Project Report

Online Event Ticket Reservation System

SEDS 504 – Spring 2025

Contents

1. Team Contributions

The project was developed by a team of three members. The table below summarizes each member's contributions.

Name	**Contributions**
Tugce Uluçay	- Implemented file-based data persistence - Developed reservation system and data saving - Developed user view/reserve operations - Created search-by-date-range functionality
Member 2	- xxx - xxx - xxx
Member 3	- xxxx - xxxx - xxx

2. Usage Scenarios

Scenario 1: Firm Adds a New Event

In this scenario, a firm user logs into the system by selecting the firm mode. From the firm menu, the user chooses to add a new event.

The user provides the event name, date, location, type, and defines two ticket categories with different prices and availability. The system confirms the successful creation of the event and it becomes available for users to search and reserve.

```
Are you a User or a Firm? (U/F): F

--- Firm Menu ---
1. Add Event
2. List Events
3. Exit
1
Enter event name: GreenTech Expo 2025
Enter event date (e.g., 2025-09-15): 2025-12-10
Enter event location: Copenhagen
Enter event type (e.g., Concert, Theatre): Conference
How many ticket categories? 2
Category name: VIP
Price: 800
Available tickets: 30
Category name: Standard
Price: 400
Available tickets: 100
Event added successfully.

--- Firm Menu ---
1. Add Event
2. List Events
3. Exit
```

Figure 1: Firm adds a new event with multiple ticket categories

Scenario 2: User Makes a Ticket Reservation

In this scenario, a user logs into the system in user mode. After viewing the list of available events, the user selects a specific event, chooses a ticket category (e.g., VIP), and specifies the quantity.

The system processes the reservation and generates a unique reservation ID. This ID allows the user to track or review the reservation in the future.

```
Are you a User or a Firm? (U/F): U

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
3
Enter your username (or leave empty to continue as guest): tugce
1. Jazz Festival on 2025-08-20 at Berlin
   - VIP: $600.0, Available: 18
   - Regular: $300.0, Available: 50
2. The Phantom Play on 2025-09-10 at Vienna
   - Front: $400.0, Available: 10
   - Balcony: $200.0, Available: 30
3. Tech Conference on 2025-11-01 at Amsterdam
   - VIP: $750.0, Available: 30
   - Regular: $350.0, Available: 100
4. GreenTech Expo 2025 on 2025-12-10 at Copenhagen
   - VIP: $800.0, Available: 30
   - Standard: $400.0, Available: 100
Select event number: 1
Enter ticket category: VIP
Enter number of tickets: 2
Reservation successful! ID: a79b6723-6ae4-41ea-810a-490aa8b5e1f0

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
```

Figure 2: User reserves VIP tickets and receives a reservation ID

Scenario 3: User Views Their Reservations

In this scenario, a registered user logs into the system and chooses the "View Reservations" option.

The system prompts for the username and retrieves all past reservations associated with that user. Each reservation includes a unique ID, the event name, ticket category, and quantity reserved. This allows users to keep track of their bookings.

```
Are you a User or a Firm? (U/F): U

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
4
Enter your username: tugce
Reservation ID: 4c5e159b-141e-4935-9a2e-835089236c18, Event: Jazz Festival, Category: vip, Quantity: 2
Reservation ID: 473c3bf5-8332-46ca-b2cc-0fae6727b8b6, Event: Jazz Festival, Category: VIP, Quantity: 2

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
```

Figure 3: User views past reservations using their username

Scenario 4: Search Events by Date Range

In this scenario, a user logs into the system and selects the "Search Events by Date Range" option.

The user provides a start date and an end date in the format YYYY-MM-DD. The system then searches for and lists all events whose dates fall within the specified interval. The result includes the event name, date, location, ticket categories, and availability.

```
Are you a User or a Firm? (U/F): U

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
2
Enter start date (YYYY-MM-DD): 2025-08-01
Enter end date (YYYY-MM-DD): 2025-09-01
Jazz Festival on 2025-08-20 at Berlin
   - VIP: $600.0, Available: 16
   - Regular: $300.0, Available: 50

1. List Events
2. Search Events by Date Range
3. Reserve Ticket
4. View Reservations
5. Exit
```

