SOCIAL MEDIA APPLICATION (API)

Introduction:

This project is a simple social media application designed to allow users to connect and interact through a streamlined and user-friendly platform. The primary purpose of this application is to provide users with the ability to:

Register and Create Profiles:
Users can sign up by providing basic details such as email and password. Upon registration, an OTP and a secret key are sent to the user's email for verification, ensuring secure access.

Profile Management:
Once logged in, users can personalise their profiles by updating their image, bio, and privacy settings. Users can also manage their profiles by deleting or modifying the information as needed.

Post Creation and Interaction:
Users can create posts that include images and descriptions, share them with their network, and engage with others by liking and commenting on posts.

Communication:
The application includes a chat feature, allowing users to communicate directly with each other.

This social media platform is designed to be straightforward, focusing on essential features that enable users to share content, connect with others, and maintain a personalised profile. The target audience includes individuals who are looking for a simple yet effective way to interact and share content online.

User Registration

1.)Endpoint: POST /registration

- **Description:** Registers a new user on the platform. After registration, an OTP and secret key are sent to the user to complete the verification process.
- **Request Body:**
    - email (str): Required, should be unique. The user's email address.
    - password (str): Required. The password for the user's account.
    - name (str): Required. The full name of the user.
- **Response:**
    - **201 Created:** Indicates successful registration.
        - otp (int): OTP sent to the user's email.
        - secret_key (str): Secret key for the user.
        - message (str): Instructions for the user to verify the OTP.
- **Error Responses:**
    - **400 Bad Request:** If a user with the provided email already exists.

Verify OTP

2.)Endpoint: GET /verify_otp

- **Description:** Verifies the OTP sent to the user's during registration. This is a crucial step for completing the registration process.
- **Query Parameters:**
    - key (str): Required. The secret key sent during registration.
    - otp (int): Required. The OTP received by the user.
- **Response:**
    - **200 OK:** Confirmation that the OTP has been verified successfully.
        - message (str): Confirmation message of OTP verification.
- **Error Responses:**
    - **400 Bad Request:** If OTP verification fails.

Forgot Password

3.)Endpoint: GET /forgot_password/{email}

- **Description:** Allows users to reset their password by sending an OTP and secret key to the registered email. This is used when a user forgets their password.
- **Path Parameter:**
    - email (str): Required. The registered email of the user.

- **Response:**
  - **200 OK:** OTP and secret key are sent to the user's email.
    - otp (int): OTP sent to the user.
    - secret_key (str): Secret key for password reset.
    - message (str): Instructions for verifying the OTP and resetting the password.
- **Error Responses:**
  - **400 Bad Request:** If the email is not found.

Reset Password

4.)Endpoint: PUT /reset_password

- **Description:** Allows users to reset their password by providing the secret key, OTP, and a new password. This endpoint is used after receiving the OTP from the forgot password request.
- **Request Body:**
  - secret_key (str): Required. The secret key sent to the user.
  - otp (int): Required. OTP received by the user.
  - new_password (str): Required. The new password for the account.
- **Response:**
  - **200 OK:** Confirmation that the password has been reset successfully.
    - message (str): Confirmation message of password reset.
- **Error Responses:**
  - **400 Bad Request:** If the provided details are incorrect or if the password reset fails.

Resend OTP

5.)Endpoint: GET /resend_otp/{email}

- **Description:** Regenerates and sends a new OTP to the user's email if the previous OTP has expired or was not received.
- **Path Parameter:**
  - email (str): Required. The user's registered email.
- **Response:**
  - **200 OK:** New OTP and secret key sent.
    - otp (int): New OTP.
    - secret_key (str): Secret key for the user.
    - message (str): Instructions to verify the new OTP.
- **Error Responses:**
  - **400 Bad Request:** If the email is not found.

Get My Profile

6.)Endpoint: GET /profile/me

- **Description:** Retrieves the profile information of the currently logged-in user. Useful for displaying user details on their profile page.
- **Response:**
    - **200 OK:** Returns the user's profile information.
        - name (str): Username.
        - id (int): User ID.
        - profile_name (str): Profile name.
        - bio (str): User bio.
        - image (str): URL to the profile image.
        - is_private (bool): Privacy status of the profile.

Get All Profiles

7.)Endpoint: GET /profile/all

- **Description:** Retrieves a list of all user profiles. Useful for displaying a list of users on the platform.
- **Response:**
    - **200 OK:** A list of profiles.
        - Each profile contains:
            - name (str): Username.
            - id (int): User ID.
            - profile_name (str): Profile name.
            - bio (str): User bio.
            - image (str): URL to the profile image.
            - is_private (bool): Privacy status.

