Diagram Document

Use Case 1.1: User Registration

Name	Content
Operation	`registerUser(string: username, string: password)`
Cross References	• username: The desired username provided by the user. • password: The password provided by the user.
Preconditions	1. The user is on the registration page. 2. The system is accessible and operational.
Postconditions	1. If successful, a new user account is created and a confirmation email is sent. 2. If unsuccessful, an error message is displayed.

Use Case 1.2: Secure User Login/Auth

Name	Content
Operation	`loginUser(string: username, string: password)`
Cross References	• username: The username provided by the user. • password: The password provided by the user.
Preconditions	1. The user is on the login page. 2. The username and password correspond to a valid and registered user.
Postconditions	1. If login credentials are valid, the user gains access to their account. 2. If unsuccessful, an error message is displayed.

Use Case 2.1: Search for Events

Name	Content
Operation	`searchEvents(string: criteria, string: filters, string: sort)`
Cross References	• criteria: Search criteria entered by the user. • filters: Filters applied by the user. • sort: Sorting option chosen by the user.
Preconditions	1. Database includes events. 2. User has permission to search events.
Postconditions	1. The operation provides the user with a list of appropriate events. 2. If no events match the criteria, a message is displayed.

Use Case 2.2: View Event Details

Name	Content
Operation	`viewEventDetails(int: eventID)`
Cross References	• eventID: The unique identifier of the event to be viewed.
Preconditions	1. The user is on the event listings page.
Postconditions	1. User successfully navigates to and views the Event Details page.

Use Case 2.3: Purchase Tickets

Name	Content
Operation	`purchaseTickets(int: eventID, int: quantity, object: paymentDetails)`
Cross References	• eventID: The unique identifier of the event. • quantity: Number of tickets to purchase. • paymentDetails: Payment information.
Preconditions	1. The user is logged in. 2. The eventID corresponds to a valid event. 3. Tickets are available for the event. 4. User has valid payment information.
Postconditions	1. If successful, tickets are purchased and a confirmation message is displayed. 2. If unsuccessful, an error message is displayed.

Use Case 2.4: Process Mock Payment

Name	Content
Operation	`processMockPayment(int: amount, string: paymentMethod)`
Cross References	• amount: The total cost of the tickets. • paymentMethod: The payment method chosen by the user.
Preconditions	1. The user is on the Event Details page. 2. The user has selected tickets to purchase.
Postconditions	1. If successful, the mock payment is processed and confirmed. 2. If unsuccessful, an error message is displayed.

Use Case 2.5: Update Ticket Inventory

Name	Content
Operation	`updateTicketInventory(int: eventID, int: quantity)`
Cross References	• eventID: The unique identifier of the event. • quantity: Number of tickets sold.
Preconditions	1. Tickets have been successfully purchased. 2. System administrator or event organizer is logged in.
Postconditions	1. The ticket inventory is updated accurately.

Use Case 2.6: Send Order Confirmation

Name	Content
Operation	`sendOrderConfirmation(int: userID, int: orderID)`
Cross References	• userID: The unique identifier of the user. • orderID: The unique identifier of the order.
Preconditions	1. The user has successfully completed a ticket purchase.
Postconditions	1. The user receives an order confirmation, tickets, and QR code via email and/or text in a specified format (e.g., PDF, text).

Use Case 2.7: Verify QR Code

Name	Content
Operation	`verifyQRCode(string: QRCode)`
Cross References	• QRCode: The QR code presented by the user.
Preconditions	1. The user has a valid ticket and QR code for the event.
Postconditions	1. If the QR code is valid, the user is granted entry to the event. 2. If the QR code is invalid, an alert is displayed to the staff.

Additional Operation Contract 1: Update User Information

Name	Content
Operation	`updateUserInfo(int: userID, object: newUserInfo)`
Cross References	• userID: The unique identifier of the user. • newUserInfo: An object containing updated user information such as name, email, etc.
Preconditions	1. The userID corresponds to a valid and registered user in the system. 2. The user is logged in.
Postconditions	1. The user's information is updated in the system. 2. A confirmation or alert is sent to the user.

Additional Operation Contract 2: Change User Password

Name	Content
Operation	`changeUserPassword(int: userID, string: oldPassword, string: newPassword)`
Cross References	• userID: The unique identifier of the user. • oldPassword: The current password. • newPassword: The new password.
Preconditions	1. The userID corresponds to a valid and registered user in the system. 2. The oldPassword matches the current password in the system.
Postconditions	1. The user's password is updated in the system. 2. A confirmation or alert is sent to the user.

Additional Operation Contract 3: Delete User Account

Name	Content
Operation	`deleteUserAccount(int: userID, string: userPassword)`
Cross References	• userID: The unique identifier of the user. • userPassword: The current password of the user.
Preconditions	1. The userID corresponds to a valid and registered user in the system. 2. The userPassword matches the current password in the system.
Postconditions	1. The user's account is deleted from the system.

Additional Operation Contract 4: Add Event

Name	Content
Operation	`addEvent(object: eventDetails)`
Cross References	• eventDetails: An object containing all the details of the event to be added.
Preconditions	1. The system administrator or event organizer is logged in.
Postconditions	1. A new event is added to the system. 2. Event listings are updated.

