Proof of Concept (PoC) Report

Tool Name: Simple URL Shortener
Author: Piyush Babele
Intern Id: 386

Executive Summary

The **Simple URL Shortener** is a Python-Flask based web application designed to generate short, easy-to-share links from long URLs. Users can input any valid web address, and the application will create a unique short code that redirects back to the original link. This PoC demonstrates how the tool manages URL mapping in a local SQLite database, ensures code uniqueness, and delivers an interactive, user-friendly interface.
This project validates the functionality of URL shortening systems, showing how they can improve usability, save space in communications, and track link usage in extended implementations.

Objective

The goal of this PoC is to validate the functionality of the Simple URL Shortener by:

- Accepting a long URL from the user.
- Generating a unique short code using Base62 encoding (letters and digits).
- Storing and retrieving URL mappings from an SQLite database.
- Redirecting short codes to their corresponding original URLs.
- Demonstrating a functional web interface with a clean design.

Scope

In Scope:

- Shortening of valid URLs.
- Persistent storage in SQLite database.
- Automatic prevention of duplicate short code generation for the same URL.
- Redirection from short URL to original URL.
- Basic HTML/CSS-based interface.

Out of Scope:

- Link analytics (click tracking, geolocation data).
- Expiration dates for links.
- Authentication or user account systems.
- Security hardening against malicious inputs (basic validation only).

Tool Overview

The application uses **Flask** as its backend framework, **SQLite** as its database, and a minimal front-end design via style.css.
The core functionality includes:

- Initializing the database with a urls table.
- Checking if a submitted URL already exists.
- Generating a random 6-character alphanumeric short code if it doesn't.
- Serving the front-end form via index.html.
- Redirecting requests from a short code to the stored long URL.

Key Features:

- Lightweight and fast local deployment.
- Persistent mapping storage in SQLite.
- Avoids duplicate entries for the same URL.
- Simple, mobile-friendly UI.

Requirements

Software Requirements:

- Python (latest version recommended)
- Flask (Python package)
- SQLite3 (comes pre-installed with Python)

Data Requirements:

- database.db — stores URL mappings.
- app.py — main application script.
- templates/index.html — HTML front-end.
- static/style.css — UI styling.

Steps to Run the Tool

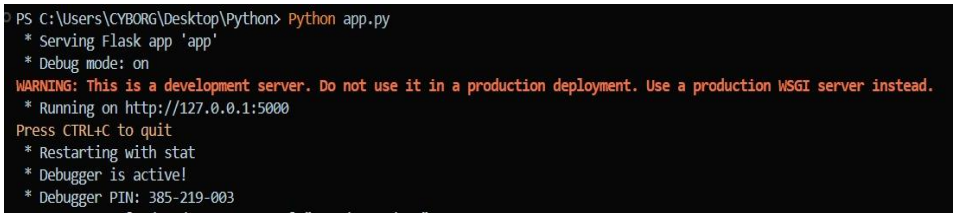
1. **Install Python and Flask**

```
pip install flask
```

2. **Ensure app.py is configured correctly**
   The init_db() function will automatically create the urls table in database.db.

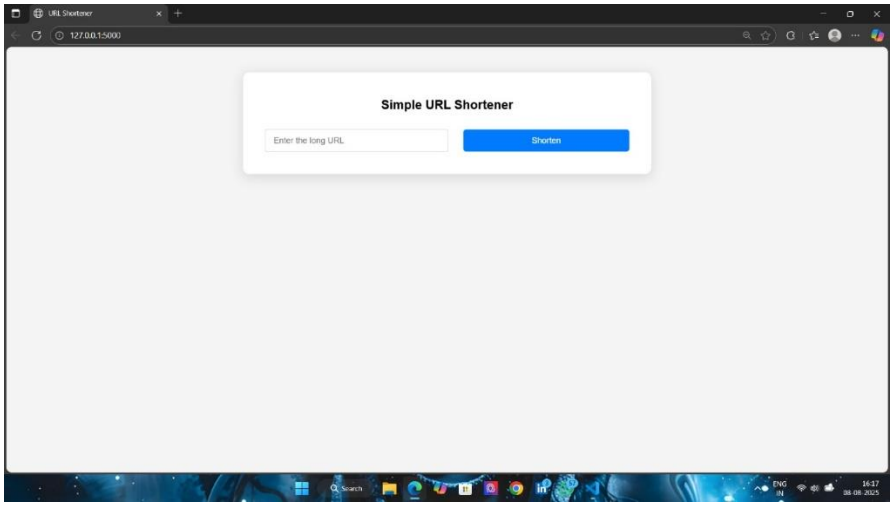
```
PS C:\Users\CYBORG\Desktop\Python> Python app.py
```

