Advanced Software Development
Individual Report & Test Cases & Diagrams

Ahmed Elsaman(21072727)

Introduction:

In the realm of modern software development, Object-Oriented Programming (OOP) has emerged as a transformative paradigm, revolutionizing the way software is designed, implemented, and maintained. Its powerful principles and concepts have laid the groundwork for advanced software development, enabling developers to create complex, scalable, and maintainable applications that cater to the ever-evolving needs of users and businesses alike.

Object-Oriented Programming is a programming approach that revolves around the concept of "objects," which encapsulate both data and behavior. By modeling real-world entities or abstract concepts as objects, developers can create software systems that closely resemble the natural world, making it easier to comprehend, design, and modify. The core pillars of OOP - encapsulation, inheritance, and polymorphism - offer a robust foundation for constructing efficient and flexible codebases.

Encapsulation, one of the fundamental principles of OOP, promotes data integrity and security by restricting access to an object's internal state and exposing only relevant methods and properties. This ensures that the complexity of an object is hidden from other parts of the program, leading to cleaner code and reducing potential sources of errors.

Inheritance facilitates code reuse and hierarchical relationships, allowing objects to inherit characteristics and behaviors from parent objects. This feature not only reduces redundancy but also enhances the maintainability and extensibility of the codebase, as changes made to parent objects can automatically propagate to their descendants.

Polymorphism, meaning "many forms," empowers objects of different types to be treated interchangeably through a common interface. This promotes flexibility and code readability by abstracting complex implementations and focusing on high-level interactions.

Advanced software development harnesses the power of Object-Oriented Programming to tackle sophisticated challenges and meet the demands of a rapidly evolving digital landscape. Developers utilize design patterns, architectural principles, and software engineering best practices to create robust, scalable, and high-performance applications.

Throughout this report, we will explore the principles of Object-Oriented Programming and delve into advanced software development techniques. We will examine the application of OOP in real-world scenarios, analyzing how it enhances code organization, facilitates collaboration among development teams, and improves software maintainability. Moreover, we will investigate cutting-edge software development methodologies and emerging trends that leverage OOP to build innovative and forward-looking applications.

Implementation:

The implemented Theatre Booking System comprises both Manager and Staff views. The Staff view features four main buttons, namely Create Booking, Fetch Booking, Update Booking, and Cancel Booking. Additionally, the Staff view includes a list of shows fetched from the shows table in the database. The Create Booking button opens a new window displaying attributes from the booking table, with each attribute having a dropdown menu populated with relevant information from the database. For instance, the city attribute's dropdown menu will contain city names fetched from the city table in the database.

The Fetch Booking button retrieves all created bookings from the booking table in the database and displays them in a new window. Meanwhile, the Update Booking button opens a window with a list of created bookings, allowing the selection of a booking to be updated. Within this window, a Confirm Update button confirms the modifications made to the selected booking, updating it in the database. The Cancel Booking button presents a list of created bookings, enabling the selection of a booking for removal from the database.

The Manager view incorporates all the features present in the Staff view and introduces seven additional buttons: Add City, Update City, Remove City, Add Theatre, Update Theatre, Remove Theatre, and Generate Strategic Report. These seven buttons are exclusive to the Manager view and do not appear in the Staff view.

The Add City button opens a new window containing text boxes for each attribute in the city table. After inputting the required information, the Confirm Addition button adds the new city to the database. The Update City button opens a window with text boxes to match the attributes in the city table, allowing for the update of a city's information. The Confirm Update button saves the modified information to the city table in the database. The Remove City button displays a list of cities from the city table in a new window, allowing the selection of a city for complete removal from the database.

Similarly, the Add Theatre button opens a new window with text boxes corresponding to the attributes in the theatre table. The Confirm Addition button adds the newly inputted theatre to the database. The Update Theatre button opens a window with text boxes for updating any theatre's information. The Confirm Update button saves the changes to the theatre table in the database. The Remove Theatre button presents a list of theatres from the theatre table, enabling the removal of a selected theatre from the database.

Lastly, the Generate Strategic Report button provides essential information, including the sum of total profits, the number of bookings cancelled, the count of bookings made for each user or in total, and the movies that were booked the most in each city.

Both the Manager and Staff views enhance the Theatre Booking System's functionality and offer convenient options for managing bookings, cities, and theatres, as well as generating strategic reports for better decision-making.

