AIU Trips & Events Management System

Instructor : Dr. Islam Elgedawy

Students Names & IDs:

Ahmed Mohamed Ahmed 23101359

Muhammed Hany Hamdy 23102359

Arwa Mohey Hamdy 23101371

Mostafa Khamis abozead 23101484

Mohamed Soliman Mohamed 23101462

Faculty : Computer science and engineering (CSE)

Level : Three

Semester : Fall 2025

Course name : System Analysis

Course code : CSE352

Objectives:

A digital system that helps the university organize and manage trips and events (such as field trips, seminars, conferences, concerts). It allows students to register and receive notifications about updates.

Problems the System Solves

1. **Manual Registration & Booking**
   - Currently, students may need to register for trips/events by filling out paper forms or visiting offices.
   - This is time-consuming, error-prone, and not convenient.
2. **Limited Capacity Management**
   - Hard to control the number of participants in trips/events.
   - Overbooking or underutilization often happens because there is no automated seat control.
3. **Lack of Real-Time Updates**
   - Students often miss important updates (change in timing, location, or cancellation).
   - Organizers cannot notify all students instantly.
4. **No Centralized Event Information**
   - Students don’t have a single platform to view all upcoming trips, seminars, and activities.
   - Information is usually scattered (posters, social media, word of mouth).
5. **Inefficient Ticketing System**
   - No digital tickets, so students rely on paper tickets or just “name lists.”
   - Risk of fraud or confusion at entry points.
6. **Poor Reporting & Analytics**
   - The university cannot easily generate reports on:

        - Number of participants.
        - Revenue collected.
        - Student feedback/satisfaction.
    - This limits planning for future events.
7. **Organizer Limitations**
    - Organizers manually track attendance, and capacity.
    - High risk of human errors (duplicate bookings, lost records).

Core Features:

- **Authentication & Profiles (Login/Create Account).**
- **Event/Trip Management (Add/Edit/Delete).**
- **Booking & Ticketing (Reservation – QR Code for entry).**
- **Notifications (Messages to students upon any update).**
- **Reports & Analytics (Participants count, income, feedback).**

Actors (Users):

1. **Student**
    - **Log in and create a profile.**
    - **Browse the list of trips and events.**
    - **Book and for tickets.**
    - **Receive notifications (time, location, updates).**
2. **Organizer (Event/Trip Organizer)**
    - **Create and manage events/trips.**
    - **Define the number of seats/participants.**
    - **Send alerts to students.**
3. **Admin (University Administrator)**
    - **Manage users (students, organizers).**
    - **Track statistics and reports (number of participants, revenue).**
    - **Control organizer permissions.**

Functional Requirements(F):

F1: Authentication & User Management

- **F1.1 The system shall allow users to register with university email, first name, last name, phone number, faculty, academic year, and password**
- **F1.2 The system shall send email verification links to newly registered users**
- **F1.3 The system shall authenticate users with email and password credentials**
- **F1.4 The system shall support role-based access (Student, Organizer, Administrator)**
- **F1.5 The system shall lock accounts after 5 consecutive failed login attempts**
- **F1.6 The system shall provide password reset functionality via email**
- **F1.7 The system shall enforce password strength requirements**
- **F1.8 The system shall redirect users to role-appropriate dashboards upon login**

F2: Event & Trip Management

- **F2.1 The system shall allow organizers to create new events/trips with details (name, description, date, time, location, capacity)**
- **F2.2 The system shall allow organizers to edit existing event/trip information**
- **F2.3 The system shall allow organizers to delete events/trips**
- **F2.4 The system shall display a list of all upcoming events/trips to students**
- **F2.5 The system shall prevent event capacity from being exceeded**

- **F2.6 The system shall track available seats in real-time**
- **F2.7 The system shall allow organizers to set registration deadlines**
- **F2.8 The system shall categorize events (field trips, seminars, conferences, concerts)**

F3: Booking & Ticketing System

- **F3.1 The system shall allow students to browse available events/trips**
- **F3.2 The system shall allow students to book/reserve seats for events**
- **F3.3 The system shall send digital tickets to students via email**
- **F3.4 The system shall prevent duplicate bookings by the same student for the same event**
- **F3.5 The system shall allow students to view their booking history**
- **F3.6 The system shall validate QR codes at event entry points**

F4: Notification System

- **F4.1 The system shall send notifications to students about new events**
- **F4.2 The system shall notify students of event updates (time, location changes)**
- **F4.3 The system shall send cancellation notifications to registered students**
- **F4.4 The system shall send reminder notifications before event dates**
- **F4.5 The system shall allow organizers to send custom messages to registered participants**

F5: Reports & Analytics

- **F5.1 The system shall generate reports on participant counts per event**
- **F5.2 The system shall calculate and display revenue collected per event**
- **F5.3 The system shall track and display total system statistics**
- **F5.4 The system shall allow administrators to export reports in PDF/CSV formats**
- **F5.5 The system shall collect and display student feedback/ratings for events**
- **F5.6 The system shall generate attendance tracking reports**
- **F5.7 The system shall provide analytics on popular event categories**
- **F5.8 The system shall track organizer performance metrics**

Non-Functional Requirements(NF):

NF1: Performance

- **NF1.1 The system shall load pages within 3 seconds under normal load**
- **NF1.2 The system shall support at least 500 concurrent users**

NF2: Security

- **NF2.1 The system shall encrypt all passwords using industry-standard hashing (bcrypt)**
- **NF2.2 The system shall use JWT tokens for session management**
- **NF2.3 The system shall implement HTTPS for all communications**
- **NF2.4 The system shall validate and sanitize all user inputs to prevent injection attacks**
- **NF2.5 The system shall comply with data protection regulations (GDPR)**
- **NF2.6 The system shall log all authentication attempts for security auditing**

NF3: Reliability & Availability

- **NF3.1 The system shall have 99.5% uptime availability**
- **NF3.2 The system shall perform automated daily backups of the database**

- NF3.3 The system shall recover from failures within 1 hour
- NF3.4 The system shall maintain data integrity during transactions

NF4: Usability

- NF4.1 The system shall have an intuitive, user-friendly interface
- NF4.2 The system shall be responsive and accessible on mobile devices
- NF4.3 The system shall support common web browsers (Chrome, Firefox, Safari, Edge)
- NF4.4 The system shall provide clear error messages to users
- NF4.5 The system shall follow accessibility standards (WCAG 2.1)

NF5: Scalability & Maintainability

- NF5.1 The system shall be containerized using Docker for easy deployment
- NF5.2 The system shall have modular architecture for easy maintenance
- NF5.3 The system shall support horizontal scaling to handle increased load
- NF5.4 The system shall have comprehensive API documentation
- NF5.5 The system shall use version control (Git/GitHub) for code management

Customer Requirements (C)

C1: User Experience Requirements

- C1.1 Students want a single platform to view all university events and trips
- C1.2 Students want instant notifications about event updates
- C1.3 Students want digital tickets accessible on mobile devices
- C1.4 Students want to easily track their booking history

C2: Organizer Requirements

- C2.1 Organizers need automated capacity management to prevent overbooking
- C2.2 Organizers need ability to communicate with all registered participants
- C2.3 Organizers need reports on attendance and revenue
- C2.4 Organizers need easy event creation and management tools

C3: Administrative Requirements

- C3.1 Administrators need centralized user management capabilities
- C3.2 Administrators need comprehensive analytics and reporting
- C3.3 Administrators need to control organizer permissions
- C3.4 Administrators need audit trails for financial transactions
- C3.5 Administrators need system-wide statistics for planning

C4: Operational Requirements

- C4.1 The university needs to eliminate manual paper-based registration
- C4.2 The university needs to reduce administrative overhead
- C4.3 The university needs to improve financial tracking and transparency
- C4.4 The university needs to enhance communication with students
- C4.5 The university needs data-driven insights for future event planning

Developer Requirements (D)

D1: Technical Architecture Requirements

- **D1.1 Frontend must be built using Next.js with TypeScript**
- **D1.2 Backend must be built using Java Spring Boot with java**
- **D1.3 Database must be MongoDB with Mongoose ODM**
- **D1.4 Authentication must use JWT tokens**
- **D1.5 UI must be styled with Tailwind CSS**

D2: Development Tools Requirements

- **D2.1 Code must be version-controlled using Git and GitHub**
- **D2.2 Application must be containerized using Docker**
- **D2.3 API testing must be performed using Postman**
- **D2.4 Database management must use MongoDB Compass**
- **D2.5 UML diagrams must be created using PlantUML**

D3: Code Quality Requirements

- **D3.1 Code must follow TypeScript best practices and typing**
- **D3.2 Code must be modular and follow separation of concerns**
- **D3.3 Code must include error handling and logging**
- **D3.4 API endpoints must be RESTful and well-documented**
- **D3.5 Code must include unit tests for critical functionality**

D4: Integration Requirements

- **D4.1 System must integrate email service for notifications**
- **D4.2 System must generate QR codes using appropriate library**
- **D4.3 System must support file uploads for event images**
- **D4.4 System must provide API endpoints for mobile app integration (future)**

D5: Deployment Requirements

- **D5.1 Application must be deployable using Docker Compose**
- **D5.2 Environment configurations must be managed via .env files**
- **D5.3 Database migrations must be version-controlled**
- **D5.4 System must have separate development, staging, and production environments**
- **D5.5 CI/CD pipeline should be implemented for automated testing and deployment**

Subsystems Define:

Subsystem Name	Subsystem Function	Subsystem Interface
User Management Subsystem	**1. signup(): This method handles the user registration process, collecting the user's email, name, phone number, university ID number, and password.** **2. login(): Authenticates users by verifying their university ID number and password, then establishes a session with role-based permissions.** **3. signout(): Logs out the user and terminates the current session.** **4. resetPassword(): Allows users to reset their password by providing their registered email address and receiving a reset link.** **5. verifyEmail(): Sends verification email to newly registered users and validates the verification token.** **6. viewProfile(): Allows users to view their profile details (name, email, phone number, university ID, role).** **7. updateProfile(): Enables users to update their profile information.** **8. deleteAccount(): Allows users to deactivate or delete their account permanently.** **9. lockAccount(): Automatically locks account after multiple failed login attempts.** **10. assignRole(): Assigns or changes user roles (Student, Organizer, Administrator).**	**Interface IUserRegistration** { Void signup(String email, String name, String phoneNumber, String universityIdNumber, String password); } **Interface IUserAuthentication** { Boolean login(String universityIdNumber, String password); Void signout(String userId); Boolean verifyCredentials(String universityIdNumber, String password); } **Interface IPasswordManagement** { Void resetPassword(String email); Void changePassword(String userId, String oldPassword, String newPassword); } **Interface IEmailVerification** { Void sendVerificationEmail(String email, String verificationToken); Boolean verifyEmail(String verificationToken); } **Interface IAccountSecurity** { Void lockAccount(String userId); Void unlockAccount(String userId); Boolean isAccountLocked(String userId); } **Interface IProfileViewer** { Profile viewProfile(String userId); } **Interface IProfileEditor** { Void updateProfile(String userId, ProfileData updatedData); } **Interface IAccountDeletion** { Void deleteAccount(String userId, String password); Void deactivateAccount(String userId); } **Interface IRoleManager** { Void assignRole(String userId, Role role); Role getUserRole(String userId); } **Interface IPermissionChecker** { Boolean checkPermission(String userId, String action); List<Permission> getUserPermissions(String userId); }
Event & Trip Management Subsystem	**1. createEvent(): Creates a new event/trip with details including name, description, date, time,**	**Interface IEventCreator** { String createEvent(String organizerId, EventDetails eventDetails); Boolean

	location, capacity, price, category, and registration deadline. **2. updateEvent(): Allows organizers to edit existing event/trip information and notifies registered students of changes.** **3. deleteEvent(): Removes an event/trip from the system and handles cancellations and refunds.** **4. viewEventDetails(): Displays comprehensive information about a specific event including available seats.** **5. listEvents(): Retrieves and displays all available events/trips with filtering options.** **6. searchEvents(): Enables users to search for events by name, category, date, or location.** **7. filterEvents(): Filters events by category, date range, price range, or availability.** **8. checkCapacity(): Verifies available seats in real-time before allowing bookings.** **9. updateCapacity(): Adjusts event capacity and updates available seats count.** **10. setRegistrationDeadline(): Defines the last date/time for event registration.**	validateEventDetails(EventDetails eventDetails); } **Interface IEventEditor** { Void updateEvent(String eventId, EventDetails updatedDetails); Boolean canEditEvent(String userId, String eventId); } **Interface IEventRemover** { Void deleteEvent(String eventId); Void cancelEvent(String eventId, String reason); } **Interface IEventViewer** { Event viewEventDetails(String eventId); Boolean isEventVisible(String eventId); } **Interface IEventLister** { List<Event> listEvents(Integer page, Integer limit); List<Event> listEventsByOrganizer(String organizerId); } **Interface IEventSearcher** { List<Event> searchEvents(String searchQuery); List<Event> searchByKeyword(String keyword); } **Interface IEventFilter** { List<Event> filterByCategory(String category); List<Event> filterByDateRange(Date startDate, Date endDate); List<Event> filterByPrice(Double minPrice, Double maxPrice); } **Interface ICapacityChecker** { Boolean checkCapacity(String eventId); Integer getAvailableSeats(String eventId); Boolean hasAvailableSeats(String eventId); } **Interface ICapacityManager** { Void updateCapacity(String eventId, Integer newCapacity); Void reserveSeat(String eventId); Void releaseSeat(String eventId); } **Interface IRegistrationDeadlineManager** { Void setRegistrationDeadline(String eventId, Date deadline); Boolean isRegistrationOpen(String eventId); }
Booking & Ticketing Subsystem	**1. createBooking(): Processes a student's booking request for an event, checking availability and preventing duplicates.**	**Interface IBookingCreator** { Booking createBooking(String studentId, String eventId); Boolean validateBookingRequest(String studentId, String eventId); } **Interface IBookingViewer** {

	2. viewBookingHistory(): Displays all past and upcoming bookings for a student. **3. cancelBooking(): Allows students to cancel their bookings within the allowed timeframe.** **4. sendTicket(): Sends the digital ticket to the student's email address.** **5. downloadTicket(): Allows students to download their ticket as a PDF file.** **6. validateTicket(): Verifies QR code tickets at event entry points.** **7. checkDuplicateBooking(): Prevents the same student from booking the same event multiple times.** **8. updateSeatsCount(): Decrements available seats when booking is confirmed.** **9. processRefund(): Handles refund processing for cancelled bookings.**	List<Booking> viewBookingHistory(String studentId); Booking getBookingDetails(String bookingId); } **Interface IBookingCanceller** { Void cancelBooking(String bookingId); Boolean canCancelBooking(String bookingId); } **Interface IDuplicateBookingChecker** { Boolean checkDuplicateBooking(String studentId, String eventId); Boolean hasActiveBooking(String studentId, String eventId); } **Interface ITicketGenerator** { Ticket generateTicket(String bookingId); String generateQRCode(String bookingId); } **Interface ITicketSender** { Void sendTicket(String bookingId, String email); Boolean sendTicketByEmail(Ticket ticket, String email); } **Interface ITicketDownloader** { File downloadTicket(String bookingId); File generateTicketPDF(Ticket ticket); } **Interface ITicketValidator** { Boolean validateTicket(String qrCode); TicketStatus checkTicketStatus(String qrCode); } **Interface ISeatUpdater** { Void updateSeatsCount(String eventId, Integer change); Void decrementSeats(String eventId); Void incrementSeats(String eventId); } **Interface IRefundProcessor** { Void processRefund(String bookingId); Boolean isRefundEligible(String bookingId); }
Notification Subsystem	**1. sendEmailNotification(): Sends email notifications to users for various events and updates.** **2. sendVerificationEmail(): Sends email verification link to newly registered users.**	**Interface IEmailSender** { Boolean sendEmail(String recipient, String subject, String message); Boolean sendBulkEmail(List<String> recipients, String subject, String message); } **Interface IVerificationEmailSender** { Void