Figure 4: User searches for events within a specific date range

3. UML Class Diagram

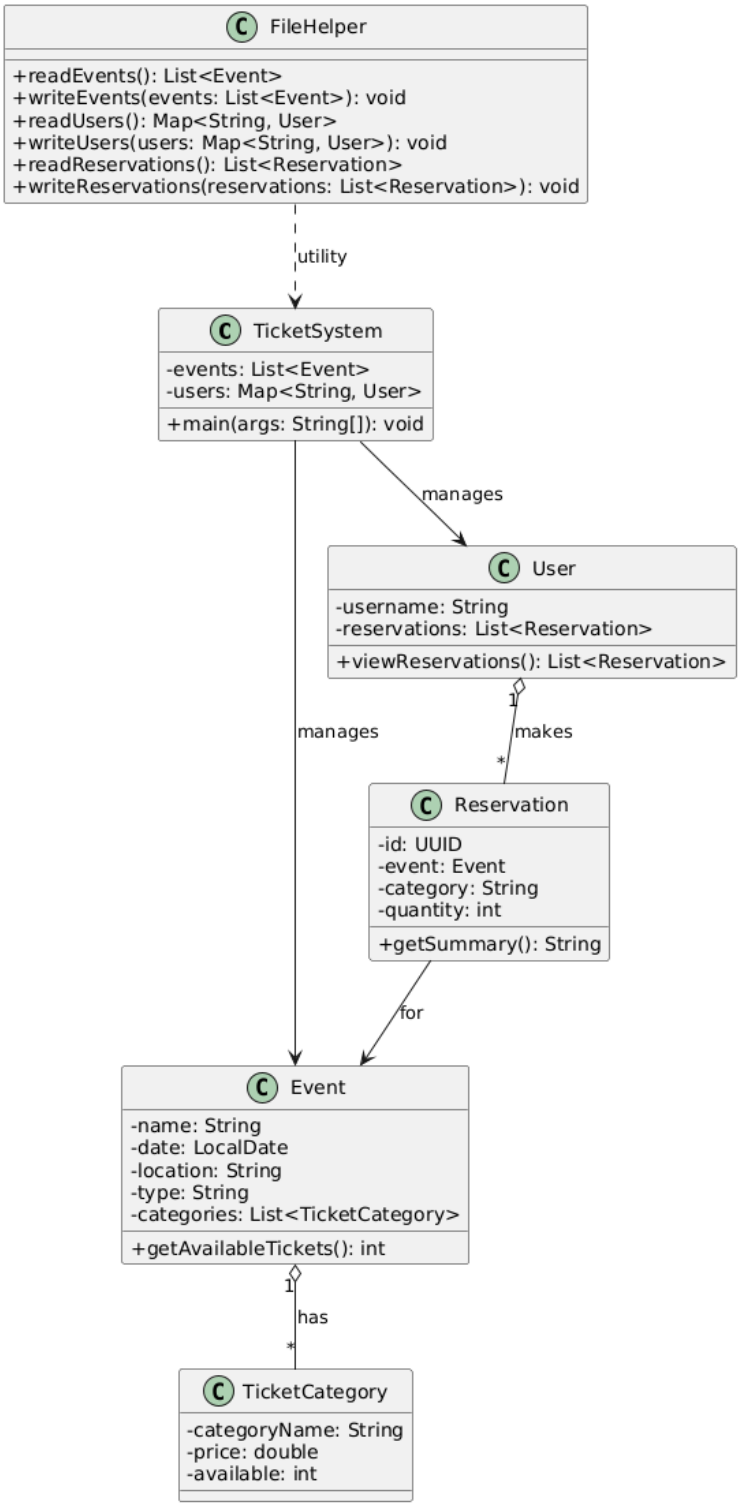

Figure 5: UML Class Diagram of the Event Ticket Reservation System

4. Explanation of Inheritance

In this project, inheritance is not explicitly used to create a class hierarchy, because the core functionality of the application revolves around distinct entities such as `Event`, `User`, `Reservation`, and `TicketCategory`.

Each class represents a concrete concept in the domain and does not share a common parent class that would benefit from inheritance. Since no abstract or base classes were necessary, and behaviors were not shared across types, inheritance was intentionally avoided to keep the system simple and readable.

However, if the application were to be expanded with different types of users (e.g., AdminUser, GuestUser), inheritance could be used to define a generic `User` superclass and specialize its behavior in subclasses.

5. Explanation of Polymorphism

Polymorphism is utilized in the system in several ways through interface-based design and functional constructs.

One example is the use of Java streams to dynamically filter ticket categories when a user selects one:

```
TicketCategory selectedCat = event.getCategories().stream()
    .filter(c -> c.getCategoryName().equalsIgnoreCase(category))
    .findFirst().orElse(null);
```

This approach abstracts the iteration logic and demonstrates polymorphic behavior of collection elements.

Additionally, Java collections such as `List<Event>` and `Map<String, User>` exhibit polymorphic use of interfaces. For example, `ArrayList` is used via the `List` interface, allowing the underlying implementation to be swapped without changing the rest of the codebase:

```
List<Event> events = new ArrayList<>();
```

While method overriding was not directly applied (since no subclassing exists), the code structure allows future polymorphic extension, such as replacing data storage with database classes implementing a common interface.

6. Data Persistence

The system implements data persistence using local text files. All key data entities—users, events, and reservations—are stored and retrieved using a helper class called `FileHelper`.

Files used:

- `users.txt` – Stores registered usernames
- `events.txt` – Stores all event details, including categories, prices, and availability
- `reservations.txt` – Stores user reservations with a unique reservation ID

How it works:

- When the application starts, all data is loaded from the text files.
- When a reservation is made or a new event is added, the files are updated immediately.
- Upon exiting the application, the current state is saved again to ensure no data loss.

Text files were preferred over database systems like MySQL for simplicity and portability, as this project is a console-based educational application. However, the system structure is flexible enough to be extended with database support in the future.

All file I/O operations are encapsulated inside the `FileHelper` class to ensure modularity and reusability.