Additional Operation Contract 5: Update Event

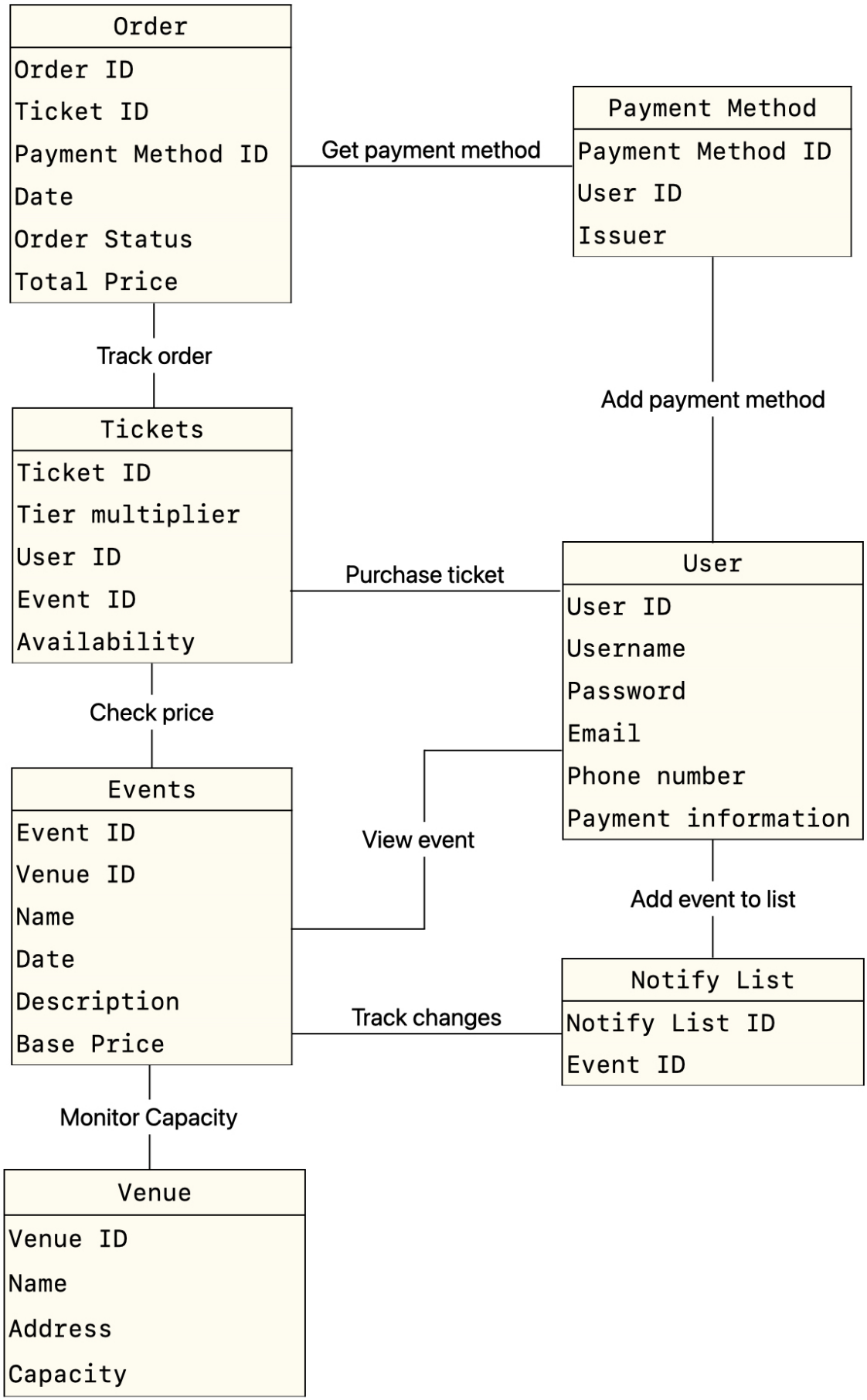
Name	Content
Operation	`updateEvent(int: eventID, object: updatedEventDetails)`
Cross References	• eventID: The unique identifier of the event. • updatedEventDetails: An object containing updated event details.
Preconditions	1. The eventID corresponds to a valid and existing event in the system. 2. The system administrator or event organizer is logged in.
Postconditions	1. The event details are updated in the system. 2. Event listings are updated.

Additional Operation Contract 6: Delete Event

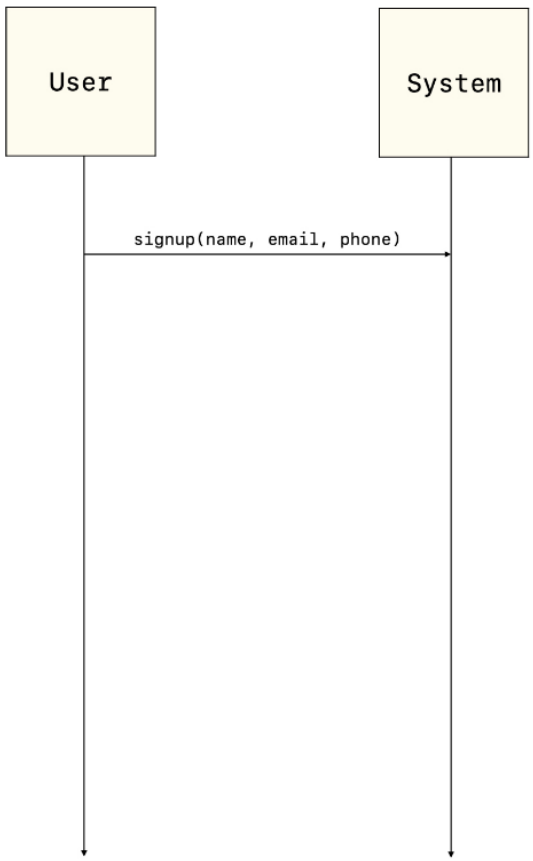
Name	Content
Operation	`deleteEvent(int: eventID)`
Cross References	• eventID: The unique identifier of the event.
Preconditions	1. The eventID corresponds to a valid and existing event in the system. 2. The system administrator or event organizer is logged in.
Postconditions	1. The event is deleted from the system. 2. Event listings are updated.

