UMASSGRAM

Social Media Platform for Umass Students

Author : Nitish Bhattad
Email : nbhattad@umassd.edu

1. Abstract:

UmassGram is a full-stack social media web application tailored for the University of Massachusetts community. The platform enables students with official university email addresses to register, post images, like and comment on others' posts, follow peers, and provide anonymous feedback. Designed with security and usability in mind, it features a clean user interface, notification system, and self-profile dashboards. Built using Flask (Python), MySQL, and HTML/CSS, UmassGram offers a practical implementation of authentication, database design, routing, and dynamic frontend-backend interaction.

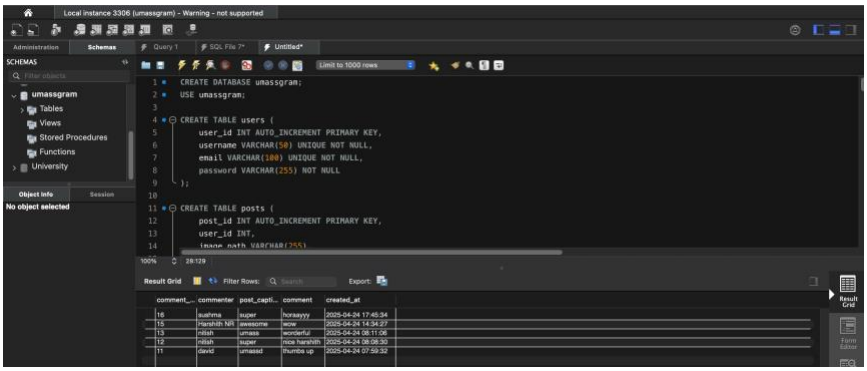
2. The main goals of UmassGram were:

- To build a **university-exclusive** social media platform for UMass students.
- To enable users to **register and login** securely using their university email address.
- To allow users to **upload posts** with captions and images.
- To implement a **like**, **comment**, **follow**, and **save** functionality to increase interactivity.
- To create a **self-profile** page displaying user's own posts, followers, and following count.
- To add an **anonymous feedback** feature for honest interactions.
- To implement a **notifications system** for likes, follows, and comments.
- To ensure clean **database design** and **secure password handling** (with hashing).
- To build a **responsive** and **visually appealing UI** using HTML/CSS with a university-themed background.
- To maintain data integrity with proper **deletion** of posts, likes, comments, and related media.

3. Tools & Technologies Used

- **Frontend**:
  - HTML5
  - CSS3
  - Bootstrap (for minimal responsive design)
  - JavaScript (for interactivity like toggling comments)
- **Backend**:
  - **Flask** (Python Web Framework)
  - Flask-Login (Session Management & Authentication)
  - Flask-WTF (Forms Handling)
  - Werkzeug (Password Hashing & Security)
- **Database**:
  - **MySQL** (Relational Database Management System)
  - **Database Engine:**
    - **InnoDB**
      (Provides transactions, foreign key support, and high reliability)
- **ORM / Query Handling**:
  - MySQL Connector / Manual SQL Queries (direct cursor operations)
- **Templating**:
  - Jinja2 (for dynamic HTML generation)
- **Version Control**:
  - Git (to track project changes)
- **Deployment/Hosting**:
  - Localhost Flask Development Server (for testing and running locally)

Database:



4. System Design and Database Schema

Entity Relationship (ER) Diagram

At the core of UmassGram, the database is structured around the following main entities:

Database_Table_Overview_for_UmassGram

Table Name	Primary Fields
users	user_id (PK), username, email, password
posts	post_id (PK), user_id (FK), image_path, caption, created_at
likes	like_id (PK), user_id (FK), post_id (FK)
comments	comment_id (PK), user_id (FK), post_id (FK), content, created_at
followers	follower_id (FK), following_id (FK)
saved_posts	saved_id (PK), user_id (FK), post_id (FK)
feedback	feedback_id (PK), post_id (FK), sender_id (FK), content, created_at
notifications	notification_id (PK), recipient_id (FK), sender_id (FK), post_id (FK), type, created_at

