Project Final Report

Code, Video, and Reports:
https://drive.google.com/drive/folders/1guERlTWJSl-RqhrW7gR76-dbjvfvAbD-?usp=sharing
Video on Youtube:
https://youtu.be/dCNwZSbvFZk

1. Title

Social Media platform for USC students based on Flask (Firebase Emulation)

2. Team: 551_pg 37

Zihan Liu zliu2193@usc.edu USC ID: 1534981630
Shuchan Zhou zhoushuc@usc.edu USC ID: 9442515557
Zeyu Li zli86605@usc.edu USC ID: 8454499854

3. Topic

In our project, we developed a Web App based on Flask, SocketIO, and MongoDB. This app makes a reference to the Instagram posts by USC recreational sports club and it could be used like a social media platform. Users can read the post content, add a new post, like a post, delete a post, comment under a post, and delete a comment.
This app supports RESTful API to create, read, update, and delete using command URL just like Firebase. Using our datasets stored in MongoDB, we set up a Web App including 3 web pages: Home, Account Home, and Post Detail. After entering the account name, the top 10 posts of this account will be displayed in Account Home. After clicking the comment button, comments under this post can be displayed in the Post Detail page.

4. Implementation

4.1 Data Collection - Web Scraping

First, we collected all the account names of USC recreational sports clubs manually from Instagram. Then, we used WebDriver API from the Selenium package in Python to get the URL of a post from USC recreational sports club. After getting the post URL, we applied the requests package and BeautifulSoup package to extract the like number, article, and comments from each post. Finally, we got 1045 posts from 43 accounts and 4166 comments under these posts.

4.2 Data Management - MongoDB

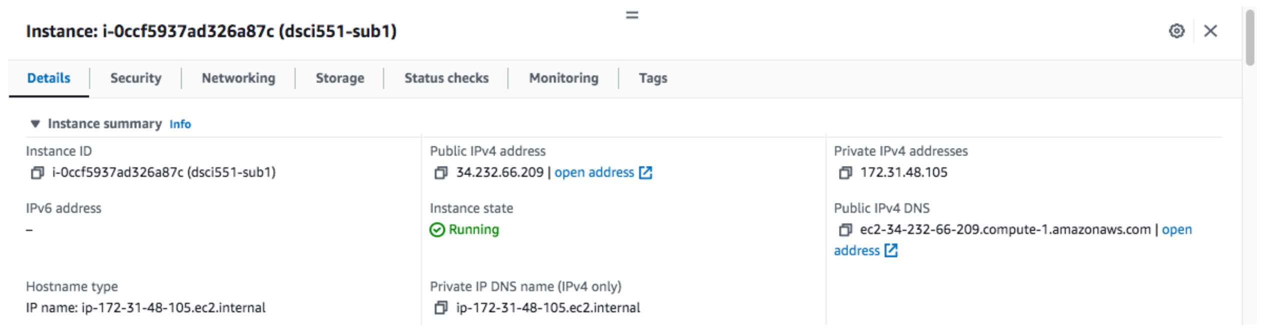
4.2.1 Create Database

Format the collected datasets into 2 JSON files, posts.json and comments,json. For each post, there are 4 attributes: _id, account_name, like_number, and article. The _id for post is the 11-character id at the end of the post URL, which can uniquely identify one post. For each comment, there are 3 attributes: _id, post_id, and comments. The _id for comment consists post_id , '_C', and the order of comment. (e.g., 'B8PyHtPJYeu_C1', 'B8PyHtPJYeu_C2', 'B8PyHtPJYeu_C3', 'B8PyHtPJYeu_C4'), which can uniquely identify a comment. The 'comments' attribute is the text content of a comment.
Then, update JSON files to the EC2 instance. Use mongoimport command to import the JSON file to MongoDB. After executing the below commands, 2 collections (posts and comments) will be added to database dsci551_project.

```
mongoimport --file posts.json --db dsci551_project --collection posts
mongoimport --file comments.json --db dsci551_project --collection comments
```

4.2.2 Connect to database - pymongo

Use the pymongo package to connect to the database constructed above. The IP address needs to be clarified when creating MongoClient. The IP address should be the Public IPV4 address of the instance.



Instance: i-0ccf5937ad326a87c (dsci551-sub1)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

▼ Instance summary Info

Instance ID i-0ccf5937ad326a87c (dsci551-sub1)	Public IPv4 address 34.232.66.209 | open address	Private IPv4 addresses 172.31.48.105
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-34-232-66-209.compute-1.amazonaws.com | open address
Hostname type IP name: ip-172-31-48-105.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-48-105.ec2.internal	

Use the code below, we can access the data stored in MongoDB using Python.

```
client = MongoClient('34.232.66.209', 27017, serverSelectionTimeoutMS=10000)
db = client.dsci551_project
```

