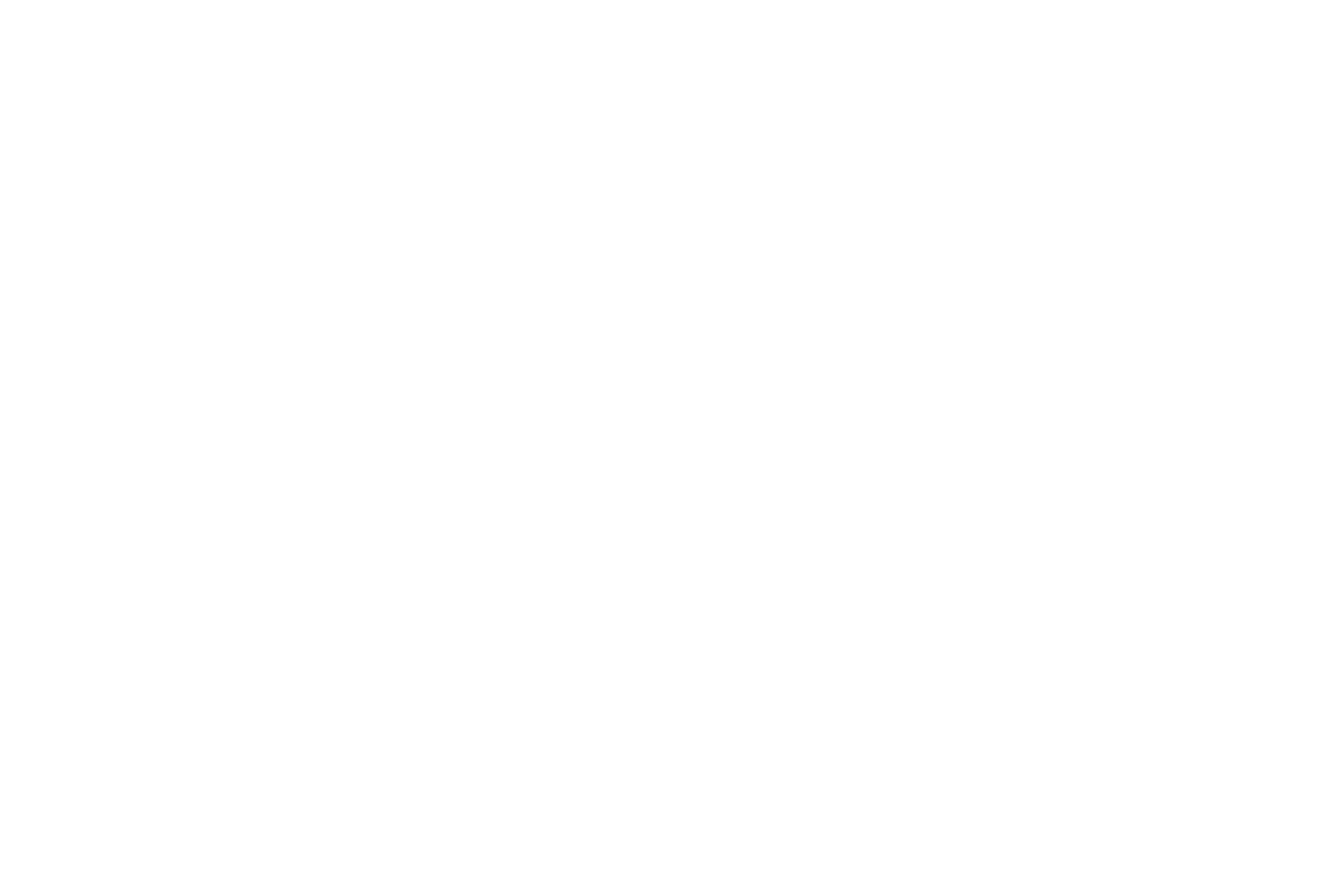
UUUnsdfvbjsfdgfd

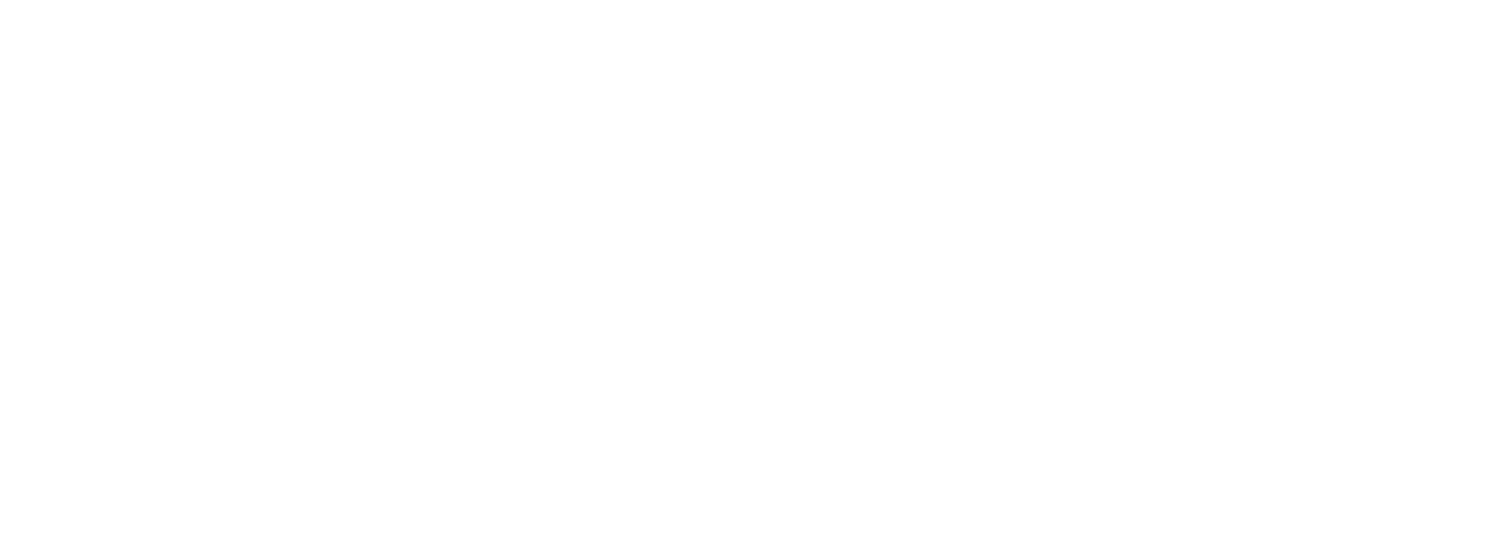


Malik, Tahira

Penn State University

Professor Nalabandhu, Santosh

May 02, 2025



Software Engineering System and Design 837: Final Project

## Course Project Proposal

Designing a Unified Academic Support Platform for Online University Students

Online universities, while offering accessibility and flexibility, often struggle to provide cohesive academic support experience. Services like virtual tutoring, academic advising, peer study sessions, and faculty office hours are frequently scattered across multiple platforms and communication channels. This disjointed setup can confuse students, reduce engagement, and make it difficult to access timely help or stay on track academically.

The design challenge is to create a unified academic support platform that brings all these services into a single, user-friendly system tailored specifically for the online learning environment. The platform should focus on consistent user interaction, seamless access across devices, time-zone aware scheduling, and role-based access for students, tutors, and advisors.

The main design ideas include keeping all information in one place, making the system easy to use and navigate, keeping track of sessions, and collecting feedback from students. The goal is to make the online experience smooth and simple, focusing on how the system is organized and used not how it’s built or what tools are used behind the scenes.

