

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CSEN2091 - OOSE BASED APPLICATION DEVELOPMENT**

**CASE STUDY**

ON

SMART CONFERENCE MANAGEMENT PLATFORM

**Submitted by,**

V. HIMA CHARITHA

BU21CSEN0300232

CSE-AIML

SEM: 7

BATCH:19

GITAM UNIVERSITY

BANGALORE.

# Introduction

* + **1.1 Project Title:** Smart Conference Management Platform (SCMP)
  + **1.2 Objective:** The objective of the *Smart Conference Management* *Platform* is to revolutionize conference management by addressing the limitations of traditional systems through a suite of advanced, user-centered features. This platform leverages innovative technology to streamline event coordination, amplify user engagement, and boost accessibility for all participants. By integrating data-driven insights, SCMP empowers organizers to make informed decisions in real-time, optimize attendee experiences, and increase the overall effectiveness of conferences. Through these capabilities, SCMP aims to set a new standard for modern, efficient, and engaging conference management.
  + **1.3 Scope:** The SCMP will provide a comprehensive solution for planning, organizing, and managing conferences. The platform will support-
* Event creation and scheduling
* Participant registration and personalisation
* Session and speaker management
* Ticketing, payment processing, and data security.
* Communication and notifications
* Feedback collection, post-conference analytics.
  + **1.4 Overview of Case Study:** The Smart Conference Management Platform will be divided into various modules, each responsible for specific functionalities, such as event listing, registration, session management, and more. Unique features like AI based Recommendations, gamification, and Virtual Exhibit Booths will be integrated into the platform to address the identified challenges.

# Software Requirements Specification:

# *The Overall Description*

# 2.1 Product Perspective

# The SCMP will serve as an advanced solution for conference management, providing organizers with tools to plan and execute conferences efficiently. It will be a web-based platform with potential integrations for mobile applications, supporting both in-person and virtual conferences.

# 2.2 Product Functions

# Key functions of the SCMP include:

# Event creation and scheduling

# Participant registration and ticketing

# Session and speaker management

# Payment processing and Data security

# Communication via email, SMS, and voice-activated assistance

# Feedback collection and AI-driven analytics

# Virtual Exhibit Booths and reporting

# 2.3 User Characteristics

# The primary users of the SCMP will include:

# Conference organizers and administrators

# Participants, including speakers, attendees, and exhibitors

# Technical support and IT personnel

# Users will vary in technical expertise, and the platform will provide a user-friendly interface with comprehensive documentation and support resources.

# 2.4 Constraints

# The system must support up to 10,000 concurrent users without performance degradation.

# Compliance with GDPR and other relevant data protection regulations is mandatory.

# Integration with existing tools and technologies, such as payment gateways and AI systems, must be seamless.

# 2.5 Assumptions and Dependencies

# Users will have access to reliable internet connections, especially for virtual and immersive conference features.

# The platform will be hosted on a scalable cloud infrastructure to handle varying loads.

# Dependencies include third-party services for payment processing and real-time translation.

# *Specific Requirements*

# 2.6 Functional Requirements

# Login: The system shall provide a secure login mechanism for users to access the platform.

# Authentication: The system shall implement multi-factor authentication (MFA) to ensure enhanced security for user accounts. The system shall support role-based access control to restrict features based on user roles (e.g., Admin, Organizer, Participant, Speaker).

# Event Listing: The system shall list all upcoming and past events with advanced search and filter options.

# Registration and Payment: The system shall enable participants to register, generate tickets, and securely process payments through integrated gateways.

# Session Management: The system shall allow organizers to create, manage, and schedule sessions, including assigning speakers.

# Notification and Alerts: The system shall send email, SMS, and voice-activated notifications to keep participants informed in real-time.

# AI-Powered Feedback and Analytics: The system shall collect participant feedback and generate actionable insights using AI-driven analytics

# Gamification: The system shall incorporate gamification features to enhance participant engagement and interaction.

# Real-Time Language Translation: The system shall provide real-time translation to ensure global accessibility for multilingual participants.

# Smart Networking Tools: The system shall facilitate intelligent matchmaking and networking among participants based on shared interests.

# AI-Powered Session Recommendations: The system shall suggest sessions to participants based on their preferences and behaviour.

# Customizable Event Options: The system shall allow organizers to use and customize pre-designed templates, create custom polls, and tailor other event-specific features to suit different types of events.

# Virtual Exhibit Booths: The system shall enable virtual exhibition booths where sponsors and vendors can interact with participants.

# 2.7 Non-Functional Requirements

# Performance: The system shall respond to user actions within 2 seconds on average.

# Scalability: The system shall support additional features and modules as required in the future.

# Security: The system shall ensure data encryption and implement role-based access control.

# Reliability: The system shall have an uptime of 99.9% and include backup and recovery mechanisms.

# Accessibility: The system shall adhere to accessibility standards (e.g., WCAG) to accommodate users with disabilities.

# Localization: The system shall support multiple languages and regional formats to cater to a global audience.

# Extensibility: The system shall allow for easy integration with third-party tools and services.

# Load Handling: The system shall efficiently manage peak loads, such as during large-scale virtual conferences, without degradation in performance.

# Energy Efficiency: The system shall optimize resource usage to minimize energy consumption, contributing to sustainable IT practices.

# 2.8 External Interface Requirements

# 2.8.1 User Requirements

# The system shall provide an intuitive and user-friendly interface.

# The system shall offer comprehensive user documentation and support resources.

# The system shall support multiple user roles with appropriate access controls.

# 2.8.2 Software Requirements

# The system shall integrate with external payment gateways for processing transactions.

# The system shall support API integrations for real-time language translation.

# The system shall interface with third-party blockchain services for certificate issuance.

# 2.8.3 Hardware Requirements

# The system shall be hosted on a scalable cloud infrastructure.

# The system shall support access from various devices, including desktops, laptops, tablets, and smartphones.

# The system shall require minimal hardware resources on the client side.

# 2.9 Logical Database Requirements

# The system shall maintain a robust database to manage user profiles, event data, session schedules, and feedback.

# The database shall support secure storage of sensitive information, including payment and personal data.

2.9.1 Design Constraints

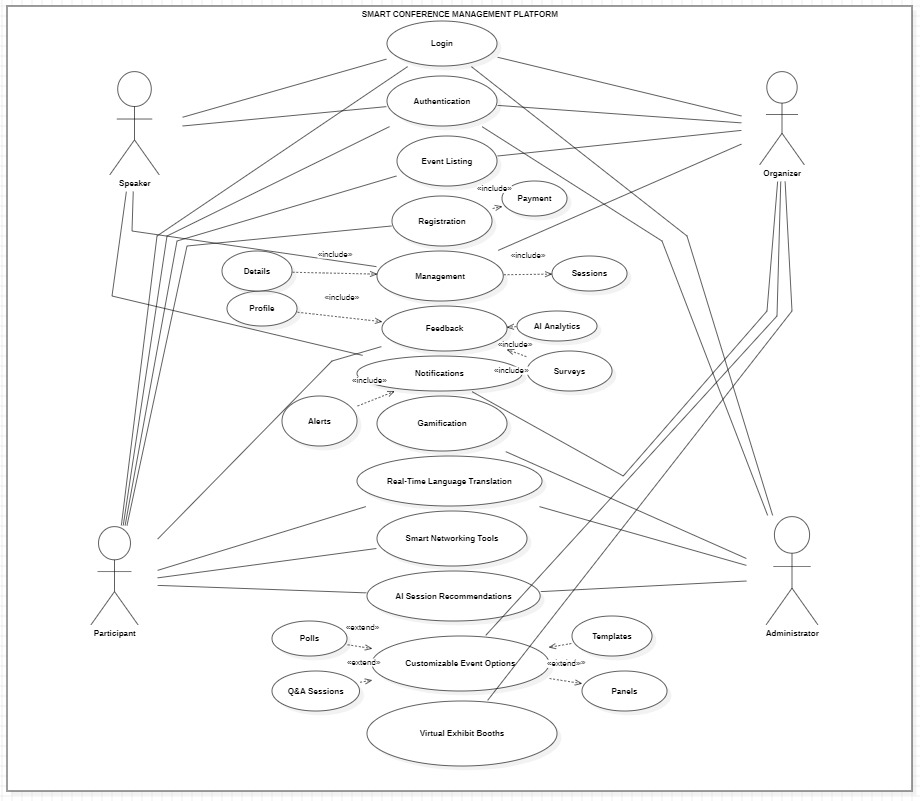
* The platform must be compatible with multiple devices and operating systems.
* The design must accommodate future technological advancements and scalability requirements.