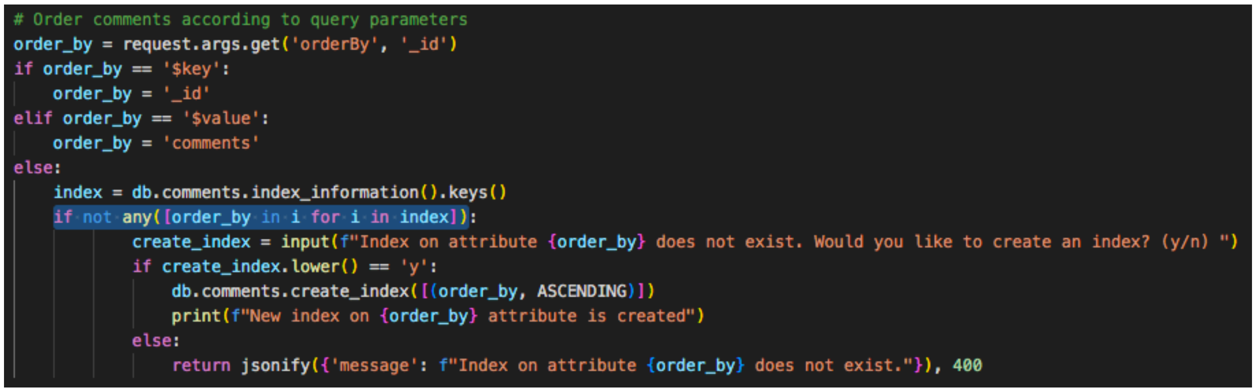
4.3 Firebase Emulation - Flask

We use Flask to built a server program (test_server.py) which can listen on clients' requests using command URL. Then, we created a route and corresponding method for each command URL. For example, when retrieving data in posts collection, the below command should be executed in terminal.
curl -X GET 'http://localhost:8000/posts.json'
Thus, there should be a corresponding route for 'http://localhost:8000/posts.json'.

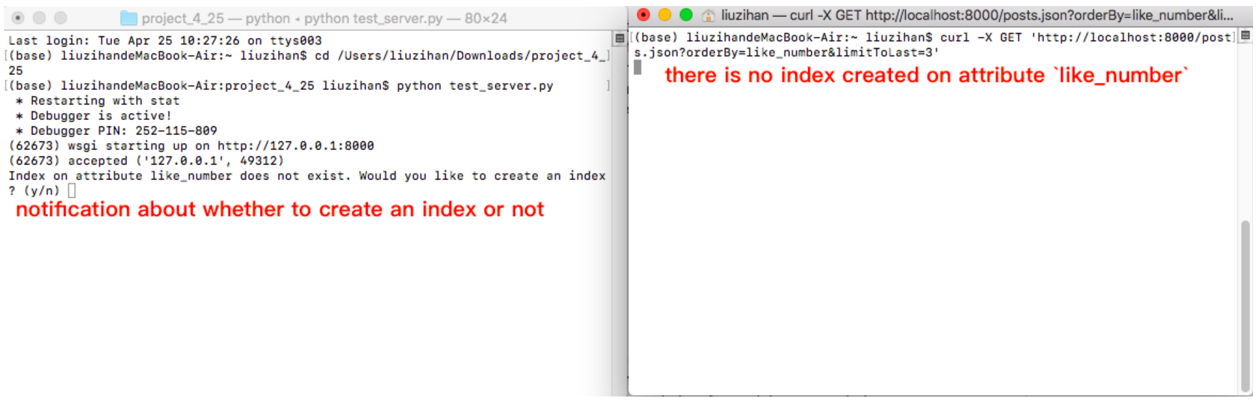
```
@app.route('/posts.json', methods=['GET'])
```

Under this route, there should be a function that can get data from posts collection in MongoDB. Also this function can deal with more complicated command URLs which include parameters orderBy, startAt, endAt, equalTo, limitToFirst, limitToLast. We used `request.args.get()` to get the corresponding value for each parameter and query with limitations.

Although the returned results can be sorted by any attribute when using MongoDB, to emulate the `orderBy` command when using Firebase, requests to create indexes on attributes should be added. Thus, we add an if statement to verify whether there is already an attribute created on that attribute.If there is no index created, a notification will pop up in the terminal window running the server. If the client enters `y`, then a new index will be created on that attribute and the sorted result will be returned.

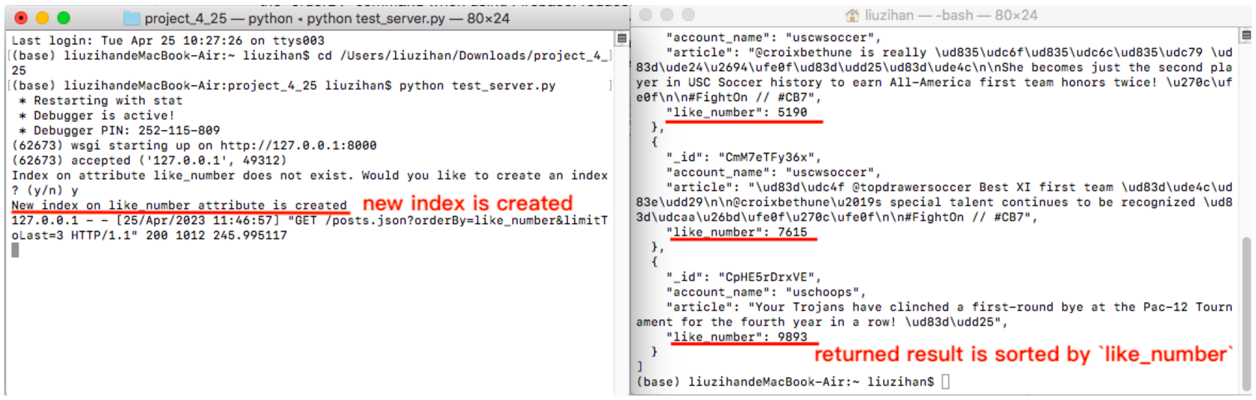


```python
# Order comments according to query parameters
order_by = request.args.get('orderBy', '_id')
if order_by == '$key':
    order_by = '_id'
elif order_by == '$value':
    order_by = 'comments'
else:
    index = db.comments.index_information().keys()
    if not any([order_by in i for i in index]):
        create_index = input(f"Index on attribute {order_by} does not exist. Would you like to create an index? (y/n) ")
        if create_index.lower() == 'y':
            db.comments.create_index([(order_by, ASCENDING)])
            print(f"New index on {order_by} attribute is created")
        else:
            return jsonify({'message': f"Index on attribute {order_by} does not exist."}), 400
```


