Software Design Document (SDD)

**The Binary Bros – “ChatFAU”**   
Jordan Small

Nico Arcelin

Alirio Da Rocha 

**October 27, 2024**

    
CEN4010 – Principles of Software Engineering

Fall 2024

Professor Safak Kayikci     
Florida Atlantic University

**Document Revisions**

|  |  |
| --- | --- |
| **Version** | **Description** |
| 1.00 | Initial commit. The document has been filled with project outline and formatted with MS Word headings for Table of Contents. |
| 1.10 | The document has been filled with all necessary information, proofread, and carefully formatted. Terms deemed arbitrarily important were added to the Glossary and references used were cited. Table of Contents updated. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table of Contents

[1. Introduction 4](#_Toc180930102)

[1.1. Purpose 4](#_Toc180930103)

[1.2. Scope 4](#_Toc180930104)

[1.3. Glossary 4](#_Toc180930105)

[1.4. References 5](#_Toc180930106)

[1.5 Overview 5](#_Toc180930107)

[2. System Architecture 6](#_Toc180930108)

[2.1 Architectural Design: 6](#_Toc180930109)

[2.2 Architectural Patterns/Styles: 7](#_Toc180930110)

[3. Detailed Design 7](#_Toc180930111)

[3.1 Component Design 7](#_Toc180930112)

[3.2 Interface Design 12](#_Toc180930113)

[4. Data Design 14](#_Toc180930114)

[4.1 Data Structures 14](#_Toc180930115)

[4.2 Database Schemas 16](#_Toc180930116)

[4.3 Data Flow Between Components 17](#_Toc180930117)

[5. Design Constraints 18](#_Toc180930118)

[5.1 Hardware Limitations 18](#_Toc180930119)

[5.2 Scalability 18](#_Toc180930120)

[5.3 Technology Constraints 18](#_Toc180930121)

[6. Security and Safety Considerations 18](#_Toc180930122)

[6.1 Security Measures 18](#_Toc180930123)

[7. Design Decisions 19](#_Toc180930124)

[7.1 Functional Decisions 19](#_Toc180930125)

[7.2 Non-functional Decisions 20](#_Toc180930126)

# Introduction

## Purpose

The overall purpose of this Software Design Document (SDD) is to provide a thorough summary of ChatFAU’s system design and architecture. The document will outline key design components, constraints, and considerations necessary for the development of a chatting platform to facilitate educational communication. Other important design aspects—such as security measures, risk mitigation, and change management—will also be addressed.

The main purpose of the SDD is to provide a blueprint for developers to follow during the implementation phase. It will act as a reference point for testing, ensuring that the system behaves exactly according to the design specifications.

This SDD will serve as a guide for project stakeholders, developers, and users throughout the software development lifecycle, providing clarity regarding the system design. The intended audience includes software developers, system architects, university administrators, and project managers; all of whom will require detailed design specifications to implement the system smoothly.

## Scope

ChatFAU will be designed as a web-based application to facilitate academic discussions among students and educators at Florida Atlantic University (FAU). This SDD will define the software design's boundaries, focusing on key components and specific modules planned for development. In doing so, this document will outline aspects and considerations of the software’s design and architecture that are necessary in developing a user-friendly environment to incorporate these modules.

Several key modules for this application that will be addressed in this document include:

* User account management, which will allow users to create and customize their profiles
* Real-time messaging functionality, enabling live chat with markup options, as well as private messaging
* Community and study group management, which will support academic discussions between educators and students
* Moderation tools and permissions, designed for designated users to ensure a safe, clean, and positive environment for users

Non-functional components in this document will address security, data storage, performance, and usability standards to ensure an intuitive experience across any device. ChatFAU will launch as a platform exclusive to Florida Atlantic University but plans for expansion to incorporate other educational institutions. Moreover, this SDD will outline all components and modules necessary for a successful implementation of ChatFAU.

