CSAI 203
Introduction to Software Engineering
Zewail City of Science, Technology and Innovation
University of Science and Technology

Academia Connect

DESIGN PHASE DOCUMENT

Team Number: 01

Team Members:
Omar Amgad Mohamed - 202400515
Mohamed Sherif Farouk - 202401051
Osama Mohsen Abdelaziz - 202401039
Mohamed Islam Elsayed - 202401058

Date: November 23, 2025

1. INTRODUCTION

1.1 Purpose of the Document

The purpose of this Design Phase Document is to serve as the comprehensive architectural blueprint for the "Academia Connect" web platform. It translates the requirements defined in the Software Requirements Specification (SRS) into a technical implementation plan. This document establishes the architectural style, data structures, user interface layouts, and logic flow required to build the system. It acts as the primary reference for the development team to ensure that the resulting software is robust, scalable, and adheres to the specified technical constraints.

1.2 Scope of the Design Phase

This document encompasses the complete technical design of the Academia Connect system. It covers the High-Level Architecture, detailing the monolithic structure using Python Flask. It details the Low-Level Design, specifically the application of the Model-View-Controller (MVC) design pattern. It includes the User Interface Design, describing the layout and structure of key application pages using raw HTML and CSS. Finally, it covers the Data Design, specifying the schema, relationships, and column definitions for the Microsoft SQL Server database, utilizing direct SQL queries for data manipulation.

1.3 Intended Audience

This document is intended for the following stakeholders:

- **The Software Development Team (Team 01):** Who will use this document to implement the backend logic, frontend templates, and database schema.
- **The Project Supervisors and Evaluators:** Who will use this document to verify that the architectural decisions meet the course requirements and constraints.
- **The Quality Assurance Testers:** Who will use the design specifications to create test cases and ensure the system functions as intended.

1.4 Overview of the Contents

Section 2 provides a system overview, outlining the design goals and constraints. Section 3 details the architectural design, including the system diagram and technology stack. Section 4 serves as the core of the document, detailing the MVC pattern, UML class and sequence descriptions, UI wireframes, and the comprehensive database design for Microsoft SQL Server. Section 5 concludes the document with a summary of the design phase.

2. SYSTEM OVERVIEW

2.1 Brief Description of the System

Academia Connect is a centralized web-based platform designed to bridge the operational gap between the Academic Sector (Students and Faculty) and the Industrial Sector (Industry Partners). The system facilitates a structured workflow where Industry Partners submit funded project proposals, Student Teams apply to execute these projects, and Faculty Supervisors provide academic oversight and grading. The system replaces informal communication channels with a formal, auditable web application.

2.2 Key Design Goals and Constraints

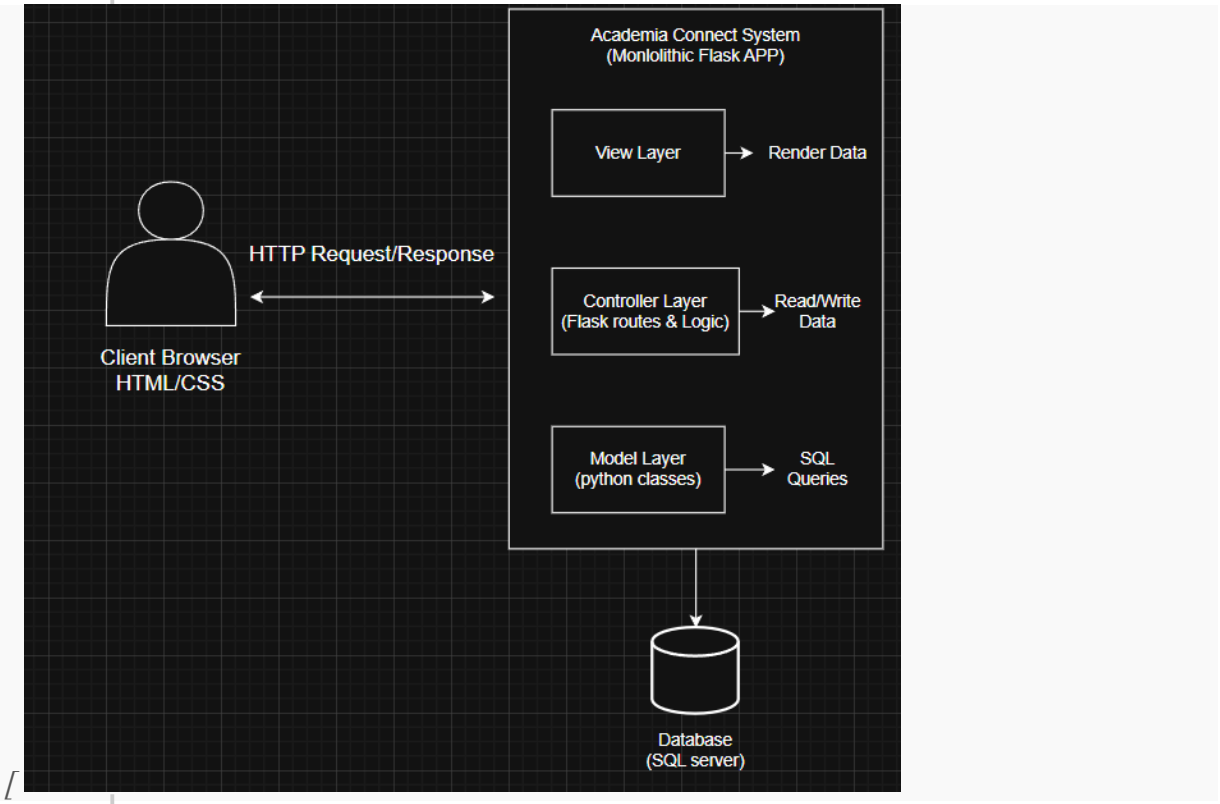
The design of Academia Connect is driven by four primary goals, strictly adhering to the project's technical constraints.