2.9.2 Change Management Process

Any changes to the SCMP requirements, design, or implementation shall follow a structured change management process. This includes:

* Submission of change requests
* Impact analysis
* Approval by relevant stakeholders
* Implementation and testing
* Documentation updates

# Use Case Modeling

* + **3.1 Use Case Diagram:**
  + **3.2 Use Case Descriptions:** Provide descriptions for each use case including:

**Use Case: Login**

* + **Actors**: Speaker, Organizer, Participant, Administrator
  + **Preconditions**: User has an account on the platform.
  + **Postconditions**: User is authenticated and gains access to personalized dashboard.
  + **Main Flow**:

1. User enters credentials and requests login.
2. System verifies credentials and grants access.
   * **Alternate Flows**:

Invalid credentials entered; system displays an error and requests re-entry.

**Use Case: Authentication**

* + **Actors**: Speaker, Organizer, Participant, Administrator
  + **Preconditions**: User initiates login with valid credentials.
  + **Postconditions**: User session is securely established.
  + **Main Flow**:

1. System authenticates user credentials.
2. Access is granted, and user is redirected to the main dashboard.
   * **Alternate Flows**:

User credentials not authenticated; user is prompted to retry.

**Use Case: Event Listing**

* + **Actors**: Speaker, Organizer, Participant, Administrator
  + **Preconditions**: User is logged into the platform.
  + **Postconditions**: User views a list of available events.
  + **Main Flow**:

1. User navigates to the event listing page.
2. System displays a list of events based on user’s role.
   * **Alternate Flows**:

No events available; system displays a message indicating no events are listed.

**Use Case: Registration**

* + **Actors**: Speaker, Organizer, Participant
  + **Preconditions**: User is logged in and has selected an event.
  + **Postconditions**: User is registered for the selected event.
  + **Main Flow**:

1. User selects an event and initiates registration.
2. System processes the registration and confirms enrollment.
   * **Alternate Flows**:

Registration full; system notifies user of unavailability.

**Use Case: Payment**

* + **Actors**: Speaker, Organizer, Participant
  + **Preconditions**: User has registered for a paid event.
  + **Postconditions**: Payment is processed, and user gains event access.
  + **Main Flow**:

1. User proceeds to payment gateway and completes payment.
2. System confirms payment and updates registration status.
   * **Alternate Flows**:

Payment fails; system prompts user to retry.

**Use Case: Management**

* + **Actors**: Organizer, Administrator
  + **Preconditions**: User is logged in with management privileges.
  + **Postconditions**: Event details are managed and updated.
  + **Main Flow**:

1. User navigates to the management dashboard.
2. System provides options for event updates, scheduling, and notifications.
   * **Alternate Flows**:

Unauthorized access attempt; system restricts access.

**Use Case: Feedback**

* + **Actors**: Participant, Speaker, Organizer
  + **Preconditions**: User has attended an event session.
  + **Postconditions**: Feedback is recorded and available for analysis.
  + **Main Flow**:

1. User accesses feedback form and submits responses.
2. System stores feedback for future analysis.
   * **Alternate Flows**:

Feedback form not submitted; system saves partial data.

**Use Case: Notifications**

* + **Actors**: Speaker, Organizer, Participant
  + **Preconditions**: User is registered for an event.
  + **Postconditions**: User receives notifications on updates and alerts.
  + **Main Flow**:

1. System sends notifications for event updates, reminders, and alerts.
   * **Alternate Flows**:

User opts out of notifications; system halts notification delivery.

**Use Case: AI Analytics**

* + **Actors**: Organizer, Administrator
  + **Preconditions**: Feedback and survey data are available.
  + **Postconditions**: Insights are generated for event improvement.
  + **Main Flow**:

1. System analyzes collected feedback and survey data.
2. Insights are generated and presented in the analytics dashboard.
   * **Alternate Flows**:

Insufficient data; system notifies user of limited analytics.

**Use Case: Surveys**

* + **Actors**: Participant, Speaker
  + **Preconditions**: User has attended an event.
  + **Postconditions**: Survey data is collected.
  + **Main Flow**:

1. User completes post-event survey.
2. System stores responses for analysis.
   * **Alternate Flows**:

Survey skipped by user; system records non-completion.

**Use Case: Real-Time Language Translation**

* + **Actors**: Participant, Speaker
  + **Preconditions**: User is attending a live session.
  + **Postconditions**: Real-time translations are provided.
  + **Main Flow**:

1. User enables translation option.
2. System translates content in real-time based on language preference.
   * **Alternate Flows**:

Translation feature not available for selected language.

**Use Case: Smart Networking Tools**

* + **Actors**: Participant, Speaker, Organizer
  + **Preconditions**: User is attending an event.
  + **Postconditions**: Networking opportunities are facilitated.
  + **Main Flow**:

1. User accesses networking tools to connect with other participants.
2. System provides contact suggestions based on interests.
   * **Alternate Flows**:

No suitable connections; system displays general networking tips.

**Use Case: AI Session Recommendations**

* + **Actors**: Participant
  + **Preconditions**: User has set preferences.
  + **Postconditions**: Recommended sessions are displayed.
  + **Main Flow**:

1. System analyzes user preferences.
2. Recommended sessions are displayed.
   * **Alternate Flows**:

No matches; system displays popular sessions.

**Use Case: Customizable Event Options**

* + **Actors**: Organizer
  + **Preconditions**: User has access to event customization tools.
  + **Postconditions**: Event options are personalized.
  + **Main Flow**:

1. Organizer customizes event using templates and panel options.
2. System saves the customized settings.
   * **Alternate Flows**:

Customization settings invalid; system suggests defaults.