Conclusion:

Object-Oriented Programming (OOP) has undoubtedly revolutionized modern software development, providing a powerful and flexible paradigm for creating complex and maintainable applications. Throughout this report, we explored the core principles of OOP - encapsulation, inheritance, and polymorphism - which have laid the foundation for constructing efficient and scalable codebases, enabling developers to model real-world entities in a natural and intuitive manner.

The implementation of the Theatre Booking System exemplifies how OOP principles can be harnessed to create intuitive user interfaces and streamlined processes for booking management and theatre administration. The Manager and Staff views showcased how OOP enhances functionality and user experience in different roles, offering powerful tools for managerial oversight and efficient booking management.

Moreover, our investigation into advanced software development demonstrated the importance of design patterns, architectural principles, and software engineering best practices. By leveraging OOP alongside emerging trends and methodologies, developers can create innovative and future-proof solutions that cater to the dynamic needs of users and businesses.

As the landscape of software development continues to evolve, Object-Oriented Programming remains a fundamental pillar in shaping cutting-edge applications. The journey of exploring OOP and its integration with advanced software development techniques has illuminated the significance of adopting robust principles and practices in designing sophisticated and high-performance applications.

In conclusion, the mastery of Object-Oriented Programming and advanced software development is indispensable for developers aspiring to thrive in the ever-changing software engineering landscape. By embracing the power of OOP and continuously refining their skills, developers can contribute to the creation of innovative and impactful software solutions.

1. `CancelBookingWindow` Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	User tries to cancel a valid booking	User selects a valid booking and chooses to cancel	The system cancels the booking and shows a confirmation message	As expected
2	User tries to cancel an invalid booking	User selects an invalid/nonexistent booking	The system shows an error message about the invalid booking	As expected

2. `UpdateBookingWindow` Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	User tries to update a valid booking	User selects a valid booking and chooses to update	The system updates the booking details and shows a confirmation message	As expected
2	User tries to update an invalid booking	User selects an invalid/nonexistent booking	The system shows an error message about the invalid booking	As expected

3. `BookingWindow` Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	User tries to make a new booking	User selects show, date, and number of tickets	The system calculates the total price, shows seat availability, and confirms booking	As expected
2	User tries to make a booking with insufficient seats	User selects show, date, and number of tickets exceeding available seats	The system shows a message about insufficient seats and doesn't confirm the booking	As expected

4. `AddTheatreWindow` Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	Admin tries to add a new theatre	Admin enters valid details of the new theatre	The system adds the new theatre and shows a confirmation message	As expected
2	Admin tries to add a theatre with duplicate name	Admin enters a duplicate theatre name	The system shows an error message about the duplicate theatre name	As expected

5. `AddCityWindow` Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	Admin tries to add a new city	Admin enters the valid name of the new city	The system adds the new city and shows a confirmation message	As expected
2	Admin tries to add a city with duplicate name	Admin enters a duplicate city name	The system shows an error message about the duplicate city name	As expected

1. `ShowBooking` Class Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	User tries to book tickets for a specific show	User selects show, date, and number of tickets	The system calculates the total price, shows seat availability, and confirms booking	As expected
2	User tries to book tickets with an insufficient quantity	User selects show, date, and number of tickets exceeding available seats	The system shows a message about insufficient available seats and doesn't confirm the booking	As expected
3	User tries to book tickets for a show that doesn't exist	User selects an invalid show	The system shows an error message and doesn't confirm the booking	TBD

Test Case #	Description	Test dataset/Input	Expected output	Actual output
4	User tries to book tickets for a date that is not available	User selects show and a date not available	The system shows an error message and doesn't confirm the booking	As expected

2. User Class Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	New user registration with valid details	Valid details of the user for registration	The system registers the user and shows a confirmation message	As expected
2	New user registration with duplicate username	User details with a duplicate username	The system shows an error message about username duplication	As expected
3	User tries to update profile with valid details	User selects "Update Profile" and enters new details	The system updates user profile details and shows a confirmation message	As expected
4	User tries to delete their account	User selects "Delete Account"	The system deletes the user's account and logs the user out	As expected

