PHASE 3: DESIGN DOCUMENT

UniConnect

Zewail City of Science, Technology and Innovation

CSAI 203- Fall 2025

contact: s-nour.ghoniem@zewailcity.edu.eg

Members	
NAME	**Id**
Nour Hosam	202401560
Ganna Allah	202401020
Shaza Kazem	202400948
Shehab Osama	202402196
Malak Mohamed	202401538

1. Introduction

1.1 Purpose of the Document

The purpose of this Design Document is to convert the requirements in the Software Requirements Specification (SRS) into a technical guide for building the UniConnect system. It gives a detailed description of the system's architecture, user interfaces, and the use of the Model-View-Controller (MVC) design pattern. This document is the reference for the development team to make sure the software is built according to the requirements and design standards.

1.2 Scope of the Design Phase

This document covers the architectural and detailed design of UniConnect web platform. The scope includes:

· System Architecture: Defining the architecture and how the Client and Server interact.

· MVC Implementation: Mapping the system components to the Model, View, and Controller layers.

· Data Design: Defining the structure of the in-memory data storage (Python lists and dictionaries) used to manage users, clubs, and offices.

· User Interface Design: Providing frames for actions such as browsing clubs and viewing contact details.

1.3 Intended Audience

This document is meant for:

· The Development Team: To understand the system structure and implement the features consistently.

· Future Maintainers: To understand the code structure for future updates or to add external database in the future.

1.4 Overview of Contents

The rest of this document is organized as follows:

· Section 2 (System Overview): A description of UniConnect and its design goals.

· Section 3 (Architectural Design): A visual and descriptive look at the system's architecture.

· Section 4 (Detailed Design): A close examination of the MVC pattern application, UML class and sequence diagrams, UI frames, and data models.

2. System Overview

2.1 Brief Description of the System

UniConnect is a web-based system designed for Zewail City of Science and Technology. It acts as a platform connecting students with university clubs, committees, and administrative offices. The system allows students to browse club details, view their details, and contact leaders. It makes communication easier by bringing together contact information (emails, roles, and photos) into one clear interface. The system uses the Flask framework and follows an MVC architecture.

2.2 Key Design Goals and Constraints

Design Goals:

· Modularity: The system follows the Model-View-Controller (MVC) pattern to separate data, logic, and user interface, which makes the code easier to maintain.

· Simplicity and Usability: The interface is designed to be clear for all students and staff, focusing on clear navigation and responses.

Constraints:

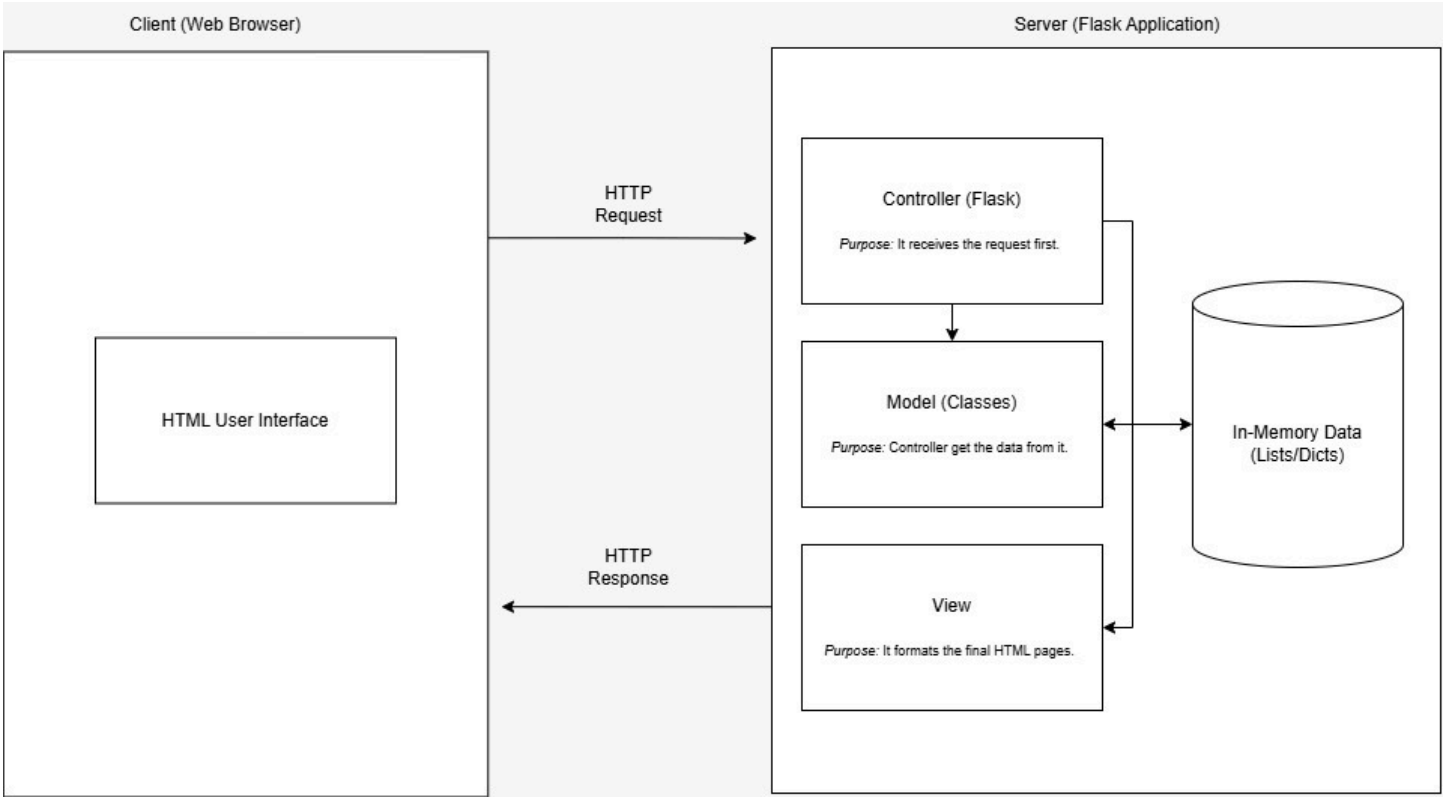
· Technology Stack: The backend implemented using Python Flask and the frontend uses HTML with CSS.

· Data Storage: Currently, there is no external database (SQL) used, data is temporarily stored in Python lists and dictionaries, meaning data resets when the server stops.

· Deployment: The system operates in Localhost.

