COMP1618 - Software Tools and Techniques

2024-25

London Musical Ticket System Report

Saiyukta Maheshkumar Parmar

001370775

MSc Computing and Information Systems

Table of Contents

1. Introduction 1
2. Design and Development 2
   2.1 UML Class Diagram 2
   2.2 GUI Design 3
   2.3 MySQL Database Integration 4
   2.4 Development Process 4
   2.5 Challenges and Solutions 4
   2.6 Screenshots of Program in Operation 5
3. Testing and Faults 9
   3.1 White Box Testing Overview 9
   Test Case Table 9
   3.2 Faults and Failures Discussion 10
4. Conclusion, further development, and reflections 12
   Further Development: 12
   Reflection: 12
References: 14

Table of Figures

Figure 1: UML for Classes 2
Figure 2: Main Frame Screenshot 5
Figure 3: Musical Selection Screenshot 5
Figure 4: Schedule Display Screenshot 6
Figure 5: Ticket Booking Form Screenshot 6
Figure 6: Receipt Generation Screenshot 7
Figure 7: Receipt.txt 7
Figure 8: Search Functionality 8

1. Introduction

The Musical Ticket Booking System was created to give consumers a simple method to view available musicals, schedules, and purchase tickets for their favorite events. The system incorporates features such a graphical user interface (GUI), timetable display, ticket booking, and receipt generating. Users can choose musicals from a list, view schedules, and book tickets of various types (e.g., adult, senior, student).

The main objectives of this system are to provide a seamless experience for users in booking tickets for a wide range of musical performances and to maintain the ability to manage and update show schedules dynamically. The system's flexibility allows for future improvements, such as incorporating online payments, integrating additional musicals, or adding a more advanced user authentication process.

2. Design and Development

This project was developed with Java, a popular programming language recognized for its portability and object-oriented design, which allowed for the rapid development of the system's functionality. NetBeans IDE was used to create the graphical user interface (GUI), while MySQL was chosen as the database solution for storing musical details and timetables. Text files were employed as an alternate storage method for transaction receipts, allowing for easy saving and loading of user data.

