Software Requirements Specification (SRS)

UniConnect

(Better view for the document)

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

The purpose of this system is to make it easier for students and staff to find and contact the people in charge of different university clubs, offices, and committees.
In our university, there are many student clubs like (IEEE Women, Robotics,.. etc) and more are being created every semester so this system helps people to keep updated by the new clubs and whom in charge, There are also official offices like the (Student Court and CATS Office,..etc). However, it's not always easy to know who to contact when we need something so we created UniConnect.

The UniConnect Directory System solves this by collecting and showing the (names, photos, emails, and roles) of the people responsible for these entities, all in one simple and organized system. This helps students communicate directly with the right people without confusion.

1.2 Scope

This system will be a web-based platform that anyone in the university can access.
It will display all clubs and offices along with their leaders and members in charge. The admin will be able to add, edit, or remove any club or office information as needed.

The main goal is to make university communication more transparent and convenient, as this targets:

- Students who need to contact someone specific.
- Admins who manage club or office data.
- Staff who need updated contact details for coordination.

he system will keep everything up-to-date, simple, and easy to use, supporting the university's growth as more clubs and offices are created.

1.3 Definitions, Acronyms, and Abbreviations

1.4 References

- Course Project Guidelines (Introduction to Software Engineering Project, Fall 2025) (classroom)
- IEEE Standard for Software Requirements Specifications (IEEE 830–1998)
- Flask documentation
- HTML/CSS W3Schools Documentation

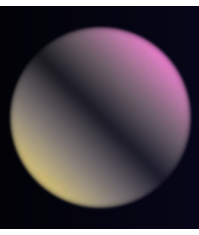
1.5 Intended Audience

The UniConnect System is mainly designed for everyone in the university community who needs quick and clear access to contact information about clubs, offices, and people in charge.

- **Students:** To easily find and contact the heads of clubs, office staff, or committees without needing to ask around or search through multiple sources.
- **Club and Office Members:** To share accurate details about their roles, make updates when needed, and help students connect with them.
- **University Staff and Administration:** To keep an organized record of all clubs and offices, manage updates, and maintain communication transparency inside the university.
- **New Students:** To explore available clubs and university services and know whom to contact for joining or asking questions.

1.6 Overview

This SRS explains what the system will do, who will use it, and how it will function. It covers both the functional and non-functional requirements, along with diagrams and models that will help in the next design and implementation phases..



2. Overall description

2.1 Product perspective

UniConnect is a web platform developed using Flask and basic HTML.
It acts as a directory for all university clubs, committees, and offices, displaying their information in an organized and user-friendly way.
The system follows the MVC (Model–View–Controller) design, where;

- The **Model** represents data such as users, clubs, members, and announcements (stored temporarily in Python lists or dictionaries).
- The **View** represents HTML pages that display this information to users.
- The **Controller** (Flask) manages user requests, updates data, and returns the appropriate web pages.

2.2 Product functions

The main functions of UniConnect include:

- Displaying all university clubs and offices with their names and short descriptions.
- Showing detailed pages for each club or office, including their hierarchy (head, vice-head, team leaders, members) with contact info.
- Allowing admin users to add, edit, or delete clubs, offices, and members.
- Allowing students to browse, search clubs and offices.
- Providing a simple contact form to send messages to club heads or office leaders.
- Displaying announcements or updates from clubs and offices.
- Managing user authentication with role-based access (student, club admin, system admin).

2.3 Design constraints

-
- The project must be developed using **Flask**.
- The interface must be implemented using **HTML** (with optional CSS).
- No external database (e.g., MySQL, SQLite) is used.
- All data resets once the server stops, as information is stored in Python memory.
- The project must follow the MVC structure.

2.4 User Characteristics

- **Students:** Regular users who can browse clubs/offices, search for information, and view contacts or announcements.
- **Club/Office Admins:** Users responsible for managing their club's information, members, and announcements.
- **System Admins:** Users with full control to add or remove clubs/offices, manage user roles, and verify information accuracy.

