Software Design Document for **Hobby-Based Social Media** App/Website (G8)

Contributors

Name	Roll Number
Aditya A. Waghmare	CS22BTECH11061
Siddhant S. Godbole	CS22BTECH11054
Gadekar S.B	CS22BTECH11022
Gaurav Choudekar	CS22BTECH11015

Contents

1. Software Architecture

1.1 Overview

1.1.1 System Overview

The Hobby-Based Social Media platform is a web application designed to help college students connect with each other based on shared hobbies and interests. The platform emphasizes anonymity and privacy, allowing users to interact without revealing their identities until they choose to do so. The system uses graph-based recommendations and an interactive questionnaire to match users with similar interests, fostering meaningful connections and group formations. The platform is institution-exclusive, ensuring that only verified students can join, thereby creating a safe and trusted environment for social interactions.

1.1.2 System Context

The system context is defined clearly in the SRS. Basically, the user is the main sink of the information. The user is also the main source of information/data.

1.1.3 Stakeholders

The main stakeholders for the system are the individual users who might use the system and the system designer/builder who will build the Hobby Based Social Media platform. The main concerns of the two stakeholders are:

- **Users:** The users are concerned with the privacy and security of their data. They are also concerned with the ease of use of the platform and the quality of the recommendations provided by the system.
- **System Designer/Builder:** The system designer/builder is concerned with the scalability and maintainability of the system. They are also concerned with the performance and reliability of the system.

1.1.4 Scope of this Document

This document describes the proposed architecture for the Hobby-Based Social Media platform. For architecture, we consider only the component and connector view.

1.1.5 Definitions and Acronyms

As given in the SRS.

1.2 Architecture Design

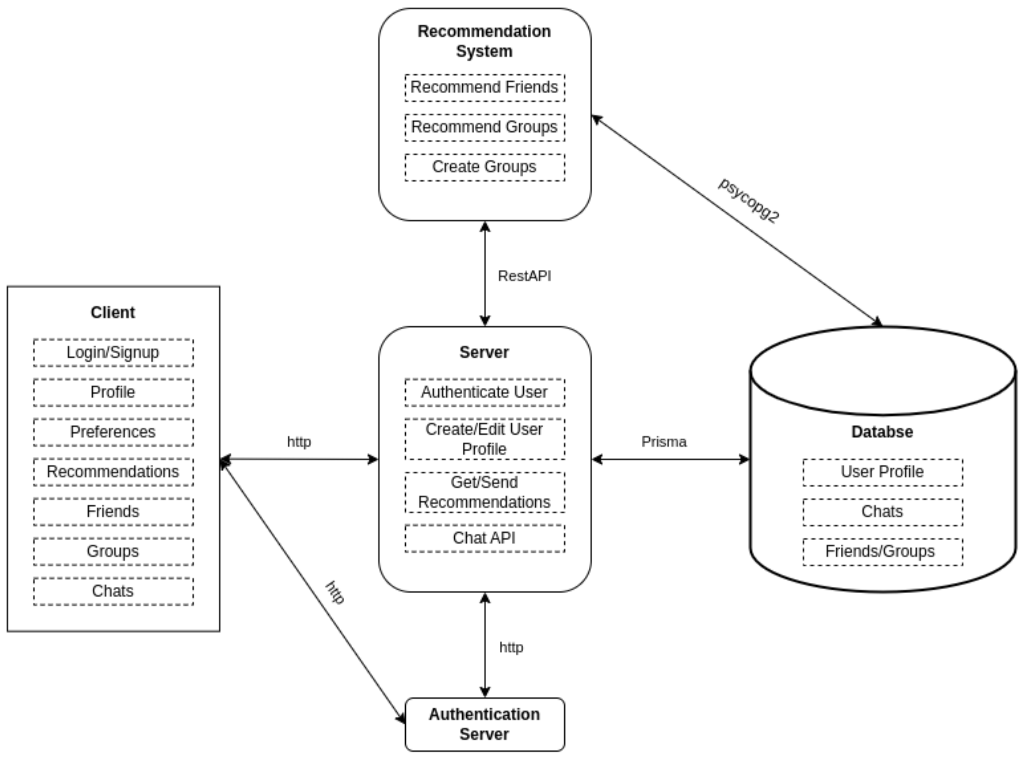

Figure 1: Component and Connector View of the Software

The system architecture follows a modular microservices-based approach to ensure scalability and maintainability. The core components of the system are:

- **Client:** Developed using React.js, the frontend is responsible for user interaction and interface rendering. It handles user login, profile management, and real-time chat.
- **Server:** The backend is implemented using Node.js. It processes user requests, manages session data, and handles real-time communication through WebSockets.