2.1 UML Class Diagram

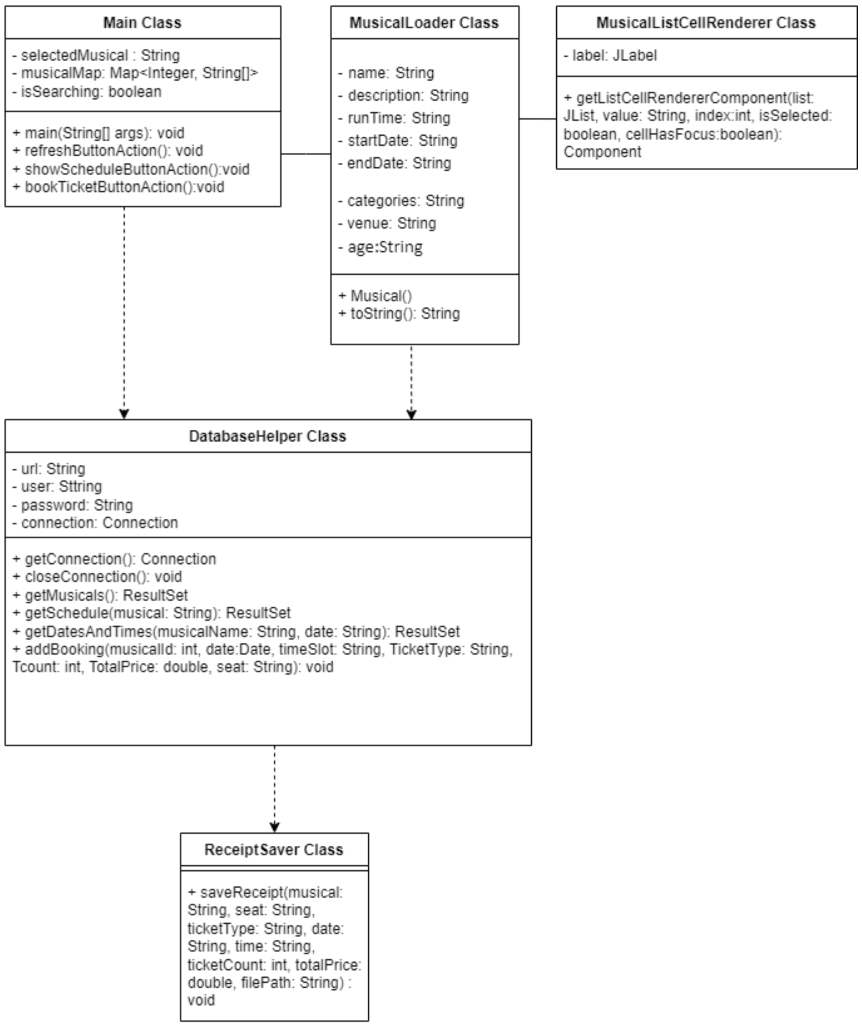

Main Class
- selectedMusical : String
- musicalMap: Map<Integer, String[]>
- isSearching: boolean
+ main(String[] args): void
+ refreshButtonAction(): void
+ showScheduleButtonAction():void
+ bookTicketButtonAction():void
MusicalLoader Class
- name: String
- description: String
- runTime: String
- startDate: String
- endDate: String
- categories: String
- venue: String
- age:String
+ Musical()
+ toString(): String
MusicalListCellRenderer Class
- label: JLabel
+ getListCellRendererComponent(list: JList, value: String, index:int, isSelected: boolean, cellHasFocus:boolean): Component
DatabaseHelper Class
- url: String
- user: Sttring
- password: String
- connection: Connection
+ getConnection(): Connection
+ closeConnection(): void
+ getMusicals(): ResultSet
+ getSchedule(musical: String): ResultSet
+ getDatesAndTimes(musicalName: String, date: String): ResultSet
+ addBooking(musicalId: int, date:Date, timeSlot: String, TicketType: String, Tcount: int, TotalPrice: double, seat: String): void
ReceiptSaver Class
+ saveReceipt(musical: String, seat: String, ticketType: String, date: String, time: String, ticketCount: int, totalPrice: double, filePath: String) : void

Figure 1: UML for Classes

Class Descriptions:

1. **Main Class:**
   Acts as the core controller of the application, coordinating interactions between other classes. It uses the Musical class for retrieving musical details, DatabaseHelper for database operations, and ReceiptSaver for saving booking receipts.
2. **Musical Class:**
   Represents musical details (e.g., name, description, runtime) and is used by the Main class to fetch and display musical information.
3. **DatabaseHelper Class:**
   Manages database operations, including fetching musical data, schedules, and saving bookings. The Main class utilizes its methods for database interactions.
4. **ReceiptSaver Class:**
   Saves booking details to a file. The Main class calls its methods to generate and save receipts.
5. **MusicalListCellRenderer Class:**
   Formats and displays musical details in a user-friendly way within a list. It works indirectly with the Main class via the JList component.

2.2 GUI Design

The GUI was designed using Java Swing to ensure cross-platform compatibility. Key components include:

- Musical Selection Panel:

  A JList component for users to select a musical.

- Schedule Display Panel:

  A dropdown menu (JComboBox) for dates and time slots.

- Booking Form:

  Inputs for ticket type, date, time, seat selection and confirmation of booking.

- Receipt Generation:

  A text area displays booking details and allows the user to print the receipt.

2.3 MySQL Database Integration

The application uses a MySQL database to store and manage data.

Tables include:

- Musicals: Stores musical information such as ID, name, and description.
- Schedule: Stores schedules with musical IDs, dates, and time slots.

2.4 Development Process

The development process involved the following steps:

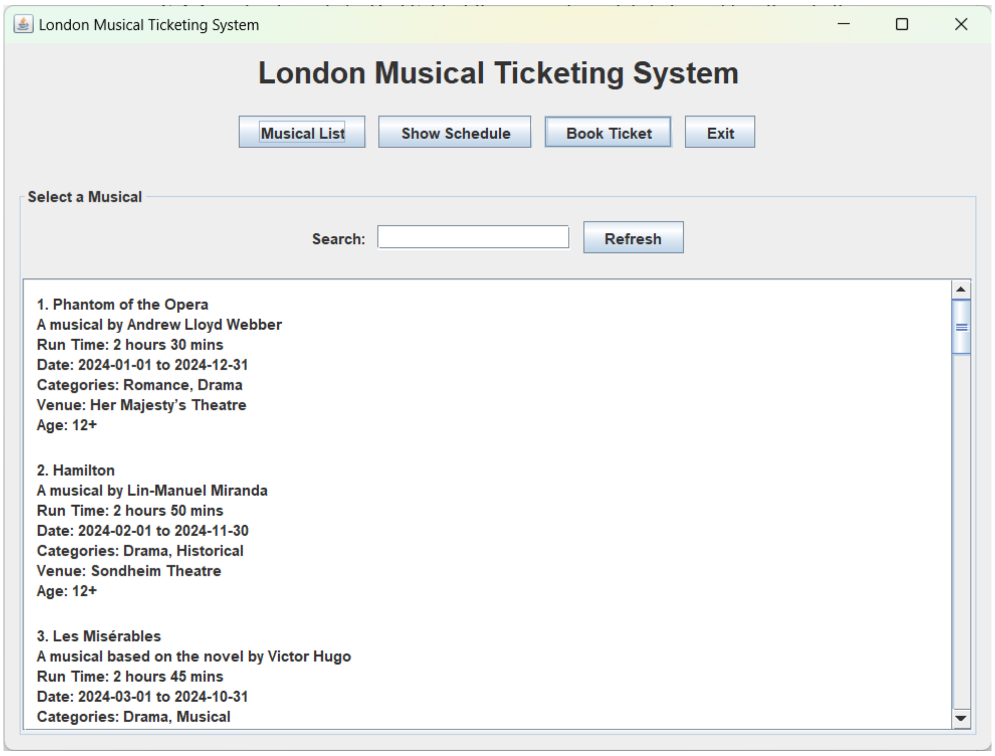
1. **Requirement Analysis**:
   - Understanding the user requirements and defining system functionalities.
2. **GUI Prototyping**:
   - Developing a mockup for the interface and implementing it using Swing components.
3. **Database Design**:
   - Creating MySQL tables and defining relationships between musicals, and schedules.
4. **Core Functionality Implementation**:
   - Writing Java code to fetch data from the database, populate GUI components, and handle user actions.
5. **Testing and Debugging**:
   - Conducting white-box testing to ensure all functionalities work as intended.

2.5 Challenges and Solutions

- **Challenge**: Duplicate entries in date and time dropdowns. **Solution**: Implemented a uniqueness check through a method isItemInComboBox()
- **Challenge**: Handling null or invalid selections. **Solution**: Used validation with appropriate error messages to guide the user.

2.6 Screenshots of Program in Operation

1. Main Frame


London Musical Ticketing System

London Musical Ticketing System

Select a Musical

Search:

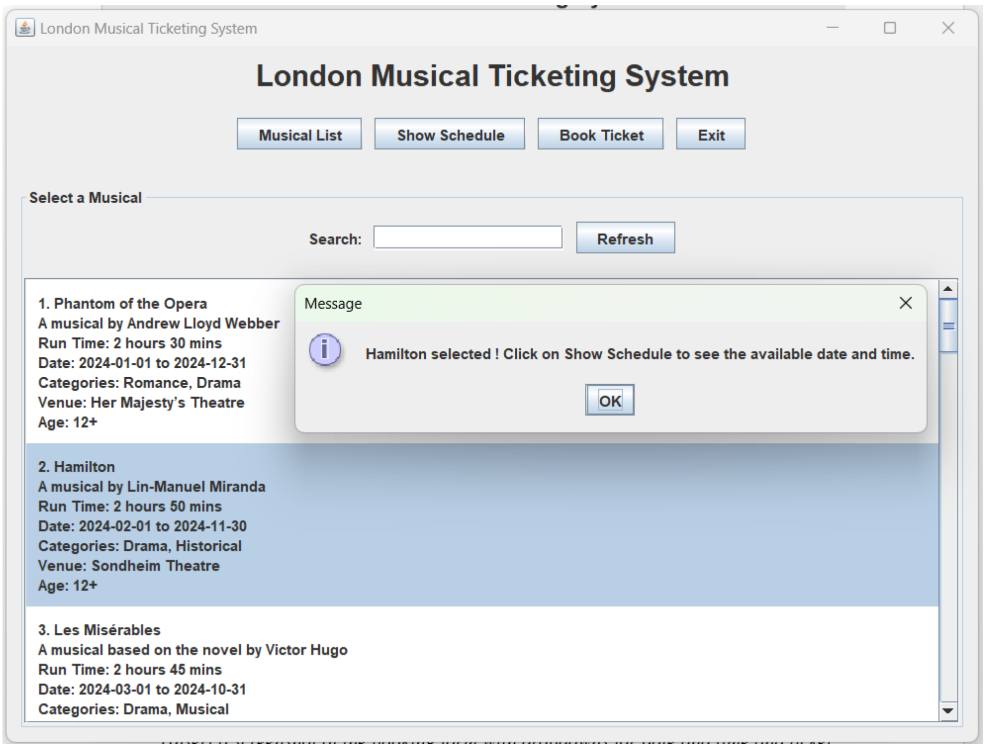
1. Phantom of the Opera
A musical by Andrew Lloyd Webber
Run Time: 2 hours 30 mins
Date: 2024-01-01 to 2024-12-31
Categories: Romance, Drama
Venue: Her Majesty's Theatre
Age: 12+

