holder name

Test Type	Test Data	Expected Result
Min (Boundary)	1 character S	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	<<*empty*>> Anything over 50 characters	Fail

7.2 User-based testing

In order to truly evaluate the performance, usability, look and accessibility of the systems, a user-based trial was conducted in person, consisting of 6 test subjects. User-based testing includes targeting users to use and evaluate a system, without having previous experiences with the software [16]. By being the developer of the project, it is easy to understand and remember how certain features work without giving too much thought on how to achieve or find them, and so, by observing and receiving constructive feedback from the test subjects, some slight modifications were made in due course, such as the seat legend and colour representation of the form's validity that was mentioned in subsection 6.5. For the desktop application, an introduction and demonstration was given in order to get them to know the software

better, because if this application was to be used by a real cinema company, the staff members would receive training first. However, for the web-based system, nothing was revealed or discussed, as the targeted audience are people who have never used this system and have no experience using it, and in a real-life scenario, they would not receive any formal guidance or assistance before using it. It is the system's responsibility to be able to be navigable, usable, and not confusing at first glance as the customers of a company are highly unlikely to receive any supporting documentation or training to use the system. No usability requirements were specified, as time constraints fall outside the scope of this project, hence the worst-planned-best case scenario for achieving tasks was not considered. Finally, a close-ended de-brief was held and a questionnaire was given at the end in order for the tests subjects to express their opinions about the two systems [17].

Table 12: Overall system usability scale [17]

Overall System Usability Scale		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1	I think that I would like to use this system frequently.				✔	
2	I found the system unnecessarily complex.	✔				
3	I thought the system was easy to use.					✔
4	I think that I would need the support of a technical person to be able to use this system.		✔			
5	I found the various functions in this system were well integrated					✔
6	I thought there was too much inconsistency in this system		✔			
7	I would imagine that most people would learn to use this system very quickly.				✔	
8	I found the system very cumbersome to use.		✔			
9	I felt very confident using the system.					✔
10	I needed to learn a lot of things before I could get going with this system.			✔		

8. Critical evaluation

8.1 System evaluation

The Cinema Ticket Booking System far exceeded the initial plans and expectations, as at first, only 24 use cases were documented, compared to the final number of functionalities; 48 (see Appendix A). A substantial amount of work has gone into this project, and all objectives, aims and goals are met, before the agreed deadline. To highlight this, the desktop application contains 28 Java classes and 5 CSS files distributed across the 3 MVC and unit testing packages, and the web-based system has 7 CS classes, 17 ASPX, 17 ASPX.CS, 18 CSS, 2 HTML and 1 text file placed in the middle and presentation layers and testing folder, making a total of 95 files. The text file contains the ID of the logged in customer using the system in order to continue being logged in after exiting the system. Since sessions end after exiting the system, it is essential to re-assign the ID of the customer by reading the text file. All functional requirements are implemented and work as expected, resulting in a robust, fully-working system that can be used by a cinema business. The aspect I am most proud of is the attention to detail, most notably the data validation checks that are represented with alert messages, the use of local date and time to disable future date of births, the prevention of booking a ticket on a date and time that has passed and the allowance of editing or cancelling a booking only if the movie has not played in the theatre yet (for registered and logged in customers only). Furthermore, I consider the process of distributing seats and demonstrating which are available, selected or booked, one of the best features of the system visually, as a customer can be guided by the seat legend and cannot easily find the seat selection process confusing, and it is one of the core features any cinema web-system ought to offer. If given more time, I would like to implement password encryption so that the accounts are safe and invulnerable to hacking, and use an external API that offers real-life payment methods and include ticket prices with discounts. Underaged children, students and how many people are booking tickets at once would each be offered a different type of discount. Moreover, I would like to upgrade the project to resemble a seat selection system as well. Automated seat recommendations for a group of people would be a great addition to the system, as the best possible row and seats would be suggested depending on the availability of the seats as close to middle of the row as practicable, making the viewing experience more pleasant. Additionally, I would like to make the systems responsive for different screen sizes with the use of CSS media queries, as currently they are only available for bigger screens. More importantly, the web-based system would have priority and be developed for multiple screen sizes, due to the fact that nowadays internet traffic on mobile phones is higher than the traffic on desktop [18]. Finally, I would like to explore the idea of having multiple cinema locations with more screens. Each location would have different Cinema Owners/Admins, and each screen could have a different layout. This last part is included in the current system, as when adding/editing a movie, a number of rows can be set, mimicking the functionality of having different screens.

8.2 Approach evaluation

The agile methodology approach that was used was the most ideal, as the two systems needed to be implemented simultaneously at one point. After the main functionalities of the desktop software were implemented, such as adding, editing, and removing movies from the database, the development of the web-based system took place and once the booking functionalities were implemented, the desktop software was revisited to implement the view/management of bookings for a movie. The systems complement each other and a lot of revisiting of both systems would happen to keep increasing their functionalities and make the overall communication between them better. Additionally, because test code is written at the same time as the development phase, it was fairly faster to ensure that a function works properly, without producing an error and potentially crashing the system if not handled properly. This saved a lot of time because if the testing stage did not take place during the development, certain functionalities would need to be tested during using the software, which would cost a lot of development time. Moreover, the meetings that were held every two weeks with the supervisor to showcase the new features and update the product backlog were associated with Agile's sprint planning meetings, which formed the life cycle of the methodology. In terms of the development approach, the MVC architecture was the most suitable pattern to develop the desktop application, due to the fact that a lot of code would be repetitive to achieve the same results. Classes and code are reusable and the view package contains methods in which the controls that are available change via the controller, resulting in a set of reusable screens that have different controls and content, without the need of implementing a new class. If this project was to be undertaken again, the Three Tier Architecture that the web-based system was designed with would be replaced with the MVC pattern as well, as it became apparent that some pages needed to be implemented again to essentially be visually identical, but offer different controls.

8.3 Tools used evaluation

The desktop application was developed using Java and its JavaFX library, written in Eclipse IDE. It was the preferred IDE of choice due to previous positive interactions, but it proved to lack an SQL server explorer to create and manage the database, and so, the Microsoft SQL Server Management Studio application was used to create the database and its tables, view the data that were inserted by the two systems, and backup the database to a local drive. Additionally, Java and the JavaFX library were also the preferred tools to build the desktop application due to past experience and prior knowledge. JavaFX is the current standard toolkit for building interactive systems and receives more updates than previous frameworks, and is also very compatible with the MVC architecture, which was the design pattern that the application was built with [19]. Database management was not a problem for the web-based system, as it was developed using C# and the ASP.NET framework, written in Microsoft's Visual Studio to be able to run the web sites via a web browser. Visual Studio offers the ability to manage a database through the IDE itself (SQL Server Object Explorer), eliminating the essential use of the management studio application. C# was chosen after deciding the language/framework for the desktop application, and due to their syntax similarity and the fact that both languages are object oriented, it proved to be the most suitable language to build the web-based system. The framework, ASP.NET, is widely known

and a very detailed API is provided with great documentation and thorough examples, which led to the decision of developing the system using it. As noted before, knowledge of web development was minimal in comparison to desktop development, and the combination of C# and ASP.NET provided a more familiar feeling, as opposed to using languages geared towards web development only, such as PHP or JavaScript, which have completely different syntax, logic, and frameworks. However, JavaScript could aid the project in certain areas as it could be used to produce custom error messages or alter the pre-existing ones (see Appendix P), as the pre-built messages could not be accessed and modified through C#, and a Label control needed to be added to the page to represent the error message, which was not very ideal in terms of visual representation and computer-human interaction.

9. Conclusion

In conclusion, the Cinema Ticket Booking System aims to offer convenience to its users, by providing business management and ticket booking facilities digitally, with the use of software and online services. The project was developed to its entirety, as all of the functional requirements are implemented and work as expected, all objectives mentioned in subsection 2.3 are met, and the final product is delivered before the agreed deadline. Unit testing to ensure that functions behave as envisioned was considered and a user-based testing trial was conducted and a questionnaire was given to assess the robustness of the project, in which the two systems were used by the untrained eye to provide constructive feedback. The main academic objectives of this project, which were to develop the system with a focus on maintainability and apply interaction design principles are achieved, as it is clear that the system would be used more frequently by the targeted audience and it provides a pleasant customer experience.

10. References

[1] Virtru (2021) *8 Benefits of Digital Transformation* [online] Virtru. Available from: https://www.virtru.com/blog/8-benefits-digital-transformation/ [Accessed 16/11/2021].

[2] Anonymous (2016) *How do you buy a movie ticket at a theater?* [online] Quora. Available from: https://www.quora.com/How-do-you-buy-a-movie-ticket-at-a-theater/answers/24039587 [Accessed 14/01/2022].

[3] Mrbiff (n.d.) *How to Go to the Movies.* [online] Instructables Living. Available from: https://www.instructables.com/How-to-go-to-the-movies [Accessed 14/01/2022].

[4] Deloitte (2020) *Milliseconds make Millions* [online] Deloitte. Available from: https://www2.deloitte.com/content/dam/Deloitte/ie/Documents/Consulting/Milliseconds_Make_Millions_report.pdf [Accessed 26/04/2022].

[5] Odeon (2022) *Can I cancel, refund or change my booking?* [online] Odeon. Available from: https://help.odeon.co.uk/hc/en-gb/articles/360021841340-Can-I-cancel-refund-or-change-my-booking- [Accessed 1/05/2022].

[6] Insightsoftware (2021) *5 Benefits of a 3-Tier Architecture*. [online] Insightsoftware. Available from: https://insightsoftware.com/blog/5-benefits-of-a-3-tier-architecture/ [Accessed 1/05/2022].

[7] Webucator (n.d.) *How to Check Case-Sensitivity in SQL Server | Webucator*. [online] Webucator. Available from: https://www.webucator.com/article/how-to-check-case-sensitivity-in-sql-server [Accessed 27/04/2022].

[8] Lalband, Neelu & Kavitha, D (2020). *Software Development Technique for the Betterment of End User Satisfaction using Agile Methodology.* TEM Journal. 9. 992-1002. 10.18421/TEM93-22.

[9] Wardynski, D. (2017) *How Long Does it Take to Build Custom Software for a Business?* [online] Brainspire.com Available from: https://www.brainspire.com/blog/how-long-does-it-take-to-build-custom-software-for-a-business [Accessed 2/05/2022].

[10] Oracle (n.d.) *MediaView (JavaFX 8)* [online] Docs.oracle.com. Available from: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/MediaView.html [Accessed 17/04/2022].

[11] Oracle (n.d.) *Media (JavaFX 8)* [online] Docs.oracle.com. Available from: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/Media.html [Accessed 17/04/2022].

[12] Oracle (n.d.) *MediaPlayer (JavaFX 8)* [online] Docs.oracle.com. Available from: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/MediaPlayer.html [Accessed 17/04/2022].

[13] Raja, P. (2018) *What is PostBack in ASP.NET*. [online] C-sharpcorner.com. Available from: https://www.c-sharpcorner.com/uploadfile/2f73dd/what-is-postback-in-Asp-Net/ [Accessed 18/04/2022].

[14] Microsoft (n.d.) *UpdatePanel Class (System.Web.UI)*. [online] Available from: https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.updatepanel?view=netframework-4.8 [Accessed 18/04/2022].

[15] Dulenko, V. (2017) *How to Write a Perfect Error Message*. [online] Medium. Available from: https://uxplanet.org/how-to-write-a-perfect-error-message-da1ca65a8f36 [Accessed 19/04/2022].

[16] Roose, J. (n.d.) *How to Conduct Usability Testing in Six Steps*. [online] Toptal Design Blog. Available from: https://www.toptal.com/designers/ux-consultants/how-to-conduct-usability-testing-in-6-steps [Accessed 21/04/2022].

[17] Klug, B. (2017) *An Overview of the System Usability Scale in Library Website and System Usability Testing*. *Weave: Journal of Library User Experience*, 1(6).

[18] Enge, E. (2021) *Mobile vs. Desktop Usage in 2020*. [online] Perficient.com. Available from: https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage [Accessed 24/04/2022].

[19] Larkin, P. (2021) *JavaFX vs Swing: The Key Differences*. [online] Career Karma. Available from: https://careerkarma.com/blog/javafx-vs-java-swing/ [Accessed 29/04/2022].

11. Appendices

Appendix A: List of functional requirements

Table 13: Functional requirements

Users		Use Case Number and Name	Priority
1	Cinema Owner	UC001 – Add admins UC002 – Remove admins UC003 – Edit account details UC004 – Register UC005 – Login UC006 – Reset password UC007 – View movie details UC008 – Browse through movies UC009 – Search for a movie UC010 – View all bookings for a movie UC011 – Search for a specific booking UC012 – Watch a movie's trailer UC013 – Logout	UC001 – High UC002 – High UC003 – Low UC004 – High UC005 – High UC006 – Medium UC007 – High UC008 – High UC009 – Medium UC010 – High UC011 – High UC012 – Medium UC013 – Medium
2	Admin	UC051 – Edit account details UC052 – Add movie to system UC053 – Remove movie from system UC054 – Edit movie details UC055 – Valid movie data UC056 – Browse through movies UC057 – Login UC058 – Reset password UC059 – View movie details UC060 – Search for a movie UC061 – View all bookings for a movie UC062 – Search for a specific booking UC063 – Watch a movie's trailer UC064 – Logout UC065 – Delete a booking UC066 – Delete all bookings for a movie	UC051 – Low UC052 – High UC053 – High UC054 – High UC055 – High UC056 – High UC057 – High UC058 – Medium UC059 – High UC060 – Medium UC061 – High UC062 – High UC063 – Medium UC064 – Medium UC065 – High UC066 – High
3	Customer	UC101 – Register UC102 – Login UC103 – Find movie based on status UC104 – Book movie ticket UC105 – Edit account details UC106 – Delete account UC107 – Reserve seat(s) UC108 – Recover password UC109 – View movie details UC110 – Watch a movie's trailer UC111 – Logout UC112 – View account's bookings UC113 – Update booking date, time, or seat UC114 – Cancel booking	UC101 – High UC102 – High UC103 – High UC104 – High UC105 – Medium UC106 – Low UC107 – High UC108 – Medium UC109 – High UC110 – Medium UC111 – Medium UC112 – Medium UC113 – Medium UC114 – High
4	Guest	UC151 – Find movie based on status UC152 – Book movie ticket UC153 – View movie details UC154 – Watch a movie's trailer UC155 – Reserve seat(s)	UC151 – High UC152 – High UC153 – High UC154 – Medium UC155 – High

Appendix B: Database test code in Java

```
@Test
public void MovieAddAndRemoveFromDatabaseSuccessful()
{
	//THE CURRENT ADDED RECORD WILL BE DELETED FROM THE DATABASE
	movie = new Movie();
	String error = "";
	//set test data to add the movie to the database
	movie.setTitle(movieTitle);
	movie.setDescription(movieDescription);
	movie.setDuration(136);
	movie.setGenre(movieGenre);
	movie.setDirector(movieDirector);
	movie.setCast(movieCast);
	movie.setReleaseDate(movieReleaseDate);
	movie.setPlayingFromDate(moviePlayingFromDate);
	movie.setPlayingToDate(moviePlayingToDate);
	String[] split = movieShowTimes.split(", ");
	for(int i = 0; i < split.length; i++)
		movie.addShowTime(split[i]);
	movie.setTrailerPath(movieTrailerPath);
	movie.setPosterPath(moviePosterPath);
	movie.setStatus();
	movie.setHeroImagePath(movieHeroImagePath);
	movie.setNumberOfRows(5);
	//check for data validation
	error = movie.dataValidation(movie.getTitle(), movie.getDescription(),
			Integer.toString(movie.getDuration()), movie.getDirector(),
			movie.getCast(), movie.getReleaseDate(),
			movie.getPlayingFromDate(), movie.getPlayingToDate(),
			movie.getTrailerPath(), movie.getPosterPath(),
			movie.getHeroImagePath(),
			Integer.toString(movie.getNumberOfRows()));
	//if no error message is returned
	if(error.equals(""))
	{
		try {
			movie.setID(database.addMovie(movie));
			/*delete the current added record so the database is not full
			  of random test data. If the error message is still an empty
			  string then this means the movie was added and deleted*/
			database.deleteMovie(movie);
		} catch (SQLException e) {
			error = "Error adding and deleting movie";
		}
	}
	assertEquals(error, "");
}
```

Figure 17: Database test code in Java

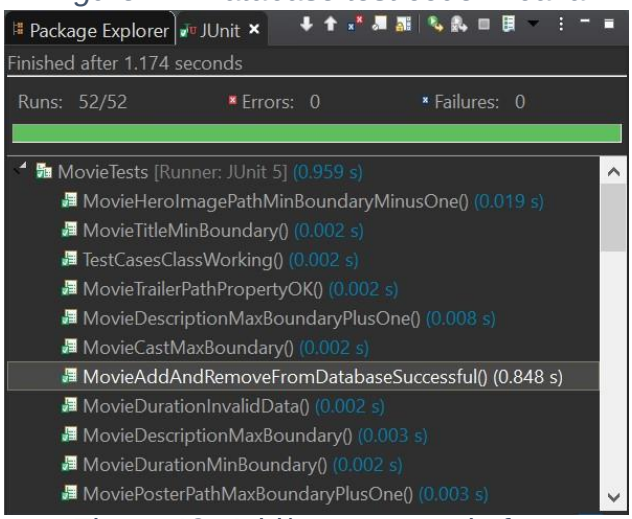

Figure 18: Add/remove movie from database test result

Appendix C: Use case descriptions part 1

Table 14: Use cases for editing account details for Cinema Owners/Admins

UC003, UC051	
Title	Edit account details (Cinema Owner/Admin)
Description	Cinema Owners/Admins are able to edit their account details
Primary pathway	1. User enters the system 2. User is presented with a menu where the edit button exists below their main details and clicks on it 3. User proceeds to edit their account details
Exception pathway	1. User authentication failed due to incorrect credentials 2. User fails to submit new information due to invalid data formatting; the User is presented with a set of instructions

Table 15: Use case for registering a Cinema Owner account

UC004	
Title	Register (Cinema Owner)
Description	Cinema Owner can register an account in the system
Primary pathway	1. User launches the application 2. User is presented with a login form and presses the register button 3. User proceeds to register an account
Exception pathway	1. User fails to create an account due to invalid data formatting or the username is taken; the User is presented with a set of instructions

Table 16: Use cases for logging in Cinema Owners/Admins

UC005, UC057	
Title	Login (Cinema Owner/Admin)
Description	Cinema Owners/Admins can login the system
Primary pathway	1. User launches the application 2. User is presented fills the login form and presses the login button 3. User is successfully authenticated
Exception pathway	1. User authentication failed due to incorrect credentials

Table 17: Use cases for resetting Cinema Owners/Admins password

UC006, UC058	
Title	Reset password (Cinema Owner/Admin)
Description	Cinema Owners/Admins can reset their password
Primary pathway	1. User launches the application 2. User is presented with a login form and presses the forgot your password hyperlink 3. User enters username and secret keyword 4. User proceeds to reset their password
Exception pathway	1. User enters incorrect username and secret keyword combination 2. User enters invalid data formatting for password

Appendix C: Use case descriptions part 2

Table 18: Use cases for browsing movies, viewing details Cinema Owner/Admin

UC007, UC008, UC056, UC059	
Title	Browse through movies/View movie details (Cinema Owner/Admin)
Description	Cinema Owners/Admins can browse through movies and view their details
Primary pathway	1. User enters the system 2. User is presented with a menu where the browse <*movie status*> movies buttons exists and clicks on one 3. User is presented with a list of movies 4. User clicks on a movie and then the view details button
Exception pathway	1. User authentication failed due to incorrect credentials 2. User does not select a movie and proceeds to view their details, an alert pops up

Table 19: Use cases for searching for a movie for Cinema Owners/Admins

UC009, UC060	
Title	Search for a movie (Cinema Owner/Admin)
Description	Cinema Owners/Admins can search for a movie and view their details
Primary pathway	1. User enters the system 2. User is presented with a menu where the search for a movie button exists and clicks on it 3. User is presented with a dropdown list of movies
Exception pathway	1. User authentication failed due to incorrect credentials

Table 20: Use cases for viewing/searching all the bookings for a movie

UC010, UC011, UC061,UC062	
Title	View/search all bookings for a movie (Cinema Owner/Admin)
Description	Cinema Owners/Admins can view all the bookings for a movie
Primary pathway	1. User enters the system 2. User is presented with a menu where the browse <*movie status*> movies buttons exists and clicks on one 3. User is presented with a list of movies 4. User clicks on a movie and then the view details button 5. User views all the details for a movie and clicks the view bookings button 6. User can search for a specific booking
Exception pathway	1. User authentication failed due to incorrect credentials 2. User does not select a movie and proceeds to view their details, an alert pops up