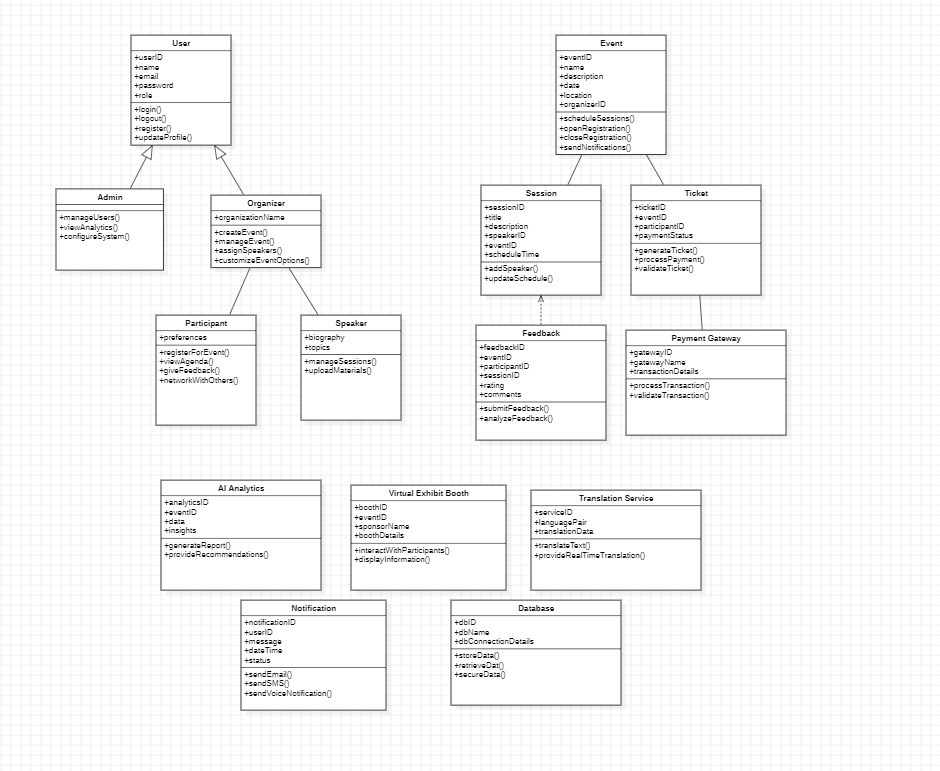
**Use Case: Virtual Exhibit Booths**

* + **Actors**: Participant, Organizer
  + **Preconditions**: Event includes virtual booths.
  + **Postconditions**: User accesses the exhibit booths.
  + **Main Flow**:

1. User selects a virtual booth to explore.
2. System provides virtual booth content.
   * **Alternate Flows**:

Booth closed or unavailable; system redirects to other booths.

# Class Diagram

* + **4.1 Class Diagram:**
  + **4.2 Class Descriptions:**

**1. Class Name: User**

* + **Attributes**:
  + userID: Unique identifier for the user.
  + name: Full name of the user.
  + email: Contact email for the user.
  + password: Password for account security.
  + role: Role type (e.g., Admin, Organizer, Speaker, Participant).
* **Methods**:
  + login(): Authenticates user credentials.
  + logout(): Logs the user out of the system.
  + register(): Registers a new user.
  + updateProfile(): Updates user's profile information.
* **Relationships**:
  + **Inheritance**: Acts as the base class for Admin, Organizer, Speaker, and Participant.

**2. Class Name: Admin**

* **Attributes**:
  + manageUsers(): Manages user accounts within the system.
  + viewAnalytics(): Views system analytics for performance insights.
  + configureSystem(): Configures overall system settings.
* **Relationships**:
  + **Inheritance**: Inherits from User.

**3. Class Name: Organizer**

* **Attributes**:
  + organizationName: Name of the organizer’s organization.
* **Methods**:
  + createEvent(): Creates a new event.
  + manageEvent(): Manages event details, scheduling, etc.
  + assignSpeakers(): Assigns speakers to sessions.
  + customizeEventOptions(): Customizes event settings.
* **Relationships**:
  + **Inheritance**: Inherits from User.
  + **Association**: Linked with the Event class for creating and managing events.

**4. Class Name: Participant**

* **Attributes**:
  + preferences: Preferred topics or session types.
* **Methods**:
  + registerForEvent(): Registers for a specific event.
  + viewAgenda(): Views event agenda.
  + giveFeedback(): Provides feedback on sessions or events.
  + networkWithOthers(): Connects and networks with other participants.
* **Relationships**:
  + **Inheritance**: Inherits from User.

**5. Class Name: Speaker**

* **Attributes**:
  + biography: Background and expertise of the speaker.
  + topics: Topics the speaker covers.
* **Methods**:
  + manageSessions(): Manages session details for the speaker.
  + uploadMaterials(): Uploads presentation materials.
* **Relationships**:
  + **Inheritance**: Inherits from User.

**6. Class Name: Event**

* **Attributes**:
  + eventID: Unique identifier for the event.
  + name: Name of the event.
  + description: Detailed description of the event.
  + location: Venue or online link for the event.
  + organizerID: ID of the organizer responsible.
* **Methods**:
  + scheduleSessions(): Schedules sessions within the event.
  + openRegistration(): Opens event registration.
  + closeRegistration(): Closes event registration.
  + sendNotifications(): Sends notifications to participants.
* **Relationships**:
  + **Association**: Associated with Session, Ticket, and Feedback.

**7. Class Name: Session**

* **Attributes**:
  + sessionID: Unique identifier for the session.
  + title: Title of the session.
  + description: Overview of session content.
  + speakerID: ID of the speaker for this session.
  + scheduledTime: Scheduled time for the session.
* **Methods**:
  + addSpeaker(): Assigns a speaker to the session.
  + updateSchedule(): Updates session schedule.
* **Relationships**:
  + **Association**: Linked to Event and Speaker.

**8. Class Name: Ticket**

* **Attributes**:
  + ticketID: Unique identifier for the ticket.
  + eventID: ID of the associated event.
  + participantID: ID of the participant.
  + paymentStatus: Status of payment (e.g., paid, unpaid).
* **Methods**:
  + generateTicket(): Generates a ticket for the participant.
  + processPayment(): Processes payment for the ticket.
  + validateTicket(): Validates the ticket for entry.
* **Relationships**:
  + **Association**: Connected with Event and Payment Gateway.

**9. Class Name: Feedback**

* **Attributes**:
  + feedbackID: Unique identifier for feedback.
  + eventID: ID of the associated event.
  + participantID: ID of the participant providing feedback.
  + rating: Rating provided by the participant.
  + comments: Additional comments from the participant.
* **Methods**:
  + submitFeedback(): Allows participant to submit feedback.
  + analyzeFeedback(): Analyzes feedback for insights.
* **Relationships**:
  + **Association**: Linked with Event and Participant.

**10. Class Name: Payment Gateway**

* **Attributes**:
  + gatewayID: Unique identifier for the payment gateway.
  + gatewayName: Name of the payment gateway provider.
  + transactionDetails: Details of the transaction.
* **Methods**:
  + processTransaction(): Processes the payment transaction.
  + validateTransaction(): Validates the transaction status.
* **Relationships**:
  + **Association**: Connected with Ticket.

**11. Class Name: AI Analytics**

* **Attributes**:
  + analyticsID: Unique identifier for the analytics report.
  + eventID: ID of the event analyzed.
  + insights: Insights generated from analytics.
* **Methods**:
  + generateReport(): Generates analytics report for the event.
  + provideRecommendations(): Provides recommendations based on analysis.
* **Relationships**:
  + **Dependency**: Depends on feedback data from Feedback.

**12. Class Name: Virtual Exhibit Booth**

* **Attributes**:
  + boothID: Unique identifier for the booth.
  + eventID: ID of the event where the booth is set up.
  + sponsorName: Name of the sponsor for the booth.
  + boothDetails: Information about the booth.
* **Methods**:
  + interactWithParticipants(): Allows interaction with booth visitors.
  + displayInformation(): Displays booth information to participants.
* **Relationships**:
  + **Association**: Associated with Event.

**13. Class Name: Translation Service**

* **Attributes**:
  + serviceID: Unique identifier for the translation service.
  + languagePair: Pair of languages for translation.
  + translationData: Data for real-time translation.
* **Methods**:
  + translateText(): Translates static text.
  + provideRealTimeTranslation(): Provides real-time translation during sessions.
* **Relationships**:
  + **Dependency**: Depends on Session for translation services.

