Flask Banking System

Project Documentation

Developer: Kaivalya Patil
Technology: Python (Flask Framework)
Live Link: https://flask-banking-system-s40a.onrender.com

NOVEMBER 13, 2025

INDEX

1. Project Overview

The **Flask Banking System** is a web-based banking application that provides users with a secure and interactive platform to perform essential banking operations. The project focuses on **user authentication, session management, and data protection**, built using the Flask web framework.

It simulates a digital banking environment where users can **register, log in, view their account details, and access transaction information**. Security measures such as **password hashing** and **token-based password reset** ensure safe data handling.

---

2. Features

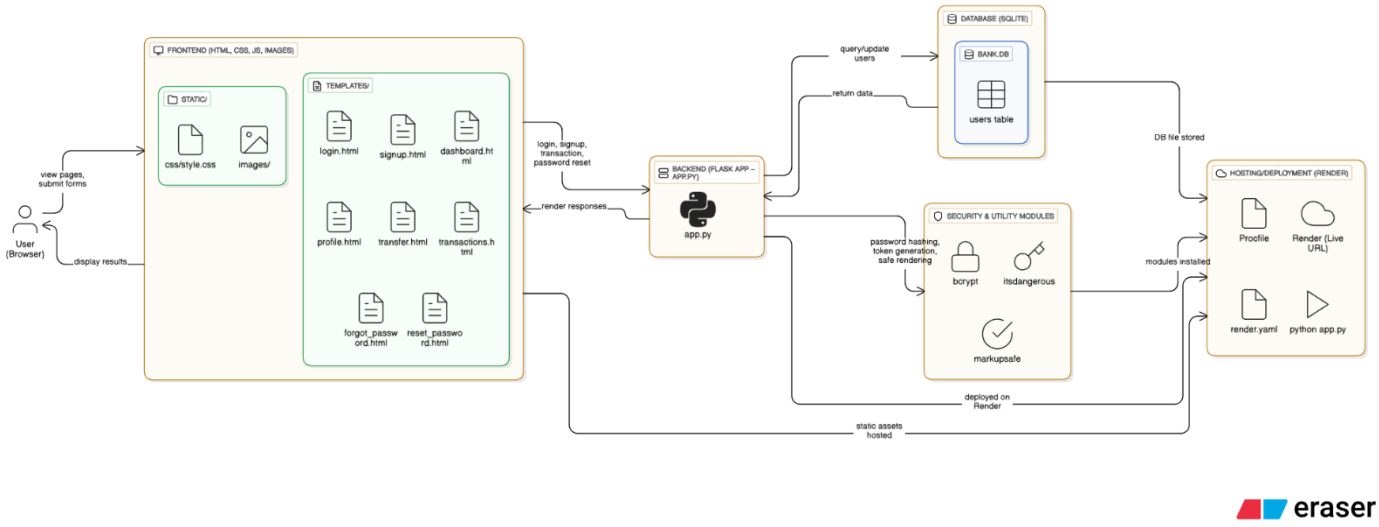
- **User Registration & Login** – Secure signup and authentication using Flask sessions.
- **Dashboard** – Displays account balance, masked email, and masked contact number.
- **Profile Management** – Shows basic user details in a clean UI.
- **Transactions Page** – Simulated transaction history (credits/debits)
- **Fund Transfer Page** – Demo functionality for fund transfers.
- **Forgot & Reset Password** – Token-based password reset system.
- **Session Security** – Restricts access after logout to prevent unauthorized views.
- **Database Integration** – SQLite used for secure and lightweight data storage.

3. Technology Stack

Layer	Technology
Frontend	HTML5, CSS3, Bootstrap
Backend	JS, Python (Flask Framework)
Database	SQLite3
Security	bcrypt, itsdangerous, markupsafe
Deployment	Render (Cloud Hosting)

---

4. Architecture Diagram



eraser

🧍 User (Browser)

- The end-user accesses the web application through a browser.
- They **view pages**, **fill forms** (like login, signup, fund transfer), and **see results** returned from the Flask server.
- Example: When the user logs in, the browser sends a request → Flask verifies it → returns a response (dashboard, etc.).

💻 Frontend (HTML, CSS, JS, Images)

This part defines what the user sees and interacts with.

📁 static/

- Contains **style.css** and **images/** used for the visual look.
- Example: bg.jpg, logo.png, and bank_profile.png.

📁 templates/

- Contains all **HTML templates** that Flask renders dynamically.
- Files include:
  - login.html, signup.html → Authentication pages
  - dashboard.html, profile.html, transactions.html, transfer.html → User and banking pages
  - forgot_password.html, reset_password.html → Password management

Flask renders these pages dynamically using the **render_template()** function in app.py.