2.5 Operating Environment

The system runs locally on a computer with:

- **Operating System:** Windows
- **Backend Framework:** Flask
- **Frontend:** HTML and CSS
- **Browser:** Any modern web browser (Chrome, Edge, Firefox)

2.6 Assumptions and Dependencies

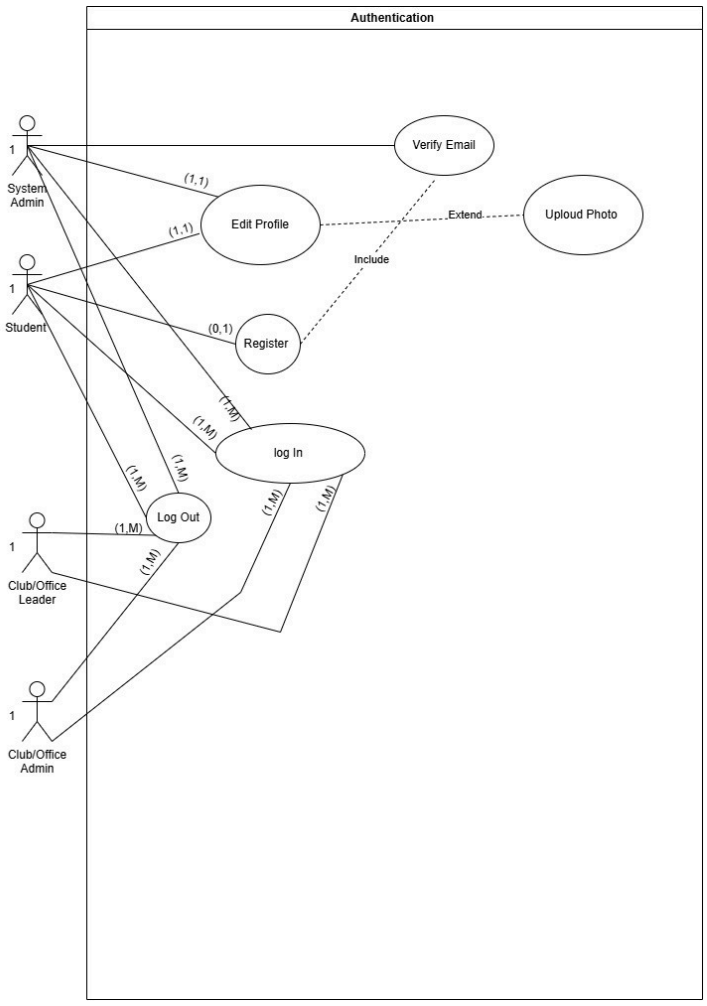
- It is assumed that all users accessing the system are verified university members.
- Data entered during runtime is not permanently stored after the Flask app is closed.
- Future versions may connect to a real database or support online deployment.