	3. sendBookingConfirmation(): Sends booking confirmation with ticket details to students. **4. sendEventUpdate(): Notifies registered students about event changes (time, location, etc.).** **5. sendCancellationNotice(): Alerts students when an event is cancelled.** **6. sendReminderNotification(): Sends reminder notifications before event dates.** **7. sendCustomMessage(): Allows organizers to send custom messages to event participants.** **8. queueNotification(): Queues notifications for batch processing.** **9. logNotification(): Records all sent notifications in the system.** **10. retryFailedNotification(): Retries sending failed notifications.**	sendVerificationEmail(String email, String verificationToken); String generateVerificationLink(String token); } **Interface IBookingConfirmationSender** { Void sendBookingConfirmation(String bookingId, String email); Void sendBookingDetails(Booking booking, String email); } **Interface IEventUpdateNotifier** { Void sendEventUpdate(String eventId, String message); Void notifyEventChange(String eventId, EventDetails oldDetails, EventDetails newDetails); } **Interface ICancellationNotifier** { Void sendCancellationNotice(String eventId); Void notifyEventCancellation(String eventId, String reason); } **Interface IReminderSender** { Void sendReminderNotification(String eventId); Void scheduleReminder(String eventId, Date reminderTime); } **Interface ICustomMessageSender** { Void sendCustomMessage(String organizerId, String eventId, String message); Boolean canSendCustomMessage(String organizerId, String eventId); } **Interface INotificationQueue** { Void queueNotification(Notification notification); List<Notification> getPendingNotifications(); } **Interface INotificationLogger** { Void logNotification(String notificationId, NotificationStatus status); List<NotificationLog> getNotificationHistory(String userId); } **Interface INotificationRetrier** { Void retryFailedNotification(String notificationId); Integer getRetryCount(String notificationId); }
Reporting & Analytics Subsystem	**1. generateParticipantReport(): Creates reports showing participant counts per event and overall statistics.**	**Interface IParticipantReportGenerator** { Report generateParticipantReport(FilterCriteria criteria); Integer

	2. generateRevenueReport(): Generates financial reports showing revenue collected per event and totals. **3. generateAttendanceReport(): Produces attendance tracking reports based on QR code validations.** **4. generateFeedbackReport(): Compiles student feedback and ratings for events.** **5. generateOrganizerReport(): Creates performance reports for event organizers.** **6. exportReportPDF(): Exports generated reports in PDF format.** **7. exportReportCSV(): Exports generated reports in CSV format.** **8. getSystemStatistics(): Retrieves system-wide statistics and metrics.** **9. analyzeEventTrends(): Analyzes event popularity and attendance trends.** **10. scheduleAutomatedReport(): Schedules reports to be generated and sent automatically.**	getParticipantCount(String eventId); } **Interface IRevenueReportGenerator** { Report generateRevenueReport(DateRange dateRange); Double calculateTotalRevenue(String eventId); } **Interface IAttendanceReportGenerator** { Report generateAttendanceReport(String eventId); Double calculateAttendanceRate(String eventId); } **Interface IFeedbackReportGenerator** { Report generateFeedbackReport(String eventId); Double getAverageRating(String eventId); } **Interface IOrganizerReportGenerator** { Report generateOrganizerReport(String organizerId, DateRange dateRange); OrganizerMetrics getOrganizerMetrics(String organizerId); } **Interface IPDFExporter** { File exportReportPDF(Report report); File convertToPDF(ReportData data); } **Interface ICSVExporter** { File exportReportCSV(Report report); String convertToCSV(ReportData data); } **Interface IStatisticsProvider** { Statistics getSystemStatistics(); Map<String, Object> getMetrics(); } **Interface ITrendAnalyzer** { TrendAnalysis analyzeEventTrends(DateRange dateRange); List<Event> getPopularEvents(Integer limit); } **Interface IReportScheduler** { Void scheduleAutomatedReport(ReportType type, String frequency, String recipient); List<ScheduledReport>

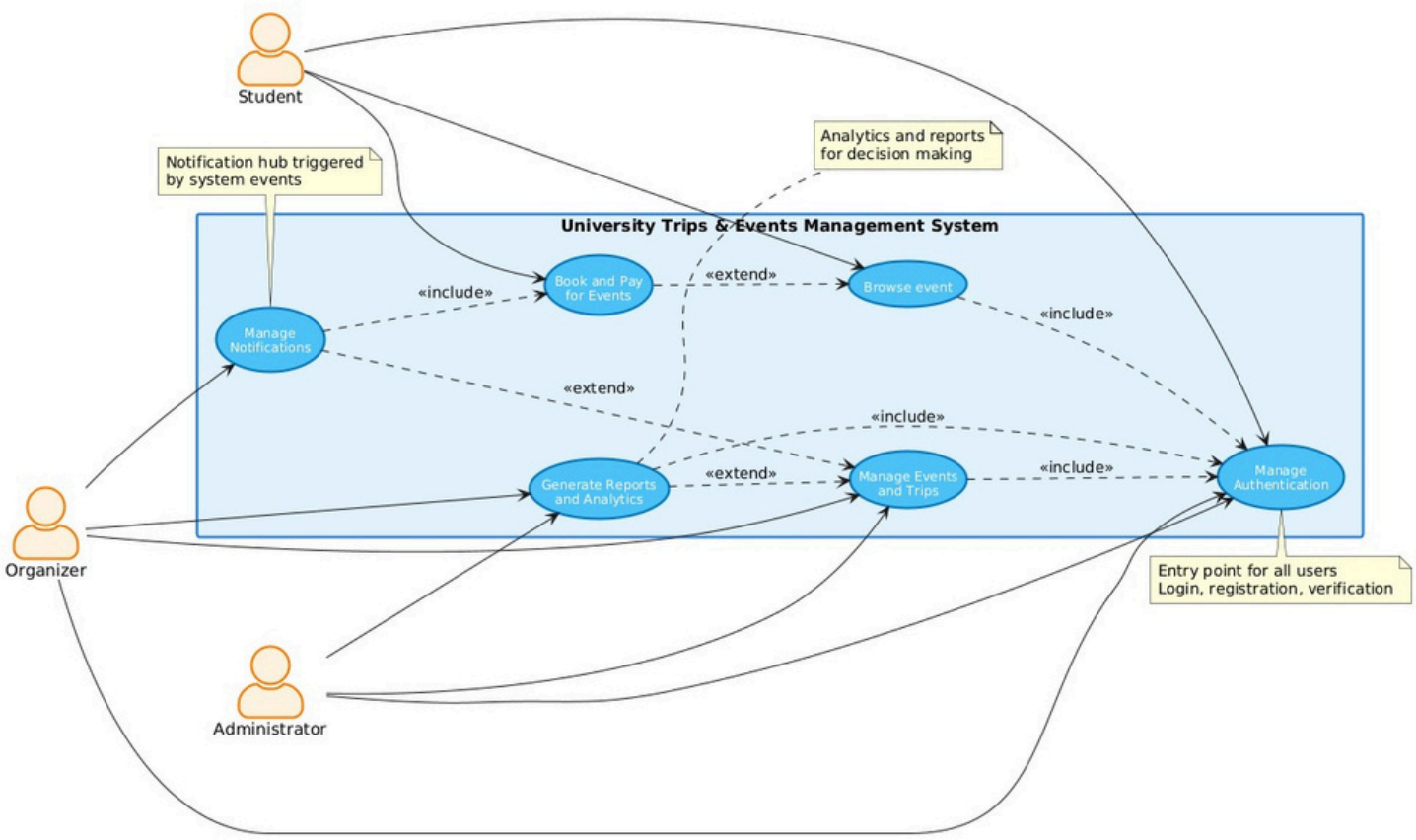
Traceability Matrix

User Requirements	User Management	Event Management	Booking & Ticketing	Notification System
User log in	✓	□	□	□
Input personal information	✓	□	□	□
Edit personal information	✓	□	□	□
Update personal information	✓	□	□	□
Deactivate or delete account	✓	□	□	□
Browse available events	□	✓	□	□
Search for specific events	□	✓	□	□
View event details	□	✓	□	□
Filter events by category	□	✓	□	□
Register for events	□	□	✓	□
View booking history	□	□	✓	□
Cancel booking	□	□	✓	□
View event capacity	□	✓	✓	□
Download digital ticket	□	□	✓	□
Receive QR code ticket	□	□	✓	✓
Validate ticket at	□	□	✓	□

entry				
Receive event notifications	☐	☐	☐	✓
Receive event updates	☐	☐	☐	✓
Receive cancellation alerts	☐	☐	☐	✓
Receive reminder notifications	☐	☐	☐	✓

Use Cases:

Diagram:



Use Case 1: Manage Authentication

Use Case Name: Manage Authentication
Use Case ID: UC-01
Actors: Student, Organizer, Administrator (initiators)
Type: Primary and essential

Description: This use case allows users to log in, register new accounts, reset passwords, and access the system based on their roles.

Main Flow:

i. **The user navigates to the login page**
ii. **The system displays the login form with fields for email and password**
iii. **The user enters their credentials and submits the form**
iv. **The system validates the credentials against the user database**
v. **The system verifies the user's role (Student, Organizer, or Administrator)**
vi. **If credentials are valid, the system authenticates the user and redirects them to the appropriate dashboard based on their role**

Cross Reference: Requirements F1.1-F1.8, User Management Subsystem

Preconditions:

- **User is not currently logged in to the system**
- **Database is accessible and operational**

Postconditions:

- **User is successfully logged in or registered**
- **The user's session is established with appropriate role-based permissions**
- **The user is redirected to the appropriate page (student dashboard, organizer dashboard, or admin dashboard)**

Alternate Scenarios:

AS1: Invalid Credentials

- **If the credentials are invalid, the system displays an error message prompting the user to re-enter their credentials**
- **The system logs the failed attempt**
- **After 5 consecutive failed attempts, the account is locked**

AS2: Account Locked

- **If the account is locked due to multiple failed login attempts, the system displays a message instructing the user to reset their password**
- **The user must use the password reset functionality to unlock the account**

AS3: Password Reset

i. **The user clicks on the "Forgot Password" link**
ii. **The system prompts the user to enter their email address**
iii. **If the email address is valid and exists in the database, the system sends a password reset link to the user's email**
iv. **The user clicks on the reset link (valid for 30 minutes)**
v. **The system displays password reset form**
vi. **The user enters a new password meeting strength requirements**
vii. **The system validates the new password, resets it, and redirects the user to the login page**
viii. **The system sends a confirmation email**

AS4: New User Registration

i. **If the user does not have an account, they click on the "Sign Up" link**
ii. **The system displays the registration form with fields for first name, last name, university email, phone number, faculty, academic year, and password**
iii. **The user fills in all required information and submits the form**
iv. **The system validates the input data and checks for duplicate email addresses**
v. **If validation passes, the system creates the account and sends a verification email to the provided email address**
vi. **The user clicks the verification link in the email to activate their account**
vii. **The system confirms the account activation and redirects the user to the login page**

Interaction Scenario:

Actor Intention	System Responsibility
User opens the application and navigates to the login page	System displays the login form with email and password fields
User enters email and password, then clicks "Login" button	System validates credentials against database
	System verifies user role
	System establishes authenticated session
	System redirects user to appropriate dashboard

Test Requirements:

- **T1.1.1: Validate that the system allows users to log in with valid credentials**
- **T1.1.2: Validate that users can log in from different devices and browsers**
- **T1.1.3: Validate that the system displays an error message for invalid login attempts**
- **T1.1.4: Validate that the system locks the account after 5 consecutive failed login attempts**
- **T1.1.5: Validate that the system redirects users to the appropriate dashboard based on their role**
- **T1.2.1: Validate that new users can successfully register with valid information**
- **T1.2.2: Validate that the system prevents duplicate email addresses**
- **T1.2.3: Validate that users receive a verification email upon registration**
- **T1.2.4: Validate that email verification is required before login**
- **T1.2.5: Validate password strength requirements**
- **T1.3.1: Validate that password reset links are sent to registered emails**
- **T1.3.2: Validate that password reset links expire after 30 minutes**
- **T1.3.3: Validate that passwords can be successfully reset**
- **T1.3.4: Validate that confirmation emails are sent after password reset**

Use Case 2: Manage Events and Trips

Use Case Name: Manage Events and Trips
Use Case ID: UC-02
Actors: Organizer (initiator), Administrator (initiator)
Type: Primary and essential
Description: This use case allows organizers and administrators to create, update, delete, and manage events and trips including capacity control.

Main Flow:

i. **The organizer logs into the system**
ii. **The system displays the organizer dashboard**
iii. **The organizer navigates to the "Manage Events" section**
iv. **The organizer clicks "Create New Event"**
v. **The system displays the event creation form**
vi. **The organizer enters event details (name, description, date, time, location, capacity, price, category, registration deadline)**
vii. **The organizer optionally uploads event images**
viii. **The organizer submits the form**
ix. **The system validates the input data**
x. **The system creates the event and stores it in the database**
xi. **The system displays a success message and the event appears in the events list**

Cross Reference: Requirements F2.1-F2.8, Event & Trip Management Subsystem

Preconditions:

- **Organizer/Administrator is logged into the system**
- **User has appropriate permissions to manage events**
- **Database is accessible**

Postconditions:

- **Event is successfully created/updated/deleted in the database**
- **Event information is available to students for browsing and booking**
- **Notifications are sent to relevant users (if event is updated/cancelled)**
- **Capacity tracking is initialized for the event**

Alternate Scenarios:

AS1: Edit Existing Event

i. **The organizer selects an existing event from the events list**
ii. **The system displays the event details in editable form**
iii. **The organizer modifies the desired fields**
iv. **The organizer submits the changes**
v. **The system validates the updates**
vi. **The system updates the event in the database**
vii. **The system sends update notifications to all registered students**
viii. **The system displays a success message**

AS2: Delete Event

i. **The organizer selects an event to delete**
ii. **The system displays a confirmation dialog**
iii. **The organizer confirms the deletion**
iv. **The system checks if students are registered for the event**
v. **If students are registered, the system sends cancellation notifications and processes refunds**
vi. **The system deletes the event from the database**
vii. **The system displays a success message**

AS3: View Event Details and Statistics

i. **The organizer selects an event to view**
ii. **The system displays comprehensive event details including:**
    - **Number of registered participants**
    - **Available seats**
    - **Revenue collected**
    - **Registration list**
    - **Booking timeline**
iii. **The organizer can export this data as needed**

AS4: Invalid Event Data

- **If the organizer enters invalid data (e.g., past date, negative capacity, invalid price), the system displays validation errors**
- **The organizer corrects the data and resubmits**

AS5: Capacity Reached

- **When an event reaches maximum capacity, the system automatically closes registration**
- **The system displays "Event Full" status to students**
- **Organizers can increase capacity if needed**

Interaction Scenario:

Actor Intention	System Responsibility
Organizer navigates to "Create New Event"	System displays event creation form with all required fields
Organizer fills in event details and uploads image	System validates input in real-time
Organizer clicks "Submit"	System validates all data
	System creates event record in database
	System initializes capacity tracking
	System displays success message
	System makes event visible to students

Test Requirements:

- **T2.1.1: Validate that organizers can create events with all required information**
- **T2.1.2: Validate that event images can be uploaded successfully**
- **T2.1.3: Validate that the system prevents creation of events with past dates**
- **T2.1.4: Validate that capacity must be a positive number**
- **T2.2.1: Validate that organizers can edit existing events**
- **T2.2.2: Validate that event updates trigger notifications to registered students**
- **T2.2.3: Validate that editing preserves existing bookings**
- **T2.3.1: Validate that organizers can delete events**
- **T2.3.2: Validate that deleting events with registrations triggers refunds**
- **T2.3.3: Validate that cancellation notifications are sent**
- **T2.4.1: Validate that events are categorized correctly**
- **T2.4.2: Validate that registration closes when capacity is reached**
- **T2.4.3: Validate that registration deadlines are enforced**

Use Case 3: Book for Events

Use Case Name: Book for Events
Use Case ID: UC-03
Actors: Student (initiator)
Type: Primary and essential
Description: This use case allows students to browse available events, make reservations and receive digital tickets.