Academic Support Platform - Actor and Use Case Descriptions

# Actor Classification Table

|  |  |  |
| --- | --- | --- |
| Type | Actor | Goal Description |
| Primary | Student | Book, join, or cancel sessions; submit feedback. |
| Primary | Tutor | Set availability and conduct support sessions. |
| Primary | Advisor | Oversee session quality and review feedback. |
| Supporting | Authentication Service | Validate user credentials for login. |
| Supporting | Calendar System | Sync session times with user calendars. |
| Offstage | Notification System | Send booking confirmations or reminders. |
| Offstage | Database | Store sessions, feedback, and availability info. |
|  |  |  |

# Fully-Dressed Use Case Descriptions

* Use Case Name: Book a Session

Goal in Context: Student successfully books a support session with a tutor.

Primary Actors: Student (Primary), Calendar System (Supporting)

Scenario:

1. Student selects a tutor and time slot.
2. System checks availability via Calendar System.
3. System confirms slot and creates session.
4. System sends booking confirmation.

* Use Case Name: Submit Feedback

Goal in Context: Student submits feedback after a completed session.

Primary Actors: Student (Primary), Database (Offstage)

Scenario:

1. Student selects completed session.
2. Student rates the session and adds comments.
3. System stores feedback in database.
4. Tutor can view feedback later.

* Use Case Name: Manage Availability

Goal in Context: Tutor updates their availability slots.

Primary Actors: Tutor (Primary), Database (Offstage)

Scenario:

1. Tutor logs in and navigates to availability.
2. Tutor adds/edits available time slots.
3. System saves updated slots in the database.

* Use Case Name: Login/Register

Goal in Context: User logs into the system successfully.

Primary Actors: Student/Tutor/Advisor (Primary), Authentication Service (Supporting)

Scenario:

1. User enters credentials.
2. System verifies credentials using Authentication Service.
3. User is granted access to dashboard.

* Use Case Name: Sync Calendar

Goal in Context: Session details are synced to user's external calendar.

Primary Actors: Calendar System (Supporting), Session Management (Primary)

Scenario:

1. Session is created or updated.
2. System sends session data to Calendar System.
3. Calendar System adds it to user’s external calendar.

**Fully-Dressed Use Case Description- Unified Academic Support Platform**

Use Case Name: Book a Session

**Scope:** Academic Support Platform System

**Level:** user-goal

**Primary Actor**: Student

**Stakeholders and Interests**: Students want to easily schedule sessions. Tutors want their availability respected.

**Preconditions**: Student must be logged into the system.

**Success Guarantee**: Session is booked and stored; tutor and student are notified.

**Main Success Scenario:**

Student navigates to tutor's profile.

System shows available slots.

Students select a time slot.

System checks availability.

System creates a session entry.

System sends confirmation notifications.

**Extensions:** If the slot is already booked, an error message is shown, and a new time must be selected.

Use Case Name: Submit Feedback

**Scope**: Academic Support Platform System

**Level**: user-goal

**Primary Actor**: Student

**Stakeholders and Interests:** Students want to provide feedback to improve session quality. Tutors want to receive constructive feedback.

**Preconditions:** Session must be completed and feedback option available.

**Success Guarantee:** Feedback is stored and visible to the tutor.

**Main Success Scenario:**

Student selects completed sessions.

Student rates the session and adds comments.

System stores feedback in database.

The tutor can view feedback later.

**Extensions:** If feedback is not submitted, system will prompt a reminder later.

**Special Requirements**: Feedback must be submitted within 24 hours of session end.

Use Case Name: Manage Availability

**Scope:** Academic Support Platform System

**Level:** user-goal

**Primary Actor:** Tutor

**Stakeholders and Interests:** Tutors want to manage their availability. System needs accurate data to prevent double-booking.

**Preconditions:** Tutor is authenticated and navigated to availability module.

**Success Guarantee:** Availability is updated and reflected in system calendar.

**Main Success Scenario:**

Tutor logs in and navigates to availability.

Tutor adds/edits available time slots.

