NARXOZ UNIVERSITY

EndTerm project
“Social Media App”

Students: Nuraiym Tanatkanova, Kaiypov Yerassyl
Supervisor: Omirali Aikumis

Almaty 2025

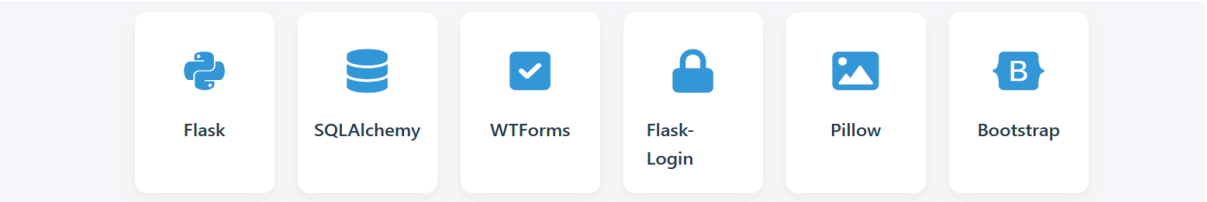
Project Requirements:

1. Project Description
   - 1.1. Develop a full-featured web application using Flask that includes:
   - 1.2. User registration and login system
   - 1.3. Data interaction through forms
   - 1.4. CRUD operations with database
   - 1.5. Session management
   - 1.6. File uploads
   - 1.7. Object-oriented architecture
   - 1.8. Use of templates for rendering pages
2. Functional Requirements
   - 2.1. Server Requests and Data Handling
   - 2.2. Implement GET and POST methods for all forms and data submissions.
   - 2.3. Use appropriate Flask routing and request handling (request.form, request.args, etc.).
3. Forms and Validation
   - 3.1. Use WTForms or manual form handling.
   - 3.2. Validate all input data.
   - 3.3. Handle errors gracefully with feedback to the user.
4. Cookies and Sessions
   - 4.1. Use Flask sessions to manage user login state.
   - 4.2. Store session information securely using a secret key.
5. Advanced Session Handling
   - 5.1. Implement session expiration and auto-logout.
   - 5.2. Optional: remember me functionality using cookies.
6. Database Integration
   - 6.1. Use SQLite or MySQL with SQLAlchemy (ORM).
   - 6.2. Design normalized tables with relationships (one-to-many, many-to-many).
   - 6.3. Handle database errors (e.g., duplicate usernames).
7. CRUD Operations
   - 7.1. Allow users to Create, Read, Update, Delete records.
     Example: blog posts, student records, products, etc.
   - 7.2. Use appropriate HTTP verbs (GET, POST, PUT, DELETE if possible).
8. User Authentication
   - 8.1. Implement secure user registration and login.
   - 8.2. Use sessions to track user state.
9. Advanced Database Techniques
   - 9.1. Use JOINs or relationships for querying related data.
   - 9.2. Provide a search feature using filter conditions.
10. File Uploads
    - 10.1. Allow users to upload images or documents.
    - 10.2. Securely store uploaded files in a static or uploads folder.
    - 10.3. Validate file type and size.
11. Object-Oriented Programming
    - 11.1. Structure the application using OOP principles.
    - 11.2. Create classes for forms, models, utilities, and controllers.
12. Refactoring to OOP
    - 12.1. Convert procedural code (if any) into class-based views or modular OOP design.
13. Modern Framework Usage
    - 13.1. Use Flask Blueprints for modular design.
    - 13.2. Use Jinja2 templates for rendering HTML.

Page numbers for direction:

Used functions:

- Python
- Bootstrap
- Flask
- WTForms
- SQLAlchemy
- Pillow
- HTML5 with Jinja
- Flask-Login



This report provides an overview of the full-stack Flask web application developed as an end-term project. The application demonstrates end-to-end implementation of a dynamic, interactive, secure, and data-driven web platform using Flask. Process performing in a website launched with Flask. By registering on the website, users are able to create posts, delete them, and manage the account.

In the files you might see functions like SQLite database, Jinja, WTForms, etc. So it is preferred to download requirements to work with the files effectively. You can find this download.txt in the project line, and it's better to install them in a virtual environment.

Objectives:

- Implement a user registration and login system.
- Handle data interaction through forms with proper validation. • Perform CRUD operations on database records.
- Manage user sessions securely with Flask sessions.
- Enable file uploads with image resizing and validation.
- Apply object-oriented programming principles and Blueprint modularity.
- Use Jinja2 templates for dynamic page rendering and a responsive UI.

Here is the introduction with these functions and their usage in the project:

Explanation of code structure (OOP & Blueprints)

Architecture

The application uses:

1. Flask Factory Pattern : The create_app() function in your code indicates that the application uses the factory pattern, which is a recommended approach for larger Flask applications.

2. Blueprints : The project is organized using Flask blueprints, which help modularize the application into logical components.

3. Object-Oriented Programming (OOP) : The application likely uses OOP principles with models representing database entities.

Main Components

Based on the file structure and the run.py file, your project likely includes:

- Models : Database models for users, posts, and possibly comments
- Routes/Views : Organized into blueprints for different sections (main, posts, users, etc.)
- Templates : Jinja2 templates for rendering HTML
- Forms : WTForms for handling form validation
- Authentication : User registration, login, and profile management

How It Works

The entry point run.py imports the application factory function and creates the Flask application instance. When run directly, it starts the development server with debugging enabled.