Appendix C: Use case descriptions part 3

Table 21: Use cases for watching a trailer (Cinema Owners/Admins)

UC012, UC063	
Title	Watch a movie's trailer (Cinema Owner/Admin)
Description	Cinema Owners/Admins can watch a trailer for a movie
Primary pathway	1. User enters the system 2. User is presented with a menu where the browse <*movie status*> movies buttons exists and clicks on one 3. User is presented with a list of movies 4. User clicks on a movie and then the view details button 5. User views all the details for a movie and clicks the watch button 6. Alternatively, when an Admins adds a movie and uploads a trailer, they can click the watch trailer button to watch it
Exception pathway	1. User authentication failed due to incorrect credentials 2. User does not select a movie and proceeds to view their details, an alert pops up 3. Admin does not upload a trailer when adding a movie, the watch trailer button is disabled

Table 22: Use cases for logging out for Cinema Owners/Admins

UC013, UC064	
Title	Logout (Cinema Owner/Admin)
Description	Cinema Owners/Admins can logout from the system
Primary pathway	1. User enters the system 2. User is presented with a menu where the logout button is located on the top right corner and clicks on it
Exception pathway	1. User authentication failed due to incorrect credentials

Table 23: Use case for validating movie data

UC055	
Title	Valid movie data
Description	Admins validate movie data before adding a movie
Primary pathway	1. User enters the system 2. User is presented with a menu where the add movie button exists and clicks on it 3. User proceeds to add a movie without producing an alert
Exception pathway	1. User authentication failed due to incorrect credentials 2. User is unable to add a movie until the data is validated, an alert pops up that informs how the data should be formatted

Appendix C: Use case descriptions part 4

Table 24: Use cases for editing account details or deleting it for Customers

UC105, UC106	
Title	Edit account details or delete it (Customer)
Description	Customers can edit their account details or delete it
Primary pathway	1. User enters the system 2. User presses the my account button 3. User can edit their details and press the update my details button 4. Alternatively, the user can click the delete account button and enter their username and password to delete it
Exception pathway	1. User authentication failed due to incorrect credentials 2. User enters data in invalid format, an alert pops up 3. User enters incorrect username and password and cannot delete their account

Table 25: Use case for recovering Customers password

UC108	
Title	Recover password (Customer)
Description	Customers can recover their password
Primary pathway	1. User enters the system 2. User presses the login button (if they are logged out) 3. User presses the forgot your password hyperlink 4. User enters their username and secret keyword to reveal their password
Exception pathway	1. User enters incorrect username and secret keyword

Table 26: Use case for logging out Customers

UC111	
Title	Logout (Customer)
Description	Customers can logout the system
Primary pathway	1. User enters the system 2. User presses the logout button (if they are logged in)
Exception pathway	1. User authentication failed due to incorrect credentials