System saves updated slots in the database.

**Extensions:** If time slot is invalid, system prompts correction.

Special Requirements:

Slots should be at least 30 minutes and saved in real-time.

Use Case Name: Login/Register

**Scope**: Academic Support Platform System

**Level**: user-goal

**Primary Actor:** Student/Tutor/Advisor

**Stakeholders and Interests:** All users need secure access to the platform.

**Preconditions**: User is not currently authenticated.

**Success Guarantee:** Valid credentials grant access to the dashboard.

**Main Success Scenario:**

User enters credentials.

The system verifies credentials using Authentication Service.

Users are granted access to dashboard.

**Extensions**: If login fails, system displays error and logs attempt.

**Special Requirements:** Authentication must follow institutional security policy.

Use Case Name: Sync Calendar

**Scope:** Academic Support Platform System

**Level:** subfunction

**Primary Actor:** Session Management

**Stakeholders and Interests:** Students and tutors want accurate calendar updates for scheduled sessions.

**Preconditions:** Session must be created or updated.

**Success Guarantee:** Calendar System reflects accurate session timing.

**Main Success Scenario:**

Session is created or updated.

System sends session data to Calendar System.

Calendar System adds it to user’s external calendar.

**Extensions:** If sync fails, system retries or queues the sync.

**Special Requirements**: System should confirm sync and retry if needed.

# UML Use case Diagram

A diagram of a program

AI-generated content may be incorrect.

# Candidate Conceptual Classes - Academic Support Platform System

|  |  |
| --- | --- |
| Conceptual Class Category | Example from Academic Support Platform |
| Physical or Tangible Objects | Student ID Card, Session Report, Feedback Form |
| Specifications, Designs or Descriptions of Things | Session Metadata, Feedback Summary, User Profile |
| Places | Virtual Meeting Room, Institution Portal |
| Transactions | Session Booking, Session Cancellation, Feedback Submission |
| Transaction Line Items | Session Slot, Feedback Entry, Notification Record |
| Roles of People | Student, Tutor, Advisor, Admin |
| Containers of Other Things | Student Profile, Tutor Schedule |
| Things in a Container | Session History, Feedback Entries, Notification Logs |
| Other Computers/Systems (external) | Authentication API, Calendar Integration, Notification API |
| Abstract Noun Concepts | User Role, Access Permission, Session Status |
| Organizations | Online University, Department, Academic Committee |
| Events | Session Scheduled, Session Completed, Feedback Received |
| Processes | Book Session, Join Session, Submit Feedback |
| Rules and Policies | Attendance Policy, Cancellation Policy, Feedback Guidelines |
| Catalogs | Tutor List, Available Services, Feedback Rating Scale |
| Records of Finance, Work, Contracts, Legal Matters, etc. | Tutor Compensation Record, Academic Agreement |
| Financial Instruments and Services | Scholarship Code, Reimbursement Request |
| Groups | Student Cohort, Tutor Pool |
| Communications | System Notification, Reminder Email |
| Schedules | Tutor Availability Schedule, Session Calendar |
| Requests | Support Ticket, Session Request Form |

**Prune EACH list of candidate conceptual classes**

Criteria for pruning:

Eliminate things that are either outside the domain, implementation-specific, or duplicative.

Focus on classes essential to the real-world healthcare domain model.  
  
**Conceptual Class Categorization - Academic Support Platform**

|  |  |
| --- | --- |
| Good Classes (Retained) | Bad Classes (Pruned) |
| Session Report | Student ID Card |
| Session Metadata | Feedback Form |
| User Profile | Virtual Meeting Room |
| Session Booking | Student Profile |
| Feedback Entry | Authentication API |
| Notification Record | Calendar Integration |
| Student | Online University |
| Tutor | Submit Feedback |
| Advisor | External Notification API |
| Tutor Schedule | UI Component |
| Session Status |  |
| Session Completed |  |

# UML Domain Model

A diagram of a program

AI-generated content may be incorrect.

