# Your Blog

## Introduction

The aim of this coursework was to design and implement a prototype of a web application using Python Flask. ‘Your Blog’ is a web application that uses login system to save user’s thoughts in posts. To achieve this, I used the virtual server Levinux and the command line logged in using SSH. The web app was built using micro-framework Python Flask, which has many useful tools, such as debugging tools or templating using Jinja2.

## Design

The URL hierarchy was designed in the following way: welcome page, login page, index page, error page and search results page. This was achieved by repurposing free Appi template [1]. I researched other blog sites, such as Blogger [2] to compare the design.

The welcome page leads to the login page, where user can use the username ‘admin’ and password ‘admin’ to log into the post’s interface. User can then add new post or delete it. Navbar takes the user back to the welcome page by clicking home or he can use ‘log out’ link to log out of the post’s interface.

Logo positioned in the left top corner is a link to take user to the post’s interface and was downloaded from the internet. It is covered by the Creative Commons licence, therefore needs owner’s permission, if the app was deployed on the publicly accessible server. The images, favicon and CSS are accessed by the Flask “url\_for” function. Search bar allows user to display the posts by searching through titles.

### Flask routing, redirects, template inheritance, sessions, requests, g

The app is deployed by ‘app.py’. Each route (web address) uses appropriate template to generate HTML page and loads the data with SQlite. Folder src contains file ‘app.py’ (that deploys the app) and static and templates folders. Static folder contains CSS, fonts, images, JavaScript and favicon. Templates folder stores all HTML pages.

The app.py file contains different trailing slashes to redirect user to different routes: welcome, login, logout, search, add and delete. HTML is generated by render\_template and return\_redirect function. The HTML templates include Jinja2 variable tags indicated by {{ }} using conditional behaviour.

The login route uses methods and request arguments to request the information from a form and error message if no or wrong details were added. Error message was pasted to base.html page by “if” statement. The Flask extension - Flask Login for managing sessions was used to protect the route to be accessed by unauthorised user by ‘login\_required’ decorator and indicate that user is logged in through a session key.

The logout session pop the value of true out of the session object and replace it with none and deletes the session key. The secret key variable – encryption key was generated by 'os' function. This should be moved to a separate file.

Flash messages were added to provide user feedback using for loop. The base.html template functions as a parent template and extends html for all the child templates using block content.

### Testing

Unit testing was added to test responses and reduce bugs in the code by creating a test.py file. ‘test.py’ tests functionality of the response status code, if the index page requires the login, logout page logouts user, posts’ data is displayed and if data and error data returned contain text from the page. Test.py is missing tests for search page. The responses were checked using IPython.

### Database

‘sql.py’ file was added to import sqlite3 database and create a 'sample.db' file. 'sample.db' file created the posts table. SQLite database browser was used to check the data. G object was used to store database connection.

The database was later switched to SQLAlchemy; data was imported using objects in ‘models.py’ to new database 'posts.db'. ID table as primary key was added to the posts. The configuration was moved to ‘config.py’ file and environment variable class for development added. This allowed for the app to have debug set to ‘False’ for deployment of the app to not expose the server to the outside world. Extension Flask-Bcrypt was added to hash the password and users added to the ‘models.py’ file.

Virtual environment tool was used to keep the required dependencies. However due to an error, at certain point virtual environment could not be activated anymore, had to be deleted and SQLAlchemy could not be used giving ‘Memory error’ when downloading. Therefore, app was rolled back to the SQLite database.

## Enhancements

To improve user experience and overall design, I would include:

* user registration,
* forgotten username/password function,
* ability to sign in with Facebook, Gmail account, etc.,
* ability to assign a URL address to publish the blog online,
* customization of the blog by adding skin templates,
* ability to add media, such as video or photos, WYSIWYG editor,
* edit/revert to draft, button,
* feedback form,
* popular/latest blogs,
* comments on public posts,
* social media sharing on public posts,
* ability to subscribe to users
* dates on posts,
* improved search with suggestions
* stats, comments, campaigns, etc. to get more readers,
* gadgets such as navbar, header, footer etc. to change the templates,
* settings to change user’s details,
* help section,
* terms of service, privacy, content policy.

## Critical evaluation

Your Blog web-app meets the criteria of the coursework. It demonstrates appropriate use of routing, static files, requests, redirects, responses, templates, sessions, logging, testing, CSS, JavaScript and data storage. It is missing multiple users and password is hard-coded and not hashed which means the app is not secured.

Comparing to different blogging website, such as Blogger, it is missing lots of functionality, which is described in enhancements. It is missing a script for latest and popular blogs on the welcome page and function to edit the posts. User should not be allowed to add posts with same titles or an empty post and search should have an error message added for wrong or no data searched.

## Personal evaluation

By completing this coursework, I learnt to use the dynamic programming language Python and its library. I constructed a web application that can run across different software platforms.

I learnt to use SQLite to build databases, use IPython and Database Browser for SQLite. I familiarised myself with sessions, requests, responses and users.

I should have delete unused files and code. Although my prototype is working to basic level, it still needs lots of improvements to be suitable for deployment on online accessible server.

To improve my Flask knowledge, I followed tutorial videos [3], [4]. To build my app I used the YouTube video [5] and tutorial [6]. To troubleshoot I used Stack Overflow, Jinja documentation [7] and Python documentation [8].

## References

[1] "Appi - Free Bootstrap Theme". *Blacktie.co/demo/appi/index.html*. N.p., 2016. Web. 29 Nov. 2016.

[2] "Blogger". *Blogger.com/blogger.g#welcome*. N.p., 2016. Web. 29 Nov. 2016.

[3] "The Build A SAAS App With Flask Course - Udemy". *Udemy.com/the-build-a-saas-app-with-flask-course/*. N.p., 2016. Web. 29 Nov. 2016.

[4] "Flask Tutorial - 2. Creating Our First App". *YouTube/watch?v=X1k36Pt-vzQ&list=PLei96ZX\_m9sWQco3fwtSMqyGL-JDQo28l&index=2*. N.p., 2016. Web. 29 Nov. 2016.

[5] "Discover Flask, Part 1 - Setting Up A Static Site, Hello World". *YouTube/watch?v=WfpFUmV1d0w*. N.p., 2016. Web. 29 Nov. 2016.

[6] "Student Record App Using Flask". *bootniyas.wordpress.com/2014/07/23/student-record-app-using-flask/*. N.p., 2016. Web. 29 Nov. 2016.

[7] "Welcome To Jinja2 — Jinja2 Documentation (2.8-Dev)". *Jinja.pocoo.org/docs/dev/*. N.p., 2016. Web. 29 Nov. 2016.

[8] "Extensions Registry | Flask (A Python Microframework)". *Flask.pocoo.org/extensions/*. N.p., 2016. Web. 29 Nov. 2016.