أحمد اشرف محمود عبد العزيز 2022170014

بولس عونى لطيف يسى مرزوق 2022170099

جلال محمد جلال محمد طعيمه 2022170111

حازم ابراهيم محمد عبد الفتاح 2022170123

محمد وائل محمد عبد الستار 2022170392

توماس سيدهم بسطوروس دميان 2022170108

Facebook Management system

team: 65

# Facebook Management System

## A. OVERVIEW

The Facebook management system is a Java-based application that simulates the functionality of Facebook. The system is composed of several models, Enums, services, and a view class called Dashboard.

## B. Models

The system has eight models:

1. User (abstract)
2. Client
3. Conversation
4. Message
5. Comment
6. Group
7. Post
8. React

## C.Enums

The system uses five Enums:

1. Friend Type (restricted or regular)
2. Gender (Male or Female)
3. Privacy Option (Friends, Public)
4. React Type (like, wow, love, sad, haha, angry, care)
5. Relationship Status (single, married, engaged)

D. Services

The system has five services:

1. Comments Service
2. Conversation Service
3. File Service
4. User Service
5. User Context

## E. Dashboard

The Dashboard is a view class that interacts with the user through the terminal. It creates a dynamic flow to the project where the user chooses a functionality that returns a specific data then returns to specific dashboard that is logical to the business case.

PROJECT FLOW

1. **Start the Application**: The application starts by displaying the main menu to the user through the Dashboard class and reading all the files.
2. **User Login/Signup**: The user has the option to login or signup using the UserService. If the user chooses to login, the login() method is called. If the user chooses to signup, the signUp() method is called then returns to main menu through the dashboard class.
3. **User Dashboard**: Once the user is logged in, they are taken to the user dashboard where they can see their friends, posts, timeline, groups, friend requests, and sent friend requests. They can also search for other users and groups, send friend requests, write posts, join groups, and more.
4. **Post Interaction**: If the user chooses to interact with a post, the PostService is used. The user can like a post, share a post, write a comment on a post, get the reactions on a post, and get the comments on a post.
5. **Comment Interaction**: If the user chooses to interact with a comment, the CommentsService is used. The user can react to a comment or reply to a comment.
6. **Conversation Interaction**: If the user chooses to interact with a conversation, the ConversationService is used. The user can see their conversations, see the content of a conversation, or send a message.
7. **File Interaction**: The FileService is used to save and read all messages, conversations, posts, users, and friends.
8. **End of Application**: The application ends when the user chooses to log out or close the application and saving to all the files using FileService.

PROJECT STRUCTURE

1. **Models**: Models represent the data and the rules that govern access to and updates of this data. In your project, models like User, Post, Comment, etc., are used to represent entities with their properties and behaviors.
2. **Services**: Services contain business logic, calculations, and database calls. They act upon the data (models) and are used to implement all the operations your application needs to perform on your models. For example, PostService contains methods to like a post, share a post, write a comment, etc. For example, the react to a post method uses the getters and setters in the Post model and in the User Model to initialize a react using the Constructor of the of the React Model then pushes this react object inside the ArrayList in the post object.
3. **Enums**: Enums, or enumerated types, are a special kind of class that represents a group of constants (unchangeable variables). In our project, enums like FriendType, Gender, PrivacyOption, etc., are used to represent a fixed set of constants, which can be used as a type for variables, method parameters, and return values.
4. **Dashboard**: The Dashboard class in your project is a part of the view component in the MVC (Model-View-Controller) architecture. It interacts with the user and displays the output from the services. It provides options for the user to perform certain functionalities in a dynamic approach.
5. **UserContext**: The UserContext class is used to maintain user session data while the user is logged in. It’s a form of global state management for the current user. When a user logs in, their user data is stored in UserContext, and then this data can be accessed from anywhere in the application without having to pass around the user object.

This structure allows for separation of concerns, where each part of the code has a specific job, making the code more organized, reusable, and easier to maintain.