project_4_25 — python ◂ python test_server.py — 80×24
liuzihan — curl -X GET http://localhost:8000/posts.json?orderBy=like_number&li...
notification about whether to create an index or not
there is no index created on attribute `like_number`

```
Last login: Tue Apr 25 10:27:26 on ttys003
(base) liuzihandeMacBook-Air:~ liuzihan$ cd /Users/liuzihan/Downloads/project_4_
25
(base) liuzihandeMacBook-Air:project_4_25 liuzihan$ python test_server.py
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 252-115-809
(62673) wsgi starting up on http://127.0.0.1:8000
(62673) accepted ('127.0.0.1', 49312)
Index on attribute like_number does not exist. Would you like to create an index
? (y/n)
```

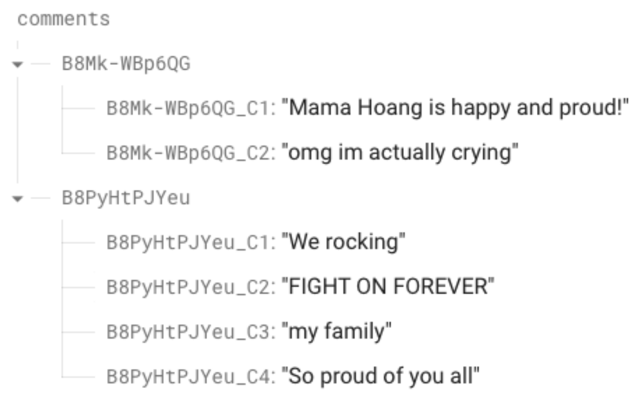
```
(base) liuzihandeMacBook-Air:~ liuzihan$ curl -X GET 'http://localhost:8000/post
s.json?orderBy=like_number&limitToLast=3'
```


project_4_25 — python ◂ python test_server.py — 80×24
liuzihan — -bash — 80×24
new index is created
returned result is sorted by `like_number`

```
Last login: Tue Apr 25 10:27:26 on ttys003
(base) liuzihandeMacBook-Air:~ liuzihan$ cd /Users/liuzihan/Downloads/project_4_
25
(base) liuzihandeMacBook-Air:project_4_25 liuzihan$ python test_server.py
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 252-115-809
(62673) wsgi starting up on http://127.0.0.1:8000
(62673) accepted ('127.0.0.1', 49312)
Index on attribute like_number does not exist. Would you like to create an index
? (y/n) y
New index on like_number attribute is created
127.0.0.1 - - [25/Apr/2023 11:46:57] "GET /posts.json?orderBy=like_number&limitT
oLast=3 HTTP/1.1" 200 1012 245.995117
```

```
    "account_name": "uscwsoccer",
    "article": "@croixbethune is really \ud835\udc6f\ud835\udc6c\ud835\udc79 \ud
83d\ude24\u2694\ufe0f\ud83d\udd25\ud83d\ude4c\n\nShe becomes just the second pla
yer in USC Soccer history to earn All-America first team honors twice! \u270c\uf
e0f\n\n#FightOn // #CB7",
    "like_number": 5190
  },
  {
    "_id": "CmM7eTFy36x",
    "account_name": "uscwsoccer",
    "article": "\ud83d\udc4f @topdrawersoccer Best XI first team \ud83d\ude4c\ud
83e\udd29\n\n@croixbethune\u2019s special talent continues to be recognized \ud8
3d\udcaa\u26bd\ufe0f\u270c\ufe0f\n\n#FightOn // #CB7",
    "like_number": 7615
  },
  {
    "_id": "CpHE5rDrxVE",
    "account_name": "uschoops",
    "article": "Your Trojans have clinched a first-round bye at the Pac-12 Tourn
ament for the fourth year in a row! \ud83d\udd25",
    "like_number": 9893
  }
]
(base) liuzihandeMacBook-Air:~ liuzihan$
```

When the `orderBy` option is set to be `$key`, the returned result of both collections will be sorted by `_id`. When the `orderBy` option is set to be `$value`, the returned result of the comments collection will be sorted by `comments`, which is the text content of a comment. That is because we assume that our data is stored like below in Firebase:


```
comments
▾ — B8Mk-WBp6QG
      B8Mk-WBp6QG_C1: "Mama Hoang is happy and proud!"
      B8Mk-WBp6QG_C2: "omg im actually crying"
▾ — B8PyHtPJYeu
      B8PyHtPJYeu_C1: "We rocking"
      B8PyHtPJYeu_C2: "FIGHT ON FOREVER"
      B8PyHtPJYeu_C3: "my family"
      B8PyHtPJYeu_C4: "So proud of you all"
```

Thus, when ordering by `$value`, the returned results will be ordered by `comments`.