Appendix D: Web system hover animations and seat legend

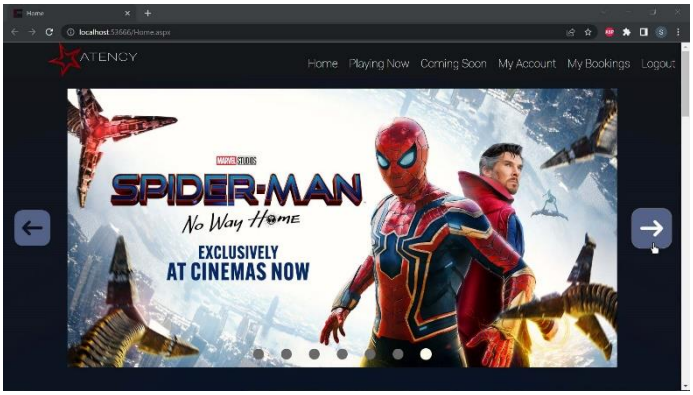

Figure 19: Next hero image button hover

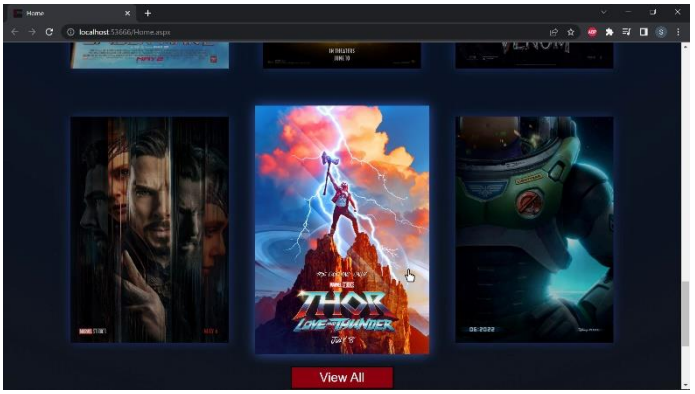

Figure 20: Movie details fast access via poster hover on home page

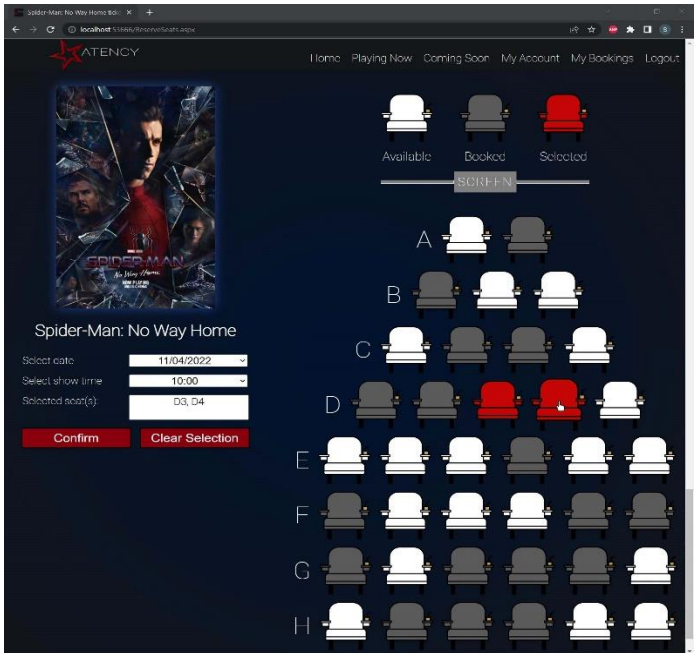

Figure 21: Mouse hover on an available/selected seat

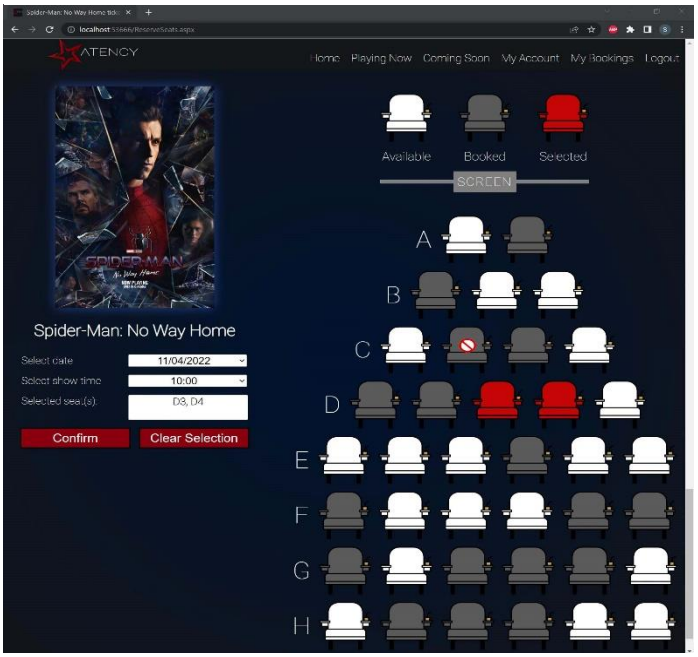

Figure 22: Mouse hover on an already booked seat

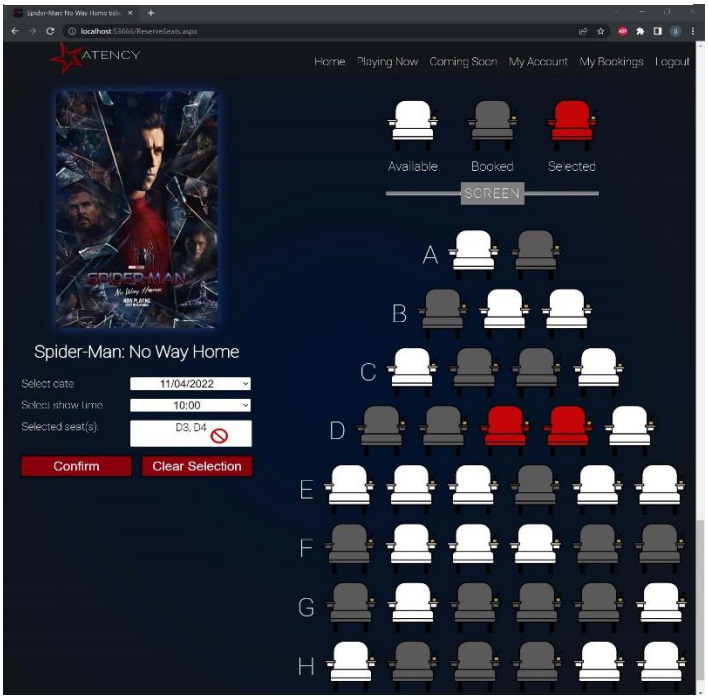

Figure 23: Manually edit seat text not allowed

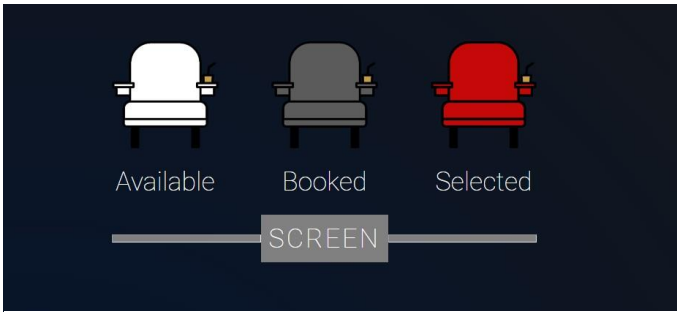

Figure 24: Seat legend

Appendix E: Screens reused to reduce code duplication part 1

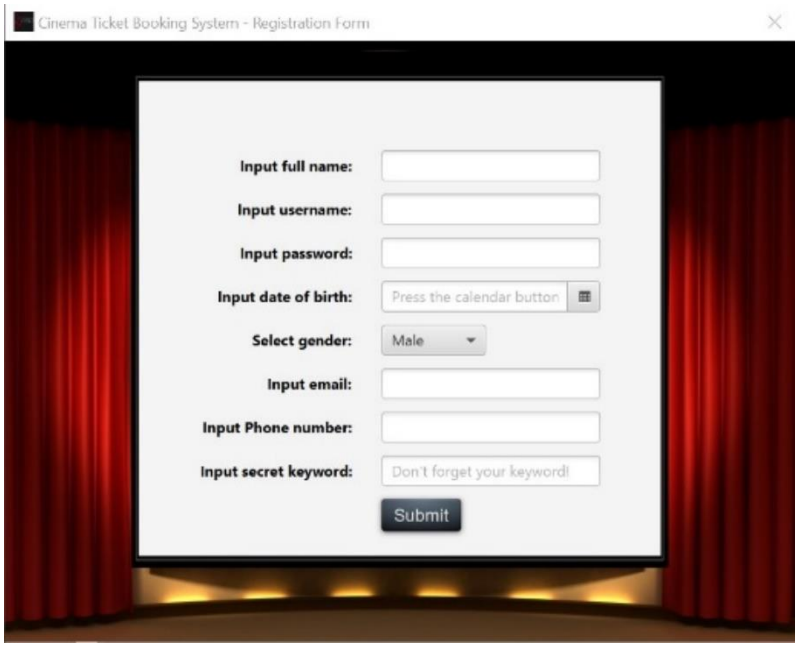

Figure 25: Cinema Owner registration form

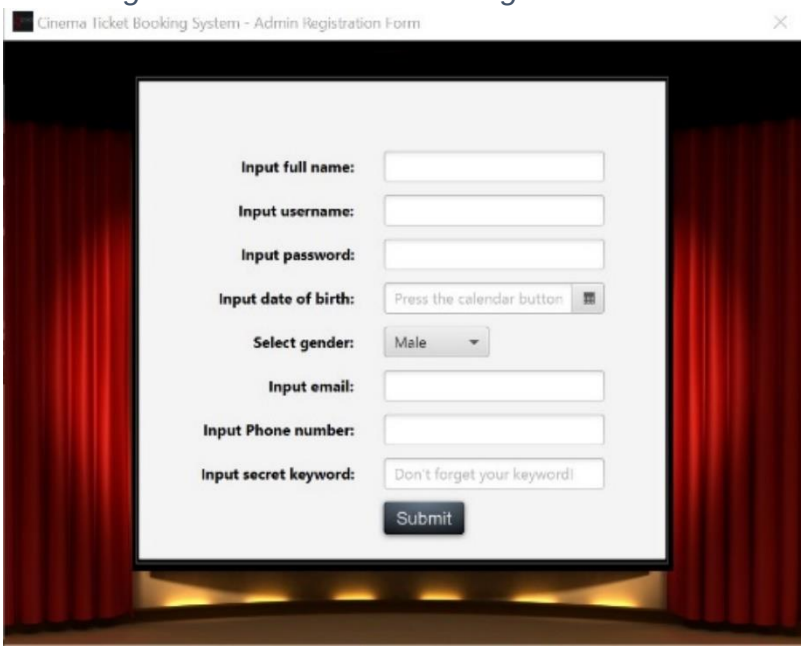

Figure 26: Cinema Owner adding an Admin

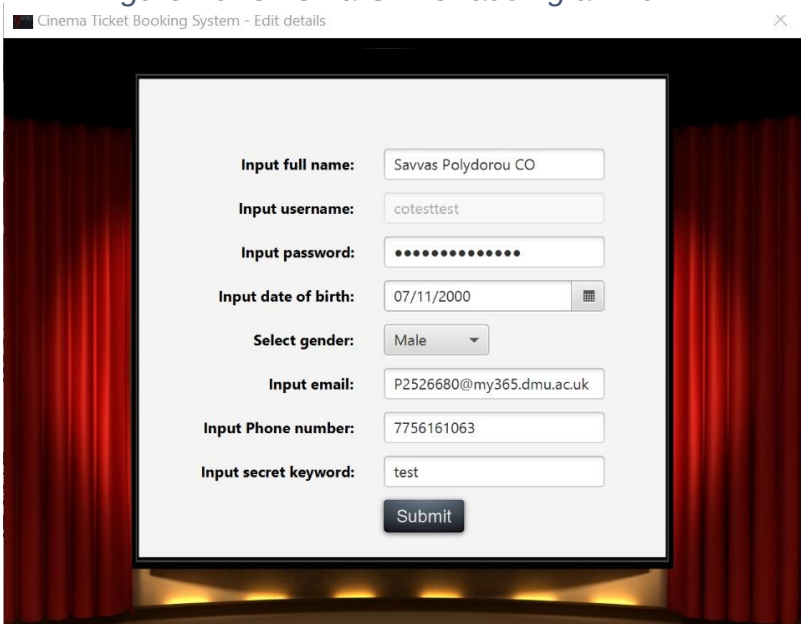

Figure 27: Cinema Owner/Admin editing their details

Appendix E: Screens reused to reduce code duplication part 2

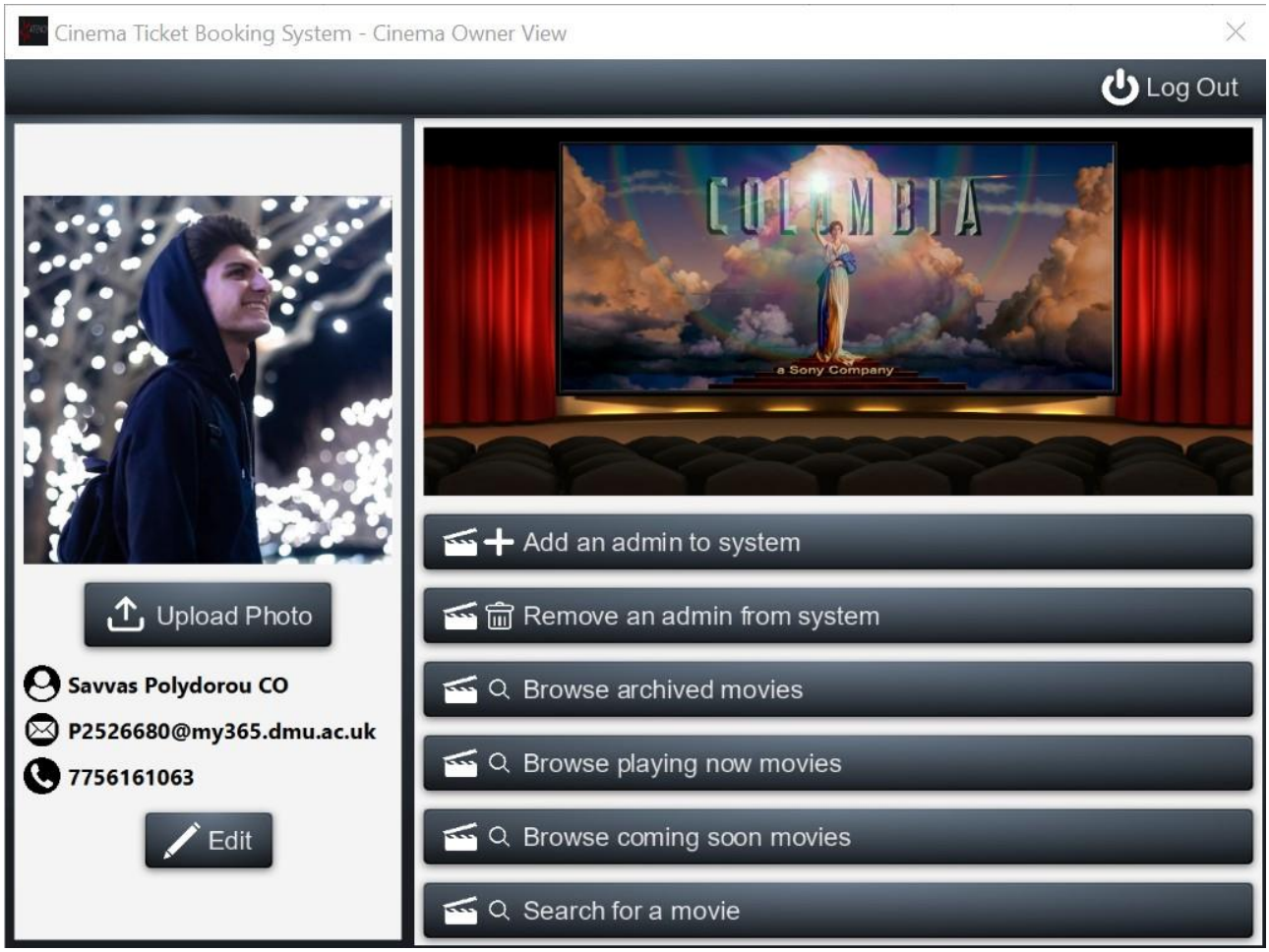

Figure 28: Cinema Owner menu controls

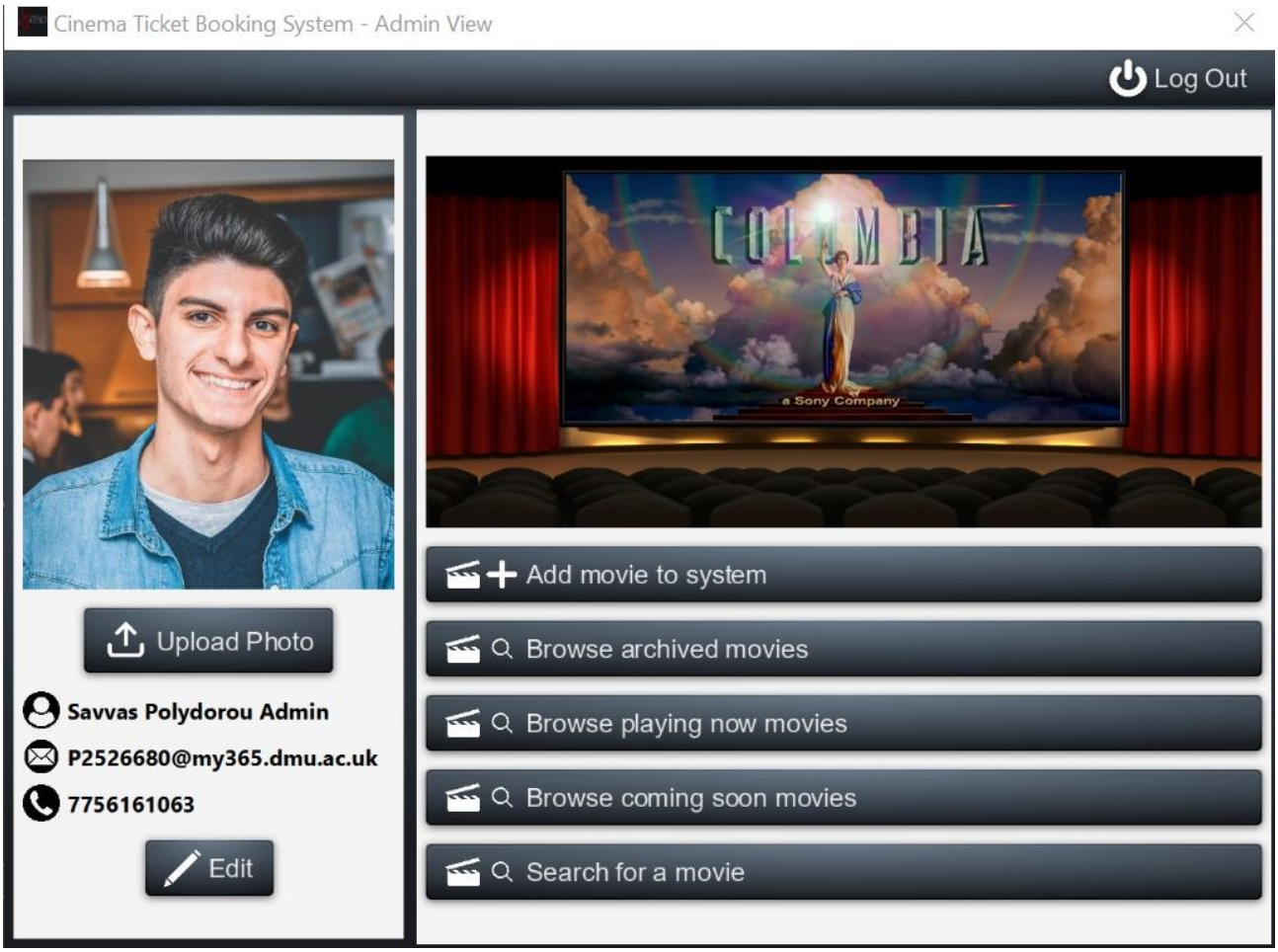

Figure 29: Admin menu controls

Appendix E: Screens reused to reduce code duplication part 3

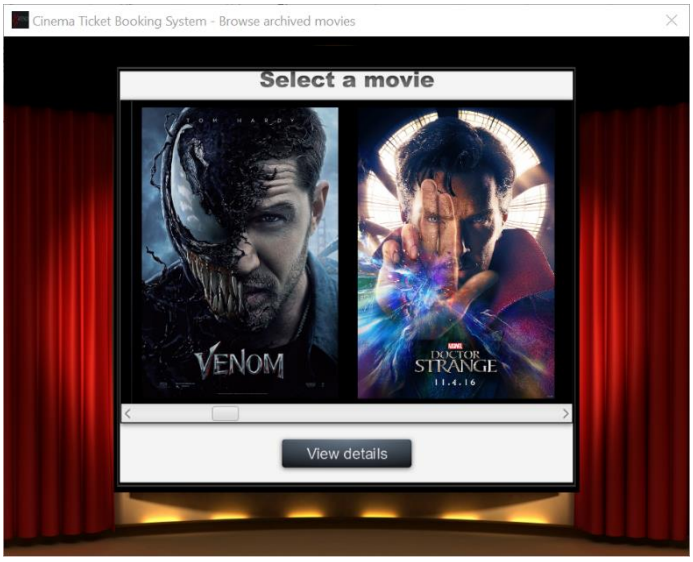

Figure 30: Browse archived movies Cinema Owner control

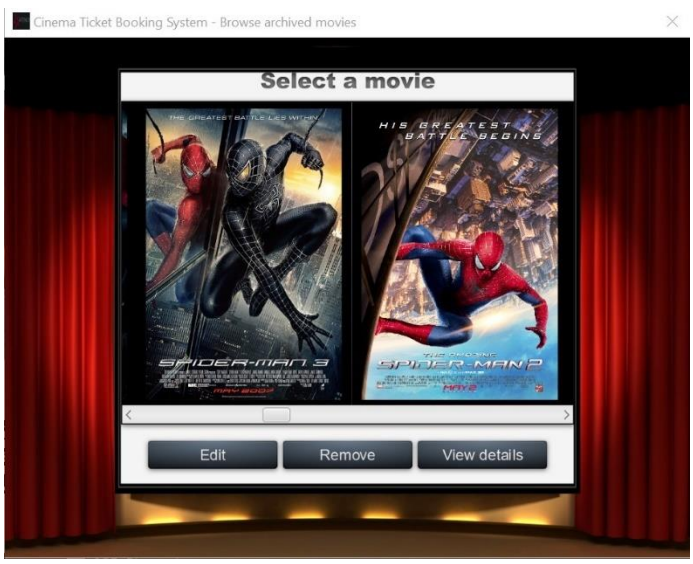

Figure 31: Browse archived movies Admin controls

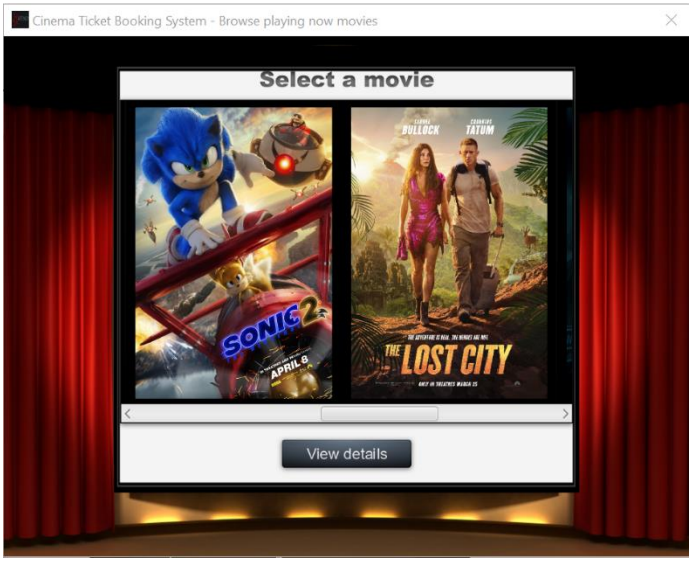

Figure 32: Browse playing now movies Cinema Owner control

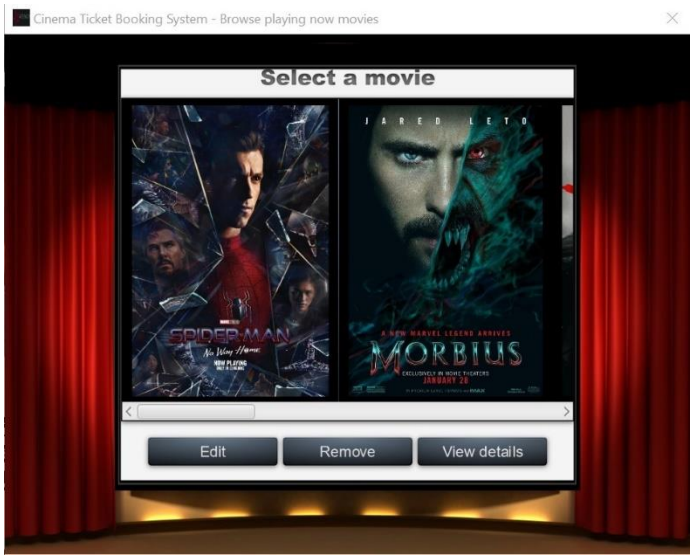

Figure 33: Browse playing now movies Admin controls

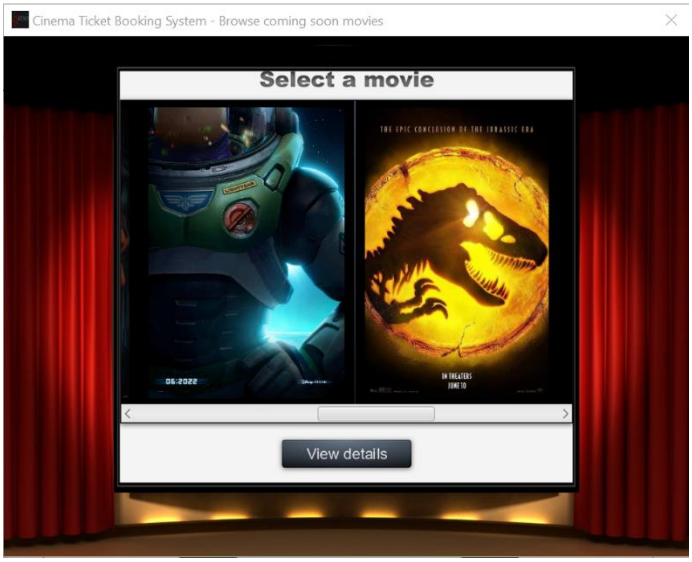

Figure 34: Browse coming soon movies Cinema Owner control

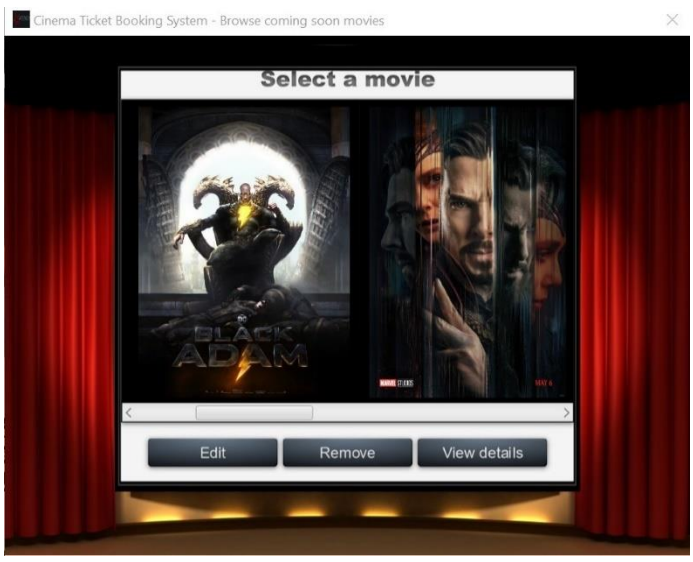

Figure 35: Browse coming soon movies Admin controls

Appendix E: Screens reused to reduce code duplication part 4

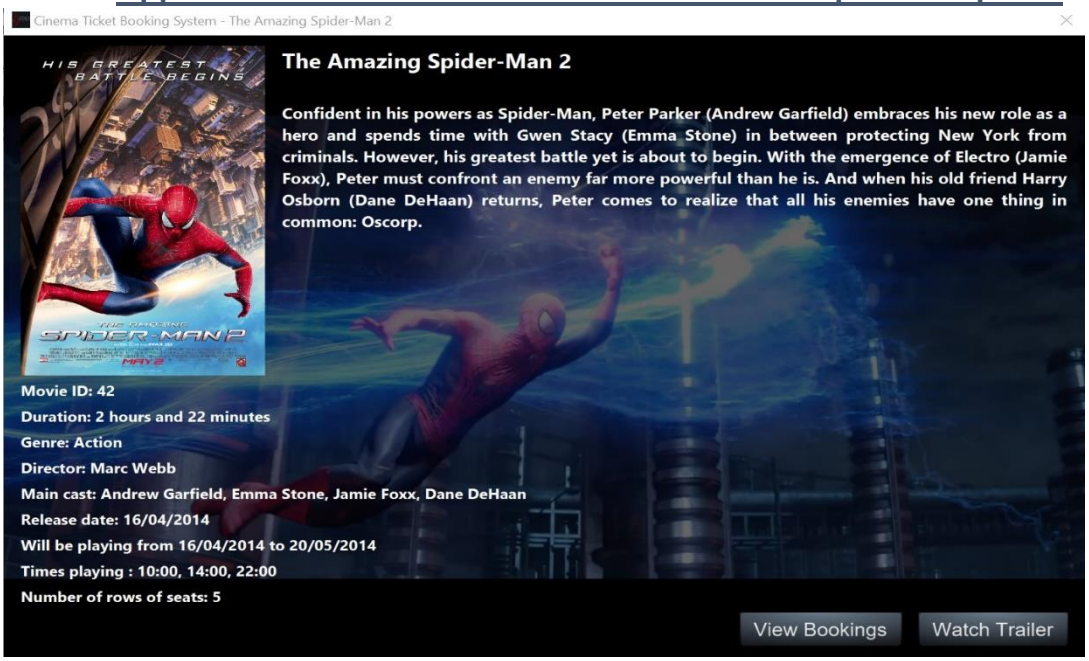

Figure 36: View movie details with background trailer

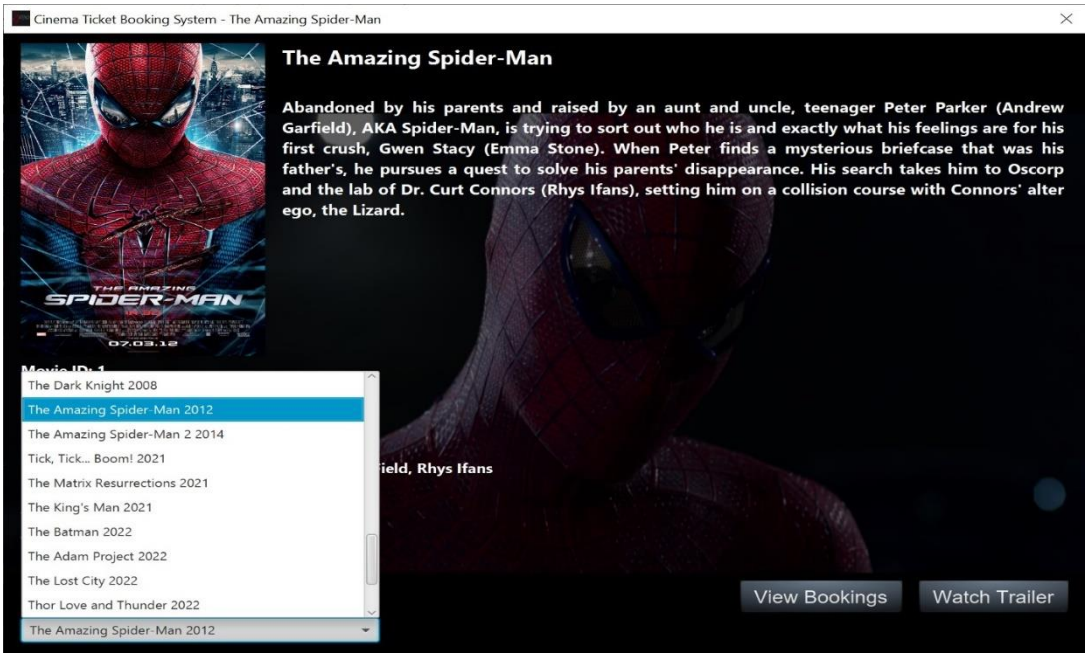

Figure 37: Search for a movie screen and view its details with background trailer

