INNOMATICS RESEARCH LABS

Project Report

Submitted to: Kanav Bansal

Submitted By: Ajay Singh

What will our Web app do?

1. As the name suggests, it shortens URLs.
2. Users can also save URLS by coming to the web app.

Why do we 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.

The project consists of 2 parts:

1. Frontend - HTML 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, 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. (Do some googling to find out how to make it possible)

URL Shortener Project - Work on Python Code:

1. Import Necessary Libraries:
   Import Flask, render template, request, redirect, Flask SQL alchemy, Flask migrate, string and os.

2. Create a Flask App:
   Create app using the code: app = Flask(__name__)

3. SQL Alchemy Configuration:
   Flask app config is the SQLALCHEMY_DATABASE_URI key. That is a connection string that tells SQL Alchemy what database to connect to. Create Flask application object, load any config, and then initialize the SQL Alchemy extension class.

4. Define Models:
   Subclass db.Model to define a model class. The db object makes the names in SQL alchemy and sqlalchemy.orm available for convenience, such as db.Column.

5. Create the Tables:
   After all models and tables are defined, call SQLAlchemy.create_all() to create the table schema in the database. This requires an application context. Since we’re not in a request at this point, create one manually. If we define models in other modules, we must import them before calling create_all, otherwise SQL Alchemy will not know about them.

6. Query the Data:
   Within a Flask view or CLI command, we can use db.session to execute queries and modify model data.
   SQL Alchemy automatically defines an __init__ method for each model that assigns any keyword arguments to corresponding database columns and other attributes.

7. Create a Shorten URL Characters:
   We need the minimum character for making short URL. We used here seven characters with uppercase, lowercase and digits. We used seven characters because of in future our short URL do not conflict with another link.

8. Create end points:
   Basically, the "endpoint" is an identifier that is used in determining what logical unit of our code should handle the request. Normally, an endpoint is just the name of a view function.

9. Run the application:
   Finally run the app in local host with port no :5000 along with debug=True.

URL Shortener Project - Work on HTML Code:

1. Create the Base HTML:
   Define the base app. we can make use of base templates and the extending of templates to reduce repetitive markup. We can refactor some of the markups we have created already in our Flask application in order to make use of templates that extend from other templates.

2. Create the URL page:
   Creating URL page, we input the any website link to make short link. For this we use form tag by using this we run the POST method.

3. Create the History page:
   This particular page user can see all old URL link here by click history nav bar, user reach the whole link which is user already used or before entered the link.

Facing the Issues:

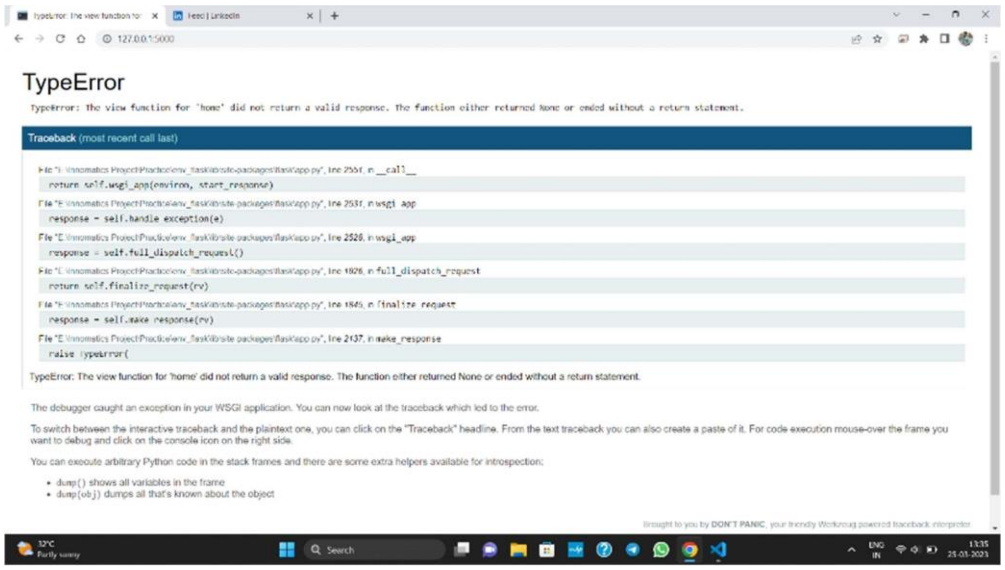
1. How to decide the minimum character?
   I see one YouTube video. In future we face the more website and that time our application not face the conflict issue so I decide the minimum characters is seven.

2. Creating mix character like digits and letters:
   Define function and put the string ascii lower and upper characters also string digit.

3. Display the short URL in same page.
   After lot off do effort for display URL in same page. For that part make the route and giving name /display/<url> and return the same template.

4. Creating a page of History:
   Here after do research, found the same template which I used last time in the Note Taking Application. Simply copy here some code here but not support some codes and giving me issues. I couldn't assign the row with particular id no. for these issues used here Jinja for loop and fix the issue.

