URL Shortener Web Application for Basic Users

- Nitisha Naigaonkar

---

1. Introduction

This project was assigned as part of the Flask task section, with the objective of developing a basic web application using Flask, database integration, and frontend technologies. The given problem statement required the implementation of a **URL Shortener Web Application** that allows users to shorten long URLs and store them for future reference.

In general usage, long URLs can be inconvenient to share or manage. The purpose of this assignment was to understand how such a problem can be handled using backend logic, database storage, and routing mechanisms in Flask. The project focuses on implementing the required functionality as specified, rather than building a complex or large-scale system.

The application was developed using **Flask (Python)** for the backend, **SQLAlchemy ORM** for database handling, and **SQLite** as the database. The frontend was implemented using **HTML**, **CSS**, **and Bootstrap**, as mentioned in the project instructions.

---

2. Objectives of the Project

The objectives of this project were defined in the given problem statement and are listed below:

1. To create a web application that shortens long URLs.
2. To store original URLs along with their shortened versions.
3. To validate the URL entered by the user before processing it.

4. To display previously shortened URLs on a history page.
5. To redirect users from the shortened URL to the original URL.

---

3. Technology Stack Used

The technology stack for this project was chosen based on the assignment requirements:

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

Flask was used because this task comes under the Flask task section, and the backend implementation was expected to be done using Flask. SQLAlchemy ORM was used to manage database operations in an organized manner, as required in the assignment. SQLite was selected because it is easy to configure and sufficient for storing the data required for this basic application.

---

4. System Design and Architecture

The application follows a simple request-response model:

1. The user interacts with the application through the browser.
2. The frontend sends requests to the Flask backend.
3. Flask routes handle the request and execute the required logic.
4. URL data is stored and retrieved using SQLAlchemy ORM.
5. The processed response is rendered back to the user.

To keep the code structured and readable, the project was divided into separate files:

- `app.py` for handling routes and application flow
- `models.py` for defining the database schema
- `services.py` for URL validation and short code generation logic

---

5. Database Design

A single database table was used to store the URL information.

Table: URLMap

- **id:** Primary key for each record
- **original_url:** Stores the original URL entered by the user
- **short_code:** Stores the generated shortened URL code
- **created_at:** Stores the timestamp when the URL was saved

This structure ensures that each shortened URL can be uniquely identified and redirected correctly.

---

6. Project Workflow

The workflow of the application is as follows:

1. The user enters a URL on the home page.
2. The application checks whether the entered URL is valid.
3. If the URL is valid, a short code is generated.
4. The original URL and short code are stored in the database.
5. The shortened URL is displayed to the user.

6. When the shortened URL is accessed, the user is redirected to the original URL.
7. The history page displays all previously stored URLs.

---

7. URL Validation

URL validation was implemented to ensure that incorrect or malformed URLs are not stored in the database. Before generating a shortened URL, the application verifies whether the input follows a valid URL format.

If the validation fails, an appropriate error message is shown to the user. This step helps in maintaining clean and usable data.

---

8. Short URL Generation Logic

The shortened URL is generated using a random combination of alphanumeric characters. The length of the generated short code is kept small to ensure that the shortened URL remains concise.

Before storing the short code, the database is checked to avoid duplication. If a duplicate short code is found, a new code is generated.

---

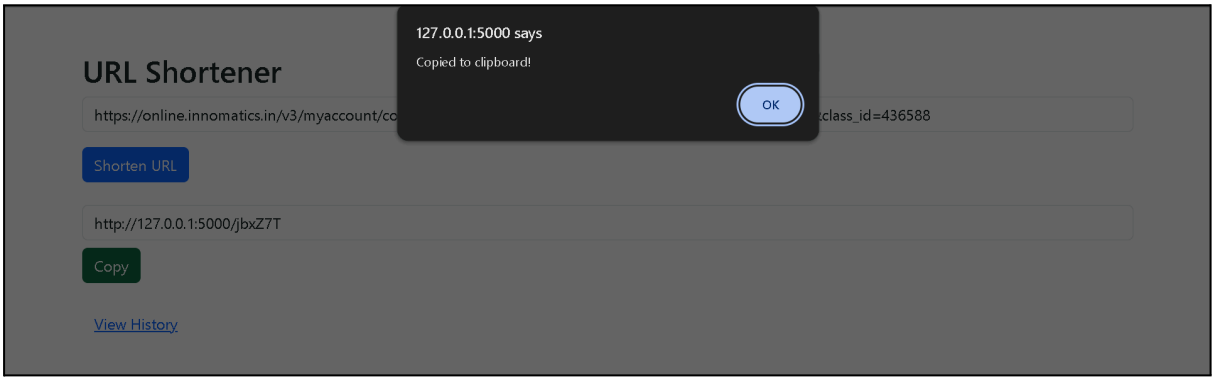
9. Redirection Mechanism

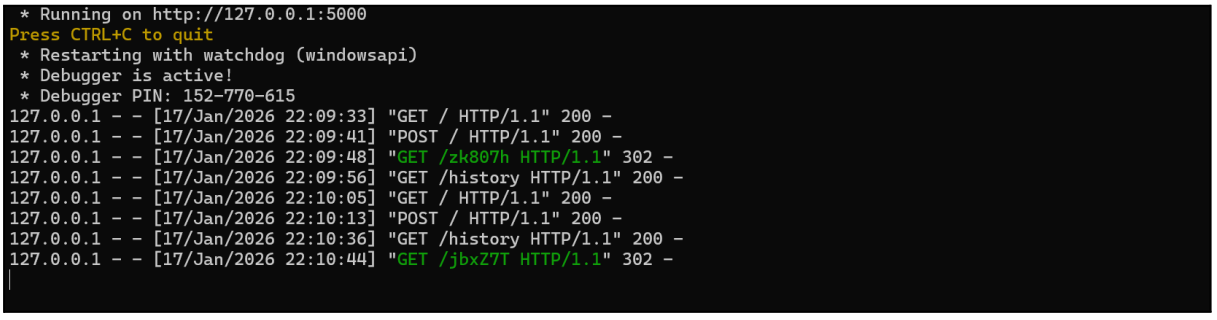
When a user accesses a shortened URL, the application extracts the short code from the URL. The corresponding original URL is retrieved from the database, and the user is redirected using an HTTP redirect response.

This approach ensures correct and efficient redirection.

---

10. Project Screenshots and Output





11. Challenges

During the implementation of this project, the following challenges were faced:

1. Ensuring that each generated short URL was unique.
2. Implementing proper URL validation.
3. Understanding Flask routing and redirection.
4. Structuring the project files according to best practices.

These challenges were resolved through testing, debugging, and referring to documentation.

---

12. Learnings

This project helped in gaining practical understanding of:

- Flask application structure and routing
- Database operations using SQLAlchemy ORM
- URL validation techniques
- Backend and frontend integration
- Debugging and testing Flask applications

---

13. Conclusion

The URL Shortener Web Application was successfully implemented according to the given assignment requirements. The application fulfills all the specified objectives, including URL shortening, database storage, and redirection.

This project provided hands-on experience with Flask-based web development and database integration using SQLAlchemy.