3. Architectural Design

3.1 System Architecture Diagram



3.2 Discussion of Architectural Style and Components

Architectural Style: Monolithic

We built UniConnect using a **Monolithic Architecture**. This simply means that the whole application, the design, the code, and the data storage runs together as one single program on our Flask server. We chose this style because it is simple to build and easy for our team to test without needing complex setups.

The Main Components:

1. **Client (Web Browser)**

· **HTML User Interface:** This is what the student or admin sees on their screen. It shows our web pages with the forms, buttons, and club details.

· **How it works:** When a user clicks a button, the browser sends a request to our server. When the server is done, the browser displays the new page sent back to it.

1. **Server (Flask Application)** This is the brain of the system. It runs the Flask code and follows the **MVC (Model-View-Controller)** pattern to keep things organized.

· **Controller (Flask):** Let's think of this as the manager. It is the first thing that receives a request from the user. It decides what to do, asks the Model for information, and picks the right page to show.

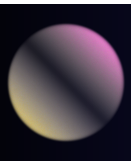
· **Model (Classes):** This part handles the logic and data. The Controller talks to the Model to get data or save updates.

· **In-Memory Data (Lists/Dicts):** Since we are not using a real external database (like SQL) yet, we store all our data inside Python lists and dictionaries. This allows the system to work fast, but the data resets if we turn off the server.

· **View:** This handles the look of the system. Once the Controller has the data, it passes it to the View. The View uses Jinja2 templates to build the final HTML page that the user sees.

3.3 Technology Stack and Tools

- **Backend:** Python Flask (handles the logic).
- **Frontend:** HTML and CSS (makes the pages look beautiful).
- **Data Storage:** Python Lists and Dictionaries (stores data temporarily).
- **Tools:** GitHub (for saving our codes) and VS Code (for writing code).



2. System Overview

2.1 Brief Description of the System

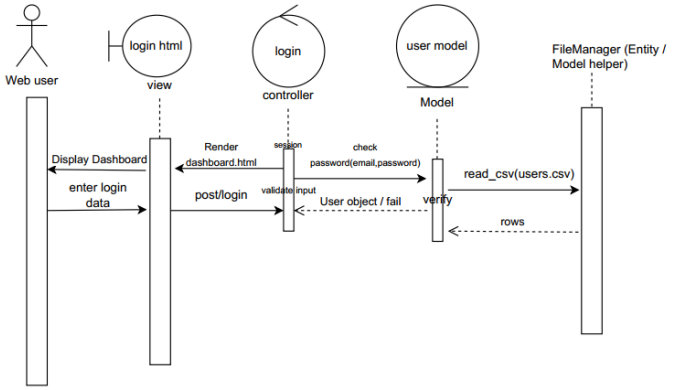
- In the design phase, UniConnect is planned in a simple and organized way so the whole system feels clean, easy to follow, and straightforward for both developers and users. The idea is to build everything around the MVC structure, which helps keep the system neat. The Model is where all the data for clubs, offices, users, and announcements is handled. The View is the actual interface the user sees, like the HTML pages and the layouts. The Controller, which is Flask in our case, connects everything together and responds to whatever the user clicks or searches for.
- The design mainly focuses on making the user experience as smooth as possible. Every page is created with the purpose of helping the user find information quickly without extra steps. The home page gives the user an easy starting point where they can search or explore. The clubs and offices pages are designed in a clean card layout that makes browsing simple and visual. The details pages then show all the important information clearly, including the people in charge, their photos, and contact information.
- Overall, the system is designed to feel light, organized, and very easy to navigate. The structure from the inside is clear, and the interface from the outside is simple, so the user never feels lost. The goal in this phase is to make sure UniConnect looks clean, works smoothly, and delivers the information students need in a direct and comfortable way.

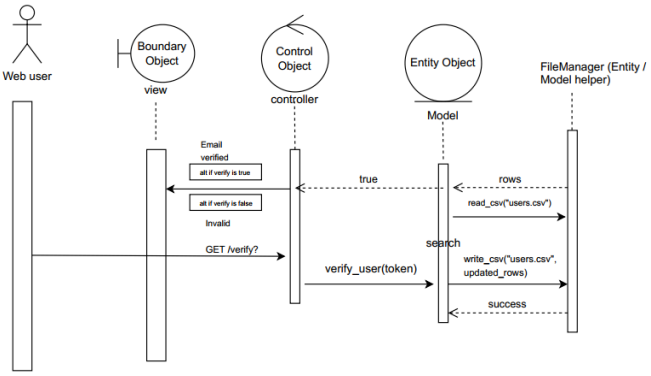
2.2 Key Design Goals and Constraints

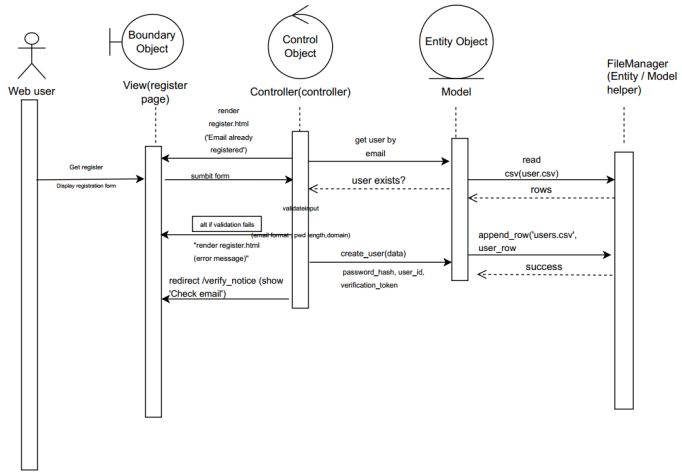
- In the design phase of UniConnect, the main goal is to build a system that feels clean, easy to navigate, and straight to the point. The whole idea is to let students and staff find the information they need without confusion or unnecessary steps. So one of the biggest design goals is simplicity — keeping the interface light, organized, and not crowded. The pages should guide the user naturally, with clear layouts and obvious actions, whether they're searching for a club, checking office contacts, or viewing someone in charge.
- Another important design goal is consistency. The system should follow the same structure, style, and flow across all pages so it feels like one complete platform. Colors, fonts, card styles, and spacing should all match to make the experience smooth and predictable. The design also aims to support fast navigation, where the user can reach any piece of information in just one or two clicks.
- However, while designing, there are also a few constraints we need to consider. Since this phase uses temporary data without an actual database, the design must stay simple enough to work with Python structures like lists and dictionaries. This limits how complex the data interactions can be. Another constraint is that the system is built using Flask with basic HTML and CSS, so the design has to stay realistic and implementable within our current tools and skills. We also need to make sure the UI elements are not overly complex or hard to code.
- Overall, the design goals push us toward creating a clean, friendly, and easy system, while the constraints remind us to keep the design practical, simple, and suitable for the technologies we're using right now.