## Glossary

* **API –** Application Programming Interface: a set of rules that allows software applications to communicate with each other
* **Authentication –** the process of verifying the identity of a user, device, or process before they are given access to a system, account, or file
* **Backend –** the part of a computer program or application that stores and processes data, and that allows the program to function
* **ChatFAU –** Chat Florida Atlantic University: our platform can be generalized, but will be launched with a special designation for FAU
* **Encryption –** the process of protecting information or data by using mathematical models to scramble it in such a way that only the parties who have the key to unscramble it can access it
* **Frontend –** the graphical user interface (GUI) that your users can directly interact with
* **Hashing –** assign a numeric or alphanumeric string to a piece of data by applying a function whose output values are all the same number of bits in length
* **Module –** a self-contained unit of code that is used to perform specific tasks
* **RLS –** Row Level Security: lets you filter data and enables access to specific rows in a table based on qualifying user conditions
* **SDD –** Software Development Document: a detailed plan for developing software that outlines its functionality, architecture, and technical details
* **SSO –** Single Sign-on:  an authentication scheme that allows a user to log in with a single ID to any of several related, yet independent, software systems
* **UI –** User Interface: how the user and a computer system interact, the use of input devices and software
* **WebSockets -** a computer communications protocol, providing a simultaneous two-way communication channel over a single Transmission Control Protocol connection

## References

Jwmsft. (2024). *Screen Sizes and break points for Responsive Design - Windows Apps*. Screen sizes and

break points for responsive design - Windows apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/apps/design/layout/screen-sizes-and-breakpoints-for-responsive-design

KianNH. (2024). *How Cloudflare works | Cloudflare Learning Paths*. Cloudflare Docs.

https://developers.cloudflare.com/learning-paths/get-started/concepts/how-cloudflare-works/

*Supabase Docs*. (2024, September 17). Supabase Docs. https://supabase.com/docs

## 1.5 Overview

ChatFAU will be designed with modular architecture to support academic discussions among students and educators at Florida Atlantic University. This section of the document will provide a high-level description of the software’s architecture and key components. It will describe a general approach to the system’s design, offering a summary of the main modules and their interactions.

As far as system architecture, ChatFAU will utilize a cloud-based database design, leveraging Supabase for efficient and reliable data management. Our database will store users, profile attributes, message histories, moderation data, groups/communities, and private messages to ensure that all interactions are recorded and easily accessible. The integration of Row Level Security (RLS) will enhance data protection by restricting access based on user roles.

At the core of the system is the User Management Module, which will be responsible for account creation, authentication, management, and profile customization. The module will support Single Sign-On (SSO) to streamline the authentication process and maintain security. Additionally, the module will ensure that users can register using their school email, enabling secure access to the platform. Passwords will be securely managed through hashing techniques to remain protected.

The Messaging Module will be responsible for all platform messages: both real-time and asynchronous. This module will support live chatting and private messaging, allowing users to engage in conversations and utilize standard markup options while doing so.

The Community and Study Group Module allows users to create and join academic communities or topics based on relevant courses, interests, or studies. This will facilitate organized and collaborative environments for discussion.

The Moderation Module keeps some of the previous ones in check, empowering designated users to manage interactions between communities. This ensures a respectful and productive environment.

The Notification Module keeps users informed regarding arbitrarily deemed important updates. Users will be offered customizable settings for different notification preferences.

These modules will interact seamlessly within the system, providing for an intuitive user experience at every step. Altogether it is a robust platform, adaptable for expansion, that will empower any academic institution to engage in relevant communication. The application will offer flexibility for future enhancements, scalability, and development to meet the needs of students and educators everywhere.