5. Some Screenshot of error:



TypeError

```
TypeError: The view function for 'home' did not return a valid response. The function either returned None or ended without a return statement.
```

Traceback (most recent call last)

```
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2551, in __call__
  return self.wsgi_app(environ, start_response)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2531, in wsgi_app
  response = self.handle_exception(e)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2528, in wsgi_app
  response = self.full_dispatch_request()
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1826, in full_dispatch_request
  return self.finalize_request(rv)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1845, in finalize_request
  response = self.make_response(rv)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2137, in make_response
  raise TypeError(
TypeError: The view function for 'home' did not return a valid response. The function either returned None or ended without a return statement.
```

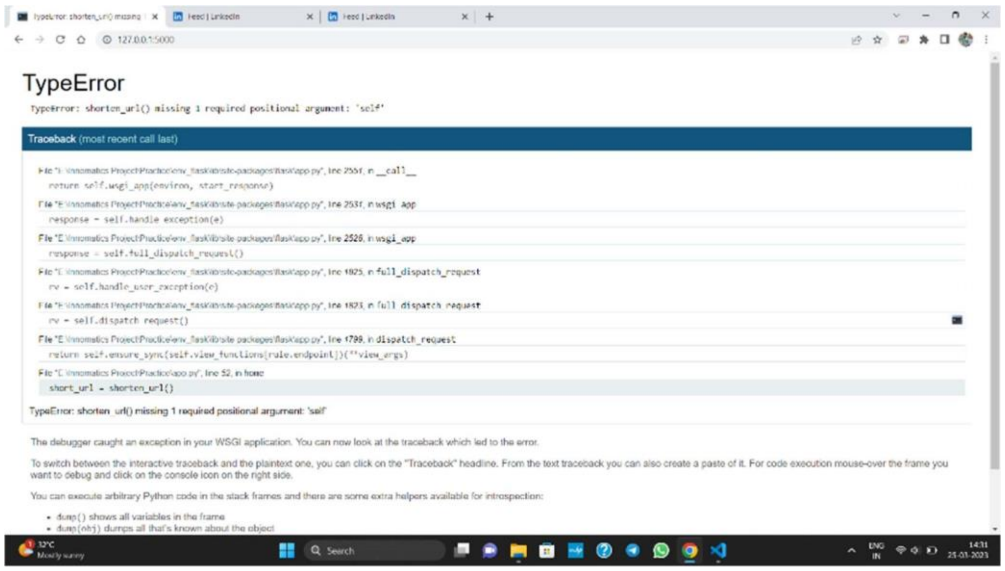
The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- dump() shows all variables in the frame
- dump(obj) dumps all that's known about the object

Brought to you by DON'T PANIC, your friendly Werkzeug powered traceback interpreter.



TypeError

```
TypeError: shorten_url() missing 1 required positional argument: 'self'
```

Traceback (most recent call last)

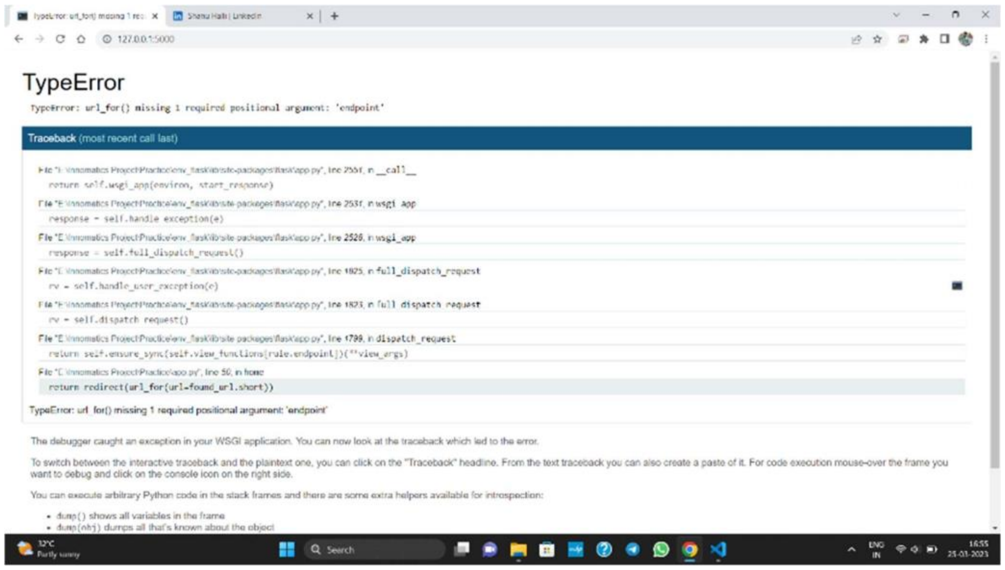
```
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2551, in __call__
  return self.wsgi_app(environ, start_response)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2531, in wsgi_app
  response = self.handle_exception(e)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2528, in wsgi_app
  response = self.full_dispatch_request()
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1825, in full_dispatch_request
  rv = self.handle_user_exception(e)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1823, in full_dispatch_request
  rv = self.dispatch_request()
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1799, in dispatch_request
  return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
File "E:\Innomatics Project\Practice\app.py", line 52, in home
  short_url = shorten_url()
TypeError: shorten_url() missing 1 required positional argument: 'self'
```

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- dump() shows all variables in the frame
- dump(obj) dumps all that's known about the object



TypeError

```
TypeError: url_for() missing 1 required positional argument: 'endpoint'
```

Traceback (most recent call last)

```
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2551, in __call__
  return self.wsgi_app(environ, start_response)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2531, in wsgi_app
  response = self.handle_exception(e)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 2528, in wsgi_app
  response = self.full_dispatch_request()
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1825, in full_dispatch_request
  rv = self.handle_user_exception(e)
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1823, in full_dispatch_request
  rv = self.dispatch_request()
File "E:\Innomatics Project\Practice\env_flask\lib\site-packages\flask\app.py", line 1799, in dispatch_request
  return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
File "E:\Innomatics Project\Practice\app.py", line 50, in home
  return redirect(url_for(url=found_url.short))
TypeError: url_for() missing 1 required positional argument: 'endpoint'
```

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- dump() shows all variables in the frame
- dump(obj) dumps all that's known about the object

THE END!!