- **Recommendation System:** Implemented in Python using Flask, the recommendation system generates friend and group suggestions based on a collaborative filtering model using user preferences data. It also creates groups using clustering methods like K means for interest group detection.
- **Database:** PostgreSQL is used for structured data such as user profiles and connections.
- **Google Firebase Authentication:** Firebase is used to authenticate users through their institutional email IDs, ensuring secure and restricted access.
- **Connectors:** The frontend and server communicate over http. The server communicates with the recommendation system using REST APIs. The server communicates with the database using prisma. The recommendation system acceses the database with psycopg2.

This architecture ensures a clear separation of concerns, allowing independent scaling of components and easier maintenance. The REST API-based communication provides flexibility, while WebSocket connections handle real-time data exchange efficiently.

1.3 ATAM Analysis of the Architecture

Scenario	**Quality Attribute**	**Trade-off**	**Possible Solution**
High User Load During Peak Hours	Performance, Scalability	Increased load may slow down real-time chat and recommendations.	Use a load balancer and increase backend replicas during peak hours.
New User Signup and Onboarding	Usability, Performance	Delay in recommendation updates after signup.	Implement asynchronous processing for recommendation updates.
Real-time Chat with Multiple Users	Performance, Scalability	Increased latency with multiple concurrent chat users.	Use WebSocket-based communication for real-time performance.
Handling Data Consistency During Network Failures	Reliability, Availability	Loss of chat or friend request data during outages.	Use transaction logging and retry mechanisms.
Privacy Concern with Identity Reveal	Security, Privacy	User discomfort with accidental identity reveal.	Double-confirmation before identity reveal.
System Crash During High Load	Fault Tolerance	Temporary unavailability.	Deploy automatic failover and container-based recovery.
User Switching Between Devices	Usability, Compatibility	Sync issues and inconsistent state.	Use session-based state storage and WebSocket reconnection.
Friend Recommendation Failing Due to Model Update	Performance, Accuracy	Slow updates in recommendations.	Cache recent recommendations and refresh models periodically.