**14. Class Name: Notification**

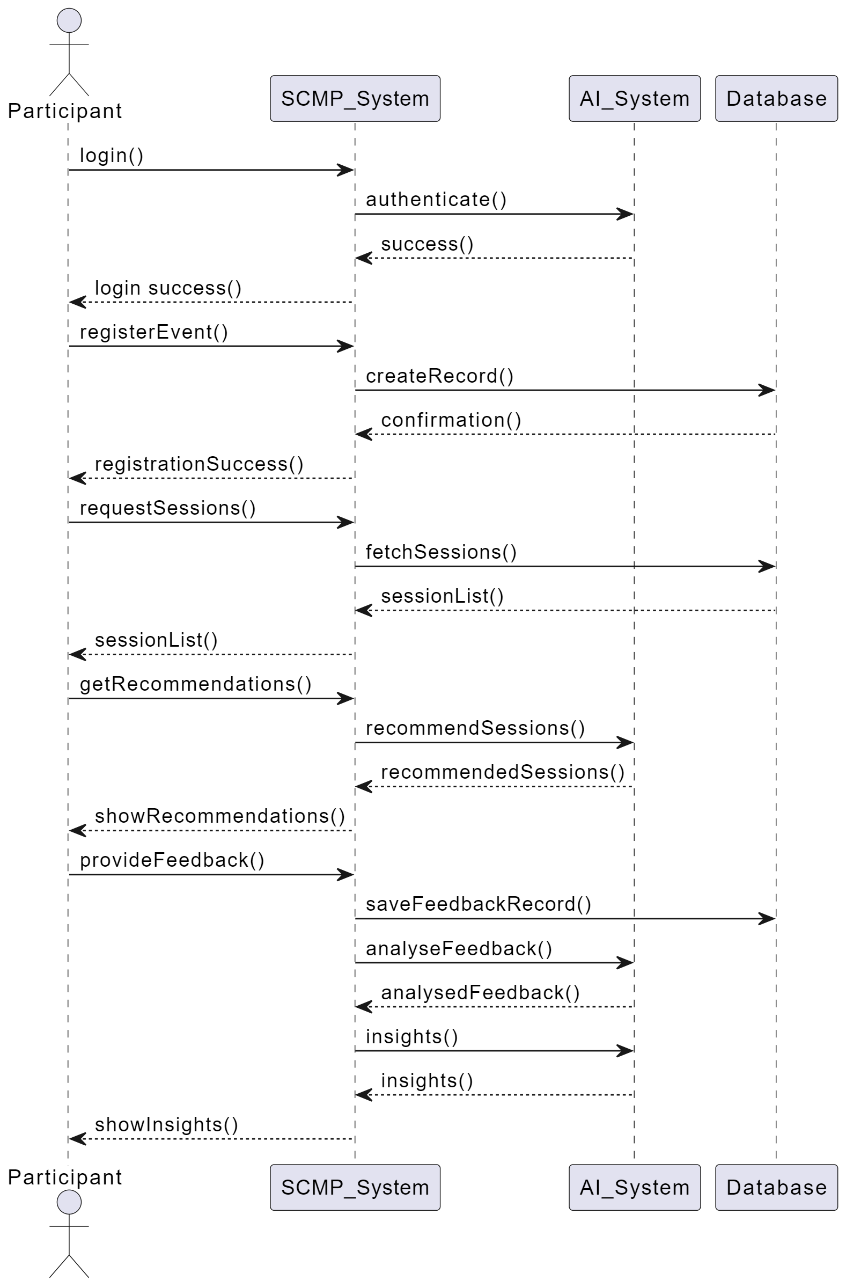
* **Attributes**:
  + notificationID: Unique identifier for the notification.
  + userID: ID of the user receiving the notification.
  + message: Message content of the notification.
  + dateTime: Date and time of notification.
* **Methods**:
  + sendEmail(): Sends notification via email.
  + sendSMS(): Sends notification via SMS.
  + sendVoiceNotification(): Sends a voice notification.
* **Relationships**:
  + **Association**: Connected with User for sending notifications.

**15. Class Name: Database**

* **Attributes**:
  + dbID: Unique identifier for the database.
  + dbName: Name of the database.
  + dbConnectionDetails: Connection details for database access.
* **Methods**:
  + storeData(): Stores data into the database.
  + retrieveData(): Retrieves data from the database.
  + secureData(): Ensures data security within the database.
* **Relationships**:
  + **Association**: Supports all classes that need data storage and retrieval.

