# PRATIK BHUJADE

- Part 1 [Bracket 8] --- Method Overview

This project was made using Google's app engine and python 2.7. The Google app engine is included in the Google Cloud SDK which allows us to develop applications which can be deployed to Google Cloud. This project also utilised bootstrap for styling the webpages. Bootstrap is a CSS framework which is used to develop responsive webpages. Bootstrap includes templates for all the basic HTML elements like buttons, text boxes, checkboxes etc. Jinja2 was also used in this project. Jinja2 is a python based web template engine which is used to generate HTML content which is then returned back to the user over HTTP.

In this project classes are used to specify and implement different functionalities. Each class has specific functions and the code required to implement a specific functionality. All classes in this project extends webapp2's RequestHandler class in order to handle HTTP requests. This class contains the application logic to handle a HTTP request. This class can be thought of as a Controller in MVC, i.e these classes processes the HTTP requests and generate an appropriate response and return it to the browser.

The defined classes work by calling a method as per the corresponding HTTP request so that the server can process the request. For eg. - Suppose I want to perform search operation and I click on the search link, initially as the page is loaded a GET request will be sent to the server which will be processed

by the Search class. Upon reaching the server, Search class's get() will be called as the incoming request from the browser is GET request. If the incoming request was a POST request then post() would have been called. The called method will process the request and return a response to the browser which will then be displayed to the user. The

methods defined in the classes take an argument self. This tells that this method can only be called via an instance of the class.

For mapping the urls meaning which class should be referred when a particular url is hit, we define an object of WSGIApplication. The class WSGIApplication contains all the code required to route requests to particular class. When defining the WSGIApplication object we specify a list which maps urls and classes. WSGIApplication object will map url requests to classes and then depending on the type of request appropriate method of that class will be called.

In this project I have structured the code in different files. All the classes representing the functionalities are specified in the main.py file. This file contains all the classes used by the application. Similarly for mapping the urls a file called urls.py is used, meaning in this file the WSGIApplication object is defined. The file app.yaml is used to define all the runtimes and libraries which are required by the application. The google app engine uses this file to know about the dependencies and libraries required by the application. Another file called constants.py is used which stores all the constants which will be required by the application. This file contains the object of jinja2.Environment class. This object tells the app engine where to get the HTML templates which will be used by the application.

In the file main.py all the logic required by the application is written. This logic is defined inside classes all of which extend the webapp2's RequestHandler class. As per the requirement appropriate functions are defined inside the classes mapped to particular type of HTTP requests. Inside these functions a lot of built-in functions are used for implementing the required logic. For eg. - Suppose I need to check whether a user is logged into the system or not for that purpose the app engine's users class has a function called "get_current_user()" which returns the user object of the currently logged in user and if no user is logged in then the function returns None. This can be then be used to check if a user is logged into the system or any other functionality which may require the user object of the logged in user can use this. The application uses jinja2's get_template() method to get the HTML template which needs to be rendered. This template is then returned as a response to the browser. If required, data can also be passed to the jinja template by passing a dictionary containing all the keys and values which will be rendered in the template. This

dictionary is passed as argument in the template's render() method. For specifying the type of response the server must return to the browser I have defined response.headers.By specifying the response.headers['Content-Type'] to 'text/html' it is being specified to the server that the function will return a response in text or HTML format.

For retrieving data from HTTP responses the get() method of the request class is used. This function will retrieve the data received in the HTTP request. For eg. - Suppose the user submits a form, all the data input by the user is sent to the server for processing. This data is sent in key-value pairs. By

using the get() function and specifying the key as argument we can retrieve the associated value. This value can then be used for processing as required.

For retrieving data from the datastore I have used the query() and keys method. With query() data can be queried using multiple fields and parameters which is a complex task on a NoSQL database like datastore. The query() method can take arguments or run without arguments. As arguments we pass the fields we want to compare and if no arguments are passed then it will return the entire list of objects of that model class. The query() allows us to perform various SQL like operations like AND, OR, IN etc. The key method uses ID's assigned to each object for retrieving the data. The use of key allows for faster retrieval of data.