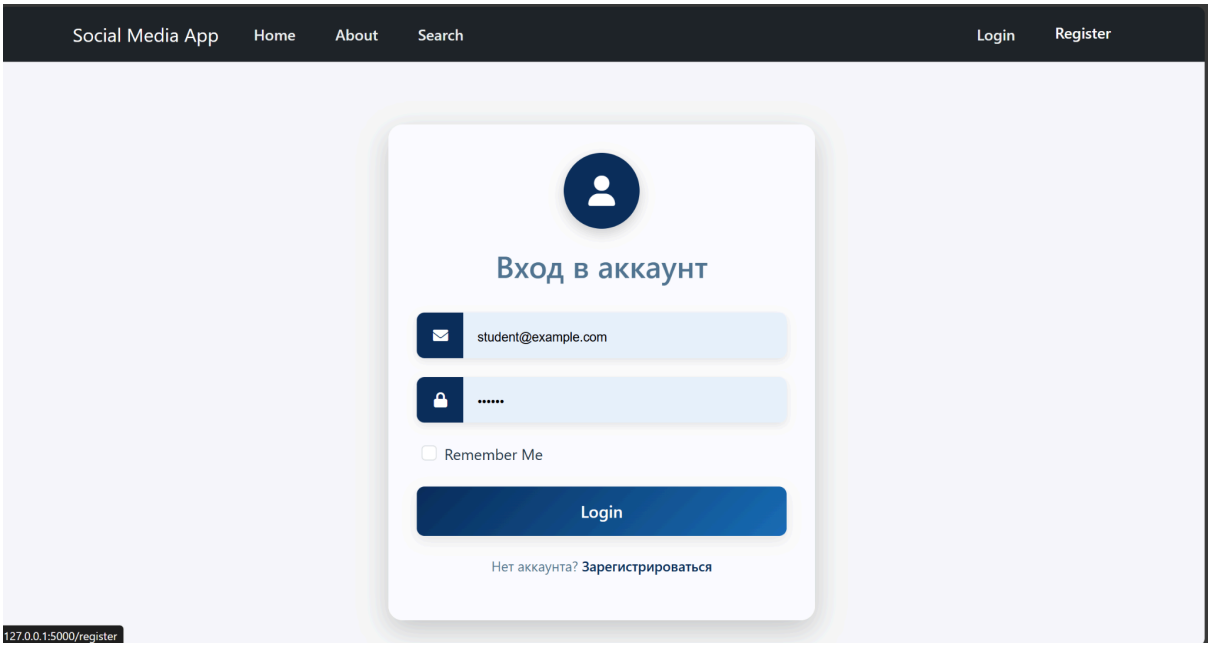
The application is likely structured to handle:

- User registration and authentication
- Creating, editing, and deleting blog posts
- Viewing posts by specific users
- Possibly commenting on posts
- Profile management

This structure follows Flask best practices by separating concerns, making the code more maintainable and testable.

