System Design Document for UniVerse

Sprint: 2 (version 2)

Course: EECS 3311

Submission Date: Tuesday December 3rd, 2024

Sprint 1 Overview

The UniVerse application is a social networking platform designed to provide users the ability to connect with others through chat functionalities, friend management, and user profile handling. In later sprints we will work on implementing a friend matching algorithm, and also group pages and activities. This application is built in Java, the application leverages the **MVC(***Model-View-Controller)* architecture to ensure modularity, scalability, and maintainability. The backend is powered by our Firebase Firestore, which enables real-time data synchronization for messaging and contact management. The frontend is implemented using Java Swing, which offers us a sturdy and intuitive graphical user interface.

This document outlines the system design of the UniVerse application, covering;

- CRC Cards for the key classes.
- A detailed MVC_based Software Architecture Diagram.
- Database Design and Integration.
- System interaction with the operating environment, external dependencies, and network requirements.
- A decomposition of the system into manageable modules.
- Strategies for handling errors and exceptions.

2 CRC Cards

CRC (Class-Responsibility-Collaborator) cards are used to describe the responsibilities of each class and the classes they collaborate with. Below is a detailed breakdown for all key classes in the UniVerse application.

Class Name:	UniVerse
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Initialize Firebase using FirebaseInitializer ● Launch the SignUporIn GUI for usr login or signup	Collaborators: ● FirebaseInitializer (to initialize Firebase services). ● SignUporIn (to display the initial GUI). ● SessionManager (to manage session variables).

Class Name:	FirestoreHandler
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Perform CRUD operations on Firestore (Create, Read, Update and Delete). ● Manage contacts and chat history. ● Authenticate user login (compare email and hashed password). ● Add, update, and delete friends and contacts. ● Store and retrieve real-time chat messages	Collaborators: ● UserProfile (for representing user data). ● SessionManager (to get the current user's ID). ● SignUporIn (to store user data during signup). ● Messaging (to retrieve and display chat messages). ● Homepage (to fetch and display user contacts).

Class Name:	SignUporIn
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Handle user signup and login. ● Collect user credentials and validate them. ● Transition between signup, login, and profile update screens.	Collaborators: ● FirestoreHandler (to authenticate and save user data). ● SessionManager (to store the logged-in user's session details). ● Homepage (to navigate to the homepage after login). ● UniVerse (to initialize and launch the GUI).

Class Name:	Messaging
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Display real-time chat messages between users. ● Send and receive messages through Firestore. ● Handle chat history for a selected contact.	Collaborators: ● FirestoreHandler (to retrieve chat history and send messages). ● SessionManager (to access the current user's session data). ● UserProfile (to display information about

	the current chat contact).

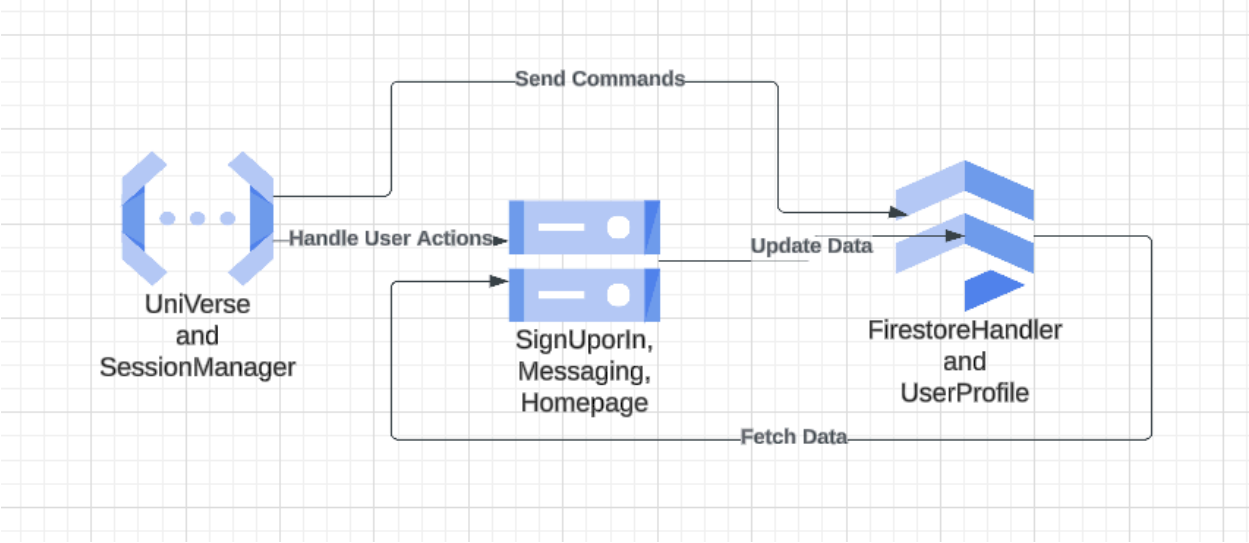
Class Name:	Homepage
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Display the logged-in user's contacts and friends list. ● Allow adding and removing friends. ● Navigate to messaging or user profile views.	Collaborators: ● FirestoreHandler (to fetch and update contact lists). ● SessionManager (to access the logged-in user's session details). ● Messaging (to navigate to the chat view for a selected contact). ● SignUporIn (to handle logout and return to login screen).