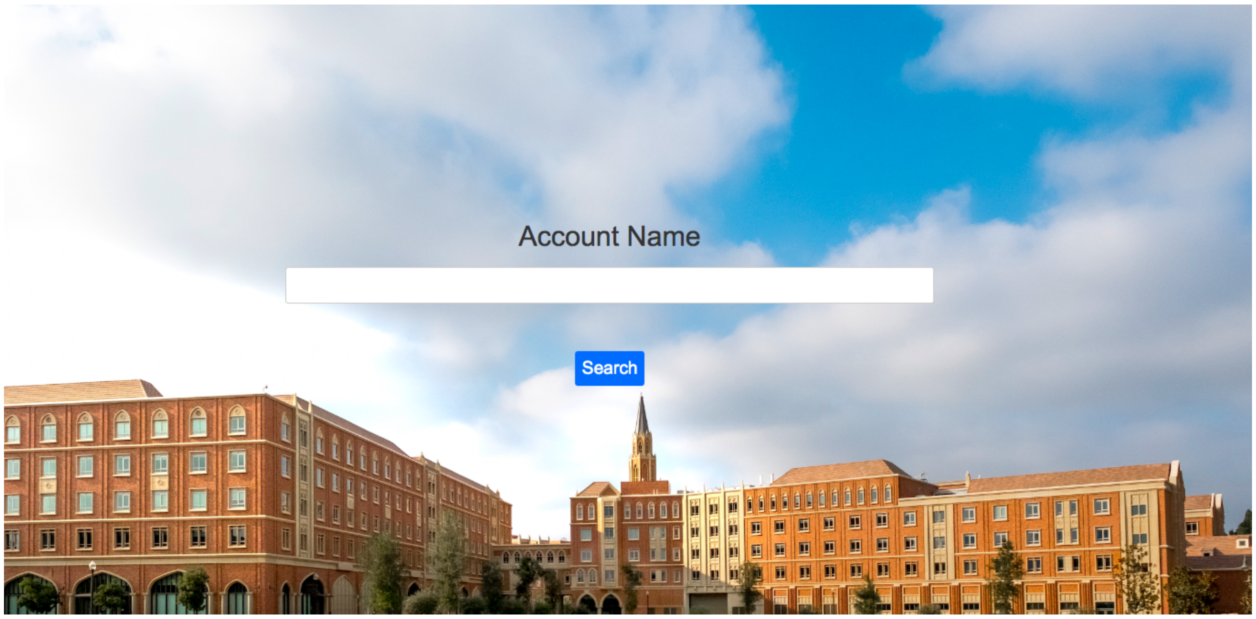
When creating a new record in posts or comments withPOST method, it is a must to specify the input data type as JSON. For example, if a new post needs to be created, the below command could be used:
curl -X POST -H "Content-Type: application/json" -d '{"account_name":"trojantennisclub"}' 'http://localhost:8000/posts.json'
The _id of newly created `post` and `comment` will be generated according to the rules. The _id of a new post consists of 11 characters which are randomly sampled from lower case and upper case alphabets, numbers, '-', and '_'. The newly generated post _id will be checked if it is already in the database. If there is duplicate, _id will be regenerated until it is not duplicated in the database.
The _id of a new comment is based on the order of comments under that post. For example, the _id of the last comment under tha post is 4, then the _id of the newly generated comment should end with 5.

Also, using the PUT method, we can overwrite a specific record or create a new record. If the _id already exists in the database, then the record will be overwritten by the input data in the command URL. If it doesn't exist, then a new record will be created with this _id and the input data in command URL.