Table 1: ATAM Analysis of Proposed Architecture

2. Data Flow Diagram (DFD)

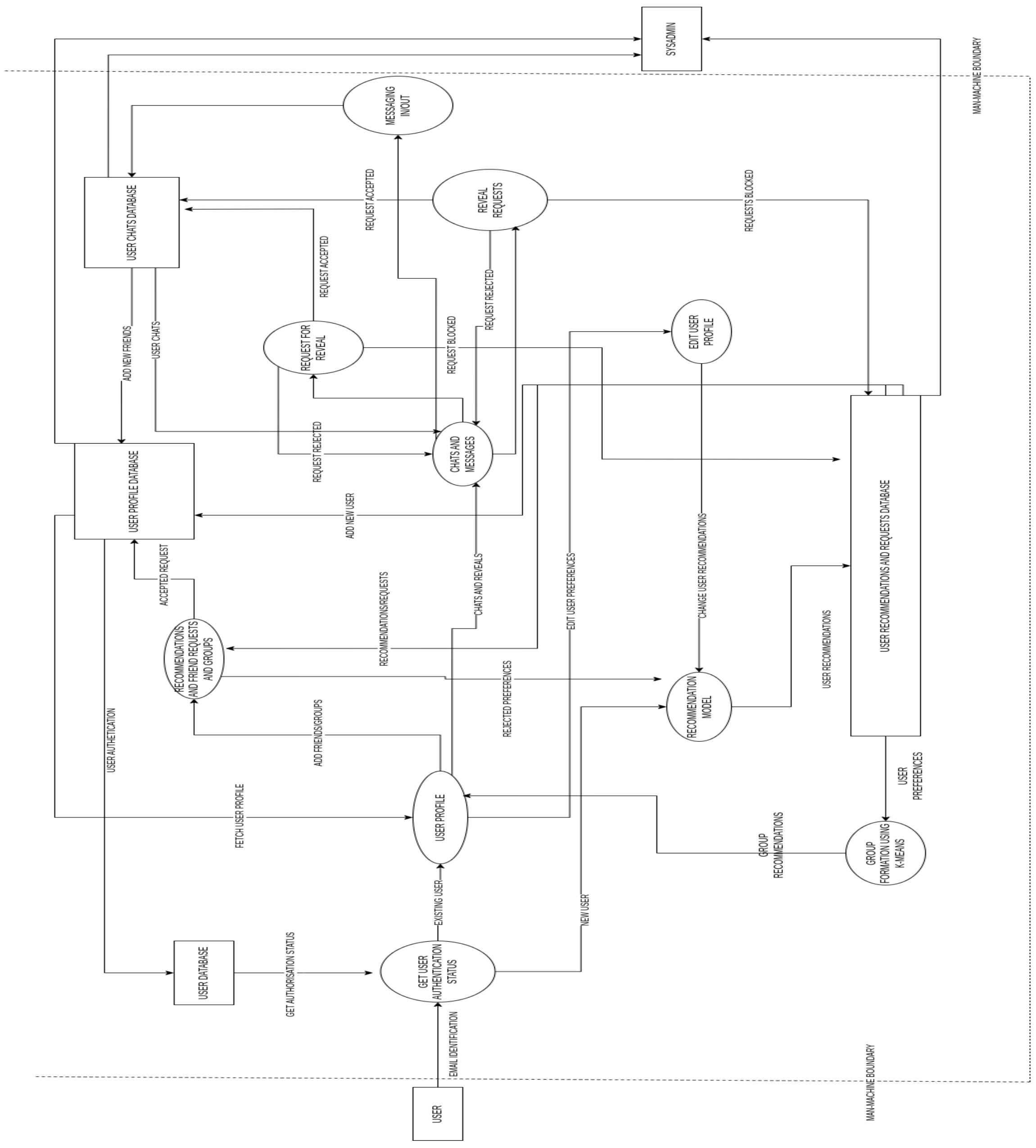

Figure 2: Data Flow Diagram

2.1 Abstract Input and Output

2.1.1 Most Abstract Input

The most abstract input to the system can be generalized as:

- **User Actions**: These encompass all interactions initiated by the user within the system. The key user actions include:
  - Registration and authentication using institutional email credentials.
  - Setting up and updating user profile information, including hobbies and preferences.
  - Sending and receiving friend/group recommendations and requests.
  - Engaging in conversations, including anonymous chatting.
  - Requesting identity reveals and accepting/rejecting such requests.
  - Blocking/unblocking users based on preferences.

2.1.2 Most Abstract Output

The system's most abstract output is:

- **User Connections and Interactions**: These are the final outcomes generated by processing the user actions. The key outputs include:
  - Successfully authenticated users gaining access to the platform.
  - Personalized recommendations for new friends and groups based on shared interests.
  - Established connections through accepted friend requests and group formations.
  - Real-time or asynchronous messaging between users.
  - Identity reveal status updates based on user interactions.
  - Blocked users and restricted communications as per user preferences.

The system takes user actions as input and processes them through multiple components, ultimately producing meaningful social connections and engagement among college students.

3. Structure Charts

3.1 First Level Factored Modules

These are the majority variations of functional modules that structure the entire system:

- **Client Side Application** – Covers user interaction modules such as login, signup, messaging, etc.
- **Server Side Application** – Manages database interactions and server-side processing.
- **Recommendation System** – Generates recommendations based on user data as well as identifies new groups based on clustering algorithms.
- **Request Handler** – Processes user requests and calls appropriate modules for handling.

