Report

URL SHORTENER WEB APPLICATION

[BASIC USERS]

URL Shortener

An URL shortener is a website that **reduces the length of your URL** (Uniform Resource Locator). The idea is to minimize the web page address into something that's easier to remember and track.

There are many URL shorteners on the market today, including Bit.ly, Goog.

How do URL shorteners work?

URL shorteners work by **creating a redirect to your long URL**.

Entering a URL into your internet browser sends an HTTP request to the web server to pull up a specific website. The long and the short URLs are both simply different starting points for an internet browser to get the same destination.

Need URL Shortener?

Sometimes we need to share or send links and this can be tiresome and annoying to copy and paste long URLs. That is where URL shorteners come in. Not only it helps in shortening the URL but it also allows the user to copy the shortened URL with a click of a button.

Problem Statement

1. Create a URL Shortening application that shortens URLs entered by user.
2. Users can also save URLS by coming to the web app.

The project consists of 2 parts:

1. Frontend (done with HTML, CSS and Bootstrap)
2. Backend - Flask (Python)
3. Backend - Database ORM (SQLAlchemy)

Front-End Information

The front-end consists of 2 web pages:

1. **Home Page** - A page will be shown where the user can enter the URL which he/she wants to shorten. After the 'shorten' button is clicked, the shortened URL is displayed in the text-field which the user can copy using the copy button.

2. **History Page** - Containing all the Original URLs along with the Shortened URLs.

Project Workflow

1. Users can enter the URL they want to shorten. After entering a URL, click on the 'Shorten' URL button to display the shortened URL in the following text-field which can be copied by clicking on the copy button.

2. After the 'Shorten' button is clicked, the URL that is entered is saved in our database with the shortened URL. It is saved in the database so that the user can look into the previous URLs he entered in our web-app with their shortened URL.

3. Try to verify the URL entered by the user is correct or not.

HTML files for Templates

* layout.html – It contains the overall designing and styling codes of all the web pages of the application.

* home.html – This file inherits layout.html and with additional functionalities needed for frontend of application. The Home webpage contains a form which accepts the original URL from user and displays the shortened version of the given URL. This app also facilitates to copy the shortened URL and to save for future purposes.

* history.html – This page also inherits layout.html and then puts its own functionalities to act as a database for this application. The History webpage contains all the shortened URLs entered by user in the past. The user can also delete those which shortened URLs if not needed for future use.

• Give CSS styling to all the html files.

• Now, go to app.py and store it in a variable using os module.

• Establish connection between database and backend using SQLAlchemy through configuration of SQLAlchemy ORM and use the variable to give the path to SQLite database.

• Pass the entire application into SQLAlchemy and store it in a variable further used for database.

App Server - Database Connection

- To define our database tables, we'll create a **app.py** file within the core package. Inside that, we can write the following code:

```
db=SQLAlchemy(app)
Migrate(app,db)

class Url(db.Model):
    __tablename__='urlshortener'
    id=db.Column(db.Integer,primary_key=True)
    url=db.Column(db.String(100))
    shorter_url = db.Column(db.String(100))

    def __init__(self,url,shorter_url):
        self.url=url
        self.shorter_url=shorter_url
```

- We created a **Url** class having **urlshortener** tablename with a few fields such as **id** (primary key), **url** (provided by the user), **shorter_url** (generated by us or provided by the user**)**

We then used **Flask-Migrate** commands to migrate the database with the new table.

- flask db init—to initialize the database at the beginning (to be used only once)
- flask db migrate—to migrate the new changes to the database (to be used every time we make changes in the database tables)
- flask db upgrade—to upgrade our database with the new changes (to be used with the migrate command)

After we run the database initialization, we get a new folder called "**migrations**" which holds the setup necessary for Alembic to run migrations against the project. Inside this

folder, we have another folder called “versions”, which will contain the migration scripts as they are created.

Create the Homepage and History Page for Shortening URLs

In this step, we will create a Flask route for the home page, which will allow users to enter a URL that we then save into the database. This route will use the custom short ID provided by the user or generate one on its own, construct the short URL, and then render it as a result.