🐍 Backend (Flask App – app.py)

This is the **core of the application** — it handles:

- **Routing** (e.g., /login, /signup, /dashboard)
- **Session management**
- **Database queries**
- **Password hashing**
- **Form validation**
- **Data rendering to templates**

It acts as a **bridge** between:

1. The **frontend** (HTML templates)
2. The **database (SQLite)**
3. **Security utilities (bcrypt, itsdangerous)**

🗄️ Database (SQLite)

- The app uses a lightweight **SQLite database** stored in bank.db.
- Contains one main table:
  - users → stores id, name, address, phone_no, email_id, password, etc.
- Flask executes **SQL queries** via sqlite3:
  - Insert new users on signup
  - Retrieve data for login validation
  - Update data for password resets

Flow:
app.py ↔ bank.db (read/write operations)

🔐 Security & Utility Modules

These modules ensure the application's safety and stability:

- **bcrypt** → encrypts user passwords before storing them.
- **itsdangerous** → creates secure tokens for password reset links.
- **markupsafe** → safely renders HTML to prevent injection attacks.

All of these are imported and used inside app.py.

☁️ Hosting / Deployment (Render)

Your app is deployed on **Render**, a cloud platform.

- Files like Procfile and render.yaml define:
    - Which command starts the app → python app.py
    - Environment setup and dependencies

Render handles:

- Running the Flask app on the server
- Hosting static assets (CSS, images)
- Making the app available online via
  🔗 **Live URL**

🔄 Overall Flow Summary

1. User opens the app in the browser.
2. Flask serves an HTML template (from templates/) with CSS/images (from static/).
3. When the user performs an action (like login):
    - Data is sent to Flask (app.py)
    - Flask queries the database (bank.db)
    - Security modules handle password encryption and tokens
4. Flask returns a dynamic response → shown to the user.
5. Everything runs live on **Render Cloud**.

5. Installation and Setup

- **Step 1: Clone the repository**

  ```
  git clone https://github.com/patil-kaivalya/flask-banking-system.git

  cd flask-banking-system
  ```

- **Step 2: Create and activate a virtual environment**

  ```
  python -m venv venv

  venv\Scripts\activate   # For Windows

  # OR

  source venv/bin/activate  # For Mac/Linux
  ```

- **Step 3: Install required dependencies**

  ```
  pip install -r requirements.txt
  ```

