

# Online Event Ticket Reservation System - Project Report

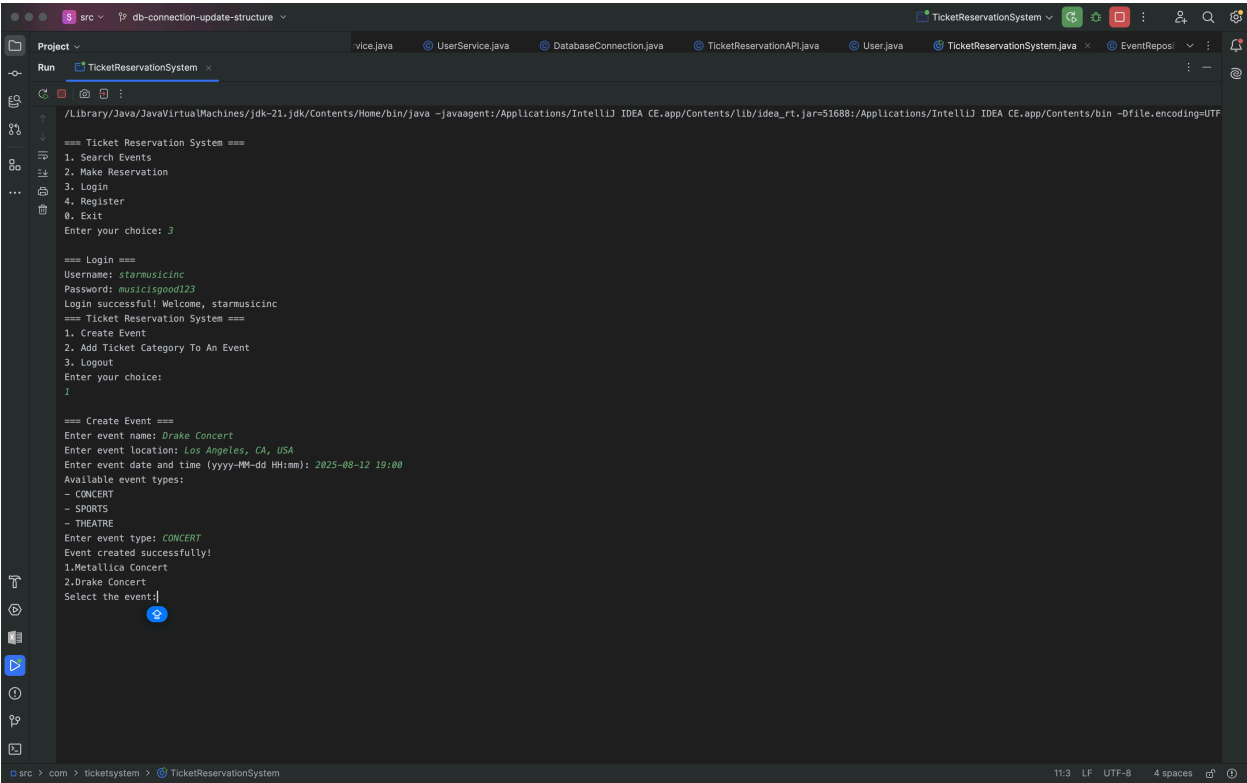
## Team Contributions Table

Team Member	Contribution
Bariş Yenigül	Implemented Firm Operations including event creation, ticket category management, and event type handling
Ahmet Can Karataş	Implemented User Operations including ticket search, reservation system, and user account management
Erhan Alasar	Implemented Data Persistence using PostgreSQL database with repository pattern

## Usage Scenarios

### Scenario 1: Creating a New Event

1. A firm representative logs into the system
2. Navigates to the event creation section
3. Enters event details:
  - Event name: "Drake Concert"
  - Date: August 12, 2025
  - Location: "Los Angeles CA, USA"
  - Type: Concert
4. Creates ticket categories:
  - VIP: \$300.99 (150 tickets)
  - Regular: \$100.99 (500 tickets)
5. System confirms event creation and displays event ID