Additional Operation Contract 7: Send Notifications

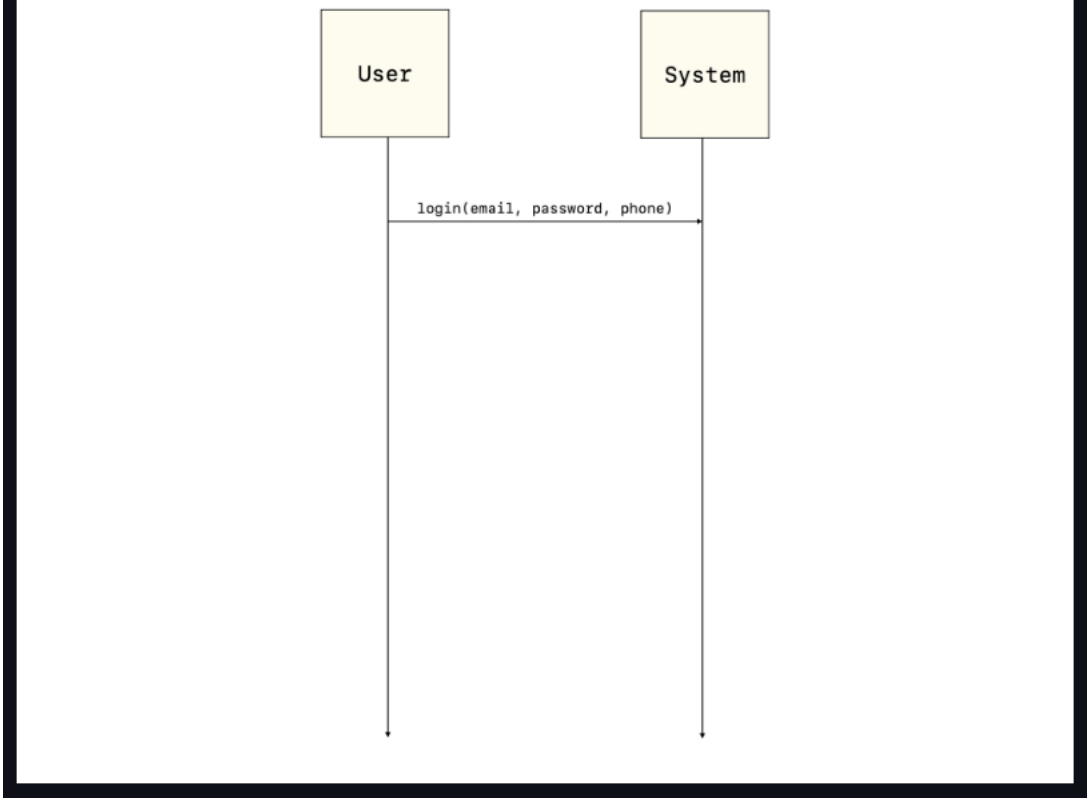
Name	Content
Operation	`sendNotification(int: userID, string: notificationType, object: notificationDetails)`
Cross References	• userID: The unique identifier of the user. • notificationType: The type of notification (e.g., "Low Ticket Alert", "New Event"). • notificationDetails: An object containing the details of the notification.
Preconditions	1. The userID corresponds to a valid and registered user in the system. 2. The types of notifications that can be sent are predefined.
Postconditions	1. A notification is sent to the user via a specified medium (e.g., email, in-app).


Order
Order ID
Ticket ID
Payment Method ID
Date
Order Status
Total Price
Get payment method
Payment Method
Payment Method ID
User ID
Issuer
Track order
Add payment method
Tickets
Ticket ID
Tier multiplier
User ID
Event ID
Availability
Purchase ticket
User
User ID
Username
Password
Email
Phone number
Payment information
Check price
View event
Add event to list
Events
Event ID
Venue ID
Name
Date
Description
Base Price
Notify List
Notify List ID
Event ID
Track changes
Monitor Capacity
Venue
Venue ID
Name
Address
Capacity