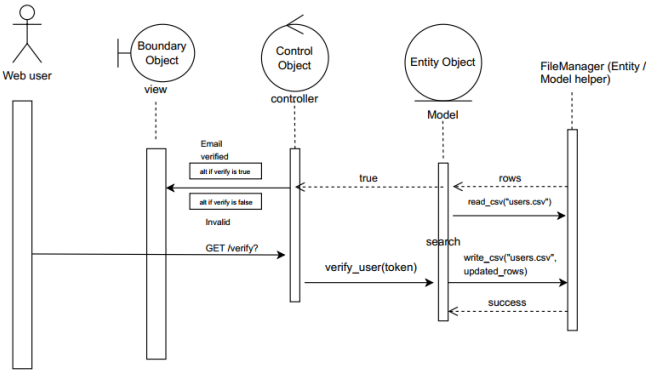
3. Architectural Design

3.1 System Architecture Diagram









4. Detailed Design

Description of MVC Pattern (1.1)

- Model:
  - Handles the data and internal logic of the system.
  - In our sequences:
    - UserModel → manages users, email verification, login checks
    - FileManager → reads/writes CSV files
    - ClubModel → manages club data, search and filtering
- View:
  - The interface the user interacts with.
  - HTML pages: register.html, login.html, verify_result.html, clubs.html, club_details.html
- Controller:
  - Mediates between View and Model, processing requests and sending responses.
  - In our sequences:
    - AuthController → registration, login, email verification
    - ClubsController → search and filtering, rendering results

Responsibilities of Model, View, and Controller (4.1.3)

Feature	Model	Controller	View
Registration	UserModel + FileManager: create_user, verify_user, save CSV	AuthController: GET & POST /register, validate input, call Model	register.html: form + error/success messages
Verification	UserModel + FileManager: search token, set is_verified, write CSV	AuthController: GET /verify, call verify_user, select View	verify_result.html: show success/fail
Login	UserModel: check_password, verify is_verified	AuthController: GET & POST /login, validate, set session	login.html: form + error messages
Search/Filter Clubs	ClubModel + FileManager: read CSV, filter results	ClubsController: POST /search, send filtered list	clubs.html: display club cards

Interaction Between Components (4.1.4)

A. Registration Sequence

1. User → View: GET /register → display registration form
2. User → Controller: POST /register with data
3. Controller validates input → calls Model: get_user_by_email
4. Model → FileManager: read users.csv → return rows
5. Model checks email existence:
   - If exists → Controller → View: display error
   - Else → create user → hash password → generate token → append_row CSV
6. Controller → View: show “Check email to verify” page

B. Email Verification Sequence

1. User → Controller: GET /verify?token=xxx
2. Controller → Model: verify_user(token)
3. Model → FileManager: read CSV, search token
4. If token found: set is_verified = True, write CSV
5. Model → Controller: return True / False
6. Controller → View: show verify_result.html (success/fail)

C. Login Sequence

1. User → View: login.html
2. User → Controller: POST /login (email/password)
3. Controller → Model: check_password + is_verified
4. Model → Controller: return User object / False
5. Controller sets session → redirect to dashboard
6. Controller → View: display dashboard.html

D. Search/Filter Clubs Sequence

1. User → View: submit search query (clubs.html)
2. View → Controller: POST /search (query)
3. Controller → Model: get_clubs_by_query(query)
4. Model → FileManager: read clubs.csv
5. Model filters results → return filtered list to Controller
6. Controller → View: render clubs.html with filtered cards

4.2 UML Diagrams

4.2.1 UML Diagrams – Introduction and Explanation

The Unified Modeling Language (UML) basically helps us turn everything we wrote in the SRS into actual visual diagrams that show how the system works. Instead of just describing things in words, UML lets us draw the structure of UniConnect and how its parts interact, which makes the whole system much easier to understand. It connects the high-level requirements to the actual design we will build in code.

Using UML is very helpful for UniConnect because the system has different user roles, several modules, and a bunch of entities that need to communicate with each other. By drawing these relationships, everyone—whether developers or team members—gets the same clear picture of how the system works before we write any code. This removes confusion and helps make sure the final system matches what we planned in the SRS.

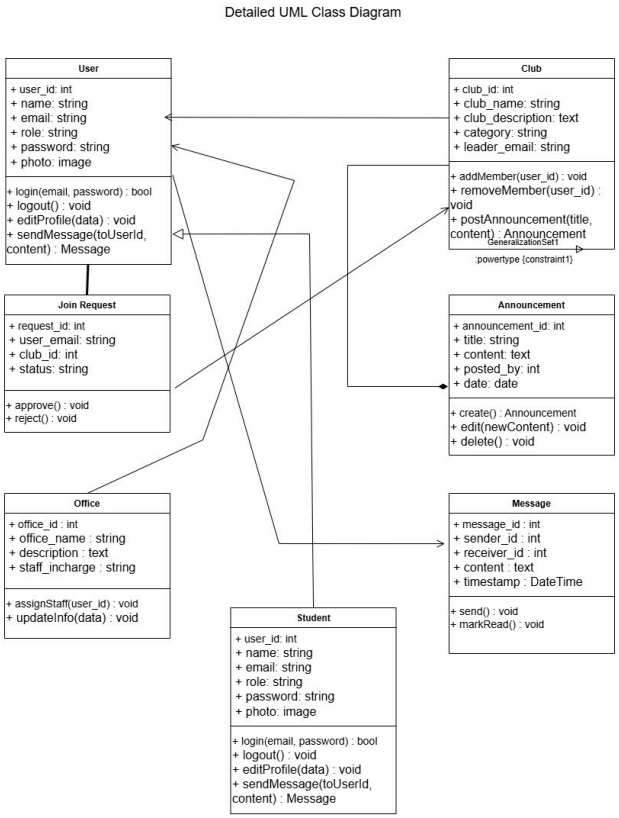
In this project, we mainly use two UML diagram types:

- Class Diagram, which shows the main classes in UniConnect, their attributes, functions, and how they relate.
- Sequence Diagrams, which show how the system behaves step-by-step, like how a student sends a join request or how an admin posts an announcement.