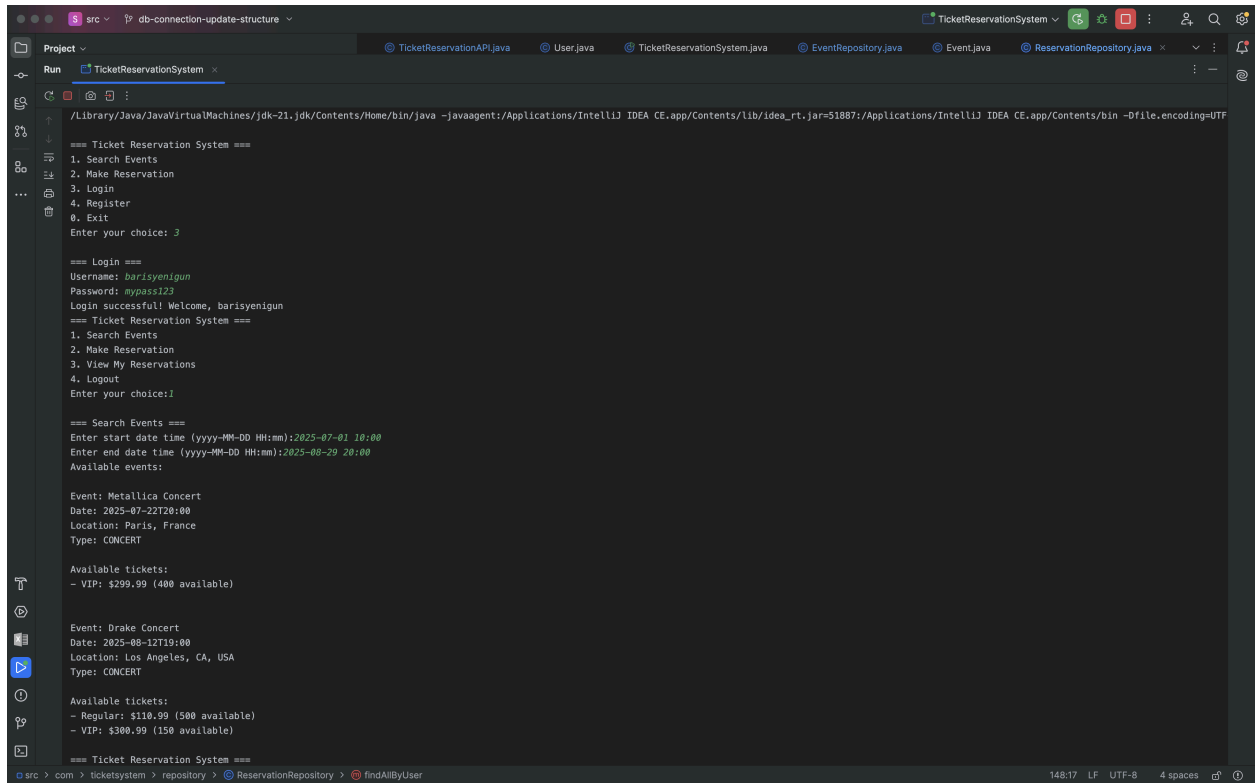
```
src > db-connection-update-structure > TicketReservationSystem
Run TicketReservationSystem
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51688:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. Login
4. Register
0. Exit
Enter your choice: 3

=== Login ===
Username: starmusicinc
Password: musicisgood123
Login successful! Welcome, starmusicinc
=== Ticket Reservation System ===
1. Create Event
2. Add Ticket Category To An Event
3. Logout
Enter your choice: 1

=== Create Event ===
Enter event name: Drake Concert
Enter event location: Los Angeles, CA, USA
Enter event date and time (yyyy-MM-dd HH:mm): 2025-08-12 19:00
Available event types:
- CONCERT
- SPORTS
- THEATRE
Enter event type: CONCERT
Event created successfully!
1.Metallica Concert
2.Drake Concert
Select the event:]
```