Main Flow:

i. **The student logs into the system**

ii. **The system displays the student dashboard with upcoming events**
iii. **The student browses available events/trips**
iv. **The student selects an event to view details**
v. **The system displays comprehensive event information (description, date, location, price, available seats)**
vi. **The student clicks "Book Now"**
vii. **The system checks seat availability**
viii. **The system displays booking confirmation page with event summary and total price**
ix. **The student confirms the booking**
x. **The system generates a unique QR code ticket**
xi. **The system sends the digital ticket to the student's email**
xii. **The system displays success message with ticket download option**
xiii. **The system updates available seats count**

Cross Reference: Requirements F3.1-F3.8, Booking

Preconditions:

- **Student is logged into the system**
- **Event has available seats**
- **Event registration deadline has not passed**
- **Student has not already booked the same event**

Postconditions:

- **Booking is recorded in the database**
- **Available seats count is decremented**
- **Digital ticket with QR code is generated and sent to student**
- **Booking confirmation notification is sent to student**
- **Revenue is tracked for the event**

Alternate Scenarios:

AS1: Event Full

- **If no seats are available when student tries to book, the system displays "Event Full" message**
- **The system offers option to join a waiting list (if implemented)**
- **No booking is created**

AS2: Duplicate Booking

- **If student has already booked the same event, the system displays an error message**
- **The system prevents duplicate booking**
- **Student is redirected to their booking history**

AS3: Registration Deadline Passed

- **If student attempts to book after registration deadline, the system displays message indicating registration is closed**
- **No booking is allowed**

AS4: View Booking History

i. **Student navigates to "My Bookings" section**
ii. **System displays all past and upcoming bookings**
iii. **Student can view ticket details and QR codes**
iv. **Student can download tickets again if needed**

AS5: Cancel Booking

i. **Student selects a booking to cancel**
ii. **System checks cancellation policy (if within allowed timeframe)**
iii. **Student confirms cancellation**
iv. **System processes refund (if applicable)**
v. **System frees up the seat for other students**
vi. **System sends cancellation confirmation**

Interaction Scenario:

Actor Intention	System Responsibility
Student browses events list	System displays all available events with filters
Student selects an event	System displays detailed event information and booking button
Student clicks "Book Now"	System checks availability
	System displays booking confirmation page
Student confirms booking	
	System generates unique QR code ticket
	System decrements available seats
	System sends digital ticket via email
	System displays success message

Test Requirements:

- **T3.1.1: Validate that students can browse available events**
- **T3.1.2: Validate that event details are displayed correctly**
- **T3.1.3: Validate that booking button is disabled when event is full**
- **T3.2.1: Validate that system checks seat availability before booking**
- **T3.2.2: Validate that duplicate bookings are prevented**
- **T3.2.3: Validate that registration deadline is enforced**

- **T3.3.1: Validate that unique QR codes are generated**
- **T3.3.2: Validate that digital tickets are sent via email**
- **T3.3.3: Validate that tickets can be downloaded and viewed**
- **T3.4.1: Validate that available seats are updated correctly**
- **T3.4.2: Validate that booking history is maintained**
- **T3.5.1: Validate that QR codes can be validated at entry**

Use Case 4: Manage Notifications

Use Case Name: Manage Notifications
Use Case ID: UC-04
Actors: Student (receiver), Organizer (initiator), System (initiator)
Type: Primary and essential
Description: This use case manages all system notifications including event updates, reminders and custom messages from organizers.

Main Flow:

i. **An event triggers a notification (e.g., new event created, event updated, booking confirmed)**
ii. **The system identifies the target recipients based on the notification type**
iii. **The system generates the notification content from templates**
iv. **The system queues the notification for delivery**
v. **The system sends the notification via email**
vi. **The system logs the notification in the database**
vii. **The system marks the notification as sent**
viii. **Recipients receive the notification in their email**

Cross Reference: Requirements F4.1-F4.6, Notification Subsystem

Preconditions:

- **Email service is configured and operational**
- **Notification templates are defined**
- **User email addresses are valid**
- **Database is accessible**

Postconditions:

- **Notification is successfully sent to intended recipients**
- **Notification is logged in the system**
- **Users are informed about relevant events, updates, or actions**
- **Notification delivery status is tracked**

Alternate Scenarios:

AS1: Organizer Sends Custom Message

i. **Organizer navigates to event details**
ii. **Organizer selects "Send Message to Participants"**
iii. **System displays message composition interface**
iv. **Organizer writes custom message**
v. **Organizer clicks "Send"**

vi. **System sends message to all registered students for that event**
vii. **System confirms message delivery**

AS2: Event Update Notification

i. **Organizer updates event details (time, location, or other information)**
ii. **System detects the change**
iii. **System automatically generates update notification**
iv. **System sends notification to all registered students**
v. **Notification includes old and new information for clarity**

AS3: Event Cancellation Notification

i. **Organizer cancels an event**
ii. **System generates cancellation notification**
iii. **System sends notification to all registered students**
iv. **Notification includes refund information**
v. **System processes refunds automatically**

AS4: Reminder Notifications

i. **System runs scheduled job to check upcoming events**
ii. **System identifies events within reminder threshold (e.g., 24 hours before)**
iii. **System generates reminder notifications**
iv. **System sends reminders to registered students**
v. **Reminder includes event details and ticket information**

AS5: Email Delivery Failure

- **If email delivery fails, the system logs the failure**
- **The system retries delivery up to 3 times**
- **If all retries fail, the system marks notification as failed**
- **Administrator is notified of persistent delivery issues**

AS6: Notification Preferences

i. **Student navigates to notification settings**
ii. **System displays notification preference options**
iii. **Student selects which types of notifications to receive**
iv. **System saves preferences**
v. **System respects preferences for future notifications**

Interaction Scenario:

Actor Intention	System Responsibility
System detects booking confirmation	System identifies student email address
	System generates confirmation message from template
	System includes booking details and QR code
	System sends email notification
	System logs notification in database
Student receives email	Email contains booking confirmation and ticket
Organizer wants to update participants	System provides message composition interface
Organizer writes and sends message	System delivers message to all registered students

Test Requirements:

- **T4.1.1: Validate that new event notifications are sent to all students**
- **T4.1.2: Validate that notification content is accurate and complete**
- **T4.2.1: Validate that event update notifications are sent to registered students only**
- **T4.2.2: Validate that update notifications include old and new information**
- **T4.3.1: Validate that cancellation notifications are sent promptly**
- **T4.3.2: Validate that cancellation includes refund information**
- **T4.4.1: Validate that confirmations include digital ticket**
- **T4.5.1: Validate that reminder notifications are sent 24 hours before events**
- **T4.5.2: Validate that reminders include event details**
- **T4.6.1: Validate that organizers can send custom messages**
- **T4.6.2: Validate that custom messages reach all registered participants**
- **T4.7.1: Validate that email delivery failures are logged**
- **T4.7.2: Validate that failed deliveries are retried**
- **T4.8.1: Validate that students can set notification preferences**
- **T4.8.2: Validate that preferences are respected**

Use Case 5: Generate Reports and Analytics

Use Case Name: Generate Reports and Analytics
Use Case ID: UC-05
Actors: Administrator (initiator), Organizer (initiator)
Type: Primary and essential
Description: This use case allows administrators and organizers to generate various reports, view analytics dashboards, and export data for decision-making and planning.

Main Flow:

i. **The administrator/organizer logs into the system**
ii. **The system displays the dashboard with analytics overview**
iii. **The user navigates to "Reports & Analytics" section**
iv. **The system displays available report types and filters**
v. **The user selects report type (participant count, revenue, attendance, feedback, etc.)**
vi. **The user sets filter criteria (date range, event category, specific event, organizer)**
vii. **The user clicks "Generate Report"**
viii. **The system retrieves relevant data from the database**
ix. **The system processes and aggregates the data**
x. **The system displays the report with visualizations (charts, graphs, tables)**
xi. **The user can export the report in PDF or CSV format**

Cross Reference: Requirements F5.1-F5.8, Reporting & Analytics Subsystem

Preconditions:

- **Administrator/Organizer is logged into the system**
- **User has appropriate permissions to view reports**
- **Database contains relevant data**
- **Database is accessible**

Postconditions:

- **Report is successfully generated and displayed**
- **User gains insights from analytics**
- **Report can be exported for external use**
- **Report generation is logged for audit purposes**

Alternate Scenarios:

AS1: Participant Count Report

i. **User selects "Participant Count Report"**
ii. **User sets date range and optional event filters**
iii. **System generates report showing:**
    - **Total participants per event**
    - **Trends over time**
    - **Breakdown by event category**
    - **Comparison with capacity**
iv. **System displays data in table and chart formats**

AS2: Revenue Report

i. **User selects "Revenue Report"**
ii. **User sets date range and optional filters**
iii. **System generates report showing:**
    - **Total revenue collected**
    - **Revenue per event**
    - **Revenue trends over time**
    - **Revenue by category**

iv. **System displays financial visualizations**

AS3: Attendance Tracking Report

i. **User selects "Attendance Report"**
ii. **User selects specific event or date range**
iii. **System generates report showing:**
   - **Number of tickets validated (QR scans)**
   - **No-show rate**
   - **Attendance percentage**
   - **Time-based attendance patterns**
iv. **System can export attendance list**

AS4: Feedback and Ratings Report

i. **User selects "Feedback Report"**
ii. **User selects event or date range**
iii. **System generates report showing:**
   - **Average ratings per event**
   - **Student feedback comments**
   - **Sentiment analysis**
   - **Areas for improvement**
iv. **System displays feedback trends**

AS5: Organizer Performance Report

i. **Administrator selects "Organizer Performance"**
ii. **Administrator sets date range**
iii. **System generates report showing:**
   - **Number of events created per organizer**
   - **Average attendance rates**
   - **Revenue generated**
   - **Student satisfaction ratings**
   - **Response time to issues**
iv. **System provides comparative analytics**

AS6: System-Wide Statistics Dashboard

i. **User views main analytics dashboard**
ii. **System displays real-time statistics:**
   - **Total registered users**
   - **Active events**
   - **Total bookings**
   - **Total revenue**
   - **Upcoming events**
   - **Popular event categories**
iii. **Dashboard auto-refreshes periodically**

AS7: Export Report

i. **After generating any report, user clicks "Export"**
ii. **System prompts user to select format (PDF or CSV)**

iii. **User selects format**
iv. **System generates downloadable file**
v. **System initiates file download**
vi. **Report is saved to user's device**

AS8: No Data Available

- **If filters return no data, system displays "No data available for selected criteria"**
- **System suggests adjusting filters**
- **No report is generated**

AS9: Schedule Automated Reports

i. **Administrator selects "Schedule Report"**
ii. **System displays scheduling interface**
iii. **Administrator sets report type, filters, and frequency (daily, weekly, monthly)**
iv. **System saves scheduled report**
v. **System automatically generates and emails reports based on schedule**

Interaction Scenario:

Actor Intention	System Responsibility
Administrator navigates to Reports section	System displays report types and filter options
Administrator selects "Revenue Report" and date range	System displays filter interface
Administrator clicks "Generate Report"	System queries database for relevant data
	System processes and aggregates financial data
	System generates visualizations (charts, graphs)
	System displays comprehensive revenue report
Administrator clicks "Export as PDF"	System generates PDF file
	System initiates download

Test Requirements:

- **T5.1.1: Validate that participant count reports are accurate**
- **T5.1.2: Validate that reports can be filtered by date range**
- **T5.1.3: Validate that reports can be filtered by event category**
- **T5.2.1: Validate that revenue calculations are correct**
- **T5.2.2: Validate that revenue reports show proper breakdowns**

- **T5.3.1: Validate that system-wide statistics are accurate**
- **T5.3.2: Validate that dashboard displays real-time data**
- **T5.4.1: Validate that reports can be exported as PDF**
- **T5.4.2: Validate that reports can be exported as CSV**
- **T5.4.3: Validate that exported files contain complete data**
- **T5.5.1: Validate that feedback reports aggregate ratings correctly**
- **T5.5.2: Validate that feedback comments are displayed properly**
- **T5.6.1: Validate that attendance tracking is accurate**
- **T5.6.2: Validate that no-show rates are calculated correctly**
- **T5.7.1: Validate that organizer performance metrics are accurate**
- **T5.7.2: Validate that comparative analytics work properly**
- **T5.8.1: Validate that reports handle empty datasets gracefully**
- **T5.8.2: Validate that large datasets are paginated properly**
- **T5.9.1: Validate that scheduled reports run automatically**
- **T5.9.2: Validate that scheduled reports are delivered via email**

User Story:

Epic 1: User Authentication & Account Management

US1.1: Student Registration

As a student

I want to register for an account using my university email

So that I can access the event management system

Acceptance Criteria:

- **Registration form includes: first name, last name, university email, phone, faculty, academic year, password**
- **Password must meet strength requirements (min 8 chars, uppercase, lowercase, number, special char)**
- **System sends email verification link**
- **Account is inactive until email is verified**
- **System prevents duplicate email registrations**

Priority: High

Story Points: 5

US1.2: User Login

As a registered user

I want to log in with my credentials

So that I can access my personalized dashboard

Acceptance Criteria:

- **Login with email and password**
- **System validates credentials**
- **Account locks after 5 failed attempts**
- **Redirect to role-appropriate dashboard**

- **Session maintained with JWT token**

Priority: High
Story Points: 3

US1.3: Password Reset
As a user who forgot my password
I want to reset my password via email
So that I can regain access to my account
Acceptance Criteria:

- **"Forgot Password" link available on login page**
- **System sends reset link to registered email**
- **Reset link expires after 30 minutes**
- **User can set new password meeting strength requirements**
- **Confirmation email sent after successful reset**

Priority: Medium
Story Points: 3

US1.4: Profile Management
As a logged-in user
I want to view and edit my profile information
So that I can keep my details up to date
Acceptance Criteria:

- **View current profile information**
- **Edit name, phone number, faculty, academic year**
- **Cannot change email address**
- **Changes saved successfully**
- **Confirmation message displayed**

Priority: Low
Story Points: 2

Epic 2: Event Discovery & Management
US2.1: Browse Events
As a student
I want to browse all available events
So that I can discover interesting activities
Acceptance Criteria:

- **Display list of upcoming events**
- **Show event name, date, location, price, available seats**
- **Events sorted by date (nearest first)**
- **Pagination for large lists**

- **Clear indication of full events**
  Priority: High
  Story Points: 3

US2.2: Search and Filter Events
As a student
I want to search and filter events
So that I can quickly find events that interest me
Acceptance Criteria:

- **Search by event name or description**
- **Filter by category (trips, seminars, conferences, concerts)**
- **Filter by date range**
- **Filter by price range**
- **Filter by availability**
- **Search results update in real-time**
  Priority: Medium
  Story Points: 5

US2.3: View Event Details
As a student
I want to view detailed information about an event
So that I can decide whether to book it
Acceptance Criteria:

- **Display full event description**
- **Show date, time, location, price**
- **Display available seats count**
- **Show registration deadline**
- **Display event category**
- **Show event image (if available)**
- **"Book Now" button visible if seats available**
  Priority: High
  Story Points: 3

US2.4: Create Event
As an organizer
I want to create a new event
So that students can discover and register for it
Acceptance Criteria:

- **Form includes all required fields (name, description, date, time, location, capacity, price, category)**
- **Optional event image upload**

- **Set registration deadline**
- **Validate all inputs (no past dates, positive capacity, valid price)**
- **Event saved to database**
- **Event immediately visible to students**
- **Confirmation message displayed**

Priority: High

Story Points: 8

US2.5: Edit Event

As an organizer

I want to edit existing event details

So that I can update information or fix errors

Acceptance Criteria:

- **Load existing event data into form**
- **Allow modification of all fields except bookings count**
- **Validate changes**
- **Update database**
- **Send notification to registered students**
- **Cannot reduce capacity below current bookings**
- **Success message displayed**

Priority: High

Story Points: 5

US2.6: Delete/Cancel Event

As an organizer

I want to cancel an event

So that students know it won't occur and receive refunds

Acceptance Criteria:

- **Confirmation dialog before deletion**
- **Check for existing bookings**
- **Process refunds for all bookings**
- **Send cancellation notifications**
- **Remove event from active listings**
- **Maintain record for reporting purposes**

Priority: Medium

Story Points: 5

Epic 3: Booking & Ticketing

US3.1: Book Event

As a student

I want to book a seat for an event

So that I can attend it
Acceptance Criteria:

- **Check seat availability before booking**
- **Prevent duplicate bookings for same event**
- **Verify registration deadline not passed**
- **Create booking record**
- **Display booking summary**

Priority: High
Story Points: 8

US3.2: View Booking History
As a student
I want to view my past and upcoming bookings
So that I can track my event participation
Acceptance Criteria:

- **Display all bookings (past and future)**
- **Show event name, date, booking status**
- **Filter by status (upcoming, past, cancelled)**
- **Access tickets for upcoming events**

Priority: Medium
Story Points: 3

US3.3: Cancel Booking
As a student
I want to cancel my booking
So that I can get a refund if I can't attend
Acceptance Criteria:

- **Select booking to cancel**
- **Check cancellation policy/deadline**
- **Confirmation dialog**
- **Process refund**
- **Free up seat for other students**
- **Send cancellation confirmation email**
- **Update booking status to cancelled**

Priority: Medium
Story Points: 5

US3.4: Generate and Receive QR Ticket
As a student who completed booking
I want to receive a QR code ticket

So that I can use it for event entry
Acceptance Criteria:

- **Generate unique QR code upon booking success**
- **QR code contains booking ID and validation data**
- **Send ticket via email immediately**
- **Include event details in email**
- **Allow ticket download as PDF**
- **QR code clearly visible and scannable**

Priority: High
Story Points: 5

US3.5: Validate Ticket at Entry
As an organizer
I want to scan and validate QR tickets
So that I can verify attendees and prevent fraud
Acceptance Criteria:

- **Scan QR code using mobile device or scanner**
- **Validate ticket against database**
- **Display booking details (student name, event)**
- **Mark ticket as used (prevent reuse)**
- **Show validation result (valid/invalid/already used)**
- **Track attendance in real-time**

Priority: High
Story Points: 5

Epic 5: Notifications & Communication
US5.1: Receive Event Notifications
As a student
I want to receive notifications about new events
So that I don't miss interesting opportunities
Acceptance Criteria:

- **Email sent when new event is created**
- **Include event details and link**
- **Filter by student preferences (future enhancement)**
- **Notification delivered within 1 minute**

Priority: Medium
Story Points: 3

US5.2: Receive Event Updates
As a registered student
I want to receive notifications about event changes

So that I stay informed about updates
Acceptance Criteria:

- **Email sent when event details change**
- **Include old and new information**
- **Clear indication of what changed**
- **Link to updated event details**

Priority: High
Story Points: 3

US5.3: Receive Event Reminders
As a registered student
I want to receive reminder before the event
So that I don't forget to attend
Acceptance Criteria:

- **Reminder sent 24 hours before event**
- **Include event details and ticket**
- **Include QR code for easy access**
- **Option to add to calendar**

Priority: Medium
Story Points: 3

US5.4: Send Custom Messages to Participants
As an organizer
I want to send messages to all registered participants
So that I can communicate important information
Acceptance Criteria:

- **Message composition interface**
- **Send to all registered students for specific event**
- **Preview before sending**
- **Confirmation of successful delivery**
- **Message log maintained**

Priority: Low
Story Points: 3

Epic 6: Reporting & Analytics
US6.1: View Analytics Dashboard
As an organizer/administrator
I want to view an analytics dashboard
So that I can monitor system performance
Acceptance Criteria:

- **Display total registered users**
- **Show active events count**
- **Display total bookings**
- **Show total revenue**
- **Display popular event categories**
- **Real-time data updates**
- **Visual charts and graphs**

Priority: Medium
Story Points: 5

US6.2: Generate Participant Report
As an organizer
I want to see participant statistics
So that I can understand event popularity
Acceptance Criteria:

- **Show participant count per event**
- **Display capacity utilization percentage**
- **Filter by date range**
- **Filter by event category**
- **Export as PDF/CSV**

Priority: Medium
Story Points: 5

US6.3: Generate Attendance Report
As an organizer
I want to track actual attendance
So that I can measure event success
Acceptance Criteria:

- **Show tickets validated (QR scans)**
- **Calculate attendance rate**
- **Display no-show rate**
- **Compare with booking count**
- **Export attendee list**

Priority: Low
Story Points: 3

Epic 7: System Administration
US7.1: Manage User Accounts
As an administrator
I want to manage user accounts
So that I can maintain system integrity
Acceptance Criteria:

- View all user accounts
- Search and filter users
- Activate/deactivate accounts
- Delete accounts
- View user activity logs
- Unlock locked accounts

  Priority: Medium

  Story Points: 5

US7.2: Manage Permissions

As an administrator

I want to assign and modify user roles

So that I can control access levels

Acceptance Criteria:

- View current user role
- Assign role (Student, Organizer, Administrator)
- Change existing roles
- Confirmation required for role changes
- Audit log of role changes

  Priority: Low

  Story Points: 3

US7.3: View System Logs

As an administrator

I want to view system logs and security events

So that I can monitor for issues and security threats

Acceptance Criteria:

- Display authentication logs
- Show system errors
- Display security events
- Filter by date range
- Filter by event type
- Export logs

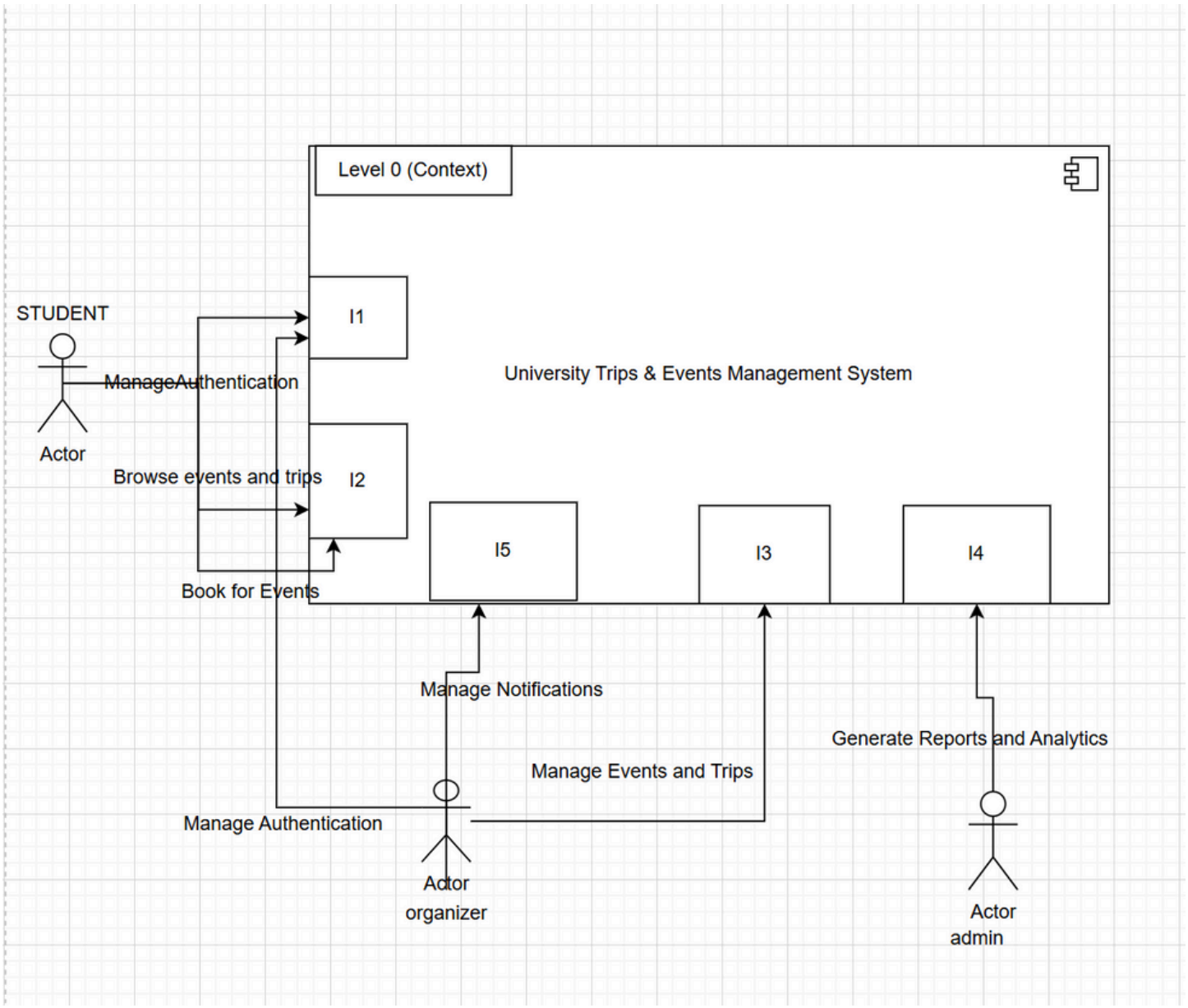
  Priority: Low

  Story Points: 3

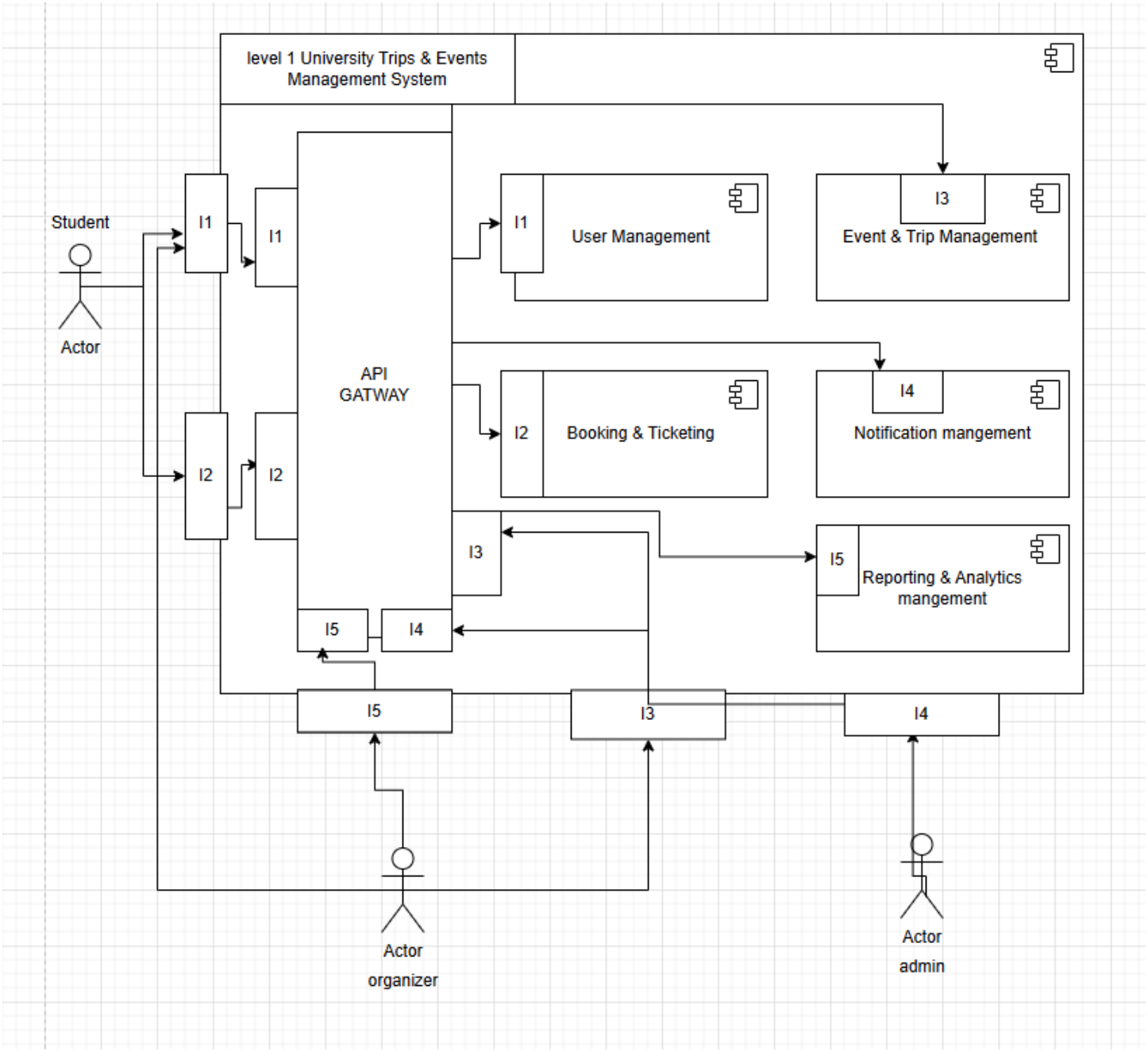
Design diagrams:

Architecture Diagram:

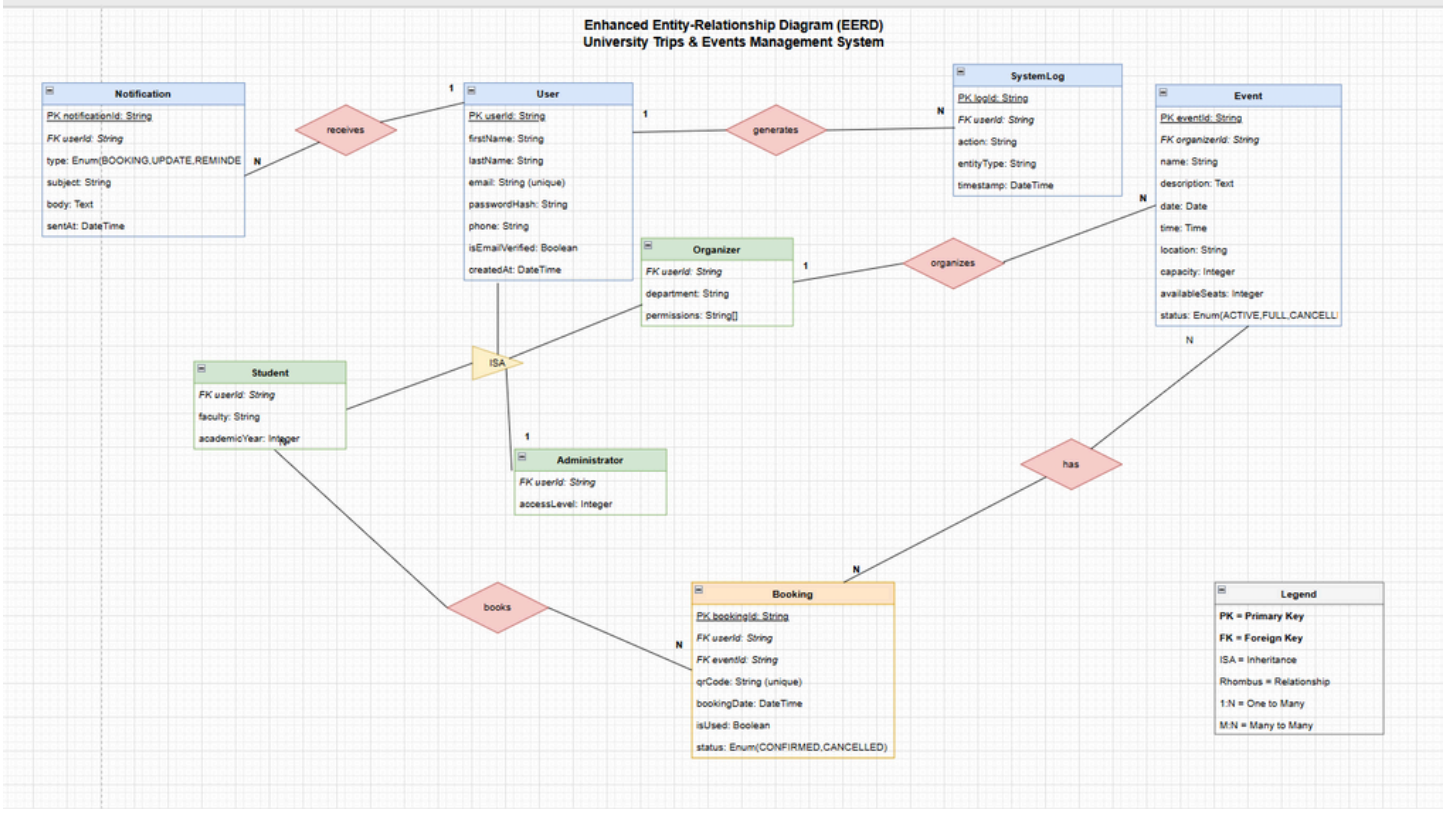
Level 0:



Level 1:

