Proof of Concept (PoC) – URL Shortener

1. Title

URL Shortener Web Application – Proof of Concept

---

2. Objective

The objective of this PoC is to demonstrate a simple Python-Flask-based web application that takes a long URL as input and generates a unique shortened version using Base62 encoding. The shortened link redirects to the original URL when accessed.

---

3. Background

Long URLs can be inconvenient to share and remember.
A **URL shortener** provides a compact alias (short code) that maps to the original link. This is widely used in platforms like **bit.ly** and **tinyurl.com** for:

- Reducing character count for social media posts
- Tracking link usage
- Simplifying link sharing

In this PoC, the application uses:

- **Base62 encoding** to generate unique short slugs
- **In-memory dictionary storage** for mapping URLs (for demonstration purposes)
- A minimal **Flask web interface** for user interaction

---

4. Scope

- Accept long URLs via a web form
- Generate a short URL slug using Base62 encoding
- Store the mapping temporarily in memory
- Redirect users from the short URL to the original link

---

5. Methodology

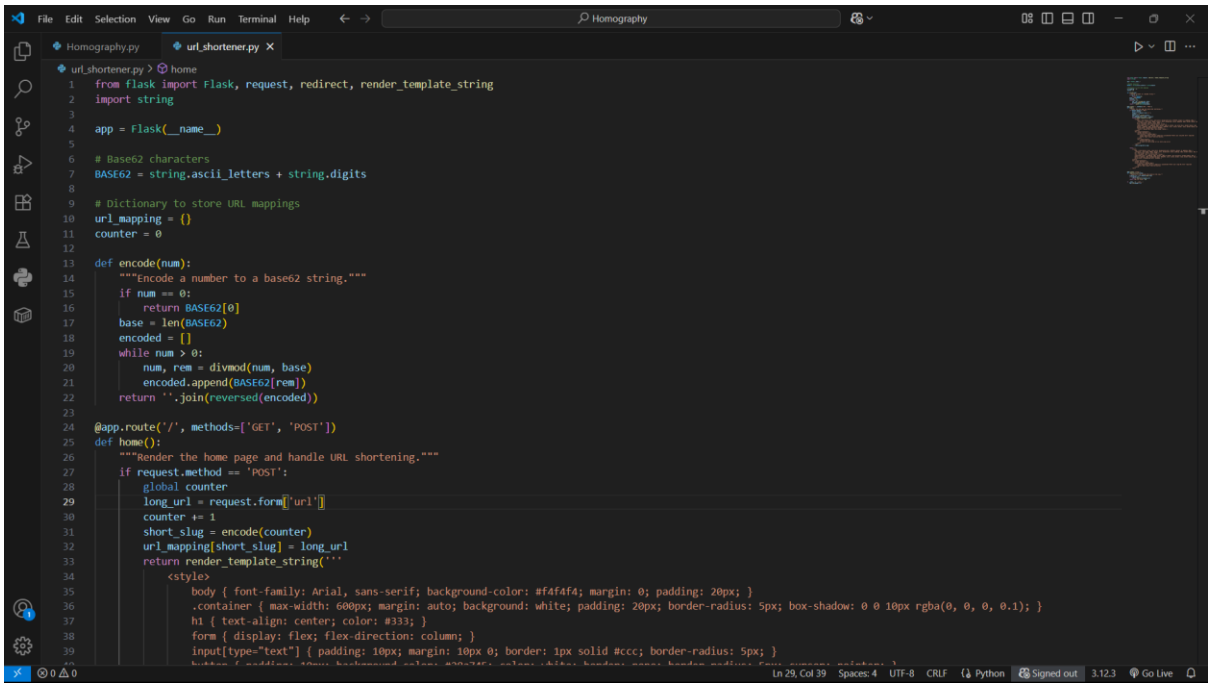
Step 1 – User Input
The user enters a long URL in the web form.

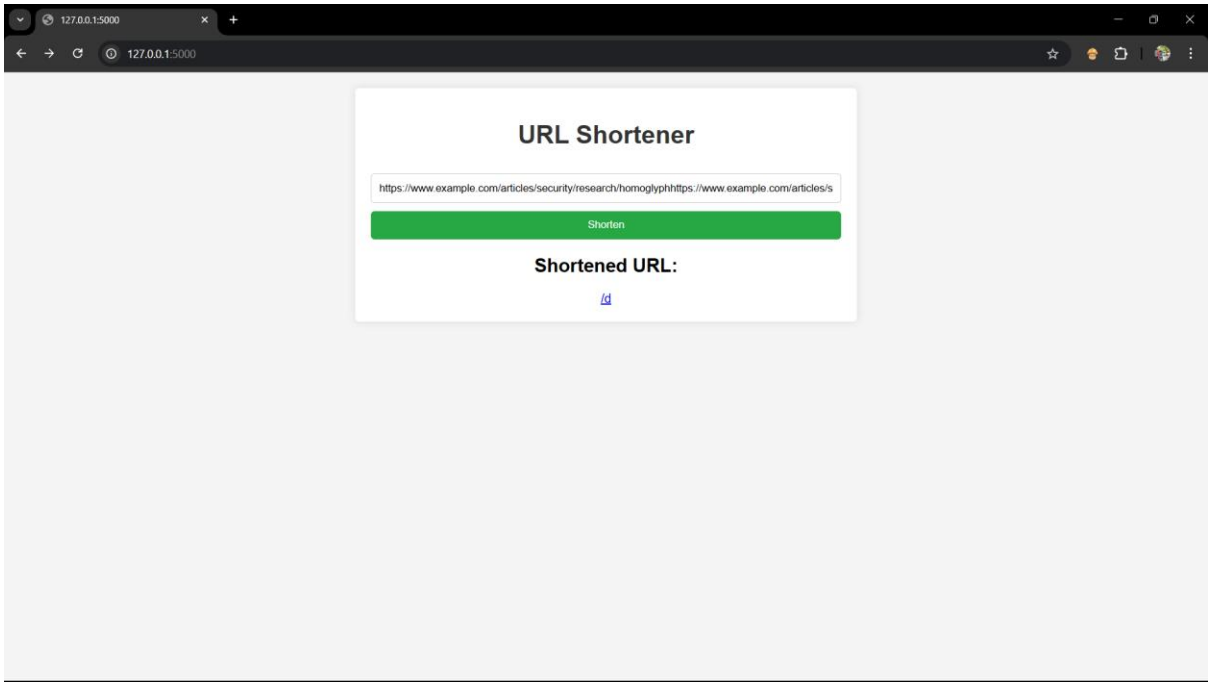
Step 2 – Counter Increment & Encoding
The application maintains a counter that increments for each new link. This counter value is converted into a **Base62 string** to create a short slug.

Step 3 – Mapping Storage
The slug and original URL are stored in an in-memory dictionary (`url_mapping`).

Step 4 – Shortened URL Display
The app returns a clickable shortened link (`/<slug>`).

Step 5 – Redirection
When the short link is visited, Flask retrieves the original URL from the mapping and redirects the user.





7. Example Execution

User Input:

https://www.example.com/articles/security/research/homoglyph

Shortened Output:
/d

Visiting: http://localhost:5000/b

Redirects to:

https://www.example.com/articles/security/research/homoglyph

8. Limitations

- URLs are stored only in memory; data will be lost when the server restarts.
- No validation for malicious URLs.
- No analytics or expiration features.
- No integration with homoglyph detection (can be added for phishing protection).

9. Conclusion

This PoC demonstrates a minimal yet functional URL shortening service using Python's Flask framework. It can be extended with database storage, analytics, user accounts, and phishing prevention techniques such as homoglyph detection.