2. Hamilton
A musical by Lin-Manuel Miranda
Run Time: 2 hours 50 mins
Date: 2024-02-01 to 2024-11-30
Categories: Drama, Historical
Venue: Sondheim Theatre
Age: 12+

3. Les Misérables
A musical based on the novel by Victor Hugo
Run Time: 2 hours 45 mins
Date: 2024-03-01 to 2024-10-31
Categories: Drama, Musical

Figure 2: Main Frame Screenshot

2. Musical Selection


London Musical Ticketing System

London Musical Ticketing System

Select a Musical

Search:

1. Phantom of the Opera
A musical by Andrew Lloyd Webber
Run Time: 2 hours 30 mins
Date: 2024-01-01 to 2024-12-31
Categories: Romance, Drama
Venue: Her Majesty's Theatre
Age: 12+

Message

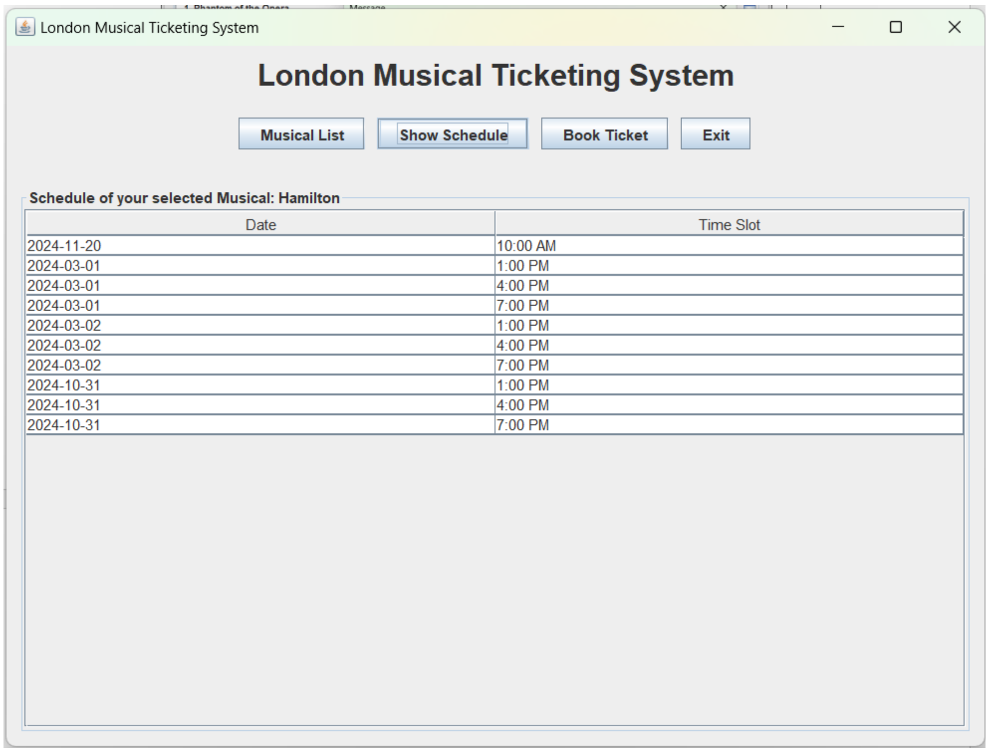
Hamilton selected ! Click on Show Schedule to see the available date and time.

2. Hamilton
A musical by Lin-Manuel Miranda
Run Time: 2 hours 50 mins
Date: 2024-02-01 to 2024-11-30
Categories: Drama, Historical
Venue: Sondheim Theatre
Age: 12+

3. Les Misérables
A musical based on the novel by Victor Hugo
Run Time: 2 hours 45 mins
Date: 2024-03-01 to 2024-10-31
Categories: Drama, Musical