- **Simplicity:** The system utilizes a Monolithic architecture to minimize deployment complexity and reduce latency, ensuring a straightforward development lifecycle for the student team.
- **Modularity:** Despite being monolithic, the codebase adheres to the Model-View-Controller (MVC) pattern to separate concerns, ensuring that business logic, data access, and user interface code remain distinct.
- **Data Integrity:** The system relies on Microsoft SQL Server to enforce referential integrity through rigid primary and foreign key constraints, ensuring that orphan records (e.g., a project without an owner) cannot exist.
- **Constraint Adherence:** The system is designed to operate with the use of raw HTML/CSS for the interface and direct SQL queries for database interactions, prioritizing fundamental software engineering skills over abstraction.

3. ARCHITECTURAL DESIGN

3.1 System Architecture Diagram

The system follows a standard three-tier Web Architecture.



- **The Presentation Tier (Client):** Consists of the user's web browser. It renders HTML and CSS content served by the backend.
- **The Application Tier (Server):** Is hosted on a Python Flask server. This tier handles HTTP requests, manages user sessions, executes business logic.
- **The Data Tier (Database):** Consists of a Microsoft SQL Server instance. It stores all persistent data, including user credentials, project details, and application records. The Application Tier communicates with the Data Tier via a Python database connector using raw SQL commands.

3.2 Discussion of Architectural Style

A Monolithic Architectural Style has been selected for Academia Connect. In this model, the user interface, business logic, and data access layer are integrated into a single deployable unit. This style was chosen because it aligns perfectly with the scope of a university course project. It simplifies the development and testing process, as there is no need to manage microservices or complex inter-service communication. It allows for rapid prototyping and debugging within a single development environment.

3.3 Technology Stack and Tools

- **Programming Language:** Python is used for all backend logic.
- **Web Framework:** Flask is used as the web server gateway interface (WSGI) application.
- **Database:** Microsoft SQL Server is the relational database management system.

- **Database Connector:** pyodbc (or similar standard connector) is used to execute raw SQL queries from Python.
- **Frontend:** HTML and CSS are used for page structure and styling.
- **Version Control:** Git and GitHub are used for source code management.
- **IDE:** Visual Studio is the primary development environment.

4. DETAILED DESIGN

4.1 MVC Design Pattern

4.1.1 Description of MVC

The application implements the Model-View-Controller (MVC) design pattern. This pattern decouples the data access logic (Model) from the user interface (View) and the control flow logic (Controller).

4.1.2 Mapping Components

- **Model:** Implemented in Python files (e.g., models.py). These classes represent the data structures (User, Project) and contain methods that execute raw SQL queries (INSERT, SELECT, UPDATE) against the MS SQL Server.
- **View:** Implemented as HTML files stored in the templates/ directory.
- **Controller:** Implemented in Python files (e.g., routes.py or app.py). These functions handle incoming HTTP requests, process input validation, call the Model methods to fetch or save data, and determine which View template to render.