ER Diagram Relationships

- **users** (1) → (many) **posts**
- **users** (1) → (many) **likes**
- **users** (1) → (many) **comments**
- **users** (1) → (many) **followers** (self-join)
- **users** (1) → (many) **saved_posts**
- **users** (1) → (many) **feedback**
- **users** (1) → (many) **notifications**
- **posts** (1) → (many) **likes**
- **posts** (1) → (many) **comments**
- **posts** (1) → (many) **feedback**
- **posts** (1) → (many) **saved_posts**
- **posts** (1) → (many) **notifications**

Special Database Configurations

- **Database Engine:** InnoDB (Supports transactions, foreign keys, and high reliability)
- **Foreign Key Constraints:** Enforced for referential integrity.
- **Primary Keys:** Auto-Increment enabled for user_id, post_id, like_id, etc.
- **Timestamps:** Automatic creation timestamps on posts, comments, feedback, notifications.

5. Implementation Overview

Step-by-Step Application Flow

1. User Registration

- New users register using their university email ID (@umass.edu format enforced).
- Registration includes username, email, and password (stored securely after hashing).

2. User Login

- Registered users can log in using their username and password.
- Email validation ensures only UMass users access the platform.
- Passwords are verified using hashed password checking.

3. Feed Page

- After login, users are redirected to the Feed.
- The feed shows all posts from users, ordered by latest first.
- Each post displays:
- - Username (clickable to profile)
- - Image
- - Caption
- - Posted date
- - Like button ❤️ / 🤍 (toggle)
- - Comment toggle 💬
- - Save/unsave ⭐ button
- - Delete option (only for post owners)
- - Anonymous feedback section 👻 (only visible to the owner)

4. Post Upload

- Users can upload an image with a caption.
- Uploaded images are saved locally inside the /static/uploads folder.
- Post metadata (user_id, caption, image_path) is stored in the posts table.

5. Likes, Comments, Saves

- Users can:
- - Like/unlike posts.
- - Comment on posts.
- - Save posts to a personal saved list.
- Actions are stored in likes, comments, and saved_posts tables respectively.

6. Explore Page

- Random posts from different users are displayed.
- Users can explore content uploaded across the platform.

7. Saved Posts Page

- Users can view all posts they have saved/bookmarked.

8. Self Profile Page

- Displays logged-in user's:
- - Username
- - Email
- - User ID
- - Followers and Following count
- - All posts uploaded by the user

9. Other User Profiles

- Clicking on a username leads to the public profile page showing:
- - Their posts
- - Their followers
- - Their following list

10. Notifications

- System generates notifications for:
- - Likes on a user's post
- - Comments on a user's post
- - New followers

11. Anonymous Feedback

- Users can send secret comments to other posts.
- Only the post owner can view the secret feedback.

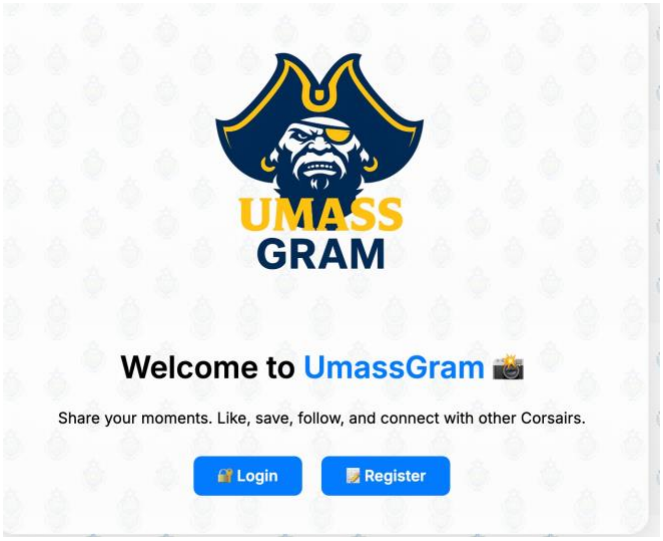
12. Post Deletion

- Users can delete their own posts.
- Deletion removes:
- - Post record from the database
- - Related likes, comments, saves
- - The actual image file from server storage

6. Key Features of UmassGram

User Authentication

- Only users with a valid **@umass.edu** email address can register and login.
- Secure password handling using **bcrypt hashing**.
- Session-based login/logout using **Flask-Login**.