These diagrams connect directly to the SRS. The Use Case Diagram in the SRS shows the main actions like browsing clubs, contacting leaders, sending join requests, and reading announcements. In the design phase, we take those actions and turn them into more detailed sequence diagrams. Also, the Data Dictionary in the SRS (with entities like User, Club, Office, Announcement, JoinRequest, and Message) is exactly what the class diagram is built from.

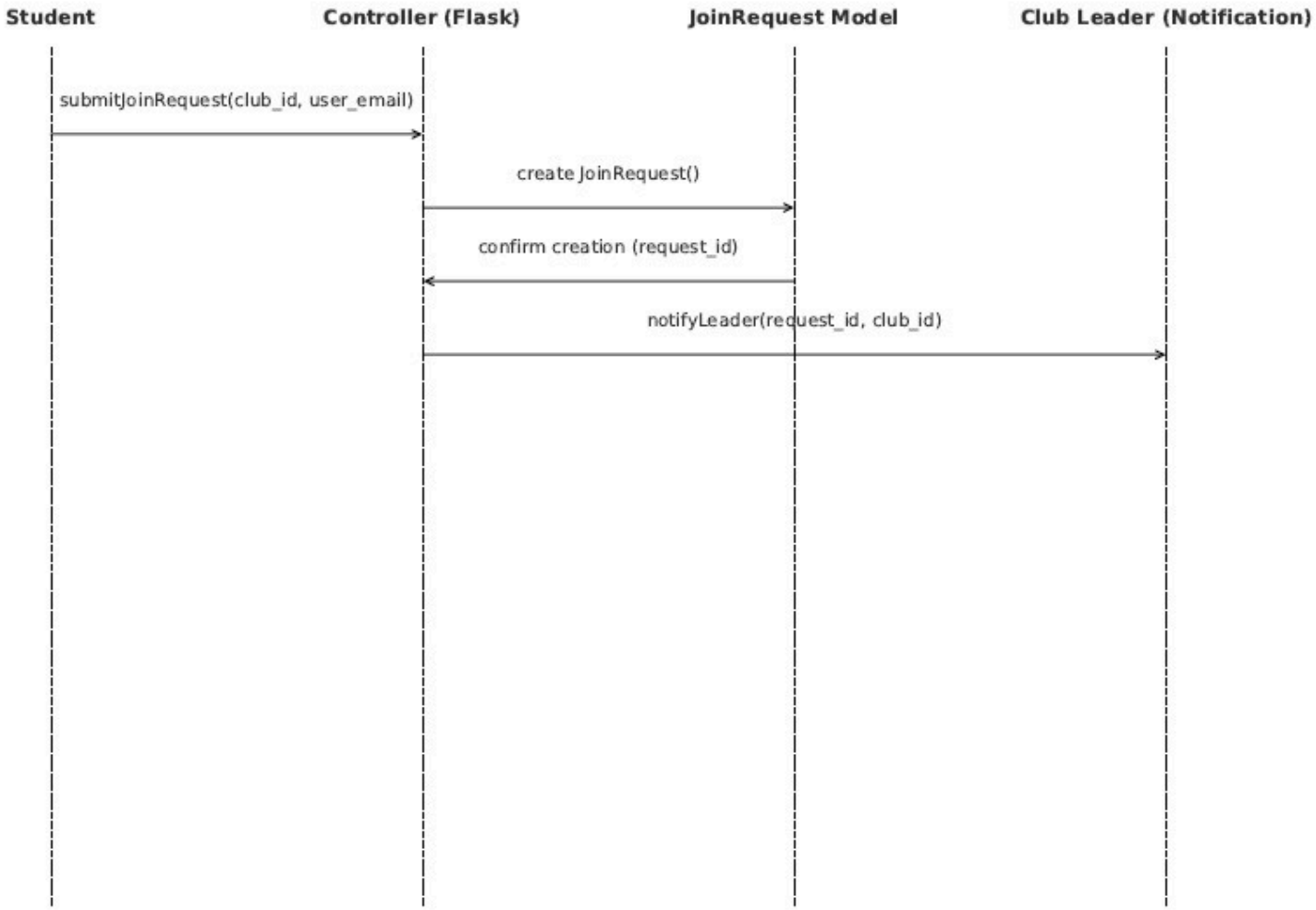
So overall, moving from SRS text to UML diagrams helps us create a clear, complete blueprint of UniConnect. It shows how everything fits together and prepares us for the implementation phase.