4.1.3 Responsibilities of each layer

- The **Model** is responsible for direct interaction with the Microsoft SQL Server. It handles opening connections, executing SQL statements, and closing connections. It does not contain any HTML code.
- The **View** is responsible for presenting data to the user. It receives data objects from the Controller and formats them into HTML tables, lists, or forms. It does not contain business logic or SQL queries.
- The **Controller** is the traffic cop. It receives the user's request (e.g., "Login"), retrieves the necessary data from the Model, performs logic checks (e.g., password verification), and returns the appropriate View (e.g., "Dashboard").

4.1.4 Interaction Flow (Login Example)

1. The user navigates to the Login page (View).
2. The user enters credentials and clicks "Submit".
3. The browser sends a POST request to the /login route (Controller).
4. The Controller extracts the username and password.
5. The Controller calls the User Model method verify_user(username, password).
6. The Model executes a SELECT query against the MS SQL Server to find the user.
7. If valid, the Controller sets a session variable and redirects the user to the Dashboard route.
8. The Dashboard route renders the Dashboard.html template (View).

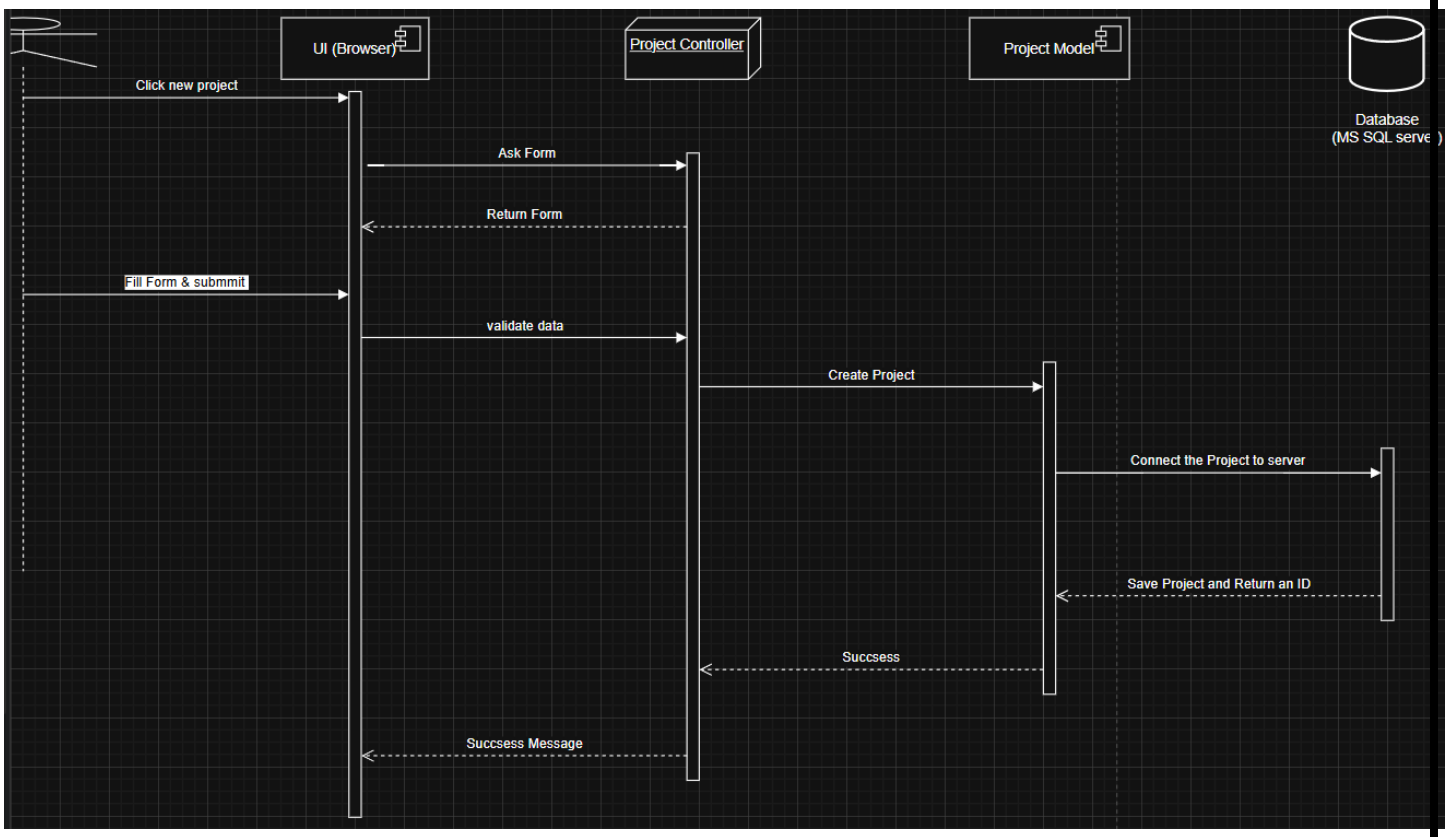
4.2 UML Diagrams

4.2.2 Detailed Class Diagram Description

The system is built upon four primary classes:

- **The User Class:** Represents all actors in the system. It contains attributes for UserID, Username, PasswordHash, Email, and Role. It serves as the parent entity for role-specific logic.
- **The Project Class:** Represents the proposals submitted by Industry. It contains attributes for ProjectID, Title, Description, FundingAmount, and Status. It has a many-to-one relationship with the User class (specifically the Industry Partner).
- **The Team Class:** Represents a group of students. It contains attributes for TeamID, TeamName, and CreationDate. It has a one-to-many relationship with the User class (Student Members).
- **The Application Class:** Represents a team's intent to work on a project. It contains attributes for ApplicationID, SubmissionDate, and Status. It acts as an associative entity linking the Team Class and the Project Class.

4.2.3 Sequence Diagram Description (Submit Project Proposal)

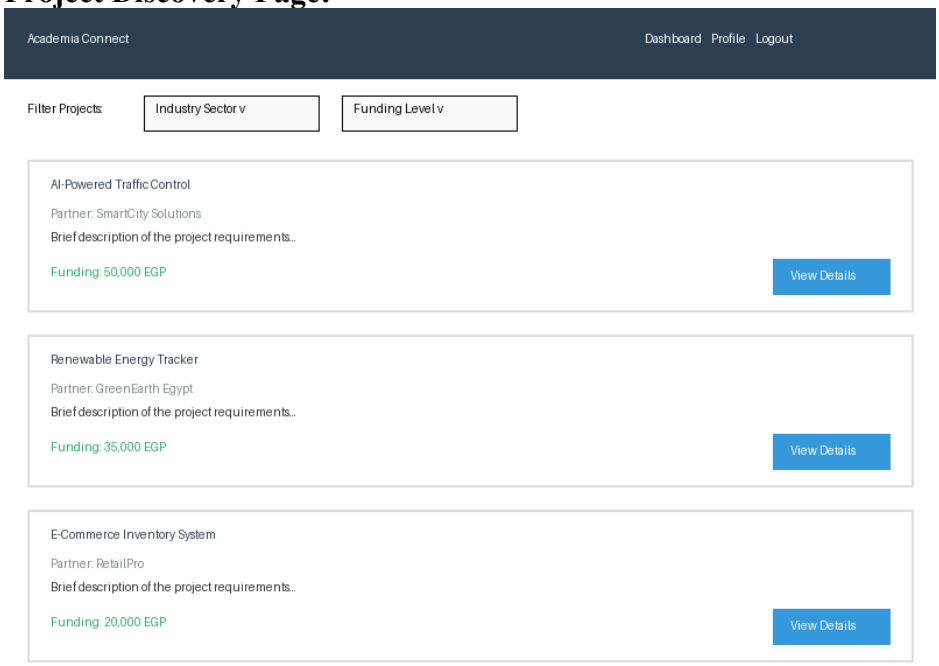


1. The flow begins when the Industry Partner Actor clicks the "New Project" button on the UI.
2. The Browser sends a GET request to the Controller, which returns the "Create Project" form (View).
3. The Industry Partner fills in the details and submits the form.
4. The Browser sends a POST request with the form data to the Controller.
5. The Controller validates the input (checking for empty fields).
6. The Controller creates a new Project Object and calls the save_project() method in the Model.
7. The Model opens a connection to the MS SQL Server and executes an INSERT INTO Projects SQL command.
8. The Database confirms the insertion and returns the new Project ID.
9. The Controller flashes a success message and redirects the Industry Partner to their Dashboard View.