## Scenario 2: Searching and Reserving Tickets

1. User searches for events between July 7- August 29, 2025
2. System displays available events with details
3. User selects "Drake Concert"
4. Views available ticket categories and prices
5. Selects 10 Regular tickets
6. System generates a unique reservation number
7. User receives confirmation with reservation details



```
src db-connection-update-structure TicketReservationSystem
TicketReservationAPI.java User.java TicketReservationSystem.java EventRepository.java Event.java ReservationRepository.java
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51887:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. Login
4. Register
0. Exit
Enter your choice: 3

=== Login ===
Username: barisyenigun
Password: mypass123
Login successful! Welcome, barisyenigun
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. View My Reservations
4. Logout
Enter your choice: 1

=== Search Events ===
Enter start date time (yyyy-MM-DD HH:mm): 2025-07-01 10:00
Enter end date time (yyyy-MM-DD HH:mm): 2025-08-29 20:00
Available events:

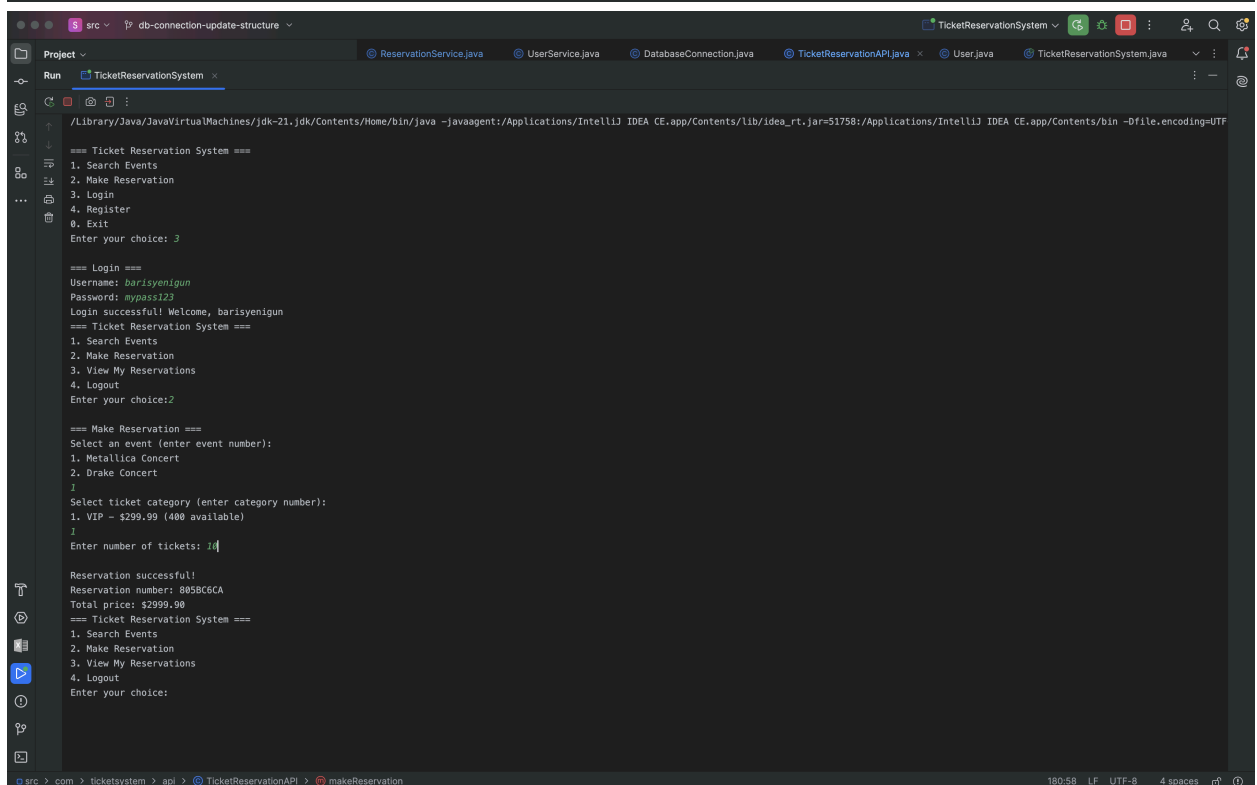
Event: Metallica Concert
Date: 2025-07-22T20:00
Location: Paris, France
Type: CONCERT

Available tickets:
- VIP: $299.99 (400 available)

Event: Drake Concert
Date: 2025-08-12T19:00
Location: Los Angeles, CA, USA