Class Name:	SessionManager
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Store global session variables, including the current user's ID and username. ● Provide access to session details across the application	Collaborators: ● All other classes (UniVerse, FirestoreHandler, SignUporIn, Messaging, Homepage) rely on SessionManager to access session data.

Class Name:	UserProfile
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Represent user data such as userId, email, bio, interests, etc. ● Serve as a data model for storing and retrieving user information.	Collaborators: ● FirestoreHandler (to convert between UserProfile objects and Firestore documents). ● Homepage (to display user details in the contacts list). ● Messaging (to display the current contact's profile information).

3 Software Architecture Diagram

The UniVerse application follows the MVC (Model-View-Controller) architecture.



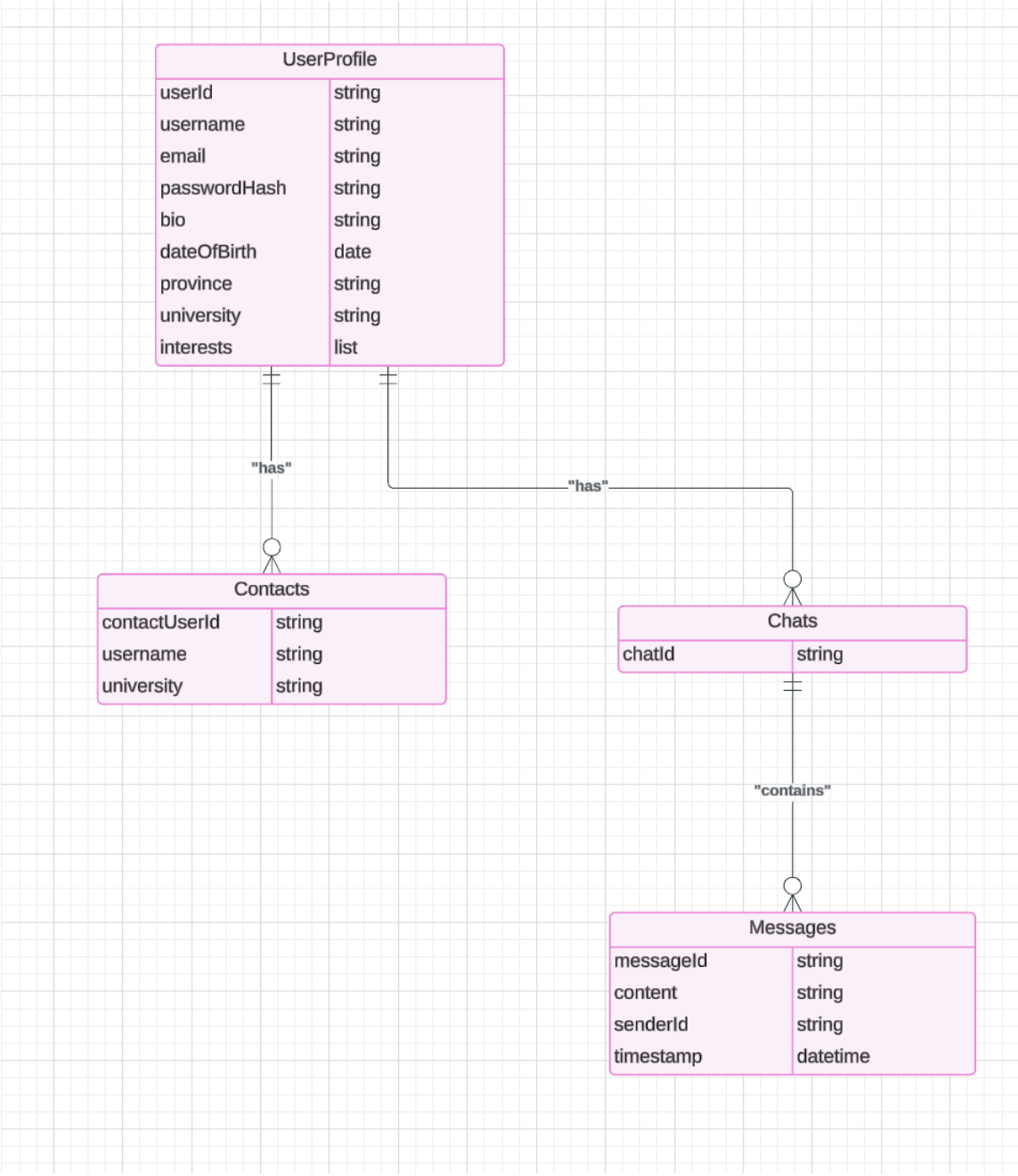
4 Database Design and Integration

4.1 Conceptual Database Overview

The UniVerse application uses Firebase Firestore as its primary database, making use of a NoSQL structure for flexibility and scalability. Below is an enhanced view of the database design.

4.2 Entity-Relationship Diagram (ERD)

The following diagram shows the relationships between the primary collections (UserProfile, contacts, chats) and their subcollections.



4.3 Collection and Field Details

Below is a detailed schema for each collection and its subcollections.

Collection Name	**Field**	**Data Type**	**Description**
UserProfile	userId	String	Unique identifier for the user
	username	String	The display name of the user
	email	String	Email address for login
	passwordHash	String	Hashed password for authentication
	bio	String	Short description of the user
	dateOfBirth	String	User's birth date in dd/mm/yyyy format
