URL Shortener Web Application

1. Introduction:

In today's digital world, users frequently share web links through emails, messaging platforms, and social media. Many URLs are long, complex, and inconvenient to share or remember. This creates a need for a system that can convert long URLs into shorter, easy-to-share links while still preserving access to the original resource.

A **URL Shortener Web Application** solves this problem by generating a compact alias (short URL) for a given long URL. When users access the short URL, they are automatically redirected to the original URL. Additionally, storing shortened URLs allows users to track previously generated links.

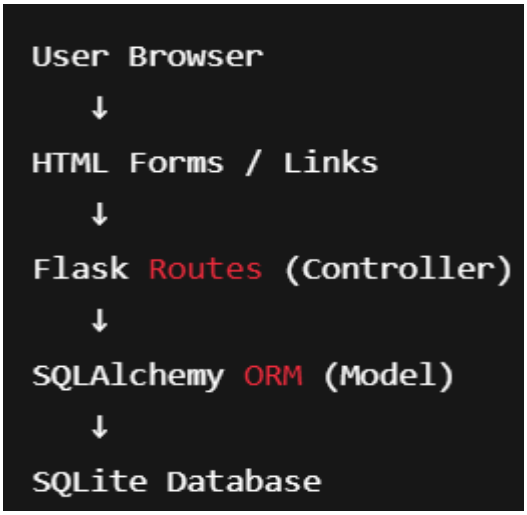
This project focuses on building a **basic URL Shortener web application** using **Flask (Python)** for backend development, **HTML/CSS/Bootstrap** for frontend design, and **SQLAlchemy ORM with SQLite** for persistent storage.

2. Objectives of the Project:

The main objectives of this project are:

- To allow users to input a long URL and generate a shortened version of it.
- To redirect users from the shortened URL to the original URL.
- To store original and shortened URLs in a database for persistence.
- To display a history of all previously shortened URLs.
- To validate user-entered URLs before processing them.
- To understand backend routing, ORM usage, and database interaction using Flask.

3. System Architecture:



4. Database Design:

Only one table is required for this project.

COLUMNS	DATA TYPES	DESCRIPTION
Id	Integer	Unique Identifier
Original_url	String	Long URL entered by user
Short_url	String	Generated Short identifier
Created_at	DateTime	Timestamp

Why a database is required:

- Ensures persistence of URLs even after server restart.
- Enables history tracking.
- Allows reliable redirection from short URL to original URL.
- Supports future enhancements like analytics and user accounts.

5. URL Shortening Logic:

To generate a short URL, a **random alphanumeric string** of fixed length is created.
Logic Used:

- Characters used: a–z, A–Z, 0–9
- Length chosen: **6 characters**

6. URL Validation:

Before shortening, the application validates whether the entered URL is valid.
Approach Used:

- Python's urllib.parse module
- Checks:
  - URL scheme (http or https)
  - Network location (domain)

7. Application Routes and Workflow:

Home Route (/):

Methods: GET, POST

GET request:

Displays input form to enter URL

POST request:

Validates URL
Generates short code
Stores data in database
Displays shortened URL

This route follows the Post–Redirect–Get pattern to avoid duplicate submission.

Redirect Route (/<short_code>)

Method: GET

Accepts short code from URL

Fetches original URL from database

Redirects user using HTTP redirection

Returns 404 if short code is not found

This route implements the core functionality of the URL shortener.

History Route (/history)

Method: GET

Fetches all stored URLs from database

Displays original URL, shortened URL, and date

Allows users to track previously generated links

8. ORM Usage Explanation:

Although raw SQL queries are not written explicitly, SQLAlchemy ORM internally generates SQL commands such as:

- CREATE TABLE
- INSERT
- SELECT

Using ORM:

- Improves security
- Reduces SQL injection risk
- Makes code cleaner and more maintainable

9. Conclusion:

This project successfully demonstrates the implementation of a basic URL shortener using Flask, SQLAlchemy ORM, and SQLite. It provides hands-on experience with backend routing, database interaction, and frontend integration. The project also highlights the importance of proper HTTP methods, persistent storage, and clean architectural design.