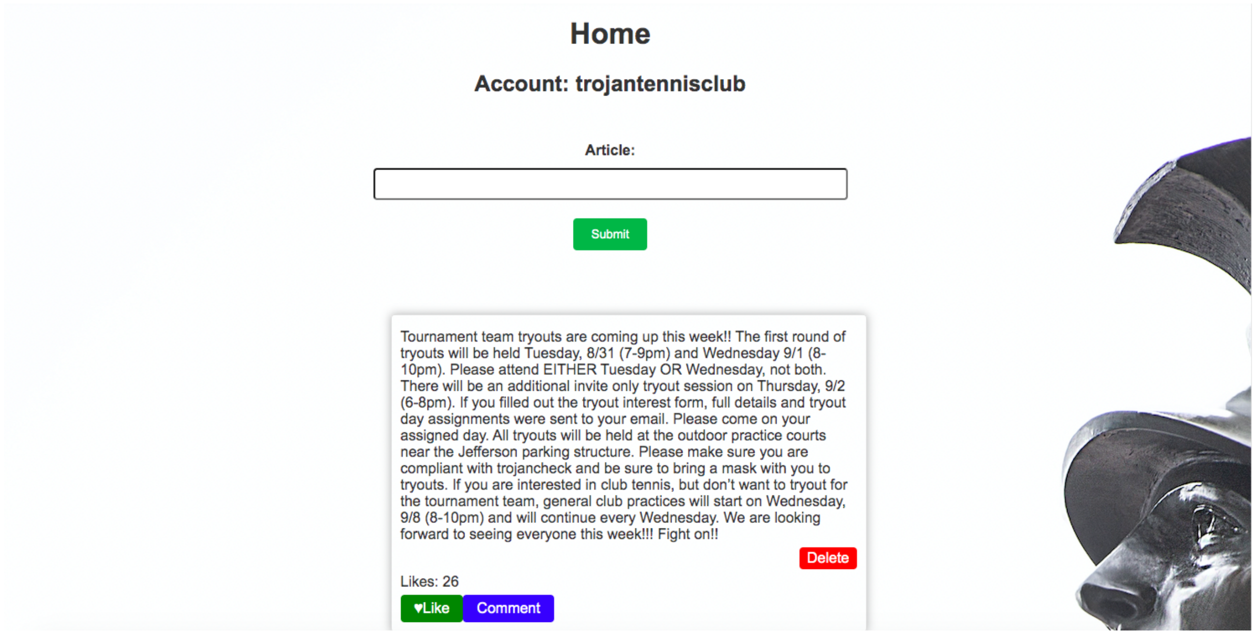
With the PATCH method, we can only update a part of attributes in one record, and this is implemented by the update_one() method in pymongo. With the DELETE method, we can delete one specific record and this is implemented by the delete_one() method in pymongo.

4.4 Web App - SocketIO

For creating the web app to that demonstrate the real-time update and syncing of the data with the server, we used the test_server.py introduced above as the server, and used JavaScript to create HTML file for each page of our web app (we have three HTML files in total: search.html, account.html, comment1.html) as clients.
First page is the Home page (search.html), we can enter the account name in that page and we will be redirected to the Account Home page. If the account name is in the database, the posts of that account will be displayed.



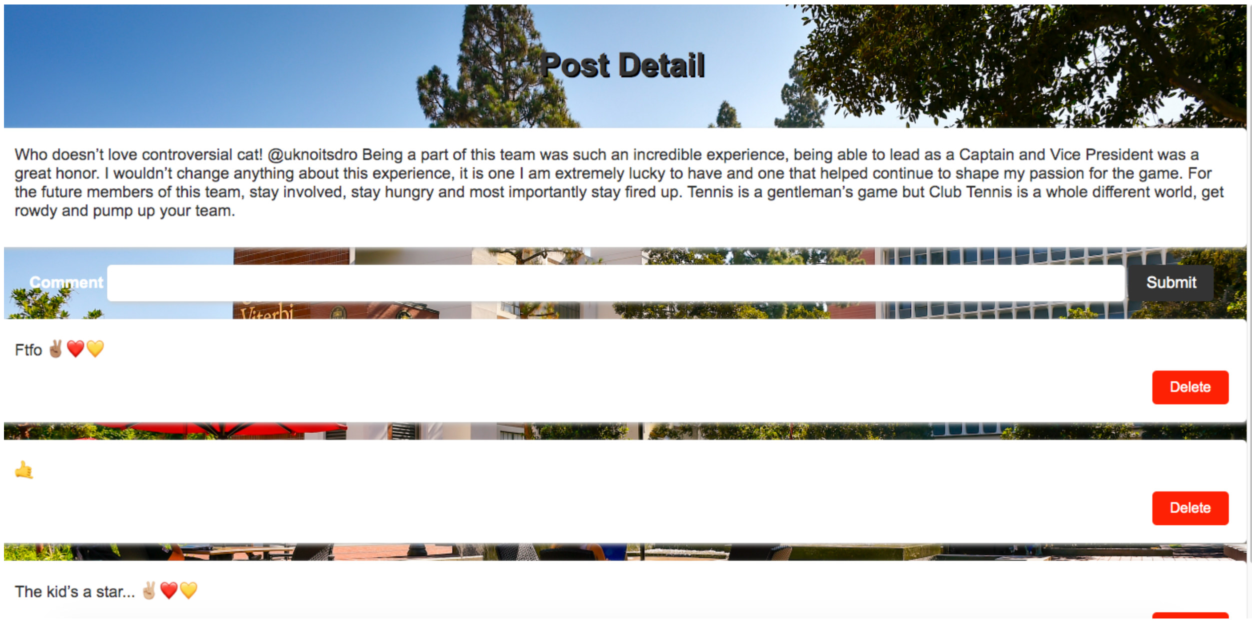
Second one is the Account Home page, which displays the top 10 posts below that account, ordered by like number in ascending order. There is a text area and a submit button at the top of the Account Home page for adding new posts. Besides each post, there is a like button, a comment button, and a delete button. We can add new posts, like a post or delete a post. If we enter the comment button after one post, we will be redirected to the Post Detail page.