- **Step 4: Run the application**

  ```
  python app.py
  ```

- **Step 5: Access in browser**

  http://127.0.0.1:5000

---

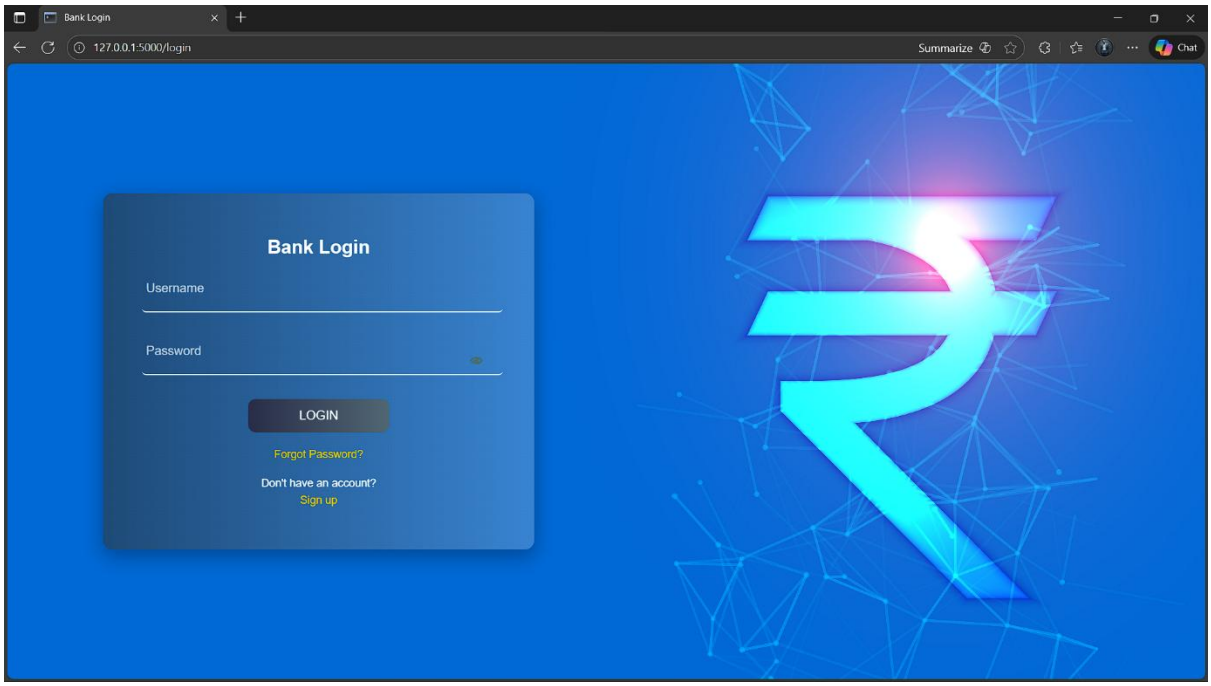
6. Usage

1. Launch the application in your browser.
2. Register a new user or log in using existing credentials.
3. Access the **Dashboard** to view your account summary.
4. Explore **Profile**, **Transactions**, and **Transfer** pages.
5. Use **Forgot Password** to reset credentials securely.
6. Log out safely using the logout option to end the session.

---

7. Screenshots

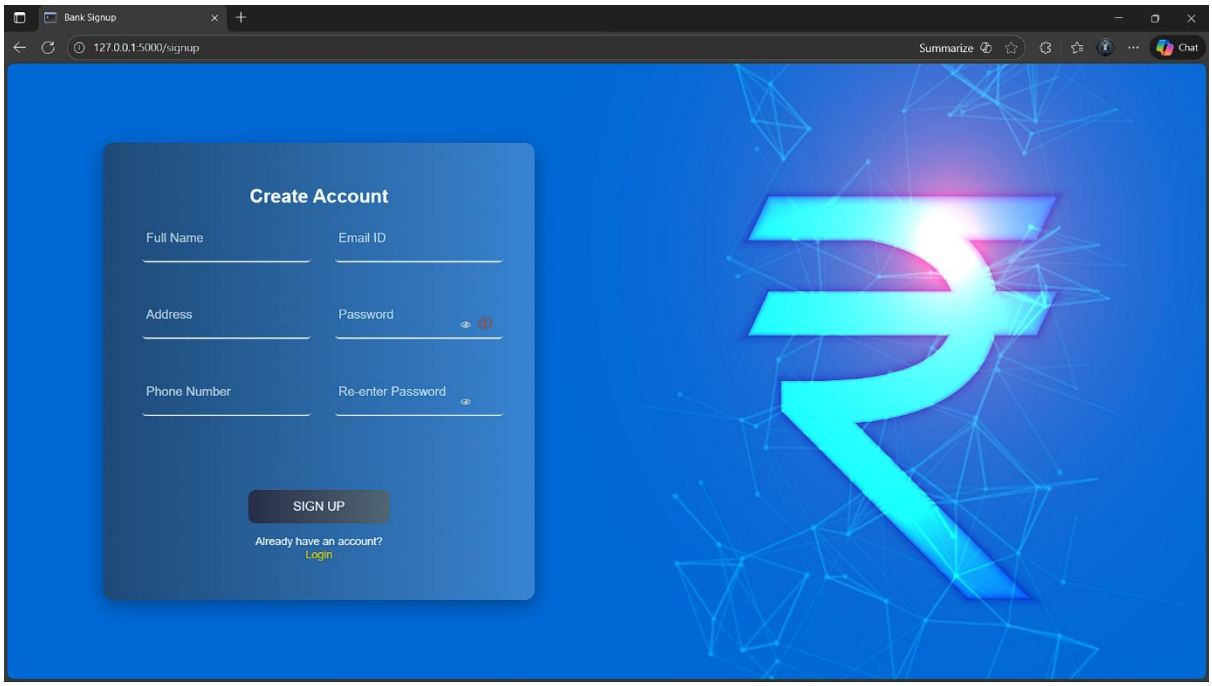
- **Login**



As mentioned in this screenshot – Login Page Explanation:

- The file is an HTML Bank Login Page template.
- Uses Flask's Jinja2 template tags ({{ ... }}) for linking CSS and routes (url_for).
- Displays flash messages for success/error feedback.
- Contains a form with two input fields:
  - email_id → for username
  - password → for password
- Includes a password visibility toggle (👁 icon) using JavaScript (showPassword()), which shows the password for 3 seconds.
- Provides navigation links to *Forgot Password* and *Sign Up* pages.
- Styled through an external stylesheet (style.css) located in Flask's static folder.
- Clean and user-friendly UI inside a centered form container.