# System Architecture

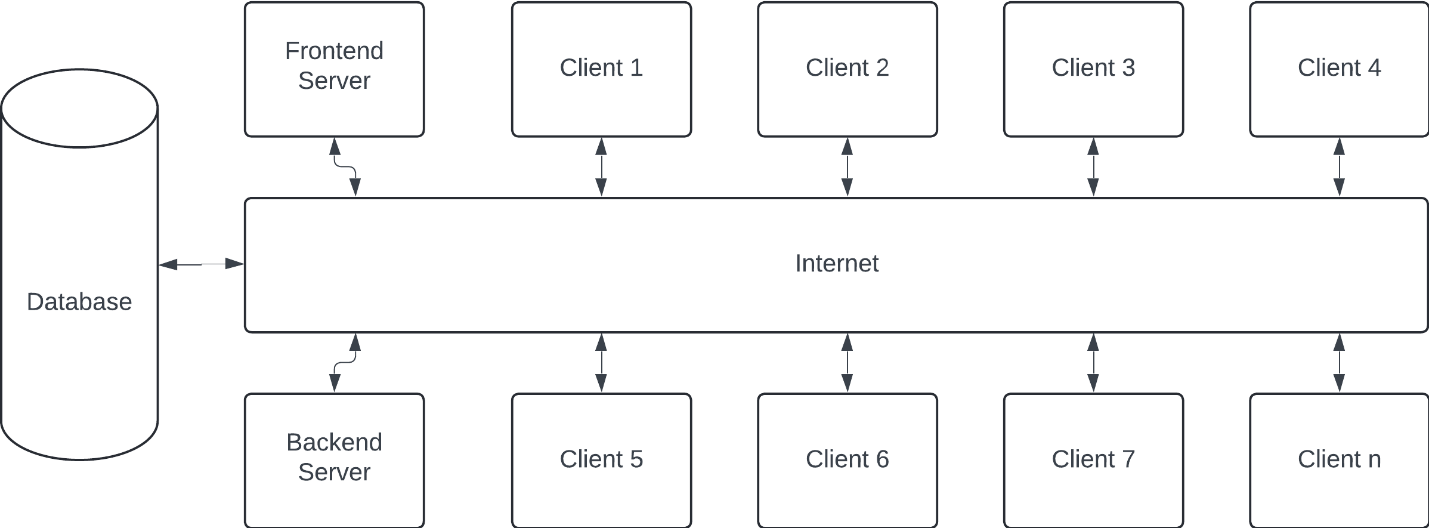
## Architectural Design:

Key Components

1. User Management Module
   * User Model: Defines teachers, students, and admin roles with respective properties and permissions
   * Authentication Service: Manages login, registration, and authentication (OAuth and/or other protocols).
2. Messaging Module
   * Message Model: Represents messages, including content, timestamp, sender, and recipient.
   * Char Service: Handles sending, receiving, and storing messages. Manages messages queues and may support real-time updates.
   * Notification Service: Manages in-app notification for new messages, group updates, or announcements.
3. Group Management Module
   * Study Group Model: Defines study groups, including group members, group-specific chat history, and any related materials or announcements.
   * Group Service: Manages group creation, membership requests, and permissions.
4. Database Layer
   * Main Approach: To have user, chat, and group data stored in one database under different tables. All will have foreign keys which will link the necessary tables.
   * Secondary Approach: Separate into three databases; User, Chat, and Group. This will create some separation, and it may be harder to link together for high level queries.
5. API Layer
   * REST API or GraphQL: Provides endpoints for the front-end to access user data, messaging, and group functionalities.
   * WebSocket Server: Enables real-time messaging capabilities.
6. Frontend (Client)
   * UI Components: Chat windows, group management interfaces, notifications, etc.
   * Controller/State Management: Manages data flow between components and updates from backend.

## Architectural Patterns/Styles:

Client– Server architecture:



# Detailed Design

## 3.1 Component Design

1. User Management Module

* Purpose: Manages user-related information such as registration, login, roles, permission, and profile settings
* Input: User details (name, email, role, password, profile updates, etc.)
* Output: Success or error response, user profile data, role-based permissions, and access tokens upon successful login
* Data Structures

User Model:

A computer screen shot of white text

Description automatically generated

Session Data:

A screenshot of a computer screen

Description automatically generated

* Algorithms
  + Password Hashing: Securely hash user passwords during registration.
  + Token Generation: Generate and assign JWTs or session tokens upon successful login.