# UML Class Diagram

**1. Eliminated Domain Classes and Associations**

|  |  |
| --- | --- |
| Eliminated Class | Rationale |
| Student ID Card | Physical UI object, irrelevant to system behavior |
| Feedback Form | Interface element; replaced with Feedback entity |
| Virtual Meeting Room | External/implementation-specific resource |
| Calendar Integration | Modeled as an external service, not domain class |
| Submit Feedback (as class) | Behavior, not a domain object |

**2. Pure Fabrications and Indirection**

|  |  |
| --- | --- |
| Class/Element | Justification |
| Notification | Decouples session/feedback logic from communication |
| Calendar (external) | Abstracts calendar logic to external system |
| AuthenticationService | Not a domain class; interface to external identity system |

**3. Polymorphism**

Polymorphism is used via the `User` superclass with the following subclasses:

- Student  
- Tutor  
- Advisor

Shared methods include `login`, `logout`, and `viewSchedule`. Each subclass may override or extend behavior such as booking, managing availability, or viewing feedback.

**4. Generalization and Specialization**

Generalization is applied to the `User` class to support reuse across `Student`, `Tutor`, and `Advisor`. This avoids duplication of shared attributes and enables polymorphic behavior.

A screenshot of a computer screen

AI-generated content may be incorrect.

# UML Sequence Diagram

A diagram of a software project

AI-generated content may be incorrect.

# UML State Diagram

This state diagram models the lifecycle of a Session object in the Academic Support Platform.

The process begins in the Created state, where a session is initially defined with a tutor, student, and time slot. Once availability is confirmed and the session is booked, it transitions to the Scheduled state.

As the session time approaches, it may enter a Pending or Upcoming state based on system rules, indicating readiness for the session to occur.

If the user joins the session successfully, it transitions to Active, representing a session in progress. Once the session is completed, it enters the Completed state, and feedback can then be submitted.

If a session is canceled before it starts, it transitions to the Canceled state. Each transition is triggered by events like book(), join(), complete(), or cancel().

This diagram helps clarify how the session entity behaves in response to user and system actions —by capturing its valid states and transitions throughout its lifecycle

A diagram of a program

AI-generated content may be incorrect.

# UML Activity Diagram

This activity diagram represents the end-to-end process of booking and attending a session in the Academic Support Platform

It begins with the student logging in and browsing available tutors. Once a tutor is selected, the student chooses an available time slot and submits a booking request.

The system then checks the tutor's availability. If the slot is available, the system creates the session and syncs it with the student’s calendar. A confirmation notification is sent to both the student and tutor.

On the day of the session, the student joins through the platform interface. After the session ends, the student is prompted to submit feedback.

Swim lanes are used to clearly separate responsibilities between the student, the system, and external services like Calendar and Notification.

This diagram effectively highlights control flow and collaboration across components, providing a clear view of how user actions translate into system behavior.

**UML Activity Diagram Using Swimlane**

A screenshot of a computer

AI-generated content may be incorrect.

Another example of UML Activity Diagram.

A diagram of a program

AI-generated content may be incorrect.

# UML Component Diagram

**🧩 Component Diagram’s Components:**

**1. User Interface**

* **Responsibilities**: Presents UI to students, tutors, and advisors; handles login and session interactions.
* **Interfaces Provided**:
  + validateCredentials(userId, password)
  + UI events: login, register, view schedule, book session
* **Interfaces Required**:
  + AuthenticationService
  + SessionManagement

**2. Session Management**

* **Responsibilities**: Manages session logic including booking, syncing, availability check, feedback routing.
* **Interfaces Provided**:
  + checkAvailability()
  + sendNotification()
  + syncSession()
  + storeFeedback()
* **Interfaces Required**:
  + Availability Module
  + Notification Service
  + Calendar Integration
  + Feedback Service
  + Database