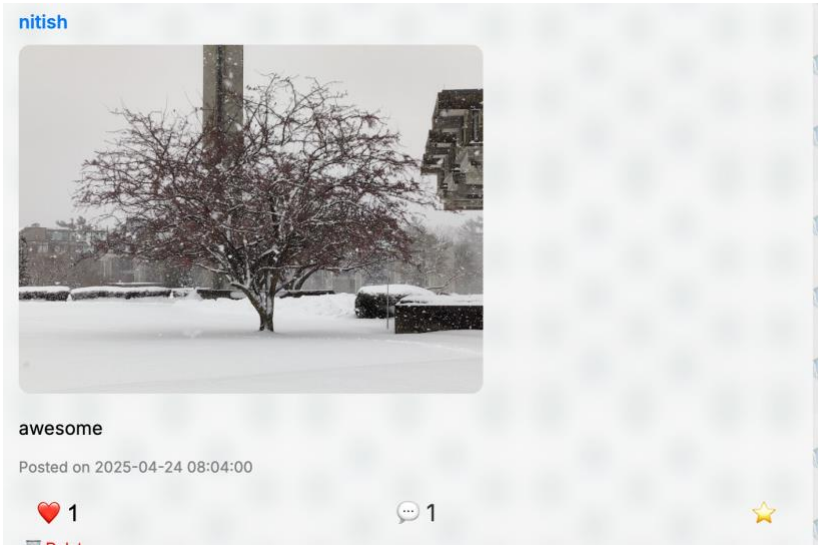


Post Management

- Upload images with captions.
- Posts are displayed in the **Feed** sorted by newest first.
- Each post includes:
  - Image
  - Caption
  - Username (clickable to profile)
  - Timestamp

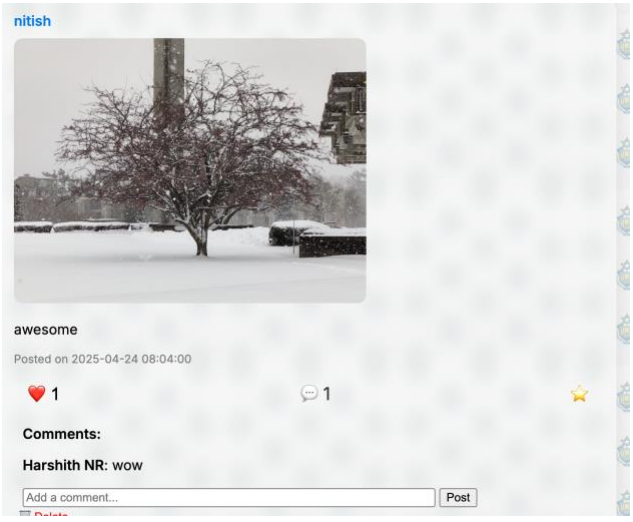
Like and Unlike

- Users can **like** and **unlike** any post.
- Like counts are updated in real-time.



