URL Shortener Web Application

(Basic Version)

1. Introduction:

- ➢ In today's digital world, sharing information through the internet has become a daily activity. URLs (Uniform Resource Locators) are used to access web pages, but many URLs are long, complex, and difficult to remember or share. Long URLs are inconvenient when sharing via emails, social media platforms, or messaging applications.
- ➢ A URL Shortener Web Application solves this problem by converting long URLs into shorter, easy-to-share links. These shortened URLs redirect users to the original long URLs when accessed. Popular platforms like Bitly and TinyURL use similar concepts.
- ➢ This project focuses on building a basic URL Shortener Web Application using Flask (Python) for the backend, HTML for the frontend, and SQLite database for storing URL data. The application allows users to shorten URLs, validate them, and view the history of previously shortened URLs.

2. Objectives of the Project:

The main objectives of this project are:

- ➢ To design and develop a web application that shortens long URLs.
- ➢ To allow users to enter any valid URL and generate a unique short URL.
- ➢ To store original URLs and their shortened versions in a database.
- ➢ To implement URL validation to ensure only correct URLs are shortened.
- ➢ To provide a history page displaying all previously shortened URLs.
- ➢ To understand the integration of frontend, backend, and database using Flask.

3. Why Do We Need a URL Shortener?

- ➢ Long URLs can be inconvenient and error-prone when manually typing or sharing.
- ➢ Some reasons why URL shorteners are useful include:
  - Easier sharing of links
  - Improved readability
  - Reduced chances of typing errors
  - Better user experience
  - Useful for analytics and tracking (advanced versions)
  - This project demonstrates how such a system works at a basic level.

4. Technologies Used:

4.1 Frontend Technologies

HTML: Used to create the structure of web pages.
Basic CSS (optional): For minimal styling.
Jinja2 Templates: Used by Flask to render dynamic content.

4.2 Backend Technologies

Python: Core programming language.
Flask: Lightweight web framework for handling requests and routing.
Flask-SQLAlchemy: ORM (Object Relational Mapper) for database operations.

4.3 Database

SQLite: Lightweight database used to store URLs locally.

4.4 Other Libraries

validators: Used to validate URLs.
random & string: Used to generate short URL codes.

5. Project Architecture:

The project follows a simple MVC (Model–View–Controller) style architecture:

Model: Handles database structure using SQLAlchemy (models.py)

View: HTML templates (index.html, history.html)

Controller: Flask routes and logic (app.py)

6. Project Folder Structure:

```
URL_Shortener_Project/
│
├── app.py
├── models.py
├── requirements.txt
├── urls.db
├── templates/
│   ├── index.html
│   └── history.html
└── static/
```

7. Database Design

The database contains a single table to store URLs.

Table: URL

Column name	Data type	Description
id	Integer	Primary Key
Original url	String	User-entered URL
Short_code	String	Generated short code

Each shortened URL is stored along with its original URL to maintain history.

8. Implementation Details

8.1 Flask Application Setup

- The Flask application is initialized and configured with SQLite database settings using SQLAlchemy

```
app = Flask(_name_)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///urls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

8.2 URL Short Code Generation

- A random combination of letters and digits is generated to create a short URL code. This ensures uniqueness and avoids collisions

```
def generate_short_code(length=6):
    characters = string.ascii_letters + string.digits
    return ''.join(random.choice(characters) for _ in
range(length)).
```

8.3 URL Validation

- The validators library is used to check whether the entered URL is valid before storing it in the database.