Get Profile by Username

8.)Endpoint: GET /profile/{user_name}

- **Description:** Searches for user profiles by username. Allows users to find other profiles by their username.
- **Path Parameter:**
    - username (str): Required. The username to search for.
- **Response:**
    - **200 OK:** A list of profiles matching the username.
- **Error Responses:**

- **400 Bad Request:** If no user is found with the provided username.

Update My Profile

9.)Endpoint: PUT /profile/me/update

- **Description:** Allows the logged-in user to update their profile information. This includes updating profile name, image, username, bio, and privacy status.
- **Form Data (Optional Fields):**
  - profile_name (str): The new profile name (must be unique).
  - image (UploadFile): The new profile image.
  - username (str): The new username.
  - bio (str): The new bio.
  - private_or_not (bool): The privacy status.
- **Response:**
  - **200 OK:** Updated profile details.
- **Error Responses:**
  - **400 Bad Request:** If the profile name is already taken.

Update Profile Bio

10.)Endpoint: PATCH /profile/me/update/bio

- **Description:** Allows the user to update their bio. This endpoint is specifically for updating the bio section of the user's profile.
- **Form Data:**
  - bio (str): Required. The new bio text.
- **Response:**
  - **200 OK:** Confirmation of the bio update.
    - message (str): Confirmation message of bio update.

Update User Name

11.)Endpoint: PATCH /profile/me/update/name

- **Description:** Allows the user to update their profile name. This endpoint is specifically for changing the display name of the user.
- **Form Data:**
  - name (str): Required. The new profile name.
- **Response:**
  - **200 OK:** Confirmation of the name update.
    - message (str): Confirmation message of name update.

Update Profile Privacy

12.)Endpoint: PATCH /profile/me/update/is_private

- **Description:** Allows the user to update their profile privacy settings. This determines whether the user's profile is public or private.
- **Form Data:**
  - private_or_not (bool): Required. The new privacy status.
- **Response:**
  - **200 OK:** Confirmation of the privacy status update.
    - message (str): Confirmation message of privacy status update.

Update Profile Image

13.)Endpoint: PATCH /profile/me/update/profile_image

- **Description:** Allows the user to update their profile image. This endpoint is specifically for changing the user's profile picture.
- **Form Data:**
  - image (UploadFile): Required. The new profile image.
- **Response:**
  - **200 OK:** Confirmation of the image update.
    - message (str): Confirmation message of image update.

Delete Profile

14.)Endpoint: DELETE /profile/me/delete

- **Description:** Deactivates the user's profile by changing their active status. This effectively removes the user from the platform without permanently deleting their data.
- **Response:**
    - **200 OK:** Confirmation of profile deactivation.
        - message (str): Confirmation message of profile deactivation.
- **Error Responses:**
    - **400 Bad Request:** If there was an error while deactivating the profile.

Post Management

Show All Posts for Current User

15.)Endpoint: GET /posts/me

- **Description:** Retrieves all posts created by the currently logged-in user. Useful for displaying the user's own posts.
- **Response:**
    - **200 OK:** A list of all posts created by the user.

Create a Post (continued)

16.)Endpoint: POST/create_post

- **Response:**
    - **201 Created:** Confirmation of the new post creation.
        - post_id (int): The ID of the newly created post.
        - message (str): Confirmation message of post creation.
- **Error Responses:**
    - **400 Bad Request:** If required fields are missing or invalid data is provided.

Update a Post

17.)Endpoint: PUT /posts/{post_id}

- **Description:** Allows users to update an existing post by modifying its content or image. This endpoint can be used to change the description or the image attached to the post.
- **Path Parameter:**
  - post_id (int): Required. The ID of the post to update.
- **Request Body:**
  - image (UploadFile): Optional. A new image to replace the existing one.
  - description (str): Optional. Updated text content of the post.
- **Response:**
  - **200 OK:** Confirmation of the post update.
    - message (str): Confirmation message of post update.
- **Error Responses:**
  - **400 Bad Request:** If invalid data is provided or the post ID is not found.

Delete a Post

18.)Endpoint: DELETE /posts/{post_id}

- **Description:** Allows users to delete a specific post from their profile. This removes the post from the platform.
- **Path Parameter:**
  - post_id (int): Required. The ID of the post to delete.
- **Response:**
  - **200 OK:** Confirmation of post deletion.
    - message (str): Confirmation message of post deletion.
- **Error Responses:**
  - **400 Bad Request:** If the post ID is not found or if deletion fails.