**3. Availability Module**

* **Responsibilities**: Stores tutor availability and responds to availability checks.
* **Interfaces Provided**:
  + checkAvailability()
* **Interfaces Required**: None

**4. Notification Service *(external)***

* **Responsibilities**: Sends alerts/notifications to users.
* **Interfaces Provided**:
  + sendNotification()
* **Interfaces Required**: None

**5. Feedback Service**

* **Responsibilities**: Handles storing and accessing feedback from sessions.
* **Interfaces Provided**:
  + storeFeedback()
* **Interfaces Required**:
  + Database

**6. Calendar Integration *(external)***

* **Responsibilities**: Syncs session data to user’s calendar.
* **Interfaces Provided**:
  + syncSession()
* **Interfaces Required**: None

**7. Authentication Service *(external)***

* **Responsibilities**: Validates and authorizes user credentials.
* **Interfaces Provided**:
  + validateCredentials(email, password)
* **Interfaces Required**: None

**8. Database (<<DB>> component)**

* **Responsibilities**: Persists session and feedback data.
* **Interfaces Provided**:
  + storeSession()
  + saveFeedback()
* **Interfaces Required**: None

**🧩 Component Interactions**

1. **User Interface → Authentication Service**
   * Initiates user login or registration
   * Calls validateCredentials(userId, password) to authenticate users
2. **User Interface → Session Management**
   * Requests actions like booking a session, joining a session, canceling, or viewing schedule
   * Acts as the front-end gateway for session-related functionality
3. **Session Management → Availability Module**
   * Sends checkAvailability(slotID) to confirm tutor availability during booking
4. **Session Management → Calendar Integration (external)**
   * Calls syncSession(session) to update the user’s external calendar
5. **Session Management → Feedback Service**
   * Delegates storeFeedback(feedbackData) after session completion and user submission
6. **Session Management → Notification Service (external)**
   * Sends sendNotification(message) to alert users of confirmations or changes
7. **Session Management → Database (<<DB>> component)**
   * Stores session data via storeSession()
8. **Feedback Service → Database (<<DB>> component)**
   * Persists feedback records through saveFeedback()

These interactions support:

* Loose coupling (via external services)
* Layered logic flow (UI → core → external systems)
* Responsibility separation (e.g., Feedback logic is not mixed with Notification or Availability)

A diagram of a system

AI-generated content may be incorrect.

# Cloud Deployment Diagram

**Deployment Diagram Description - Academic Support Platform System**

**Deployment Nodes and Their Roles**

**Client Device (Web/Mobile App)**

- Deployed Components: User Interface

- Protocol: HTTPS for secure communication with the App Server

**Web Server / Application Server**

- Deployed Components: System Controller, Authentication Service, Session Management, Availability Module, Feedback Service

- Environment: Docker container, or App Framework (e.g., Node.js/Flask/Django)

- Protocols: Internal RESTful APIs over HTTPS, Service calls via HTTP/JSON

**Database Server**

- Deployed Components: Database Component (stores users, sessions, feedback, availability)

- Environment: MySQL, PostgreSQL, or MongoDB

- Protocol: JDBC, ORM, or SSL-secured DB access

**Communication Links**

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Target | Protocol | Description |
| Client Device | Web/App Server | HTTPS | User logs in, views, and interacts with the system |
| Web/App Server | Database Server | JDBC/SSL | Store/retrieve session, feedback, user availability |
| App Server | Calendar/Notif API | HTTPS | Sync session calendar and push notifications |
| App Server | Authentication API | HTTPS | Credential validation |

**Security Overview**

- HTTPS between User Interface and Backend ensures encrypted traffic.

- JDBC/SSL secures communication with the database.

- Service interfaces between internal components are RESTful and JSON-based.

A diagram of a cloud service

AI-generated content may be incorrect.

# Skeleton Classes and Tables Definition

**Skeleton Classes and Tables Definition**

This section defines the core classes and database schema for the Academic Support Platform system.