Type: CONCERT

Available tickets:
- Regular: $110.99 (500 available)
- VIP: $300.99 (150 available)

=== Ticket Reservation System ===
src > com > ticketssystem > repository > findAllByUser 148:17 LF UTF-8 4 spaces
```



```
src db-connection-update-structure TicketReservationSystem
ReservationService.java UserService.java DatabaseConnection.java TicketReservationAPI.java User.java TicketReservationSystem.java
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51758:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. Login
4. Register
0. Exit
Enter your choice: 3

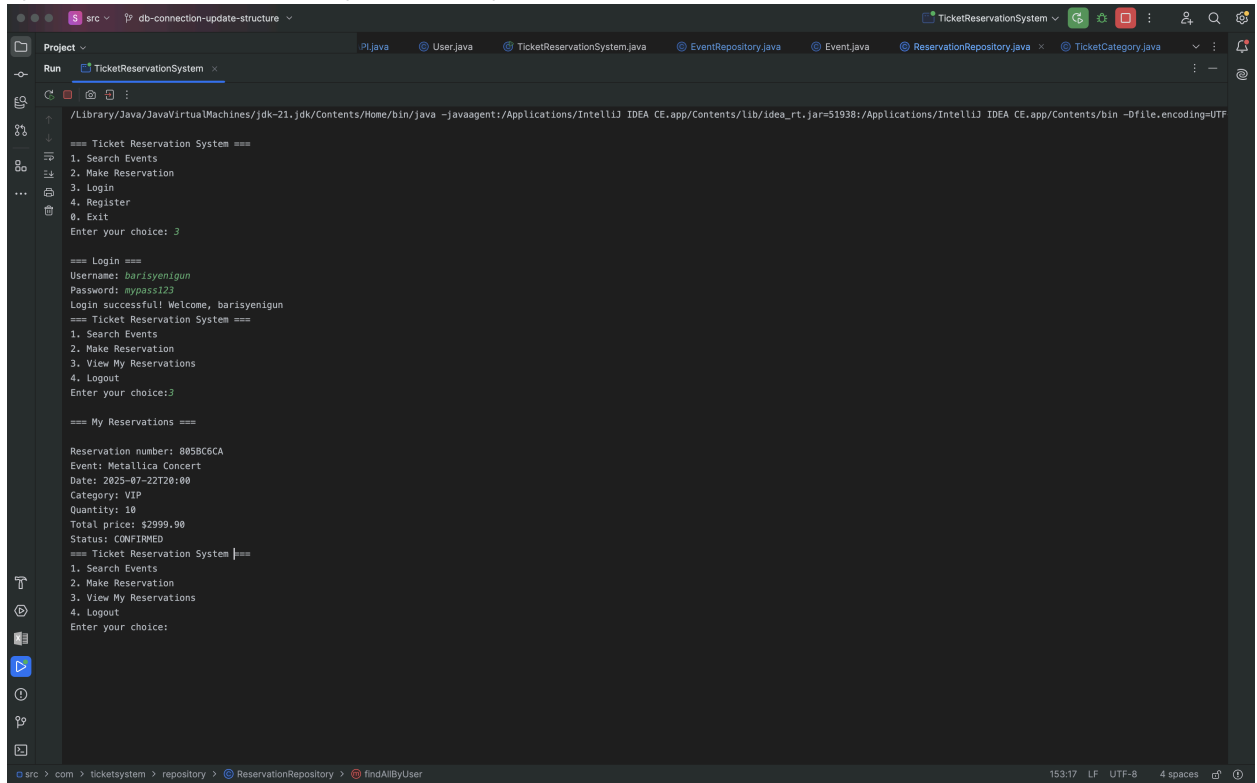
=== Login ===
Username: barisyenigun
Password: mypass123
Login successful! Welcome, barisyenigun
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. View My Reservations
4. Logout
Enter your choice: 2

=== Make Reservation ===
Select an event (enter event number):
1. Metallica Concert
2. Drake Concert
1
Select ticket category (enter category number):
1. VIP - $299.99 (400 available)
1
Enter number of tickets: 10

Reservation successful!
Reservation number: 805BCECA
Total price: $2999.90
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. View My Reservations
4. Logout
Enter your choice:
src > com > ticketssystem > api > TicketReservationAPI > makeReservation 180:58 LF UTF-8 4 spaces
```