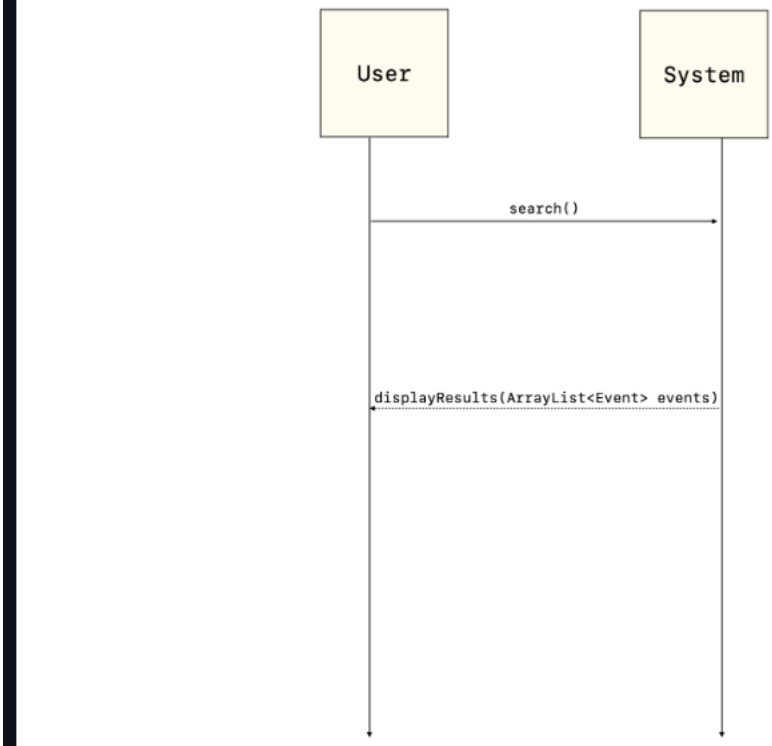
User Registration



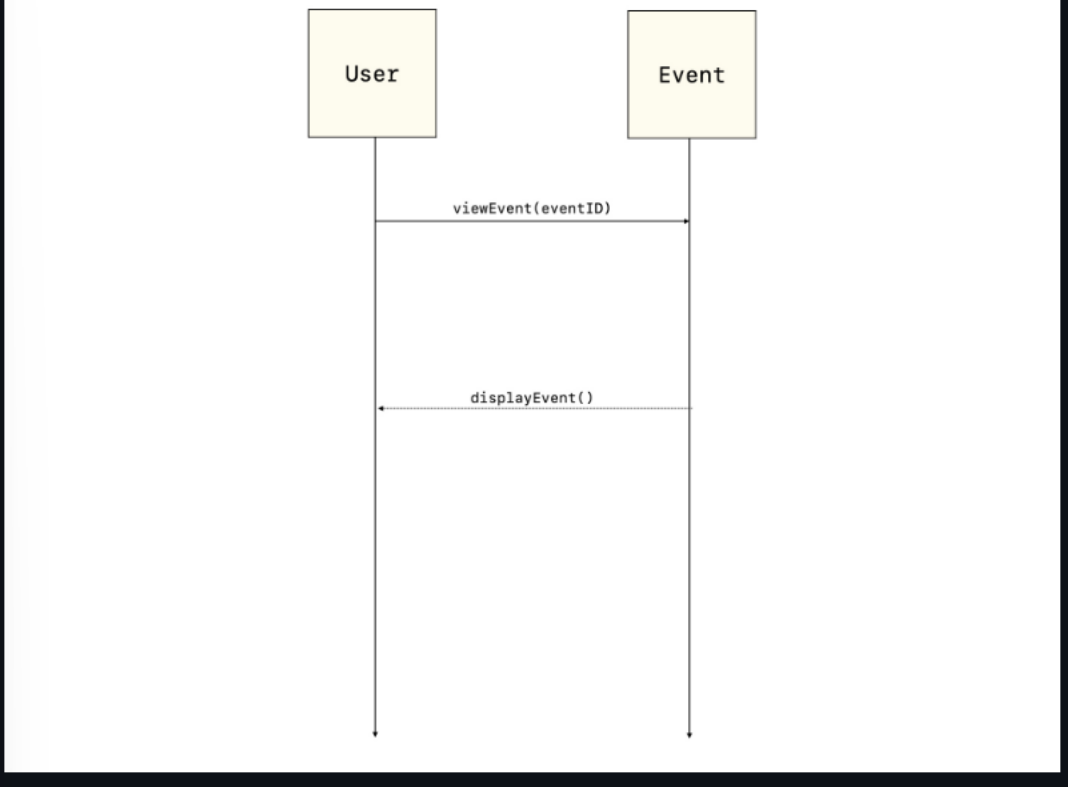
User Login



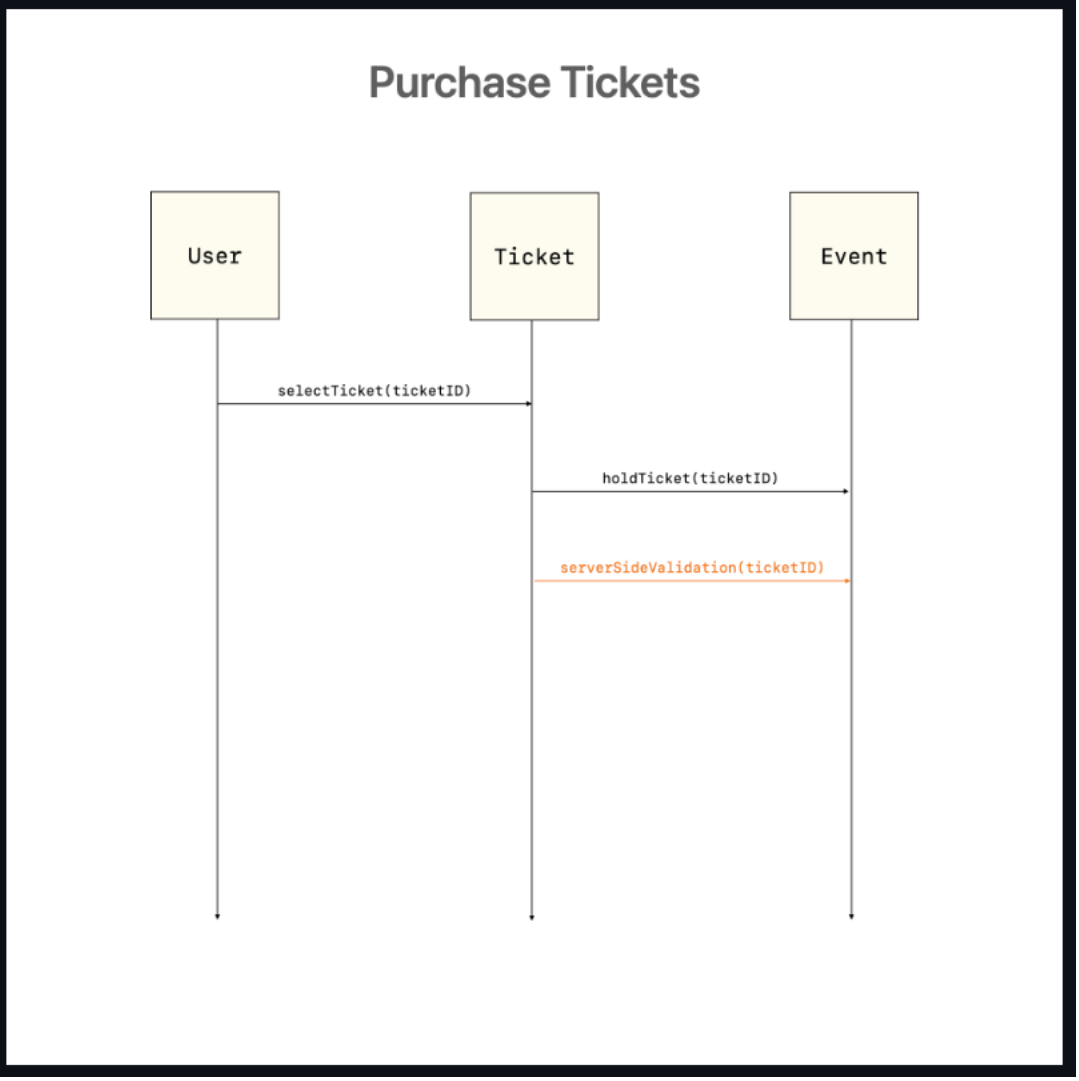
Search for Events



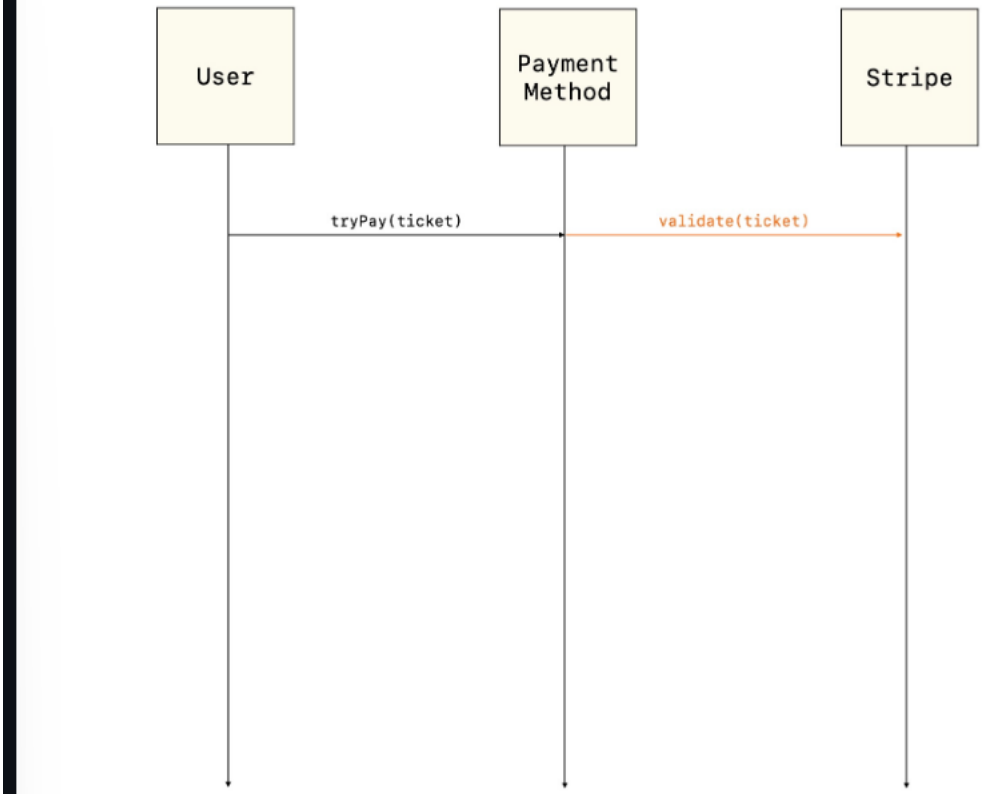
View Event Details



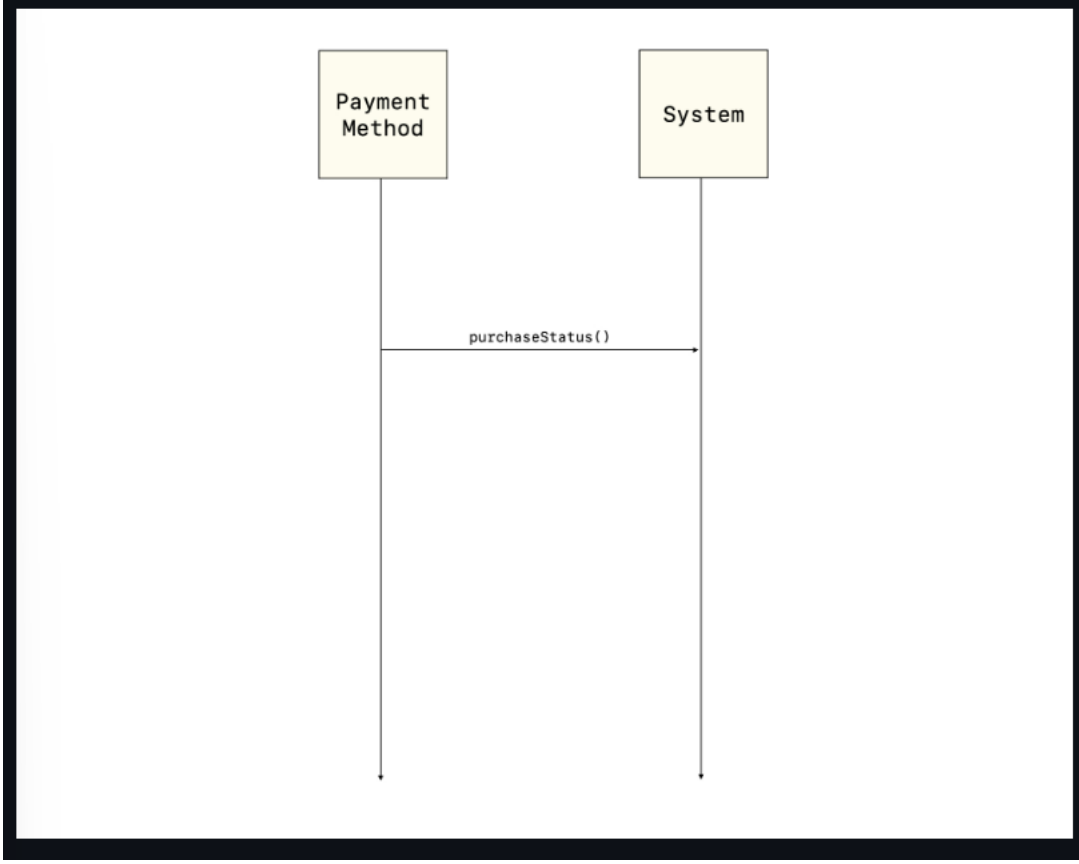
Purchase Tickets



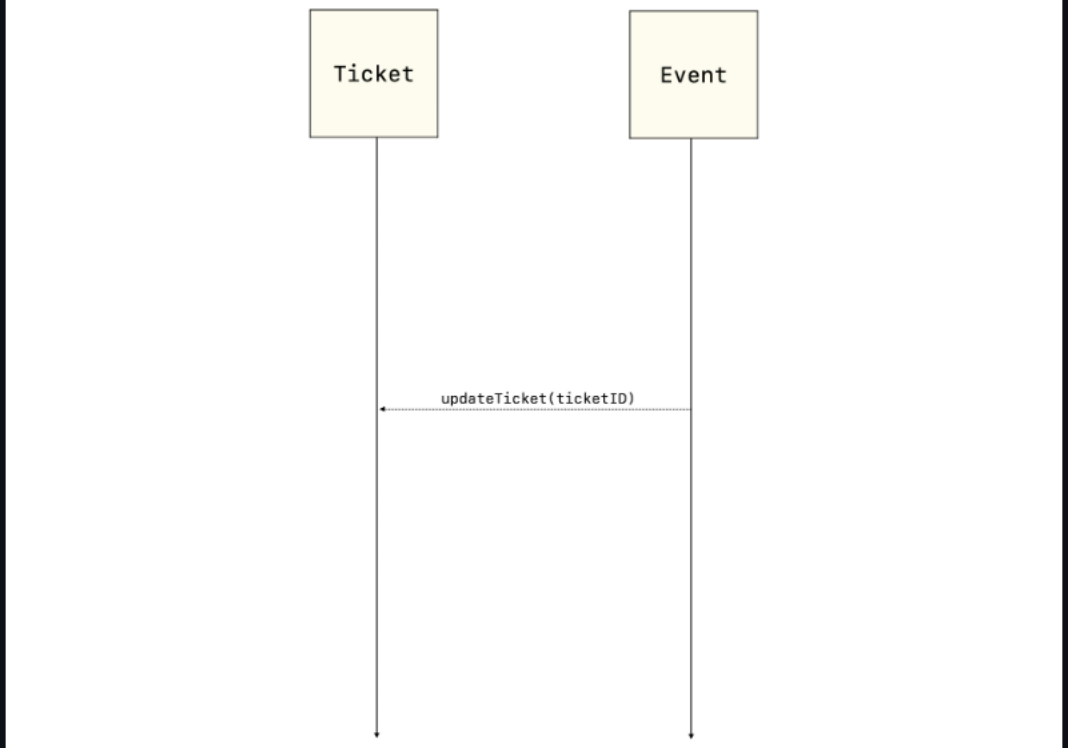
Process Payment



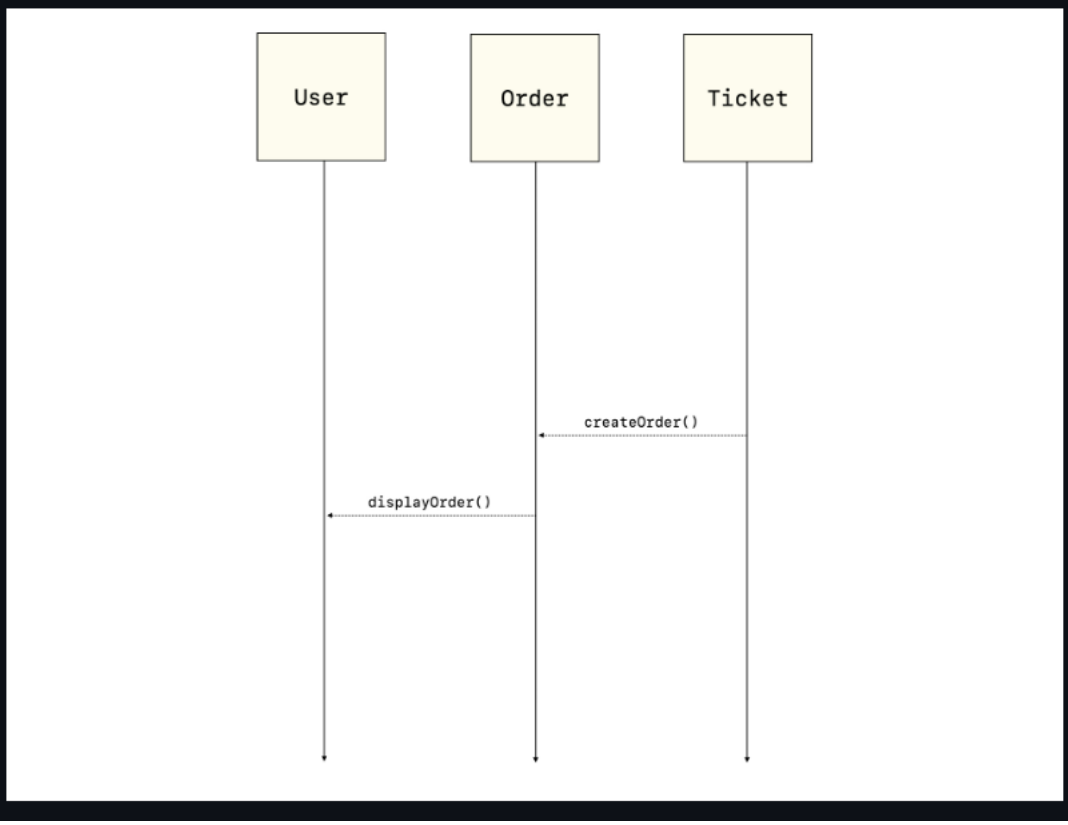