1. Group Management Module

* Purpose: Manages study groups, including creation, membership, permissions, and group-specific settings.
* Input: Group creation data (name, description, initial members), user requests to join or leave groups.
* Output: Group data, membership list, success, or error response.
* Data Structures

Study Group Model:

A screen shot of a computer code

Description automatically generated

Membership Data:

A screen shot of a computer code

Description automatically generated

* Algorithms
  + Group Creation: Create a new group with initial data and store it in the database.
  + Member Management: Add, remove, or update group members and their roles based on permissions
  + Check if a user is eligible to join or leave the group and update the membership data accordingly

1. Messaging Module

* Purpose: Manages direct messages between users and group chats, including message creation, storage, retrieval, and notifications for new messages.
* Input: Message content, sender ID, recipient or group ID, timestamp.
* Output: Message data, status of message delivery, notification of new messages.
* Data Structures

Message Model:

A screen shot of a computer program

Description automatically generated

Membership Data:

A black screen with white text

Description automatically generated

* Algorithms
  + Send Message: Validate message content, assign timestamp, and store in the database.
  + Retrieve Messages: Fetch messages for a specific user or group, ordered by timestamp.
  + Push Notifications: Upon message creation, trigger a notification to the intended recipient(s).
  + Status Update: Update message status (delivered or read) based on recipients’ actions.

1. Database Layer

* Purpose: Stores all persistent data for ChatFAU, including user information, messages, groups, and other related metadata.
* Input: Data to be stored or updated, such as user profiles, char history, and group membership information.
* Output: Data retrieval results, such as user profiles, chat history, and group membership information.
* Data Structures

User Table:

A computer screen shot of a program

Description automatically generated

Message Table:

A screen shot of a computer program

Description automatically generated

Study Group Table:

A screen shot of a computer code

Description automatically generated

* Algorithms
  + Data Retrieval Algorithms: SQL queries to retrieve users, groups, and message history efficiently.
  + Data Integrity Checks: Ensure data consistency and validation, such as maintaining unique user emails or valid group memberships.

1. API Layer

* Purpose: Serves as the intermediary between the client and the backend. It processes requests from the frontend, interacts with the database, and returns data or success/error responses
* Input: HTTP requests (GET, POST, PUT, DELETE) from the client containing data like user login credentials, message content, or group creation requests.
* Output: JSON responses containing requested data, success or error messages, and HTTP status codes.
* Data Structures
  + RESTful Endpoints:
    - GET */api/users:* Retrieve user information.
    - POST /*api/messages:* Send a new message.
    - GET */api/groups:* Retrieve list of study groups.
    - POST */api/auth/login:* User login, returns an access token.
    - WebSocket: Real-time updates for messages and notifications
* Algorithms
  + Authentication and Authorization: Check if a request is from an authenticated and authorized user.
  + Rate Limiting: Limit API requests per user to prevent abuse.
  + Data Transformation: Format and filter data based on user roles and permissions.
  + Error Handling: Handle errors like unauthorized access, missing data, and internal server errors.

1. Frontend (Client)

* Purpose: Provides the user interface for ChatFAU, allowing users to interact with the system, including sending messages, viewing groups, and managing profiles.
* Input: User interactions (clicks, forms submissions) and data inputs like message content, login details, and group search queries.
* Output: Display of retrieved data, feedback messages, notifications, and error messages.
* Data Structures
  + State Management: Holds app-wide state, such as the current user session, active conversations, and group memberships.
  + Chat Window: Displays messages and allows users to send new messages.
  + Group View: Lists groups the user is part of and allows them to join new groups.
  + User Profile: Shows user information and provides option to update it.
* Algorithms
  + Two-Way Binding: Sync user inputs with component state for real-time updates.
  + Fetch and Render: Retrieve data via API requests and render it in the UI.
  + WebSocket Connection: Maintain a real-time connection to display new messages immediately.
  + Routing: Manage navigation between chat, groups, and profile screens.
  + State Updates: Efficiently update the UI based on user interactions and API responses

## 3.2 Interface Design

1. User Management Module

* Interacts with the Database Layer to store and retrieve user information, handles authentication requests via the API Layer, and communicates with the Frontend to manage user session and permissions.
* API Descriptions:
  + POST /api/auth/register: Registers a new user with details (name, email, password, role, etc.)
  + POST /api/auth/login: Authenticates the user and returns an access token.
  + GET /api/users/: Retrieves a user profile based on user ID
* Communication Protocols: Primarily uses HTTPS for secure communication. Sensitive data like passwords are hashed and transmitted securely with JWT-based token authentication.