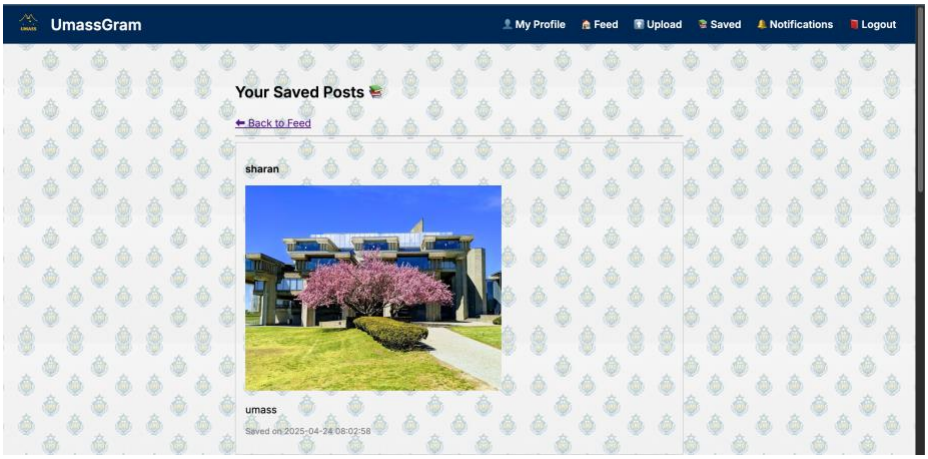
Comments

- Users can **comment** on any post.
- Comment counts are displayed.
- Comments can be toggled to view/hide below each post.



Save Posts

- Users can **save** favorite posts.
- Saved posts can be accessed through a separate **Saved Posts** page.

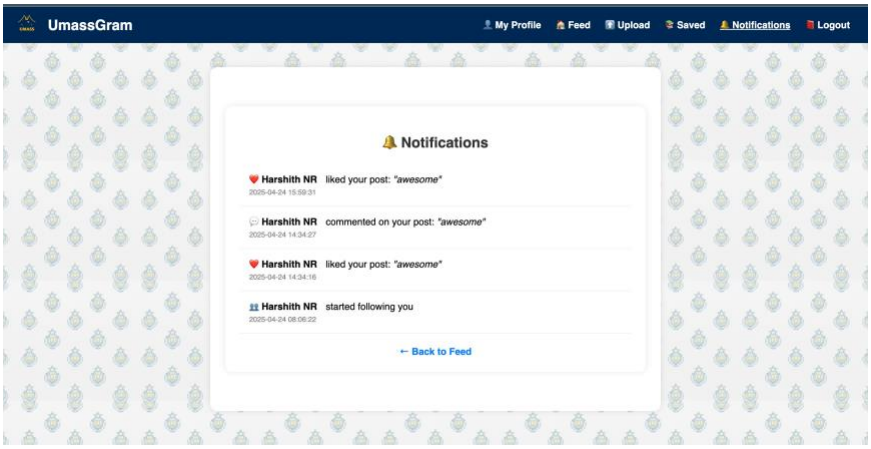


Follow System

- Users can **follow** or **unfollow** other users.
- **Followers** and **Following** counts are shown on profiles.

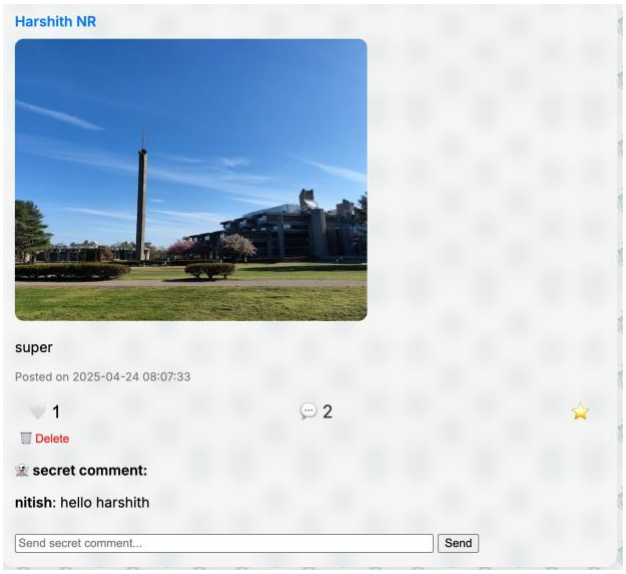
Notifications

- Users receive notifications when:
  - Someone likes their post.
  - Someone comments on their post.
  - Someone follows them.