Display Purchase Status



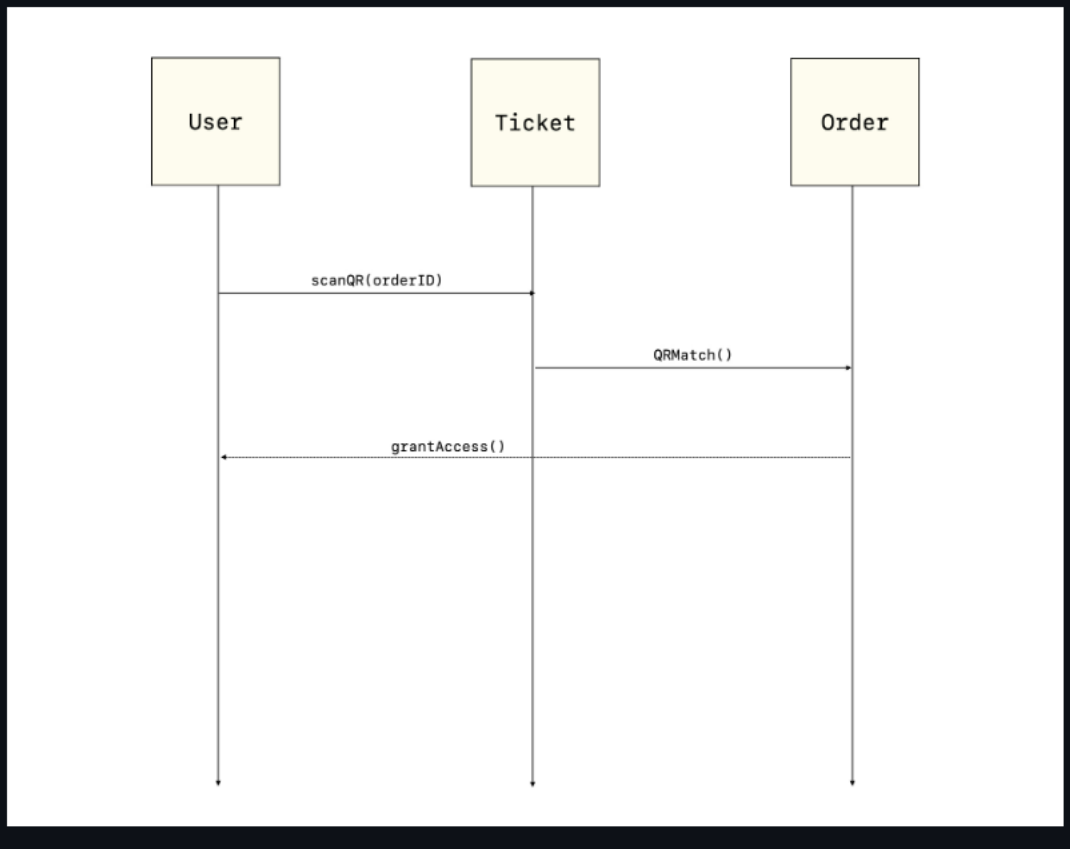
Update Ticket Inventory

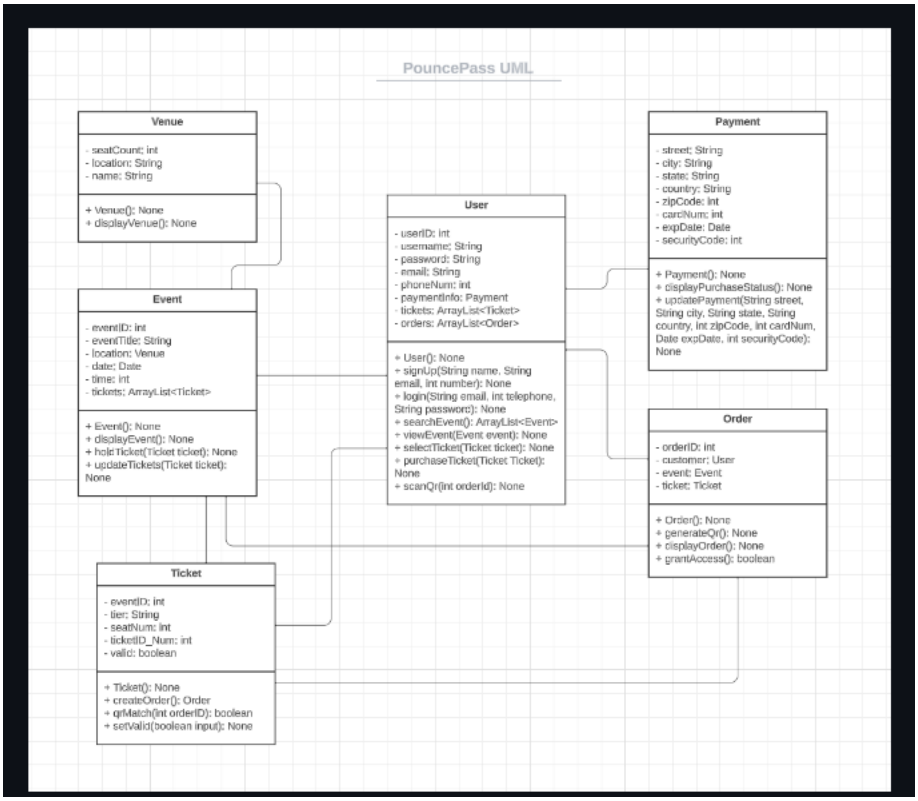


Deliver Tickets



Scan QR Ticket




PouncePass UML
Venue
- seatCount: int
- location: String
- name: String
+ Venue(): None
+ displayVenue(): None
Event
- eventID: int
- eventTitle: String
- location: Venue
- date: Date
- time: int
- tickets: ArrayList<Ticket>
+ Event(): None