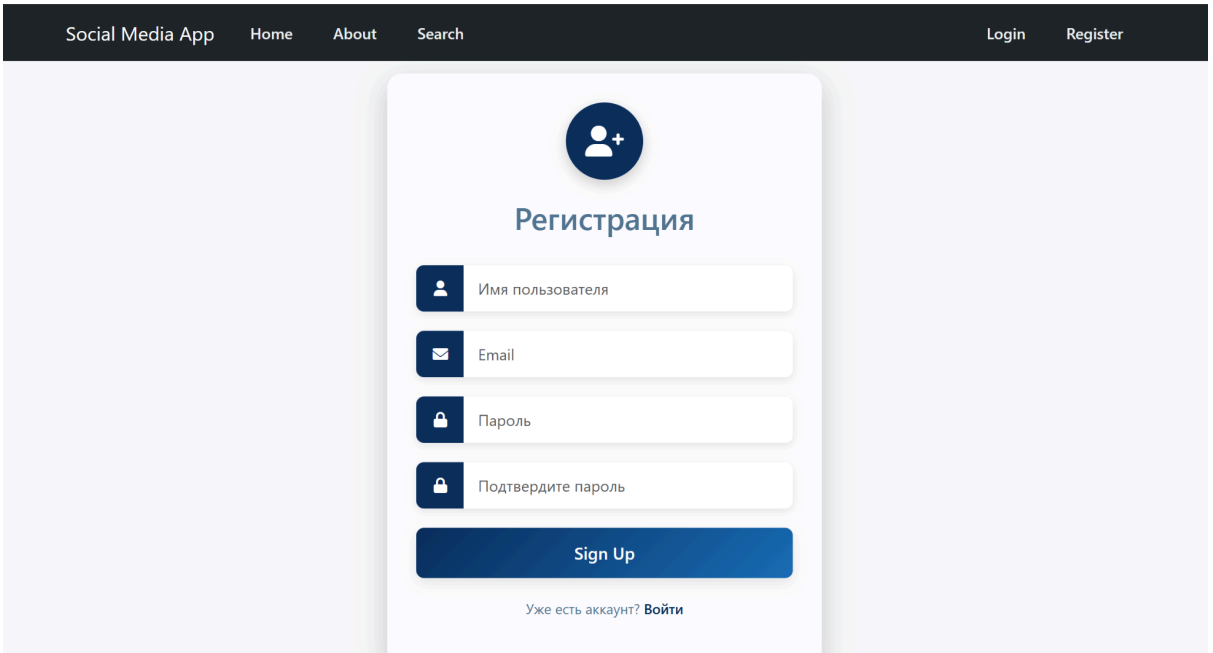
Screenshots of key pages

Authentication Pages



Login Page

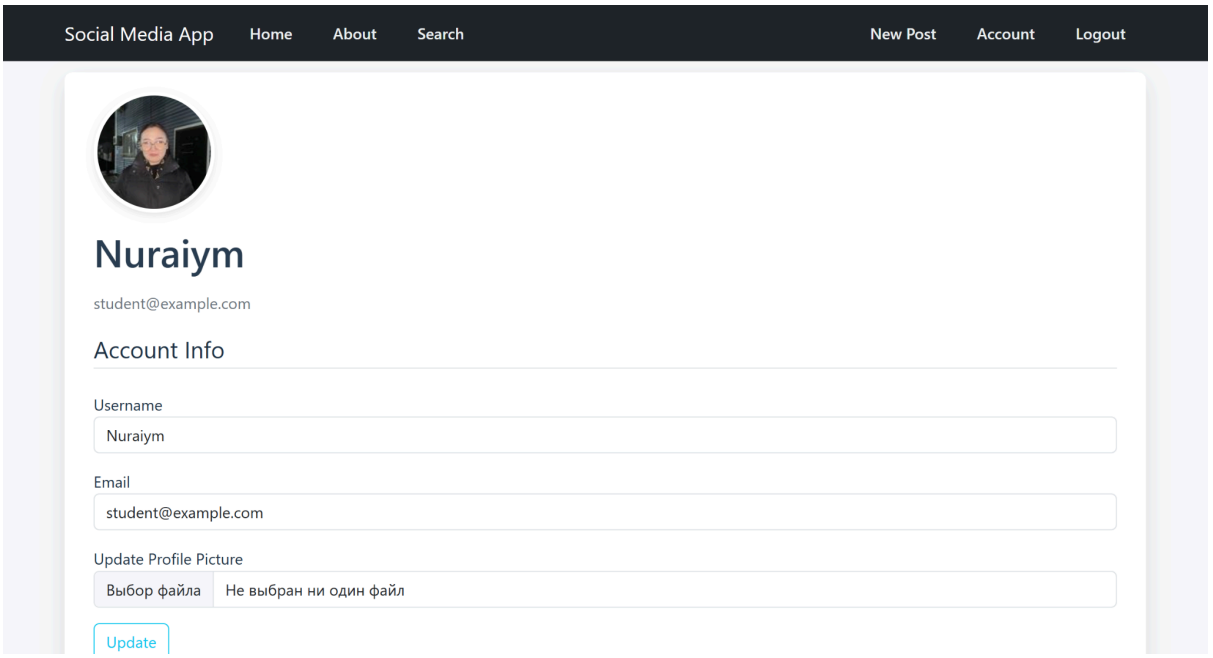
- Allows users to authenticate with username/email and password
- Likely includes "Remember Me" functionality
- Probably has password reset options
- Redirects authenticated users to their dashboard or home page



Registration Page

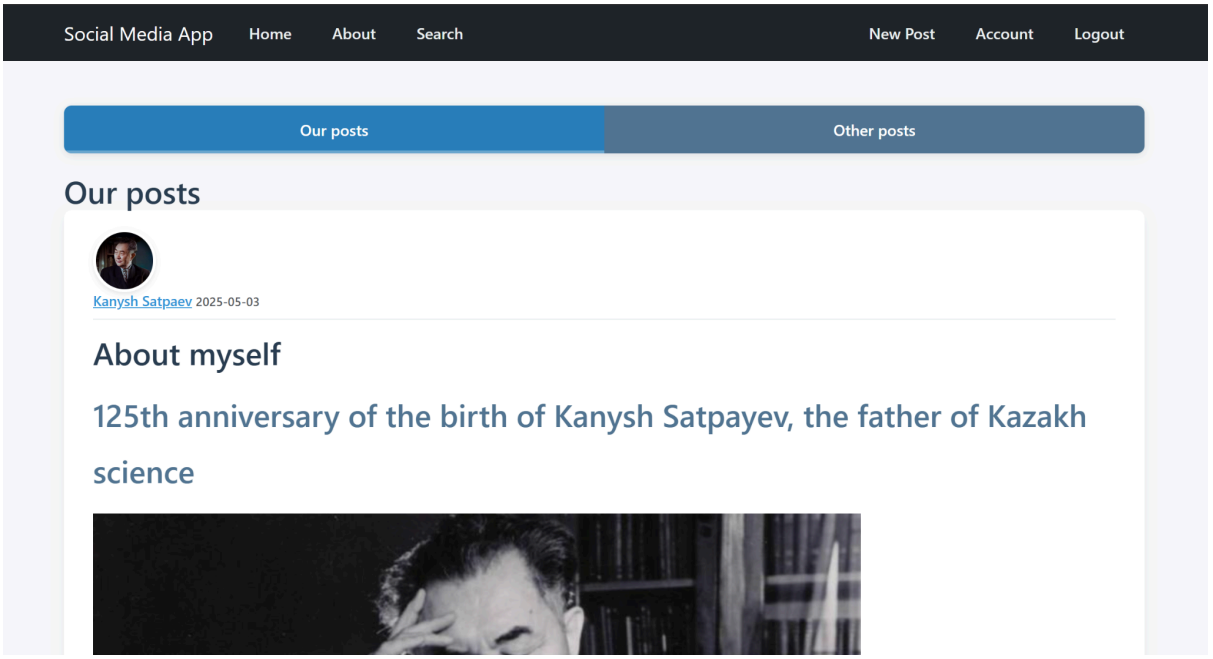
- Collects new user information (username, email, password)
- Performs validation (unique username/email, password strength)
- May include email verification
- Creates new user accounts in the database