Figure 3: Musical Selection Screenshot

3. Schedule Display



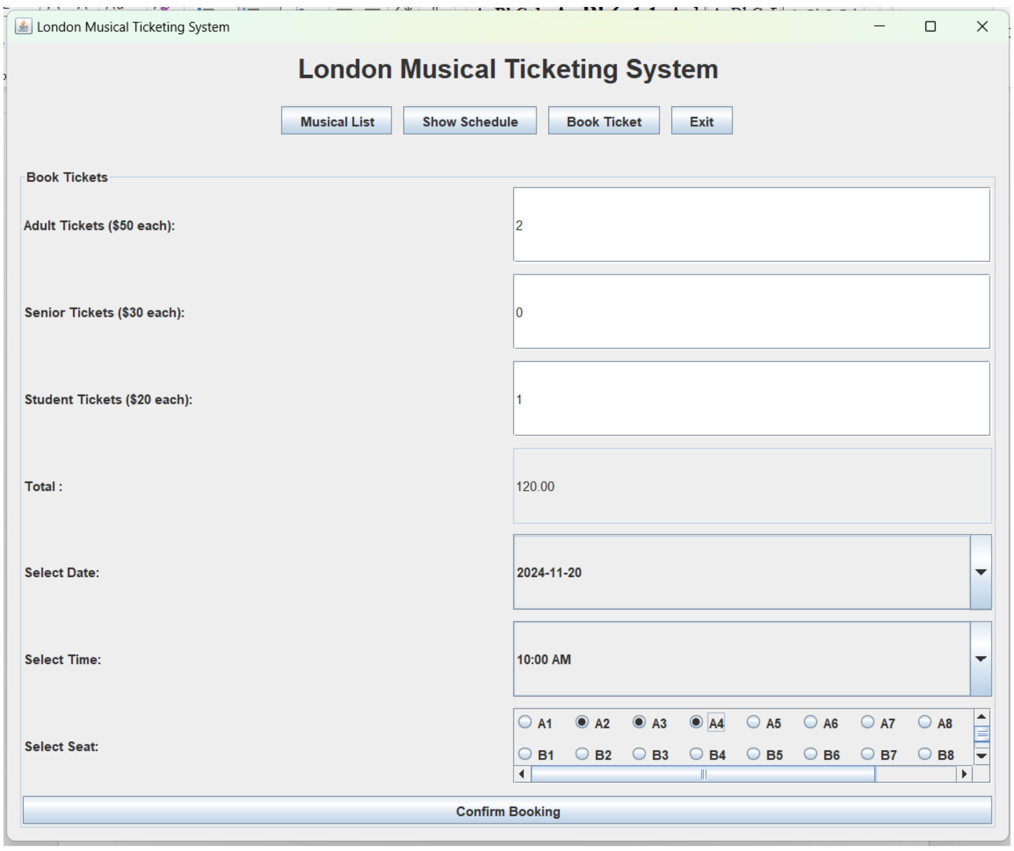
London Musical Ticketing System

Schedule of your selected Musical: Hamilton

Date	Time Slot
2024-11-20	10:00 AM
2024-03-01	1:00 PM
2024-03-01	4:00 PM
2024-03-01	7:00 PM
2024-03-02	1:00 PM
2024-03-02	4:00 PM
2024-03-02	7:00 PM
2024-10-31	1:00 PM
2024-10-31	4:00 PM
2024-10-31	7:00 PM