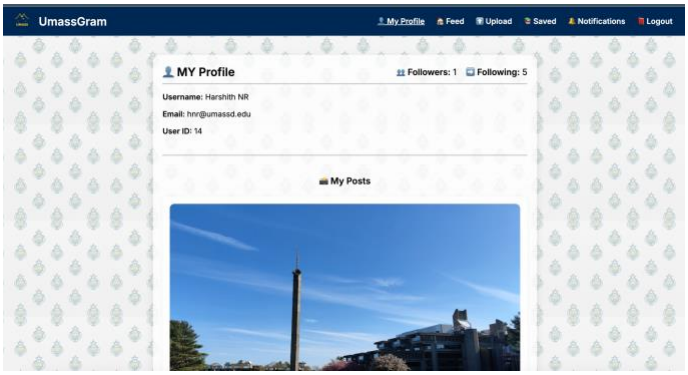
Anonymous Feedback

- Any user can send **anonymous feedback** to any post.
- Only the **post owner** can view the feedback privately.



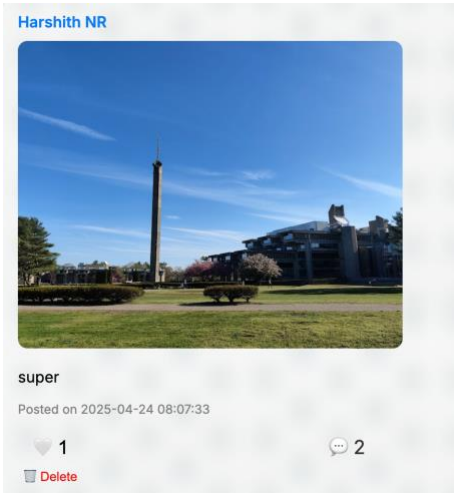
Self Profile Page

- Logged-in users can view their:
  - Username
  - Email
  - User ID
  - Number of followers
  - Number of following
  - All posts uploaded by them



Post Deletion

- Users can delete their own posts.
- Deletion removes:
  - The post
  - Associated likes, comments, saves



7. Testing and Results

Testing Approach

To ensure UmassGram was working properly, the following testing steps were carried out:

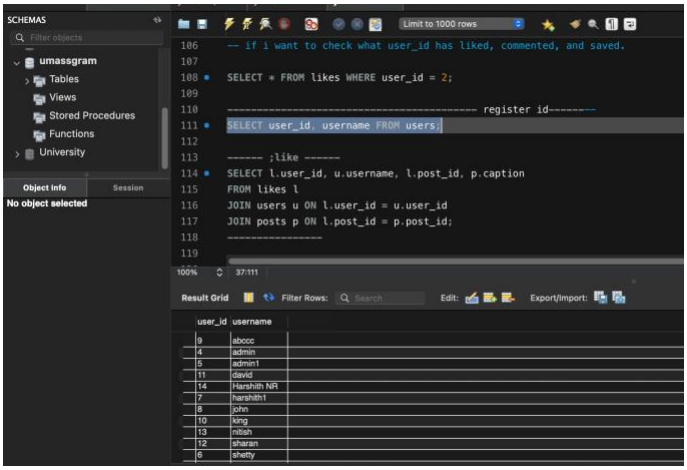
Test Case	Expected Result	Actual Result
Register with a valid @umass.edu email	User successfully registers	✅ Success
Register with invalid email (e.g., Gmail)	Registration blocked with error message	✅ Success
Login with correct credentials	User logged in and redirected to feed	✅ Success
Login with wrong password	Flash message: Invalid username or password	✅ Success
Upload a post (image + caption)	Post appears in feed	✅ Success
Like and unlike a post	Like button toggles and counter updates	✅ Success
Comment on a post	Comment appears under the post	✅ Success
Save a post	Post appears in Saved Posts page	✅ Success
Explore page shows random posts	Posts are randomized each refresh	✅ Success
Follow and unfollow another user	Follow button toggles and follower count updates	✅ Success
Notifications when liked/commented/followed	Notification appears in the Notifications page	✅ Success
Anonymous feedback submitted	Feedback visible only to the post owner	✅ Success
Delete own post	Post deleted along with image and related data	✅ Success
View Self Profile	User info (username, email, ID, posts) displayed	✅ Success