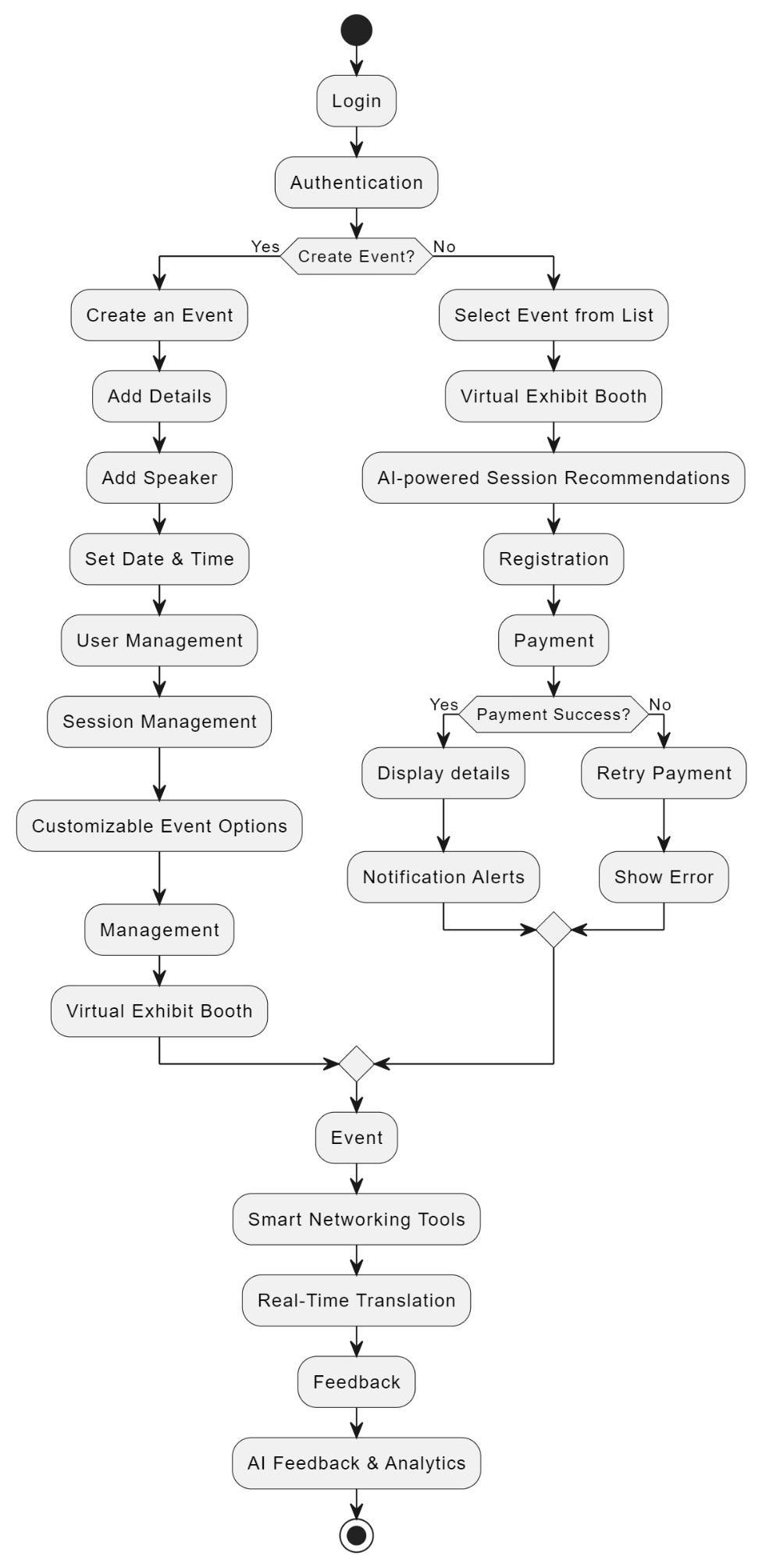
# Sequence Diagrams

* + **5.1 Sequence Diagrams for Key Scenarios:**

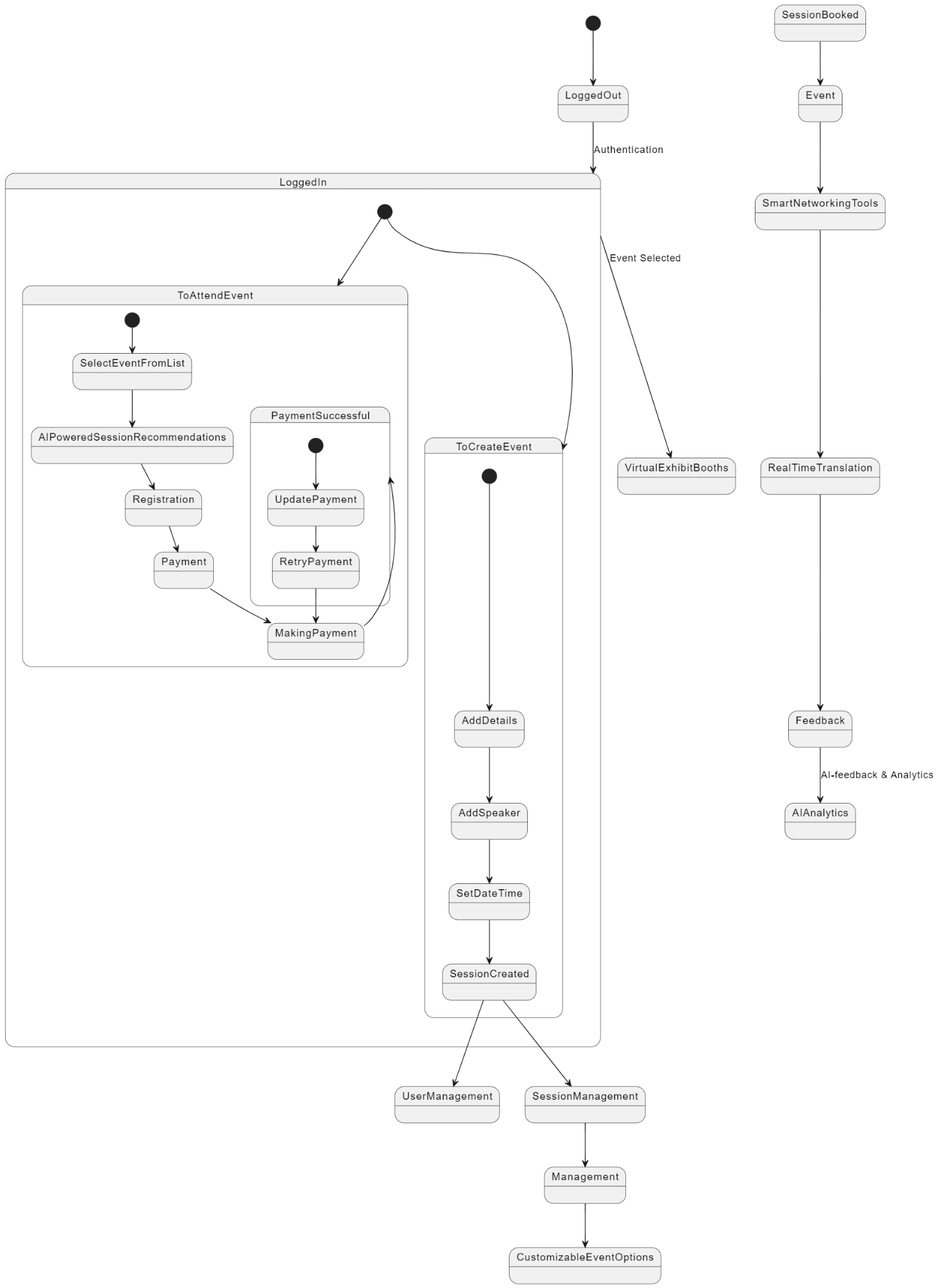


# Activity Diagrams

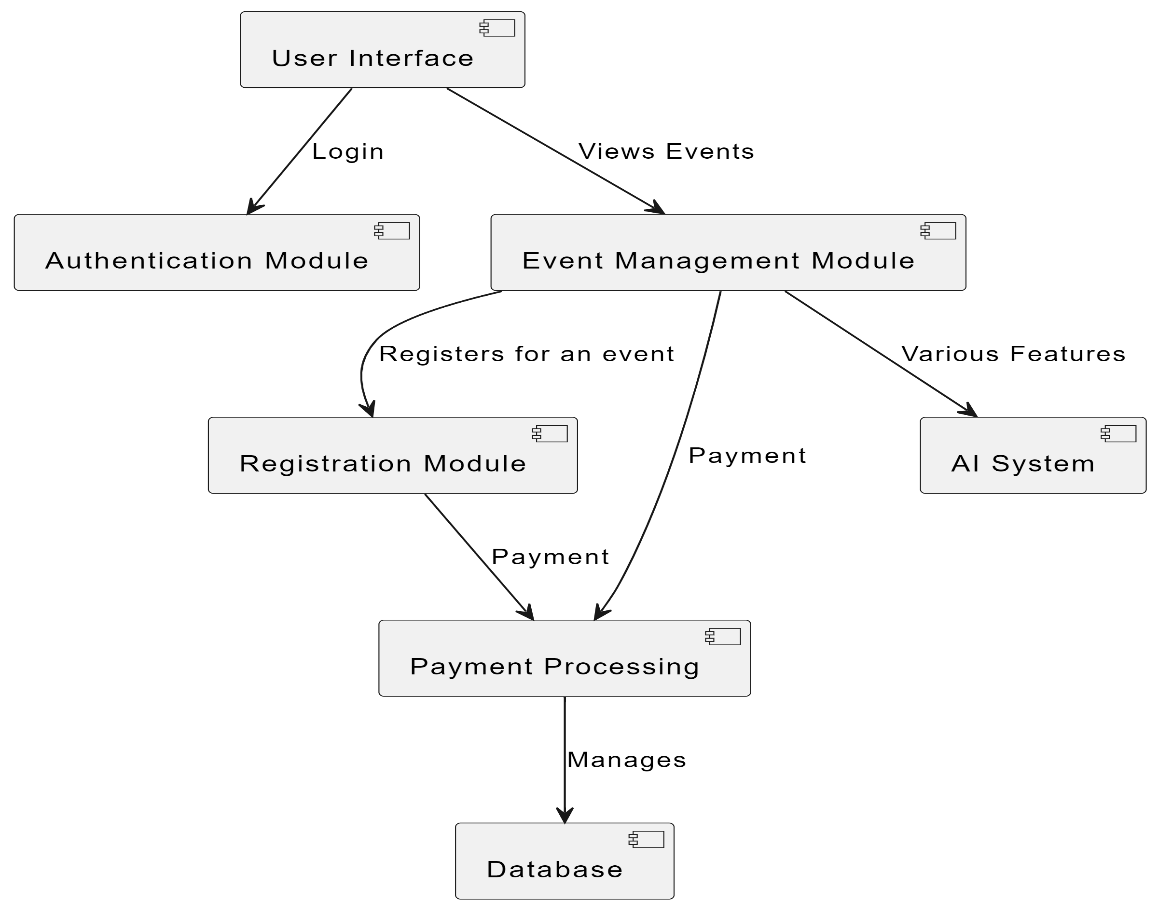
* + **6.1 Activity Diagrams for Processes:**