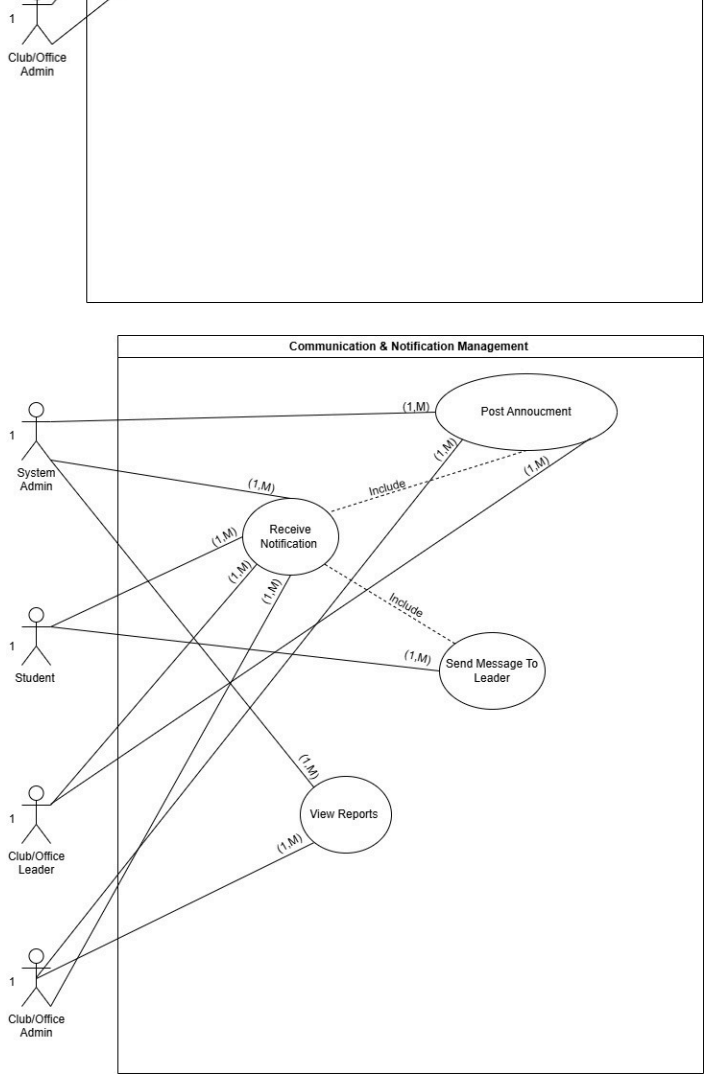
3. Specific Requirements

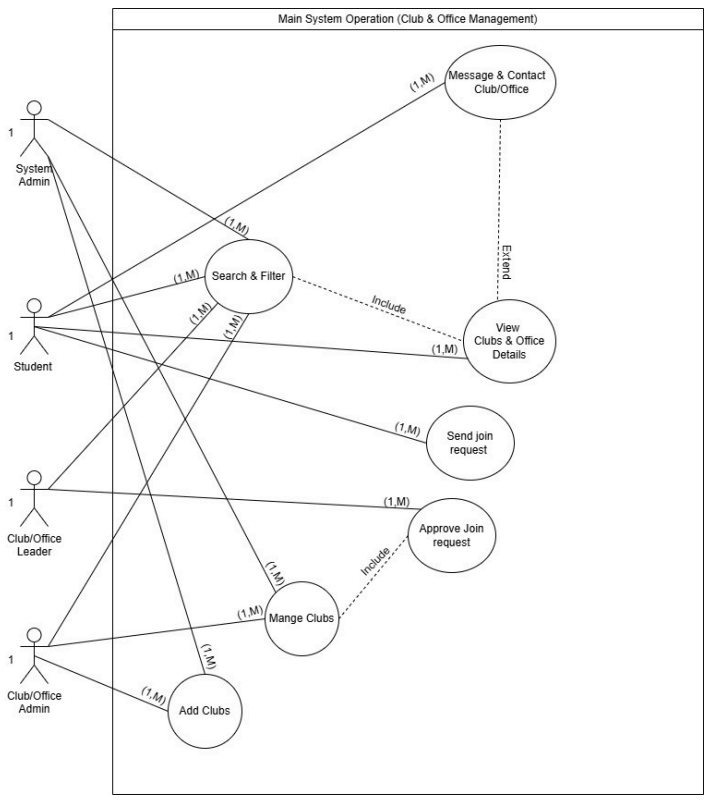
3.1 Functional Requirement

Functional Requirements	Description
User Registration	The system shall allow users (students, admins, and club leaders) to register and log in using their university email.
Search & Filter Clubs	The system allows users to search for clubs by name, category, or faculty and filter by interest.
View Club Details	The system shall show each club's description, contact info, leaders, and members.
View Office Details	The system displays details about official university offices (Student Affairs, CATS Office, etc.).
Join Club Request	Students shall be able to send a request to join a club.
Approve/Reject Requests	Club leaders shall be able to accept or reject join requests.
Messaging & Contact	The system allows users to message club leaders or send inquiries.
Post Announcements/Updates	Club admins or office admins shall be able to post announcements, upcoming events, or news.
Notification System	The system notifies users when their join requests are approved or when new updates are posted.
Profile Management	Each user shall be able to edit their personal information.
Add New Club/Office	Admin can create new entries for new clubs or university offices.
Edit or Delete Club/Office	Admin can delete specific club or office and edit information of any club or office.
Assign Hierarchical Roles	Define member positions (Head, Vice, Team Leader, Member).
Reports & Analytics	Admins shall view reports such as total clubs, member counts, and most active clubs.

3.2.3 Use Case Model & Domain Model







3.4 Non-Functional Requirements

1. Security

- · **Restricted Access**: Only verified Zewail City students and staff can register or log in. Registration is limited to users with @zewailcity.edu.eg email addresses.
- · **Password Protection**: All passwords are hashed using secure Flask libraries such as PBKDF2 or Bcrypt. Weak or common passwords are rejected during registration.
- · **Role-Based Authorization**: Each user has one of the following roles — student, club admin, office admin, or system admin — and system actions are limited accordingly.
- · **Secure Communication**: All data transfers occur over HTTPS. Cookies are set with Secure and HttpOnly flags.
- · **Input Validation**: All form inputs are validated and sanitized. Uploaded images (logos or photos) are restricted to safe formats such as PNG or JPG and a 2 MB maximum size.
- · **Login Protection**: The system limits failed login attempts to five per 10 minutes per IP address and records all login and admin actions in audit logs.