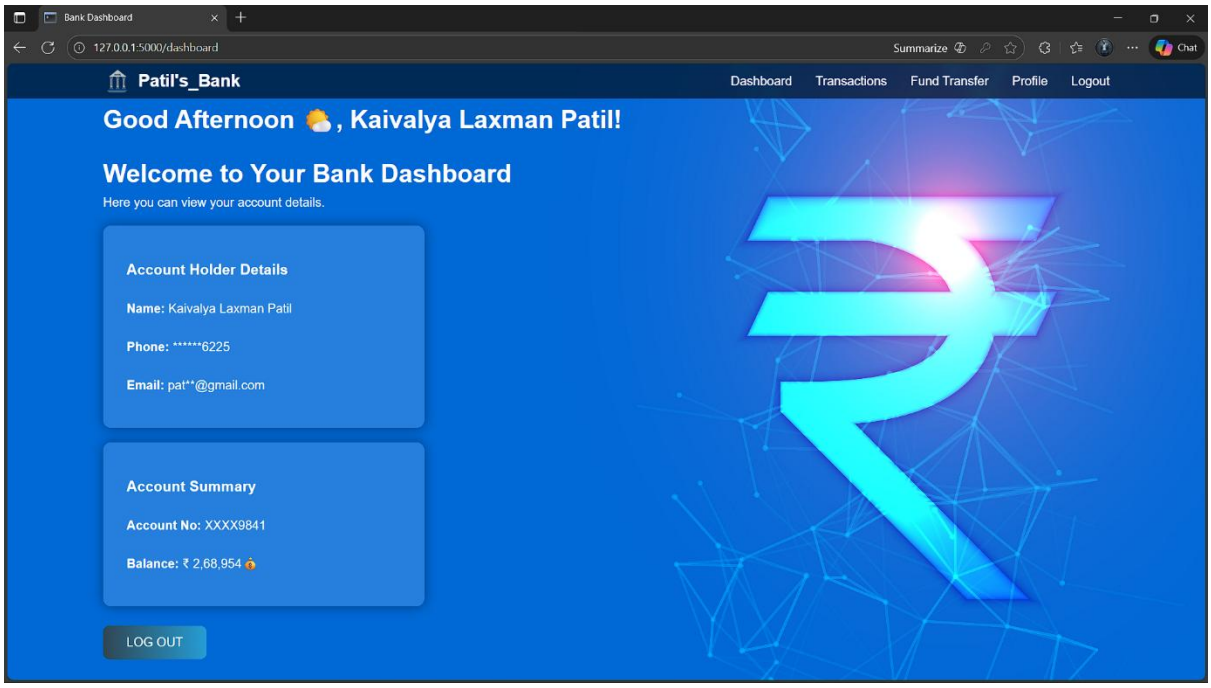
- **Sign Up**



As mentioned in this screenshot – Signup Page Explanation:

- This file is the **Bank Signup Page** that allows users to **create a new account**.
- Uses Flask's **Jinja2 template** syntax ({{ url_for() }}) for linking styles and routes.
- Designed using a **two-column layout** — left for personal details and right for login credentials.
- Input fields include:
  - Full Name, Address, Phone Number (with 10-digit validation).
  - Email ID, Password, and Re-enter Password.
- **Password field validation** uses a regex pattern that requires:
  - Minimum 8 characters
  - At least 1 letter, 1 number, and 1 special symbol
- Features a **popup info box (ℹ️)** explaining the password requirements.
- Includes **eye icons (👁)** to toggle password visibility for both password fields.
- A **password match indicator** highlights green/red border based on match status.
- After signup, users are guided to the **Login page** via a redirect link.
- Styled using both an external style.css file and inline CSS for interactive elements.