1. Messaging Module

* Interacts with the Database Layer to store and retrieve messages. It also uses the API Layer and WebSocket server for real-time message updates to the Frontend.
* API Descriptions:
  + POST /api/messages: Sends a new message from one user to another or to a group.
  + GET /api/messages/: Retrieves all messages within a specific conversation.
  + WebSocket /we/messages: Real-time messaging channel for new messages, typing indicators, and message read status.
* Communication Protocols: Uses HTTPS for standard API communication and WebSocket for real-time messaging, supporting bi-directional data flow for interactive chat features.

1. Study Group Module

* Interacts with the Database Layer to manage group data, including creating groups, adding/removing members, and managing group roles. It communicates through the API Layer and provides data to the Frontend for group-related operations.
* API Descriptions:
  + POST /api/groups: Creates a new study group with specified members.
  + GET /api/groups: Retrieves a list of all groups.
  + PUT /api/groups/members: Adds or removes members in a group.
* Communication Protocols: Uses HTTPS with RESTful endpoints to ensure secure and reliable data transmission between the client and the server.

1. Database Layer

* Interacts with all other modules. It handles requests for data storage, retrieval, and updates from the User Management, Messaging, and Study Group modules via the API Layer.
* No direct APIs exposed. Acts as a backend service layer accessed through queries from other modules.
* Communication Protocols: Typically communicates with the API Layer through SQL queries or ORM calls within the server, ensuring data consistency and security.

1. API Layer

* Interacts with the Database and Frontend. Aces as an intermediary between the Database Layer and the Frontend, User Management, Messaging, and Study Group modules. It routes requests, authenticates users, and enforces permissions.
* API Descriptions:
  + Exposes the RESTful APIs to external clients, allowing data exchange between the server and client.
* Communication Protocols: Uses HTTPS to ensure encrypted communication. It also supports WebSocket for real-time interactions, especially for messaging.

1. Frontend (Client)

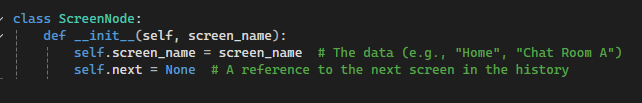
* Interacts with the API Layer to retrieve data for users, messages, and groups. The front end receives real-time updates through the WebSocket server.
* Frontends interact with various API endpoints shown in other layers to perform actions like login, send messages, view group data etc.
* Communication Protocols: Uses HTTPS for secure API calls and WebSocket for real-time messaging, ensuring responsive and interactive user experiences.

# 4. Data Design

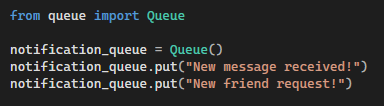
## Data Structures

The data structures used in the ChatFAU application will be various. The most appropriate for various aspects of the system are:

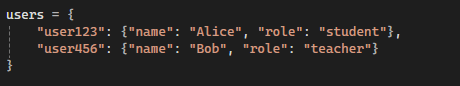
* Arrays: Strong lists of users, messages or notifications that need to be processed or displayed sequentially. As an example, it will be retrieving messages from a chat, they can be stored in an array to display the correct order.
  + Example in python: 
* Linked lists: Managing a lot of dynamic data such as chat logs that messages are continuously added in real time, with this it allows for efficient insertions and deletions without shifting elements compared to with an array. Linked lists could be used to implement a navigation history like clicking the home screen, going to your profile, and clicking the back button to go back to the home screen.
  + Example in python:



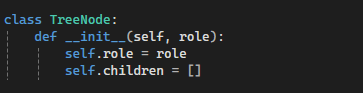
* Queues: This will handle message queues, notification delivery or task scheduling. Implementing queues for incoming messages will ensure they are processed in the order they arrive, first in, first out.
  + Example in python:



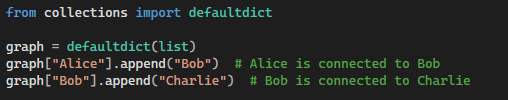
* Hash tables: Will efficiently map user-related information, such as UserID to user details, storing configurations and quick lookups for user roles or permission, as well as mapping notifications or badges to their respective places.
  + Example in python:



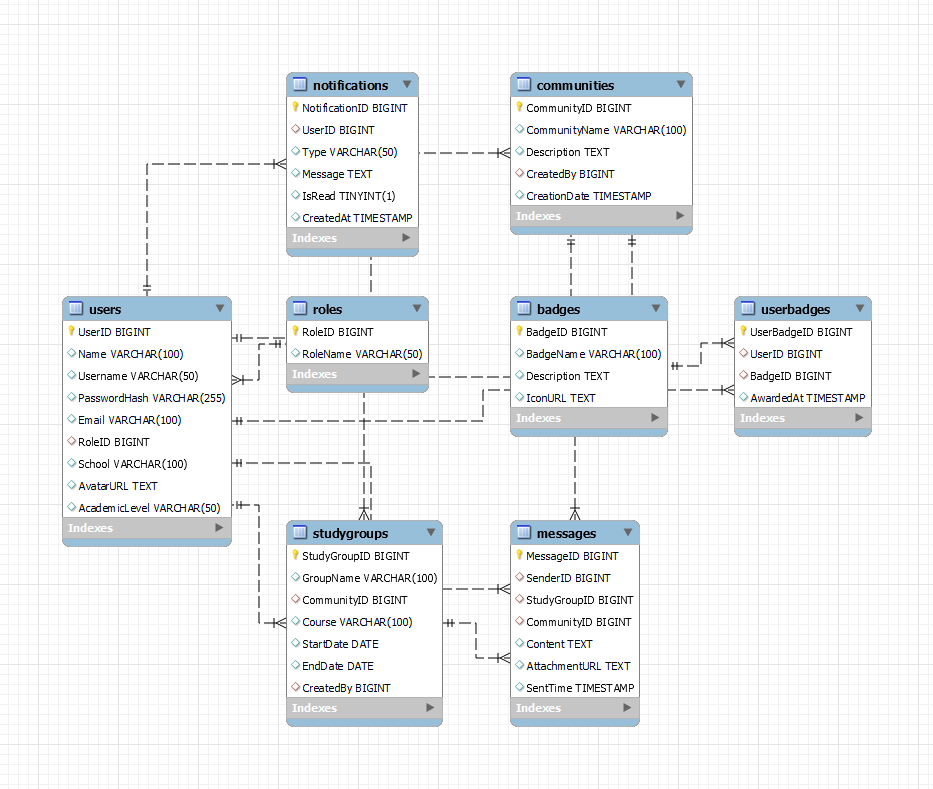
* Trees: Binary search trees are used to organize hierarchical data, such as user roles and permissions, as an example, Admin > Teacher > Student. Could also be useful to manage threaded conversations where messages are replies to other messages.
  + Example in python:



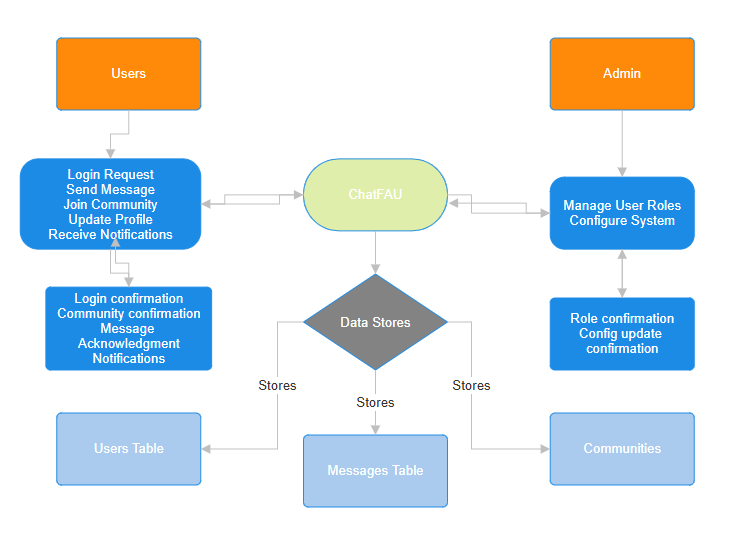
* Graphs: Used to represent relationships between users, can be friendships, group membership, or study group collaborations. Each node in it can represent a user, and edges can represent interactions or connections. It is also used to model connections between communities and study groups.
  + Example in python:

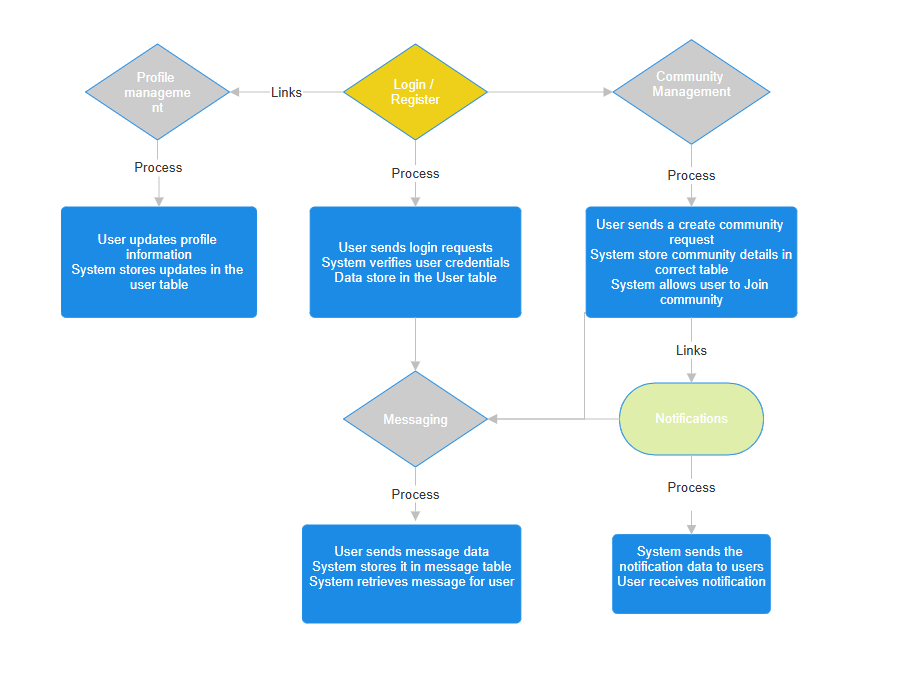


## Database Schemas



## 4.3 Data Flow Between Components





# 5. Design Constraints

## Hardware Limitations

Server specifications: The system relies on cloud-based servers for its hosting which must have sufficient CPU power, memory, and storage to handle real time communication, data storage and user interactions. If this is running on a local server setup, the hardware must be able to support multiple concurrent users, requiring servers with higher performance.

Client devices: The platform is put in a way to run on web browsers across multiple devices. Therefore, it must accommodate low powered devices, making sure that the web application remains responsive to devices with limited memory and processing power.

## Scalability

This refers to the system's ability to handle increased loads, such as additional users, more messages, and growing numbers of communities without losing performance. First, we will implement load balancing, the system will employ this to distribute requests evenly across available servers, preventing a single server from becoming a bottle neck to the application. This helps ensure that real-time chat remains fast and reliable even during peak usage.

## 5.3 Technology Constraints

Programming language: The backend of the application will be developed using JavaScript Node.js or Python, allowing for faster development, easier maintenance, and integration with existing technologies.

Database: Supabase built on PostgreSQL is the database management service used for ChatFAU, providing strong security features and scalability. This system will use Row Level Security to manage data access based on user roles.

Cloud platform: The app is hosted on GCP, Google Cloud Platform, that provides reliable scalability, infrastructure, and security services.

Web tech: the platform developed as a web application that will be compatible with major browsers such as Chrome, Firefox, Safari, Edge. WebSocket's will be used for real-time communication and making sure that users can send messages without delay.

Encryption Protocols: All data will be encrypted using TLS/SSL and encryption for sensitive data at rest will be performed using AES-256.

# 6. Security and Safety Considerations

## Security Measures

Data encryption: All sensitive data is encrypted while stored in the database. This makes sure that even if the database becomes compromised the data will remain secure. Algorithms such as AES-256 are used for encrypting sensitive information, providing more robust security.