**a. Skeleton Class Definitions**

**User**

Attributes: user\_id, name, email, role

Methods: login(password), logout()

**Student (inherits from User)**

Methods: book\_session(session), cancel\_session(session\_id), submit\_feedback(feedback)

**Tutor (inherits from User)**

Methods: set\_availability(slot), view\_feedback()

**Session**

Attributes: session\_id, student, tutor, datetime, status

Methods: join(), cancel()

**Feedback**

Attributes: feedback\_id, session\_id, rating, comments

**b. Database Table Structures**

### **Table: Users**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Constraints |
| user\_id | VARCHAR | PRIMARY KEY |
| name | VARCHAR | NOT NULL |
| email | VARCHAR | UNIQUE, NOT NULL |
| role | VARCHAR | CHECK (Student, Tutor, Advisor) |

### **Table: Sessions**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Constraints |
| session\_id | VARCHAR | PRIMARY KEY |
| student\_id | VARCHAR | FOREIGN KEY → Users |
| tutor\_id | VARCHAR | FOREIGN KEY → Users |
| datetime | TIMESTAMP | NOT NULL |
| status | VARCHAR | CHECK (Scheduled, Cancelled, Completed) |

### **Table: Feedback**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Constraints |
| feedback\_id | VARCHAR | PRIMARY KEY |
| session\_id | VARCHAR | FOREIGN KEY → Sessions |
| rating | INT | CHECK (1–5) |
| comments | TEXT | NULLABLE |

### **Table: Availability**

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Constraints |
| slot\_id | VARCHAR | PRIMARY KEY |
| tutor\_id | VARCHAR | FOREIGN KEY → Users |
| date | DATE | NOT NULL |
| start\_time | TIME | NOT NULL |
| end\_time | TIME | NOT NULL |
| is\_booked | BOOLEAN | DEFAULT FALSE |

# Design Patterns

**Design Patterns and Best Practices**

This section outlines the GRASP, SOLID, GoF, and Microservices design patterns and best practices applied in the Academic Support Platform design.

**1. GRASP Patterns**

**Controller**: The `SessionManager` class acts as a controller, handling requests from the UI and coordinating responses from service components such as Availability, Feedback.

**Information Expert**: Classes like `Session` and `Feedback` encapsulate and manage their own data and logic, promoting cohesion and clarity.

**2. SOLID Principles**

**Single Responsibility Principle (SRP):** Each class like Feedback, Availability, NotificationService is responsible for one clearly defined functionality.

**Open/Closed Principle (OCP):** Interfaces and external services like NotificationService, CalendarService are designed to allow extension without modifying core logic.

**Liskov Substitution Principle (LSP):** Subclasses (student, tutor, advisor) inherit from the base User class and can be used interchangeably where a `User` is expected.

**Interface Segregation Principle (ISP):** Services like Notification and Calendar use small, focused interfaces suited for single use cases.

**Dependency Inversion Principle (DIP):** High-level modules (SessionManager) depend on abstractions, not concrete implementations of services.

**3. GoF (Gang of Four) Patterns**

Observer: The NotificationService observes booking changes and sends alerts to users when sessions are confirmed or cancelled.

-Factory Method: Session creation is abstracted behind a method in SessionManager, allowing flexibility in how sessions are instantiated based on user type.

Strategy (Optional): Could be used to handle different types of notifications (email, SMS, push) without changing the NotificationService logic.

**4. Microservices Design and Best Practices**

Service Decomposition: The system is broken into logical services (Session, Feedback, Notification, Availability) that could be independently deployed and scaled.

Loose Coupling: Services communicate via defined interfaces and could use REST or event-driven architecture.

Scalability and Fault Tolerance: Each service (Notification, Calendar) can scale independently to handle demand.

Separation of Concerns: By isolating responsibilities into components and services, the design enhances maintainability and deployment flexibility.