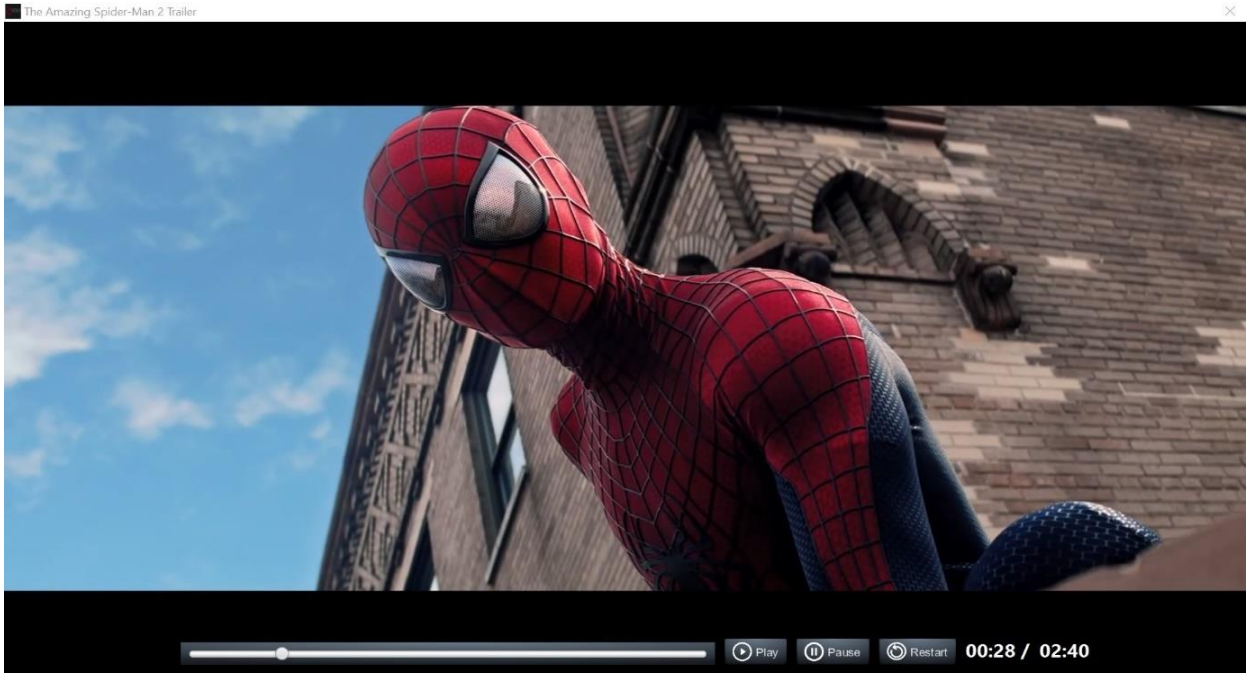

Figure 38: Movie trailer on full screen with sound and controls

Appendix E: Screens reused to reduce code duplication part 5

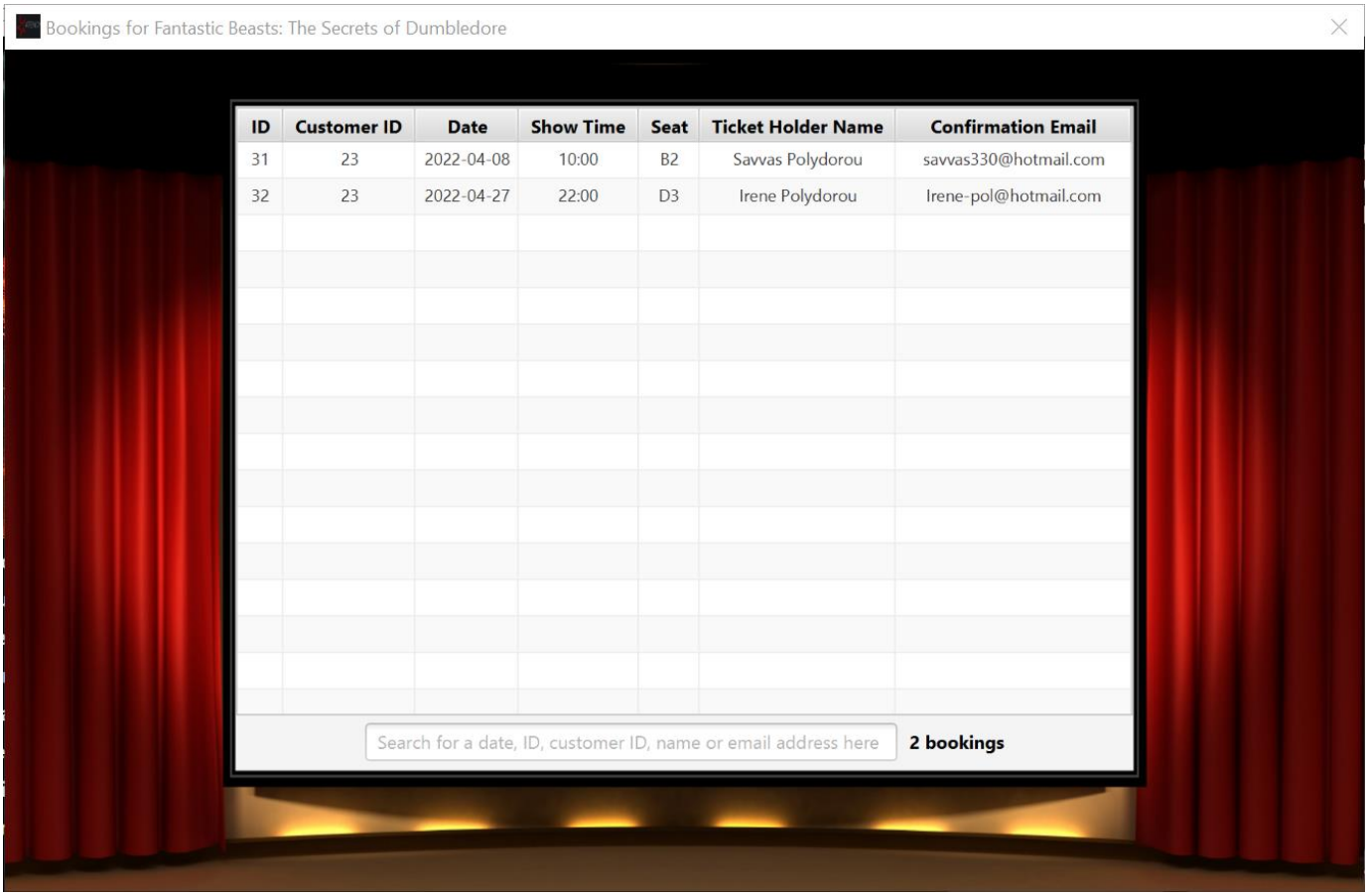

Figure 39: Movie bookings Cinema Owner controls

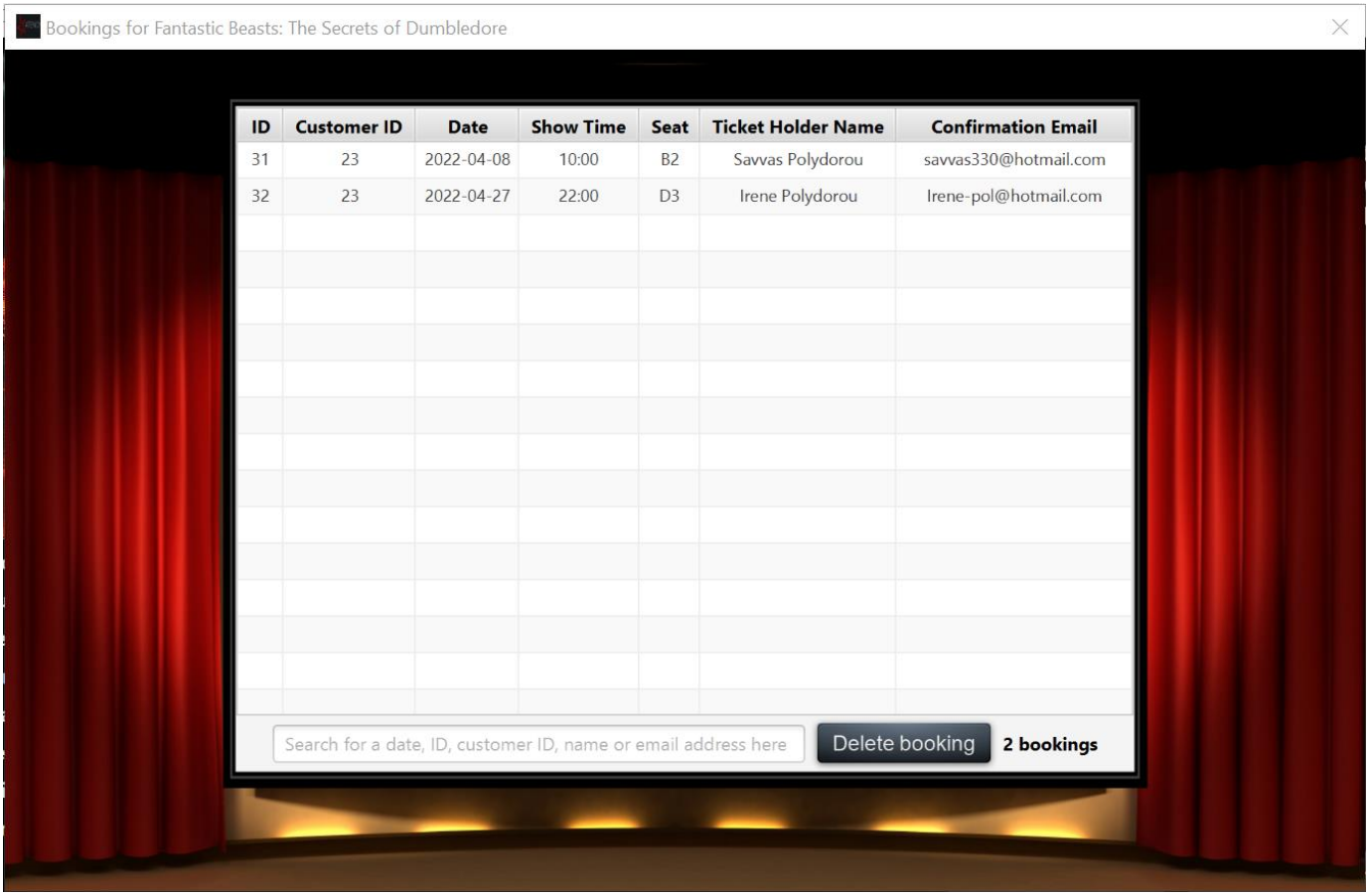

Figure 40: Movie bookings Admin controls

Appendix E: Screens reused to reduce code duplication part 6

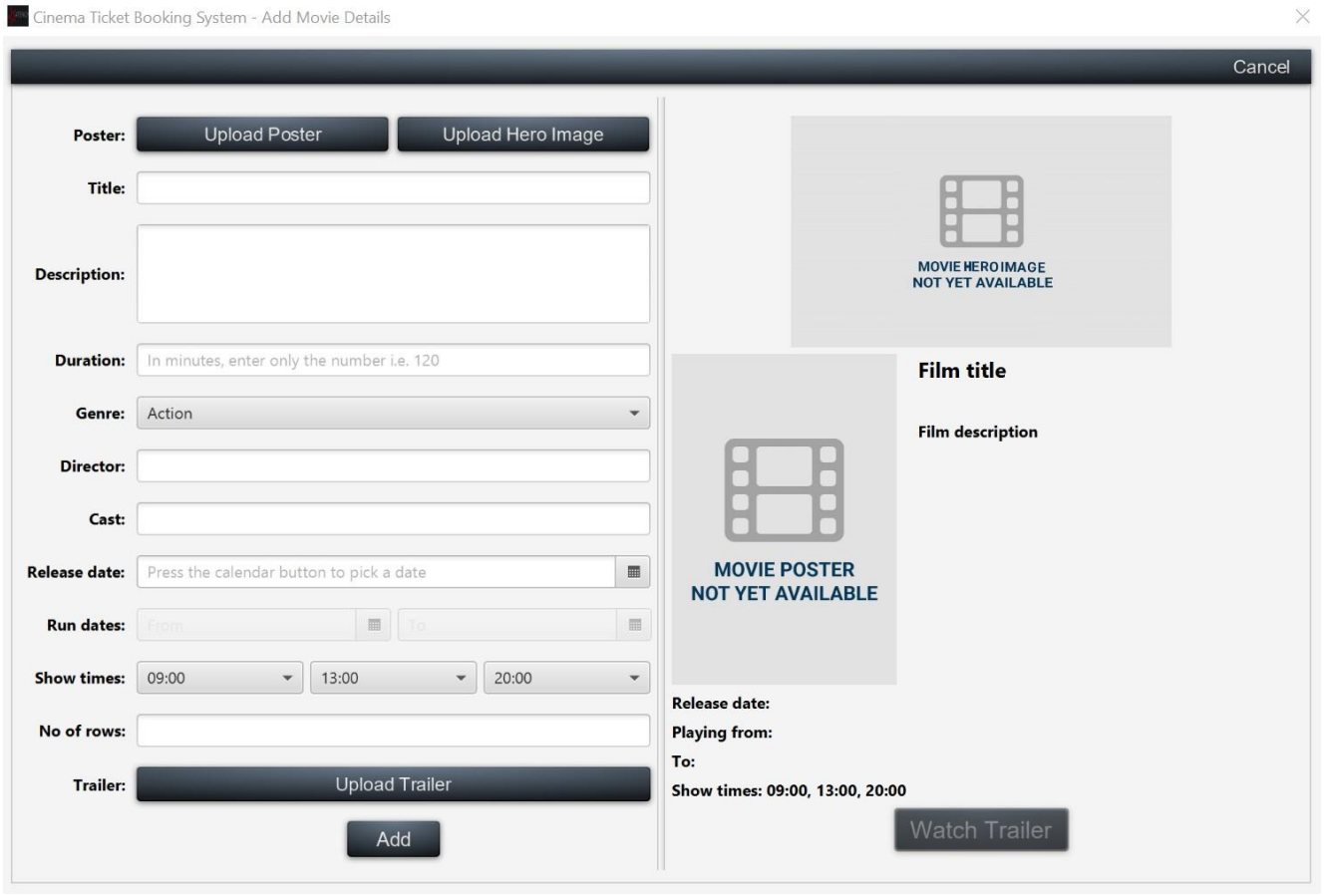

Figure 41: Add movie to system

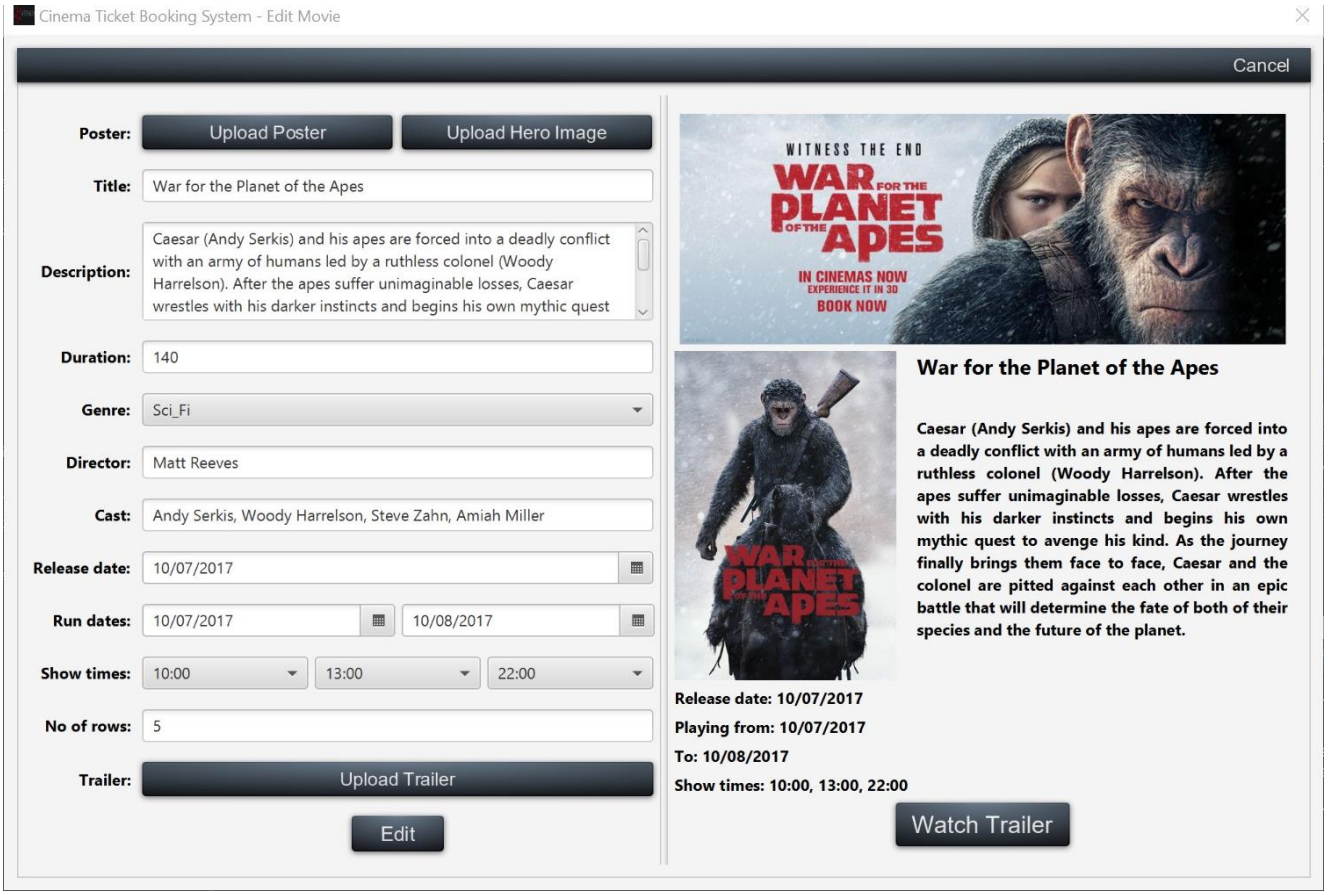

Figure 42: Edit movie details

Appendix F: Database connectivity in Java

```
public ArrayList<Bookings> findBookingsByMovieID(int MovieID) throws SQLException
{
    Bookings booking = new Bookings();
    ArrayList<Bookings> bookings = new ArrayList<Bookings>();

    Connection connection = null;
        try {
                connection = DriverManager.getConnection(url, user, password);
                String sql = "SELECT * FROM tblBooking WHERE [MovieID] = ?";

                PreparedStatement sqlstatement = connection.prepareStatement(sql);
                sqlstatement.setInt(1,MovieID);
                ResultSet rs = sqlstatement.executeQuery();
                while(rs.next())
                {
                        int bookingID = rs.getInt(1);
                        int movieID = rs.getInt(2);
                        int customerID = rs.getInt(3);
                        Date date = rs.getDate(4);
                        String showtime = rs.getString(5);
                        String seat = rs.getString(6);
                        String ticketHolderName = rs.getString(7);
                        String emailConfirmation = rs.getString(8);
                        booking.setID(bookingID);
                        booking.setCustomerID(Integer.toString(customerID));
                        booking.setMovieID(movieID);
                        booking.setDate(date.toLocalDate());
                        booking.setShowTime(showtime);
                        booking.setSeat(seat);
                        booking.setTicketHolderName(ticketHolderName);
                        booking.setEmailConfirmation(emailConfirmation);
                        bookings.add(booking);
                        booking = new Bookings();
                }
        }
        catch(SQLException e)
        {
                e.printStackTrace();
        }
        finally
        {
                connection.close();
        }

    return bookings;
      }
```

Figure 43: Database connectivity in Java

This piece of code allows the Cinema Owner/Admin to get all the bookings made for a particular movie. The bookings are made by the Customers/Guests using the web-based system, and the data are read by the desktop software.

Appendix G: Database connectivity in C#

```
        public int addBooking(clsBooking booking)
        {
            connection.Open();
            int bookingID = 0;
            try
            {
                //insert values to table
                comm = new SqlCommand
                    ("INSERT INTO tblBooking (MovieID, CustomerID, Date, ShowTime, Seat,
TicketHolderName, ConfirmationEmail) "
                    + "VALUES (@MovieID, @CustomerID, @Date, @ShowTime, @Seat,
@TicketHolderName, @ConfirmationEmail); SELECT SCOPE_IDENTITY();", connection);
                comm.Connection = connection;
                comm.CommandType = CommandType.Text;
                {
                    comm.Parameters.AddWithValue("@MovieID", booking.MovieID);
                    comm.Parameters.AddWithValue("@CustomerID", booking.CustomerID);
                    comm.Parameters.AddWithValue("@Date", booking.Date);
                    comm.Parameters.AddWithValue("@ShowTime", booking.ShowTime);
                    comm.Parameters.AddWithValue("@Seat", booking.Seat);
                    comm.Parameters.AddWithValue("@TicketHolderName",
booking.TicketHolderName);
                    comm.Parameters.AddWithValue("@ConfirmationEmail",
booking.ConfirmationEmail);
                    bookingID = Convert.ToInt32(comm.ExecuteScalar());
                }
            }
            catch (SqlException e)
            {
                throw e;
            }
            finally
            {
                connection.Close();
            }
            return bookingID;
        }
```

Figure 44: Database connectivity in C#

This piece of code allows the Customers/Guests to make a booking for a particular movie and add it to the database table.