This project also stored files which was enabled by creating classes which extended the BlobstoreUploadHandler and BlobstoreDownloadHandler class for uploading and downloading the files. The class CreatePost extends the BlobstoreUploadHandler class and uses the function get_uploads to get the uploaded file and store it in the UserPost object. The class GetImage extends the BlobstoreDownloadHandler and uses the send_blob function to send the blob object. Images are accessed by using urls. A request is made to the specific url which is mapped to the GetImage class. This class then returns the blob to the webpage.

For building login and logout system I have used app engine's built-in authentication system. To implement this functionality users class's create_login_url() method is used to generate the url required for logging into the system. Similarly for logout

create_logout_url() method is used to generate the url required for logging out of the system.

- Part 2 [Bracket 9] --- Models and Data Structures

This application uses datastore to store data. Datastore is a NoSQL database which is offered on the google cloud platform. Datastore works on the key value pair system.

In this project all the models are stored inside a folder called “models”. Inside this folder there are python files which represent tables in a database. For allowing the datastore to recognise the files as datastore models the model classes must extend the ndb.Model class. This allows the datastore to store and retrieve our classes. In our model class we define all the properties of the data models.

In this project I have used 3 models, first is the UserPost model, second is the UserComment and third is the User model. The UserPost model has the following properties - post_date, post_file, post_caption, post_user. As mentioned above this class also extends the ndb.Model class. This model has used StringProperty(), DateTimeProperty() and BlobKeyProperty(). According to the requirement post_caption and post_user uses StringProperty(), post_date uses DateTimeProperty() and for storing files post_file the BlobKeyProperty().

The model UserComment is defined in the usercomment.py file. This model represents the comments made by the users. The UserComment model has the following properties - user_email, user_post, user_comment, user_commentdate. This model has used StringProperty() and DateTimeProperty(). As per the requirement user_email, user_post and

user_comment uses StringProperty() and user_commentdate uses DateTimeProperty().

The User model is defined in the user.py file. This model represents the user object. The User model has the following properties - user_email, user_followers, user_following, user_posts. This model has used only StringProperty() along with repeated = True argument form few properties. This arguments tells the data store that this column will store a list of values of a particular type, in this case it will be a list of strings. The properties user_followers, user_following and user_posts has used repeated = True.

- Part 3 [Bracket 10] --- UI Design Home Page

On the home page the user is first asked to login if he/she is not logged in and then after logging in on the navigation bar list operations available are displayed to the user like “Create Post”, “User Profile”, “Search” and “Logout”. Without login the user can’t do anything in the application. On this page a list of posts is displayed to the users. These posts are of the user itself and of the other users which he/she is following. This posts are displayed in a vertical scroll manner. Along with the posts, the caption, comments and also the post owner’s username is displayed. On this page a maximum of 50 posts is displayed. Also per post 5 comments are displayed at max and if the post has more than 5 comments a button to view the remaining comments is displayed also the user can add new comments from the input box displayed below each post.

**Search Page**

On this page the nav bar is displayed for ease of navigation and an input box is displayed where the user can input the name of

the user they want to search. After clicking search, the results are displayed below the input box and also the results are a link which can be clicked to redirect to respective users's profile page.

**Profile Page**

On this page the profile of the user is generated. This page displays the username and the number of users they are following and being followed by. This followers and following is a link which can be clicked to view the full list of followers and following users. Below this on a vertical list, a list of all the posts created by the user is displayed along with the comments and the expand comment option in case there are more than 5 comments also the user can add new comments from the input box displayed below each post. If the profile being viewed is of a different user then an option to follow and unfollow depending upon the situation is displayed.

**Create Post Page**

From this page the user can create new posts. This page allows the user to select a file and add a caption to that post from the input fields displayed. The user is only allowed to select .png and .jpg files.

**Post Details Page**

In this page the entire post is displayed along with all the comments made by the users.

**Followers and Following Page**

This page displays a list of all the followers/ following users. This list is displayed as links so that the user can view the profile of those users if desired.