Figure 4: Schedule Display Screenshot

4. Ticket Booking



London Musical Ticketing System

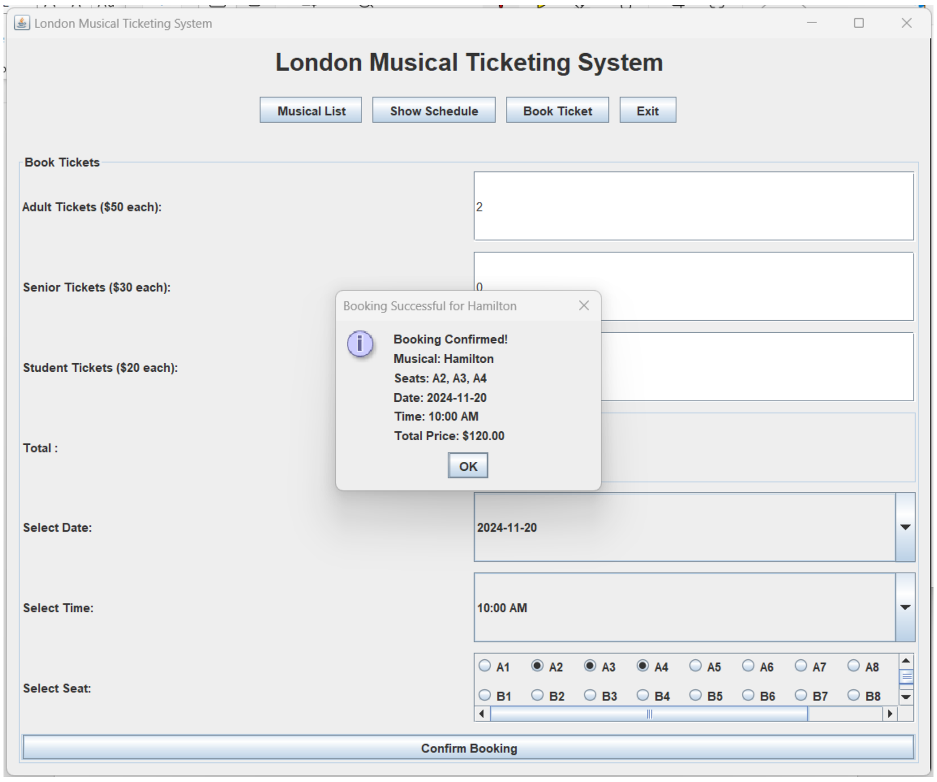
Book Tickets

Adult Tickets ($50 each):	2
Senior Tickets ($30 each):	0
Student Tickets ($20 each):	1
Total :	120.00
Select Date:	2024-11-20
Select Time:	10:00 AM
Select Seat:	A1 [✓ A2] [✓ A3] [✓ A4] A5 A6 A7 A8 / B1 B2 B3 B4 B5 B6 B7 B8

Confirm Booking

Figure 5: Ticket Booking Form Screenshot

5. Receipt Generation

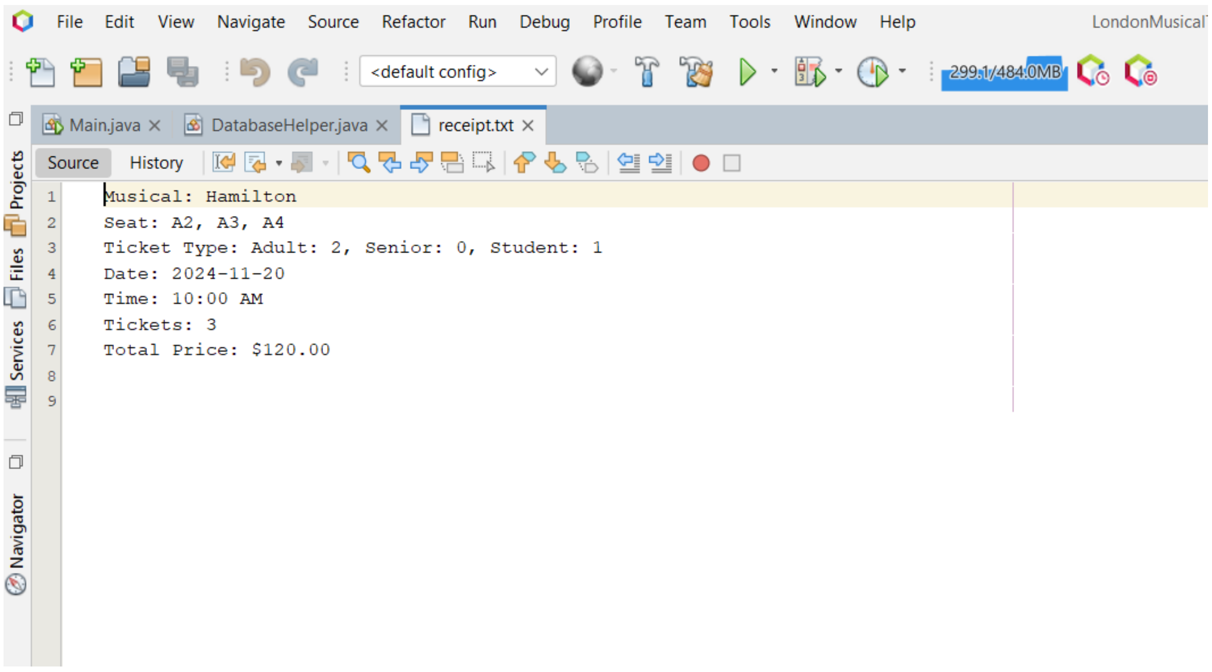

London Musical Ticketing System
London Musical Ticketing System
Musical List
Show Schedule
Book Ticket
Exit
Book Tickets
Adult Tickets ($50 each):
Senior Tickets ($30 each):
Student Tickets ($20 each):
Total :
Select Date:
Select Time:
Select Seat:
Booking Successful for Hamilton
OK
Confirm Booking

Adult Tickets ($50 each): 2
Senior Tickets ($30 each): 0
Select Date: 2024-11-20
Select Time: 10:00 AM
Select Seat: A1 [✓ A2] [✓ A3] [✓ A4] A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8

Booking Confirmed!
Musical: Hamilton
Seats: A2, A3, A4
Date: 2024-11-20
Time: 10:00 AM
Total Price: $120.00