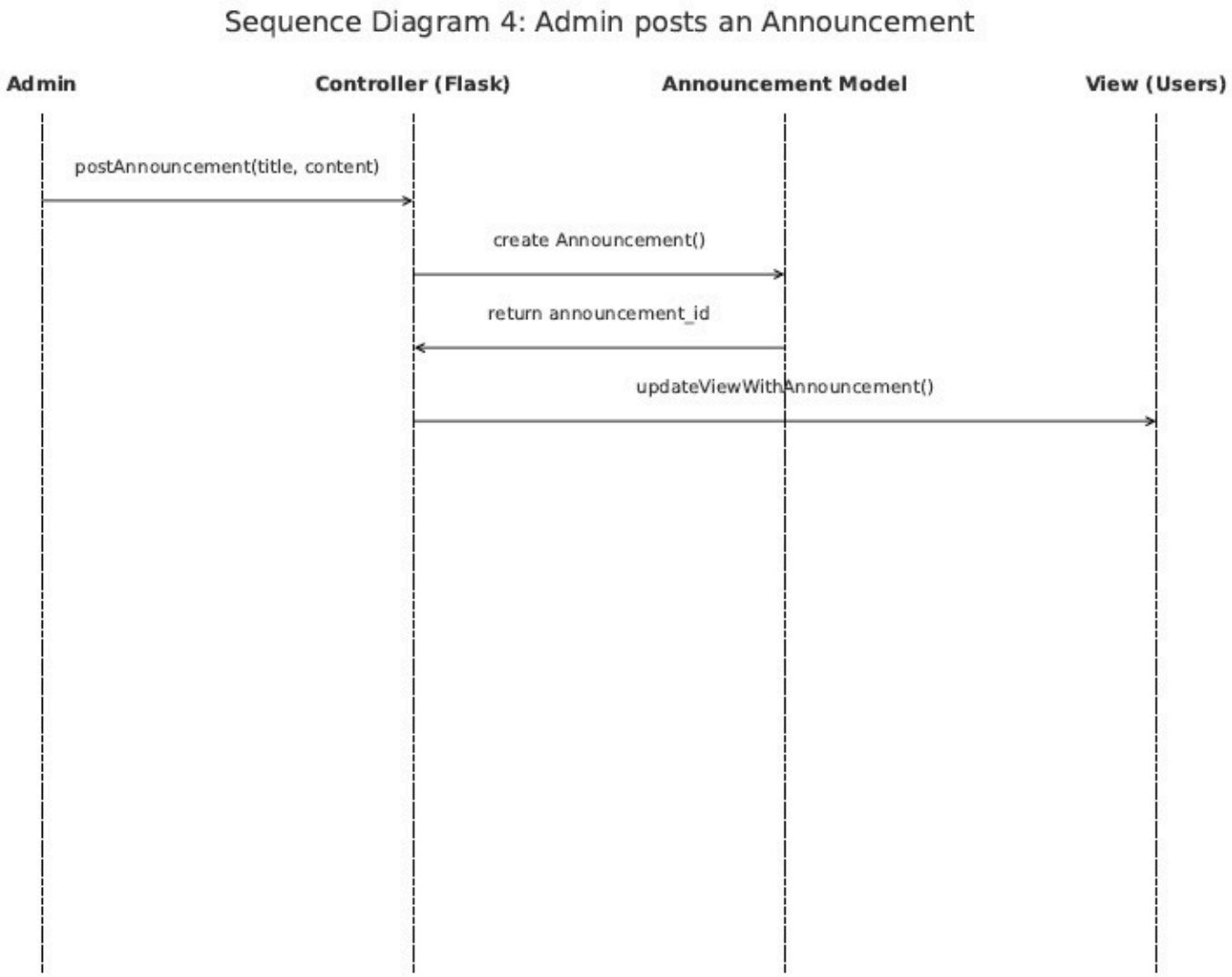
4.2.2 Detailed Class Diagram



4.2.3 Sequence Diagrams

Sequence Diagram 3: Student sends a Join Request

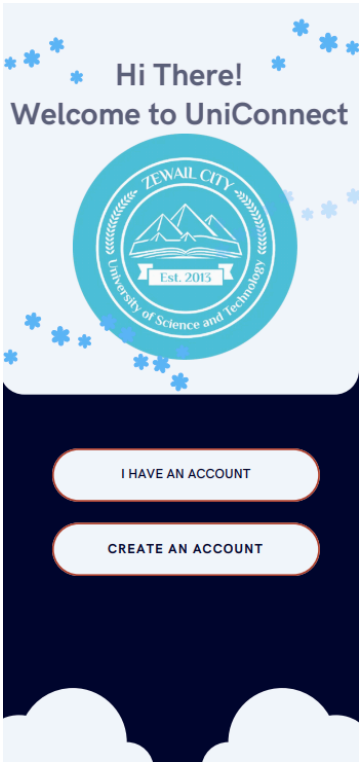


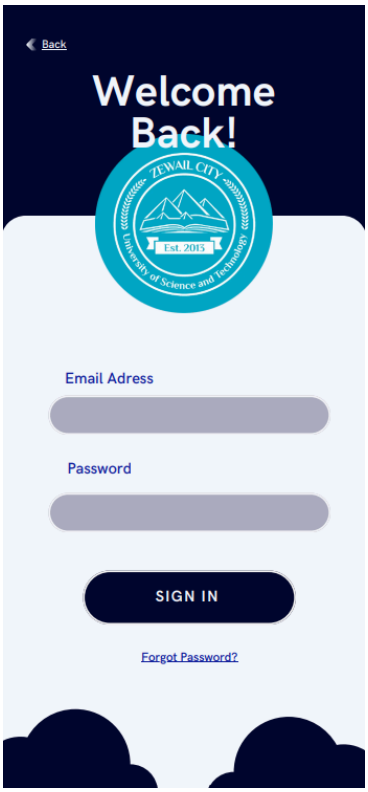


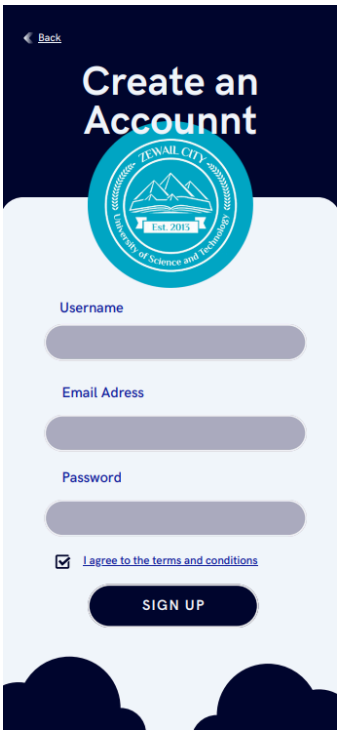
4.3 UI/UX Design

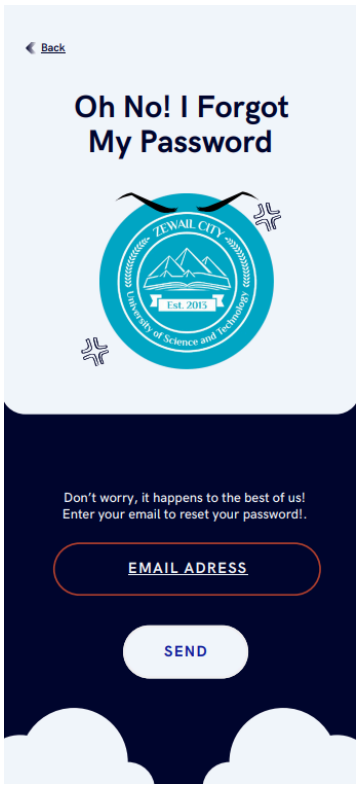
The UniConnect application is designed as a web-based system, focusing on a responsive interface that delivers seamless access to essential university directories. The design ensures clarity, efficient navigation, and consistent branding aligned with Zewail City University of Science and Technology.