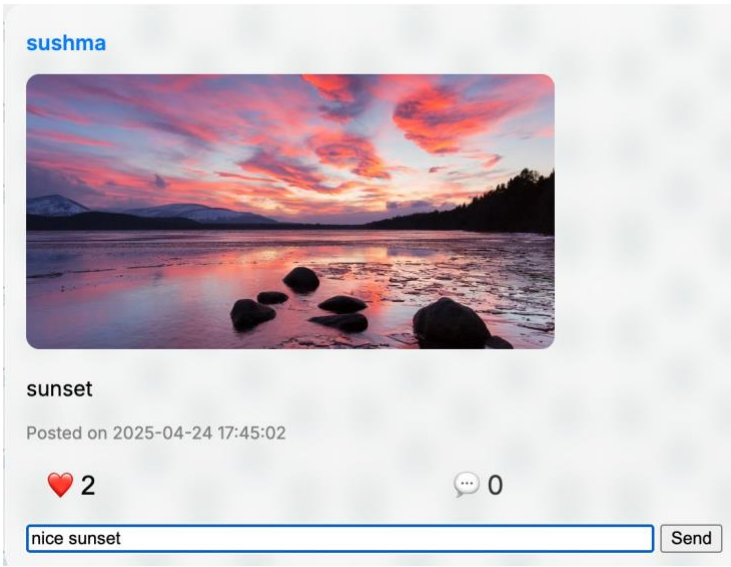
Testing Tools Used

- **Manual Testing:** Tested all major functionalities through browser interactions (Chrome, Edge).
- **Database Validation:** Checked MySQL tables after each operation:
  - New users inserted correctly.
  - Posts, likes, comments, saved posts, and feedback properly recorded.
- **Server Monitoring:** Observed Flask development server logs for errors and warnings during testing.

Results

- All major features worked **successfully without critical bugs**.
- **Error messages** displayed properly when needed (e.g., invalid login, invalid email).
- **UI responsiveness** tested on different devices (desktop, laptop).
- **Database integrity** maintained during delete operations (e.g., deleting a post also deleted related likes, comments, saves, and image file).





8. Challenges Faced

- Managing **Image Uploads and Deletions:**
  Ensured that when posts are deleted, the related image file is also safely removed from the server.
- Handling **Notifications System:**
  Structured database and logic carefully to trigger accurate notifications for likes, comments, and follows.
- Implementing **Anonymous Feedback:**
  Allowed feedback to be visible only to post owners, while keeping sender identity hidden from other users.
- Designing **a Clean and Attractive Frontend:**
  Used base templates, CSS styling, and background effects to improve the visual appeal and consistency of all pages.

9. Conclusion

UmassGram successfully recreates a mini social media experience tailored for the UMass community.
It includes core features like:

- User registration/login with UMass email validation.
- Post uploads with captions.
- Like, comment, save functionalities.
- Anonymous feedback system.
- Notifications for likes, comments, and follows.
- Followers and Following system.
- Personal profiles and post management.
- A clean and responsive user interface.

This project demonstrated full-stack development skills involving **Flask**, **MySQL**, **HTML/CSS**, and **secure authentication systems**.

10. Future Scope

- **Add Stories Feature:**
  Similar to Instagram, allow users to post disappearing "stories" (24-hour lifespan).
- **Chat Messaging System:**
  Implement direct user-to-user messaging inside the platform.
- **Profile Picture Uploads:**
  Let users upload and display their profile pictures.
- **Admin Panel:**
  Create a backend admin panel to manage users, posts, and reported content.
- **Post Editing:**
  Allow users to edit their captions after posting.
- **Like Animations and Real-Time Updates:**
  Use WebSocket technologies (e.g., Flask-SocketIO) to make likes/comments update in real-time without page refresh.
- **Mobile-Friendly App:**
  Build a PWA (Progressive Web App) or Flutter-based mobile version of UmassGram.