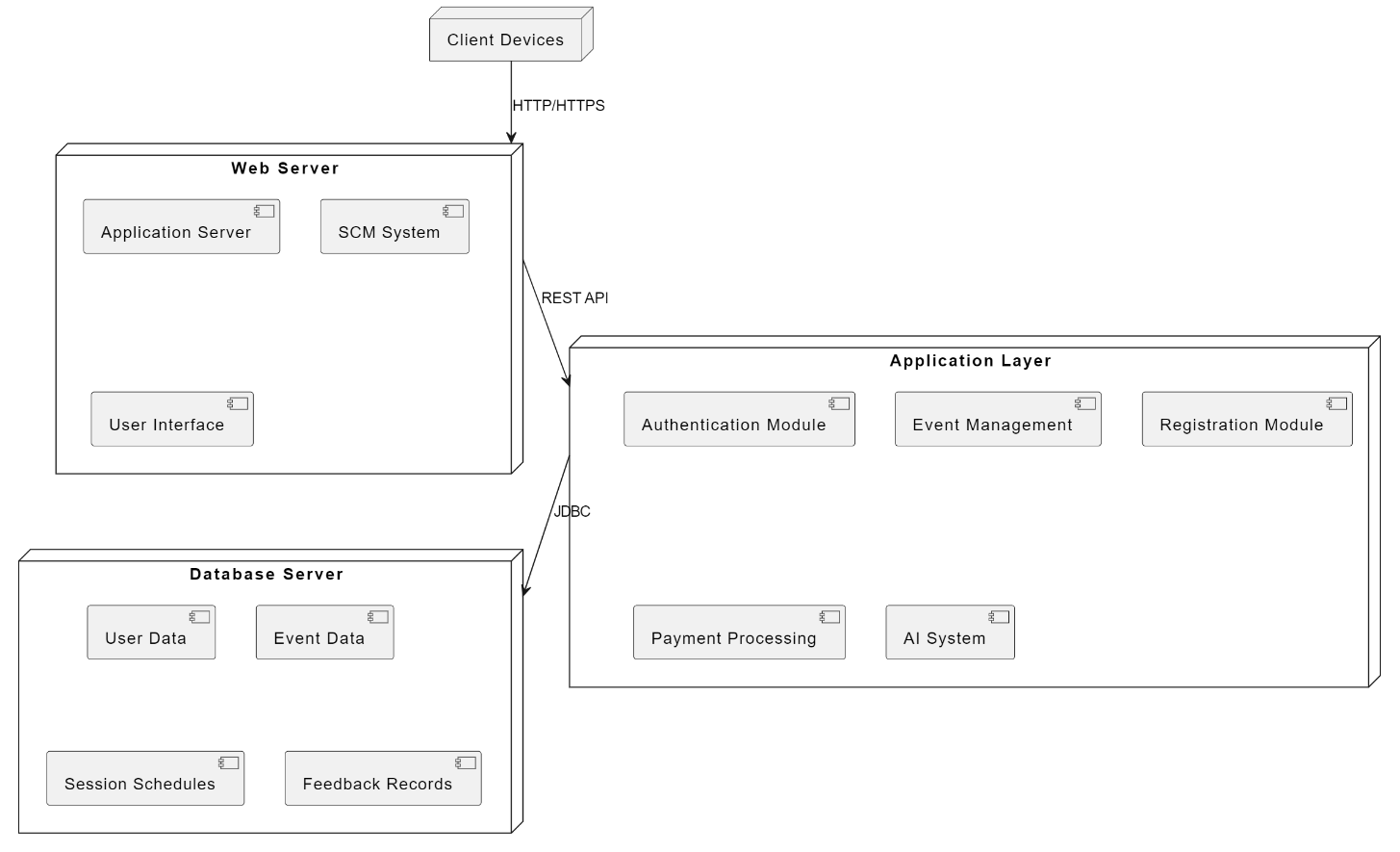
# State Diagrams

* + **7.1 State Diagram(s) for Main Entities:**

# Component Diagram

**8.1 Component Diagram:**

# Deployment Diagram

* + **9.1 Deployment Diagram:**

# User Interface Design:

# 

# 

1. **Testing:**

**11.1 Software Testing:**

Development of a new software, like any other product, remains incomplete until it subjected to exhaustive tests. The primary objective of testing is not to verify that all desired features have been implemented correctly. However, it also includes verification of the software behavior in case of "bad inputs".

Testing software is an important part of the development life cycle of a software. It is an expensive activity. Hence, appropriate testing methods are necessary for ensuring the reliability of a program. According to the ANSI/IEEE 1059 standard, the definition of testing is the process of analyzing a software item, to detect the differences between existing and required conditions i.e. defects/errors/bugs and to evaluate the features of the software item.

The purpose of testing is to verify and validate a software and to find the defects present in a software. The purpose of finding those problems is to get them fixed.

* **Verification** is the checking or we can say the testing of software for consistency and conformance by evaluating the results against pre-specified requirements.
* **Validation** looks at the systems correctness, i.e. the process of checking that what has been specified is what the user actually wanted.
* **Defect** is a variance between the expected and actual result. The defect’s ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

# 11.2 Testing Frameworks:

Following are the different testing frameworks:

* jUnit - for Java unit test
* Selenium - is a suite of tools for automating web applications for software testing purposes, plugin for Firefox
* HP QC - is the HP Web-based test management tool. It familiarizes with the process of defining releases, specifying requirements, planning tests, executing tests, tracking defects, alerting on changes, and analyzing results. It also shows how to customize project
* IBM Rational - Rational software has a solution to support business sector for designing, implementing and testing software

# 11.3 Need for Software Testing:

There are many reasons for why we should test software, such as:

* Software testing identifies the software faults. The removal of faults helps reduce the number of system failures. Reducing failures improves the reliability and the quality of the systems.
* Software testing can also improve the other system qualities such as maintainability, usability, and testability.
* In order to meet the condition that the last few years of the 20th century systems had to be shown to be free from the ‘millennium bug’.
* In order to meet the different legal requirements.
* In order to meet industry specific standards such as the Aerospace, Missile and Railway Signaling standards.

# Test Cases and Test Suite:

A test case describes an input description and an expected output description. Input is of two types: preconditions (circumstances that hold prior to test case execution) and the actual inputs that are identified by some testing methods. The set of test cases is called a test suite. We may have a test suite of all possible test cases.

# 11.4Types of Software Testing:

Testing is done in every stage of software development life cycle, but the testing done at each level of software development is different in nature and has different objectives. There are different types of testing, such as stress testing, volume testing, configuration testing, compatibility testing, recovery testing, maintenance testing, documentation testing, and usability testing. Software testing is mainly of following types:

1. **Unit Testing**
2. **Integration Testing**
3. **System Testing**
4. **Unit Testing**

Unit testing is done at the lowest level. It tests the basic unit of software, that is the smallest testable piece of software. The individual component or unit of a program are tested in unit testing. Unit testing is of two types.