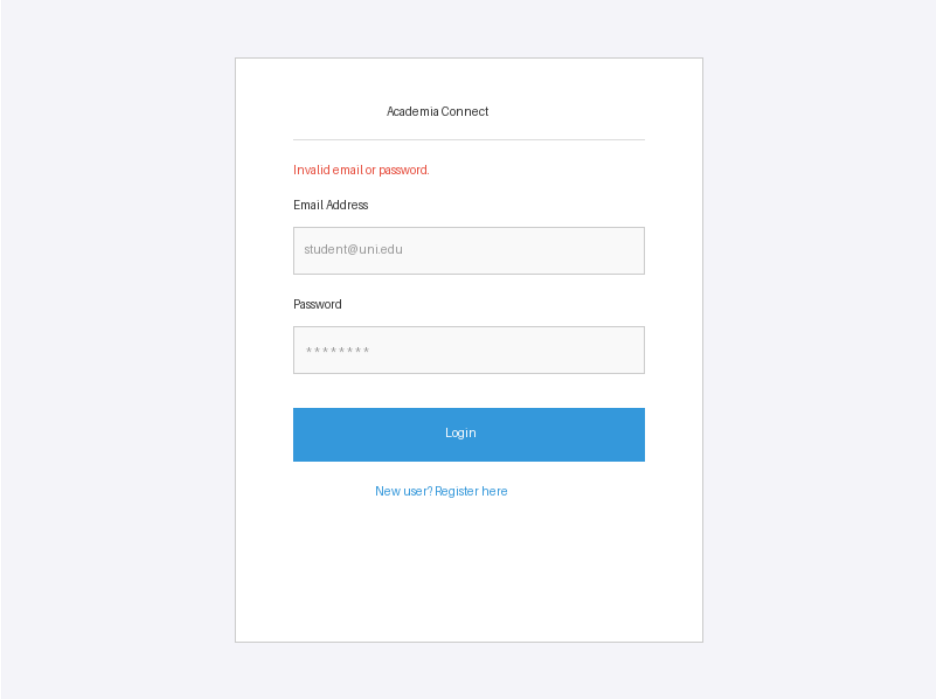
4.3 UI/UX Design

4.3.1 Wireframe Descriptions

Project Discovery Page:



Login Page:



4.4 Data Design

4.4.1 Database Schema

The database is implemented on Microsoft SQL Server.

- **The Users Table:** Stores authentication details. The Primary Key is UserID (Integer, Identity).
- **The Projects Table:** Stores proposal data. The Primary Key is ProjectID (Integer, Identity). It contains a Foreign Key, OwnerID, referencing Users(UserID).
- **The Teams Table:** Stores student groups. The Primary Key is TeamID (Integer, Identity). It contains a Foreign Key, SupervisorID, referencing Users(UserID) for the Faculty Supervisor.
- **The Applications Table:** Stores project applications. The Primary Key is ApplicationID (Integer, Identity). It contains two Foreign Keys: TeamID referencing Teams(TeamID) and ProjectID referencing Projects(ProjectID).

4.4.2 Data Storage Model

Database connection parameters (Server IP, Port, Database Name, User, Password) are stored in a secure configuration file or environment variables, not hardcoded in the source code. The application uses a Python DB-API compliant connector to establish connections. For file deliverables, such as PDF reports uploaded by students, the system uses the server's local filesystem. The database stores only the relative file path (string) to the document, minimizing database bloat and maintaining performance.

4.4.3 Data Dictionary

- **Users table:** UserID (INT, PK, Auto-increment), Username (VARCHAR(50), Unique), Email (VARCHAR(100), Unique), PasswordHash (VARCHAR(255)), and Role (VARCHAR(20), restricting values to Admin, Industry, Faculty, Student).
- **Projects table:** ProjectID (INT, PK, Auto-increment), Title (VARCHAR(150)), Description (TEXT), FundingAmount (DECIMAL(10, 2)), Status (VARCHAR(20)), and IndustryID (INT, FK referencing Users).
- **Teams table:** TeamID (INT, PK, Auto-increment), TeamName (VARCHAR(100)), CreationDate (DATETIME), and LeaderID (INT, FK referencing Users).
- **Applications table:** ApplicationID (INT, PK, Auto-increment), ProjectID (INT, FK referencing Projects), TeamID (INT, FK referencing Teams), SubmissionDate (DATETIME), and Status (VARCHAR(20)).

5. CONCLUSION

5.1 Summary of Design Phase

This Design Phase Document has outlined the architectural and technical specifications for the Academia Connect platform. By adopting a Monolithic architecture with a Python Flask backend and a Microsoft SQL Server database, the design prioritizes simplicity and data integrity while using raw HTML/CSS and direct SQL queries. The MVC pattern ensures that the code will remain modular and maintainable. This document now serves as the definitive guide for Team 01 to begin the implementation phase.