1. `ShowList` Class Test Cases

Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	Manager tries to add a theatre	Manager logs in, selects "Add Theatre" radio button, enters theatre details, and clicks "Execute"	Theatre details are added to the database and appear in the list	As expected
2	Manager tries to remove a theatre	Manager logs in, selects "Remove Theatre" radio button, enters theatre ID, and clicks "Execute"	Theatre is removed from the database and no longer appears in the list	As expected
3	Manager tries to update a theatre	Manager logs in, selects "Update Theatre" radio button, enters theatre ID and new details, and clicks "Execute"	Theatre details are updated in the database and reflect in the list	As expected
4	Manager tries to add a city	Manager logs in, selects "Add City" radio button, enters city details, and clicks "Execute"	City details are added to the database and appear in the list	As expected
5	Manager tries to remove a city	Manager logs in, selects "Remove City" radio button, enters city ID, and clicks "Execute"	City is removed from the database and no longer appears in the list	As expected

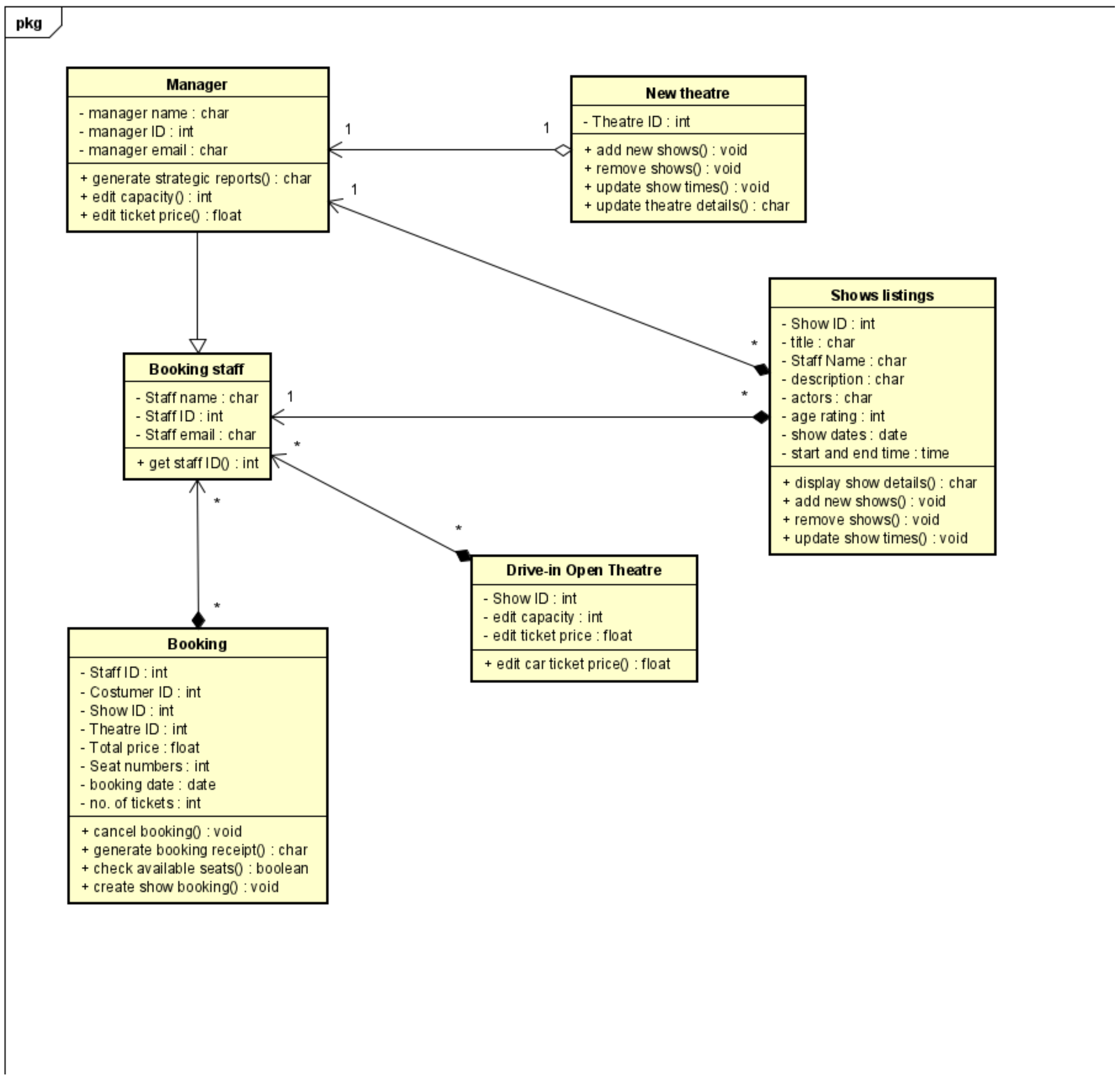
Test Case #	Description	Test dataset/Input	Expected output	Actual output
6	Manager tries to update a city	Manager logs in, selects "Update City" radio button, enters city ID and new details, and clicks "Execute"	City details are updated in the database and reflect in the list	As expected
7	Manager generates a strategic report	Manager logs in and clicks on "Generate Strategic Report"	A report window opens up with all the relevant statistics about bookings and profits	As expected