* + **Black box testing**: This is also known as **functional testing**, where the test cases are designed based on input output values only. There are many types of Black Box Testing but following are the prominent ones.
* **Equivalence class partitioning**: In this approach, the domain of input values to a program is divided into a set of equivalence classes. e.g. Consider a software program that computes whether an integer number is even or not that is in the range of 0 to 10. Determine the equivalence class test suite. There are three equivalence classes for this program. - The set of negative integer - The integers in the range 0 to 10 - The integer larger than 10
* **Boundary value analysis:** In this approach, while designing the test cases, the values at boundaries of different equivalence classes are taken into consideration. e.g. In the above given example as in equivalence class partitioning, a boundary values based test suite is { 0, - 1, 10, 11 }
  + **White box testing**: It is also known as **structural testing**. In this testing, test cases are designed on the basis of examination of the code. This testing is performed based on the knowledge of how the system is implemented. It includes analyzing data flow, control flow, information flow, coding practices, exception and error handling within the system, to test the intended and unintended software behavior. White box testing can be performed to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities. This testing requires access to the source code. Though white box testing can be performed any time in the life cycle after the code is developed, but it is a good practice to perform white box testing during the unit testing phase.

# Integration Testing

Integration testing is performed when two or more tested units are combined into a larger structure. The main objective of this testing is to check whether the different modules of a program interface with each other properly or not. This testing is mainly of two types:

# Top-down approach

* + **Bottom-up approach**

In bottom-up approach, each subsystem is tested separately and then the full system is tested. But the top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level ‘skeleton’ has been tested, the immediately subroutines of the ‘skeleton’ are combined with it and tested.

# System Testing

System testing tends to affirm the end-to-end quality of the entire system. System testing is often based on the functional / requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability are also checked. There are three types of system testing

* + **Alpha testing** is done by the developers who develop the software. This testing is also done by the client or an outsider with the presence of developer or we can say tester.
  + **Beta testing** is done by very few numbers of end users before the delivery, where the change requests are fixed, if the user gives any feedback or reports any type of defect.
  + **User Acceptance testing** is also another level of the system testing process where the system is tested for acceptability. This test evaluates the system's compliance with the client requirements and assess whether it is acceptable for software delivery

An error correction may introduce new errors. Therefore, after every round of error-fixing, another testing is carried out, i.e. called regression testing. Regression testing does not belong to either unit testing, integration testing, or system testing, instead, it is a separate dimension to these three forms of testing.

# 11.5 Regression Testing:

The purpose of regression testing is to ensure that bug fixes and new functionality introduced in a software do not adversely affect the unmodified parts of the program. Regression testing is an important activity at both testing and maintenance phases. When a piece of software is modified, it is necessary to ensure that the quality of the software is preserved. To this end, regression testing is to retest the software using the test cases selected from the original test suite.

# Example

Write a program to calculate the square of a number in the range 1-100 #include <stdio.h>

int main()

{

int n, res;

printf("Enter a number: "); scanf("%d", &n);

if (n >= 1 && n <= 100)

{

res = n \* n;

printf("\n Square of %d is %d\n", n, res);

}

else if (n<= 0 || n > 100) printf("Beyond the range");

return 0;

}

# Output

Inputs Outputs

I1 : -2 O1 : Beyond the range

I2 : 0 O2 : Beyond the range

I3 : 1 O3 : Square of 1 is 1

I4 : 100 O4 : Square of 100 is 10000 I5 : 101 O5 : Beyond the range

I6 : 4 O6 : Square of 4 is 16

I7 : 62 O7 : Square of 62 is 3844

# Test Cases

T1 : {I1 ,O1}

T2 : {I2 ,O2}

T3 : {I3, O3}

T4 : {I4, O4}

T5 : {I5, O5}

T6 : {I6, O6}

T7 : {I7, O7}

A prevalent misconception among the beginners is that one should be concerned with testing only after coding ends. Testing is, in fact, not a phase towards the end. It is rather a continuous process. The efforts for testing should begin in the form of preparation of test cases after the requirements have been finalized. The Software Requirements Specification (SRS) document captures all features to be expected from the system. The requirements so identified here should serve as a basis towards preparation of the test cases. Test cases should be designed in such a way that all target features could be verified. However, testing a software is not only about proving that it works correctly. Successful testing should also point out the bugs present in the system, if any

**11.6 Test Suites for Smart Conference Management Platform Application:**

Designing test suites for an SCMP system application in a UML lab involves creating various test cases that ensure the correctness, performance, security, and user interaction of the SCMP software. Here is a structured approach to designing such test suites:

**11.6.1 SCMP Functionalities**

The key functionalities of the SCMP are:

* Event creation, scheduling, and management
* Participant registration and ticketing
* Session management (create, manage, assign speakers)
* Notifications and alerts (email, SMS)
* AI-powered feedback and analytics
* Gamification and smart networking
* Real-time language translation for accessibility
* Customizable event options and virtual exhibit booths

**11.6.2 Use Case Diagrams for SCMP**

Since this platform involves functionalities like event management, participant engagement, and analytics, the Use Case Diagrams should include:

* **Actors**: Organizer, Participant, Speaker, Admin, AI System
* **Use Cases**:
  + Create Event
  + Register for Event
  + Manage Sessions
  + Attend Event
  + Provide Feedback
  + Access Analytics
  + AI-Powered Recommendations

**11.6.3 Test Suite Design**

**Test Suite for Event Creation**

* **Test Case 1: Successful Event Creation**
  + **Precondition**: The user is logged in as an Organizer.
  + **Input**: Navigate to "Create Event", fill in valid event details (name, date, venue), and click "Create".
  + **Expected Output**: Event is successfully created, and confirmation is displayed.
  + **Priority**: High
* **Test Case 2: Missing Required Fields**
  + **Precondition**: The user is logged in as an Organizer.
  + **Input**: Navigate to "Create Event", leave mandatory fields blank (e.g., event name), and click "Create".
  + **Expected Output**: Error message "Please fill all required fields."
  + **Priority**: Medium
* **Test Case 3: Invalid Event Date**
  + **Precondition**: The user is logged in as an Organizer.
  + **Input**: Navigate to "Create Event", enter an invalid date (e.g., past date), and click "Create".
  + **Expected Output**: Error message "Invalid event date."
  + **Priority**: Medium

**Test Suite for Participant Registration**

* **Test Case 1: Successful Participant Registration**
  + **Precondition**: An event is created, and registration is open.
  + **Input**: Search for the event, click "Register", enter valid participant details, and complete payment.
  + **Expected Output**: Registration is successful, and a confirmation email is sent.
  + **Priority**: High
* **Test Case 2: Registration Without Payment**
  + **Precondition**: An event is created, and registration is open.
  + **Input**: Search for the event, click "Register", enter participant details, skip payment, and attempt to register.
  + **Expected Output**: Error message "Payment required to complete registration."
  + **Priority**: High
* **Test Case 3: Duplicate Registration Attempt**
  + **Precondition**: The participant has already registered for the event.
  + **Input**: Attempt to register for the same event again.
  + **Expected Output**: Error message "You are already registered for this event."
  + **Priority**: Medium

**Test Suite for Session Management**

* **Test Case 1: Successful Session Creation**
  + **Precondition**: An event is created, and the Organizer is logged in.
  + **Input**: Navigate to the event dashboard, select "Create Session", enter session details, and assign a speaker.
  + **Expected Output**: Session is created and displayed in the event schedule.
  + **Priority**: High
* **Test Case 2: Duplicate Session Name**
  + **Precondition**: A session with the same name already exists for the event.
  + **Input**: Navigate to "Create Session", enter a session name that already exists.
  + **Expected Output**: Error message "Session with this name already exists."
  + **Priority**: Medium

**Test Suite for Notifications and Alerts**

* **Test Case 1: Successful Email Notification**
  + **Precondition**: The participant has registered for an event.
  + **Input**: Complete registration, trigger email notification.
  + **Expected Output**: Registration confirmation email is sent.
  + **Priority**: Medium
* **Test Case 2: Invalid Email Address**
  + **Precondition**: The participant provides an invalid email during registration.
  + **Input**: Complete registration with an invalid email address.
  + **Expected Output**: Error message "Invalid email address."
  + **Priority**: Medium