11. Important Code Snippets

1. User Registration Route

```python
@main.route("/register", methods=["GET", "POST"])
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        email = form.email.data.strip().lower()

        if not email.endswith("@umassd.edu"):
            flash("Only UMass email addresses are allowed to register.", "danger")
            return redirect(url_for("main.register"))

        # Check if username already exists
        cur = mysql.connection.cursor()
        cur.execute("SELECT * FROM users WHERE username = %s", (form.username.data,))
        if cur.fetchone():
            flash("Username already exists", "danger")
            return redirect(url_for("main.register"))

        hashed_pw = generate_password_hash(form.password.data)
        cur.execute("INSERT INTO users (username, email, password) VALUES (%s, %s, %s)",
                    (form.username.data, email, hashed_pw))
        mysql.connection.commit()
        cur.close()
        flash("Registration successful. Please log in.", "success")
        return redirect(url_for("main.login"))
    return render_template("register.html", form=form)
```

This route handles new user registration, validates UMass email, hashes passwords securely, and stores user data in the database.

2. User Login Route

```python
@main.route("/login", methods=["GET", "POST"])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        cur = mysql.connection.cursor()
        cur.execute("SELECT * FROM users WHERE username = %s", (form.username.data,))
        user = cur.fetchone()
        cur.close()

        if user and check_password_hash(user['password'], form.password.data):
            login_user(User(user['user_id'], user['username'], user['email'], user['password']))
            return redirect(url_for('main.feed'))
        else:
            flash("Invalid username or password", "danger")
    return render_template("login.html", form=form)


@main.route("/logout")
```

This route checks login credentials, verifies password hashes, and securely logs the user into the session.

3. Post Upload Route

```python
@main.route("/upload", methods=["GET", "POST"])
@login_required
def upload():
    form = UploadForm()
    if form.validate_on_submit():
        image = form.image.data
        if image is None or image.filename == "":
            flash("Please select an image to upload.", "danger")
            return redirect(url_for('main.upload'))

        filename = secure_filename(str(uuid.uuid4()) + "_" + image.filename)
        upload_folder = os.path.join('app', 'static', 'uploads')
        os.makedirs(upload_folder, exist_ok=True)
        upload_path = os.path.join(upload_folder, filename)
        image.save(upload_path)

        cur = mysql.connection.cursor()
        cur.execute("INSERT INTO posts (user_id, image_path, caption) VALUES (%s, %s, %s)",
                    (current_user.id, filename, form.caption.data))
        mysql.connection.commit()
        cur.close()
        flash("Post uploaded!", "success")
        return redirect(url_for('main.feed'))
    return render_template("upload.html", form=form)
```

This route lets users upload an image with a caption, saves the file to the server, and stores metadata in the database.

4. Like and Unlike a Post

```python
@main.route("/like/<int:post_id>", methods=["POST"])
@login_required
def like(post_id):
    cur = mysql.connection.cursor()

    cur.execute("SELECT * FROM likes WHERE user_id = %s AND post_id = %s", (current_user.id, post_id))
    existing_like = cur.fetchone()

    if existing_like:
        # Unlike
        cur.execute("DELETE FROM likes WHERE user_id = %s AND post_id = %s", (current_user.id, post_id))
    else:
        # Like
        cur.execute("INSERT INTO likes (user_id, post_id) VALUES (%s, %s)", (current_user.id, post_id))

        #  Notification of likes
        cur.execute("SELECT user_id FROM posts WHERE post_id = %s", (post_id,))
        owner = cur.fetchone()

        if owner and owner['user_id'] != current_user.id:
            cur.execute("""
                INSERT INTO notifications (recipient_id, sender_id, post_id, type)
                VALUES (%s, %s, %s, 'like')
            """, (owner['user_id'], current_user.id, post_id))

    mysql.connection.commit()
    cur.close()
    return redirect(url_for('main.feed'))
```

This route allows users to toggle (like/unlike) a post by inserting or deleting from the likes table.