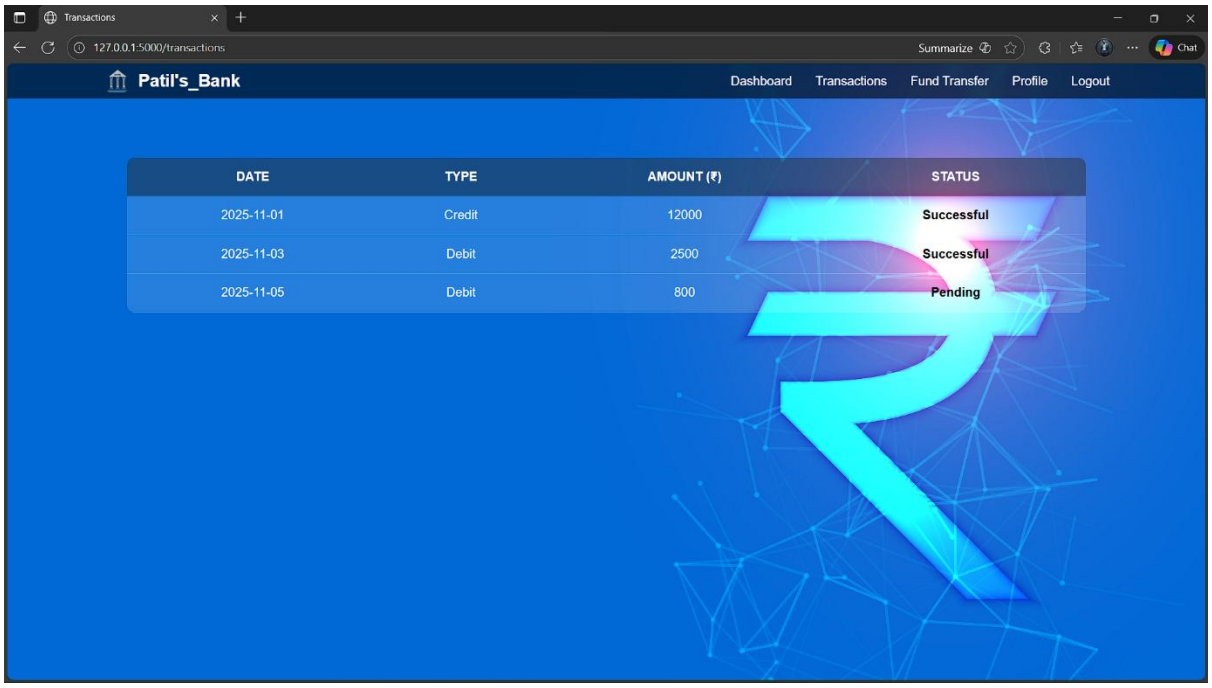
- **Dashboard**



As mentioned in this screenshot – Dashboard Page Explanation:

- This file represents the **Bank Dashboard page** of the web application.
- Uses Flask's **template rendering** (e.g., {{ name }}, {{ balance }}, {{ url_for() }}) to display user-specific data.
- Includes a **navbar** through {% include 'navbar.html' %} for consistent navigation.
- The **background image** is applied using /static/images/bg.jpg with full-screen coverage.
- Contains two main sections:
    - **Account Holder Details** – shows masked name, phone, and email.
    - **Account Summary** – displays account number and animated balance count.
- Includes a **"Log Out" button** with gradient styling and confirmation alert before logging out.
- Implements **browser back-button restriction** using JavaScript to prevent users from navigating back after login.
- Adds a **dynamic greeting** ("Good Morning 🌞 / Afternoon 🌤️ / Evening 🌙") based on the user's system time.
- The **balance** value animates smoothly from 0 to the actual balance using a timed counter effect.
- The page design ensures readability and avoids navbar overlap using padding-top: 100px;.