Tournament team tryouts are coming up this week!! The first round of tryouts will be held Tuesday, 8/31 (7-9pm) and Wednesday 9/1 (8-10pm). Please attend EITHER Tuesday OR Wednesday, not both. There will be an additional invite only tryout session on Thursday, 9/2 (6-8pm). If you filled out the tryout interest form, full details and tryout day assignments were sent to your email. Please come on your assigned day. All tryouts will be held at the outdoor practice courts near the Jefferson parking structure. Please make sure you are compliant with trojancheck and be sure to bring a mask with you to tryouts. If you are interested in club tennis, but don't want to tryout for the tournament team, general club practices will start on Wednesday, 9/8 (8-10pm) and will continue every Wednesday. We are looking forward to seeing everyone this week!!! Fight on!!

Likes: 26

The Post Detail page displays the article content of the post, and also shows all comments of this post. Similarly to the Account Home page, we can add new comments and delete existing comments in the Post Detail page.



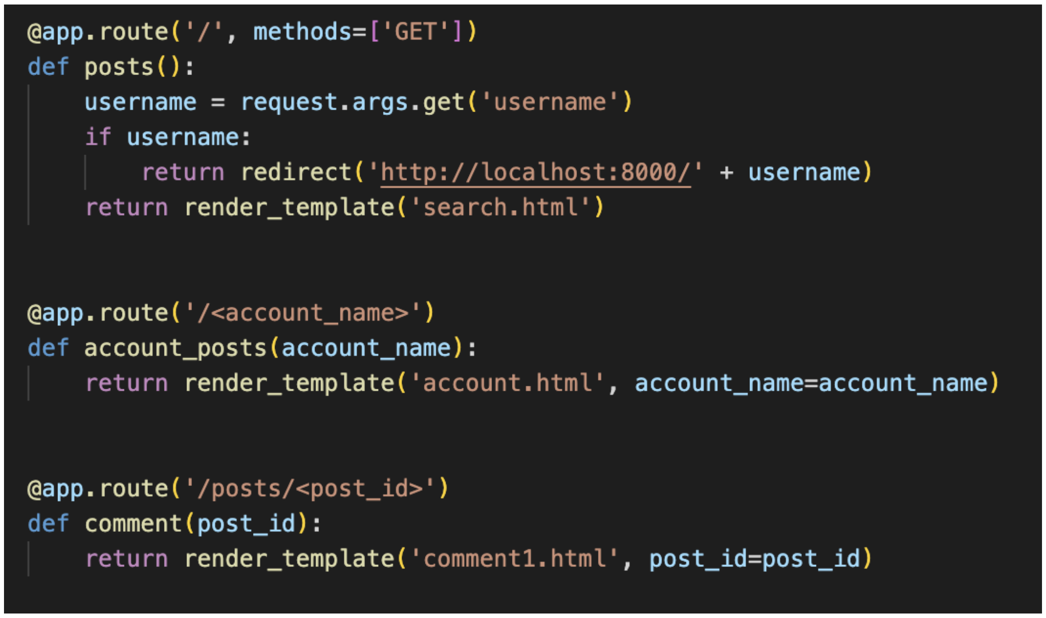
Who doesn't love controversial cat! @uknoitsdro Being a part of this team was such an incredible experience, being able to lead as a Captain and Vice President was a great honor. I wouldn't change anything about this experience, it is one I am extremely lucky to have and one that helped continue to shape my passion for the game. For the future members of this team, stay involved, stay hungry and most importantly stay fired up. Tennis is a gentleman's game but Club Tennis is a whole different world, get rowdy and pump up your team.

Ftfo ✌🏽❤️💛

🤙

The kid's a star... ✌🏼❤️💛

In our test_server.py, we used render_template to redirect each route to the corresponding html. The following are the codes:



```python
@app.route('/', methods=['GET'])
def posts():
    username = request.args.get('username')
    if username:
        return redirect('http://localhost:8000/' + username)
    return render_template('search.html')


@app.route('/<account_name>')
def account_posts(account_name):
    return render_template('account.html', account_name=account_name)


@app.route('/posts/<post_id>')
def comment(post_id):
    return render_template('comment1.html', post_id=post_id)
```

To realize the real-time updates between server and client, we used socket.io (used socketio.emit() to send a message, and socketio.on() to receive a message). Using SocketIO, the updates in the database will immediately show in the client page, the action in the client page will also transmit to the server and then update the database.

The following table includes the messages that are used for server and client communication, and the details of what client and server do with a given message.