Like a Post

19.)Endpoint: POST /posts/{post_id}/like

- **Description:** Allows users to like a specific post. This increments the like count for the post.
- **Path Parameter:**
  - post_id (int): Required. The ID of the post to like.
- **Response:**

    - **200 OK:** Confirmation of the post like.
        - message (str): Confirmation message of post like.
- **Error Responses:**
    - **400 Bad Request:** If the post ID is not found or if the post has already been liked by the user.

Comment on a Post

20.)Endpoint: POST /posts/{post_id}/comment

- **Description:** Allows users to comment on a specific post. Comments are added to the post, allowing for user interactions.
- **Path Parameter:**
    - post_id (int): Required. The ID of the post to comment on.
- **Request Body:**
    - comment (str): Required. The text of the comment.
- **Response:**
    - **201 Created:** Confirmation of the new comment creation.
        - comment_id (int): The ID of the newly created comment.
        - message (str): Confirmation message of comment creation.
- **Error Responses:**
    - **400 Bad Request:** If the comment text is missing or if the post ID is invalid.

Interaction Features

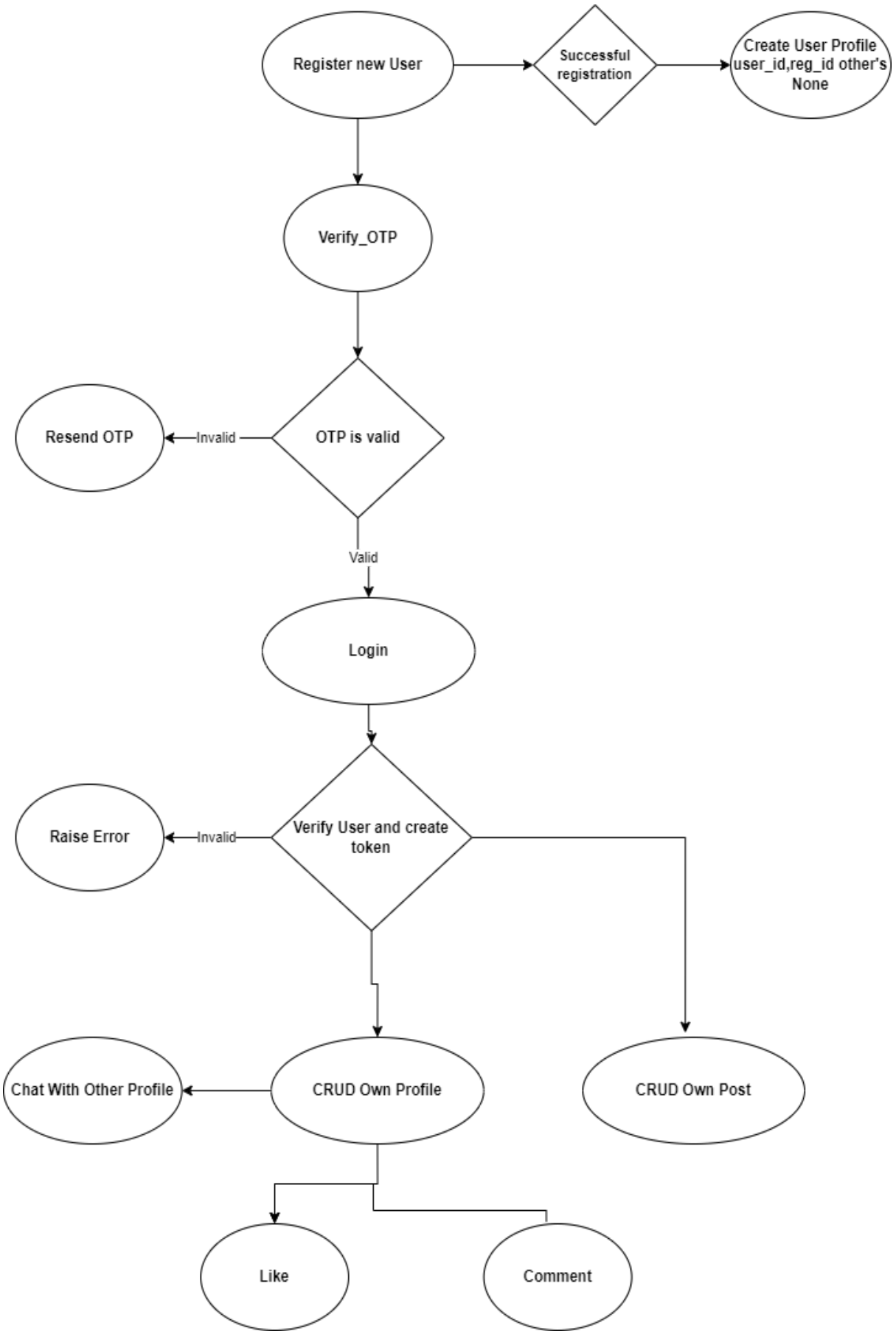
21.)List Recent Messages

- **Endpoint:** GET /chat/profile/
- **Description:** Lists recent messages for the current user, showing interactions with other profiles. This endpoint provides an overview of recent chats.
- **Response:**
    - **200 OK:** A list of recent messages.
        - Each message contains:
            - user_id (int): The ID of the other user.
            - username (str): The username of the other user.
            - last_message (str): The text of the last message.
            - timestamp (str): The time of the last message.
- **Error Responses:**
    - **400 Bad Request:** If there is a failure in retrieving the messages.

Send a Message

22.)Endpoint: POST /chat/{user_id}/

- **Description:** Sends a message to another user. This allows users to initiate or continue conversations.
- **Path Parameter:**
    - user_id (int): Required. The ID of the recipient user.
- **Request Body:**
    - message (str): Required. The text of the message to be sent.
- **Response:**
    - **200 OK:** Confirmation of message sent.
        - message (str): Confirmation message of the sent message.
- **Error Responses:**
    - **400 Bad Request:** If the recipient ID is invalid or the message content is not provided.


Register new User
Successful registration
Create User Profile user_id,reg_id other's None
Verify_OTP
OTP is valid
Invalid
Resend OTP
Valid
Login
Verify User and create token
Invalid
Raise Error
Chat With Other Profile
CRUD Own Profile
CRUD Own Post
Like
Comment

API solves several key problems related to user management, profile handling, content creation, and interaction within a social media platform.

1. User Registration and Authentication

- **Problem:** Ensuring secure user registration and login while protecting user data.
- **Solution:**
    - The API handles user registration securely, requiring a unique email and password. An OTP verification system ensures that only valid users can create accounts, adding an extra layer of security.
    - Password reset functionality via OTP allows users to regain access to their accounts without compromising security.

2. Account Recovery