We need to create a template for the home page that will be served by the home route. This template will have a simple form where a user can input the original URL and submit it. But we’ll not create home.html directly. Using the Template Inheritance concept in Jinja2, we create a templates directory within the core package and create a layout.html file inside that. You can paste the HTML code into that file.

For styling, we’re using Bootstrap here. Most of the code in the preceding block is standard HTML code required for Bootstrap. The <meta> tags provide information for the web browser, the <link> tag links the Bootstrap CSS files, and the <script> tags are links to JavaScript code that allows some additional Bootstrap features.

- The **<title> {% block title %} {% endblock %} </title>** tag allows the inheriting templates to define a custom title.

- The **{% block content %} {% endblock %}** placeholder is where inheriting templates place the content so that all templates have access to this base template, which avoids repetition.

Next, create the home.html file that will extend this layout.html file that define a title, and create a form with a text box named **‘Enter a URL’** for user to input the original URL**.** The URL input entered will allow users to get URLs in shorten form. It has a value of request. form['url'], which stores data in cases of submission failure (that is if the user provides no URL). We then have a submit button named ‘**Shorten the URL’.**

Then we check if the shorter_url variable has any value—this is true if the form submits and the short URL generates successfully. If the condition is true, we display the short URL under the form.

> **home_page ()**

This function is a Flask view function, which is a function decorated using the special **@app.route**() decorator. Its return value gets converted into an HTTP response that an HTTP client, such as a web browser, displays.

Inside the home_page () view function, we accept both **GET and POST** requests by passing methods = ['GET', 'POST'] to the app. route() decorator.

Then if the request is a GET request, it renders homepage.

If request.method == 'POST' condition until the last line. This is where we render a template called home.html, which will contain a form for users to enter a URL to shorten.

If the request is a POST request, we render a template called home.html, which will contain a form for users to enter a URL to shorten and the user can submit a URL as input. This function also checks whether the provided URL is valid or not using **validators** module, if not then display an error message otherwise shorten the URL.

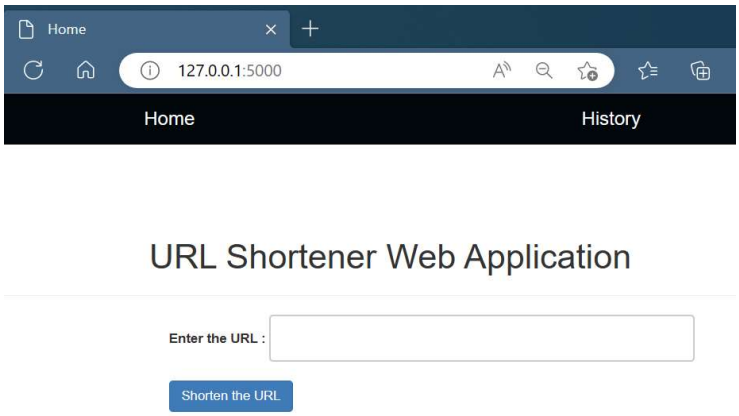
If URL already present in history, then function returns the same shorten URL. Finally, the URL display in a nonwritable textbox and you can now copy the shorten link using copy button beside the textbox.

If the user has submitted an empty form, then we display the message **Please Enter a Valid URL...!!!** and redirect to the home page.

> **history_page ()**

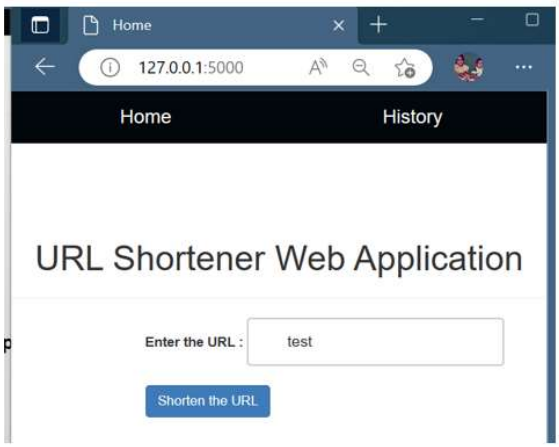
This function fetches all the details from database and show all the entered original URLs with their shorten URLs in the history page. You can also delete a URL if you don’t want to use it anymore.