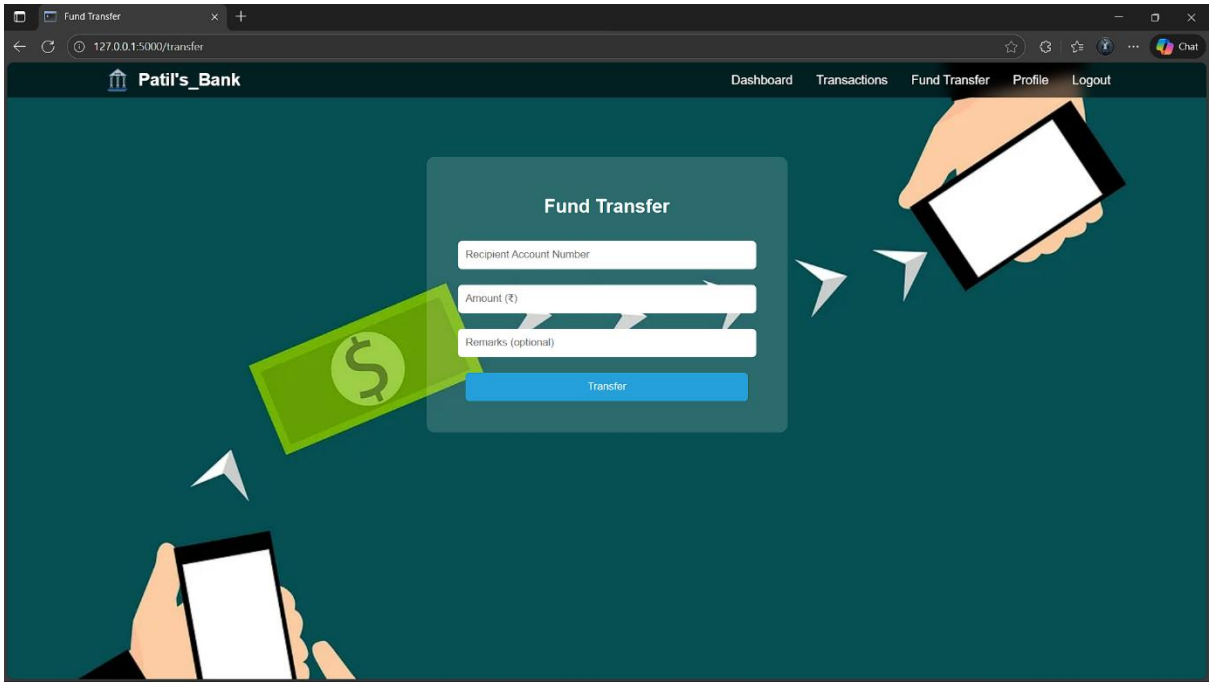
- **Transactions**



As mentioned in this screenshot – Transactions Page Explanation:

- This file represents the **Transactions Page** of the banking web application.
- Uses Flask's **template looping ({% for txn in transactions %})** to dynamically display all user transactions from the database.
- Includes a **navbar** using {% include 'navbar.html' %} for navigation consistency.
- Displays a **styled table** showing transaction details:
  - **Date**, **Type**, **Amount (₹)**, and **Status**.
- The **background** uses /static/images/bg.jpg with full-screen coverage.
- The **table design** has:
  - Transparent background (rgba(255,255,255,0.15))
  - Rounded corners and hover effect for better readability
  - Uppercase headers with a subtle dark overlay
- The **status column** is highlighted in **black bold text** for clear visibility.
- Table content is centered and responsive with width: 80% for balanced layout.
- Provides a **clean and minimal interface** for users to review their past transactions easily.

- **Fund Transfer**



As mentioned in this screenshot – Fund Transfer Page Explanation:

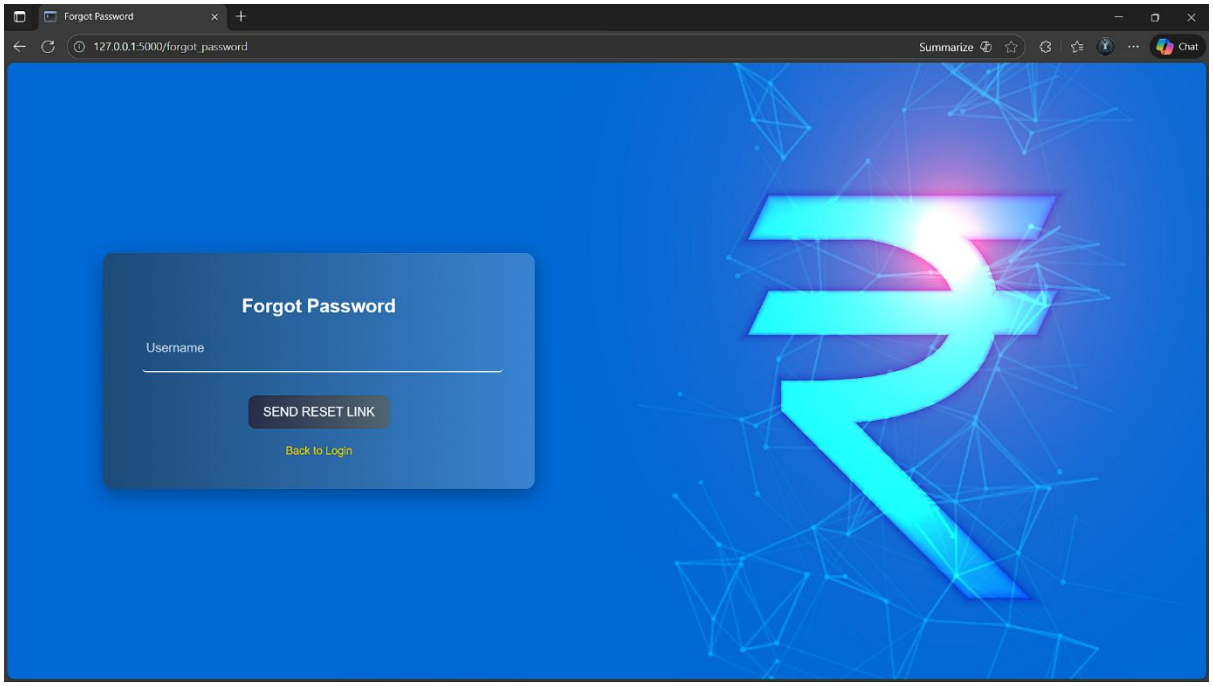
- This file represents the **Fund Transfer Page** of the banking web application.
- Uses Flask's **navbar include** ({% include 'navbar.html' %}) for seamless navigation.
- The **background image** (transfer_funds.png) gives a professional banking theme with a centered layout.
- Contains a **fund transfer form** where users can enter:
    - Recipient **Account Number**
    - **Amount (₹)** to transfer
    - **Remarks** (optional)
- The form is styled inside a semi-transparent container (rgba(255,255,255,0.15)) with rounded corners for a modern look.
- The **input fields and button** are evenly spaced and responsive (width: 90%).
- The **Transfer button** uses a gradient color effect that changes on hover (#26a0da → #314755).
- Ensures a **clean, user-friendly interface** to initiate fund transfers safely and quickly.