Figure 6: Receipt Generation Screenshot

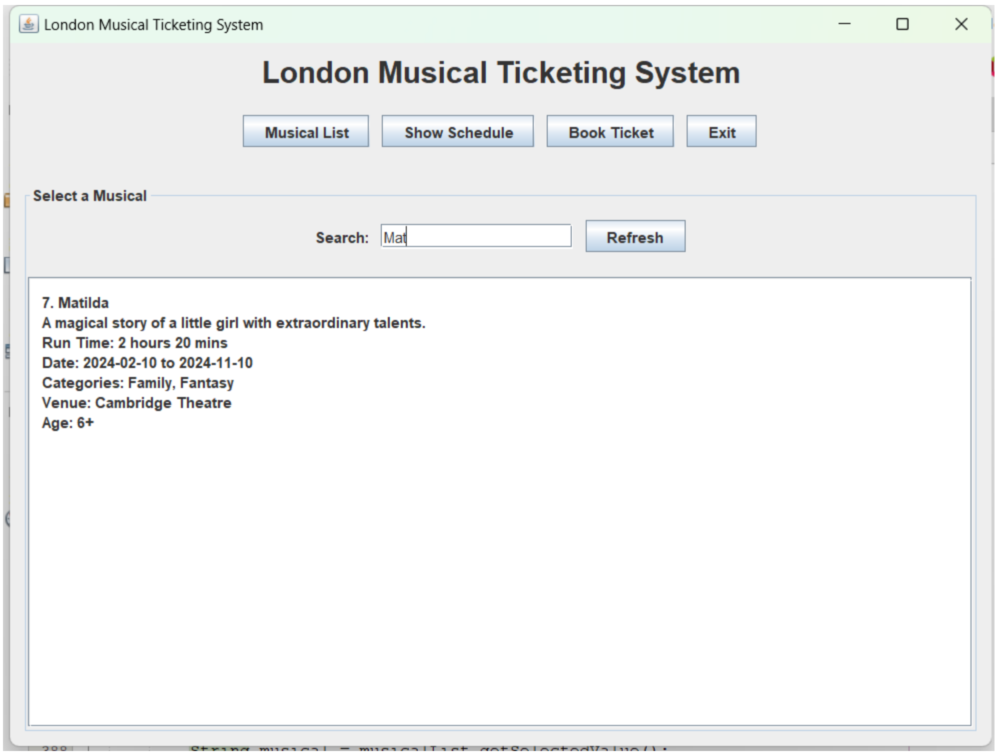
6. Receipt saved externally in a text file



```
Musical: Hamilton
Seat: A2, A3, A4
Ticket Type: Adult: 2, Senior: 0, Student: 1
Date: 2024-11-20
Time: 10:00 AM
Tickets: 3
Total Price: $120.00
```

Figure 7: Receipt.txt

7. Search Functionality



7. Matilda
A magical story of a little girl with extraordinary talents.
Run Time: 2 hours 20 mins
Date: 2024-02-10 to 2024-11-10
Categories: Family, Fantasy
Venue: Cambridge Theatre
Age: 6+

Figure 8: Search Functionality

3. Testing and Faults

3.1 White Box Testing Overview

White box testing involves examining the internal structure, logic, and implementation of the code to ensure its functionality aligns with the expected outcomes. This approach was applied to the London Musical Ticket System to validate the logical flow, boundary conditions, and input handling across different functionalities such as musical selection, ticket booking, and receipt generation.

According to Myers et al. (2011), white box testing provides a comprehensive understanding of software behavior by focusing on paths, conditions, and decision points within the code. This method was particularly suited for our project as it allowed us to verify the seamless integration of components and identify potential errors during execution.

Test Case Table

Test Case ID	Test Objective	Input Data	Expected Output	Actual Output	Pass/Fail
TC-01	Validate musical selection logic	Select "Hamilton" from the list	Displays show details for "Hamilton"	Correct details displayed	Pass
TC-02	Verify boundary conditions for ticket quantity	Input: 0, -1, or non-numeric value	Error message for invalid input	Appropriate error messages displayed	Pass
TC-03	Test receipt generation for multiple tickets	Select "Wicked," 3 tickets, Student	Receipt generated with accurate details	Correct details generated	Pass

TC-04	Check error message shown if a field is not selected	Do not select seat	Error: "Please select Seats"	Correct error message displayed	Pass
TC-05	Check whether receipt is saved in text file	Click confirm booking button after filling booking form	Dialog message for confirmation should be shown and receipt should be saved at given path	Correct message shown and receipt saved at given path	Pass
TC-06	Test database connectivity	Load musical data from MySQL	Data successfully loaded	Data loaded without issues	Pass

3.2 Faults and Failures Discussion

While executing white box tests, a few **faults and failures** were encountered. Here is a detailed discussion:

Fault 1: Negative Ticket Count Input

- **Issue:** When the user entered a negative value for the number of tickets, the system did not display an error message immediately, and instead allowed the negative number to proceed through the system, which caused incorrect total price calculations.
- **Cause:** Input validation logic was missing for non-negative ticket values.
- **Resolution:** Unresolved

Fault 2: Search Functionality Limitations

- **Issue:** The search function only displayed a single musical instead of filtering for all musicals that matched the search term.
- **Cause:** The search query did not consider partial matches properly, and the code was only fetching the exact match.
- **Resolution:** Updated the query logic to allow for partial string matching to return a list of all musicals containing the search term.