Message	Client	Server
'get_account_posts'	Send this message to the server, with the account name, ask the server about the posts under this account. When it receives back the message ‘posts’ with the posts data for the given account, it displays each data in the frontend, along with three buttons (like, delete, comment button).	Receive the message, find the first 10 posts under the given account in MongoDB database using db.posts.find(), sorted by like number in ascending order. Then send a message called ‘posts’ back to the client, with the corresponding posts data.
'submit_post'	Emit message includes account name and article from account.html page to the server.	Receives the message from client, then execute function named handle_submit_post(). In this function, generate _id for the new post, set the default like number as 0. Then, insert the new post into database. After creating new post, get the new top 10 posts, and emit the message “posts” with newly retrieved data to client.
'like_post'	Send this message to the server, with the post_id, and the operation of adding one to the like number of this post. When it receives back the message ‘posts’, new post data (with the like number updated) will be displayed on the frontend page.	Receives this message, updating the database by increasing the like number of the corresponding post by one, then sends back the updated data to the client, in a message named ‘posts’.
'delete_post'	Send this message to the server, with the post_id, and the operation of deleting this post. When it receives back the message ‘posts’, new post data will be displayed on the frontend page.	Receives this message, updating the database by deleting the corresponding post, then sends back the updated data to the client, in a message named ‘posts’.
'get_single_post'	Send this message to the server, with the post_id, asking about the content of this post. When it receives back the message ‘post’, it will display the content of the post on the frontend page.	Receives this message, finding the content of the post by checking the mongodb database, and sends back a message named ‘post’ back to the client, along with the corresponding data.
'submit_comment'	Emit the message including post id and user-input comments to the server.	Receives this message from the client, then executes a function named handle_submit_comment(). In this function, implement the

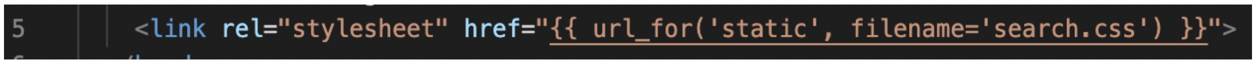
		comment's id, and new comment. Then find the post_id in the database and update its comments. Finally, emit a message called “comments”, with the latest database information to the client.
'get_post_comments'	Emit the message including the post id to the server. When it receives the message ‘comments’ from the server, the content of the comments will be displayed on the frontend page.	Receive the message from the client, then execute a function named handle_get_post_comments(). After that, find comments under the post_id in the comments database, and emit a message called ‘comments’ with the comments_list to the client.
'delete_comment'	After clicking the “delete” button, the client will emit the message called “delete_comment” and comment_id of the comment that it wants to delete to the server. When it receives the message ‘comments’ from the server, the content of the new comments will be displayed on the frontend page.	Receives the message from the client, then executes a function named handle_delete_comment(). After that, delete the comment in the database, and emit the latest message called ‘comments’ with the updated comments_list to the client.

To summarize briefly, each time when the server receives a message about searching, adding, deleting, or updating a part of data, we used pymongo package in the test_server.py to do the change to the data in the MongoDB database, then send back a list of data that the client is requesting to the client. After receiving the returned message from the server, the client will display the updated data on the frontend.

4.6 Frontend Page - HTML and CSS

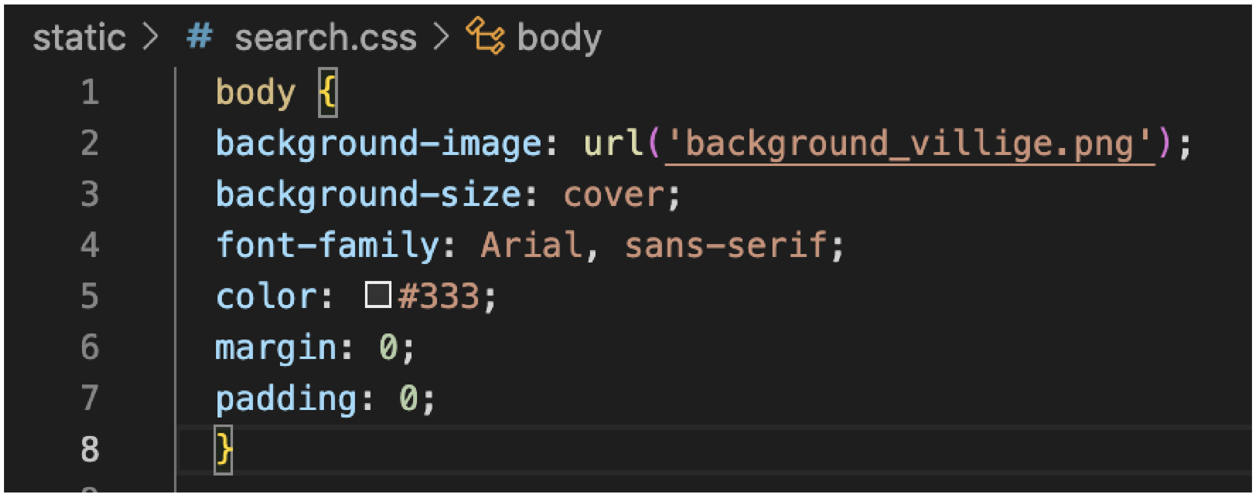
For the frontend, we used CSS files to describe the presentation and styling of each HTML file. We applied elements’ appearance and properties, such as font size, color, margin, padding, and layout according to the elements in html. HD images from the USC website as the background image are downloaded and inserted into the CSS file. We enhanced the readability and aesthetics of HTML pages by setting font sizes and styles under different tags in each HTML file, and inserted the designed CSS file into the corresponding html file.