3.2 Factored Input Modules

The input modules are responsible for handling user inputs and requests. These modules include:

- Sign up
- Log in
- Log out
- Set Preferences / Profile
- Edit preferences
- Edit profile
- Send a Friend Request
- Accept a Friend Request
- Reject a Friend Request
- Send a Message To Another Group/User

- Send a reveal request
- Accept Reveal Request
- Reveal Identity to another User
- Block a User/Group
- Reject a Group join Invitation
- Accept a Group join Invitation
- Send Messages
- Send Reveal Request

3.3 Factored Output Modules

The output modules are responsible for generating and displaying the results based on user inputs and system processing. These modules include:

- View profile
- View preferences
- Get Recommendations from server
- Get Current Friends And Groups
- Receive New/Old Messages/Requests From other Group/User
- Fetch Reveal Request
- Fetch Profile / Preferences
- Fetch Recommendations
- Fetch Current Friends/Groups
- Fetch Messages/Requests
- Fetch Users reveal’s and Revealed List
- Add New Groups
- Get Groups from Database
- Get Users from Database

3.4 Factored Transform Modules

The transform modules are responsible for processing the data and performing the necessary computations to generate the desired outputs. These modules include:

- Create User Profile
- Save Profile / Preferences
- Update Friendships
- Update Group membership
- Update Identity Revealed Status
- Update Users Block List
- Recommend Groups
- Recommend Friends
- Get list of new Groups