+ displayEvent(): None
+ holdTicket(Ticket ticket): None
+ updateTickets(Ticket ticket): None
User
- userID: int
- username: String
- password: String
- email: String
- phoneNum: int
- paymentInfo: Payment
- tickets: ArrayList<Ticket>
- orders: ArrayList<Order>
+ User(): None
+ signUp(String name, String email, int number): None
+ login(String email, int telephone, String password): None
+ searchEvent(): ArrayList<Event>
+ viewEvent(Event event): None
+ selectTicket(Ticket ticket): None
+ purchaseTicket(Ticket Ticket): None
+ scanQr(int orderId): None
Payment
- street: String
- city: String
- state: String
- country: String
- zipCode: Int
- cardNum: int
- expDate: Date
- securityCode: int
+ Payment(): None
+ displayPurchaseStatus(): None
+ updatePayment(String street, String city, String state, String country, int zipCode, int cardNum, Date expDate, int securityCode): None
Order
- orderID: int
- customer: User
- event: Event
- ticket: Ticket
+ Order(): None
+ generateQr(): None
+ displayOrder(): None
+ grantAccess(): boolean
Ticket
- eventID: int
- tier: String
- seatNum: int
- ticketID_Num: int
- valid: boolean
+ Ticket(): None
+ createOrder(): Order
+ qrMatch(int orderID): boolean
+ setValid(boolean input): None