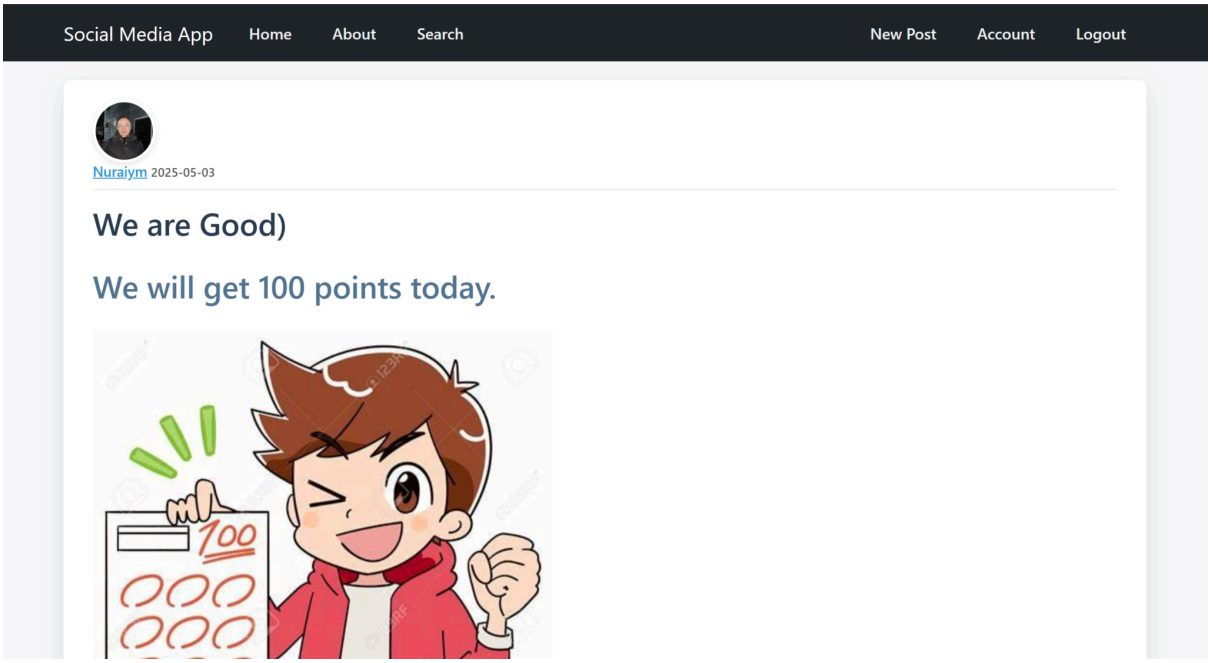
Profile user's page



- Displays user information (username, email, profile picture)
- Shows posts created by the user
- Allows users to update their profile information

