**Technical Documentation**

**Data Structures**

We have used three different data structures such as graph for social networking, LinkedList for User connections and posts (used for implementing graph as separate chaining) and array for different options.

1. **Implementation of Graph**: Graph is implemented using separate chaining. Class named as SocialMediaApp represents graph which contains single LinkedList and this LinkedList of users is of datatype ‘User’ where User class includes attributes such as username, password, gender, email, and profile picture, LinkedList<String> friends, LinkedList <String>posts.
2. **Implementation of LinkedList:**

* LinkedList<String> friends: This attribute of user represents his/her friends.
* LinkedList<String>posts: This attribute of user represents his/her posts. In this case post can be images so the URL or addresses of images are stored in this linked list

1. **Implementation of Array:** An array is used to store the options for the type of post the user can create.

**Algorithms Implementation Details**

At the launch of application, existing users are loaded using filing system.

**1. Signup Algorithm :** The signup process ensures that:

* Usernames are unique.
* Passwords and emails follow specific validation rules.
* User data and profile pictures are stored properly.

**Algorithm Details:**

1. **Input**: Username, password, email, gender, and profile picture file.
2. **Check Username Uniqueness**:
   * Iterate through the Users list.
   * If a user with the same username exists, display an error.
3. **Password Validation**:
   * Check the password's length (8-16 characters).
   * Ensure it contains at least one uppercase letter, lowercase letter, digit, and special character.
4. **Email Validation**:
   * Check if the email matches the regex pattern for @gmail.com.
5. **Profile Picture Handling**:
   * Read the profile picture file and save it in a directory (profile\_pictures/<username>).
6. **Store User Data**:
   * Save user details (e.g., username, email) in a user.txt file in the corresponding directory.
   * Save the profile picture as profile.jpeg.
7. **Create User Object**:
   * Create a new User object and add it to the Users list.
8. **Output**: Success message or error message.

**2. Login Algorithm :** The login process verifies a user’s credentials and redirects them to the homepage.

**Algorithm Details:**

1. **Input**: Username and password.
2. **Authentication**:
   * Iterate through the Users list.
   * If a match is found with the given username and password:
     + Mark the user as LoggedinUser.
     + Redirect to the homepage.
   * If no match is found, show an error message.
3. **Forgot Password**:
   * If the user forgets the password:
     + Open a new screen.
     + Ask for the email address.
     + If the email is valid and matches a stored user:
       - Display the password in a text field.
4. **Output**: Redirect to homepage on success or show error messages for failure.

**3. Feed Algorithm :** The feed aggregates posts from the logged-in user and their friends.

**Algorithm Details:**

1. **Input**: Logged-in user.
2. **Fetch User Posts**:
   * Retrieve all posts from the loggedInUser.posts.
3. **Fetch Friends' Posts**:
   * Iterate through loggedInUser.friends to get their names.
   * Retrieve posts from each friend’s posts list.
4. **Display Posts**:
   * Create a panel for each post:
     + If it's an image post (e.g., Image: path), display the image.
     + If it's text, truncate it to 20 words and display the text.
   * Add the panels to the feed area.
5. **Output**: Scrollable feed UI showing posts.

**4. Friend Suggestions Algorithm :** This feature shows a list of users who are not friends with the logged-in user.

**Algorithm Details:**

1. **Input**: loggedInUser object.
2. **Find Non-Friends**:
   * Iterate through all users in app.Users.
   * If the user is not in loggedInUser.friends and not the loggedInUser themselves:
     + Add them to the suggestion list.
3. **Display Suggestions**:
   * For each suggestion:
     + Display their name and profile picture.
     + Add an "Add Friend" button.
4. **Add Friend**:
   * When the "Add Friend" button is clicked:
     + Add the selected user to loggedInUser.friends.
     + Show a confirmation message.
5. **Output**: Updated friends list and refreshed UI.

**5. Post Creation Algorithm :** Users can create text or image posts.

**Algorithm Details:**

1. **Input**: Post content or image file path.
2. **Process Post**:
   * If it’s a text post:
     + Add the text directly to loggedInUser.posts.
   * If it’s an image post:
     + Save the image in the user’s directory.
     + Add the file path to loggedInUser.posts with the prefix Image:.
3. **Update Feed**:
   * Re-render the feed to include the new post.
4. **Output**: Updated posts list and refreshed feed UI.

**6. Profile Display Algorithm :** Displays the profile details of the logged-in user.

**Algorithm Details:**

1. **Input**: loggedInUser.
2. **Retrieve Details**:
   * Fetch username, email, gender, and profile picture path.
3. **Display Profile**:
   * Show the profile picture.
   * Display username, email, and gender as labels.
4. **Output**: Profile UI.

**7. Friend List Algorithm :** Displays all friends of the logged-in user.

**Algorithm Details:**

1. **Input**: loggedInUser.friends.
2. **Retrieve Friends**:
   * Iterate through the Users list.

For each friend in loggedInUser.friends, retrieve their profile picture and username.

1. **Display Friends**:
   * Create a panel for each friend with their profile picture and name.
2. **Output**: Scrollable friend list UI.

**8. Logout Algorithm :** Logs out the current user and redirects to the welcome page.

**Algorithm Details:**

1. **Clear User Session**:
   * Set loggedInUser to null.
2. **Redirect**:
   * Clear the current frame and call createAndShowGUI() to display the welcome page.
3. **Output**: Welcome page UI.

**9. Data Storage and Retrieval**

**User Data:**

1. User data is stored in a text file (user.txt) in each user's directory.
2. On app startup:
   * Read all user directories.
   * Load user data into memory (app.Users list).
3. Profile pictures are stored in profile\_pictures/<username> and referenced via their paths.

**10. Validation Algorithms**

* **Password Validation**:
  + Regex ensures that the password:
    - Has at least 8 characters.
    - Contains uppercase, lowercase, digit, and special character.
* **Email Validation**:
  + Regex ensures the email ends with @gmail.com.