**Developer’s Guide - Siyu Zhao**

**Overview**

This project is a blog-based website implemented with Flask. The purpose of this website is to provide a platform enabling product designers & other tech people to post their ideas and thoughts during this social distancing period. Via this website, the users will be able to post texts regarding news that interests them. They will also be able to view other posts and sort the posts by authors. This will help the users exchange ideas and get further inspiration.

**Condensed version of the final(!) planning specs**

* Sign in/out, manage profile: **implemented**. Users are now able to log in/out, and update the profile information including username, email and profile image.
* Post text-based contents: **implemented**. Users are now able to create text-based content and post it.
* Comment on others’ posts: **not implemented.** Might do that in the future work.
* Layout of the posts on the homepage: **new added.** Now at the homepage, the max number of posts is limited to 5. User can switch from page to page by clicking the button.
* Filter the posts via author:**new added.** Now by clicking on the author’s name, the users can view all the posts by the specific author.

**Install/deployment/admin issues:**

Nothing else needs to be installed after completing the steps in the user’s guide. In terminal, cd to the root folder, then type in ‘python run.py’ to launch the application,

**User interaction and flow through your code ("walkthrough")**

**Brief Recap**

* Step 1: the first time user has to register an account
* Step 2: the user has to login with their email address and password
* Step 3: the user updates their profile information
* Step 4: the user creates a post
* Step 5: the user updates his/her post or deletes his/her post
* Step 6: the user filter all posts by the author

**Details of Each Step**

* Step 1: the first time user has to register an account
  + Routes.py:
    - Function ‘register’: link the ‘register.html’ page. Create a new user profile with the input information via db. Provide feedback to the users informing a successful registration via flash. If all the information is valid, return back to the homepage with the newly created account logged in.
  + Forms.py:
    - Class ‘RegistrationForm(FlaskForm)’: store the username and email from the input (only if they are the valid username and email, check the information via validators)
    - Function ‘validate\_username’: via query, check if the new username already matches an existing username from the record.
    - Function ‘validate\_email’: same as ‘validate\_username’
  + Models.py:
    - Function ‘load\_user’: load the user data by user\_id
    - Class ‘User’: user data package, including username, email, image\_file, password, and posts.
  + Register.html:
    - Class ‘form-group’
    - Class ‘invalid-feedback’
    - Class ‘border-top pt-3’: Bootstrap style
    - Class ‘text-muted’: Bootstrap style
* Step 2: the user has to login with their email address and password
  + Routes.py:
    - Function ‘Login’: help the user to login. Render the ‘login.html’ page from the templates folder. After the user input the information, check if the information matches the record via query. If the information is valid, go back to the homepage. Otherwise, flash a warning to inform the user.
    - Function ‘logout’: logout the user via pre-setup package. Then redirect the user back to the homepage.
  + Forms.py
    - Class ’LoginForm’: via several validators (imported), check if the information from input is valid regarding the format. Provide a ‘remember me’ checkbox and a ‘submit’ button.
  + Login.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
* Step 3: the user updates their profile information
  + Routes.py:
    - Function ‘account’: enable the user to update their profile information. First, render the ‘account.html’ page. Then, render the ‘UpdateAccountForm’ and get the user input. Rewrite the recorded user data with the input. Finally, give the user a feedback and go back to the account page.
    - Function ‘save\_picture’: resize the picture to 125x125 size and create a thumbnail image. Save the thumbnail in the picture\_path.
  + Forms.py:
    - Class ‘UpdateAccountForm’: check if the user input is valid via several validators: check if the username has a length of 2-20 characters; check if the email address is a valid email; check if the picture has an extension name of jpg or png. If valid, rewrite the user data regarding the input.
    - Function ‘validate\_username’: check if the username is already taken via query the record.
    - Function ‘validate\_email’: check if the email is already taken via query the record.
  + Account.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
* Step 4: the user creates a post
  + Routes.py:
    - Function ‘new\_post’: first, check if the user is logged in. then, render the ‘create\_post.html’ page. Collect the input from the form and store them in 3 variables: title, content, and author. Create this new record via db. Finally, give
    - user a feedback and go back to the homepage.
    - Function ‘post’: by clicking on the post title, redirect the user to the ‘post.html’ page.
  + Models.py:
    - Class ‘Post’: post data package, including title, date\_posted, content, and user\_id.
  + Forms.py:
    - Class ‘PostForm’: post form, including title input, content input, and submit button.
  + Create\_post.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
* Step 5: the user updates his/her post or deletes his/her post
  + Routes.py:
    - Function ‘update\_post’: first, check if the user is logged in. then, check if the user is also the author of the post via query. Then, redirect the user to the ‘create\_post.html’ page. Then, rewrite the title and the content of the post regarding the input and commit the changes via db. Finally, give the user feedback and go back to the homepage.
    - Function ‘delete\_post’: first, check if the user is logged in. then, check if the user is also the author of the post via query. Then commit the change via db. Finally, give the user feedback and go back to the homepage.
  + User\_posts.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
  + Create\_post.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
* Step 6: the user filter all posts by the author
  + Routes.py:
    - Function ‘user\_posts’: sort the posts by authors. Get the posts of this selected author via query ‘username’. Then, render the ‘user\_posts.html’ page. Limit the maximum of 5 posts per page via paginate and list the posts by a time order.
    - Function ‘home’: after clicking the home button, redirect to the homepage. Limit the posts per page to 5.
    - Function ‘about’: after clicking the about button, redirect to the about page.
  + Home.html:
    - Class ‘media content-section’: this class has the data of title, content, author name, and posted date.
    - Other bootstrap class to assign styles.
  + Post.html
    - Class ‘content section’
    - Class ‘form group’
    - Other bootstrap classes
* Other:
  + Run.py: import app from flaskblog folder. run the whole package and activate the debugger.
  + \_\_init.py\_\_: import essential packages, run the routes.py file.
  + Main.css: basically cite from the Bootstrap standard css package. Could be able to do some customization.

**Known Issues:**

* Major: currently no major bugs detected.
* Minor:
  + For the account page, while updating a new profile image, the system cannot crop it to 1:1 ratio image. Now the pic will be transformed to a 1:1 ratio image which will make it look weird.

**Future work:**

* Reset password: as a full-function blog, it is also good to enable the user to reset their password via email. I haven’t fulfilled this function but it should have some parallel with the “updating profile” function, which you can find the code at the route.py and forms.py files.
* Enable the users to insert images in their post: now the user can only type text in the post. A button to upload relevant images will not be a bad idea in the post-editing section. Also, we might want to consider how to arrange the images in the text to make it look more organized.
* Enable the users to comment on others’ posts: I didn’t have enough time to fulfill this function. Basically, what we need here is to create a separate section after each post. We can regard it as a “sub-post” of the initial post. An additional form should be added in the forms.py and route.py files to get it processed.