Appendix H: Database table designs

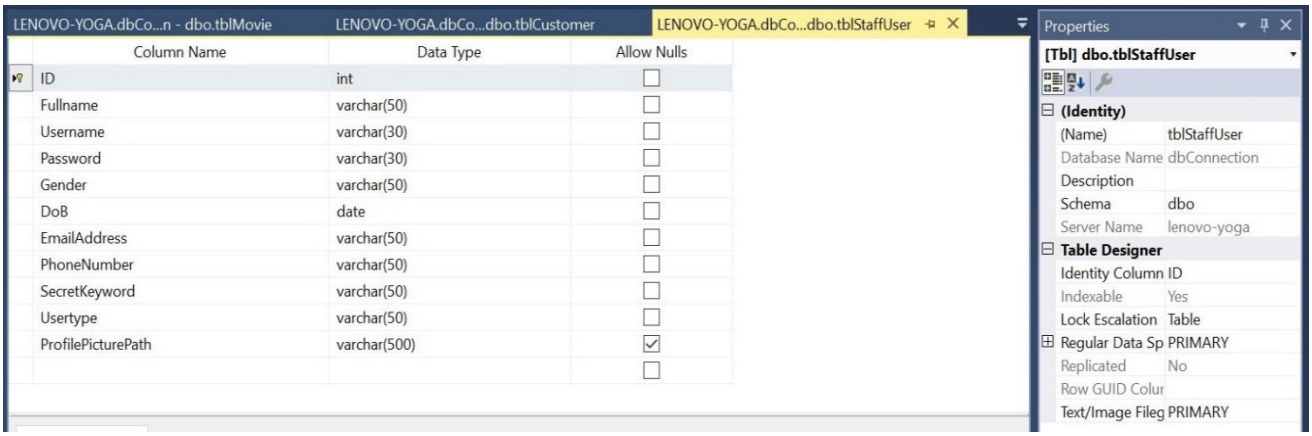

Figure 45: Cinema Owner/Admin table design

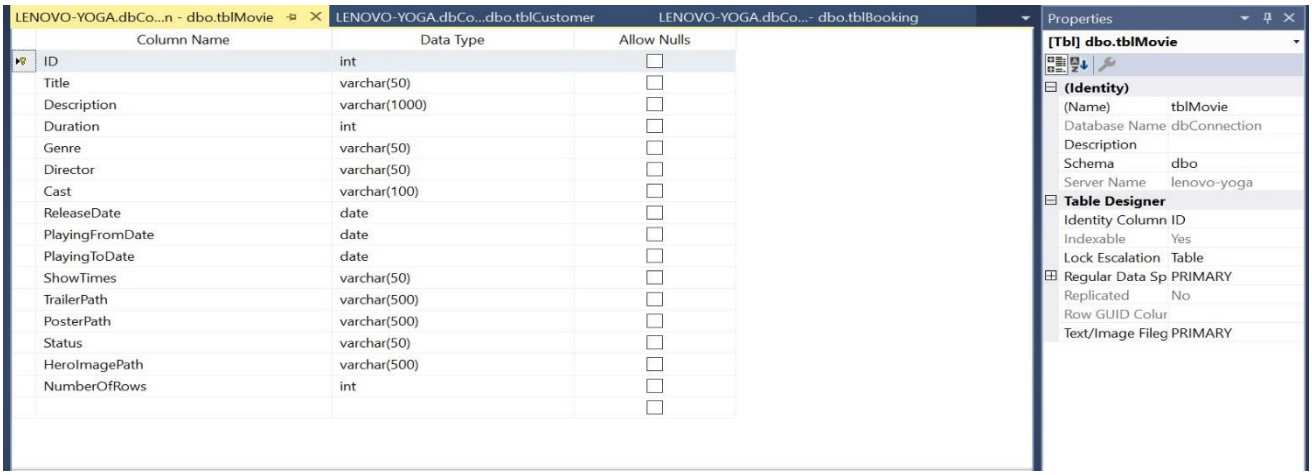

Figure 46: Movie table design

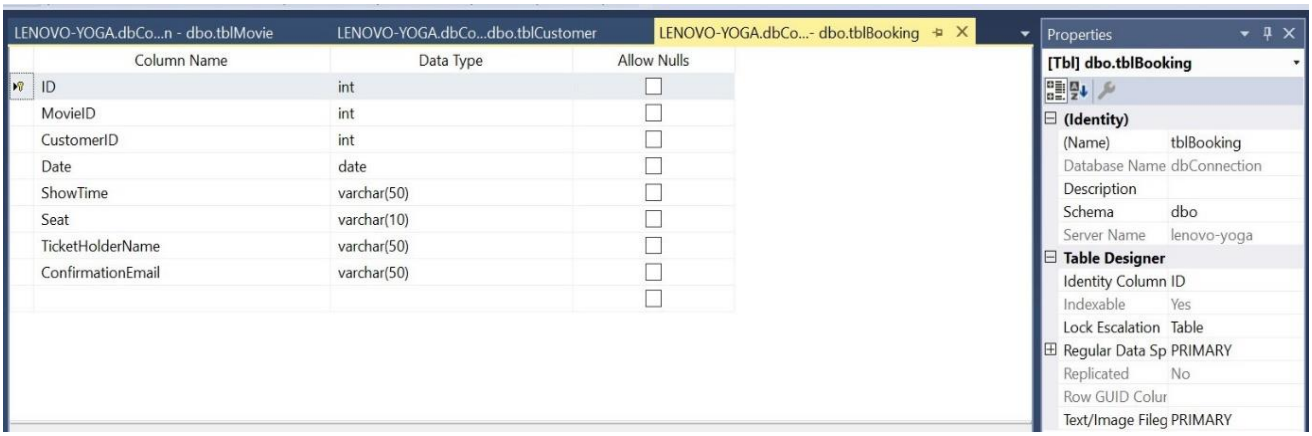

Figure 47: Bookings table design

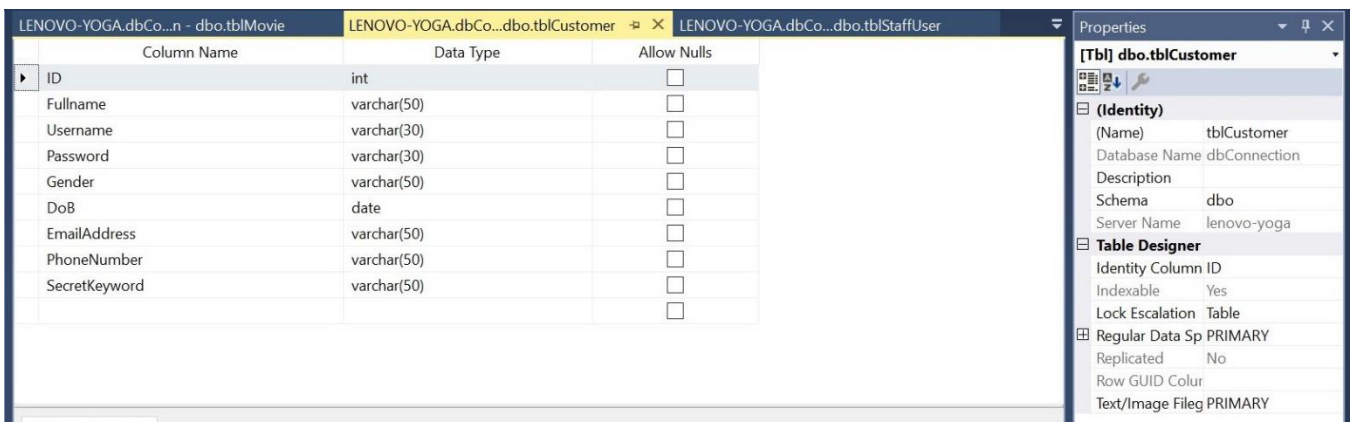

Figure 48: Customers table design

Appendix I: Converting milliseconds to minutes and seconds function snippet

```
            private String getTime(Duration time)
            {
                  int minutes = (int) time.toMinutes();
                  int seconds = (int) time.toSeconds();
                  //if seconds go 61 then reset to 1
                  if(seconds > 59)
                        seconds%= 60;
                  //if minutes go 61 then reset to 1 (unlikely scenario)
                  if(minutes > 59)
                        minutes%= 60;
                  //format to display two digits, a :, and another two digits
                  return String.format("%02d:%02d", minutes, seconds);
            }

            //sets the max value of the slider to the total duration of the video
            mpVideo.totalDurationProperty().addListener(new ChangeListener<Duration>(){

                  @Override
                  public void changed(ObservableValue<? extends Duration>
observableValue, Duration oldDuration, Duration newDuration)
                  {
                        slider.setMax(newDuration.toSeconds());
                        lblTotalTime.setText(getTime(newDuration));

                  }

            });

//bind the current time label with the current time of the video
lblCurrentTime.textProperty().bind(Bindings.createStringBinding(new Callable<String>(){
                  @Override
                  public String call() throws Exception
                  {
                        return getTime(mpVideo.getCurrentTime()) + " /";
                  }
            }, mpVideo.currentTimeProperty()));
```

Figure 49: Time conversion and use in JavaFX Media Player

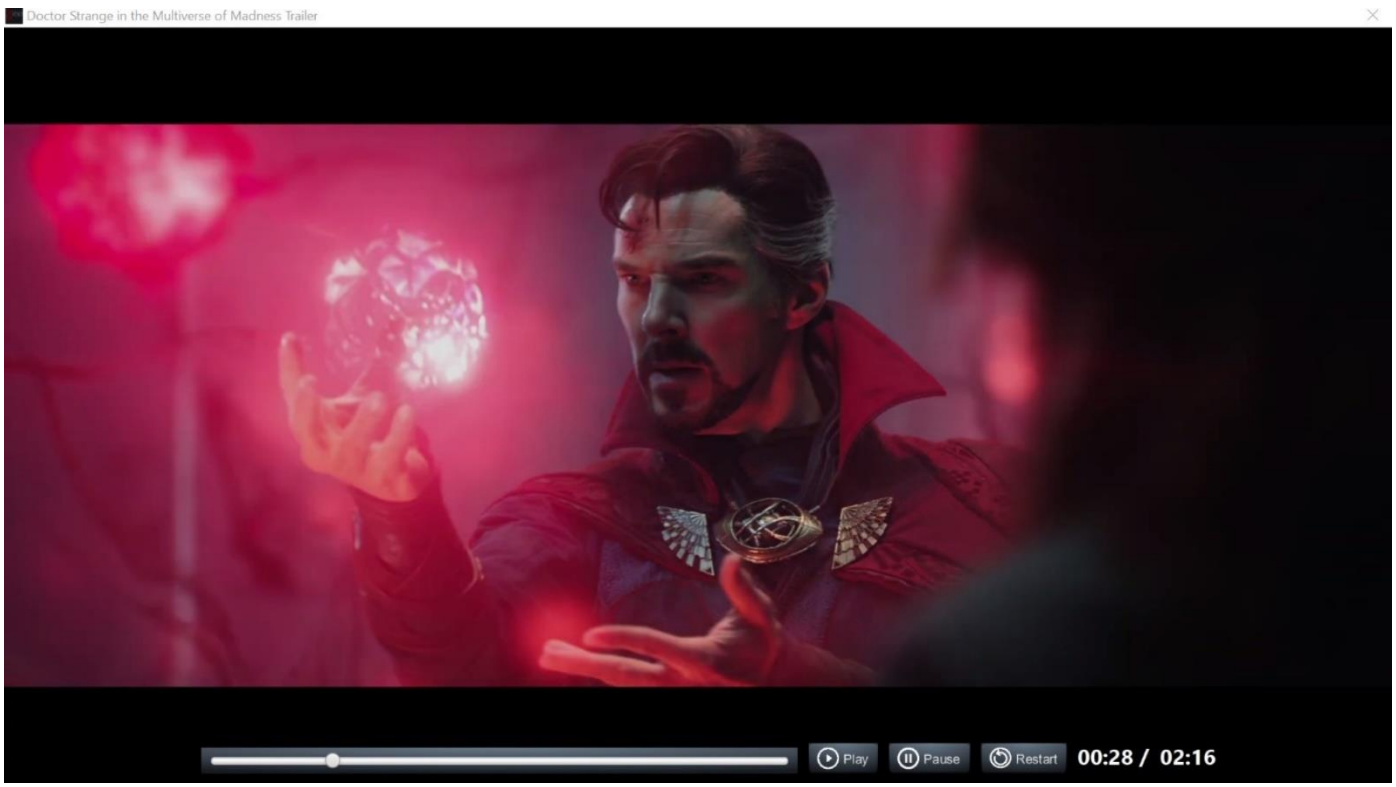

Figure 50: Video durations in minutes:seconds format

Appendix J: HTML5 video tag with controls code snippet

```
<div class="movieTrailer">
    <video id="trailer" runat="server" controls="controls"></video>
</div>
```

Figure 51: Built-in HTML5 video tag

The controls= “controls” attribute is where the browser is told to display the pre-built controls to the bottom of the video’s screen.

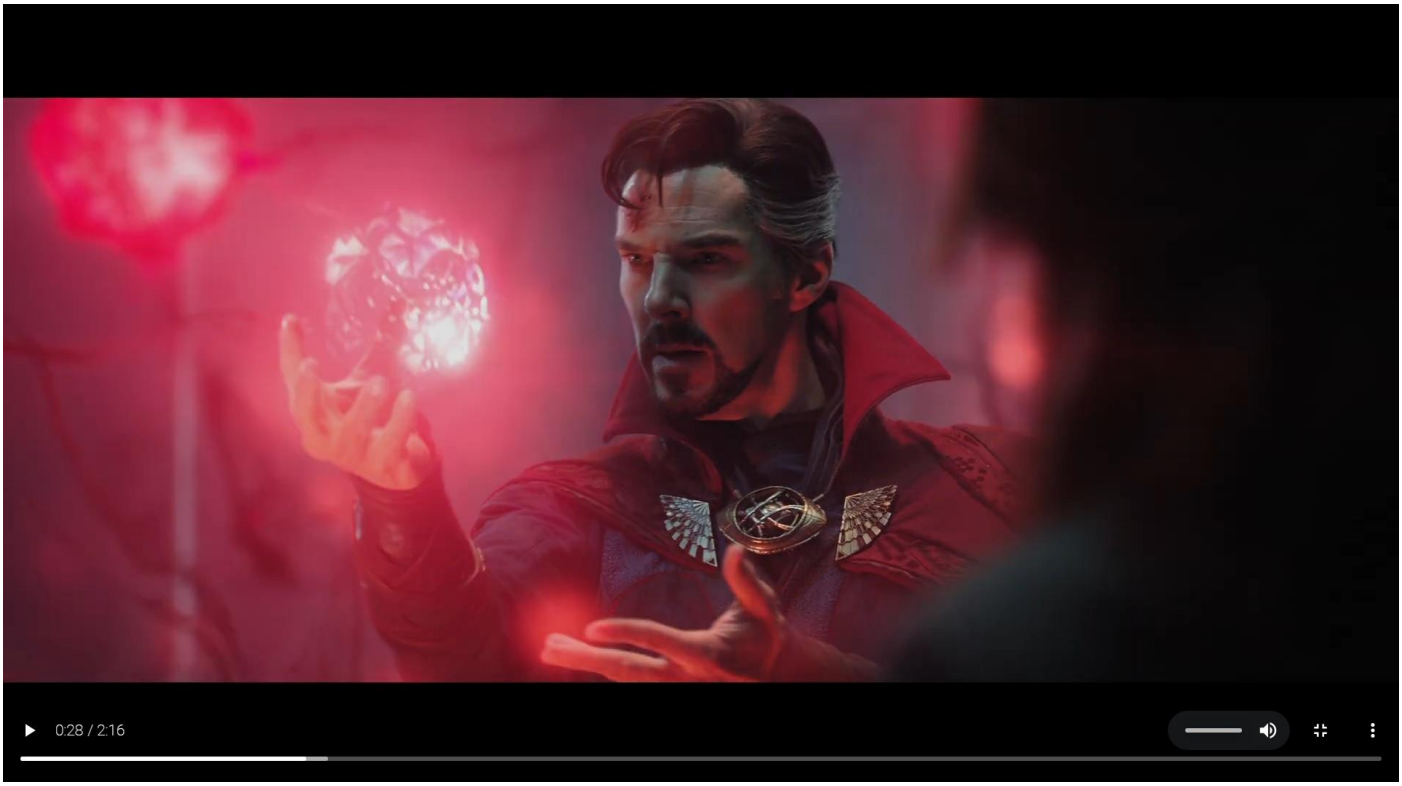

Figure 52: HTML5 video with controls

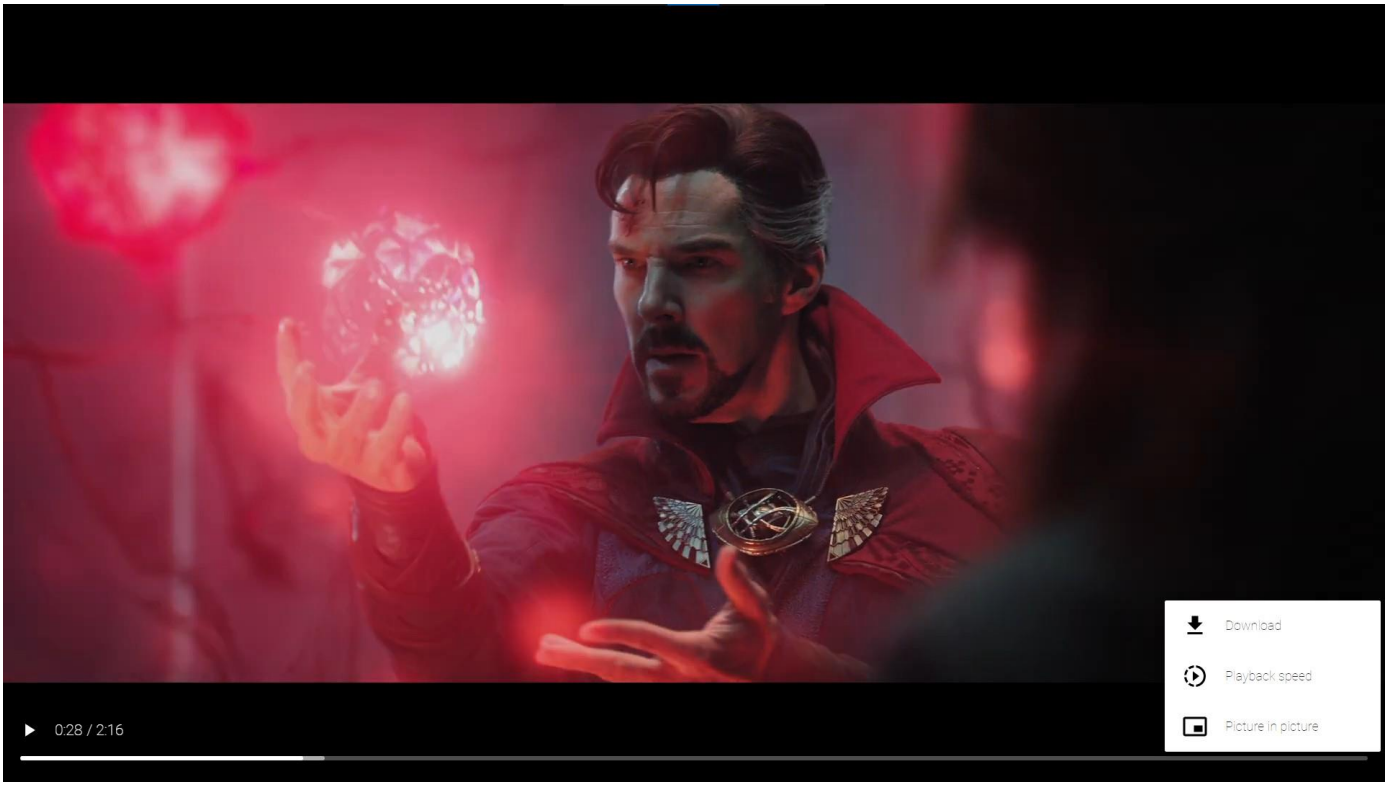

Figure 53: HTML5 video with advanced controls

Appendix K: Seat distribution and availability check functions

```
    private void insertRowsAndColumnsWithSeats(int index, int seats)
    {
        //each row must have a panel with display flex
        rowsPanel = new Panel();
        rowsPanel.Attributes.Add("class", "rows");
        //will display alphabetic characters for each row
        char letter = (char)('A' + (char)(index % 26));
        Label lblLetter = new Label();
        lblLetter.Text = letter.ToString();
        rowsPanel.Controls.Add(lblLetter);
        for (int i = 0; i < seats; i++)
        {
            ImageButton chair = new ImageButton();
            chair.ID = letter + (i + 1).ToString();
            chair.ImageUrl = "/sprites/ChairAvailable.png";
            chair.Attributes.Add("class", "available");
            chair.AlternateText = "Available chair";
            chair.Click += Chair_Click;
            rowsPanel.Controls.Add(chair);
        }
        //increment seats so next time the method is called
        //there will be 1 more chair in each row
        if (index < 4)
            minColumns++;
        seatSelection.Controls.Add(rowsPanel);
    }
```

Figure 54: Seat distribution

```
    private void CheckAllControls(Control parentControl)
    {
        //Get all controls
        foreach (Control childControl in parentControl.Controls)
        {
            //Get all image buttons in the parent control
            if (childControl.GetType().ToString() == "System.Web.UI.WebControls.ImageButton")
            {
                var imageButton = childControl as ImageButton;
                //check all bookings for the selected movie
                //check the values of the drop down lists
                //if they match those of a booking set the seat to booked
                //and disable it
                for (int i = 0; i < bookings.Count; i++)
                    if (imageButton.ID.Equals(bookings[i].Seat))
                        if
(dropDownDates.SelectedValue.Equals(bookings[i].Date.ToShortDateString()))
                            if (dropDownShowTimes.SelectedValue.Equals(bookings[i].ShowTime))
                            {
                                imageButton.Enabled = false;
                                imageButton.ImageUrl = "/sprites/ChairBooked.png";
                                imageButton.AlternateText = "Booked seat";
                            }
            }
            //Loop for all row panels
            if (childControl.HasControls())
                CheckAllControls(childControl);
        }
    }
```

Figure 55: Seat availability check

Appendix L: Usage of the computer's local date and time part 1

```
int hourOfMovie = Convert.ToInt32(customer.Bookings[i].ShowTime.Substring(0,
customer.Bookings[i].ShowTime.IndexOf(':')));
int minutesOfMovie =
Convert.ToInt32(customer.Bookings[i].ShowTime.Substring(customer.Bookings[i].ShowTime.IndexOf
(':') + 1));

            if (DateTime.Now.Date > customer.Bookings[i].Date)
            {
                btnCancel.Enabled = false;
                btnEdit.Visible = false;
                btnEdit.Enabled = false;
                btnCancel.Text = "Booking completed";
            }
            else if (DateTime.Now.Date == customer.Bookings[i].Date)
                if (DateTime.Now.TimeOfDay > new TimeSpan(hourOfMovie, minutesOfMovie, 0))
                {
                    btnCancel.Enabled = false;
                    btnEdit.Visible = false;
                    btnEdit.Enabled = false;
                    btnCancel.Text = "Booking completed";
                }
```

Figure 56: Using local date and time to control functions

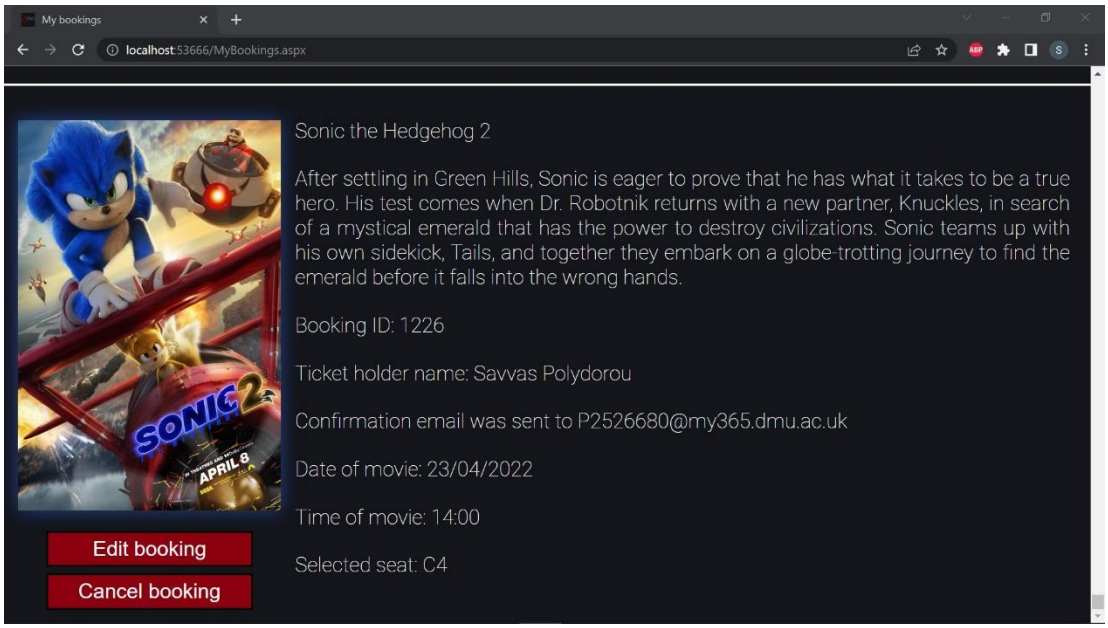

Figure 57: Editing or cancelling booking available (date, time is in the future)

