URL Shortener Web Application with Login

Bug Report and Implementation Summary

1. Project Overview

This project is a URL Shortener web application built using Flask, Flask-SQLAlchemy, Flask-Login, SQLite, and HTML/CSS/Bootstrap. The application allows users to sign up, log in, shorten long URLs, copy the shortened URLs, and view their personal URL history on the same web app. Each user has their own account and only sees the URLs they have shortened.

The main objectives are:

- To shorten long URLs into compact, easy-to-share links.
- To store both original and shortened URLs in a database.
- To provide authentication so that users can log in and manage their own URL history.

---

2. Architecture and Workflow

Technology stack

- Backend: Flask (Python)
- ORM: SQLAlchemy
- Authentication: Flask-Login
- Database: SQLite
- Frontend: HTML, CSS, Bootstrap

Data models

1. **User**
   - `id` (Integer, primary key)
   - `username` (String, 5–9 characters, unique, required)
   - `password_hash` (String, required)
2. **URL**
   - `id` (Integer, primary key)
   - `original_url` (String, required)
   - `short_code` (String, unique, required)
   - `user_id` (ForeignKey to `User.id`, required)

Main routes

- `/` – Home page with “Login” and “Sign Up” buttons.
- `/signup` – User registration with username and password.

- `/login` – User login form.
- `/logout` – Logs the user out.
- `/shortener` – Main URL shortener page; only accessible after login.
- `/<short_code>` – Redirects to the original URL based on the short code.

User workflow

1. User opens the home page and chooses to sign up or log in.
2. During sign up, the username length and uniqueness are validated.
3. After logging in, the user is redirected to the URL shortener page.
4. The user enters a long URL and clicks the **Shorten** button.
5. The system validates the URL, generates a unique short code, saves the mapping to the database, and shows the shortened URL.
6. A history table shows all original and shortened URLs created by that user.
7. Visiting a short URL path (e.g., `/abc123`) redirects to the original URL.

---

3. Bugs and Issues Encountered

Bug 1 – Python and pip not recognized

Symptom

Initially, running `python` or `pip` in the command prompt showed errors like:

```
'python' is not recognized as an internal or external command
'pip' is not recognized as an internal or external command
```

Root Cause

Python was not properly installed or added to the system PATH, and Windows was using the Microsoft Store "Python" alias instead of a real Python installation.

Fix

- Disabled the Python app execution aliases from Windows Settings → Apps → Advanced app settings → App execution aliases.
- Downloaded and installed Python from the official website (python.org).
- During installation, checked **"Add Python to PATH"**.
- Opened a new command prompt and confirmed the installation with `python --version` and `pip --version`.

Result

Python and pip were available from the terminal, allowing the creation of a virtual environment and installation of required packages.

Bug 2 – Flask modules not found inside the virtual environment

Symptom

After creating the project and running `python app.py`, the application failed with:

```
ModuleNotFoundError: No module named 'flask'
```

Root Cause

The virtual environment was activated, but Flask and other dependencies had not yet been installed inside that environment.

Fix

Inside the active virtual environment, the following command was executed:

```
bash
pip install flask flask_sqlalchemy flask_login validators
```

This installed Flask, Flask-SQLAlchemy, Flask-Login, and the validators library within the venv.

Result

All required packages were available, and Flask imports in `app.py` worked correctly.

Bug 3 – Import error for `db` from `models`

Symptom

Running the app produced:

```
ImportError: cannot import name 'db' from 'models'
```

Root Cause

The `models.py` file either did not exist or did not define the `db` object and the `User` and `URL` models expected by `app.py`.

Fix

Created and completed `models.py` with:

```
python
from flask_sqlalchemy import SQLAlchemy
```

```
from flask_login import UserMixin

db = SQLAlchemy()

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(9), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    urls = db.relationship("URL", backref="user", lazy=True)

class URL(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    original_url = db.Column(db.String(2048), nullable=False)
    short_code = db.Column(db.String(10), unique=True, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
```

In `app.py` the import is:

```
python
from models import db, User, URL
```

Result

The database object and models were correctly imported, and `db.create_all()` successfully created the SQLite tables.

Bug 4 – Blank page on the home route

Symptom

Accessing `http://127.0.0.1:5000/` showed a blank white page, even though the console logged `GET / 200`.

Root Cause

Flask was rendering `home.html`, but the `templates/home.html` file was either missing, empty, or not placed in the correct folder.

Fix

- Verified the folder structure: a `templates` folder at the same level as `app.py`.
- Created a very simple version of `home.html` to test rendering, then replaced it with the final Bootstrap layout containing heading and login/signup buttons.

Result

The home page displayed correctly, proving that the template rendering and folder structure were correct.

Bug 5 – TemplateNotFound: `shortener.html`

Symptom

After logging in successfully, the browser showed an error page:

```
jinja2.exceptions.TemplateNotFound: shortener.html
```

Root Cause

The route `/shortener` called `render_template("shortener.html", ...)`, but the corresponding `shortener.html` file did not exist or had a different name.

Fix

- Created `templates/shortener.html` with a form for entering the original URL, a **Shorten** button, an output field for the shortened URL with a **Copy** button, and a table showing the user's URL history.
- Ensured the file name was exactly `shortener.html` and placed directly inside the `templates` folder.

Result

After login, the user is now redirected to the correct shortener page, and the template loads without error.

---

4. Implementation Details

4.1 Authentication and user management

- Used **Flask-Login** to manage sessions.
- Implemented a `LoginManager`, `user_loader`, `login_user`, `logout_user`, and the `@login_required` decorator for protected routes.
- Passwords are stored as hashes using `generate_password_hash` and validated with `check_password_hash`.
- The signup route enforces:
  - Username length between **5 and 9 characters**.
  - Username uniqueness; if taken, an error message "This username already exists…" is shown.
- Login validates credentials and redirects to the `/shortener` route on success.

4.2 URL shortening logic

- Used the `validators.url()` function to verify that the user input is a valid URL including scheme (e.g. `http://` or `https://`). If invalid, an error message is displayed instead of shortening.

- Implemented a `generate_short_code` function that creates a random string of letters and digits and ensures uniqueness by checking against existing entries in the database.
- When a valid URL is submitted:
    - A new `URL` record is created with `original_url`, generated `short_code`, and the current user's `user_id`.
    - The full short URL is built using `request.host_url` plus the `short_code` and displayed in the UI.

4.3 Redirection and history

- The dynamic route `/<short_code>` queries the `URL` table; if a matching record is found, the user is redirected to `original_url`.
- On the `/shortener` page, the application queries all URLs belonging to the logged-in user and displays them in a table (Original URL and Short URL).
- A "Copy" button uses JavaScript (`navigator.clipboard.writeText`) to copy the shortened URL to the clipboard with a single click.

---

5. Final Result

The final application satisfies the given requirements:

- Users can **sign up** with a unique username (5–9 characters) and a password.
- Users can **log in** and access the main URL shortener interface.
- Users can enter long URLs, get shortened URLs, and **copy** them easily.
- Each original and shortened URL pair is **stored in SQLite** and displayed in a history table for the logged-in user.
- Visiting the short URL path redirects to the original URL.

The project also involved solving environment issues (Python and pip), dependency management, setting up Flask with SQLAlchemy and Flask-Login, and debugging template and routing problems to achieve a clean, working full-stack web application.