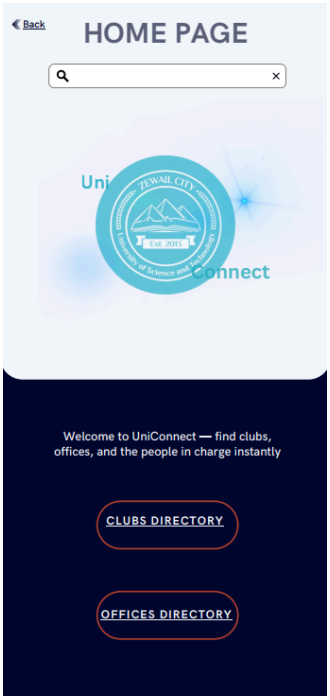
4.3.1 Wireframes / Mockups (User Interface Screens/Layouts)

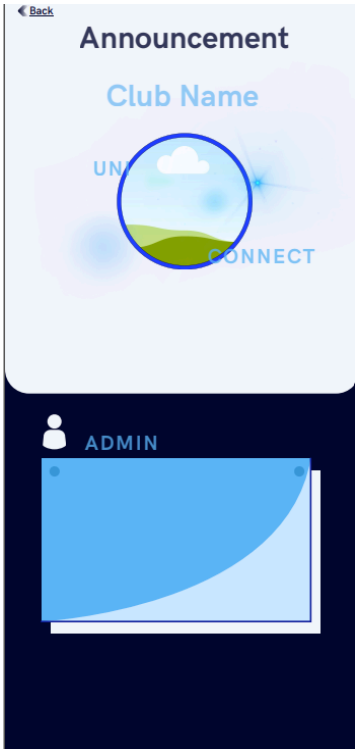

Hi There!
Welcome to UniConnect
ZEWAIL CITY
Est. 2015
University of Science and Technology
I HAVE AN ACCOUNT
CREATE AN ACCOUNT

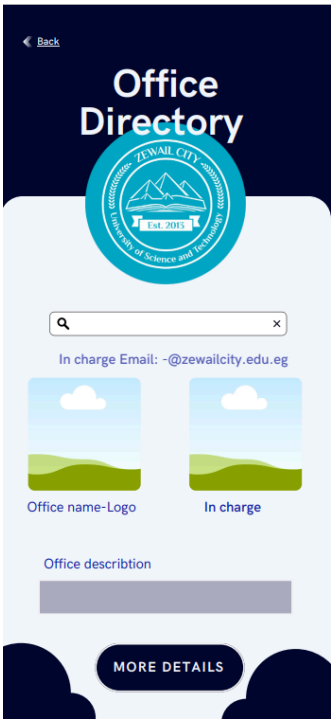

Back
Welcome Back!
ZEWAIL CITY
Est. 2015
University of Science and Technology
Email Adress
Password
SIGN IN
Forgot Password?

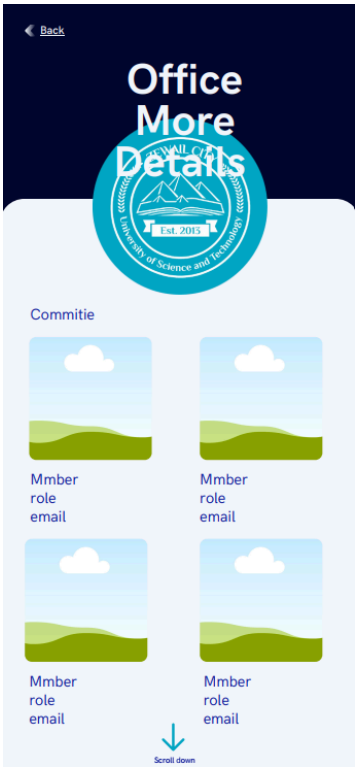

Back
Create an Accounnt
ZEWAIL CITY
Est. 2015
University of Science and Technology
Username
Email Adress
Password
I agree to the terms and conditions
SIGN UP

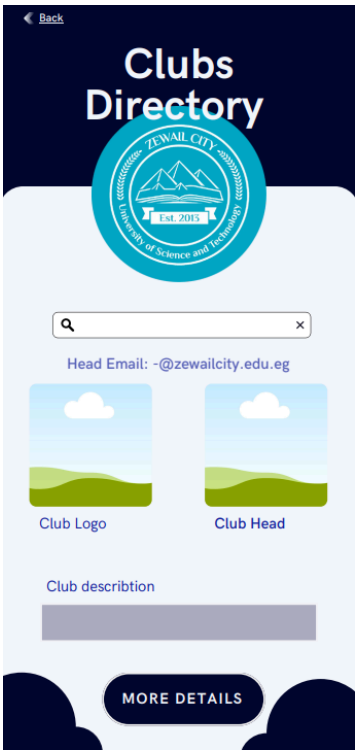

Back
Oh No! I Forgot My Password
ZEWAIL CITY
Est. 2015
University of Science and Technology
Don't worry, it happens to the best of us!
Enter your email to reset your password!.
EMAIL ADRESS
SEND

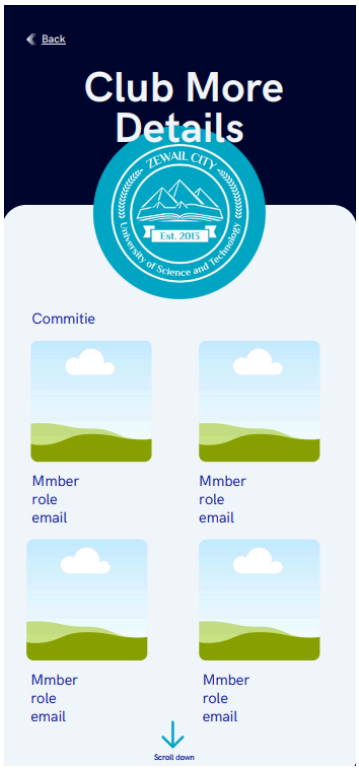

Back
HOME PAGE
Uni
Connect
Welcome to UniConnect — find clubs, offices, and the people in charge instantly
CLUBS DIRECTORY
OFFICES DIRECTORY


Back
Announcement
Club Name
UNI
CONNECT
ADMIN


Back
Office Directory
In charge Email: -@zewailcity.edu.eg
Office name-Logo
In charge
Office describtion
MORE DETAILS


Back
Office More Details
Commitie
Mmber
role
email
Mmber
role
email
Mmber
role
email
Mmber
role
email
Scroll down





UI-UX flow

1. Authentication and Account Management Flow

This sequence governs how a user accesses the system or manages their account credentials.

- Initial Entry: The user first arrives at the Welcome Screen.
  - The screen prompts a decision: either sign in as an existing user (I HAVE AN ACCOUNT) or register as a new user (CREATE AN ACCOUNT).
- Sign In Path:
  - The user is directed to the Sign In Screen.
  - They input their Email Adress and Password and submit the credentials.
  - *System Response:* If credentials are valid, the user is granted access and moves to the Home Page.
  - Account Recovery Sub-Path: If the user cannot log in, they initiate the password reset process via the Forgot Password? link, which leads to the Forgot Password Screen. Here, they enter their EMAIL ADRESS and click SEND. The system then handles the off-app email communication for recovery.
- Sign Up Path:
  - The user is directed to the Create Account Screen.
  - They must provide a Username, Email Adress, and a new Password.
  - Registration requires explicit consent by checking the box indicating they agree to the terms and conditions.
  - Upon clicking SIGN UP, the account is created, and the user is directed to the Home Page.