	province	String	The province or state where the user resides
	university	String	Users university
	interests	List<String>	List of user-selected interests
----------	----------	----------	----------
Contacts	**contactUserId**	**String**	**Unique ID of the connect users**
(subcollection)	username	String	The contacts display name
	university	String	The contacts affiliated university
----------	----------	----------	----------
Chats	chatId	String	Unique identifier for each chat
(collection)	messages	Subcollection	Subcollection containing chat messages.
----------	----------	----------	----------
Message	**messageId**	**String**	**Unique ID for each message**
(subcollection)	content	String	The text of the chat message

	senderId	String	The ID of the message sender
	timestamp	Timestamp	Time when the message was sent

4.4 Database Workflow

Workflow 1: User Signup

1. **Action:** The user enters details on the SignUporIn screen.
2. **FirestoreHandler:** The addUserData method creates a document in the UserProfile collection.
3. **Result:** The user's profile is saved to Firestore.

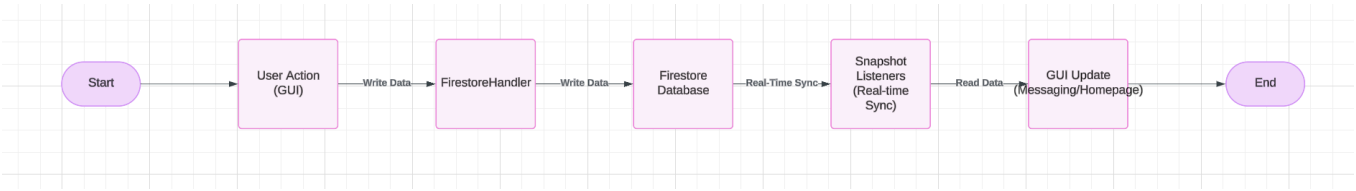
Workflow 2: Adding a Contact

1. **Action:** The user clicks "Add Friend" on the Homepage.
2. **FirestoreHandler:** The addFriend method adds a document to the contacts subcollection under the logged-in user's profile.
3. **Result:** The contact is added and displayed in the UI.

Workflow 3: Sending a Message

1. **Action:** The user types and sends a message in Messaging.
2. **FirestoreHandler:**
    - The saveMessages method creates a document in the messages subcollection under the appropriate chat ID.
    - The message is mirrored in the reverse chat document (e.g., userId_contactId and contactId_userId).
3. **Result:** The chat UI updates in real-time using Firestore snapshot listeners.