Content Pages





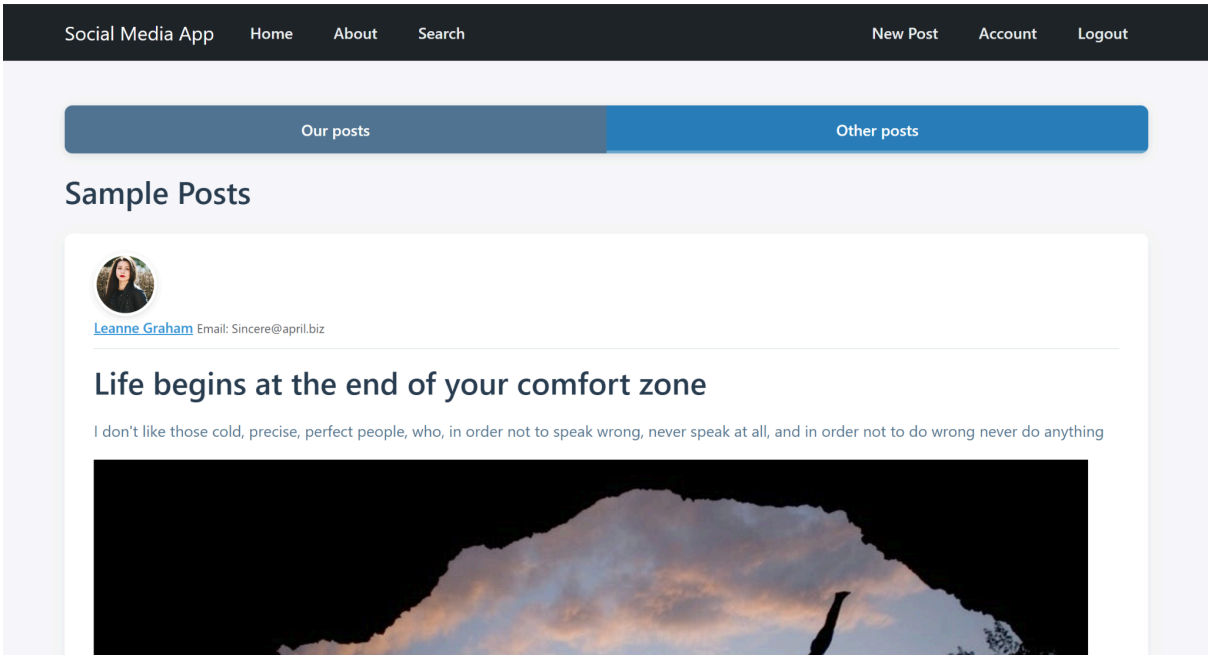
Our posts

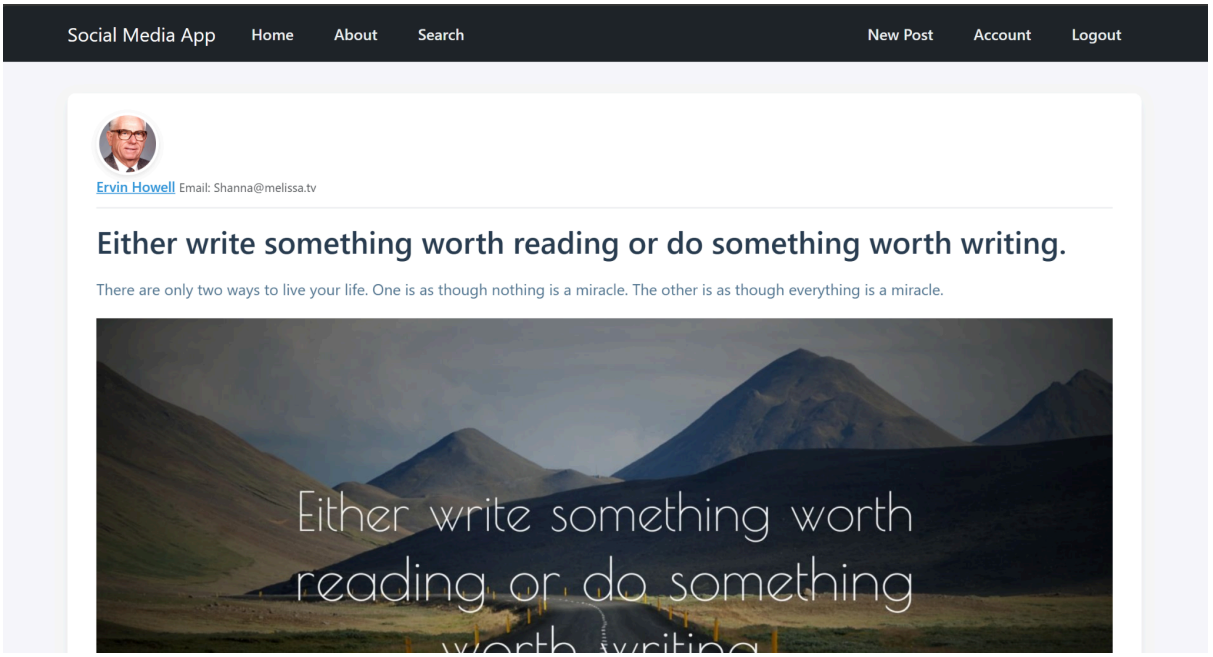
- Home page displays a paginated list of recent posts
- Individual post pages show full content with author information
- Post creation/editing forms for authenticated users
- Delete confirmation for post owners
- Possibly includes commenting functionality

User Posts

- Filtered view showing posts from a specific user
- Similar layout to the main posts page but limited to one author

Other Pages

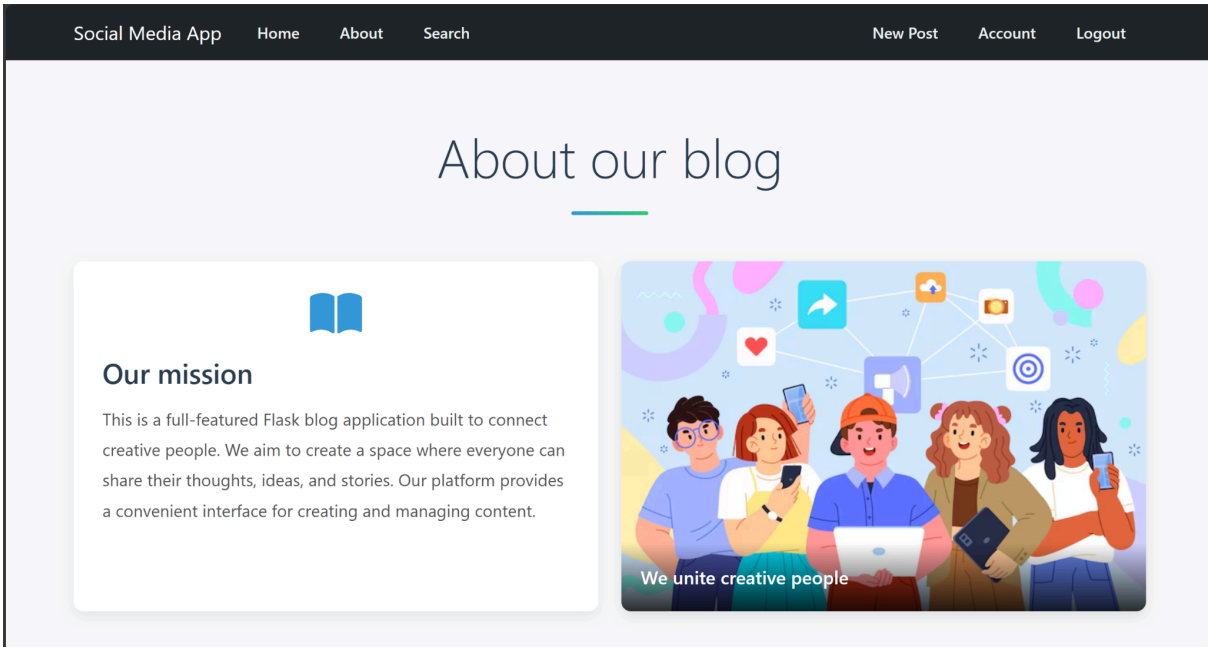

Social Media App
Home
About
Search
New Post
Account
Logout
Our posts
Other posts
Sample Posts
Leanne Graham Email: Sincere@april.biz
Life begins at the end of your comfort zone
I don't like those cold, precise, perfect people, who, in order not to speak wrong, never speak at all, and in order not to do wrong never do anything