2. Main Navigation and Directory Access Flow

After successful authentication, the user lands on the primary hub of the application.

- Home Page: This is the central control panel.
  - The user has immediate access to the Search functionality to quickly find resources.
  - The main access points are the directory links: CLUBS DIRECTORY and OFFICES DIRECTORY.

    - *User Action:* The user clicks on either directory link to proceed to the main listings.
- Directory Index Views:
    - The user arrives at the Clubs Directory or Offices Directory listing.
    - Each directory presents a high-level summary of organizations, including the organization's Logo, a brief Description, and the contact Email of the person In charge or the Club Head.

3. Detailed Information Retrieval Flow

This sequence allows the user to investigate the personnel structure within a specific organization.

- Detail Initiation: From any listing on the Directory Index View (Clubs or Offices), the user clicks the "MORE DETAILS" button associated with a specific organization.
- Club or Office Detail View:
    - The application transitions to the detailed view, which focuses on the organization's Commitie (Committee) or staff structure.
    - The core information presented is the roster, where each Mmber (Member) is listed with their specific role and contact email.
    - The view is designed for easy navigation, including a "Scroll down" indicator for extensive rosters.

4.4 Data Design / Database Schema Description

1. Main Data Entities

a. User

Represents anyone who interacts with the system—students, club leaders, office staff, or administrators.

It stores essential profile information such as name, email, password, role, and photo.

b. Club

Contains details about each university club, including its name, description, category, and leader.

c. Office

Stores basic information about official university offices, such as the office name, description.

d. Request

Tracks requests submitted by students who want to join different clubs. It also records the status of each request (pending, approved, or rejected).

e. Announcement

Represents announcements created by clubs or offices. Each announcement is linked to whoever posted it (a user) and to the club or office it belongs to.

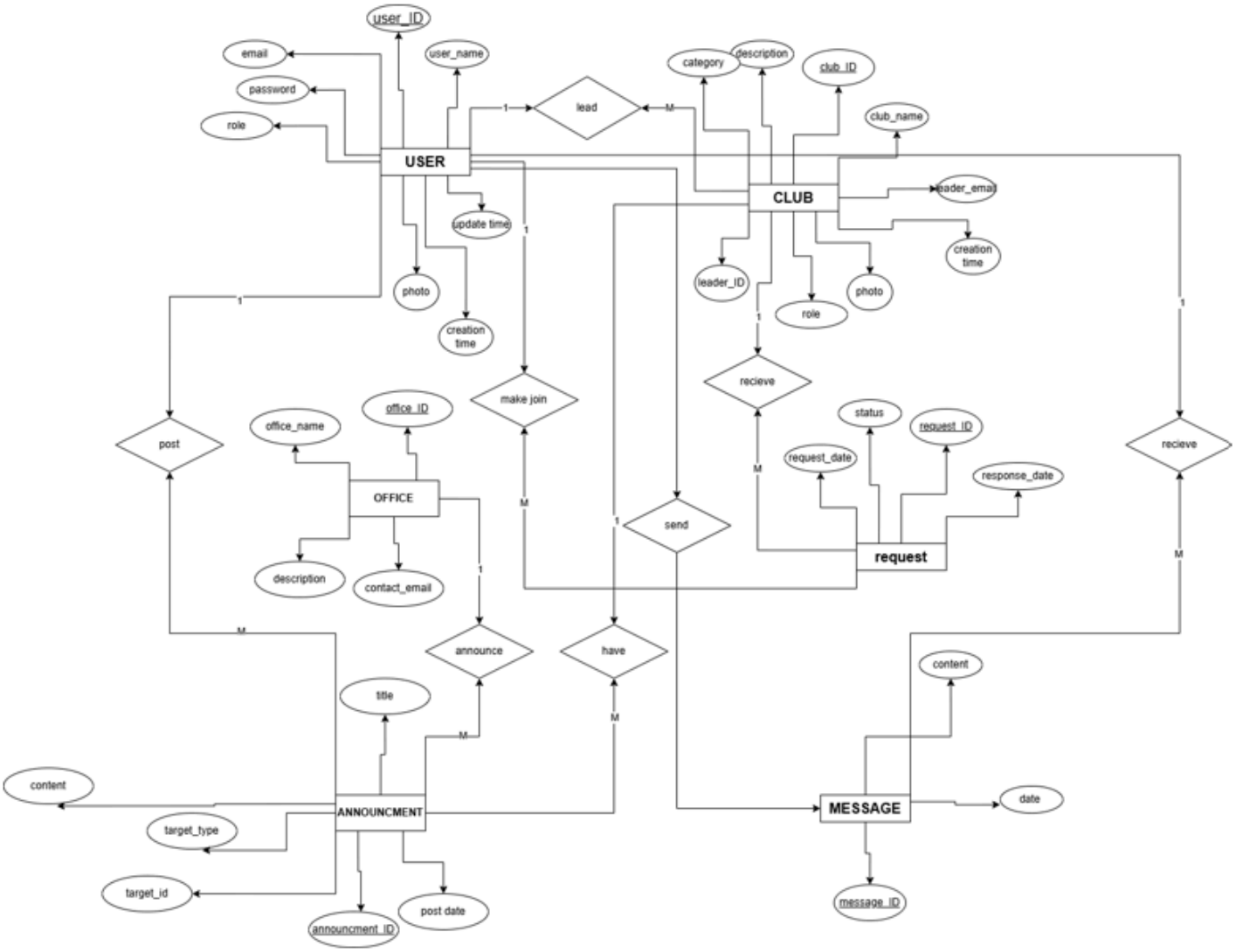
f. Message

Stores private messages exchanged between users and read/unread status.