```
if validators.url(original_url):
```

8.4 Routing Logic

- Home Page (/)
- Displays input field for URL.
- Handles GET and POST requests.
- Redirect Route (/<short_code>)
- Redirects users to the original URL.
- History Page (/history)
- Displays all shortened URLs from the database.

9. Project Workflow:

- User opens the home page.
- User enters a URL in the input field.
- Application validates the URL.
- A unique short code is generated.
- Original URL and short code are stored in the database.
- Shortened URL is displayed to the user.
- User can view all past URLs in the history page.
- Clicking a short URL redirects to the original URL.

10. Challenges Faced and Solutions:

Challenge 1: Database Initialization Error

- **Issue:** SQLAlchemy database not initializing.
- **Solution:** Used app.app_context() while creating database tables.

Challenge 2: Method Not Allowed Error

- **Issue:** 405 Method Not Allowed on /history
- **Solution:** Ensured correct HTTP method (GET) was used.

Challenge 3: URL Validation

- **Issue:** Invalid URLs being saved.

- **Solution:** Used validators.url() to check URL correctness.

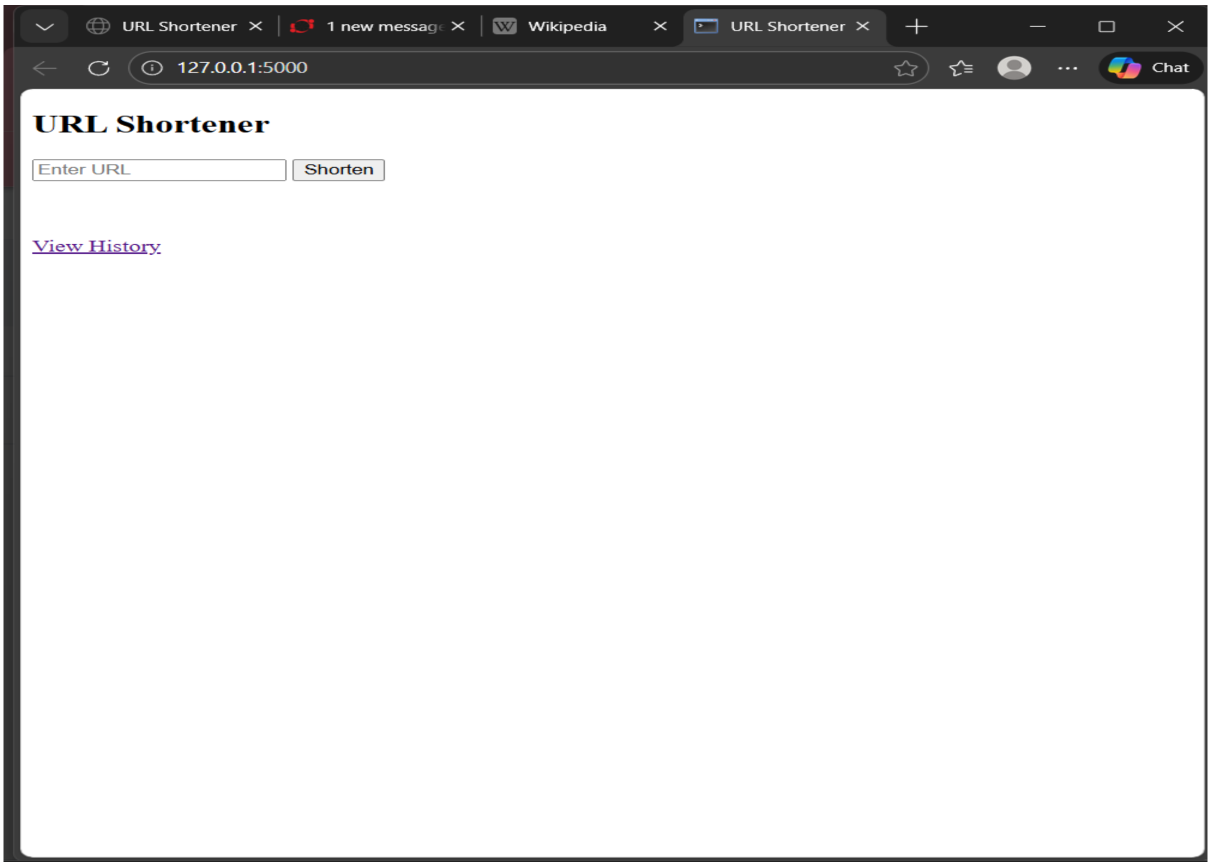
11. Future Enhancements:

- This project can be enhanced by:
- Adding user authentication
- Preventing duplicate URLs
- Custom short URLs
- Analytics and click tracking
- Deployment on cloud platforms (Heroku/AWS)
- Adding CSS and Bootstrap for better UI

12. Conclusion:

The URL Shortener Web Application project successfully demonstrates the fundamentals of web development using Flask. It integrates frontend, backend, and database components into a single functional system. Through this project, key concepts such as routing, database ORM, URL validation, and dynamic HTML rendering were learned.

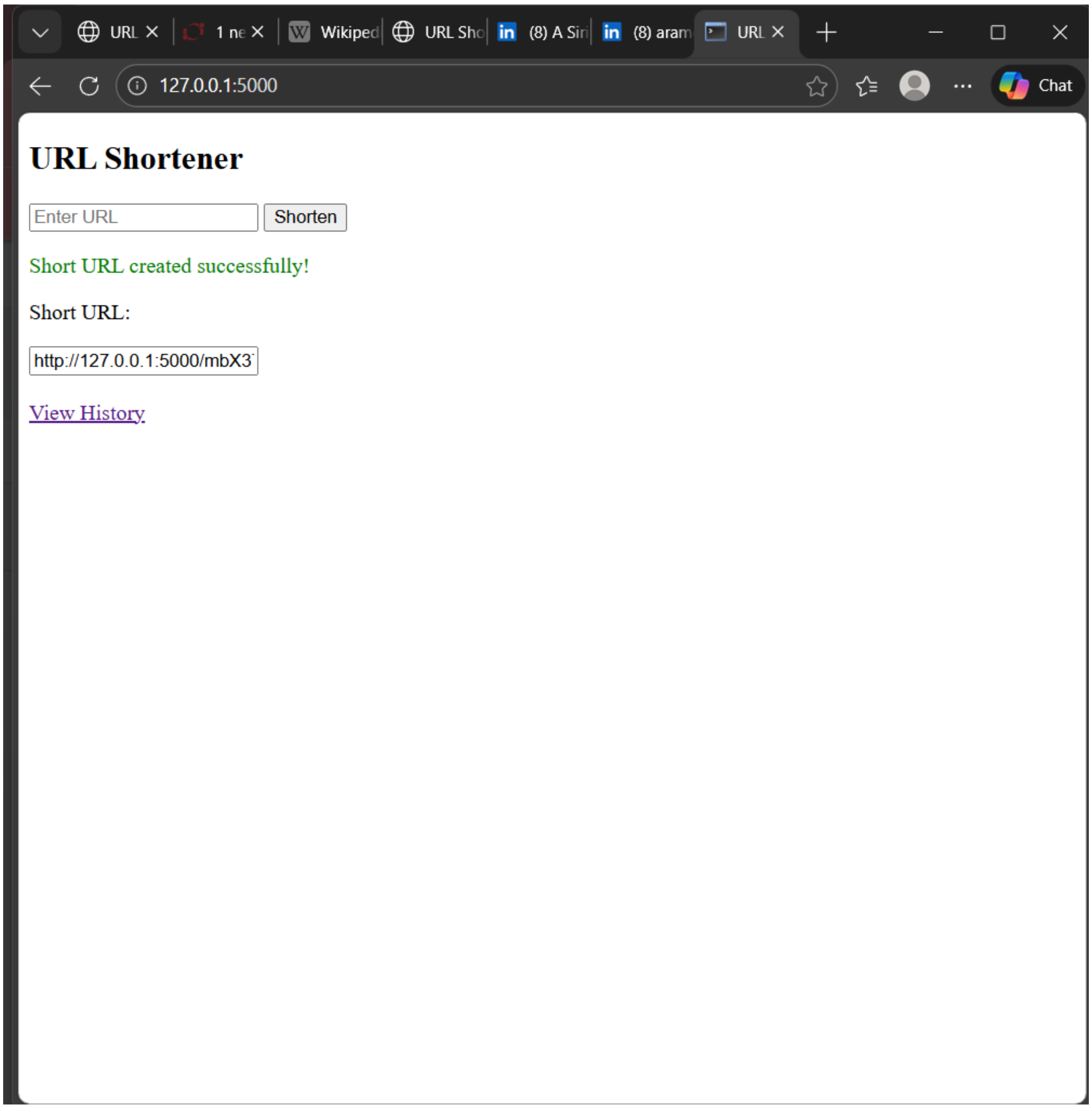
This project helped strengthen practical knowledge of Python Flask and provided hands-on experience in building real-world web applications.



URL Shortener

Enter URL Shorten

View History



URL Shortener