4.5 Data Flow in UniVerse



How Firestore Enhances the Application

1. **Real-Time Updates:** Firestore's snapshot listeners ensure the UI updates instantly whenever changes occur.
2. **Scalability:** The NoSQL structure allows for flexible data organization and horizontal scaling.
3. **Ease of Use:** Firestore's schema-less design simplifies database interactions, making it ideal for evolving applications.

5 System Interaction with the Environment

5.1 Operating Environment

- **Operating Systems:**
  - **macOS:** Fully supported on Apple machines with M1, M2, M3, M4 and Intel architectures.
  - **Windows:** Compatibility with Windows 10 and later, utilizing Java Runtime Environment (JRE).
  - **Linux:** Runs on distributions such as Ubuntu and Fedora, with proper Java setup.
- **Hardware Requirements:**
  - Minimum: 4 GB RAM, Intel i3, and SSD storage.
  - Recommended: 8 GB RAM, Intel i5/i7, and 256 GB SSD for faster performance during Firestore transactions.
- **Database:**
  - Firebase Firestore, a NoSQL document-based database.
  - Cloud-hosted, reducing the need for local server maintenance.
- **Programming Language:**
  - Java 17 is used for its compatibility with Swing GUI and robust ecosystem.

5.2 External Dependencies

Firebase SDK

- Facilitates Firestore database communication, authentication, and real-time updates.
- Provides seamless integration with Firestore, with automatic scaling for high traffic.

Java Swing Framework

- A GUI library used to create interactive and visually appealing user interfaces.
- Provides flexibility for integrating custom components like profile cards and message panels.

Google Authentication Libraries

- Used to securely authenticate user logins, ensuring encrypted communication between the client and database.

Maven Dependencies:

- Firebase Admin SDK for backend database interactions.
- Google Cloud Firestore SDK for database CRUD operations.

6 System Decomposition and Exception Handling

6.1 System Decomposition

The UniVerse application is divided into modules for improved maintainability, scalability, and testing.

Module 1: User Management

Responsibilities:

- Handle user signup and login.
- Validate user credentials using hashed passwords.
- Manage user sessions through SessionManager.

Interactions:

- **GUI:** SignUporIn interface collects user inputs.
- **Controller:** FirestoreHandler processes login requests and validates credentials.
- **Database:** User data is stored and retrieved from the UserProfile collection in Firestore.

Module 2: Contact Management

Responsibilities:

- Add, view, and remove friends dynamically.
- Ensure that each user maintains a unique contact list.

Interactions:

- **GUI:** Homepage allows users to view and manage their contact list.
- **Controller:** FirestoreHandler manages Firestore's contacts subcollection.
- **Database:** Contacts are stored as subcollections within the UserProfile document.

Module 3: Messaging

Responsibilities:

- Send, retrieve, and display chat messages.
- Maintain real-time synchronization between users.

Interactions:

- **GUI:** Messaging provides a chat interface for users.
- **Controller:** FirestoreHandler.saveMessages writes messages to Firestore, while getChatHistory retrieves them.
- **Database:** Messages are stored in messages subcollections under chats.

6.2 Exception Handling

Our application included a lot of mechanisms to handle common errors and edge cases

1. Invalid User Input

Scenario: A user submits incomplete or invalid data during signup.
Solution:

- Input validation ensures all fields are filled.
- Error dialogs inform the user of the issue.

Example:

```
if (username.isEmpty() || email.isEmpty() || password.isEmpty()) {      You, 2 days ago • get messaging log
    JOptionPane.showMessageDialog(frame, "All fields are required.", "Error", JOptionPane.ERROR_MESSAGE);
    return;
}
```

2. Network Errors

Scenario: Poor internet connectivity disrupts Firestore operations.
Solution:

- Retry mechanisms attempt database operations up to three times.
- User is notified if retries fail.