3.5 Final structure chart showing all the modules

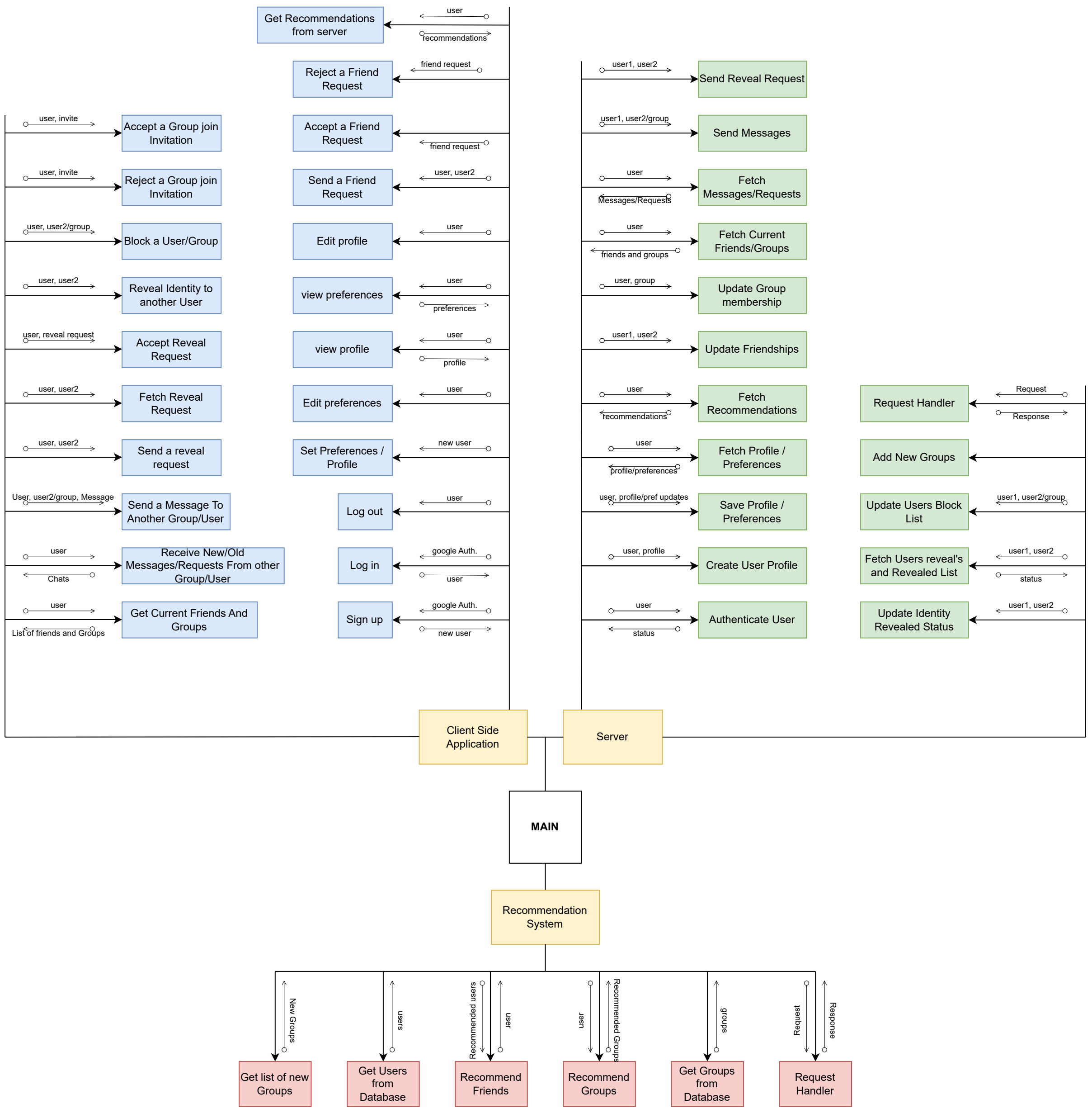

Figure 3: Structure Chart

4. Design Analysis

4.1 Module List and Description

Module	Type	Cohesion	Description
Sign up	Input	Functional	Handles user registration using institutional email.
Log in	Input	Functional	Authenticates user and grants access to the platform.
Log out	Input	Functional	Ends user session and logs out the user.
Set Preferences / Profile	Input	Functional	Allows users to set their preferences and profile details.
Edit preferences	Input	Functional	Enables users to update their preferences.
View profile	Output	Functional	Displays user profile information.
View preferences	Output	Functional	Shows user preferences.
Edit profile	Input	Functional	Allows users to update their profile details.
Send a Friend Request	Input	Functional	Sends a friend request to another user.
Accept a Friend Request	Input	Functional	Accepts a received friend request.
Reject a Friend Request	Input	Functional	Rejects a received friend request.
Get Recommendations from server	Output	Functional	Fetches friend and group recommendations from the server.
Get Current Friends And Groups	Output	Functional	Retrieves the list of current friends and groups.

Table 2: List of All Modules

Receive New/Old Messages/Requests From other Group/User	Output	Procedural	Receives messages and requests from other users or groups.
Send a Message To Another Group/User	Input	Functional	Sends a message to another user or group.
Send a reveal request	Input	Functional	Sends a request to reveal identity to another user.
Fetch Reveal Request	Output	Functional	Fetches reveal requests from other users.
Accept Reveal Request	Input	Functional	Accepts a reveal request from another user.
Reveal Identity to another User	Input	Functional	Reveals user's identity to another user.
Block a User/Group	Input	Functional	Blocks a user or group.
Reject a Group join Invitation	Input	Functional	Rejects a group join invitation.
Accept a Group join Invitation	Input	Functional	Accepts a group join invitation.
Authenticate User	Coordinate	Functional	Authenticates user credentials.
Create User Profile	Transform	Procedural	Creates a new user profile.
Save Profile / Preferences	Transform	Functional	Saves user profile and preferences.
Fetch Profile / Preferences	Output	Functional	Fetches user profile and preferences.
Fetch Recommendations	Output	Functional	Retrieves friend and group recommendations.
Update Friendships	Transform	Functional	Updates user friendships.
Update Group membership	Transform	Functional	Updates user group memberships.
Fetch Current Friends/Groups	Output	Functional	Retrieves current friends and groups.
Fetch Messages/Requests	Output	Functional	Fetches messages and requests.