2. `LoginWindow` Class Test Cases

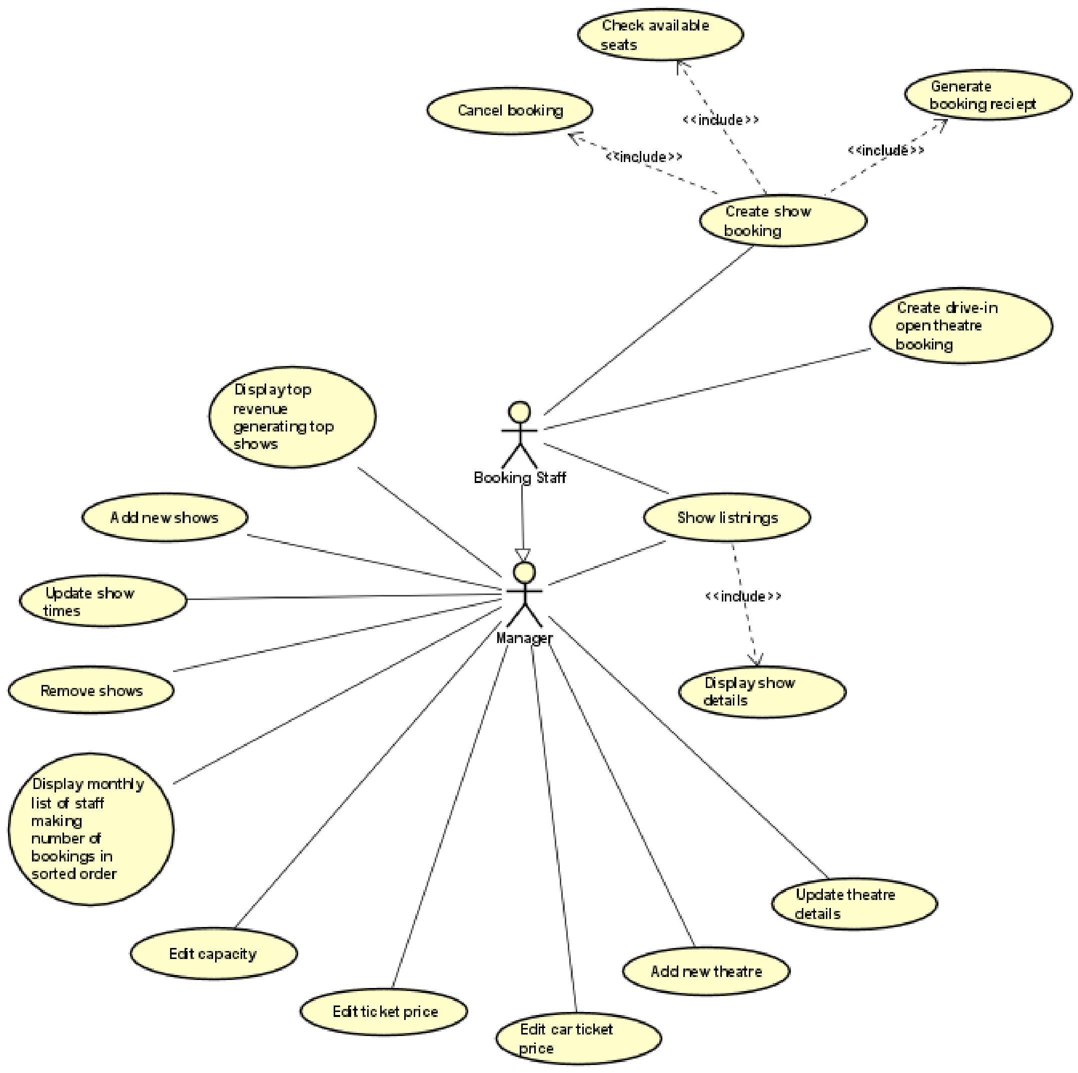
Test Case #	Description	Test dataset/Input	Expected output	Actual output
1	User tries to log in with correct credentials	Input correct username and password	The system logs the user in and shows the main screen	As expected
2	User tries to log in with incorrect credentials	Input incorrect username and password	The system prevents the login and shows an error message	As expected