Passwords: Passwords need a standard of at least 8 characters with numbers and letters with a special character and an option for the user to add multi-factor authentication.

Encryption in Transit (HTTPS): All communication between the user’s device and the server is encrypted using HTTPS (TLS/SSL). This will prevent the attackers from intercepting data during any transmission, making sure that all interactions are private and secure. Certificates from trusted Certificate Authorities are used to establish secure connections.

Data masking and hashing: Sensitive data such as the passwords are hashed using strong cryptographic hashing algorithms before storage, this will ensure that even if the data is breached and accessed it will not be easily decrypted. Personal data can be masked in various scenarios to prevent unauthorized viewing, for example showing on a part of an email address.

Database security: The access to the database is restricted to authorized services and applications. Database firewall rules and access control lists limit who can connect to the database server, helping to reduce the risk of unauthorized access. Regular backups are performed and encrypted snapshots will be stored securely to ensure data recovery in any case of system failures.

Monitoring and Auditing: The system also logs important actions such as login attempts, for auditing and monitoring. This allows the detection of any unusual behavior or in case of unauthorized access attempts, will be for example how there are so many unauthorized attempts to any Microsoft accounts. Automated monitoring tools, artificial intelligence, tools can be used to detect potential threats and alert admins for further investigation.

Penetration testing: Performed to ensure that the platform can withstand real world attack scenarios.

User awareness: Users will always be advised not to click on unknown links that could be used for phishing and telling them to make secure and strong passwords. The system will prompt users to update their passwords regularly to maintain account security

# 7. Design Decisions

The design of ChatFAU involved several key decisions that shaped its functionality, architecture, and user experience. These additional design decisions make the application’s focus on user engagement, security, and experience truly clear. While doing so, there are both immediate and inherent trade-offs that come with careful consideration when implementing these features.

Listed below are the most impactful design decisions that were made during the development process, along with their respective rationales and the trade-offs that were considered in terms of security and performance.

## 7.1 Functional Decisions

Profile Customization and Study Groups: the decision for user customization, including the creation or joining of study groups was made to enhance engagement and personalization. Users will be able to highlight achievements and interests, encouraging collaboration and seamless networking. The trade-off here is that more complexities are introduced for data management, which can potentially impact performance and security.

Moderation Tools Development: the consideration of moderation with specific permissions and tools was a strategic decision made to empower designated users to maintain positive communication. Commands for pruning chats, muting, or banning users will address potential issues regarding disruptive or offensive behavior. When making this decision, moderator training needed to be considered as well. Designated users will need to be carefully selected and trained, as misuse of these permissions could lead to conflicts or negative user experiences.

User-Centric Design Principles: the user interface (UI) was designed with a focus on simplicity and ease, using established design patterns from other popular chat applications. This choice was made to improve user experience, increasing familiarity when using the application. There is a risk that oversimplification might hinder additions of specialty features that users may seek, calling for a careful balance.

## 7.2 Non-functional Decisions

Single-Sign-On (SSO) integration: using SSO enhances security protocols and simplifies authentication for users, allowing them to login using their school credentials. This also helps improve user experience, as it reduces the number of passwords users must remember. The trade-off here involves reliance on the external SSO provider; if their service experiences downtime, user access might be impacted.

Real-Time Messaging via WebSocket's: using WebSockets for real-time messaging ensures low-latency communication, which is the most crucial aspect of a chat application. This approach will provide for a smooth user experience, allowing users to interact in real-time without noticeable delay. This was weighed against potential complexities in maintaining persistent connections, which could lead to security vulnerabilities and increased server load if not effectively managed.

Use of Supabase as the Database Solution: Supabase was chosen for its robust features and capabilities as a cloud-based backend service. It offers automatic API generation, real-time capabilities, and offers built-in authentication. This will provide for efficient data management and support rapid system development. Reliance on a third-party service can introduce concerns regarding costs, especially when usage begins to scale significantly.

Modular Architecture: implementing modular architecture will make the system easier to develop, test, and maintain by separating individual components of the application. Existing modules can be modified, or new ones can be added without disturbing the entire system, which in turn will support scalability. Modular architecture can, however, lead to overhead in internal communications, which could potentially impact performance if not effectively managed.