## Scenario 3: Managing User Reservations

1. User logs into their account
2. Views current and past reservations
3. System updates ticket availability automatically



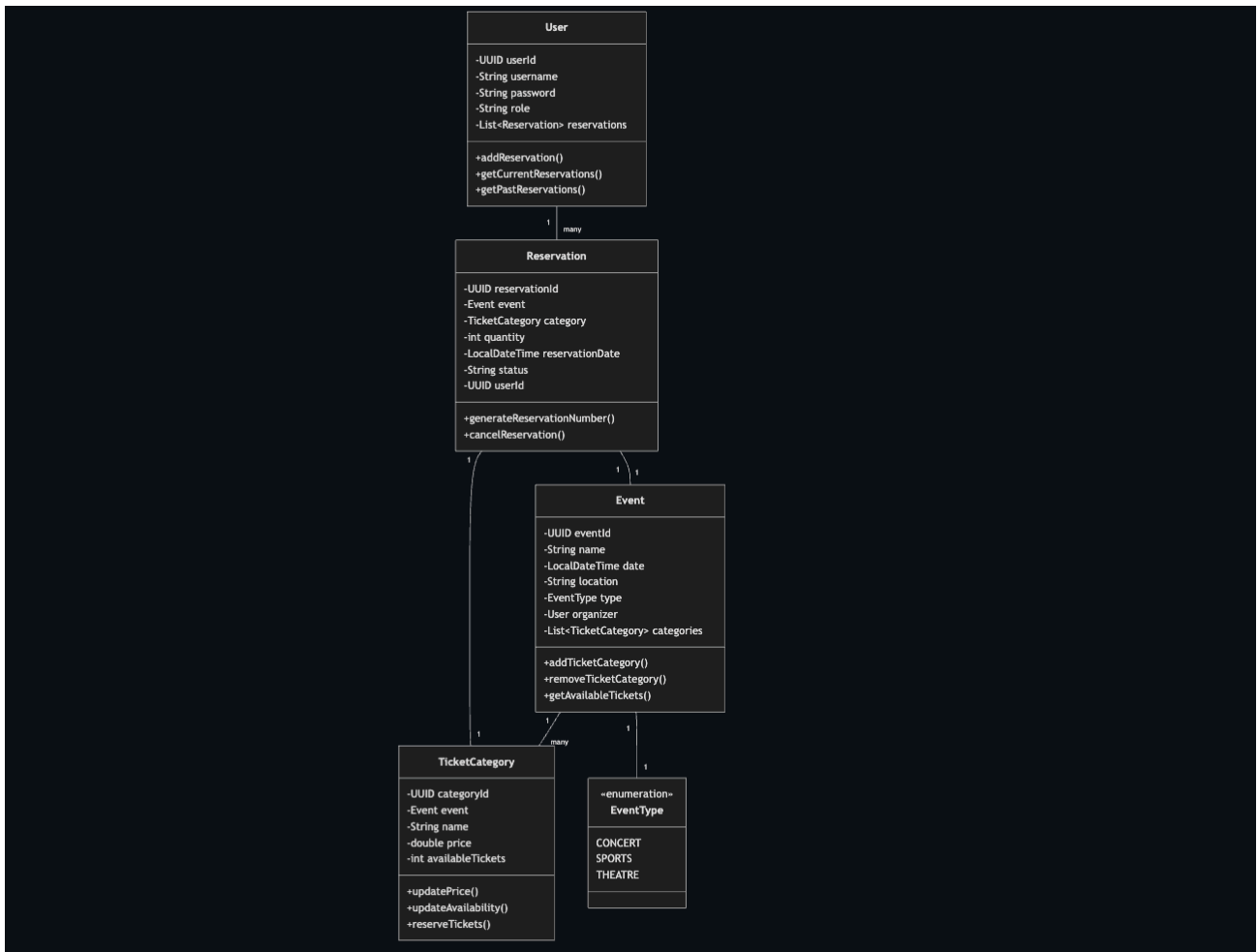
```
src db-connection-update-structure TicketReservationSystem
Project PL.java User.java TicketReservationSystem.java EventRepository.java Event.java ReservationRepository.java TicketCategory.java
Run TicketReservationSystem
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=51938:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. Login
4. Register
0. Exit
Enter your choice: 3

=== Login ===
Username: barisyeenigun
Password: mypass123
Login successfull Welcome, barisyeenigun
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. View My Reservations
4. Logout
Enter your choice:3

=== My Reservations ===

Reservation number: 805BC6CA
Event: Metallica Concert
Dates: 2025-07-22T20:00
Category: VIP
Quantity: 10
Total price: $2999.90
Status: CONFIRMED
=== Ticket Reservation System ===
1. Search Events
2. Make Reservation
3. View My Reservations
4. Logout
Enter your choice:
```