- **Profile**

As mentioned in this screenshot – Profile Page Explanation:

- This file represents the **User Profile Page** of the banking web application.
- Uses Flask's **template variables** ({{ user.name }}, {{ masked_email }}, etc.) to dynamically show user information.
- Includes a **navbar** using {% include 'navbar.html' %} for consistent navigation across pages.
- The **background image** (bank_profile.png) gives a professional and clean look.
- Displays a **profile box** containing:
    - User's **Name**, **Masked Phone**, **Masked Email**, **Account Number**, and **Balance**.
- The **balance amount** is animated from 0 to the actual value using JavaScript for a smooth count-up effect.
- The **profile box** has:
    - Semi-transparent background (rgba(255,255,255,0.15))
    - Rounded corners, soft shadow, and fade-in animation for an elegant appearance.
- Layout is **left-aligned** using flexbox with proper padding for spacing.
- Provides a **simple, readable, and interactive** design focused on user details.

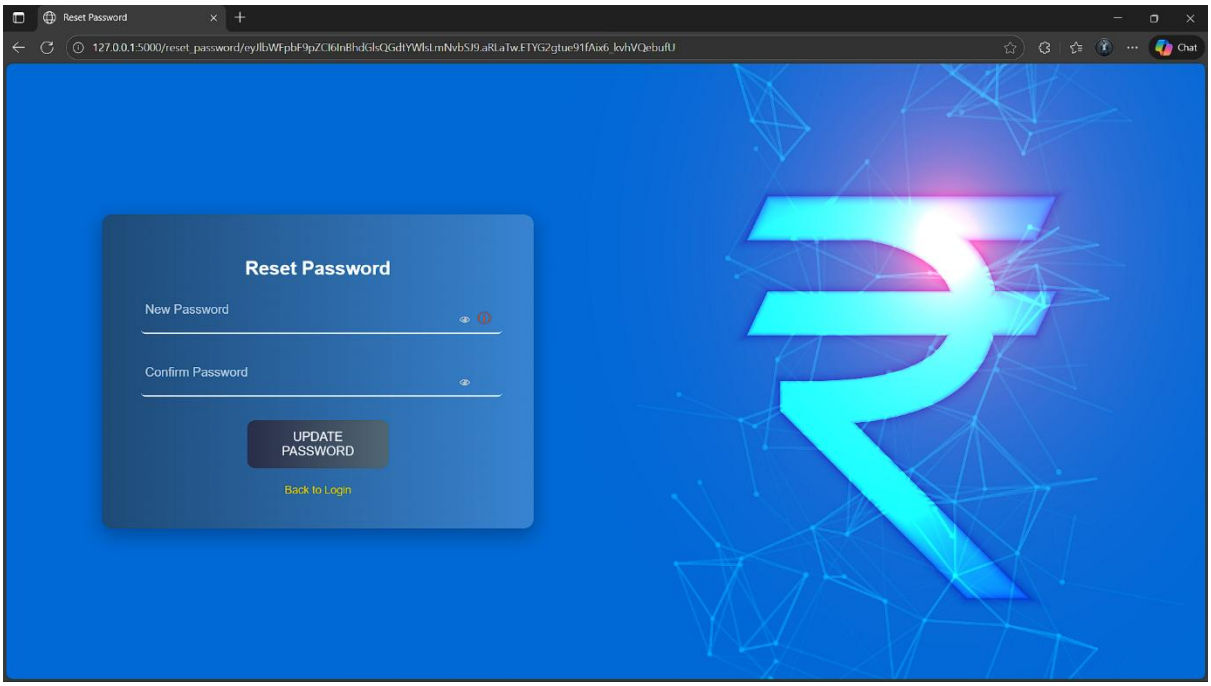
- **Forgot Password**



As mentioned in this screenshot – Forgot Password Page Explanation:

- This file represents the **Forgot Password Page** of the banking web application.
- Allows users to **reset their password** by entering their registered **email/username**.
- Uses Flask's **get_flashed_messages()** function to display success or error alerts dynamically.
- Includes a **single input field** (email_id) for entering the user's email address.
- The form submits via the **POST method**, triggering the server to send a **password reset link**.
- Uses Flask's **url_for()** to navigate between pages (e.g., *Back to Login*).
- Styled with the common external stylesheet (style.css) for consistency with other forms.
- The layout is contained in a **centered form box** with a clear heading and button.
- Provides a **simple and user-friendly interface** to recover account access securely.

- **Reset Password**



As mentioned in this screenshot – Reset Password Page

- Designed for users to securely reset their password.
- Contains two input fields — **New Password** and **Confirm Password**.
- 👁 **Eye icon** temporarily shows the password for **3 seconds**.
- ℹ️ **Info icon** displays password strength rules (min 8 chars, 1 letter, 1 number, 1 special symbol).
- ⚠️ **Real-time validation** checks if both passwords match and shows red/green borders with messages.
- Uses **HTML5 pattern validation** for strong password enforcement.
- Styled with CSS for popup info, animations, and modern UI.
- Flask backend handles flash messages and password update logic.

Main Application (app.py)

- **Framework Used:** Flask (Python web framework).
- **Database:** SQLite (bank.db) for storing user data.
- **Password Security:** Passwords hashed using **bcrypt** for strong encryption.
- **Session Management:** Flask session stores user info securely during login.

Key Functionalities

1. **Home Route (/)**
    - Redirects users to the login page.
2. **Login System (/login)**
    - Validates email & password from the database.
    - Uses bcrypt to verify hashed passwords.
    - On success → redirects to **Dashboard**.
3. **Signup (/signup)**
    - Registers a new user after validation.
    - Ensures unique email & phone number.
    - Stores hashed passwords in the database.
4. **Logout (/logout)**
    - Clears all session data and redirects to login.
5. **Forgot Password (/forgot_password)**
    - Generates a **secure reset token** using itsdangerous.
    - Displays reset link in the console and as a flash message.
6. **Reset Password (/reset_password/<token>)**
    - Verifies token validity and expiry (15 minutes).
    - Allows user to reset password securely with validation.

7. **Dashboard (/dashboard)**
    - Displays masked email, phone number, random account number, and balance.
    - Prevents caching for security.
8. **Transactions (/transactions)**
    - Shows dummy transaction history (Credit/Debit).
9. **Fund Transfer (/transfer)**
    - Opens transfer page (future functionality ready).
10. **Profile (/profile)**
    - Shows masked user data and current balance.

Security Features

- Password hashing using **bcrypt**.
- Token-based password reset via **itsdangerous**.
- Session validation on every protected route.
- No-cache headers to prevent sensitive data storage in browser.

---

8. Future Enhancements

1. **Email Integration for Password Reset:**
    - Currently, the reset link is displayed on the console.
    - In the future, SMTP or EmailJS can be integrated to send reset links directly to the user's registered email.
2. **Real-Time Fund Transfer Module:**
    - Implement an actual backend transaction system where users can send or receive money securely with transaction history stored in the database.
3. **Admin Dashboard:**
    - Add an admin panel to manage users, monitor transactions, and handle fraud detection or suspicious activities.
4. **Two-Factor Authentication (2FA):**
    - Improve login security by adding OTP verification via email or SMS.
5. **Enhanced User Profile Management:**
    - Allow users to update personal details, upload profile pictures, and view detailed account statistics.
6. **Data Visualization Dashboard:**
    - Integrate charts and analytics for better transaction insights and spending patterns.
7. **Deployment with Cloud Database:**
    - Migrate SQLite to cloud databases like PostgreSQL or MySQL for scalability and performance.
8. **Mobile-Friendly and Responsive Design:**
    - Optimize UI for mobile and tablet users for better accessibility.

---

9. Conclusion

This **Flask Banking Web Application** successfully demonstrates how core banking functionalities can be implemented securely using Python and Flask.
The project covers key operations such as **user registration, login, password management, dashboard display, and data masking** — ensuring user privacy and security.

By using technologies like **Flask, SQLite, bcrypt, and itsdangerous**, the application achieves a strong balance between simplicity and data protection.
Although currently developed as a prototype, it lays a solid foundation for building a **fully functional online banking system** with real-time transactions, secure email communication, and admin management in the future.