Test Case #	Description	Test dataset/Input	Expected output	Actual output
3	User tries to register with valid details	User clicks on "Register", inputs valid details, and submits	The system creates a new user account and logs the user in	As expected
4	User tries to register with an already used username	User clicks on "Register", inputs details with an already used username, and submits	The system prevents registration and shows an error message about the username being used	As expected

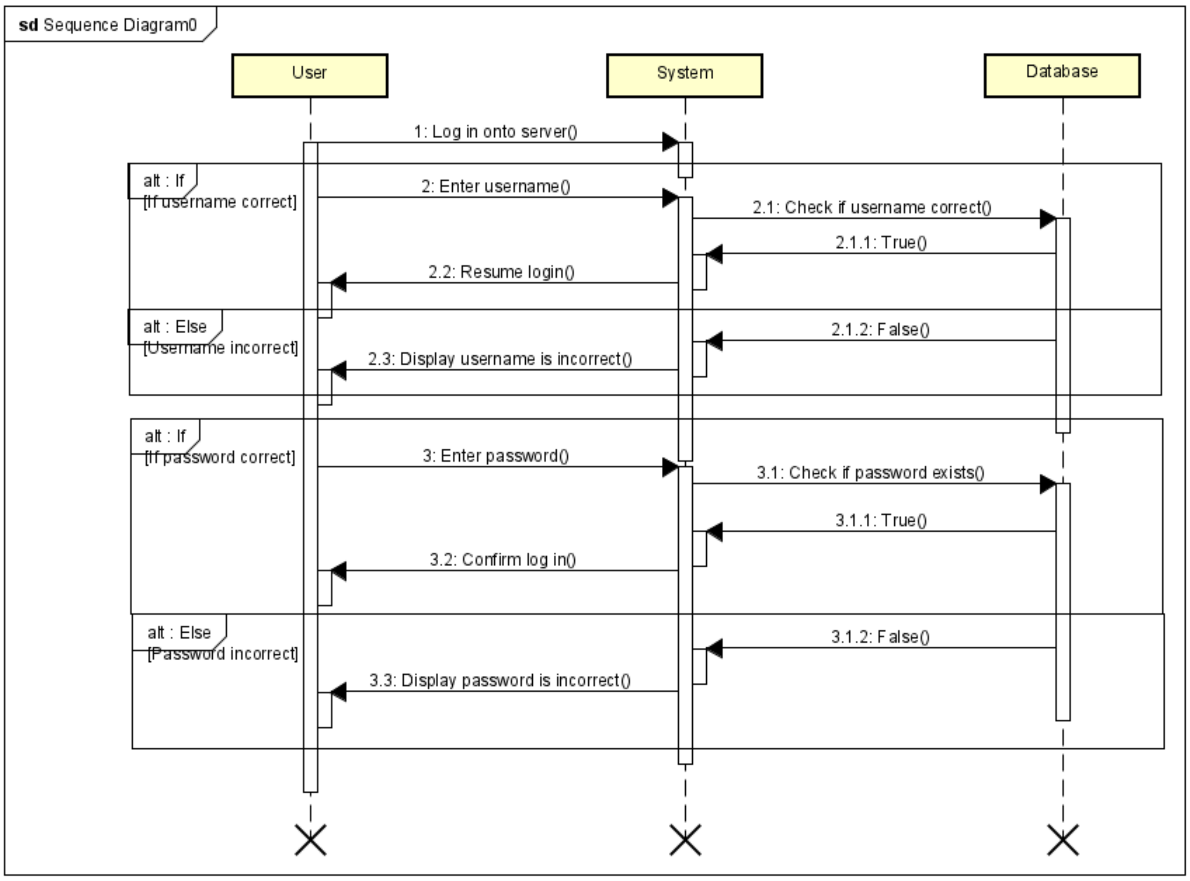
Class Diagram:

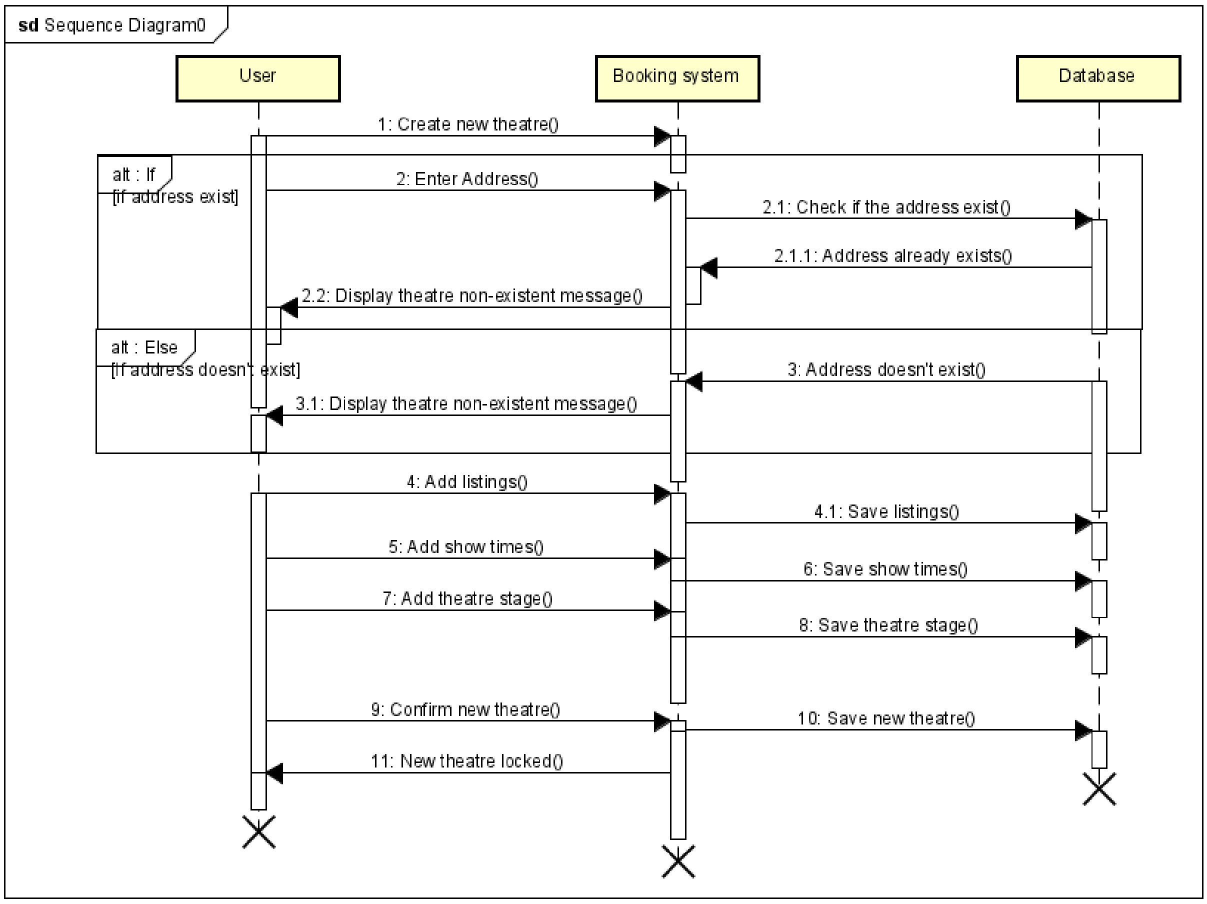


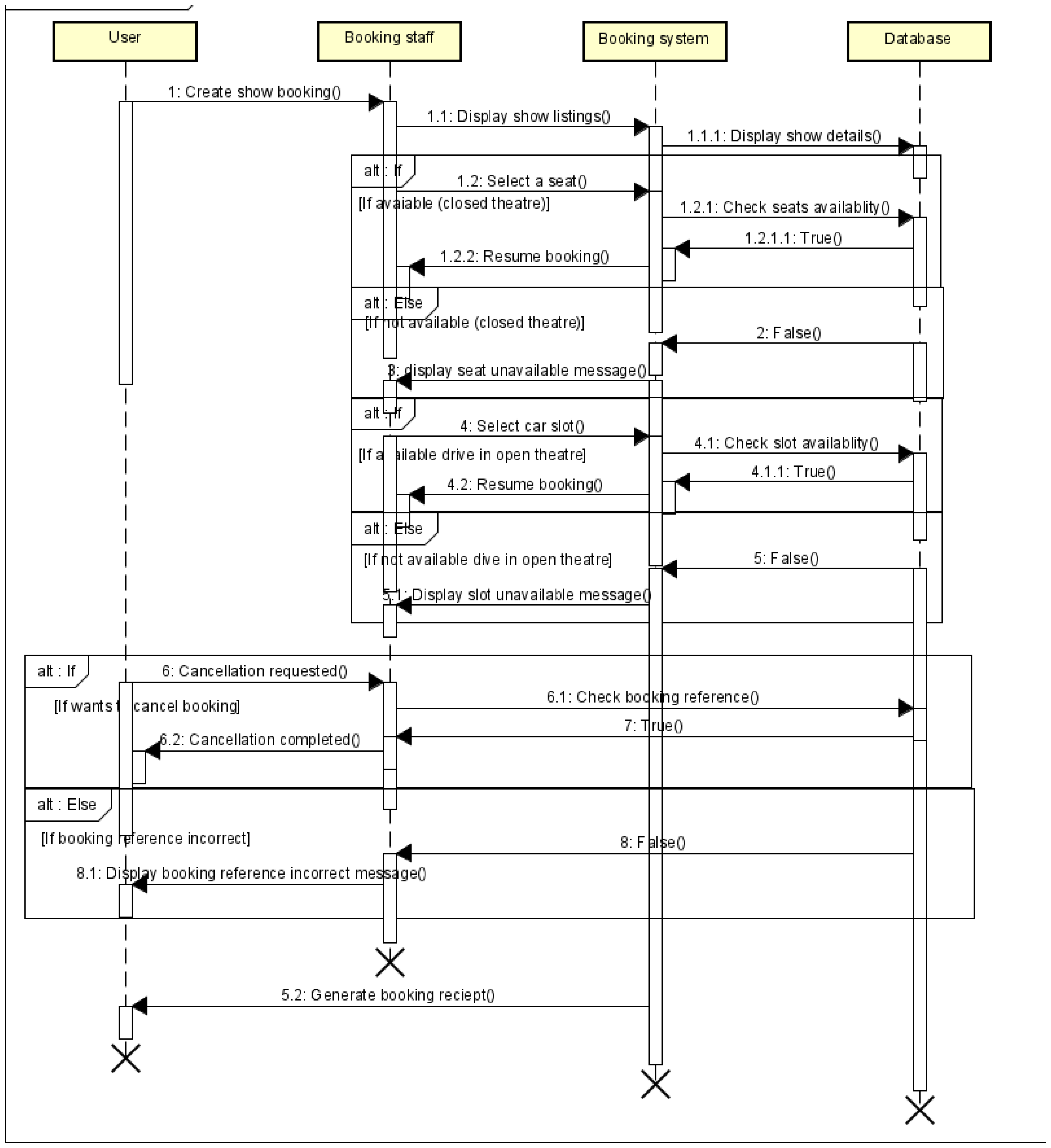
UseCase Diagram:



Sequence Diagrams:






User
Booking staff
Booking system
Database
1: Create show booking()
1.1: Display show listings()
1.1.1: Display show details()
alt : If
1.2: Select a seat()
[If avaiable (closed theatre)]
1.2.1: Check seats availablity()
1.2.1.1: True()
1.2.2: Resume booking()
alt : Else
[If not available (closed theatre)]
2: False()
3: display seat unavailable message()
alt : If
4: Select car slot()
[If available drive in open theatre]
4.1: Check slot availablity()
4.1.1: True()
4.2: Resume booking()
alt : Else
[If not available dive in open theatre]
5: False()
5.1: Display slot unavailable message()
alt : If
6: Cancellation requested()
[If wants to cancel booking]
6.1: Check booking reference()
7: True()
6.2: Cancellation completed()
alt : Else
[If booking reference incorrect]
8: False()
8.1: Display booking reference incorrect message()
5.2: Generate booking reciept()