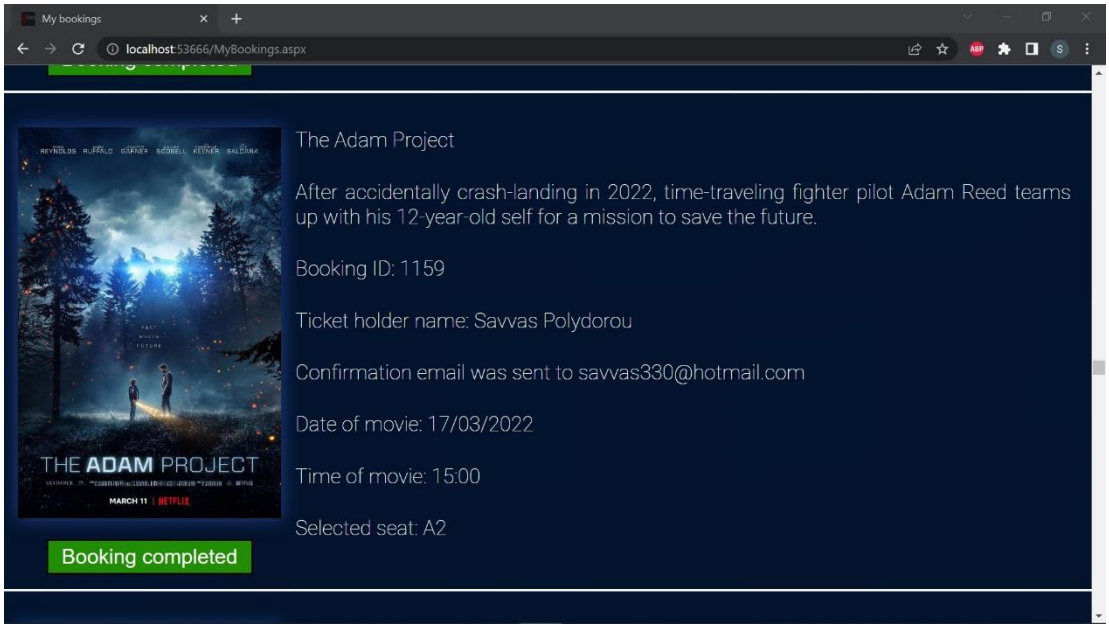

Figure 58: Booking completed as the date of the movie is in the past

Appendix L: Usage of the computer's local date and time part 2

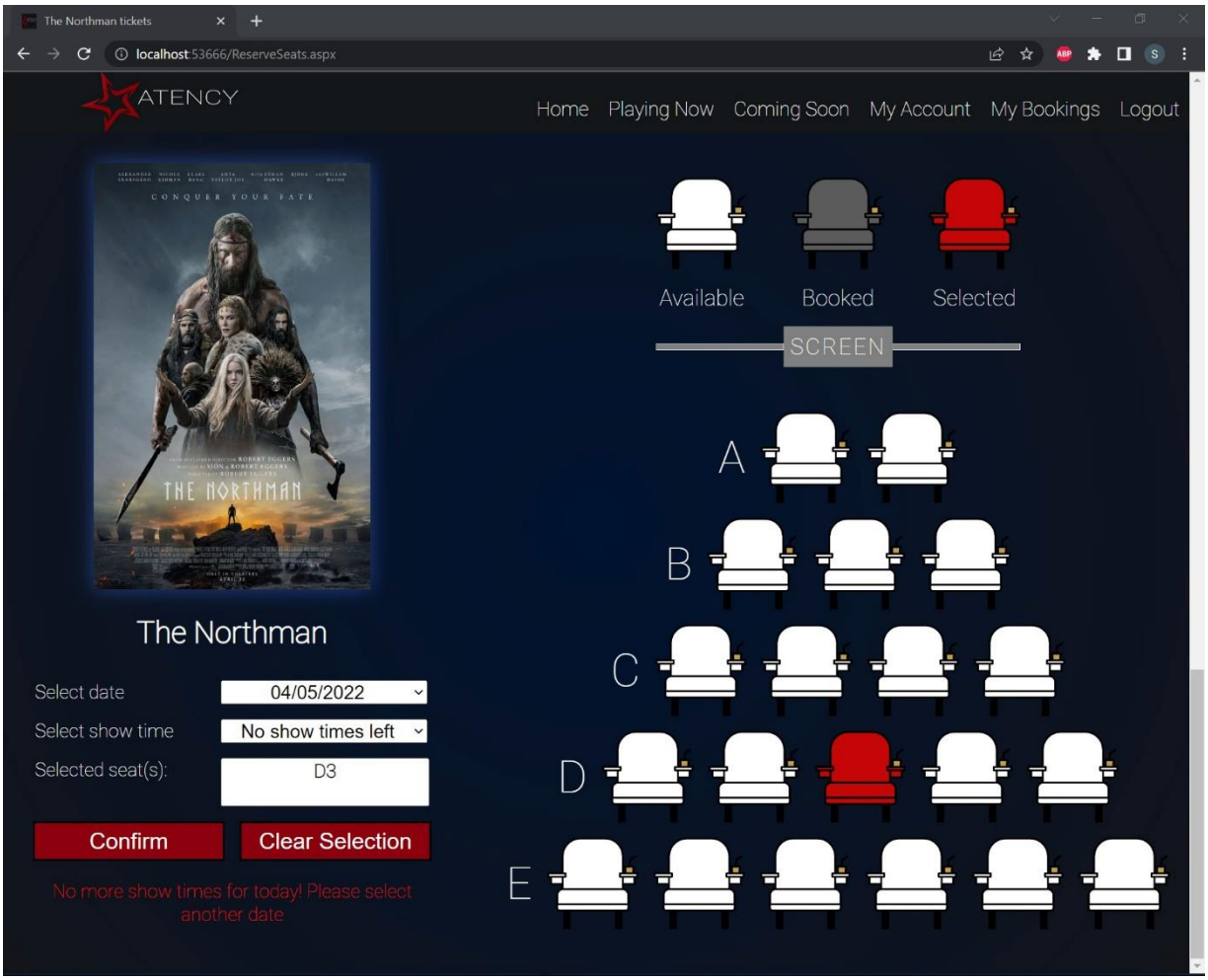

Figure 59: No show times left for today

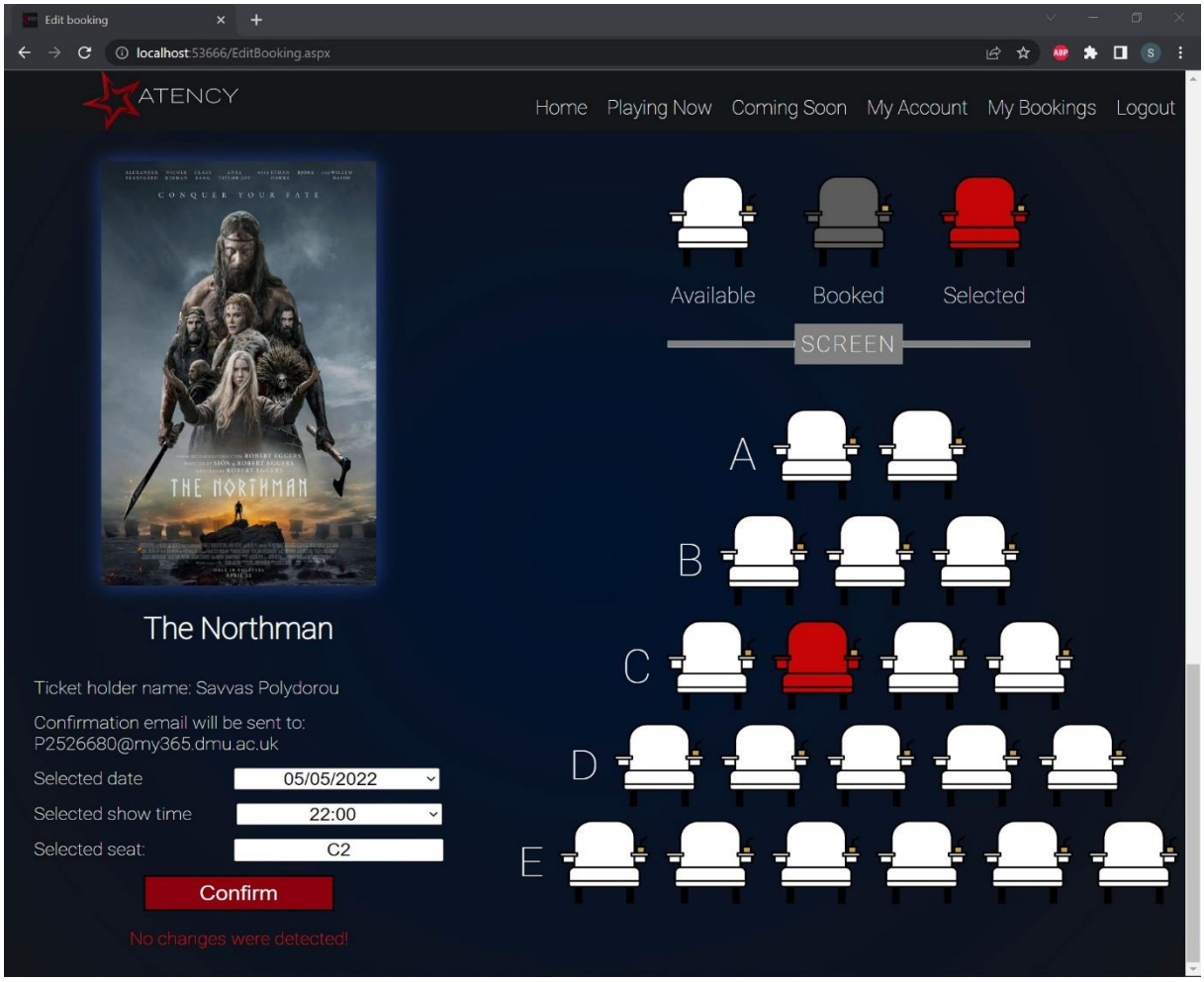

Figure 60: Customer tries to update a booking without making any changes

Appendix M: Movie data validation function in Java part 1

```
	public String dataValidation(String mTitle, String mDescription, String mDuration,
			String mDirector, String mCast, LocalDate releaseDate,
			LocalDate playingFromDate, LocalDate playingToDate,
			String mTrailerPath, String mPosterPath, String mHeroImagePath, String
mNumberOfRows)
	{
		String errorMessage = "";
		final int minBoundaryForEverything = 1;
		final int maxBoundaryDescription = 1000;
		final int maxBoundaryCast = 100;
		final int maxBoundaryForHeroImageAndPosterAndTrailerPaths = 500;
		final int maxBoundaryForTitleDirector = 50;
		//if any of the fields are empty return the error message
		if(mTitle.isEmpty() || mDescription.isEmpty() || mDuration.isEmpty() ||
				mDirector.isEmpty() || mCast.isEmpty() || releaseDate == null ||
				playingFromDate == null ||  playingToDate == null ||
mTrailerPath.isEmpty() || mPosterPath.isEmpty() || mHeroImagePath.isEmpty())
		{
			errorMessage = "Fields cannot be empty! Please make sure you also upload
a poster, a hero image and a trailer!";
			return errorMessage;
		}
		//==============Title validation==================
		if(!((mTitle.length() >= minBoundaryForEverything) && (mTitle.length() <=
maxBoundaryForTitleDirector)))
			errorMessage+= "The title must be between " + minBoundaryForEverything +
" and " +  maxBoundaryForTitleDirector + " characters!\n";
		//==============Description validation==================
		if(!((mDescription.length() >= minBoundaryForEverything) &&
(mDescription.length() <= maxBoundaryDescription)))
			errorMessage+= "The description must be between " +
minBoundaryForEverything + " and " + maxBoundaryDescription + " characters!\n";

		//==============Duration validation==================
		boolean invalidDataTypeDuration = false;
		boolean outOfBoundsDuration = false;
		try
		{
			int tempDuration = Integer.parseInt(mDuration);
			invalidDataTypeDuration =  false;
			if(!((tempDuration >= minBoundaryForEverything) && (tempDuration <=
Integer.MAX_VALUE)))
				outOfBoundsDuration = true;
		}
		catch(Exception e)
		{
			invalidDataTypeDuration = true;
		}

		boolean invalidDuration = invalidDataTypeDuration || outOfBoundsDuration;
		if(invalidDuration)
			errorMessage+= "\nThe duration can only be positive numbers and is
between logical limits!\n";
		//==============Number of rows validation==================
		boolean invalidDataTypeNumberOfRows = false;
		boolean outOfBoundsNumberOfRows = false;
```

CONTINUED IN NEXT PAGE

Figure 61: Movie data validation

Appendix M: Movie data validation function in Java part 2

```
        try
        {
                int tempNumberOfRows = Integer.parseInt(mNumberOfRows);
                invalidDataTypeNumberOfRows =  false;
                        if(!((tempNumberOfRows >= 5)  && (tempNumberOfRows <= 26)))
                        outOfBoundsNumberOfRows = true;

        }
        catch(Exception e)
        {
                invalidDataTypeNumberOfRows = true;
        }

boolean invalidNumberOfRows = invalidDataTypeNumberOfRows || outOfBoundsNumberOfRows;
                if(invalidNumberOfRows)
                        errorMessage+= "\nThe number of rows can only be greater or equal to 5
and lower or equal to 26!\n";
                //==============Director validation==================
                if(!((mDirector.length() >= minBoundaryForEverything) && (mDirector.length()
<= maxBoundaryForTitleDirector)))
                        errorMessage+= "\nDirector must be between " + minBoundaryForEverything
+ " and " +  maxBoundaryForTitleDirector + " characters long!\n";
                //==============Cast validation==================
                if(!((mCast.length() >= minBoundaryForEverything) && (mCast.length() <=
maxBoundaryCast)))
                        errorMessage+= "\nCast must be between " + minBoundaryForEverything + "
and " + maxBoundaryCast + " characters long!\n";
                //==============Release date, playing from, playing to
validation==================
                //no need for validation, future dates and text inputs are already disabled in
AddEditMoviePane
                //==============Trailer path validation==================
                if(!((mTrailerPath.length() >= minBoundaryForEverything) &&
(mTrailerPath.length() <= maxBoundaryForHeroImageAndPosterAndTrailerPaths)))

                        errorMessage+= "\nTrailer path must be between " +
minBoundaryForEverything + " and " + maxBoundaryForHeroImageAndPosterAndTrailerPaths + "
characters long!\n";
                //==============Poster path validation==================
                if(!((mPosterPath.length() >= minBoundaryForEverything) &&
(mPosterPath.length() <= maxBoundaryForHeroImageAndPosterAndTrailerPaths)))

                        errorMessage+= "\nPoster path must be between " +
minBoundaryForEverything + " and " + maxBoundaryForHeroImageAndPosterAndTrailerPaths + "
characters long!\n";
                if(!((mHeroImagePath.length() >= minBoundaryForEverything) &&
(mHeroImagePath.length() <= maxBoundaryForHeroImageAndPosterAndTrailerPaths)))

                        errorMessage+= "\nHero image path must be between " +
minBoundaryForEverything + " and " + maxBoundaryForHeroImageAndPosterAndTrailerPaths + "
characters long!\n";
                return errorMessage;
        }
```

Figure 62: Movie data validation continued

This function is called in the controller class and the parameters are the textboxes that the user entered the information in. Lengths and correct data types are checked to ensure that the information can be accepted. If the String returned is empty, then the information entered was logically correct, and the Admins can edit or add the movie to the database table. If the String is not empty, an Alert will pop up and let the Admin know what they did wrong, and how to fix the error to proceed.

Appendix N: Code snippet of using the movie data validation function

```
        String movieTitle = aemp.getTitle();
        String movieDescription = aemp.getDescription();
        String movieDuration = aemp.getDuration();
        String movieDirector = aemp.getDirector();
        String movieCast = aemp.getCast();
        LocalDate movieReleaseDate = aemp.getReleaseDate();
        LocalDate moviePlayingFromDate = aemp.getPlayingFromDate();
        LocalDate moviePlayingToDate = aemp.getPlayingToDate();
        String moviePosterPath = posterPath;
        String movieTrailerPath = trailerPath;
        String movieHeroImagePath = heroImagePath;
        String movieNumberOfRows = aemp.getNumberOfRows();
        String errorMessage = "";
        errorMessage = movie.dataValidation(movieTitle, movieDescription, movieDuration,
movieDirector, movieCast, movieReleaseDate, moviePlayingFromDate, moviePlayingToDate,
movieTrailerPath, moviePosterPath, movieHeroImagePath, movieNumberOfRows);
             if(!errorMessage.equals(""))
                  alertDialogBuilder(AlertType.ERROR, "Error", "Error log", errorMessage);
             else
             {
                    //clear any pre-existing data
                    movie.setTitle(movieTitle);
                    movie.setDescription(movieDescription);
                    movie.setDuration(Integer.parseInt(movieDuration));
                    movie.setGenre(aemp.getGenre());
                    movie.setDirector(movieDirector);
                    movie.setCast(movieCast);
                    movie.setReleaseDate(movieReleaseDate);
                    movie.setPlayingFromDate(moviePlayingFromDate);
                    movie.setPlayingToDate(moviePlayingToDate);
                    movie.addShowTime(aemp.getFirstShowTime());
                    movie.addShowTime(aemp.getSecondShowTime());
                    movie.addShowTime(aemp.getThirdShowTime());
                    movie.setPosterPath(moviePosterPath);
                    movie.setTrailerPath(movieTrailerPath);
                    movie.setHeroImagePath(movieHeroImagePath);
                    movie.setNumberOfRows(Integer.parseInt(movieNumberOfRows));
                    //will assign the status value according to the dates
                    movie.setStatus();
                    try {
                           //get the returned ID from the database new row and set it
                           movie.setID(database.addMovie(movie));
                           alertDialogBuilderSuccess(AlertType.INFORMATION, "Movie added
successfully", "Success", "The movie has been added successfully to the " + tempStatusTitle
+ " section");
                    } catch (SQLException e1) {
                                  alertDialogBuilder(AlertType.ERROR, "Error", "Error
adding movie to database", "An unexpected error has occurred. Please contact support");
                           }
             }
```

Figure 63: Code snippet of using the movie data validation function

This code snippet represents how the data validation functions is used in the controller. The code here is not fully copy/pasted, but gives a sense of how a movie is added to the database after validating the data.