3. **Run the Application**



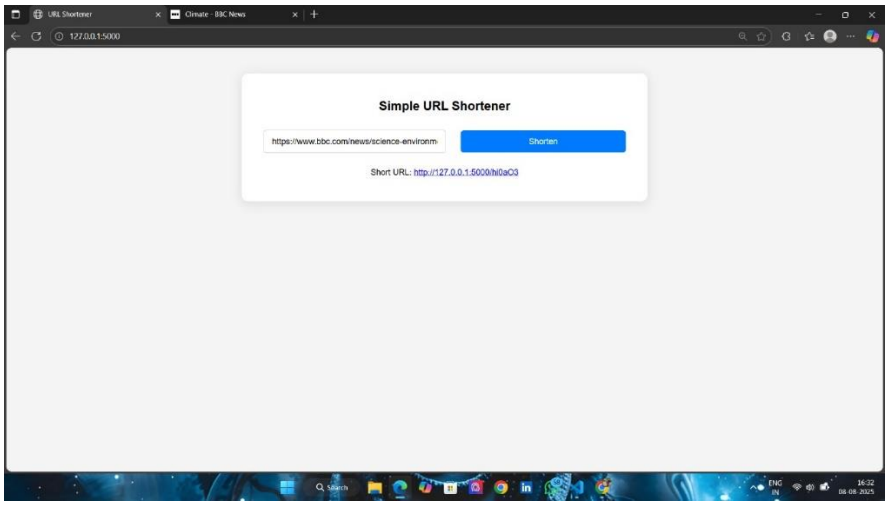
This will start the Flask development server at http://127.0.0.1:5000/.