2. Performance

- The system should respond to user actions within two seconds at most.
- Searching for clubs, offices, or people should return results almost instantly.
- The system should run smoothly even when several users access it at the same time.
- Since data is stored temporarily in Python (lists or dictionaries), it should process requests quickly without delays.
- Page transitions (like moving from home to details view) should be fast and seamless.
- The system should remain stable and responsive even as more clubs or offices are added in the future.

3.Usability and Accessibility

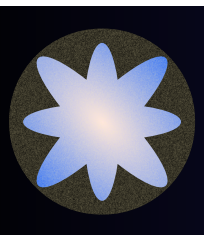
- The system should have a simple and clean interface that's easy for anyone to use without training.
- All pages should follow a consistent layout (same colors, buttons, and structure).
- Text and buttons should be clear and readable on both desktop and mobile screens.
- The system should be fully responsive, adjusting automatically to different screen sizes.
- Navigation should be straightforward, with users able to reach any page in just a few clicks.
- Basic accessibility rules should be followed (proper labels, alt text for images, and keyboard navigation).
- Error messages should be friendly and helpful, guiding users on what to do next.

4.Maintainability

- The system's code should be well-organized and easy to read, following the **MVC** structure (Model–View–Controller).
- Each feature should be modular, so updates or fixes can be made without affecting other parts.
- Variable names, functions, and files should be clearly named and documented for easy understanding.
- The code should be commented where needed to explain logic and flow.
- If a small change (like updating the interface or adding a new club) is required, it should be easy to apply without rewriting the whole system.
- The project should use GitHub for version control, allowing tracking of changes and collaboration among team members.

5.calability

- The system should be built in a way that allows new features or modules (like event management or feedback forms) to be added later.
- It should handle more data (new clubs, offices, and members) without slowing down.
- The code should be structured and extendable, so shifting to a real database (like SQL) in the future can be done easily.
- The system design should allow more users to access it at once if deployed online.
- The layout and UI should still perform well and look clean even as the amount of displayed information grows.



3.5 External Interface Requirements

3.5.1 User Interface

UniConnect will have a simple and clean web interface that's easy for anyone to use. The design will focus on clarity and consistency, so every page feels connected.

- The main pages will include Home, Clubs, Offices, and Contact Info.
- Buttons, forms, and search bars will be clearly labeled to guide users.
- The layout will stay the same across all pages for a smooth experience.
- The interface will be fully responsive, meaning it works well on both laptops and phones.
- The colors and text will be readable, simple, and professional to match the university theme.

3.5.2 Hardware Interface

UniConnect doesn't require any special hardware.

- It will run on any normal computer or laptop that can host a Flask server.
- Users can open it from any device that has a web browser — laptops, tablets, or even phones.
- No external devices or sensors are needed, just a stable internet connection.

3.5.3 Software Interface

The system will use a few main software tools to work properly:

- **Flask (Python)** will be used to handle the backend logic and routing.
- **HTML and CSS** will be used to build the frontend pages that users see.
- **Python lists and dictionaries** will be used for storing temporary data (no external database for now).
- In the future, the system can be easily connected to a real database like SQL if needed.
- The project will run locally on a Flask development server.

3.5.4 Communication Interface

UniConnect will use a simple web-based communication flow.

- Users will interact with the system through their web browser.
- Flask will manage all communication between the user and the server using **HTTP requests**.
- When a user searches or clicks a button, Flask processes it and sends back the correct page or data.
- During development, the system will run on **localhost**, but in the future, it can be hosted on the university network or a cloud server.

4. Appendices

4.1 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 contact (emial in our case as zewail use it as the official way to contact)	shazaah@zewailcity.edu.eg
	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	-@zewailcity.edu.eg

			sending the request.	
	club_id	intger	Club the user wants to join.	12
	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 sent.	2026-11-05 18:30

4.2 Glossary

Term	Definition
Admin	A user responsible for managing the entire system, including adding or editing clubs, offices, and users.
Student	A regular user who can view clubs/offices, send join requests, and message leaders.
Club Leader	The head or main representative of a student club who manages members and handles join requests.
Club Admin	Assists the club leader in managing announcements and member requests.
Office	An official university department (like the CATS Office or Student Court).
Announcement	A post shared by a club or office to update students about events or news.
Join Request	A request a student sends to join a specific club.
Flask	A Python web framework used to build the system's backend and manage routing.
MVC	Model–View–Controller — the design pattern followed by the system.
Model	Represents the data and logic (like users, clubs, and offices).
View	The web pages that display data to the user.
Controller	The Flask layer that handles user input and system responses.
User Interface (UI)	The visual part of the system that users interact with.
Localhost	The local development environment where Flask runs during testing.