Appendix O: JavaFX Alert reusable custom functions

```
	private void alertDialogBuilder(AlertType type, String title, String header, String
content)
	{
		Alert alert = new Alert(type);
		//set the window icon of the alert
		Stage alertStage = (Stage) alert.getDialogPane().getScene().getWindow();
		alertStage.getIcons().add(new Image("/view/sprites/appicon.jpg"));
		alert.setTitle(title);
		alert.setHeaderText(header);
		//label so we can set the text alignment
		Label contentLabelJustify = new Label(content);
		contentLabelJustify.setWrapText(true);
		contentLabelJustify.setTextAlignment(TextAlignment.JUSTIFY);
		alert.setContentText(contentLabelJustify.getText());
		alert.showAndWait();
	}
```

Figure 64: JavaFX error alert function

```
	private void alertDialogBuilderSuccess(AlertType type, String title, String header,
String content)
	{
		Alert alertSuccess = new Alert(type);
		alertSuccess.setTitle(title);
		alertSuccess.setHeaderText(header);
		alertSuccess.setContentText(content);
		//set the window icon of the alert
		Stage alertStage = (Stage)
alertSuccess.getDialogPane().getScene().getWindow();
		alertStage.getIcons().add(new Image("/view/sprites/appicon.jpg"));
		//add a custom success tick icon and set it
		Image icon = new Image(new
File("src/view/sprites/success.png").toURI().toString());
		ImageView success = new ImageView();
		success.setImage(icon);
		success.setFitHeight(60);
		success.setFitWidth(48);
		alertSuccess.getDialogPane().setGraphic(success);
		alertSuccess.showAndWait();
	}
```

Figure 65: JavaFX success alert function

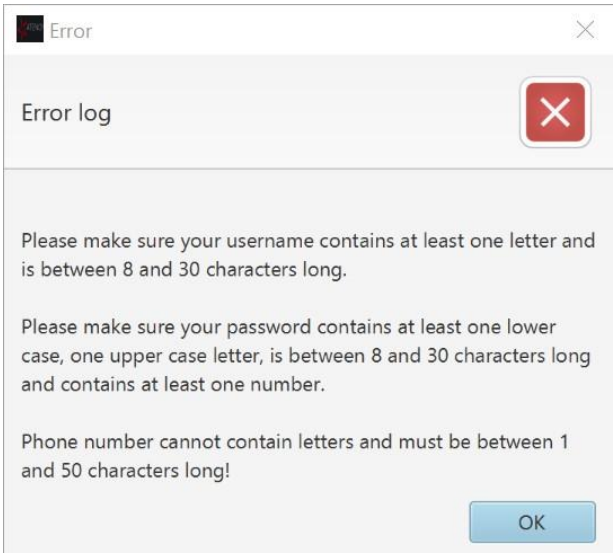

Figure 66: Example of an error log alert

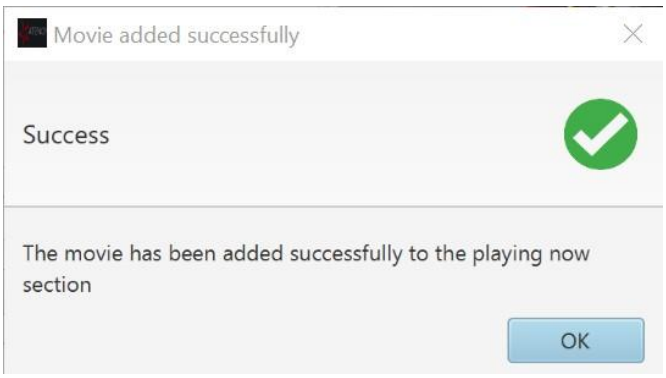

Figure 67: Example of a successful action alert

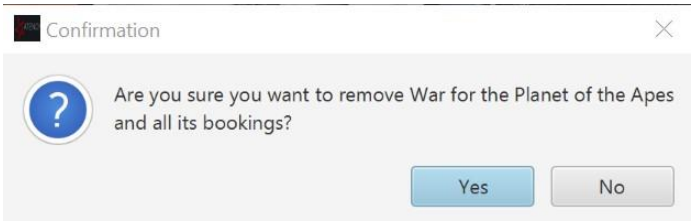

Figure 68: Example of a confirmation alert

Appendix P: Web-based system alert example

```
<asp:Panel class="content" runat="server">
    <asp:Label runat="server" Text="Input email: "></asp:Label>
    <input maxlength="50" type="email" id="txtEmailAddress" name="txtEmailAddress"
        runat="server" required="required" autocomplete="off" />
</asp:Panel>
```

Figure 69: ASP.NET HTML5 input tag

The *type* attribute is used to indicate that the text field must be in email format as per the figure below, and the required attribute is used to ensure that the field cannot be left empty. The green left border illustrates that the information is valid, and the red border demonstrates that an error exists. If the user tries to submit the form with invalid data, the error message will pop up and the user will not be able to proceed until the error is fixed. If it is not a format error (such as an already existing username), then a custom message is presented in red to alert the user that they need to change their username, as it is already taken.

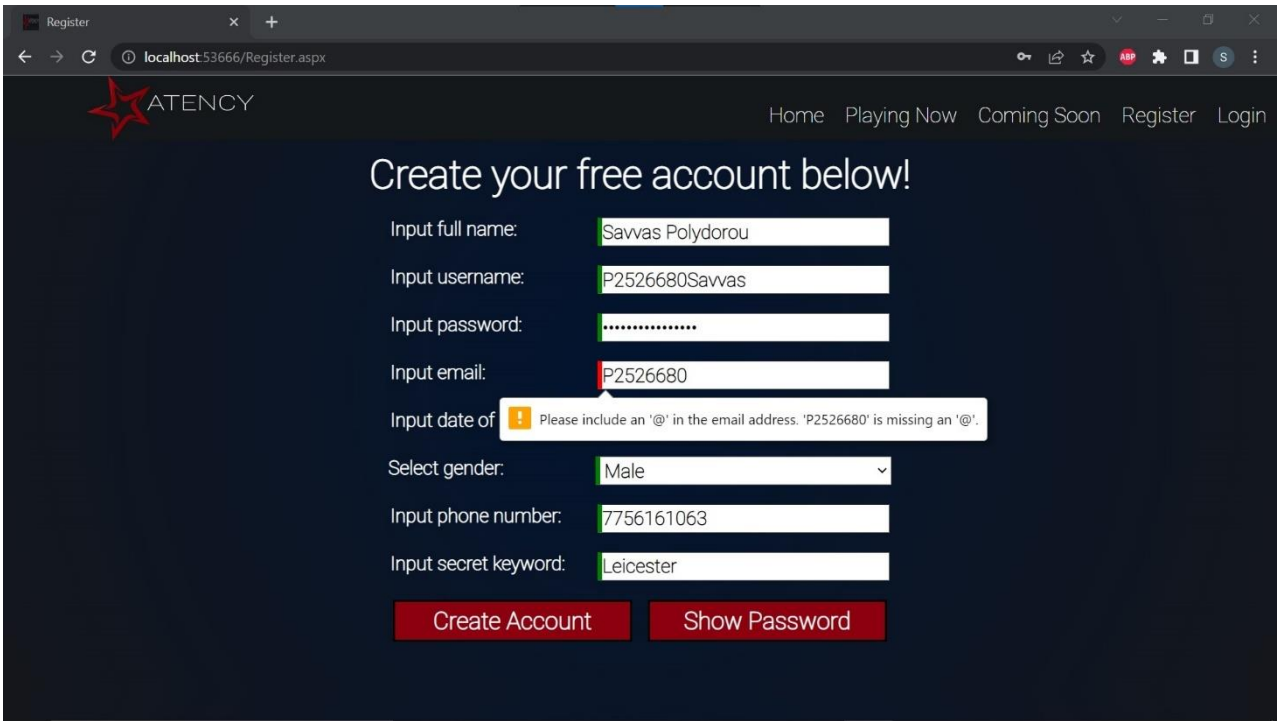

Figure 70: HTML5 built-in error message alert

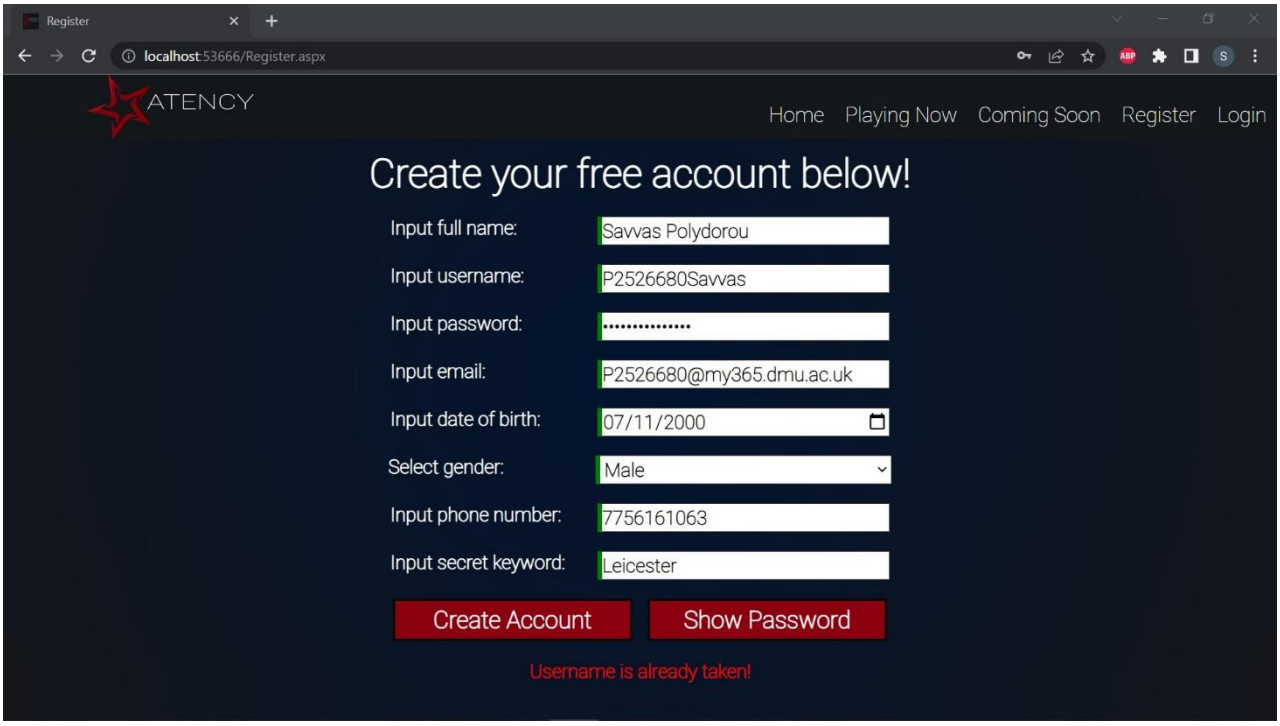

Figure 71: Custom error message

Appendix Q: Email function

```
    protected void btnConfirm_Click(object sender, EventArgs e)
    {
        string email = "cinematicketbookingsystemfyp@gmail.com";
        string password = "*Account password*";
        string recipient = emailaddress;
        SmtpClient smtpClient = new SmtpClient("smtp.gmail.com")
        {
            Port = 587,
            Credentials = new NetworkCredential(email, password),
            EnableSsl = true,

        };
        var mailMessage = new MailMessage
        {
            From = new MailAddress(email, "Atency Cinemas"),
            Subject = "Your booking confirmation for " + movie.Title,
            IsBodyHtml = true,
        };
        mailMessage.To.Add(recipient);
        booking.MovieID = movie.ID;
        booking.CustomerID = customer.ID;
        booking.Date = DateTime.Parse(date);
        booking.ShowTime = showtime;
        booking.ConfirmationEmail = emailaddress;
        for (int i = 0; i < selectedSeats.Count; i++)
        {
            booking.TicketHolderName = userFullNames[i];
            booking.Seat = selectedSeats[i];
            booking.addBooking(booking);
            if(bookingIDS.Length == 0)
            bookingIDS = booking.ID.ToString();
            else
                bookingIDS += ", " + booking.ID.ToString();
            customer.Bookings.Add(booking);
        }
        mailMessage.AlternateViews.Add(Mail_Body());
        smtpClient.Send(mailMessage);
        Response.Redirect("BookingConfirmed.aspx");
    }

    private AlternateView Mail_Body()
    {
        string namesWithSeats = "";
        for (int i = 0; i < selectedSeats.Count; i++)
        {
            namesWithSeats += "<span>Ticket holder name for seat " + selectedSeats[i]
                + ": " + userFullNames[i] + "</span><br>";
        }

        string bookingIDEmail = "";
        if (!bookingIDS.Contains(','))
            bookingIDEmail = "<span>Booking ID: " + bookingIDS + "</span><br>";
        else
            bookingIDEmail = "<span>Booking IDs: " + bookingIDS + "</span><br>";
        string path = Server.MapPath(@"posters/" + posterPath.Name);
        LinkedResource Img = new LinkedResource(path, MediaTypeNames.Image.Jpeg);
        Img.ContentId = "MyImage";
        string str = "<img src = cid:MyImage id = 'img'/>"
                    + "<h2>" + movie.Title + "</h2>"
                    + bookingIDEmail
                    + "<span>Email sent to: " + emailaddress + "</span><br>"
                    + "<span>Booking date: " + date + "</span><br>"
                    + "<span>Show time: " + showtime + "</span><br>"
                    + namesWithSeats
                    + "<br><br>"
                    + "<div style='text-align: justify;'>"
                    + "<span>This automated email was sent as part of the project " +
                    "Cinema Ticket Booking System developed by Savvas Polydorou " +
                    "(P2526680@my365.dmu.ac.uk) for his final year project. Not to be taken
seriously.</span><br>"
                    + "</div>"
                    + "<br>"
                    + "<div style='text-align: center;'>"
                    + "<span style='text-align: center'>&copy; Savvas Polydorou 2022</span><br>"
                    + "</div>";
        AlternateView AV =
        AlternateView.CreateAlternateViewFromString(str, null, MediaTypeNames.Text.Html);
        AV.LinkedResources.Add(Img);
        return AV;
    }
```

Figure 72: Email function

Appendix R: Full name test code snippet in Java

```
private String userFullname = "Savvas Polydorou";
private final int minBoundaryFullnameEmailPhoneNoSecretKeyword = 1;
private final int maxBoundaryFullnameEmailPhoneNoSecretKeyword = 50;

	@Test
	public void FullnamePropertyOK()
	{
		model = new User();
		String testData = userFullname;
		model.setName(testData);
		assertEquals(model.getName(), testData);
	}

	@Test
	public void FullNameMinBoundaryMinusOne()
	{
		model = new User();
		String error = "";
		String testData = "";
		error = model.dataValidation(testData, userUsername, userPassword, userDoB,
userEmailAddress, userPhoneNumber, userSecretKeyword, userProfilePicturePath);
		//this should produce an error message
		assertNotEquals(error,  "");
	}

	@Test
	public void FullNameMinBoundary()
	{
		model = new User();
		String error = "";
		String testData = "S";
		error = model.dataValidation(testData, userUsername, userPassword, userDoB,
userEmailAddress, userPhoneNumber, userSecretKeyword, userProfilePicturePath);
		//this should not produce an error message
		assertEquals(error,  "");
	}

	@Test
	public void FullNameMaxBoundary()
	{
		model = new User();
		String error = "";
		String testData = "";
		for(int i = 0; i < maxBoundaryFullnameEmailPhoneNoSecretKeyword; i++)
			testData+= "a";
		error = model.dataValidation(testData, userUsername, userPassword, userDoB,
userEmailAddress, userPhoneNumber, userSecretKeyword, userProfilePicturePath);
		//this should not produce an error message
		assertEquals(error,  "");
	}

	@Test
	public void FullNameMaxBoundaryPlusOne()
	{
		model = new User();
		String error = "";
		String testData = "";
		for(int i = 0; i < maxBoundaryFullnameEmailPhoneNoSecretKeyword + 1; i++)
			testData+= "a";
		error = model.dataValidation(testData, userUsername, userPassword, userDoB,
userEmailAddress, userPhoneNumber, userSecretKeyword, userProfilePicturePath);
		//this should produce an error message
		assertNotEquals(error,  "");
	}
```

Figure 73: Full name test code snippet in Java

Description of item to be tested:

Username testing for Cinema Owners/Admins/Customers. The username validation test can be found when the user registers, needs to reset their password or when a Customer wants to delete their account. **(UC001, UC004, UC006, UC058, UC101, UC106, UC108).** The validation will not be forced in the login screens for security purposes. **Field cannot be empty**. Username must be unique, contain at least one letter and be between 8 and 30 characters. To ensure the username is unique, the string will be randomly generated in the test case function. Data type accepted is varchar(30)/String.

Table 27: Test data for username

Test Type	Test Data	Expected Result
Min (Boundary)	8 characters savvaspo	Pass
Max (Boundary)	30 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Pass
Invalid data type	Integer, float, double etc. (Numbers only in general)	Fail
Other tests	*<<empty>>* *<<an already existing username>>* savvasp 12345678 Anything over 30 characters	Fail

Description of item to be tested:

Password testing for Cinema Owners/Admins/Customers. The password validation test case can be found when the user registers, edits their details or needs to reset their password, and when a Customer wants to delete their account **(UC001, UC003, UC004, UC006, UC051, UC058, UC101, UC105, UC106)**. It will not be tested in the login screens for security purposes. **Field cannot be empty**. Password must contain at least one uppercase, one lowercase letter, one number and be between 8 and 30 characters. Data type accepted is varchar(30)/String.

Table 28: Test data for password

Test Type	Test Data	Expected Result
Min (Boundary)	8 characters Savvasp1	Pass
Max (Boundary)	30 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaA1	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* savvasp1 ABCabcab ABCA1234 abca1234 12345678 1234567a Anything less than 8 characters Anything over 30 characters	Fail

Description of item to be tested:

Email address testing for Cinema Owners/Admins/Customers. The email address validation test case can be found when the user registers and edits their details, and when booking a ticket **(UC001, UC003, UC004, UC051, UC101, UC104, UC105)**. **Field cannot be empty**. Email address must be between 3 and 50 characters. Data type accepted is varchar(50)/String.

Table 29: Test data for email address

Test Type	Test Data	Expected Result
Min (Boundary)	3 characters a@a	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@a	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* An email address without characters before and after the @ symbol Anything over 50 characters	Fail

Description of item to be tested:

Phone number testing for Cinema Owners/Admins/Customers. The phone number validation test case can be found when the user registers and edits their details **(UC001, UC003, UC004, UC051, UC101, UC105)**. **Field cannot be empty**. Phone number must be between 4 and 50 characters. Data type accepted is varchar(50)/String.

Table 30: Test data for phone number

Test Type	Test Data	Expected Result
Min (Boundary)	4 characters 1234	Pass
Max (Boundary)	50 characters 12345678901234567890123456789012345678901234567890	Pass
Invalid data type	Anything other than numbers	Fail
Other tests	*<<empty>>* Anything less than 4 characters Anything over 50 characters	Fail

Appendix S: Test case descriptions part 3

Description of item to be tested:

Secret keyword testing for Cinema Owners/Admins/Customers. The secret keyword validation test case can be found when the user registers and edits their details, and when they want to reset their password **(UC001, UC003, UC004, UC006, UC051, UC058, UC101, UC108)**. **Field cannot be empty**. Secret keyword must be between 4 and 50 characters. Data type accepted is varchar(50)/String.

Table 31: Test data for secret keyword

Test Type	Test Data	Expected Result
Min (Boundary)	4 characters aaaa	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything less than 4 characters Anything over 50 characters	Fail

Description of item to be tested:

Profile picture path testing for Cinema Owners/Admins. The profile picture path validation test case can be found when the user logs in and uploads a profile picture. Profile picture paths can be empty but must be less than 500 characters. Data type accepted is varchar(500)/String.

Table 32: Test data for profile picture path

Test Type	Test Data	Expected Result
Min (Boundary)	*<<empty>>*	Pass
Max (Boundary)	500 characters *<<local folder path>>*	Pass
Invalid data type	N/A	N/A
Other tests	Anything over 500 characters	Fail

Appendix S: Test case descriptions part 4

Description of item to be tested:

Title testing for movies. The title validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Title must be between 1 and 50 characters. Data type accepted is varchar(50)/String.

Table 33: Test data for movie title

Test Type	Test Data	Expected Result
Min (Boundary)	1 character a	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything over 50 characters	Fail

Description of item to be tested:

Description testing for movies. The description validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Description must be between 1 and 1000 characters. Data type accepted is varchar(1000)/String.

Table 34: Test data for movie description

Test Type	Test Data	Expected Result
Min (Boundary)	1 character a	Pass
Max (Boundary)	1000 characters *<<A description with a maximum of 1000 characters>>*	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything over 1000 characters	Fail

Appendix S: Test case descriptions part 5

Description of item to be tested:

Duration testing for movies. The duration validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Duration must be a positive integer below Integer.*MAX_VALUE*. Data type accepted is int.

Table 35: Test data for movie duration

Test Type	Test Data	Expected Result
Min (Boundary)	1	Pass
Max (Boundary)	2147483647	Pass
Invalid data type	Anything other than an integer	Fail
Other tests	*<<empty>>* 0 Negative number A number greater than 2147483647	Fail

Description of item to be tested:

Director testing for movies. The director validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Director must be between 1 and 50 characters. Data type accepted is varchar(50)/String.

Table 36: Test data for movie director

Test Type	Test Data	Expected Result
Min (Boundary)	1 character a	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything over 50 characters	Fail

Description of item to be tested:

Cast testing for movies. The cast validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Cast must be between 1 and 100 characters. Data type accepted is varchar(100)/String.

Table 37: Test data for movie cast

Test Type	Test Data	Expected Result
Min (Boundary)	1 character a	Pass
Max (Boundary)	100 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything over 100 characters	Fail

Description of item to be tested:

Showtimes testing for movies. The showtimes validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Showtimes must be between 1 and 50 characters. Data type accepted is varchar(50)/String.