Example:

```
try {
    writeResult.get();
} catch (InterruptedException | ExecutionException e) {
    JOptionPane.showMessageDialog(null, "Network error. Please try again.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

3. Authentication Errors

Scenario: A user enters incorrect login credentials.
Solution:

- Credentials are hashed before comparison.
- A user-friendly error message is displayed for mismatches.

4. Real-Time Sync Issues

Scenario: Snapshot listeners fail to fetch real-time updates.
Solution:

- Automatically reconnect listeners when connectivity is restored.
- Maintain a local cache to avoid data loss.

Benefits of Exception Handling

- Prevents application crashes.
- Improves user experience with clear feedback.
- Ensures system robustness in varying network conditions.

7 Conclusion

7.1 Summary of the system

The UniVerse application successfully fulfills its purpose as a social networking platform for university students. It integrates essential features like user profile management, real-time messaging, and contact handling, all built on a strong foundation using Java and Firebase Firestore. By following an **MVC architecture**, the system is structured to separate concerns, making it easier to maintain and expand. The design prioritizes usability and performance, ensuring a smooth and responsive experience for users.

7.2 Key Strengths

1. **Real-Time Communication:**
   Firestore's real-time capabilities allow chat messages and updates to sync instantly across users without delays.
2. **Scalable Backend:**
   The Firestore database is designed to handle growth, ensuring that as more users join, the system continues to perform efficiently.
3. **User-Friendly Interface:**
   The Java Swing-based GUI is intuitive, ensuring users can easily navigate the platform without technical barriers.
4. **Secure Data Handling:**
   Passwords are hashed, and Firebase ensures encrypted communication, protecting user data from unauthorized access.

7.3 Areas for Improvement

While the application achieves its main goals, there are opportunities to refine and expand its capabilities:

1. **Group Functionality:**
   Allow users to create and join groups based on shared interests or academic goals. Groups could include features like announcements, shared files, and discussion threads, making UniVerse a valuable tool for collaboration.

2. **Enhanced Messaging Features:**
   - Enable multimedia sharing (e.g., images, videos, and files) to make conversations richer.
   - Introduce group chats, allowing users to communicate in teams or clubs.
3. **Improved Performance with Query Optimization:**
   Implement Firestore indexing to improve query efficiency, especially as the database grows larger with increased user activity.
4. **Stronger Error Handling:**
   Enhance exception handling for network-related issues to provide a smoother user experience during connectivity disruptions.

7.4 Final Thoughts

The UniVerse application demonstrates a strong foundation for a university-oriented social platform. Its combination of secure user management, real-time communication, and scalable backend design ensures a reliable and engaging experience for users. With the suggested improvements, UniVerse has the potential to grow beyond its current scope, becoming not just a social networking platform but also a central hub for academic and social interaction in university communities.

Sprint 2 Updates

8.1 CRC Cards

Class Name:	Messaging