The following code inserts the search.css file into the search.html file:



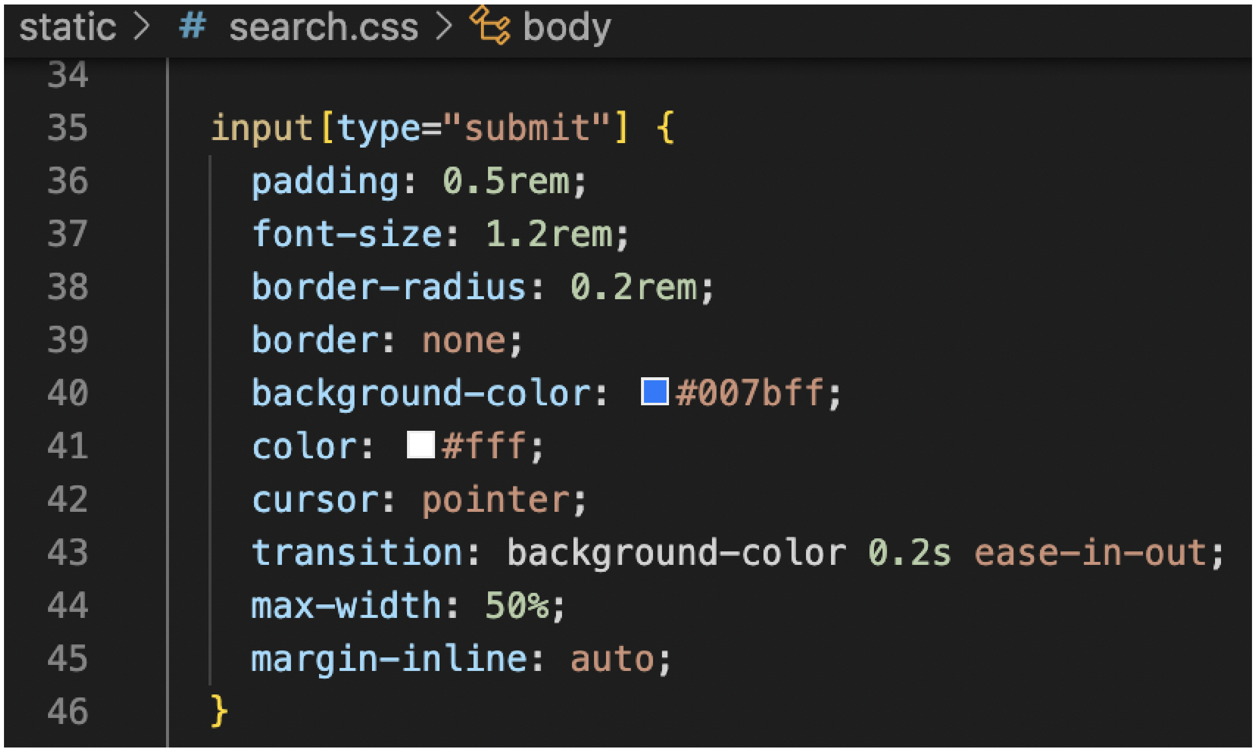
```
    <link rel="stylesheet" href="{{ url_for('static', filename='search.css') }}">
```

The following code shows how to insert a background image called background_villige.png into the search.css file and the body design of search.css file.



```css
body {
background-image: url('background_villige.png');
background-size: cover;
font-family: Arial, sans-serif;
color: #333;
margin: 0;
padding: 0;
}
```

The following code shows the design of the submit button in the search page.



```css

input[type="submit"] {
  padding: 0.5rem;
  font-size: 1.2rem;
  border-radius: 0.2rem;
  border: none;
  background-color: #007bff;
  color: #fff;
  cursor: pointer;
  transition: background-color 0.2s ease-in-out;
  max-width: 50%;
  margin-inline: auto;
}
```

The above images show only part of the code, but more detailed CSS code is already in the zip file.

5. Learning Experiences

1. Data Collection
   Through this project, we learned how to collect data using web crawler. First, we need to identify the target websites. Then, identify the HTML elements that contain the data and use the scraping tool such as BeatifulSoup and Selenium to extract data.

2. Database Management

In this project, data management plays an important role. It is vital to decide how to construct logical schema including number of attributes in one collection, data type of attributes, and construction of unique identifiers(key). It is more flexible to manage data with a NoSQL database such as MongoDB, but we must take the database consistency into consideration in our program.

3. Server and Client
   In this project, in order to emulate Firebase, we used Flask to build an application which can receive the requests from command URL. After starting the server application, the request to the specific host and portal number will be handled by the server program. It is important to know the relationship between server and client, how the requests are sent and received, and how the server interacts with the database according to the clients' requests.

4. Update and Synchronization
   From this project, we learned the methods of interaction and synchronization between front-end and back-end. Through front-end buttons, user input, and other operations, the server can perform corresponding operations and connect with the back-end database to achieve the interaction between front-end and back-end. The most important functions for update and synchronization are socket.on() and socket.emit().
   To be more specific, for example, when the user clicks the delete button, the button triggers the corresponding function in the HTML interface, which then emits a message to the server. The server listens and receives the message, and then performs the operation of deleting data on the database. After the operation is completed, the updated content of the database is returned to the HTML interface.

5. Frontend Implementation
   In addition, we also learned how to draw the front-end interface, using CSS and JavaScript operations to make the HTML interface more user-friendly and improve the user experience. Although the CSS design of the front-end is not a major part of this course, this project has given us a preliminary understanding of front-end design.