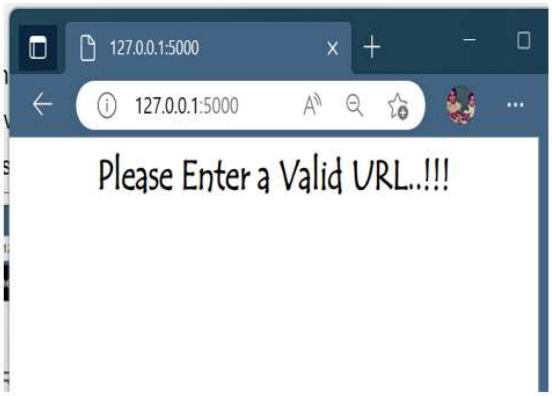
Home Page



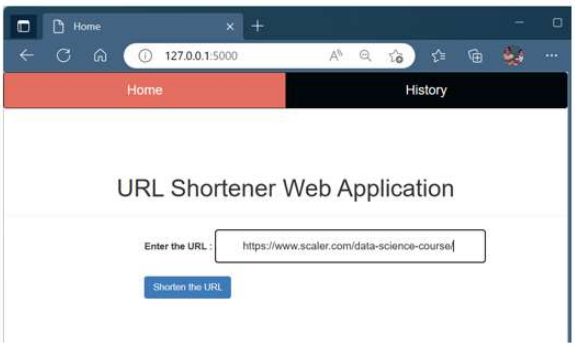
When the user enters the URL to be shortened in the text box, it checks whether the URL is valid or not.

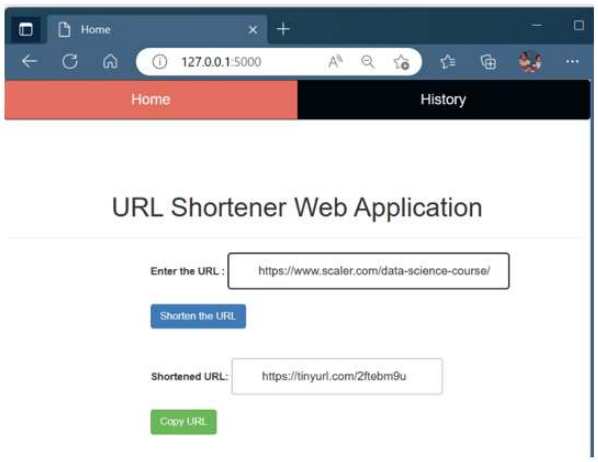
a) If the user entered URL is not valid, it redirects to an error page.