- **Problem:** Users may forget their passwords, and without a recovery mechanism, they could lose access to their accounts.
- **Solution:**
    - The **Forgot Password** and **Reset Password** endpoints allow users to securely reset their password using OTP verification, minimizing frustration and support overhead.
    - The API regenerates OTPs when needed, ensuring users can recover their accounts even if they miss the initial OTP.

3. Profile Management

- **Problem:** Users need the ability to manage their profiles and update details (bio, image, privacy settings) to reflect changes in their preferences or information.
- **Solution:**
    - The API offers a range of endpoints for users to update their profile name, bio, image, and privacy settings, ensuring that profiles can be customized easily.
    - Users can also deactivate their profiles if they wish, giving them full control over their data and presence on the platform.

4. Content Creation and Sharing

- **Problem:** Users want to share content (posts with text and images) on the platform, but need a streamlined way to do this while maintaining control over their content.
- **Solution:**
    - The **Create Post**, **Update Post**, and **Delete Post** endpoints allow users to easily manage their posts. They can share images and

descriptions, update them if needed, and delete content that they no longer want to be visible.
    - This empowers users to maintain their digital presence while having control over what they share.

5. User Interaction

- **Problem:** Users want to interact with posts by liking, commenting, and messaging, but need a structured way to engage with other users' content.
- **Solution:**
    - The API allows users to like posts, comment on posts, and chat with other users, fostering a community atmosphere. This interactivity helps build engagement and connections between users.
    - The messaging system is particularly useful, allowing users to communicate privately outside of public post comments.

6. Privacy Control

- **Problem:** Some users prefer their profiles to be private, limiting who can view their content.
- **Solution:**
    - The API gives users the option to set their profile to private, ensuring that only approved users can view their profile or posts, which provides a sense of security and privacy.

7. Profile Discovery

- **Problem:** Users need an easy way to discover other users, especially when they want to connect with people with similar interests or friends.
- **Solution:**
    - The API allows users to search for profiles by username, helping them find and connect with other users easily. This solves the problem of profile discovery in large user bases.

8. User Retention

- **Problem:** Ensuring users return to the platform and continue interacting with it can be challenging.
- **Solution:**
    - By providing features like profile customization, content creation, and interaction through likes, comments, and messaging, API helps drive user retention, keeping users engaged and invested in the platform.

9. Data Security and Validation

- **Problem:** Ensuring user data security and preventing unauthorised access or manipulation is crucial for trust in the platform.
- **Solution:**
    - The API uses OTPs and secret keys for verification during sensitive actions like password reset and account registration, ensuring only authorised users can access their accounts or make critical changes.