Failure 3: Price Calculation Logic

- **Issue:** There was an issue where the total ticket price was incorrectly calculated for certain combinations of ticket types.
- **Cause:** The problem stemmed from an incorrect formula in the price calculation code, where the wrong multiplier was being applied for the student ticket type.
- **Resolution:** The logic was reviewed and corrected to ensure proper multiplication for each ticket type.

4. Conclusion, further development and reflections

The London Musical Ticket System was successfully implemented, allowing customers to select musicals, showtimes, and ticket types while providing a user-friendly interface. The system's functionality progressed through stages, starting with the GUI layout and followed by the addition of features like ticket booking, receipt generation, and validation. Data was stored and manipulated effectively using a MySQL database, enhancing the program's efficiency.

Further Development:

Given three more months, I would focus on the following improvements:

- **User Experience (UX):** Enhance the GUI further by refining the layout and adding more interactive elements to improve user navigation.

- **Mobile Version:** Develop a mobile app version of the system, making it accessible for users on the go.

- **Additional Features:** Implement a loyalty program for customers who book tickets frequently, as well as a social sharing feature for users to share their bookings.

- **Database Expansion:** Expand the database to include user data and booking history for personalized recommendations and tracking.

Reflection:

Achievements with this Element of Learning

Throughout the development of the London Musical Ticket System, I enhanced my skills in Java, GUI design, and core features like ticket booking, validation, and database integration. A key achievement was creating a functional GUI that seamlessly integrates with the back-end. Additionally, I gained valuable experience in white box testing, enabling me to identify and fix bugs by thoroughly analyzing and testing the system's internal logic.

The most challenging aspects of the project were integrating the database and implementing the functional logic behind ticket bookings. Initially, connecting the MySQL database to Java was a complex task, requiring attention to detail in managing database connections, writing SQL queries, and ensuring data integrity. The database interactions were tricky because they involved different data types and validation processes, which made it essential to carefully test the connections and troubleshoot errors.

The input validation was also a difficult part of the, ensuring that the system handled all possible edge cases, such as improper search queries. The complexity arose because each user input had to be validated carefully to ensure the integrity of the data, and any mistakes could lead to incorrect pricing or booking errors. This required an in-depth understanding of validation logic and careful attention to detail.

Another challenging aspect was path testing during white box testing. Ensuring that all possible paths were tested in the system's logic was time-consuming, especially given the number of decision points and the complexity of the booking flow. The challenge was in ensuring every part of the code was covered and that no bugs went unnoticed.
Most straightforward parts, especially the basic GUI layout design and musical selection. Since I was familiar with Java Swing components like buttons, lists, and text fields, creating the basic interface and connecting it with the backend logic was relatively easy. The musical selection feature, which allowed users to choose a musical from a list, was also fairly simple to implement since it required only basic list handling and event handling logic.

Additionally, error handling was another straightforward part. Once the input validation logic was implemented, it was easy to catch common mistakes such as missing inputs or invalid ticket numbers and display appropriate error messages.

Why Were They Difficult or Easy? The more difficult parts of the project were challenging because they required careful attention to edge cases and ensuring that the system behaved correctly in all scenarios. Input validation required understanding the business rules (such as valid ticket quantities and date formats) and translating them into code that could handle every possible user interaction.

The easier parts were easier because they were more focused on straightforward tasks with fewer possible pitfalls. For example, creating a list of musical options was simple because it only involved displaying data, and there were fewer factors to consider in terms of user interactions.

References:

Beizer, B. (1995) Black-Box Testing: Techniques for Functional Testing of Software and Systems. New York: John Wiley & Sons.

Myers, G.J., Sandler, C. and Badgett, T. (2011) The Art of Software Testing. 3rd edn. Hoboken, NJ: Wiley.

Oracle (no date) The Java Programming Language. Available at: https://www.oracle.com/java/ (Accessed: 21 November 2024).

NetBeans (no date) NetBeans IDE for Java Development. Available at: https://netbeans.apache.org/ (Accessed: 21 November 2024).