*CEP T3 Project Documentation*

**Mission statement**

*To create a website which allows users to post their blogs and at the same time, view those that others have posted. The inspiration for this website is taken from* [*https://www.tumblr.com*](https://www.tumblr.com)*.*

**Features**

1. To design all the html pages, I didn’t use a static CSS file but instead used tailwind CSS which is something like bootstrap and allows you to handle all the styling within the HTML file. Instead of assigning each element a class or an id and writing CSS code for that element individually, tailwind CSS allows you to assign predefined classes to elements to style them. These predefined classes were very intuitive and easy to use (eg. pt-0 for padding top 0, py-1 for padding top and bottom 0.25rem etc.) I had originally coded the CSS file for the first few pages but then realized that it would not be feasible to hardcode CSS for 16 over HTML pages. I also noticed that my pages rendered noticeably faster with tailwind CSS (since the file is lighter without a dedicated CSS file). Using tailwind CSS also exposed me to how websites are built on a large scale, a step above hardcoding CSS for smaller websites. For images, I used the source.unsplash API to generate different but relevant images every time a page is rendered. References: <https://tailwindcss.com>, <https://tailwindcss.com/docs>, <https://tailblocks.cc>, <https://github.com/tailwindtoolbox/Minimal-Blog>, <https://www.tailwindtoolbox.com>, <https://source.unsplash.com>
2. Adding on, I implemented the use of blog handling, allowing a blog to be posted, displayed, edited, and deleted. Originally, while starting off this project, I was stuck in a major dilemma. Since I can have more than 1 post and I don’t know how many posts there are, it was almost certain that a for loop would need to be implemented for the “Blogs” page to display all the blogs. However, I also wanted the “learn more” button to bring the user to a single page which would display the contents of the post (‘display\_post.html’). Knowing that a for loop was being used, I was stuck wondering how to make each button lead to different links in the same for loop. Upon further research, I found that variables can be passed into the URL so I decided to assign a unique id to each post and push this unique id to your\_blogs.html where it would then be assigned to the Learn More button and retrieved by display\_post.html, which would then display the contents of the post with that id. This was a lifesaver and I also think that this is one of Flask’s very useful features that I’ve discovered. In the create\_post page, I also implemented hidden inputs for date, author name and id, which can be automatically assigned and does not have to be keyed in by the user. (Date is retrieved from the datetime module, name is retrieved from global variable user\_info and id is stored in a count.txt file which is incremented every time a blog is posted, ensuring that each blog gets its own unique id). In create\_post.html, I also assigned a value to each input, to ensure that in the case of an error where the form is submitted but the data is not loaded onto the json file, the data is not lost. For the edit\_post route, I ensured that the values of all the inputs are predefined with the first unedited data in blog\_data.json, allowing the blog to be edited and not re-written. For the delete\_post route, I made sure that the method is limited to POST only to ensure that it is not accidentally deleted by entering a wrong URL. The form also shows a conformation message before deleting a post. Finally, I implemented abort(404) ‘s to be called in times of need. Realistically, these error handlers should never be called unless and until somebody tampers with the json files or the app code while the server is running, so they’re there just as a safety precaution. References: <https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3>, <https://flask.palletsprojects.com/en/2.0.x/>
3. I implemented user authentication for the blogsite, including a sign\_up, login and logout route. To achieve this, I used global variables such logged\_in and user\_info to determine the state of the user website and accordingly, what is and is not visible to them. Upon signing up, the user’s data is stored in auth\_data.json where their name, email and password is tagged to their email since its unique to the user. The login route then validates the user’s credentials and logs them in if they are correct, by changing the global variable logged\_in to ‘True’ and assigning the user’s credentials to the global variable user\_info.
4. I also implemented a profile handling system which allows the user to view, edit and delete their profile. Each user is assigned default values for their profile which are stored in profile\_data.json when they login for the very first time. They can then edit their profile data, similar to how they would edit a blogpost. The key to this is the global variable ‘user\_info’ which allows the current active user’s name, email and password to be accessed by any route in the app. Once logged out, the value of this variable becomes ‘None’ and the logged\_in variable becomes ‘False’. The delete\_profile route again only accepts POST requests and it removes the user’s data from both profile\_data.json and auth.json, deleting both the user’s profile and account.

**Bottom Line**

To test out the usability of my website, I asked my parents to navigate through my app and give me some feedback (they had not tried out the app before and didn’t have access to the code). They pointed out areas where the styling of elements could be better, and the usability of the app could be improved. I found these comments extremely valuable as I would not have been able to figure out these inconsistences as the one coding the website. Overall, the experience of creating an application and writing both frontend and backend code was one that was tough but fulfilling. Despite the many roadblocks and dilemmas that I had midway through the project, I managed to pick up various important skills.

Extensive comments can be found in the code files to increase readability of the code. Also, please do check out the blogpost that I wrote on weather all countries should be held equally culpable in causing climate change, just to post on my website for this project. I am confident in the end product that I’ve arrived at, and I hope you like it too! Thank you!

Note: The entire website has been coded using Safari and Visual Studio Code. The same code should work with all browsers and editors, but I cannot guarantee this as I experienced some inconsistencies in output with different browsers in the previous project. If convenient, please use Safari and VS Code to run and test the application. Also, please only open the “Final Submission Code” folder in VS Code and not the entire “CEP T3 Project” because as I’ve tried, doing so messes up the json file directory. Thank you!