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Handle ChatGPT integration for user messages.	Collaborators: ● FirestoreHandler (for message storage) ● OpenAI API (for ChatGPT responses)

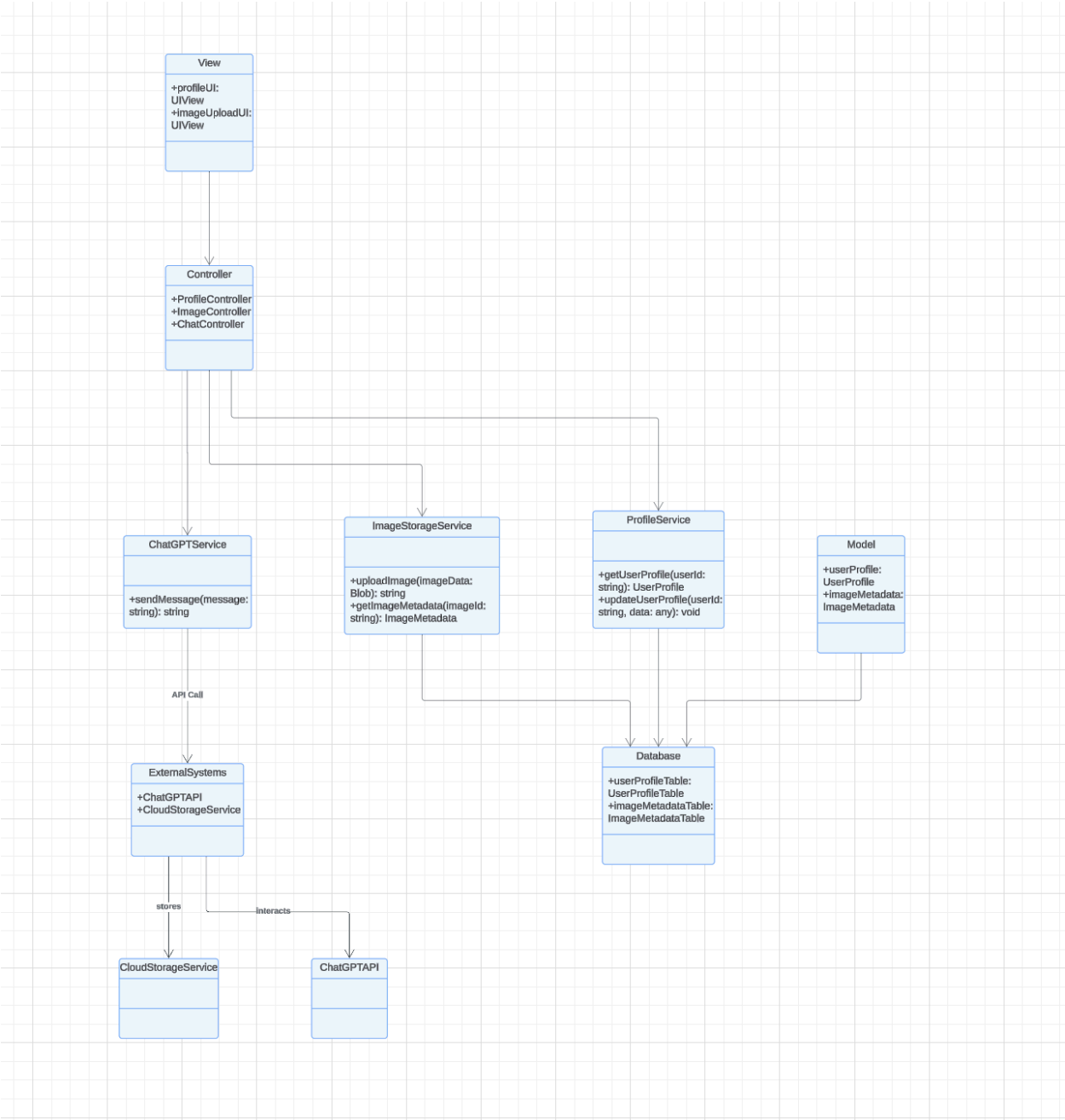
Class Name:	Homepage
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Support profile updates, including image uploads.	Collaborators: ● FirestoreHandler (for saving profile images)

Class Name:	Notification
Parent Class: Subclasses:	n/a n/a
Responsibilities: ● Allow users to receive notifications for new messages and new matches. ● View and navigate to messages from the notifications page.	Collaborators: ● FirestoreHandler(for storing notifications till they are read by user)

Class Name:	Groups
Parent Class: Subclasses:	n/a n/a

Responsibilities: ● Allow users to create new groups based on shared interests. ● Collect group details such as name, description, interest tags, and rules. ● Enable users to join or leave groups. ● Display a list of group members which users can add as friends. ● Provided link to chat ● Allow group creators (admins) to delete groups ● Integrate group data with the database (e.g., Firestore) for real-time updates	Collaborators: ● FirestoreHandler(for storing notifications till they are read by user) ● UserProfile(for extracting user group information)

8.2 Software Architecture Diagram



8.3 Database Design

Collection Name	Field	Data Type	Description
UserProfile	userId	String	Unique identifier for the user
	username	String	The display name of the user

	email	String	Email address for login
	passwordHash	String	Hashed password for authentication
	bio	String	Short description of the user
	dateOfBirth	String	User's birth date in dd/mm/yyyy format
	province	String	The province or state where the user resides
	university	String	Users university
	interests	List<String>	List of user-selected interests
new	***profilePicture***	***Base64 String***	***Stored the string value of the profile picture.***
Notifications	userId	***String***	
new	groups	***String***	***Stores the string value of the groups people joined***
	CreatorId	***String***	Unique identifier for the user
	groupDescription	***String***	*User written group descriptions*
	groupName	***String***	*User chosen group name*
	members	***String***	Unique identifier for all members of the group
	relatedInterests	***String***	*Users related interests*
	Rules	***String***	*Users group rules*

8.4 System Interaction

1. ChatGPT integration:
    a. Users sends a message on the messaging page to the UniVerse Bot
    b. The message is forwarded to the OpenAI API
    c. The API responds with a reply, which is then displayed in the chat interface
2. Profile Updates:
    a. User can upload a profile picture updated when signing up or through the home page
    b. The image is converted to Base64 and stored in Firestore
3. Notification
    a. Users receive notifications when a new message is sent.
    b. Users receive notifications when another user adds them as a friend.
    c. Users can navigate to the messaging chat when they click on view on the notifications page.
4. Groups
    a. Users can create groups based on their interests.

b. Users can see a displayed list of groups that were created.
c. Users can see information and rules for the group before joining.
d. Users can join any group(s) based on their interests.
e. Users can manage their group memberships (leave group).
f. Users can add friends directly from group pages.
g. Users can delete groups they've previously created.