Table 3: List of All Modules

Send Messages	Input	Functional	Sends messages to users or groups.
Send Reveal Request	Input	Functional	Sends a reveal request to another user.
Update Identity Revealed Status	Transform	Functional	Updates the status of identity reveal.
Fetch Users reveal's and Revealed List	Output	Functional	Fetches the list of users who have revealed their identity.
Update Users Block List	Transform	Functional	Updates the list of blocked users.
Add New Groups	Output	Functional	Adds new groups to the system.
Server Request Handler	Coordinate	Communication	Handles server's incoming requests and routes them to appropriate modules.
Recommendation System Request Handler	Coordinate	Communication	Handles incoming requests and routes them to appropriate modules.
Get Groups from Database	Output	Functional	Retrieves groups from the database.
Recommend Groups	Transform	Functional	Generates group recommendations.
Recommend Friends	Transform	Functional	Generates friend recommendations.
Get Users from Database	Output	Functional	Retrieves users from the database.
Get list of new Groups	Transform	Functional	Creates a list of new groups.

Table 4: List of All Modules

4.2 Module Count by Type

Module Type	Count
Input	18
Output	14
Transform	9
Coordinate	3
Composite	0

Table 5: Count of Modules by Type

4.3 Complex or Error-Prone Modules

- **Input: Send a Message To Another Group/User** - This module is complex due to the need for real-time message delivery and handling various edge cases like network failures.
- **Transform: Recommend Friends** - This module is error-prone because it involves complex algorithms for generating accurate friend recommendations based on user preferences and activities.
- **Output: Fetch Messages/Requests** - This module is complex as it needs to handle a large volume of messages and requests efficiently, ensuring timely delivery and consistency.

4.4 Top-3 Modules by Fan Out and Fan In

Module	Fan Out	Fan In
Server Request Handler	14	22
Recommendation System Request Handler	3	2
Create User profile	2	1

Table 6: Top-3 Modules by Fan Out and Fan In

5. Detailed Design Specification

Interface of all the final level factored modules in the form of classes with attributes and methods:

```

class SignUp:
    def __init__(self, gmail):
        self.email = gmail

    def register(self):
        pass


class LogIn:
    def __init__(self, gmail):
        self.email = gmail

    def authenticate(self):
        pass


class LogOut:
    def __init__(self, user_id):
        self.user_id = user_id

    def logout(self):
        pass


class SetPreferencesProfile:
    def __init__(self, user_id, preferences):
        self.user_id = user_id
        self.preferences = preferences

    def set_preferences(self):
        pass


class EditPreferences:
    def __init__(self, user_id, preferences):
        self.user_id = user_id
        self.preferences = preferences

    def edit_preferences(self):
        pass


class ViewProfile:
    def __init__(self, user_id):
        self.user_id = user_id

    def view_profile(self):
        pass


class ViewPreferences:
    def __init__(self, user_id):
        self.user_id = user_id

    def view_preferences(self):
        pass


class EditProfile:
    def __init__(self, user_id, profile_data):
        self.user_id = user_id
        self.profile_data = profile_data

    def edit_profile(self):
        pass


class SendFriendRequest:
    def __init__(self, user_id, friend_id):
        self.user_id = user_id
        self.friend_id = friend_id

    def send_request(self):
        pass


class AcceptFriendRequest:
    def __init__(self, user_id, friend_request_id):
        self.user_id = user_id
        self.friend_request_id = friend_request_id
```

```python

    def accept_request(self):
        pass


class RejectFriendRequest:
    def __init__(self, user_id, friend_request_id):
        self.user_id = user_id
        self.friend_request_id = friend_request_id

    def reject_request(self):
        pass


class GetRecommendationsFromServer:
    def __init__(self, user_id):
        self.user_id = user_id

    def get_recommendations(self):
        pass


class GetCurrentFriendsAndGroups:
    def __init__(self, user_id):
        self.user_id = user_id

    def get_friends_groups(self):
        pass


class ReceiveMessagesRequests:
    def __init__(self, user_id, chat_id):
        self.user_id = user_id
        self.chat_id = chat_id

    def receive_messages_requests(self):
        pass


class SendMessageToGroupOrUser:
    def __init__(self, user_id, chat_id, message):
        self.user_id = user_id
        self.chat_id = chat_id
        self.message = message

    def send_message(self):
        pass


class SendRevealRequest:
    def __init__(self, user_id, recipient_id):
        self.user_id = user_id
        self.recipient_id = recipient_id

    def send_reveal_request(self):
        pass


class FetchRevealRequest:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_reveal_request(self):
        pass


class AcceptRevealRequest:
    def __init__(self, user_id, requester_id):
        self.user_id = user_id
        self.requester_id = requester_id

    def accept_reveal_request(self):
        pass


class RevealIdentityToUser:
    def __init__(self, user_id, recipient_id):
        self.user_id = user_id
        self.recipient_id = recipient_id

    def reveal_identity(self):
        pass


class BlockUserGroup:
    def __init__(self, user_id, chat_id):
        self.user_id = user_id
        self.chat_id = chat_id

    def block(self):
        pass


class RejectGroupJoinInvitation:
    def __init__(self, user_id, group_id):
        self.user_id = user_id
        self.group_id = group_id

    def reject_invitation(self):
        pass


class AcceptGroupJoinInvitation:
    def __init__(self, user_id, group_id):
        self.user_id = user_id
        self.group_id = group_id

    def accept_invitation(self):
        pass


class AuthenticateUser:
    def __init__(self, email):
        self.email = email

    def authenticate(self):
        pass


class CreateUserProfile:
    def __init__(self, user_data):
        self.user_data = user_data

    def create_profile(self):
        pass


class SaveProfilePreferences:
    def __init__(self, user_id, profile_data):
        self.user_id = user_id
        self.profile_data = profile_data

    def save_profile_preferences(self):
        pass


class FetchProfilePreferences:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_profile_preferences(self):
        pass


class FetchRecommendations:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_recommendations(self):
        pass


class UpdateFriendships:
    def __init__(self, user_id, friend_id, action):
        self.user_id = user_id
        self.friend_id = friend_id
        self.action = action

    def update_friendships(self):
        pass


class UpdateGroupMembership:
    def __init__(self, user_id, group_id, action):
        self.user_id = user_id
        self.group_id = group_id
        self.action = action

    def update_group_membership(self):
        pass