## Input/Output Scenarios

1. PostService

* like(Post post, User currentUser)
  + Input: A Post object and a User object representing the current user.
  + Output: A reaction is added to the specified post by the current user.
* writeComment(Post post, User user)
  + Input: A Post object and a User object.
  + Output: A comment is written by the user on the specified post.
* share(User account, Post post)
  + Input: A User object representing the account and a Post object.
  + Output: The specified post is shared by the account.
* getReacts(Post post)
  + Input: A Post object.
  + Output: Returns the reactions on the specified post.
* getComments(Post post)
  + Input: A Post object.
  + Output: Returns the comments on the specified post.
* Timeline(ArrayList<Client> clients)
  + Input: An ArrayList of Client objects.
  + Output: Returns the timeline of the clients in a random order.

### UserService

* login()
  + Input: None
  + Output: Logs in the user and returns the CurrentUser object if correct data.
* signUp()
  + Input: None
  + Output: Signs up a new user and returns to main menu dashboard.
* seeFriends()
  + Input: None
  + Output: logs a list of the user’s friends then returns to user dashboard.
* seePosts()
  + Input: None
  + Output: Returns a list of the user’s posts.
* seeTimeline()
  + Input: None
  + Output: Returns the user’s timeline.
* seeGroups()
  + Input: None
  + Output: Returns a list of the groups the user is part of.
* getFriendRequests(User currentUser)
  + Input: A User object representing the current user.
  + Output: logs a list of friend requests for the current user and then you choose to accept or reject request then returns the User Dashboard.
* getSentFriendRequests(User currentUser)
  + Input: A User object representing the current user.
  + Output: logs a list of friend requests sent by the current user then returns to User Dashboard.
* userSearch(String input)
  + Input: A String representing the search input.
  + Output: Returns a list of users that contain the substring of the search input.
* sendFriendRequest(User currentUser)
  + Input: A User object representing the current user.
  + Output: logs list of users then allows the user to choose an id of another then send a friend request from the current user.
* writePost(User currentUser)
  + Input: A User object representing the current user.
  + Output: Writes a post by the current user then returns to the post dashboard of the post written.
* groupSearch(String input)
  + Input: A String representing the search input.
  + Output: Returns a list of groups that match the search input.
* joinGroup()
  + Input: None
  + Output: Joins a group.
* getFriendship(User currentUser)
  + Input: A User object representing the current user.
  + Output: Returns the Common Posts of the current user and the other user he will choose through the function using the userSearch()..
* getMutualFriends(User currentUser)
  + Input: A User object representing the current user.
  + Output: Returns a list of mutual friends of the current user and the other user he will choose through the function using the userSearch().
* getClients()
  + Input: None
  + Output: Returns a list of clients.
* saveUsers(ArrayList<Client> clients)
  + Input: An ArrayList of Client objects.
  + Output: Saves the clients.

### ConversationService

* seeConversations(User user)
  + Input: A User object.
  + Output: logs a list of conversations of the user then the user chooses to chat or not, if not returns to user Dashboard, if chat the user will start chatting process.
* seeConversationContent(User user, User chosenUser)
  + Input: Two User objects representing the user and the chosen user.
  + Output: logs the conversation content between the user and the chosen user then returns to Conversation Dashboard.
* sendMessage()
  + Input: None
  + Output: Sends a message.

### CommentsService

* react(User user, Comment comment)
  + Input: A User object and a Comment object.
  + Output: Adds a reaction to the comment from the user then returns to Post Dashboard.
* reply(User user, Comment comment)
  + Input: A User object and a Comment object.
  + Output: Replies to the comment from the user then returns to Post Dashboard.

### FileService

* saveAllMessages()
  + Input: None
  + Output: Saves all messages.
* readAllMessages()
  + Input: None
  + Output: Reads all messages.
* saveAllConversation()
  + Input: None
  + Output: Saves all conversations.
* readAllConversations()
  + Input: None
  + Output: Reads all conversations.
* saveAllPosts(UserService userService)
  + Input: A UserService object.
  + Output: Saves all posts.
* readAllPosts()
  + Input: None
  + Output: Reads all posts.
* saveAllUsers(ArrayList<Client> users)
  + Input: An ArrayList of Client objects.
  + Output: Saves all users.
* ReadAllUsers()
  + Input: None
  + Output: Reads all users.
* saveAllFriends(ArrayList<Client> users)
  + Input: An ArrayList of Client objects.
  + Output: Saves all friends.
* readUserFriends(ArrayList<Client> clients)
  + Input: An ArrayList of Client objects.
  + Output: Reads all friends of the clients.