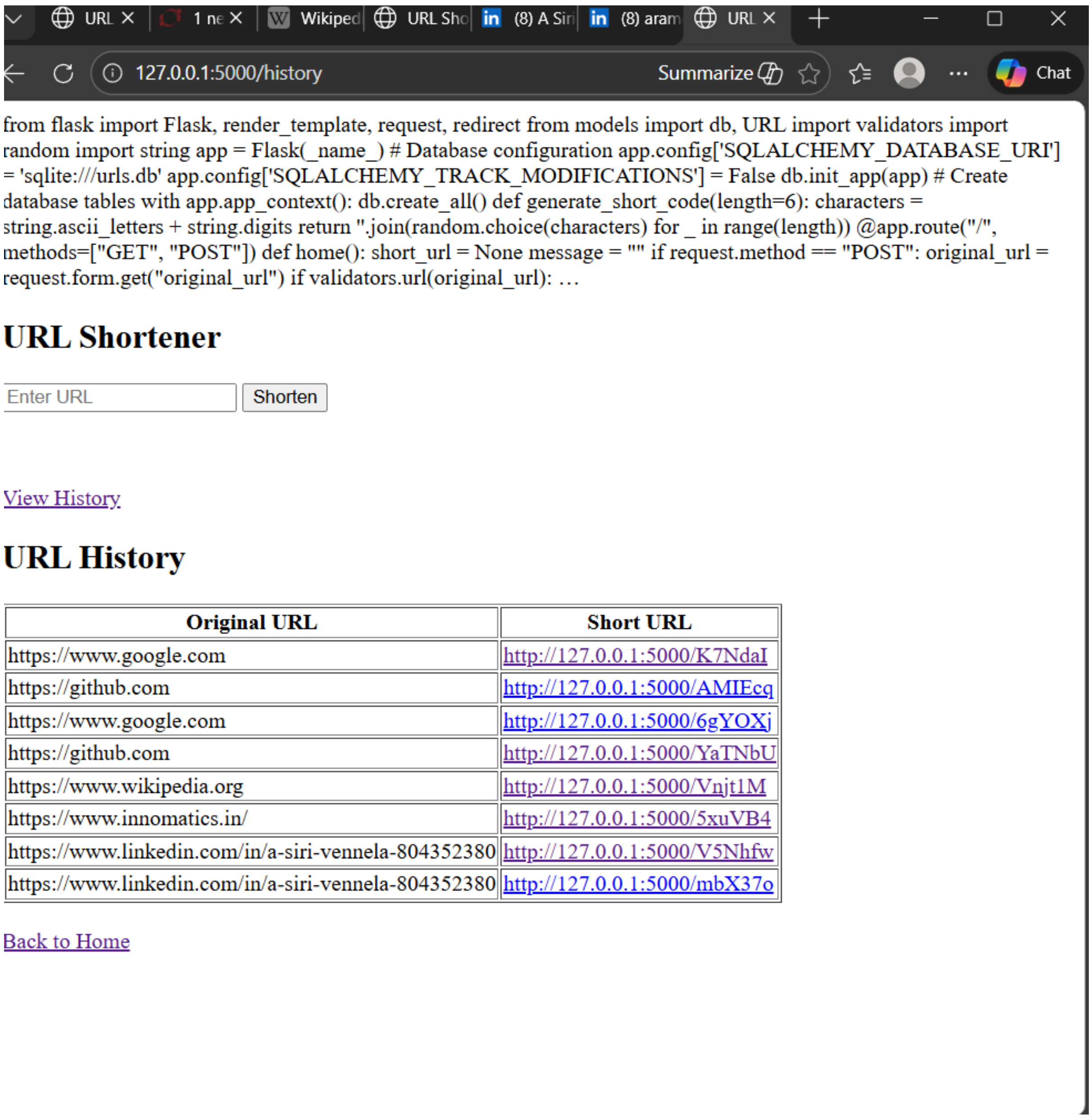
Enter URL | Shorten

Short URL created successfully!

Short URL:

http://127.0.0.1:5000/mbX3

View History



from flask import Flask, render_template, request, redirect from models import db, URL import validators import random import string app = Flask(_name_) # Database configuration app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///urls.db' app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False db.init_app(app) # Create database tables with app.app_context(): db.create_all() def generate_short_code(length=6): characters = string.ascii_letters + string.digits return ''.join(random.choice(characters) for _ in range(length)) @app.route("/", methods=["GET", "POST"]) def home(): short_url = None message = "" if request.method == "POST": original_url = request.form.get("original_url") if validators.url(original_url): …

URL Shortener

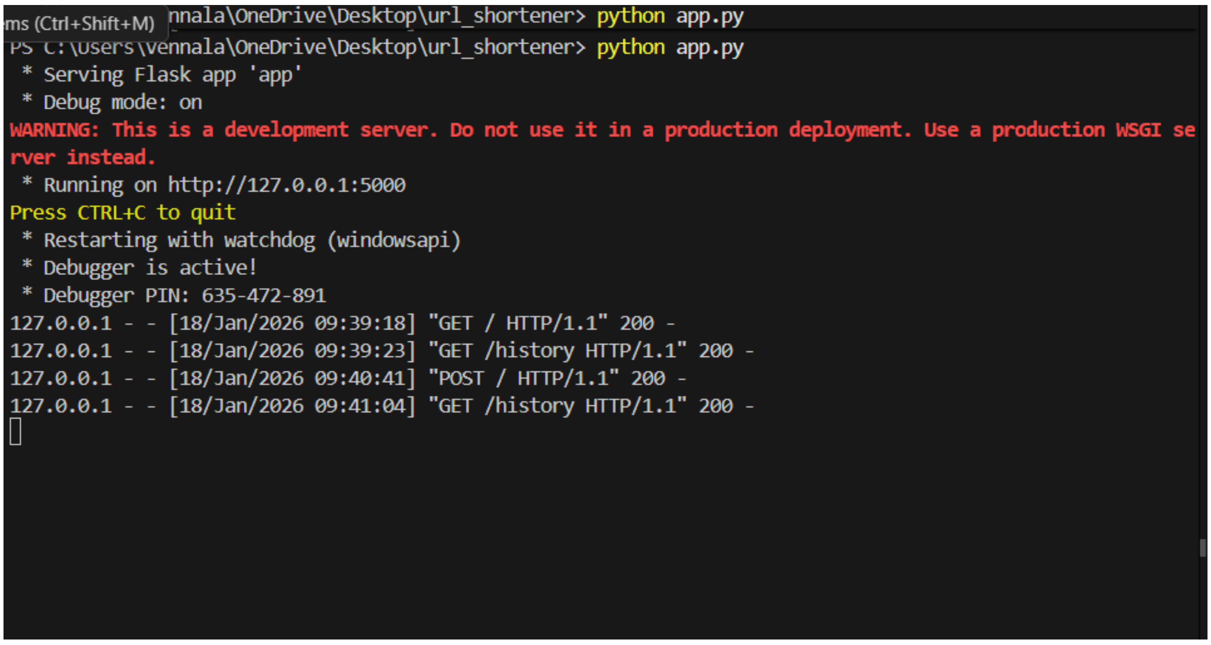
Enter URL | Shorten

View History

URL History

Original URL	Short URL
https://www.google.com	http://127.0.0.1:5000/K7NdaI
https://github.com	http://127.0.0.1:5000/AMIEcq
https://www.google.com	http://127.0.0.1:5000/6gYOXj
https://github.com	http://127.0.0.1:5000/YaTNbU
https://www.wikipedia.org	http://127.0.0.1:5000/Vnjt1M
https://www.innomatics.in/	http://127.0.0.1:5000/5xuVB4
https://www.linkedin.com/in/a-siri-vennela-804352380	http://127.0.0.1:5000/V5Nhfw
https://www.linkedin.com/in/a-siri-vennela-804352380	http://127.0.0.1:5000/mbX37o

Back to Home



```
nnala\OneDrive\Desktop\url_shortener> python app.py
PS C:\Users\Vennala\OneDrive\Desktop\url_shortener> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
rver instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 635-472-891
127.0.0.1 - - [18/Jan/2026 09:39:18] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2026 09:39:23] "GET /history HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2026 09:40:41] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2026 09:41:04] "GET /history HTTP/1.1" 200 -
```