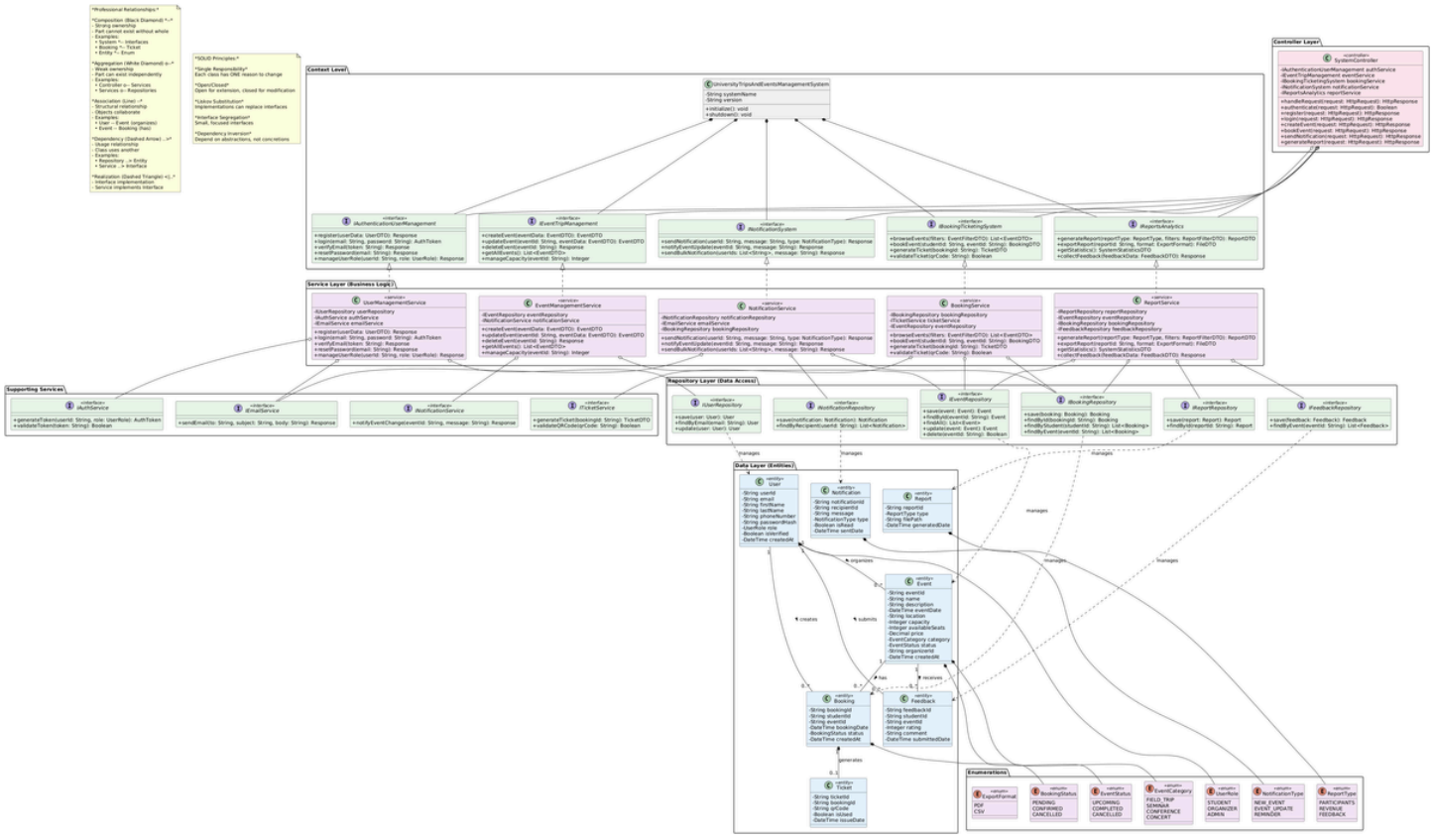

level 1 University Trips & Events Management System
Student
Actor
I1
I1
I2
I2
API GATWAY
I1
User Management
I3
Event & Trip Management
I2
Booking & Ticketing
I4
Notification mangement
I3
I5
Reporting & Analytics mangement
I5
I4
I5
I3
I4
Actor organizer
Actor admin

ERD Diagram:

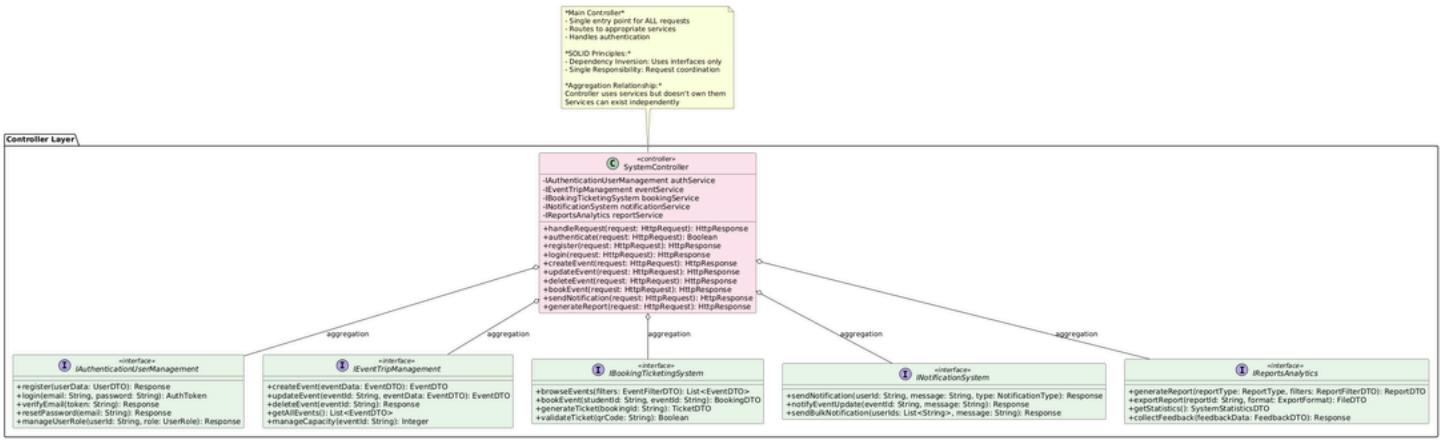


Class Diagram:

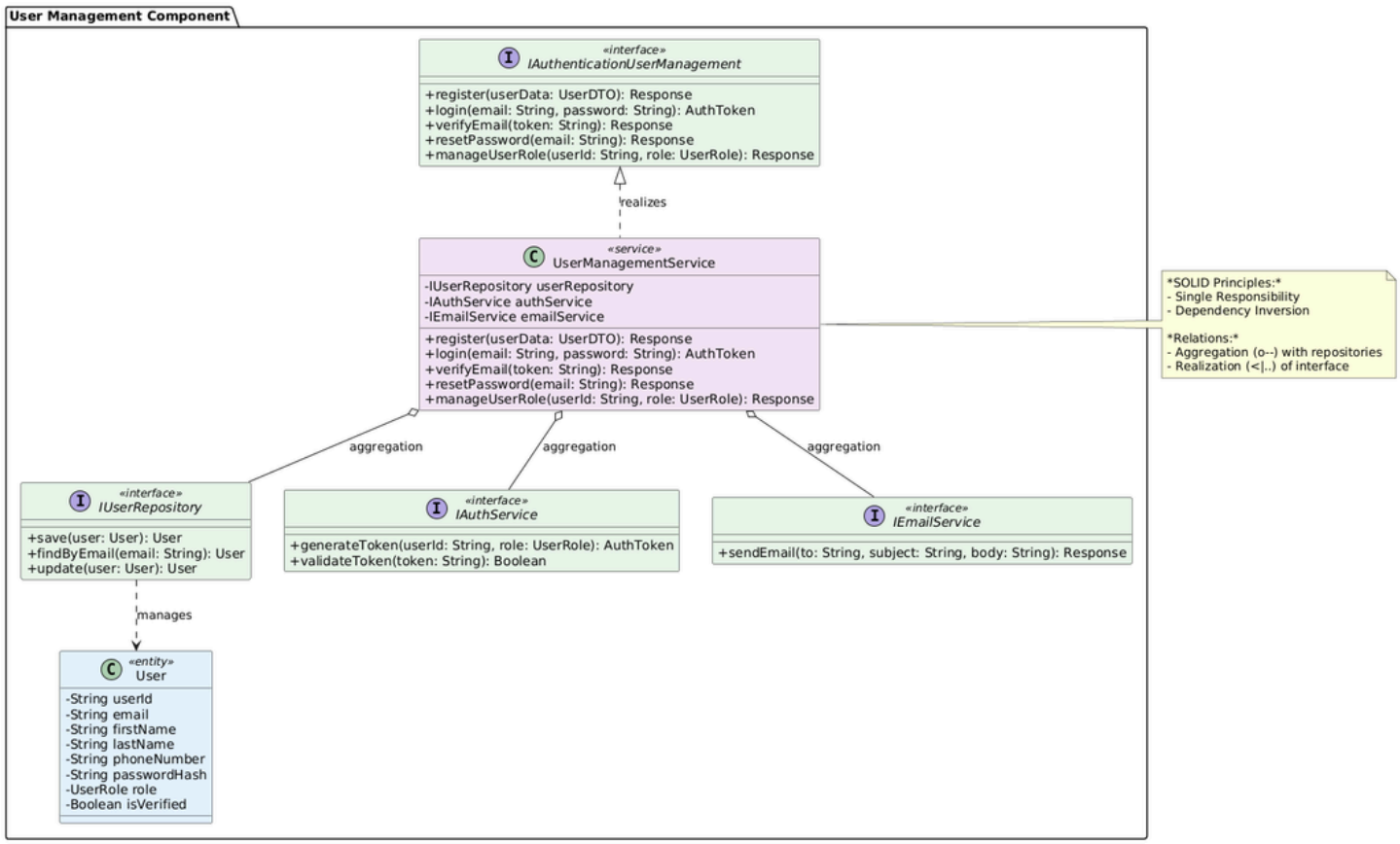
Full System:



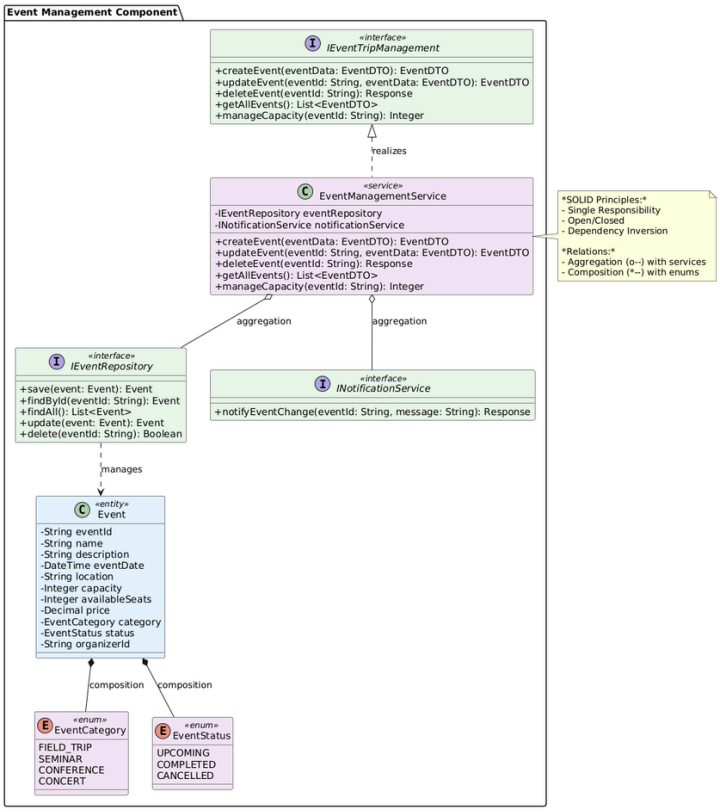
Controller:



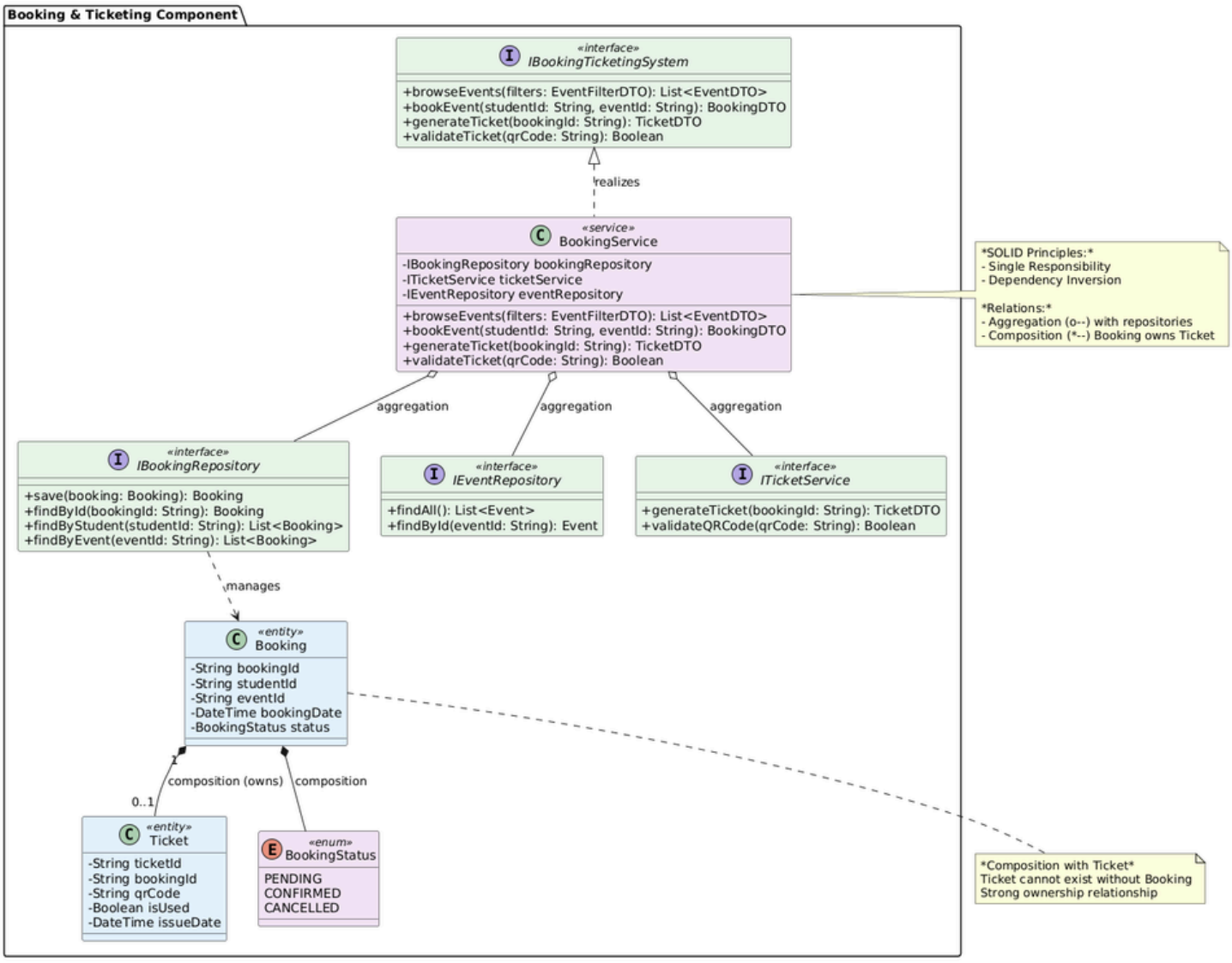
User management:



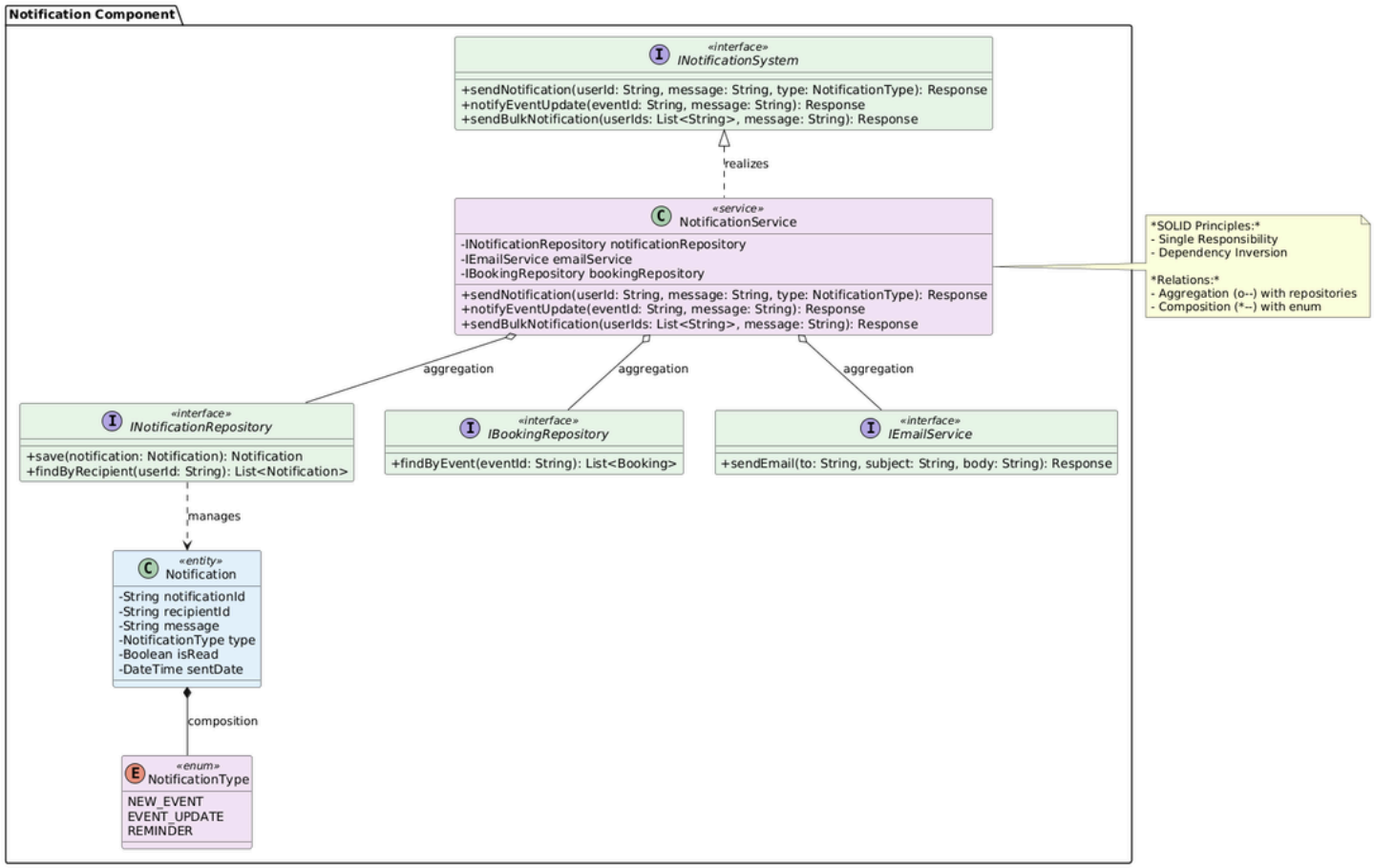
Event Management:



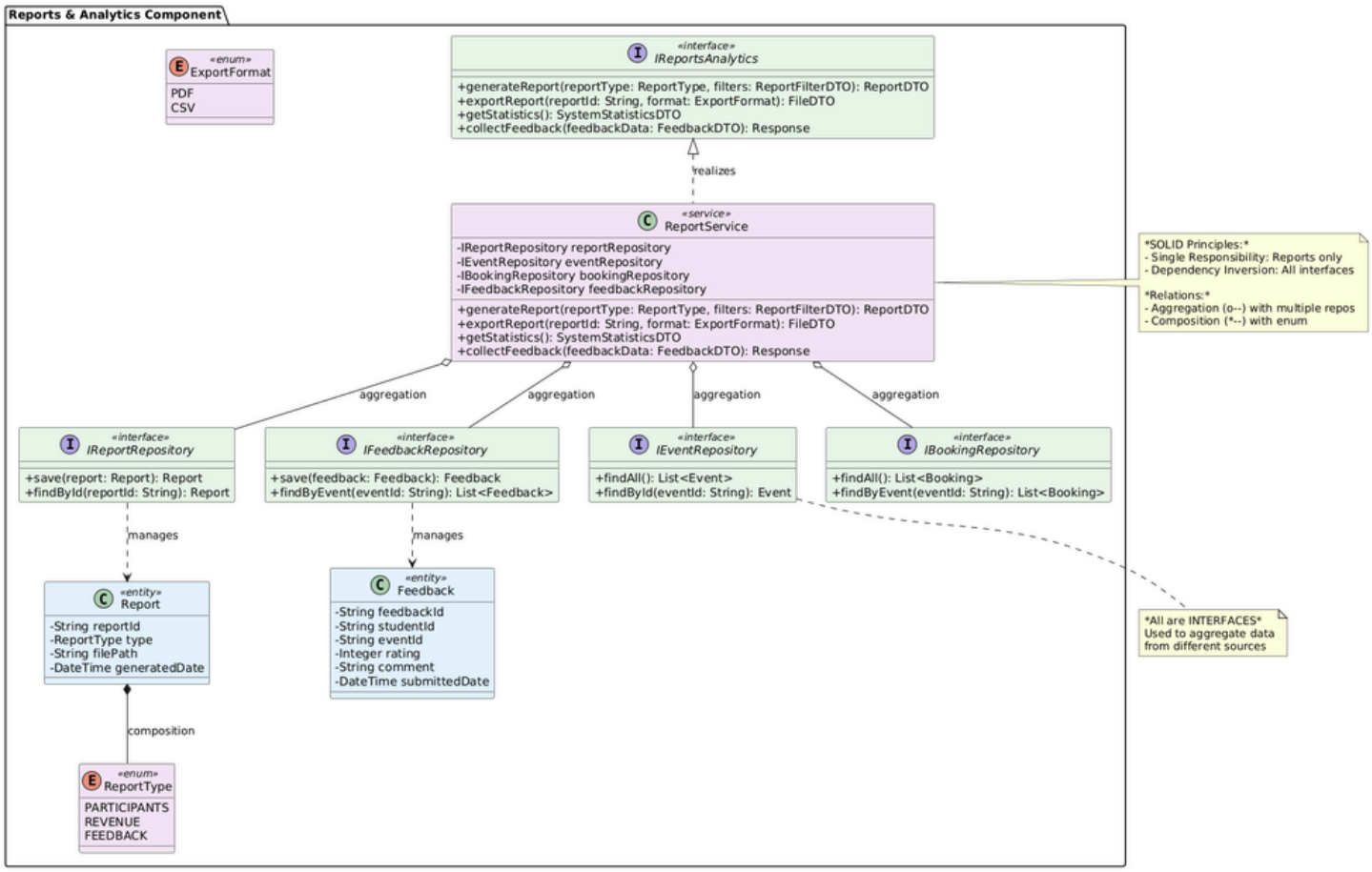
Booking Ticketing:



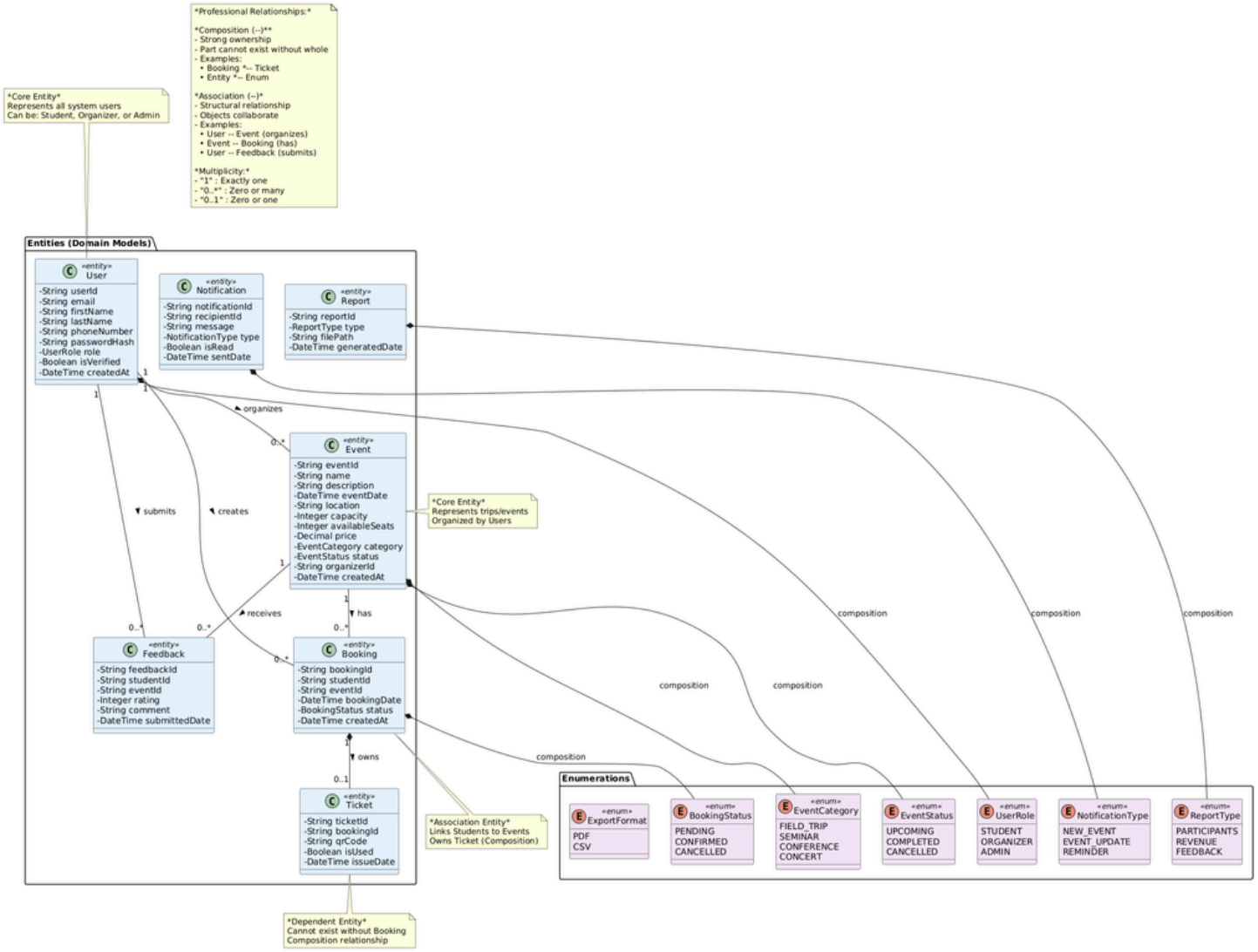
Notification:



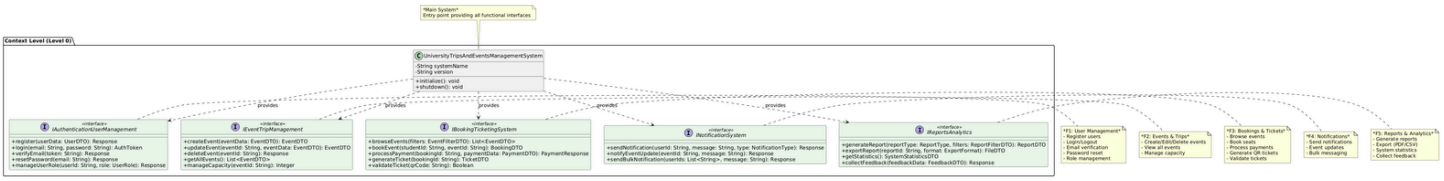
Reports Analysis:



Data Layer:

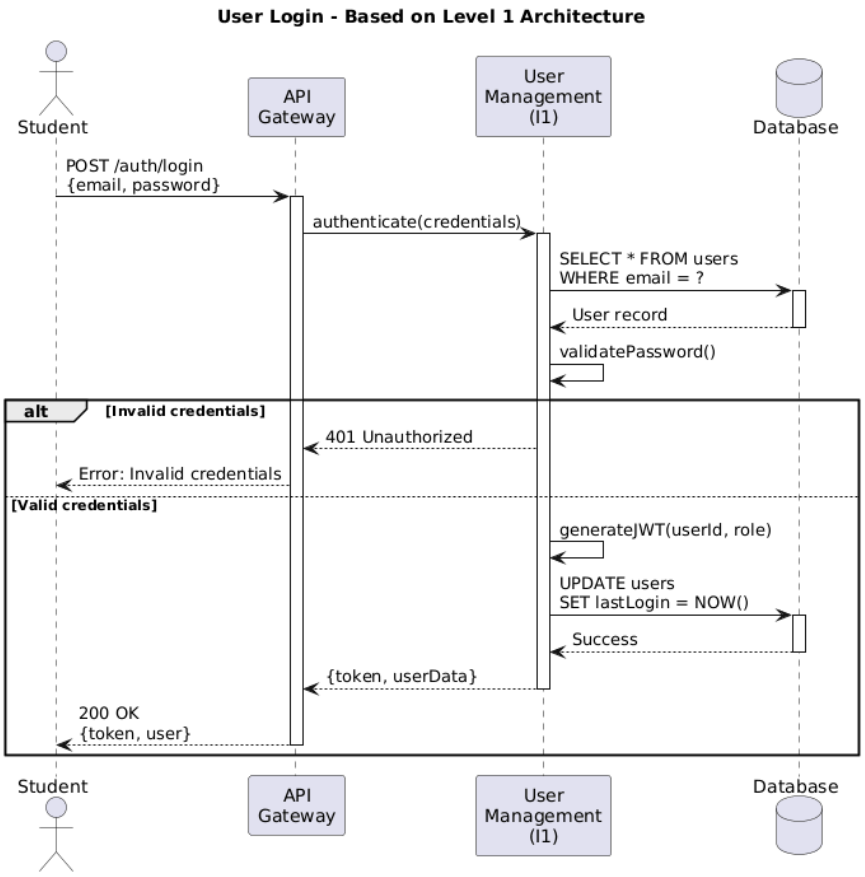


Context Level:

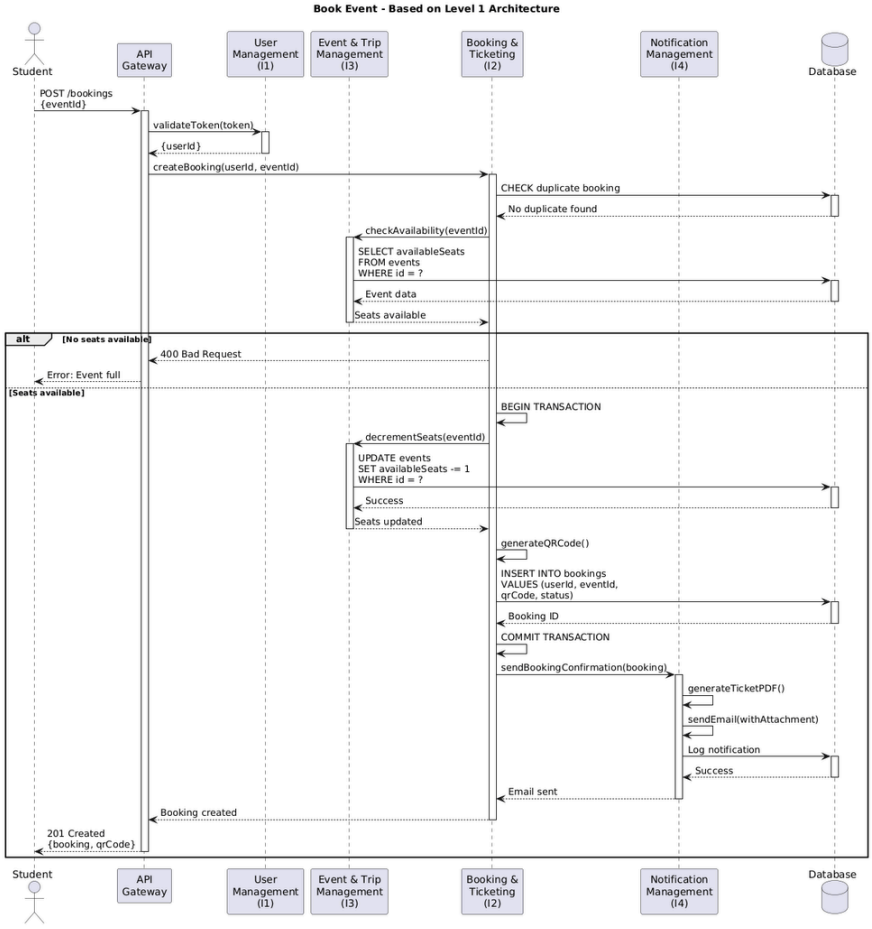


sequence Diagram:

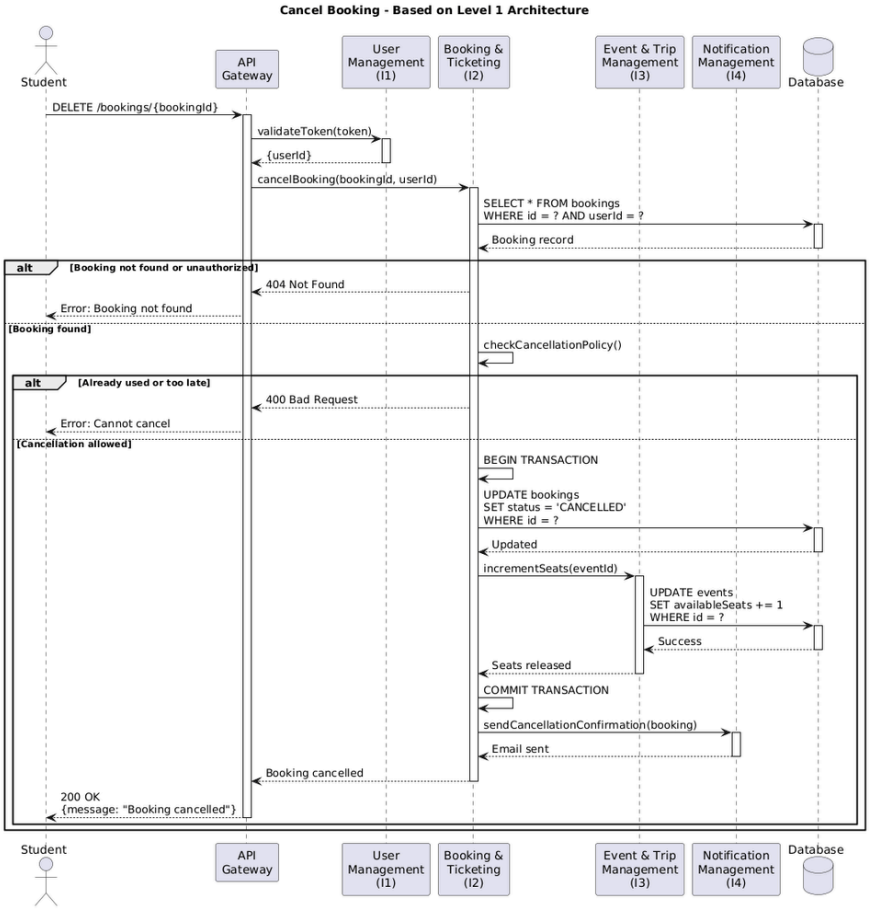
User Login :



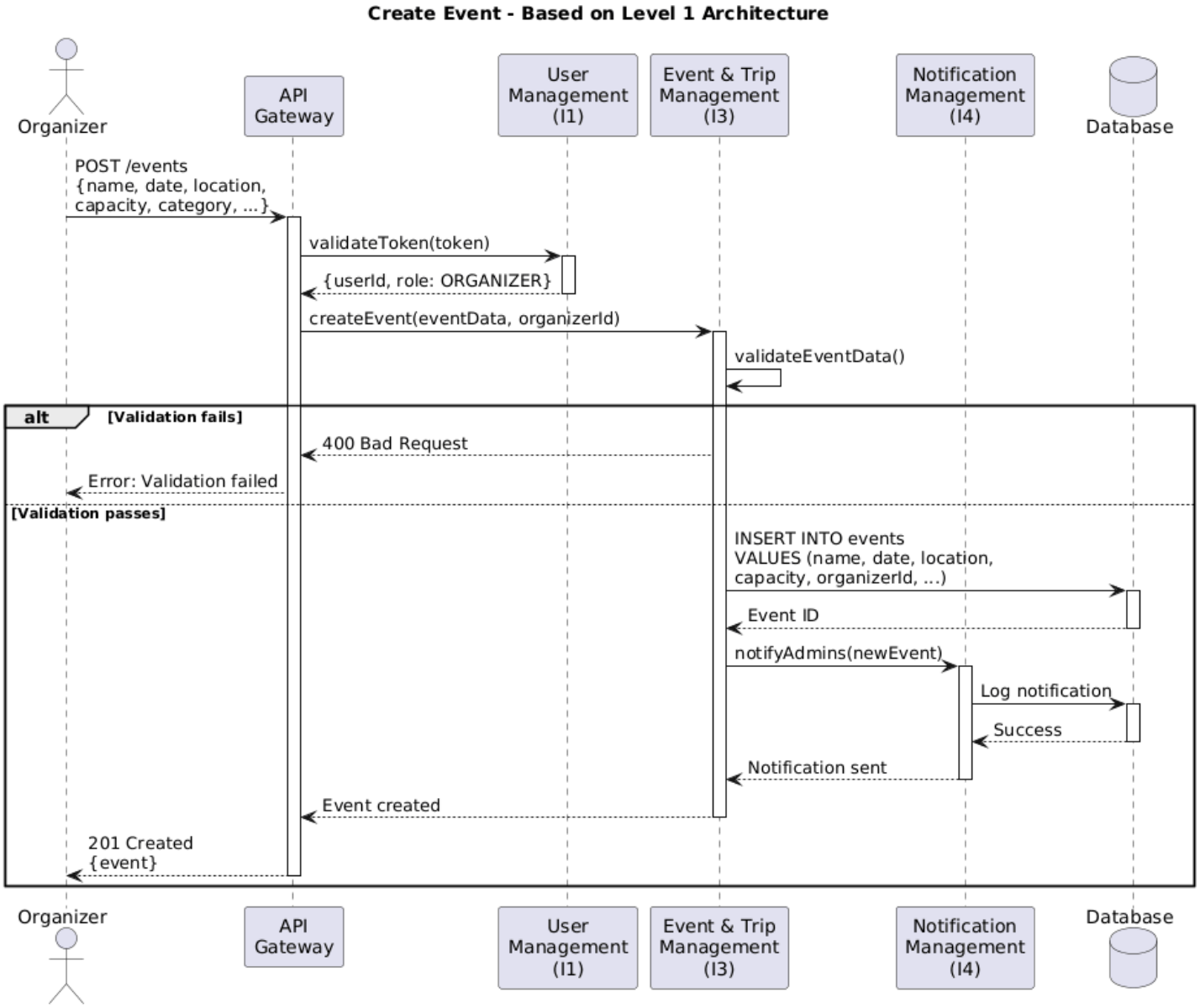
Book Event:



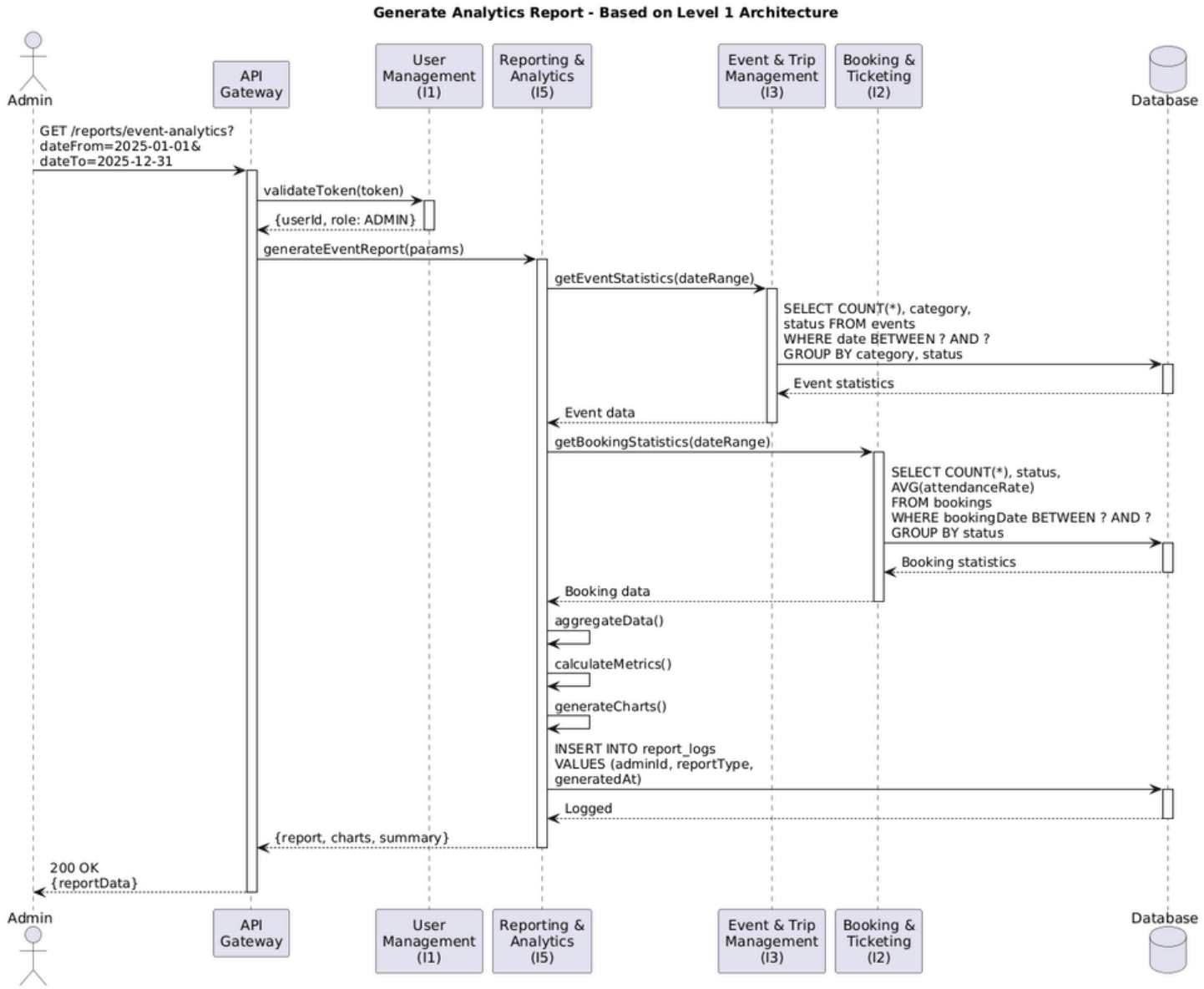
Cancel Booking:



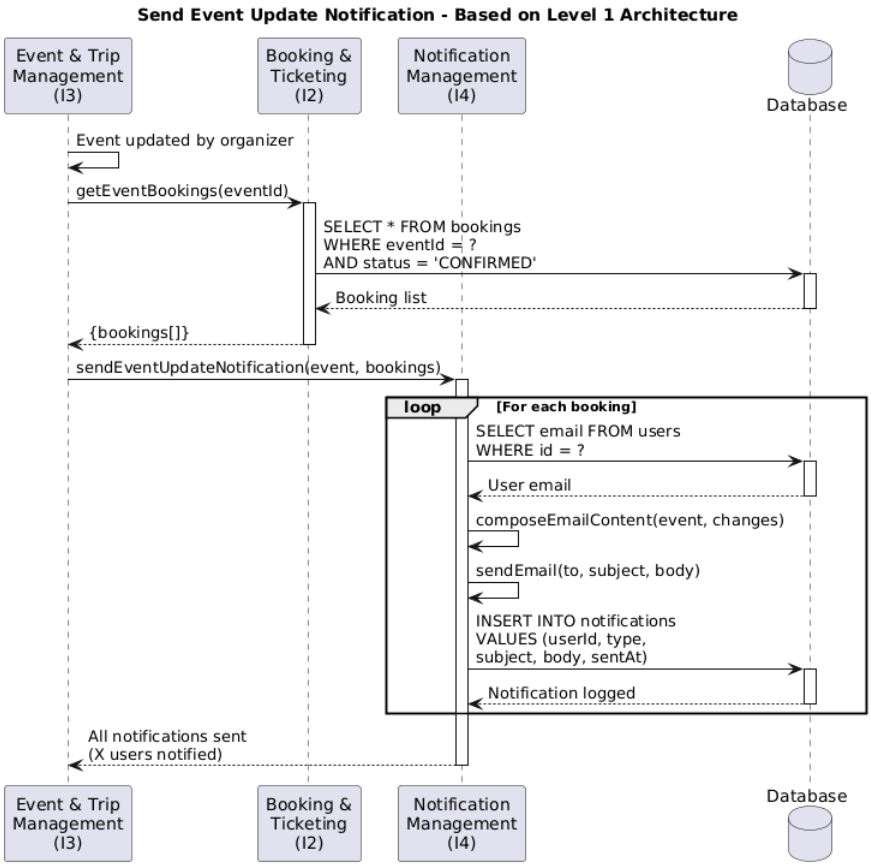
Create Event:



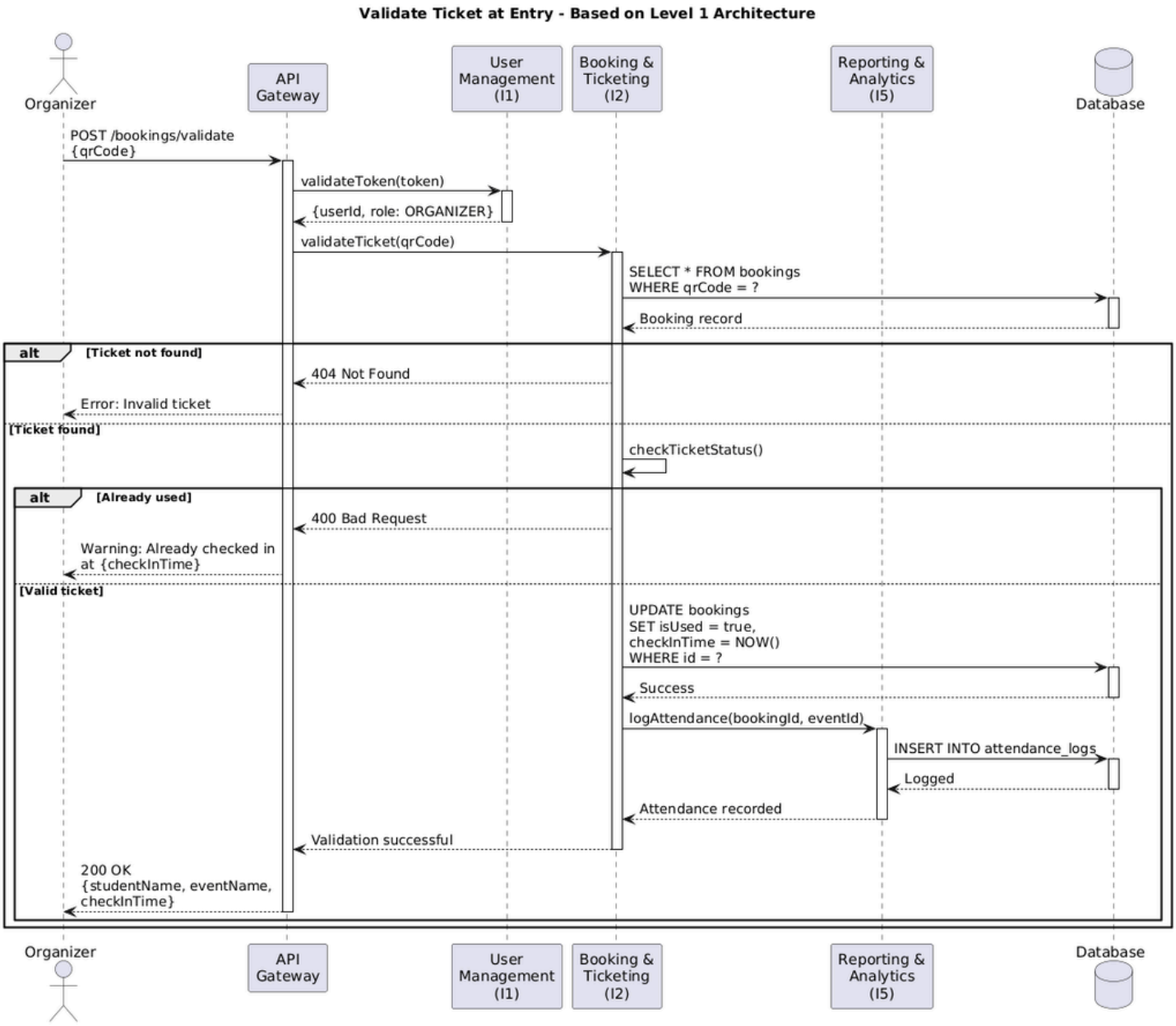
Generate Analytics Report:



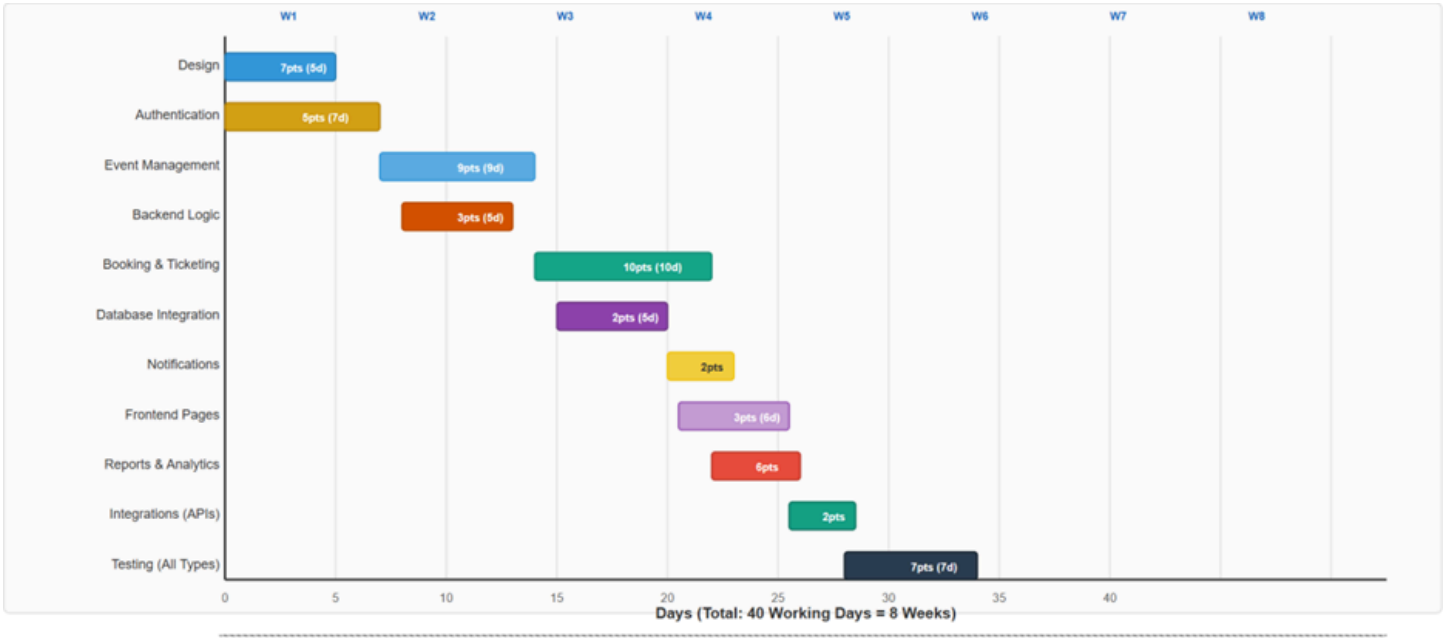
Send Event Update Notification:



Validate Ticket:

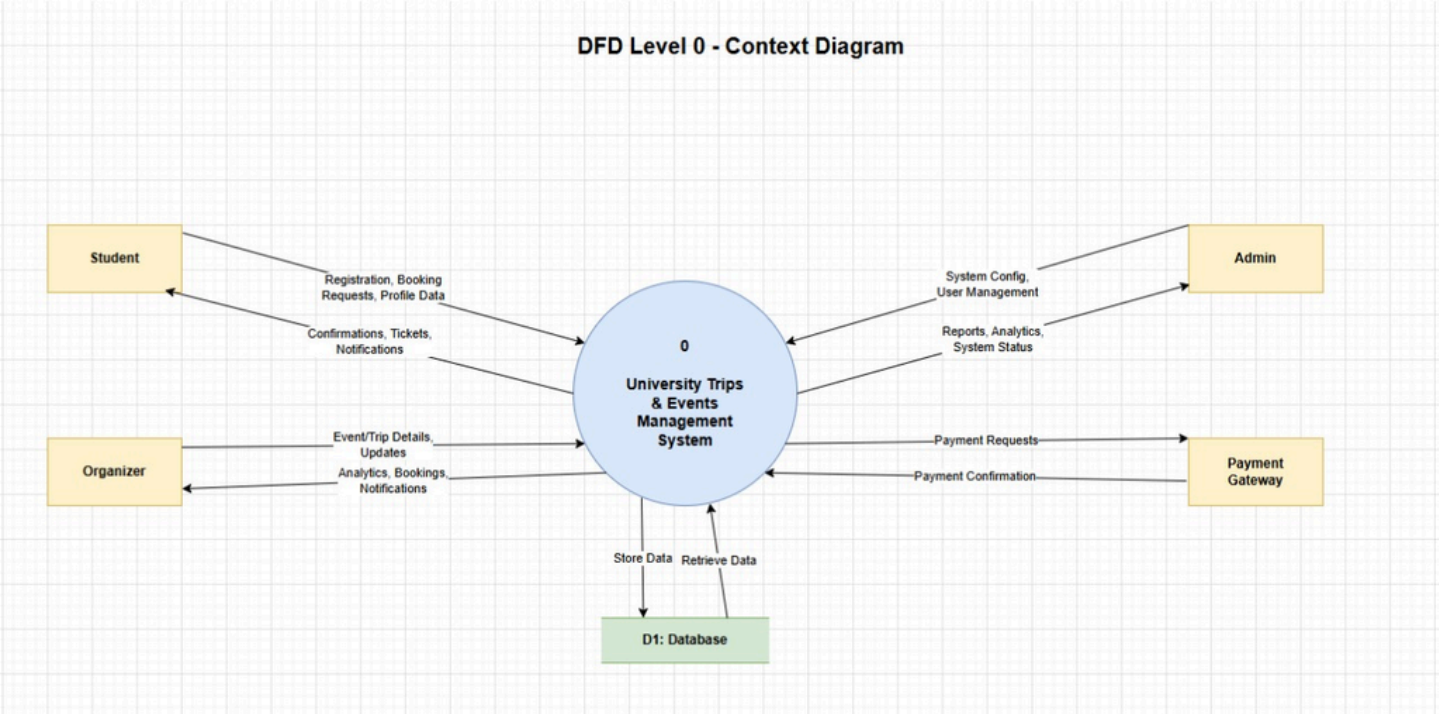

Validate Ticket at Entry - Based on Level 1 Architecture
Organizer
API Gateway
User Management (I1)
Booking & Ticketing (I2)
Reporting & Analytics (I5)
Database
POST /bookings/validate {qrCode}
validateToken(token)
{userId, role: ORGANIZER}
validateTicket(qrCode)
SELECT * FROM bookings WHERE qrCode = ?
Booking record
alt [Ticket not found]
404 Not Found
Error: Invalid ticket
[Ticket found]
checkTicketStatus()
alt [Already used]
400 Bad Request
Warning: Already checked in at {checkInTime}
[Valid ticket]
UPDATE bookings SET isUsed = true, checkInTime = NOW() WHERE id = ?
Success
logAttendance(bookingId, eventId)
INSERT INTO attendance_logs
Logged
Attendance recorded
Validation successful
200 OK {studentName, eventName, checkInTime}

Gantt Chart:

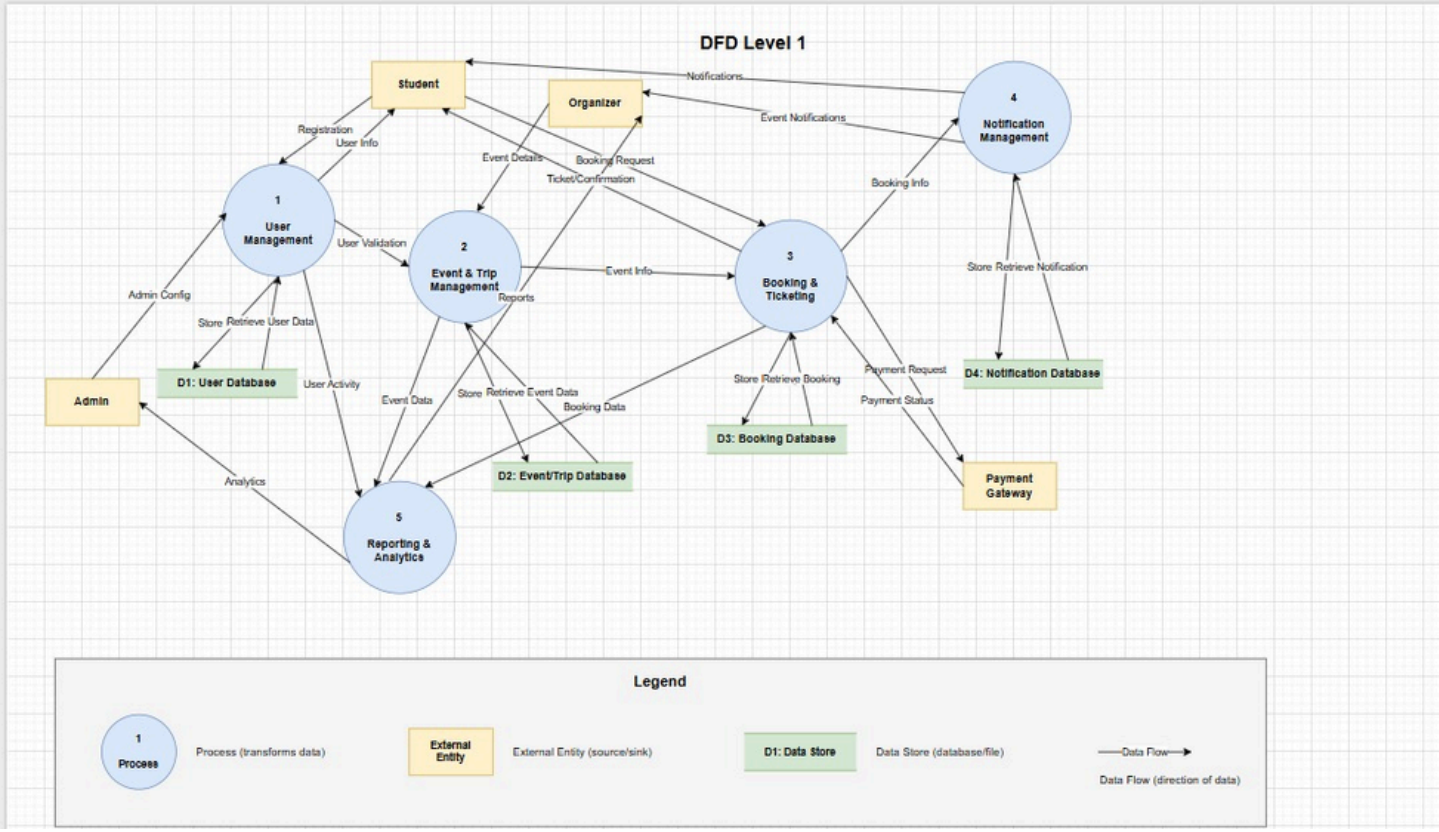

W1
W2
W3
W4
W5
W6
W7
W8
Design
7pts (5d)
Authentication
8pts (7d)
Event Management
9pts (9d)
Backend Logic
3pts (5d)
Booking & Ticketing
10pts (10d)
Database Integration
2pts (5d)
Notifications
2pts
Frontend Pages
3pts (6d)
Reports & Analytics
6pts
Integrations (APIs)
2pts
Testing (All Types)
7pts (7d)
0
5
10
15
20
25
30
35
40
Days (Total: 40 Working Days = 8 Weeks)

DFD Diagram:

Level 0:



Level 1:



Project Estimation:

AIU Trips s Events Management System Step 1:

Introduction

This report provides the estimation and scheduling for the **AIU Trips s Events**

Management System based on the **Fibonacci Function Point Estimation** method.

The project is divided into phases, subsystems, and class-level features. Each feature is assigned a Fibonacci point according to its complexity, then converted into effort days to estimate the total duration and project size.

Step 2: Approach

- **Estimation Method:** Fibonacci-based Function Point Estimation
- **Breakdown Level:** From Use Cases → Subsystems → Classes
- **Conversion Rate:** 1 Fibonacci Point = 5 Days
- **Team Size:** 5 Developers
- **Project Duration:** 2 Months (40 working days per developer → 200 developer-days total)

Step 3: Phase Breakdown and Estimation Phase 1 – Use Cases s Core Functionalities

Subsystem	Class / Feature	Fibonacci Points	Effort (Days)
Authentication	**UserRegistration**	**2**	**10**
	Login	**1**	**5**
	ResetPassword	**1**	**5**
	EmailVerification	**1**	**5**
Event Management	**EventCreator**	**3**	**15**
	EventEditor	**2**	**10**
	EventRemover	**1**	**5**
	CapacityChecker	**3**	**15**
Booking s Ticketing	**BookingCreator**	**3**	**15**
	BookingCanceller	**2**	**10**

	DuplicateChecker	1	5
	TicketGenerator (QR)	2	10
	TicketValidator	1	5
Notifications	NotificationSender	1	5
	ReminderScheduler	1	5
Reports s Analytics	ReportGenerator	2	10
	TrendAnalyzer	3	15
	ExportManager	1	5

Subtotal (Phase 1): 32 Points → 160 Days

Phase 2 – Design

Task	Description	Points	Days
System Architecture	Define system layers and interactions	3	15
Database Schema	Create ERD and relationships	2	10
UML Diagrams	Class and sequence diagrams	1	5
UI Mockups	Interface sketches	1	5

Subtotal (Phase 2): 7 Points → 35 Days

Phase 3 – Implementation

Task	Description	Points	Days
Backend Logic	Business logic and API controllers	3	15
Frontend Pages	User interface C validation	3	15
Integrations	Email, QR, and payment APIs	2	10
Database Integration	CRUD operations	2	10

Subtotal (Phase 3): 10 Points → 50 Days

Phase 4 – Testing

Task	Description	Points	Days
Unit Testing	Test individual classes	2	10
Integration Testing	Test interactions between modules	2	10
User Acceptance Testing	Simulate user scenarios	1	5
Non-Functional Testing	Security C performance	2	10

Subtotal (Phase 4): 7 Points → 35 Days

Phase 5 – Deployment s Documentation

Task	Description	Points	Days
Deployment Setup	**Docker and environment configs**	**2**	**10**
Documentation	**User C developer guide**	**1**	**5**
Maintenance	**Minor fixes after release**	**1**	**5**

Subtotal (Phase 5): 4 Points → 20 Days

Step 4: Total Estimation

Total Points	Conversion	Total Effort
40 Fibonacci Points	**1 Point = 5 Days**	**200 Developer-Days (Fits Exactly)**

Step 5: Capacity vs Effort Analysis

Parameter	Calculation	Result
Team Capacity	**5 Developers × 40 Days**	**200 Developer-Days**
Required Effort	**From Estimation**	**200 Developer-Days**

Step 6: Schedule Summary

The estimated effort of **200 developer-days** fits perfectly into the 2-month duration.

Each developer contributes **40 working days**, covering all project phases including design, implementation, testing, and deployment.

This ensures balanced workload distribution and on-time project completion.

Step 7: Final Conclusion

The total estimation for the **AIU Trips s Events Management System** equals **40 Fibonacci Points (200 developer-days)**.

This estimation fits perfectly within the 2-month project duration for a 5-member team. The breakdown ensures balanced effort, realistic timelines, and alignment with the estimation

methodology explained in the lecture.

Total Effort Summary

- **Total Fibonacci Estimate:** 40 Points
- **Total Effort:** 200 Days
- **Fits perfectly in 8 weeks (5 developers × 5 days/week × 2 months)**

Assumptions

- **Team Capacity:** 5 Developers
- **Workdays per Week:** 5 Days
- **Total Working Duration:** 2 Months (≈ 40 working days per developer)
- **Conversion Rate:** 1 Fibonacci Point = 5 Days
- **Daily Capacity:** 5 developers × (1 point / 5 days) = **1 point/day**
- **Total Duration:** 40 Fibonacci Points ÷ 1 point/day = **40 working days (≈ 8 weeks)**

Technology Used:

Presentation Layer

- **Languages: HTML , Typescript**
- **Frameworks/Libraries: Next.js , Tailwind CSS**

Logic Layer

- **Languages: Java**
- **Frameworks/Libraries: Java Spring Boot**
- **Data Base: H2 SQL Database**

General

- **Tools: Docker , Git , GitHub , Postman , Plant UML, Draw io**
- **AI IDEs/Software : Windsurf , Jules , Gemini CLI , Vs Code**