**Test Suite for Feedback and Analytics**

* **Test Case 1: Successful Feedback Submission**
  + **Precondition**: The event has concluded, and the feedback session is open.
  + **Input**: Navigate to the feedback form, enter valid feedback, and submit.
  + **Expected Output**: Feedback is submitted successfully, and a confirmation message is displayed.
  + **Priority**: Medium
* **Test Case 2: Empty Feedback Submission**
  + **Precondition**: The event has concluded, and the feedback session is open.
  + **Input**: Navigate to the feedback form and submit without entering any feedback.
  + **Expected Output**: Error message "Feedback cannot be empty."
  + **Priority**: Low

**Test Suite for Smart Networking**

* **Test Case 1: Successful Networking Match**
  + **Precondition**: The participant is logged in, and networking options are enabled.
  + **Input**: Navigate to the "Smart Networking" feature, input preferences, and initiate match search.
  + **Expected Output**: Matching participant profiles are displayed.
  + **Priority**: High
* **Test Case 2: No Networking Matches Found**
  + **Precondition**: The participant is logged in, but no participants meet the networking criteria.
  + **Input**: Input specific preferences and initiate match search.
  + **Expected Output**: Error message "No matches found."
  + **Priority**: Low

**Test Suite for Gamification and Rewards**

* **Test Case 1: Points Accumulation for Participation**
  + **Precondition**: Participant attends multiple sessions during the event.
  + **Input**: Participate in various sessions and check points accumulation on the dashboard.
  + **Expected Output**: Points are accumulated and displayed on the participant’s profile.
  + **Priority**: Medium
* **Test Case 2: Redeeming Points for Rewards**
  + **Precondition**: Participant has sufficient points for a reward.
  + **Input**: Navigate to the rewards section, select a reward, and redeem points.
  + **Expected Output**: Reward is successfully redeemed, and points are deducted.
  + **Priority**: Medium

**11.6.4 Non-Functional Test Cases**

Non-functional tests for the Smart Conference Management Platform (SCMP) could involve:

* **Performance Testing:** Measure the response time and system stability under high load during peak event hours (e.g., 5000+ concurrent users accessing the platform for registration or session participation).
* **Security Testing:** Validate that participant and organizer credentials are encrypted during login, ensure secure payment gateway integration for event registrations, and check compliance with GDPR for data storage and access permissions.
* **Usability Testing:** Ensure the platform's user interface is intuitive, providing easy navigation for both organizers and participants. The UI should be accessible for all users, including those with disabilities.
* **Scalability Testing:** Assess how well the platform scales when new events, sessions, and participants are continuously added over time without degradation in performance.
* **Reliability Testing:** Verify that the platform remains stable and functional over extended periods of usage during long conferences and across multiple devices (mobile, web, etc.).
* **Compatibility Testing:** Ensure the SCMP is compatible with various web browsers, operating systems, and devices, providing a seamless experience regardless of platform.

**11.6.5 Sequence Diagrams and Test Execution**

For testing purposes, the sequence diagrams can cover scenarios like:

1. **Event Creation Flow**: Organizer initiates event creation → Inputs details → Event saved in the system → Confirmation sent to organizer
2. **Participant Registration Flow**: Participant searches for event → Registers → Payment processed → Ticket and confirmation email sent

**11.6.6 Test Suite Structure**

* **Test Suite 1**: Event Creation & Registration
  + Test cases include event creation, participant registration, payment processing
* **Test Suite 2**: Session Management
  + Test cases include session creation, speaker assignment, and session scheduling
* **Test Suite 3**: Notification and Alerts
  + Test cases cover email and SMS notifications during various actions like registration and event updates
* **Test Suite 4**: Analytics and Feedback
  + Test cases on AI-powered analytics, feedback submission, and insights generation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Description** | **Preconditions** | **Steps** | **Expected Result** | **Status** |
| TC1 | Test event creation | Organizer logged in | 1. Login as organizer,  2. Navigate to event creation,  3. Input event details, 4. Save event | Event created successfully, listed on the dashboard | Pass |
| TC2 | Test participant registration | Event is created | 1. Login as participant,  2. Search for event,  3. Register,  4. Complete payment | Participant registered; confirmation email sent | Pass |
| TC3 | Test session creation and speaker assign | Event is created, Organizer in | 1. Login,  2. Navigate to session management,  3. Create session,  4. Assign speaker | Session created; speaker assigned | Pass |
| TC4 | Test notification sending | Participant registered | 1. Register for event, 2. Trigger email notification on confirmation | Confirmation email sent to participant | Pass |
| TC5 | Test system under high load | 5000+ users accessing platform | 1. Simulate high load,  2. Monitor response times | Platform response within 2 seconds | Pass |

# Conclusion

**12.1 Summary of Findings**

The UML model developed for the **Smart Conference Management Platform (SCMP)** provides a comprehensive view of its core functionalities and interactions among various actors. This platform enables users to manage and participate in conferences by facilitating activities like event creation, participant registration, session management, and real-time updates. Each actor, including the Organizer, Participant, Speaker, and Admin, has clearly defined roles, ensuring an organized flow of operations. The class diagrams, use case diagrams, and sequence diagrams illustrate the relationships, associations, and dependencies between different components of the system. This model is structured to support efficient interactions while maintaining data flow integrity and process reliability.

Key findings from the UML model include:

* **User Interaction**: Participants can register, browse events, and join sessions, while organizers can set up and manage event details.
* **Event Management**: Organizers have access to tools for scheduling sessions, adding speakers, and tracking event status, enhancing the platform's usability.
* **Session and Participant Management**: The system allows for easy participant management, real-time session updates, and seamless transition between sessions.
* **Role-Based Access Control**: Each actor has specific access privileges, ensuring system security and functionality based on their roles.

**12.2 Limitations and Future Enhancements**

While the UML model provides a solid foundation for SCMP, there are certain limitations and areas for improvement to be considered in future versions:

* **Limitations**:
  + **Scalability Challenges**: The current model may face challenges in supporting a large number of concurrent users, especially during large-scale events.
  + **Limited Analytics and Reporting**: The system lacks detailed analytics for event performance, participant engagement, and speaker effectiveness.
  + **Basic Notification Mechanisms**: Notifications for participants are simple and could be expanded for a better user experience.
  + **Payment Integration**: Payment processing for premium events or registrations is minimally defined and could be enhanced with additional security.
* **Future Enhancements**:
  + **Advanced Analytics and Reporting**: Implementing detailed analytics for organizers to track participant engagement, peak times, and session feedback.
  + **Enhanced Notification System**: Adding a more sophisticated notification system with options for real-time updates and reminders before sessions.
  + **Scalable Architecture**: Implementing a scalable design to handle large events with thousands of participants.
  + **Expanded Payment Gateway Support**: Integrating multiple payment gateways and offering flexible payment options for premium events.
  + **Multi-Language Support**: Adding multi-language options for a more diverse audience.
  + **Improved Security Features**: Enhancing security measures for data privacy, especially for high-profile events with sensitive data.

**13. References**

* **Tools for UML Modelling**:
  + *StarUML*: Used for creating UML diagrams to visualize the architecture and design of the SCMP.
  + *SmartDraw*: An online tool used to develop UML diagrams and workflows.
* **Front-End Design Tool**:
  + **React**: Used for designing and prototyping the user interface of the SCMP.
* **Additional Resources**:
  + **Textbooks and Online Guides on UML**:
    - <https://jti.polinema.ac.id/wp-content/uploads/2019/02/Buku-Learning-UML-2.0.pdf>
    - <https://users.utcluj.ro/~civan/thesis_files/2017_MoldovanA_Ethesis.pdf>