Table 38: Test data for movie showtimes

Test Type	Test Data	Expected Result
Min (Boundary)	1 character a	Pass
Max (Boundary)	50 characters aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaa	Pass
Invalid data type	N/A	N/A
Other tests	*<<empty>>* Anything over 50 characters	Fail

Description of item to be tested:

Trailer, poster, and hero image path testing for movies. The trailer, poster, and hero image path validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Fields cannot be empty**. Trailer, poster, and hero image paths must be between 1 and 500 characters. Data type accepted is varchar(500)/String.

Table 39: Test data for movie trailer, poster, and hero image path

Test Type	Test Data	Expected Result
Min (Boundary)	1 character	Pass
Max (Boundary)	500 characters <<*local folder path*>>	Pass
Invalid data type	N/A	N/A
Other tests	Anything over 500 characters	Fail

Description of item to be tested:

Number of rows testing for movies. The number of rows validation test case can be found when the Admins add or edit a movie in the system **(UC052, UC053). Field cannot be empty**. Number of rows must be greater than or equal to 5 and less than or equal to 26 (English alphabet letters count). Data type accepted is int.

Table 40: Test data for movie number of rows of seats

Test Type	Test Data	Expected Result
Min (Boundary)	5	Pass
Max (Boundary)	26	Pass
Invalid data type	Anything other than an integer	Fail
Other tests	<<*empty*>> Negative number A number less than 5 A number greater than 26	Fail

Appendix T: Browse playing now movies web-based system

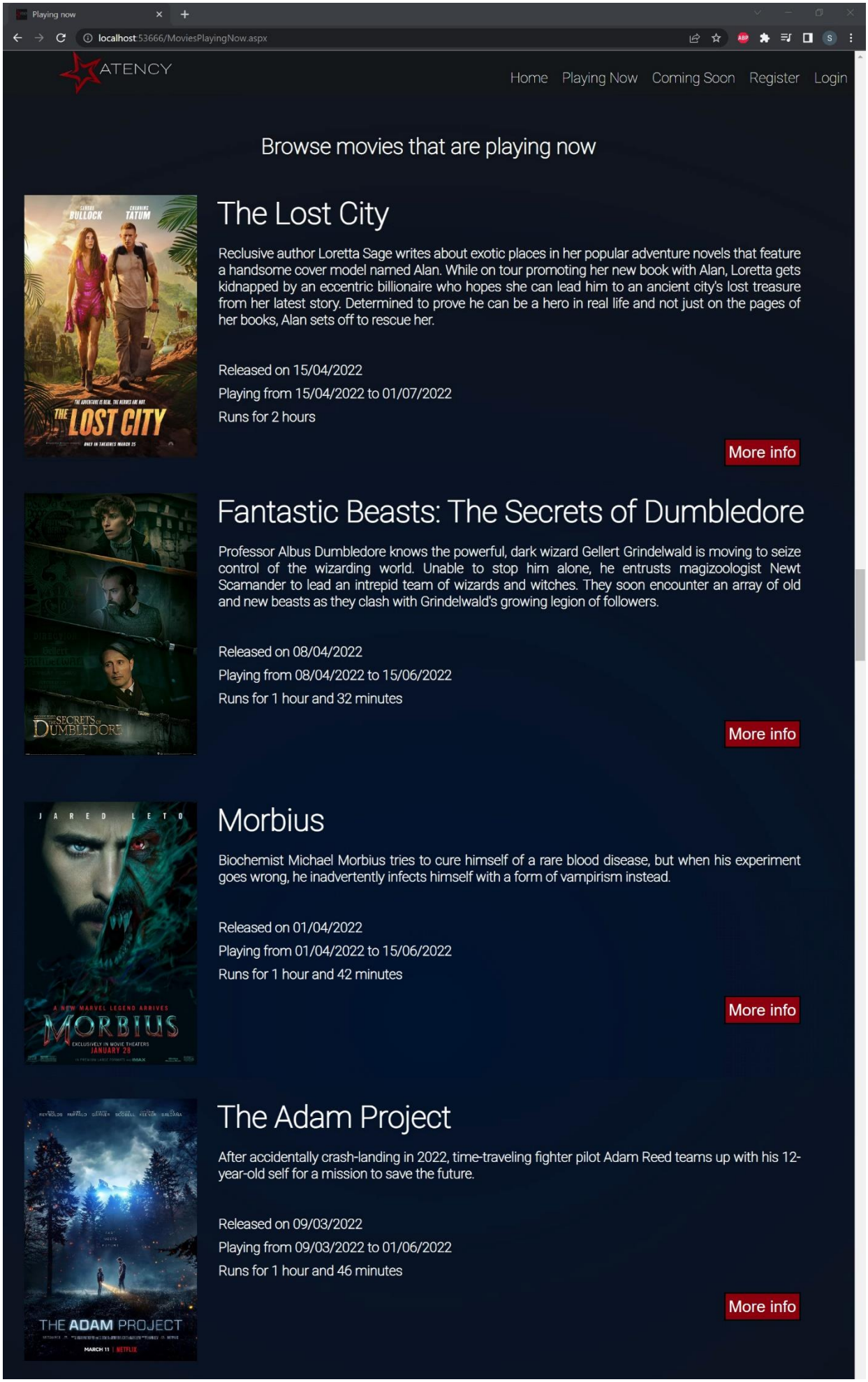

Figure 74: Browse playing now movies web-based system

Appendix U: Browse coming soon movies web-based system

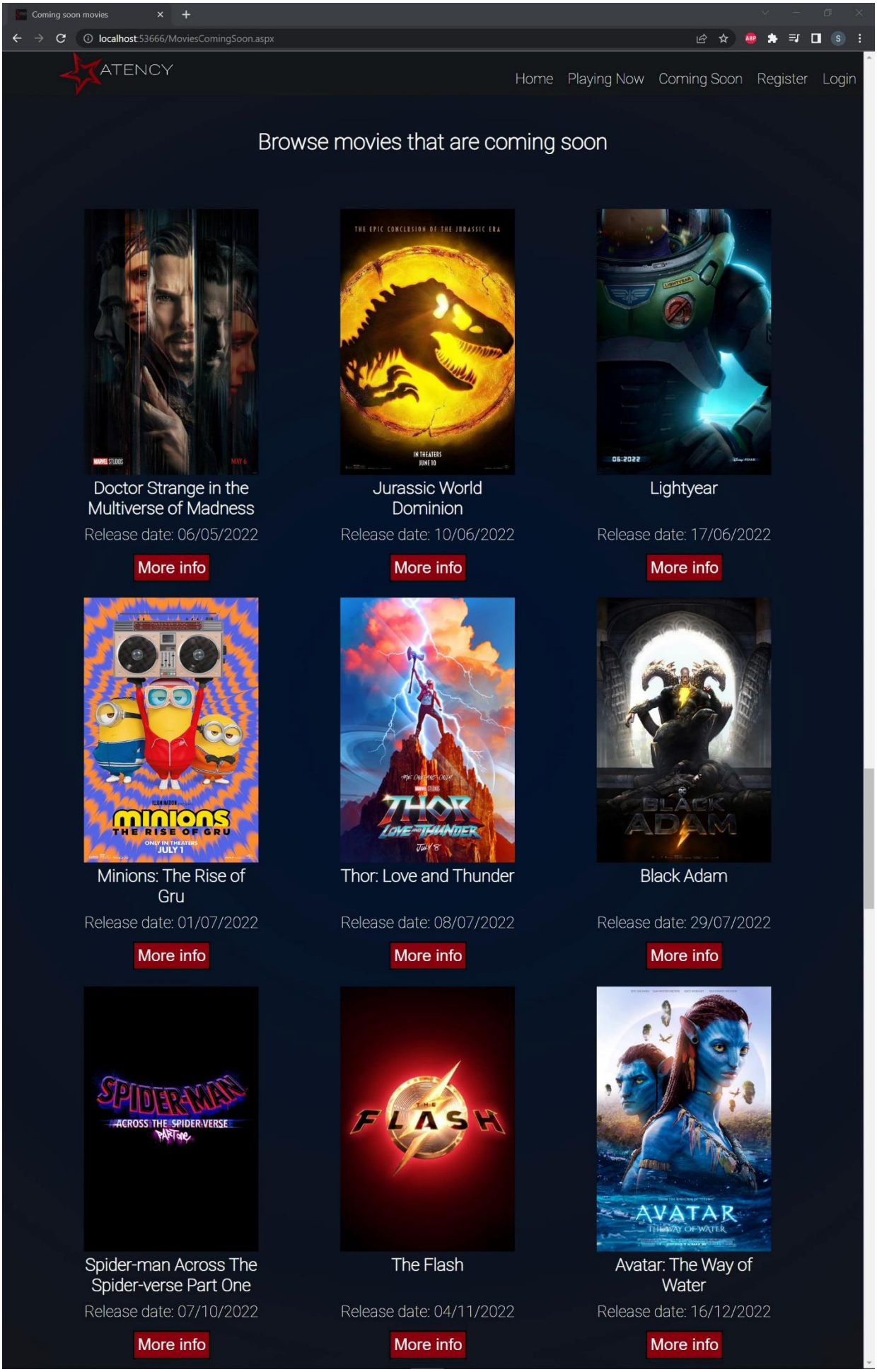

Figure 75: Browse coming soon movies web-based system

Appendix V: Confirm/Cancel booking

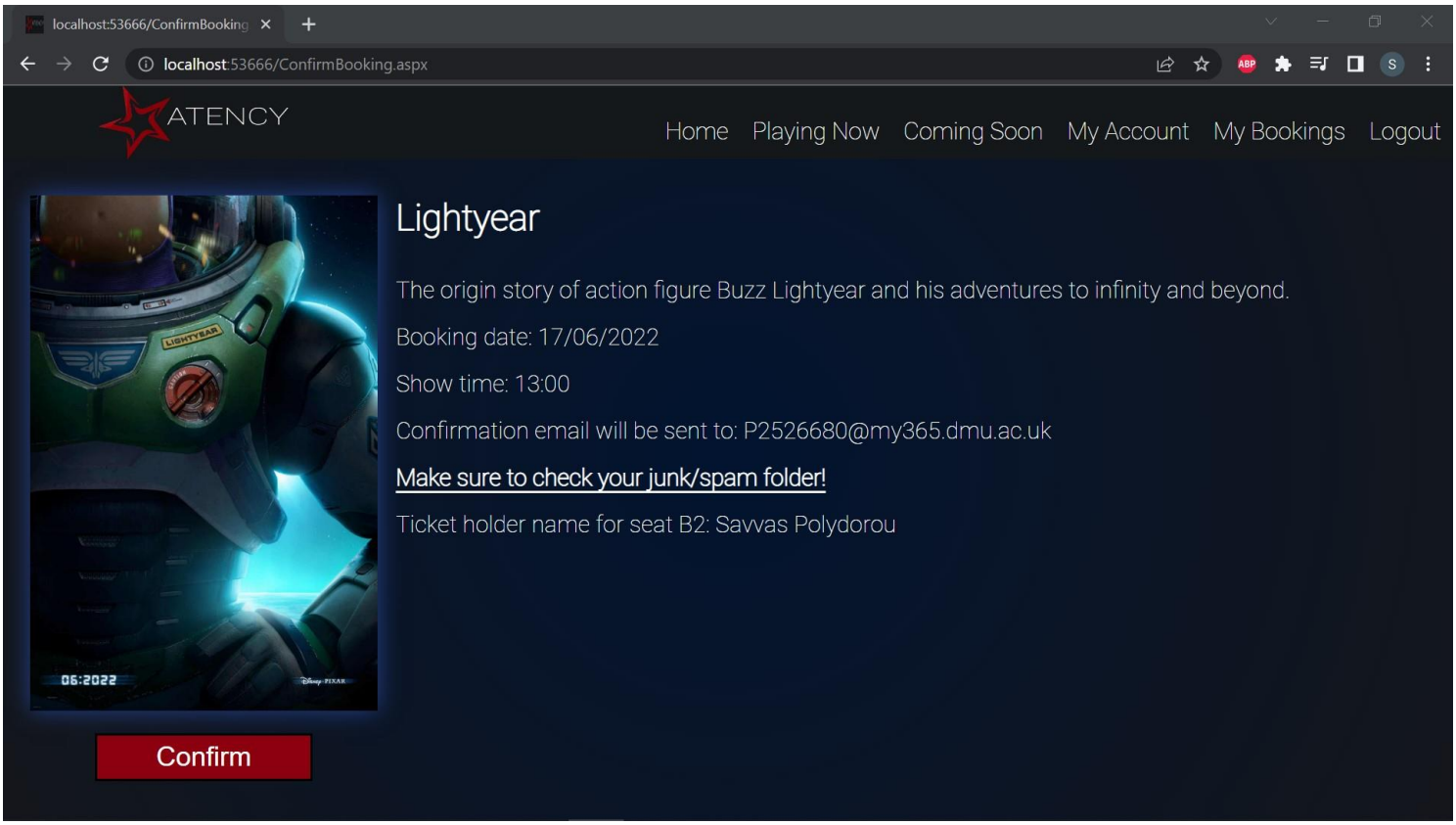

Figure 76: Confirm booking

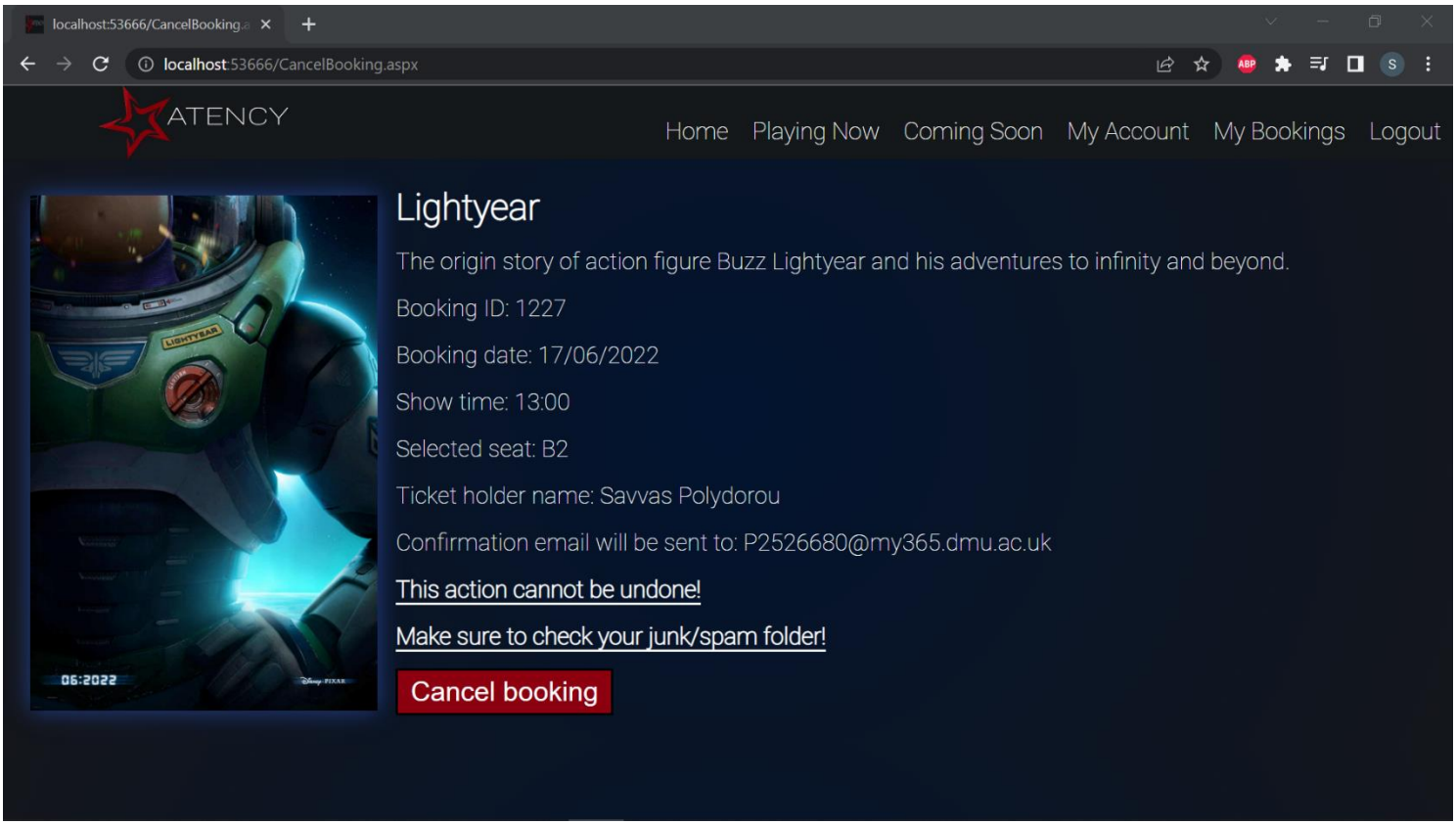

Figure 77: Confirm booking cancellation

Appendix W: UML Diagram

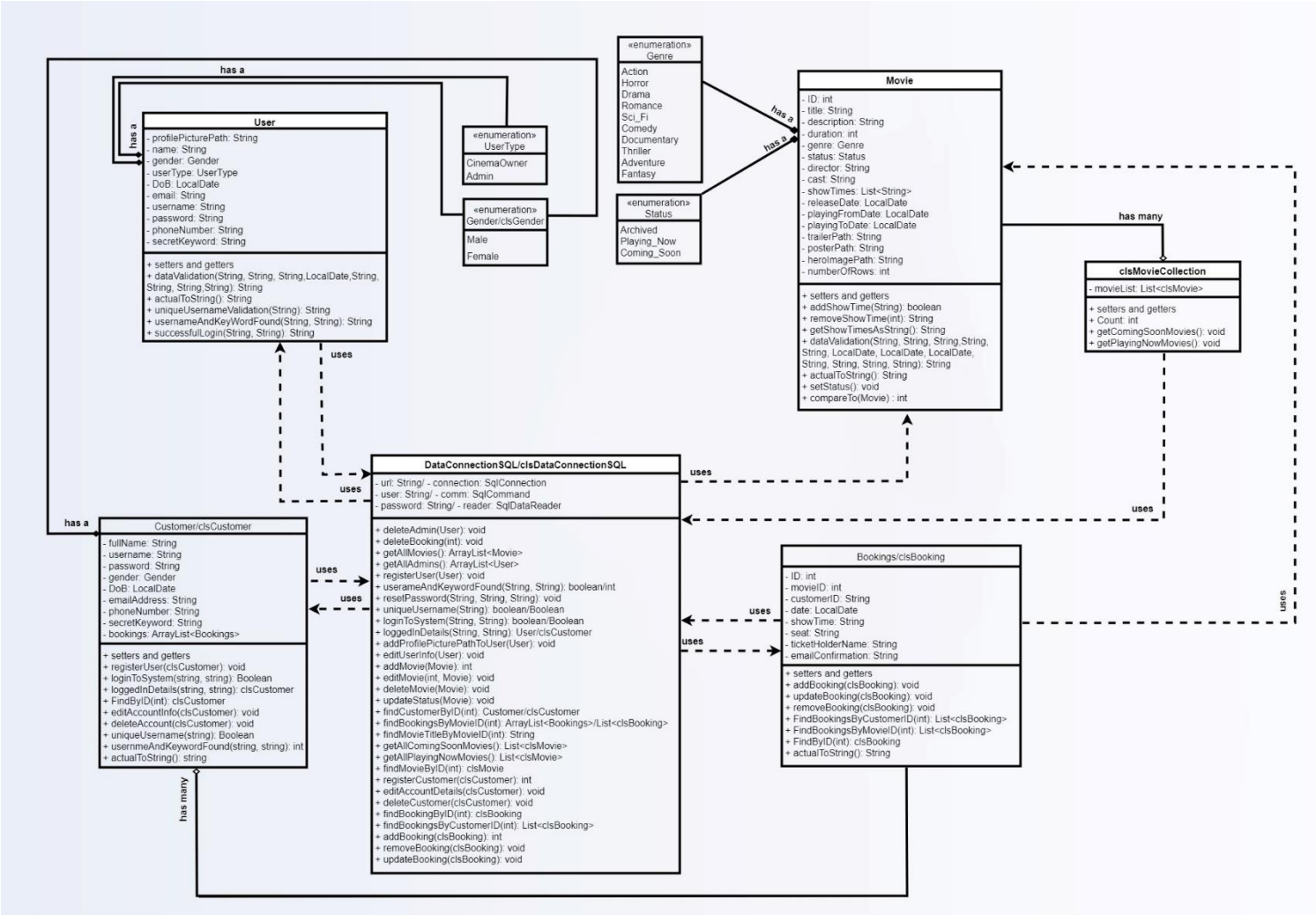

Figure 78: UML Diagram