```

```python
class FetchCurrentFriendsGroups:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_current_friends_groups(self):
        pass


class FetchMessagesRequests:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_messages_requests(self):
        pass


class SendMessages:
    def __init__(self, user_id, chat_id, message):
        self.user_id = user_id
        self.recipient_id = chat_id
        self.message = message

    def send_messages(self):
        pass


class SendRevealRequest:
    def __init__(self, user_id, recipient_id):
        self.user_id = user_id
        self.recipient_id = recipient_id

    def send_reveal_request(self):
        pass


class UpdateIdentityRevealedStatus:
    def __init__(self, user_id, recipient_id, status):
        self.user_id = user_id
        self.recipient_id = recipient_id
        self.status = status

    def update_identity_revealed_status(self):
        pass
```

```python


class FetchUsersRevealsList:
    def __init__(self, user_id):
        self.user_id = user_id

    def fetch_users_reveals_list(self):
        pass


class UpdateUsersBlockList:
    def __init__(self, user_id, chat_id, action):
        self.user_id = user_id
        self.chat_id = chat_id
        self.action = action

    def update_users_block_list(self):
        pass


class AddNewGroups:
    def __init__(self):

    def add_new_groups(self):
        pass


class ServerRequestHandler:
    def __init__(self, request):
        self.request = request

    def handle_request(self):
        pass


class RecommendationSystemRequestHandler:
    def __init__(self, request):
        self.request = request

    def handle_request(self):
        pass


```

```python
class GetGroupsFromDatabase:
    def __init__(self):
        pass

    def get_groups(self):
        pass


class RecommendGroups:
    def __init__(self, user_id):
        self.user_id = user_id

    def recommend_groups(self):
        pass


class RecommendFriends:
    def __init__(self, user_id):
        self.user_id = user_id

    def recommend_friends(self):
        pass


class GetUsersFromDatabase:
    def __init__(self):
        pass

    def get_users(self):
        pass


class GetListOfNewGroups:
    def __init__(self):
        pass

    def get_list_of_new_groups(self):
        pass
```