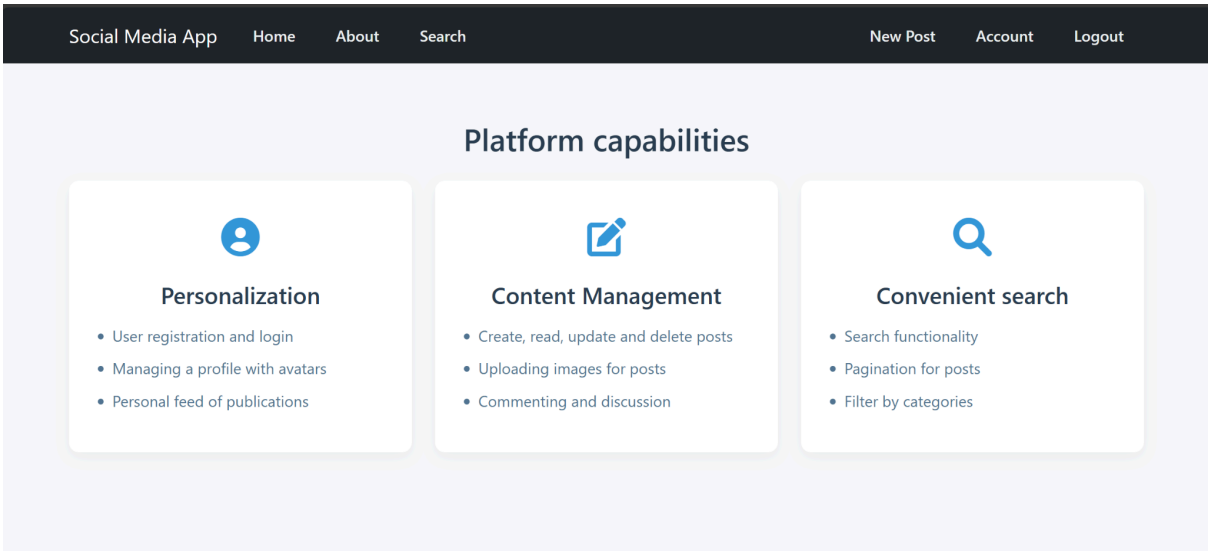

Social Media App
Home
About
Search
New Post
Account
Logout
Ervin Howell Email: Shanna@melissa.tv
Either write something worth reading or do something worth writing.
There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.
Either write something worth
reading or do something

About/Contact

- Static or semi-static pages with information about the site
- May include contact forms or other information
- The "Other posts" page that displays sample content from external sources

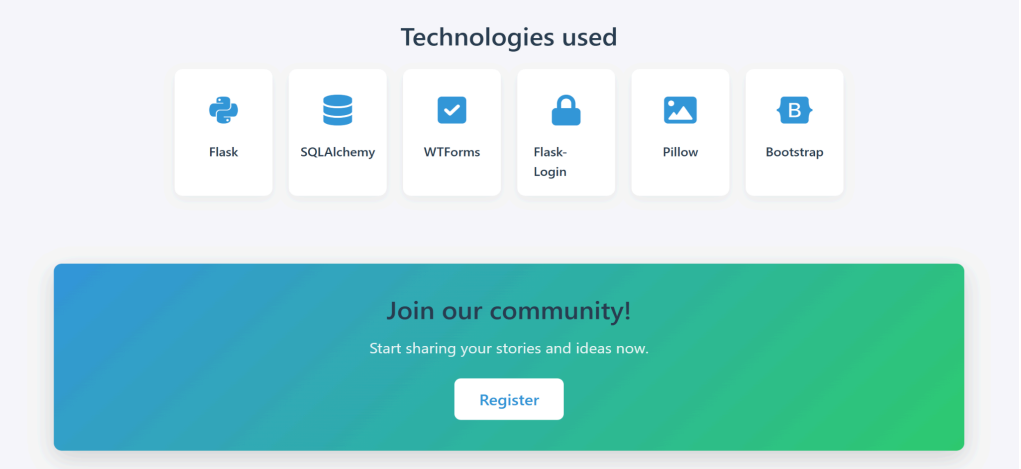
About Page





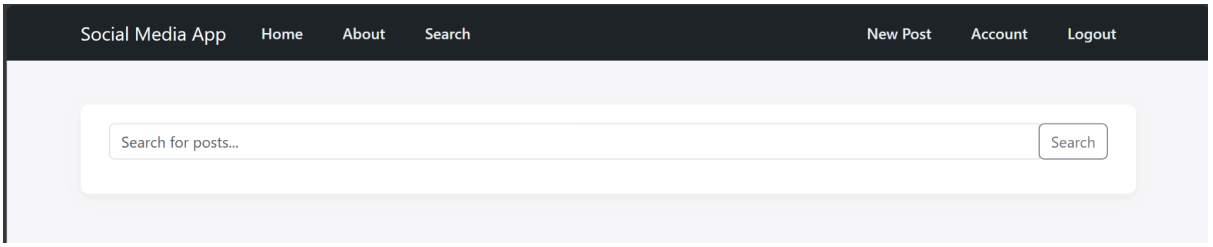
The About page likely provides information about your blog, its purpose, and possibly the creators or maintainers. It's typically a static page that:

-



Introduces visitors to the purpose of the blog
- May include mission statements or goals
- Could contain contact information or links to social media
- Might have team member information if it's a collaborative blog

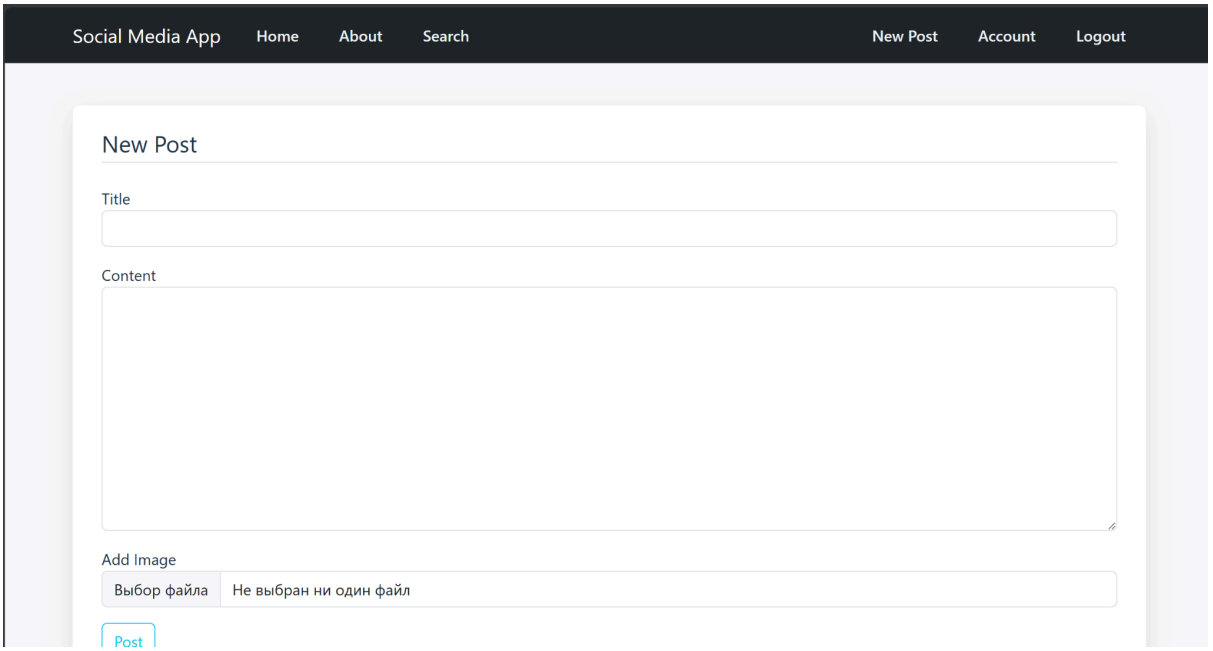
Search page



The Search page allows users to find specific content within your blog:

- Features a search form with query input
- Returns results matching the search criteria from post titles and content
- Likely displays results in a similar format to the main posts listing
- May include filtering options (by date, author, tags, etc.)
- Could highlight matching terms in the search results

Post New Page



The Post New page is where authenticated users can create new blog posts:

1. Accessible only to logged-in users
2. Contains a form with fields for:
3. Post title
4. Content (likely with a rich text editor)
5. Optional image upload
6. Includes validation to ensure required fields are completed
7. Has preview functionality (possibly)
8. Contains submit and cancel buttons
9. Redirects to the newly created post upon successful submission

The roles of the functions in our project:

Flask Blueprint

1) Purpose: Modularize your app into independent components (e.g. auth, posts, main).

2) Key Functions/Usage:

    a) Blueprint('name', __name__, url_prefix='/prefix') — create a blueprint.

    b) @blueprint.route('/path') — add routes to that blueprint.

    c) app.register_blueprint(blueprint) — hook it into your application in create_app().

    We used them in auth.py for the registration form and upload.py to handle files.

    Benefits: Keeps your code organized, makes it easy to split up large apps, and enables reusing or sharing modules.

Flask

1) Purpose: A lightweight web framework for handling requests, routing, sessions, etc.

2) Key Functions/Usage:

    a) Flask(__name__) — app instance.

    b) @app.route('/') — map URLs to view functions.

    c) app.run() — start the development server.

    d) request, session, flash, redirect, url_for — core helpers for I/O and flow control.

SQLAlchemy

1) Purpose: Object-Relational Mapper (ORM) to define Python classes (models) that map to database tables.

2) Key Functions/Usage:

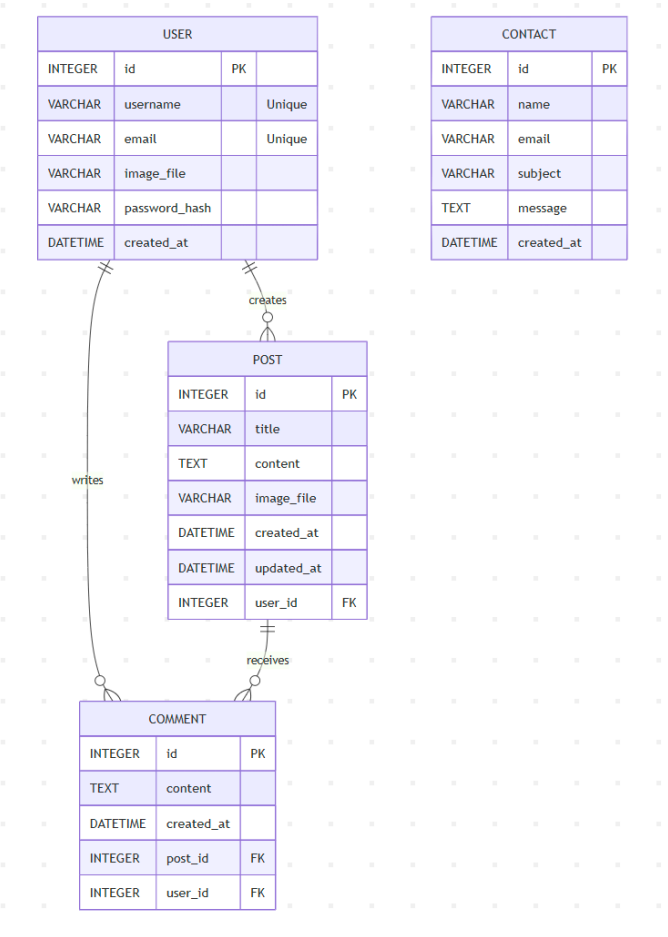
    a) db = SQLAlchemy(app) or db.init_app(app) in factories.

    b) Define models as class User(db.Model): ... with db.Column, db.relationship.

    c) db.create_all(), db.session.add(), db.session.commit() — create tables and persist objects.

    d) Querying via User.query.filter_by(...), .all(), .paginate().

The ER diagram of the database:



HTML5 + Jinja

1) Purpose: Jinja is Flask's templating engine; you write HTML5 interlaced with Jinja tags to render dynamic content.

2) Key Functions/Usage:
    a) {{ variable }} — interpolate Python data into HTML.

    b) {% for item in list %}…{% endfor %} — loops.

    c) {% if condition %}…{% endif %} — conditionals.

    d) Template inheritance: {% extends "layout.html" %} + {% block content %}.

Bootstrap

1) Purpose: CSS (and JS) framework for responsive, mobile-first UI components.

2) Key Functions/Usage:
    a) Add <link> and <script> for Bootstrap CDN in your layout.

    b) Use pre-built classes like .container, .row, .col-md-6, .btn, .navbar, etc.

    c) Utility classes (spacing, text-color, flex) allow rapid styling without custom CSS.

WTForms

- Purpose: Form-handling and validation library that integrates with Flask via Flask-WTF.

- Key Functions/Usage:

Define forms as Python classes inheriting from FlaskForm:

```
class LoginForm(FlaskForm):
  email = StringField('Email', validators=[DataRequired(), Email()])
  password = PasswordField('Password', validators=[DataRequired()])
  submit = SubmitField('Login')
```

-

In templates: {{ form.field(class="form-control") }} and display form.field.errors.

Pillow

1) Purpose: Python Imaging Library for opening, processing, and saving image files.

2) Key Functions/Usage:

    a) from PIL import Image.

    b) img = Image.open(file_stream) — load an uploaded image.

    c) img.thumbnail((width, height)) or img.resize((w,h)).

    d) img.save(path) — write the processed image to disk.

Flask-Login

1) Purpose: Simplify user session management and route protection.

2) Key Functions/Usage:

    a) login_manager = LoginManager(app) — initialize.

    b) @login_manager.user_loader — callback to reload a user from the session:

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

    c) login_user(user, remember=True) — log a user in.

    d) logout_user() — log out.

    e) Decorator @login_required — protect views so only authenticated users can access them.

    f) current_user proxy — access the logged-in user in views and templates

Challenges faced and solutions

Challenges Faced and Solutions in the Flask Blog Project

Based on the project structure and implementation, here are some common challenges that were likely faced during development and their solutions:

Database Management Challenges

Challenge: Setting up proper relationships between users and posts while maintaining data integrity.

Solution: Implemented SQLAlchemy ORM with proper relationship definitions, using foreign keys to link posts to users and ensuring cascade deletions where appropriate.

Authentication System

Challenge: Creating a secure user authentication system with password hashing and session management.

Solution: Utilized Flask-Login for session management and Werkzeug's security functions for password hashing. Implemented remember-me functionality and proper session timeouts.

File Uploads

Challenge: Handling image uploads for user profiles and post content securely.

Solution: Implemented file validation, secure filename generation, and image resizing to standardize profile pictures and post images while preventing security vulnerabilities.

Pagination

Challenge: Managing large numbers of posts efficiently on the home and user pages.

Solution: Implemented Flask-SQLAlchemy's pagination features to limit the number of posts per page and provide navigation controls.

Search Functionality

Challenge: Creating an efficient search system across post content.

Solution: Implemented a search function using SQLAlchemy queries with LIKE operators and proper indexing to improve performance.

Blueprint Organization

Challenge: Organizing the application code in a maintainable way as it grew in complexity.

Solution: Structured the application using Flask blueprints to separate concerns (users, posts, main routes) and make the codebase more modular and maintainable.

Form Validation

Challenge: Ensuring proper validation of user inputs across registration, login, and post creation.

Solution: Used WTForms with custom validators to enforce data integrity rules and provide meaningful error messages to users.

Responsive Design

Challenge: Making the application work well on both desktop and mobile devices.

Solution: Implemented responsive design principles using Bootstrap's grid system and responsive components.

These challenges and solutions demonstrate the thoughtful architecture and implementation of the Flask blog application, resulting in a robust and user-friendly platform.