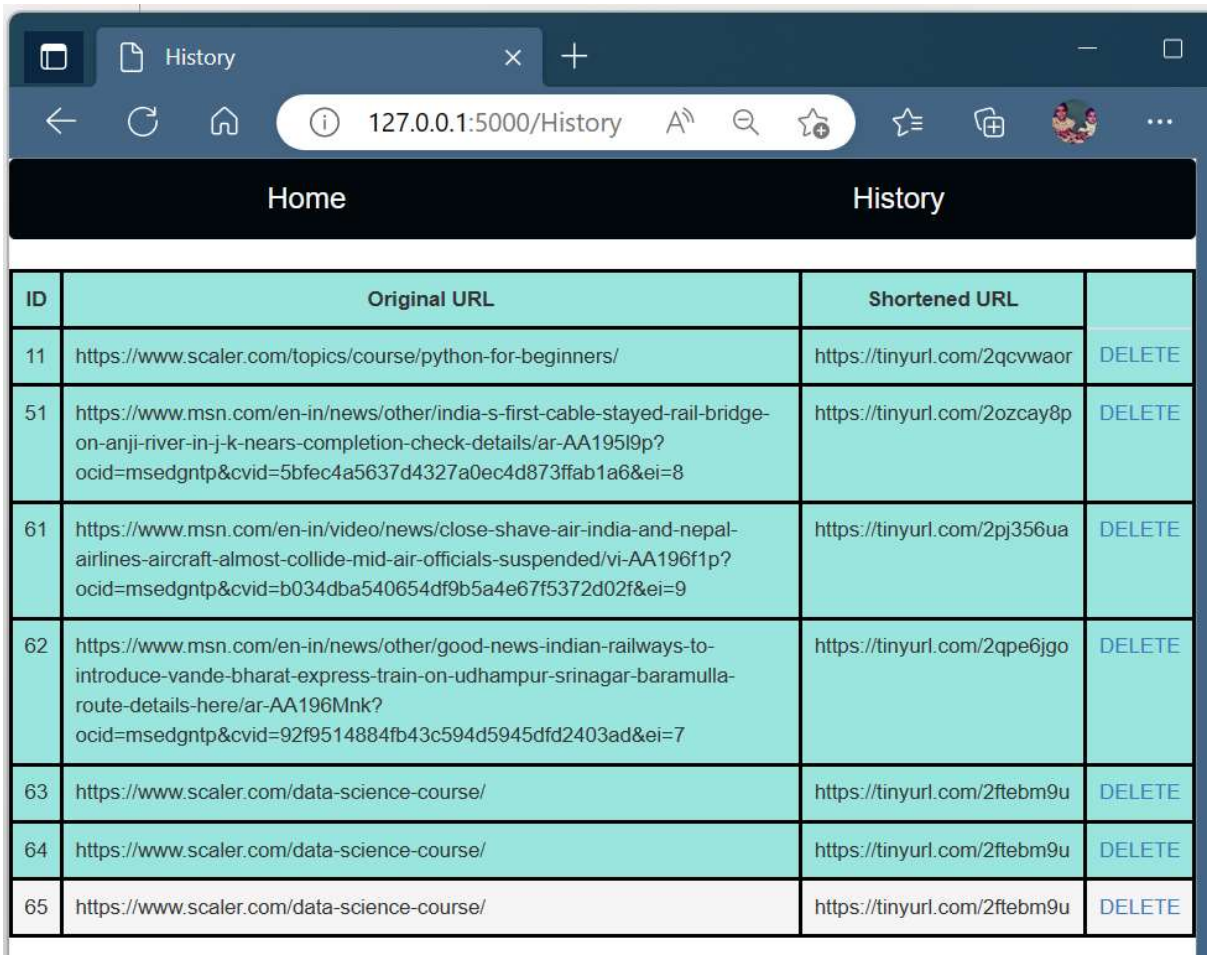
b) If valid, we get shortened URL which can be saved/copied for future use.





In the History page, we get the original URL and shortened URL which we saved for our reference. We can also delete the URL that are not desired.

History Page



ID	Original URL	Shortened URL	
11	https://www.scaler.com/topics/course/python-for-beginners/	https://tinyurl.com/2qcvwaor	DELETE
51	https://www.msn.com/en-in/news/other/india-s-first-cable-stayed-rail-bridge-on-anji-river-in-j-k-nears-completion-check-details/ar-AA195l9p?ocid=msedgntp&cvid=5bfec4a5637d4327a0ec4d873ffab1a6&ei=8	https://tinyurl.com/2ozcay8p	DELETE
61	https://www.msn.com/en-in/video/news/close-shave-air-india-and-nepal-airlines-aircraft-almost-collide-mid-air-officials-suspended/vi-AA196f1p?ocid=msedgntp&cvid=b034dba540654df9b5a4e67f5372d02f&ei=9	https://tinyurl.com/2pj356ua	DELETE
62	https://www.msn.com/en-in/news/other/good-news-indian-railways-to-introduce-vande-bharat-express-train-on-udhampur-srinagar-baramulla-route-details-here/ar-AA196Mnk?ocid=msedgntp&cvid=92f9514884fb43c594d5945dfd2403ad&ei=7	https://tinyurl.com/2qpe6jgo	DELETE
63	https://www.scaler.com/data-science-course/	https://tinyurl.com/2ftebm9u	DELETE
64	https://www.scaler.com/data-science-course/	https://tinyurl.com/2ftebm9u	DELETE
65	https://www.scaler.com/data-science-course/	https://tinyurl.com/2ftebm9u	DELETE