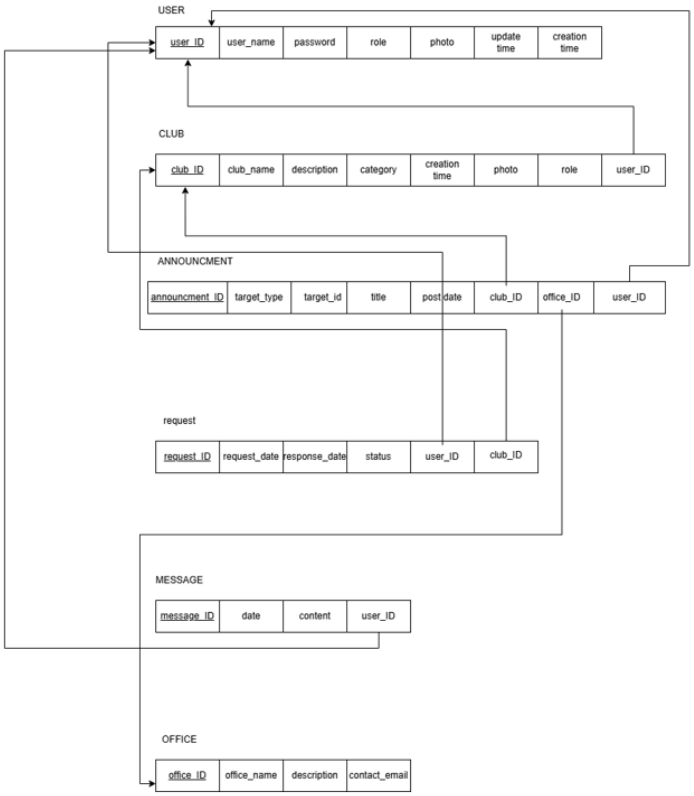
2. Relationships Between Entities

- A User can submit many Join Requests(1,m).
- Each Club can receive multiple Join Requests(1,m).
- A User can lead one or more Clubs(1,m).
- A User can publish many Announcements(1,m).
- A Club or Office can have many Announcements(1,m).
- A User can send and receive many Messages(1,m).

4.4.1 Entity Relationship Diagram “ERD”



4.4.1 Schema Diagram



4.4.2 Data Storage Model Description

User.csv

Column name	Description
User_ID	**Unique ID for each user**
name	**full name**
email	**University email (unique)**
password	**Hashed password**
role	**student / club_leader / office_admin / system_admin**
photo	**Filename of user photo**
Creation time	**Account creation timestamp**
Updated time	**Last update time**

Club.csv

Column name	Description
club_id	Unique identifier
club_name	Name of the club
club_description	Overview of the club
ctegory	Academic/social
Leader_id	User_id of club leader
Leader_email	Leader’s email
Leader photo	Leader’s photo filename
Creation time	date of club creation

Office.csv

Column name	Description
Office_id	Unique id
Office name	Name of the office
description	Role of the office
Contact email	Email for communication

Announcment.csv

Column name	Description
Announcement id	**Unique id**
title	**Announcement title**
content	**Announcement text**
Target_type	**Club/office/user/all**
Target_id	**Id of the club or user or office**

Request.csv

Column name	Description
Request_id	**Unique request id**
Request date	**The user who submitted the request**
status	**Pending/approved/rejected**
Response date	**When it was processed**

Message.csv

Column name	Description
Message_id	**Unique id**
content	**Body of the message**
date	**Date of sending**

4.3 Data Dictionary

Entity	Attribute	Data-Type	Description	Example
User	user_id	intger	Unique ID for each user.	202401560
	name	string	Full name of the user.	Nour Hossam
	email	string	University email used to log in.	nour.hossam@zewailcity.edu.eg
	Role	string	Defines whether the user is a student, leader, or admin.	Student
	password	string	Hashed password for login.	********
	photo	image	Profile picture uploaded by the user.	profile.png
club	club_id	integer	Unique identifier for each club.	12
	club_name	string	The name of the club.	IEEE Robotics
	club_description	text	Short overview of what the club does.	Focused on robotics projects and competitions.
	category	string	Type of club (academic, social, etc.).	Academic
	leader_email	string	shows the leader of the club official	shazaah@zewailcity.edu.eg

			contact (emial in our case as zewail use it as the official way to contact)	
	Leader_name	string	shows the leader of the clubs ‘s name	shaza ahmed
	leader_photo	image	Profile picture uploaded by the leader.	profile.png
	club_commitie	string	the other head members in the club (head of PR, HR,..etc)	name,email,photos
Office	office_id	intger	Unique ID for each office.	3
	office_name	string	The name of the office.	CATS Office
	description	text	Description of the office’s purpose.	Career advising and training for students.
	staff_incharge	string	The main person responsible for this office.	Dr. Maha
JoinRequest	request_id	intger	Unique ID for each join request.	456
	user_email	string	email of the student sending the request.	-@zewailcity.edu.eg
	club_id	intger	Club the user	12

			wants to join	
	status	string	Request status (Pending, Approved, Rejected).	Approved
Annocement	annocement_id	intger	unique ID for each annocement	1
	title	string	title of the annocement	Upcoming Robotics Workshop
	content	text	details about the annocement	Workshop this Friday at 3 PM
	posted_by	string	Name of the admin/leader who posted it.	205
	Date	date	Date the announcement was created	2025-11-03
Messege	messege_id	intger	Unique ID for each message.	909
	sender_id	intger	ID of the user sending the message	101
	receiver_id	intger	ID of the user receiving the message.	205
	content	text	the body or inquey	“Hi, I would like to join the – club”
	timestamp	DateTime	Time the message was	2026-11-05 18:30

			sent.	

5 Conclusion

The design phase provides a complete roadmap for building UniConnect. All system components—data models, user flows, and interactions—are clearly defined, ensuring smooth and organized development in the next stages.

The design also keeps the system flexible so it can grow and upgrade with future requirements.

5.1 Summary of Design Phase

1. Overall Architecture

UniConnect is built using the MVC (Model–View–Controller) structure:

• The Model handles data for users, clubs, offices, join requests, messages, and announcements.

• The View includes all HTML pages users interact with.

• The Controller (Flask) decides how the system responds to user actions.

This structure makes the system organized, easy to modify, and simple to maintain.

2. Data Structure & Modeling

We identified all key entities in the system and defined their attributes and the relationships between them.

Even though no database is used now, the design is compatible with SQL databases, which makes upgrading in future phases much easier.

3. User Interface & Interaction Design

The layout is kept simple, clean, and consistent.

Users can easily explore clubs, offices, announcements, and messages. Navigation is straightforward, and the system is built to work smoothly on both desktops and mobile devices.

4. Functional and Logical Design

The system includes:

• Role-based access control

• Management of club and office data

• Handling join requests and approvals

• Posting announcements

• User-to-user messaging

Each feature is designed in a modular way so updates can be made without affecting other parts of the system.

5. Storage Approach

Because the project must run without a real database, data is stored temporarily in Python's memory while the Flask server is running. The schema, however, is already prepared for a database for future expansion.