5. Save or Unsave a Post

```python
@main.route('/save/<int:post_id>', methods=['POST'])
@login_required
def save_post(post_id):
    cur = mysql.connection.cursor()
    cur.execute("SELECT * FROM saved_posts WHERE user_id = %s AND post_id = %s", (current_user.id, post_id))
    existing = cur.fetchone()

    if existing:
        cur.execute("DELETE FROM saved_posts WHERE user_id = %s AND post_id = %s", (current_user.id, post_id))
    else:
        cur.execute("INSERT INTO saved_posts (user_id, post_id) VALUES (%s, %s)", (current_user.id, post_id))

    mysql.connection.commit()
    cur.close()
    return redirect(url_for('main.feed'))
@main.route("/explore")
@login_required
def explore():
    cur = mysql.connection.cursor()
    cur.execute("""
        SELECT posts.*, users.username FROM posts
        JOIN users ON posts.user_id = users.user_id
        ORDER BY RAND()
    """)
    posts = cur.fetchall()
    cur.close()
    return render_template("explore.html", posts=posts)

@main.route('/saved')
@login_required
def saved_posts():
    cur = mysql.connection.cursor()

    cur.execute("""
        SELECT posts.*, users.username
        FROM saved_posts
        JOIN posts ON saved_posts.post_id = posts.post_id
        JOIN users ON posts.user_id = users.user_id
        WHERE saved_posts.user_id = %s
        ORDER BY saved_posts.created_at DESC
    """, (current_user.id,))
    posts = cur.fetchall()
    cur.close()
```

Users can save or unsave posts to their personal collection stored in the saved_posts table.

6. Anonymous Feedback Submission

```python
@main.route('/feedback/<int:post_id>', methods=['POST'])
@login_required
def feedback(post_id):
    content = request.form.get('feedback')
    if content:
        cur = mysql.connection.cursor()
        cur.execute("""
            INSERT INTO feedback (post_id, sender_id, content)
            VALUES (%s, %s, %s)
        """, (post_id, current_user.id, content))
        mysql.connection.commit()
        cur.close()
        flash("Anonymous feedback sent!", "success")
    return redirect(url_for('main.feed'))
```

Allows users to send secret feedback/comments to a post that only the post owner can see.

7. Notification System for Likes, Comments, Follow

```python
@main.route("/notifications")
@login_required
def notifications():
    cur = mysql.connection.cursor()
    cur.execute("""
        SELECT n.*, u.username AS sender_name, p.caption
        FROM notifications n
        LEFT JOIN users u ON n.sender_id = u.user_id
        LEFT JOIN posts p ON n.post_id = p.post_id
        WHERE n.recipient_id = %s
        ORDER BY n.created_at DESC
    """, (current_user.id,))
    notes = cur.fetchall()
    cur.close()
    return render_template("notifications.html", notifications=notes)
```

When a user likes, comments, or follows, a notification entry is added to the notifications table for the recipient.

References

1. Flask Documentation. (n.d.). *Welcome to Flask Documentation (2.3.x)*. Retrieved from https://flask.palletsprojects.com
2. MySQL Documentation. (n.d.). *MySQL 8.0 Reference Manual*. Retrieved from https://dev.mysql.com/doc/
3. W3Schools. (n.d.). *HTML, CSS, and JavaScript Tutorials*. Retrieved from https://www.w3schools.com
4. MDN Web Docs. (n.d.). *Web technology for developers*. Retrieved from https://developer.mozilla.org
5. Bootstrap. (n.d.). *Bootstrap Documentation*. Retrieved from https://getbootstrap.com
6. Stack Overflow. (n.d.). *Community Discussions & Code Examples*. Retrieved from https://stackoverflow.com
7. Font Awesome. (n.d.). *Free Icons*. Retrieved from https://fontawesome.com