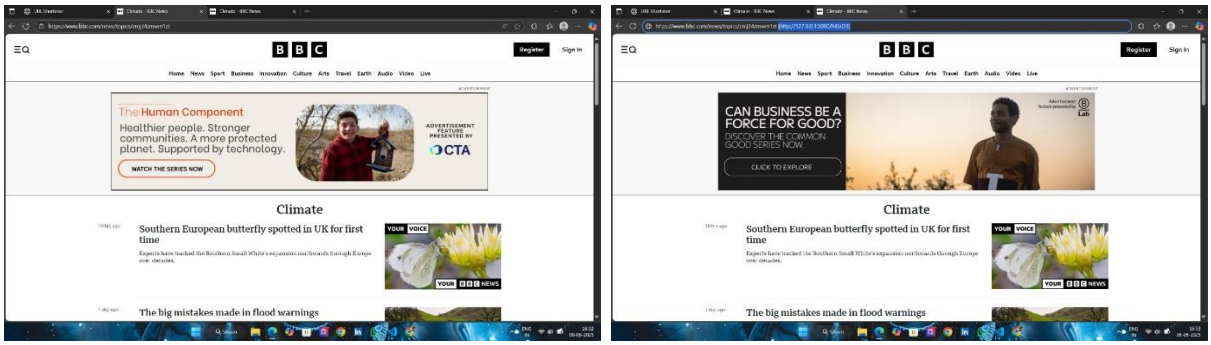
4. **Access the Web Interface**



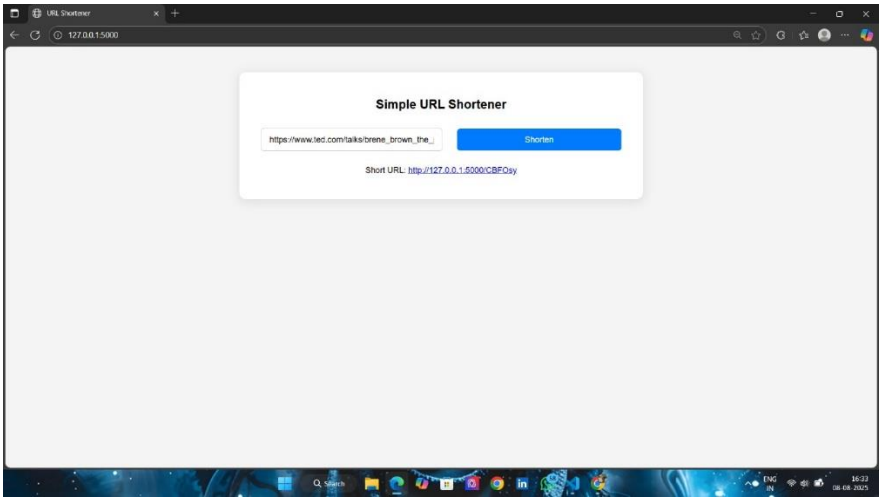
- Open your browser and navigate to http://127.0.0.1:5000/.
- Enter a long URL and click **Shorten**.
- Copy and use the generated short link.

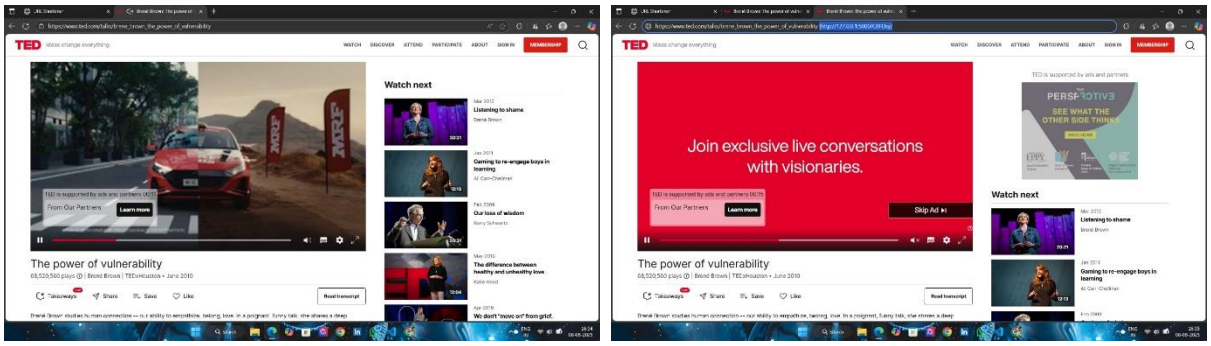
PoC Test Execution





BBC.com Short Link





Ted.com Short Link

Use Case Scenario

A content creator wants to share a long YouTube playlist link on social media. Instead of pasting the lengthy URL, they use this tool to generate a short, clean link. The audience can then easily click and access the original video playlist without clutter in the post.

Advantages

- **Simplicity:** Easy to set up locally.
- **Persistent Storage:** All mappings are saved until explicitly removed.
- **Reusability:** Same link always returns the same short code.
- **UI-Friendly:** Clean, minimalistic interface styled with modern CSS.

Threat Impact Analysis

Potential Risks Without This Tool:

- Sharing long, unattractive URLs that may discourage clicks.
- Risk of breaking links when sent via character-limited platforms.

Possible Security Concerns (if unaddressed):

- **Phishing abuse** — attackers could mask malicious sites under short links.
- **Spam** — automated submissions without rate limits.

Impact Reduction Measures (Future Work):

- Add URL scanning before shortening.
- Implement analytics and abuse detection.
- Allow password-protected short links.

Future Enhancements

- **Custom Short Codes** — allow users to specify their own.
- **Click Analytics** — track usage and sources.
- **Link Expiry** — automatically delete after a set date.
- **API Support** — enable automated shortening via HTTP requests.
- **Security Validation** — integrate with phishing/malware scanners.

Conclusion

This PoC confirms that the **Simple URL Shortener** successfully converts long URLs into short, shareable links and stores them for persistent access. With its lightweight Flask backend, SQLite storage, and simple interface, the tool is functional for small-scale use cases. By adding analytics, security features, and customization, it can scale into a more robust public-facing URL shortening service.