## UML Class Diagram



## Explanation of Inheritance

The system implements inheritance through the repository pattern:

### 1. IRepository<T, ID> Interface

- Base interface defining common CRUD operations
- Generic type parameters for entity type and ID type
- Implemented by all specific repositories

### 2. Specialized Repository Interfaces

- IEventRepository**: Extends **IRepository** with event-specific queries
- IUserRepository**: Extends **IRepository** with user-specific queries
- IReservationRepository**: Extends **IRepository** with reservation-specific queries
- ITicketCategoryRepository**: Extends **IRepository** with ticket category-specific queries

Example of a specialized repository interface:

```
public interface IEventRepository<T, ID> extends IRepository<T, ID> {
    List<T> findEventsByTimeInterval(LocalDateTime startDate, LocalDateTime endDate);
}
```

```
List<T> findEventsByOrganizer(UUID organizerId);  
}
```

This inheritance hierarchy allows for:

- Code reuse across repositories
- Consistent CRUD operations
- Type-safe implementations
- Easy extension for new repository types

## Explanation of Polymorphism

---

Polymorphism is applied in several areas:

### 1. Repository Pattern

- All repositories implement the `IRepository` interface
- Different implementations can be swapped without changing client code
- Example: `UserRepository` implements `IUserRepository`

### 2. Event Types

- `EventType` enum allows for different event categories
- System can handle different event types uniformly
- Easy to add new event types without changing existing code

Benefits:

- Flexible and extensible code
- Reduced coupling between components
- Easy maintenance and updates

## Explanation of Data Persistence

---

The system uses PostgreSQL database for data persistence with the following implementation:

### 1. Database Structure

- Tables: users, events, ticket\_categories, reservations
- Relationships maintained through foreign keys

### 2. Repository Pattern Implementation

- Each entity has its own repository class
- Repositories handle all database operations

### 3. Data Operations

- CRUD operations for all entities

Example of transaction management:

```
public void save(Reservation obj) {  
    String sql = "INSERT INTO reservations (reservation_id, event_id, category_id, quantity, reservation_date, s
```

```
try {
    connection.setAutoCommit(false); // Start transaction
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        // Set parameters and execute
        stmt.executeUpdate();
        connection.commit(); // Commit transaction
    }
} catch (SQLException e) {
    try {
        connection.rollback(); // Rollback on error
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
}
```