REPORT FOR CREATION OF THE URL SHORTENER APP

The attached code describes the creation of a URL shortener app using Python Flask as the app server and SQLAlchemy for the database and HTML and CSS to structure and style the web app. The URL web app acts by taking in a long and complex URL and generates a shorter and more manageable URL that redirects the user to the site of the long URL when accessed.

For the creation of this app, the following modules and packages were imported.

```
from flask import Flask, render_template,request, redirect
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
import os
import uuid
import re
```

The ‘render_template’ function was imported to generate an output from the created html pages to show specific web pages. The ‘request’ is used to obtain data from the user through the webpage and send it to the flask backend for processing. The ‘redirect’ is used to send users to a different URL from the one clicked.The SQLAlchemy is an Object Relational Mapper that links the python objects to a database relation.It helps the Python backend to communicate with relational databases by constructing complex SQL queries in a pythonic way.The ‘Migrate’ is used to generate scripts that handles and controls changes in a database schema, aiding management of database across different environments. The ‘os’ module is used to retrieve the base directory of the database file created. The ‘uuid’ was imported to aid the generation of a set of random characters to be used in the app logic. The ‘re’ is used as a validation tool in this app.

```
app.config['SQLALCHEMY_DATABASE_URI']=
'sqlite:///'+os.path.join(basedir,'data.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False
```

The app configuration was carried out by generating and setting the SQLite database file Uniform Resource Identifier.The track modifications configuration was set to false to prevent tracking of the modifications to the database because this is a smaller scale app.

```
class URL(db.Model):
   __tablename__= "urls"
   id=db.Column(db.Integer,primary_key=True)
   long_url=db.Column(db.String(500))
   short_url=db.Column(db.String(20))
```

This shows the creation of a class to denote a database that inherits from the ‘db,Model, which defines the model for a SQLite database’ with a table named ‘urls’. The three columns created id,long_url and short_url are made by inheriting from the ‘db.Column’, and setting the desired data type for each column.

```
def __init__(self,long_url,short_url):
       self.long_url=long_url
       self.short_url=short_url

   def __repr__(self):
       return "Long Url-{}and Short_Url-{}".format(self.long_url,self.short_url)
```

The first method is created to initialize the instances of the class URL and they are called whenever a new instance of the class is made.It takes in long_url and short_url as arguments, which are treated as variables in instances of the class. The second method is a representation of new instances in the class,it is made for logging purposes and returns a string containing the long url and short url for each instance of the class created.

```
def is_valid_url(url):
   pattern = re.compile(r'^https?://(?:www\.)?.+\..+$')
   return bool(pattern.match(url))
```

This function is defined to validate all URLs entered using a regex compile function.The regex pattern matches URLs that starts with ‘http’ or ‘https’, followed optionally by a ‘www’ subdomain and ending with a domain name that has at least one character. The function returns a boolean value, True for only urls that match with the regex pattern and False for those that don’t.

```
@app.route('/', methods=['GET','POST'])
def index():
   if request.method == 'POST':
       long_url=request.form['url_address']
       if not is_valid_url(long_url):
           return render_template('home.html', error='Invalid URL')
       short_url=str(uuid.uuid4())[:7]
       new_url=URL(long_url=long_url, short_url=short_url)
       db.session.add(new_url)
       db.session.commit()
       return render_template('home.html', short_url=short_url)
   else:
       return render_template('home.html')
```

The above code uses a decorator to create a route to an endpoint in the flask app object, which takes in a GET and POST HTTP request method. The endpoint takes in a function that takes in a url_address from an html form in the ‘home.html’, it also validates it using the ‘is_valid_url’ function created earlier. The function also takes in a short url generated using the ‘uuid’ module that creates 7 random characters and is converted to a string. A new_url is created by setting the urls in the URL class to the urls received in the function. The newly generated urls are then added to the database and changes committed. The home.html containing the generated short_url to be clicked is then rendered using the ‘render_template’ function. If the request method is POST,the above described function is carried out, else only the ‘home.html’ page is rendered.

```
@app.route('/<short_url>')
def redirect_url(short_url):
   url=URL.query.filter_by(short_url=short_url).first_or_404()
   return redirect(url.long_url)
```

This code is for the ‘/’ endpoint and it takes in the short_url variable in and creates a function that redirects it to long_url. It does this by querying the database and filtering it by the short_url, then returning the specific long_url that matches the short url. A first_or_404() returns a 404 error when the short_url doesn’t have a corresponding long_url in the database. Basically, it shows a short url on the ‘/’ page, that when clicked, sends a user to the long_url.

```
@app.route('/history')
def history():
   urls = URL.query.all()
   return render_template('history.html', urls=urls)
```

This code uses a decorator to create an application endpoint that is defined by a function that queries a URL database and returns all available urls. This works by the route retrieves all instances of the URL object and passess them into a history.html template which displays all long_urls entered and short urls generated.

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>{% block title%} {%endblock%}</title>
   <styles> {% block styles %} {% endblock %} </styles>
 </head>
 <body>
```

```
   <nav>
     <a href="{{url_for('index')}}">Home </a>
     <a href="{{url_for('history')}}">History</a>
   </nav>
   {%block body%} {%endblock%}
 </body>
</html>
```

The above block of code is a layout.html code that creates a base template for other child templates in the web app to inherit from. It sets up the core parts of the html page for consistency and uses jinja templating to create blocks for the titles, styles and body specific to each page. It also displays a navigation bar that helps users navigate throughout the pages of the web app. This base template aids ease of operation and consistency as it allows other web pages to use the same layout of the base template.

```
<form action="/" method="post">
 <input
   type="url"
   name="url_address"
   id="url_address"
   placeholder="Enter your url"
 />
 <label for="url">URL:</label>
 <input type="submit" value="Shorten" />
</form>
```

This is a form in the home.html page that creates a form where the user inputs the long url address that is used in the backend and also adds a submit button which the user uses to send the html data to the flask.

```
{% if short_url %}
 <p>
   Shortened URL:
   <a href="{{ url_for('redirect_url', short_url=short_url) }}"
     >{{ short_url }}</a
   >
 </p>
{% endif %}
```

If the short url generated in the backend is clicked on the html page it accesses a link which uses the redirect_url function to send it to the url of the original long_url. This code block also uses jinja templating which aids writing of pythonic type code on the html template.

```
{%block body%}
<h1>URL Shortener - History</h1>
<table>
 <thead>
   <tr>
     <th>Long URL</th>
     <th>Short URL</th>
   </tr>
 </thead>
 <tbody>
   {% for url in urls %}
   <tr>
     <td>{{ url.long_url }}</td>
     <td>
       <a href="{{ url_for('redirect_url', short_url=url.short_url) }}"
         >{{ url.short_url }}</a
       >
     </td>
   </tr>
   {% endfor %}
 </tbody>
</table>
</div>
{%endblock%}
```

This is a history.html code block that creates a dataframe using the entered long and short urls. The <table> tag indicates creation of a table in html. The <thead> tag sets the first row of the table which is the header. The <tr> tag represents a row of cells in the table, <th> represents the header of each column in the table. The <tbody> sets the rest of the table and uses the <td> tag to add data to each cell in the column.

In conclusion, this web app functions by receiving a long URL from a user, generates a short URL and redirects a subsequent user to the longer URL when the short URL is accessed.