

San Jose State University
Computer Engineering Department
Spring 2014



CMPE 275
Enterprise Application Development

Project-2 Report

PinterestLike Project2

Submitted To

Professor John Gash

Submitted By

TEAM INFINITY

Jain Sarvagya (009269377)
Pancholi Swapnil (009303684)
Kesarwani Gaurav (009278815)
Mittal Ayush (009311068)

May 12, 2014

Table of Contents

Introduction	3
Project Architecture	3
Functional Requirements	4
RESTful APIs	4
Bottle.....	4
Database	4
AEROSPIKE.....	4
Learnings	5
COUCHDB	6
File System	9
Reason for Implementation	9
Common REST APIs	10
Assumptions.....	10
Implemented APIs.....	10
1. Get All Pins	10
2. Get All Boards	10
3. Get Board	10
4. Get Pin.....	11
5. Registration	11
6. Login.....	11
7. Get User Info/ Boards	11
8. Upload Pin.....	11
9. Create Board	12
10. Attach Pin :	12
11. Delete Board :	12
12. Add comment.....	12
Future Implementations	13
Learnings	13
Conclusion.....	13

Introduction

A PinterestLike project is a project to implement Pinterest like features. It is a re-enactment of Pinterest, a visual dictionary for people to collect their ideas in the form of pictures. In this project, our aim is to learn and develop a few RESTful APIs which enable a client to implement a website like Pinterest.

The class has discussed and have come to a common interface specification. The implementation of RESTful APIs has to be done accordingly. The project focuses on features like User's registration and login, Pin creation, attaching it to a board, Board creation, deletion and adding comments to a pin. The server is developed in Python language along with Bottle micro framework to support REST APIs. The database used is CouchDB. The local system is made as a server for storing image files in it. The client is nothing but just the shell commands to check the REST APIs.

Project Architecture

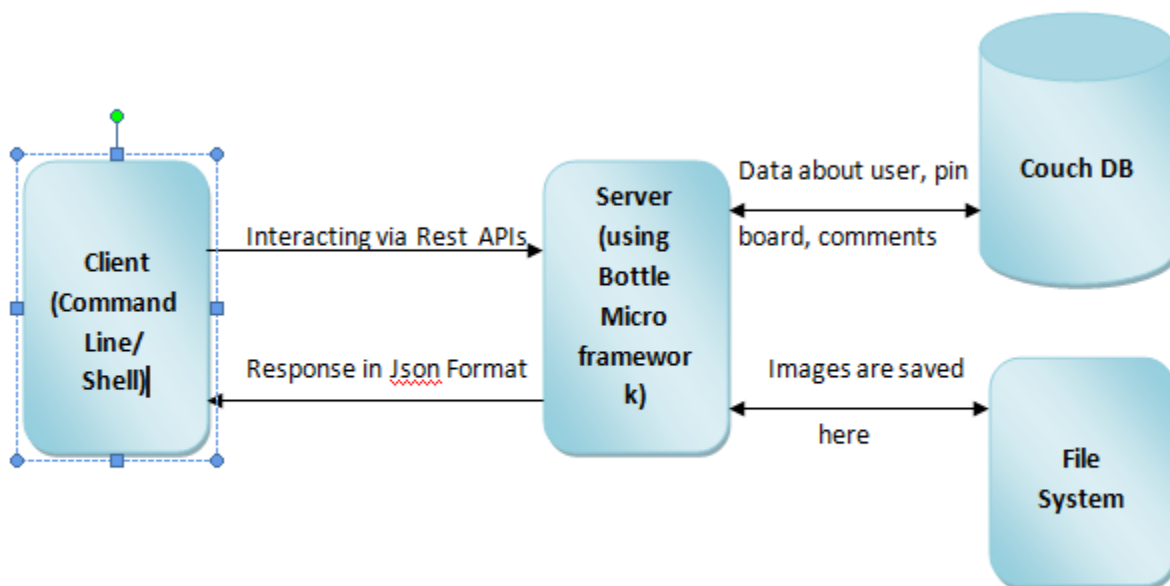


Figure : 3 - Tier Architecture

Functional Requirements

RESTful APIs

REST is an architectural style for building web services which are using HTTP protocol. Rest web services are treated as resource and can be accessed and identified by their URL unlike SOAP web services which were defined by WSDL. It enforces a stateless client server design. The REST Apis support more formats other than just XML unlike SOAP. It supports JSON which is much easier to use, understand and is light also unlike XML.

The challenges faced in this was to come to design the whole project, arriving to a common consensus what functionalities will be there in the project and what will be their request URL, payloads and responses.

Bottle : It is a WSGI micro framework for python. It is simple and lightweight framework such that it has no other dependencies other than python libraries and a single bottle micro framework's file.

Database : There were a lot many challenges faced in terms of database.

AEROSPIKE

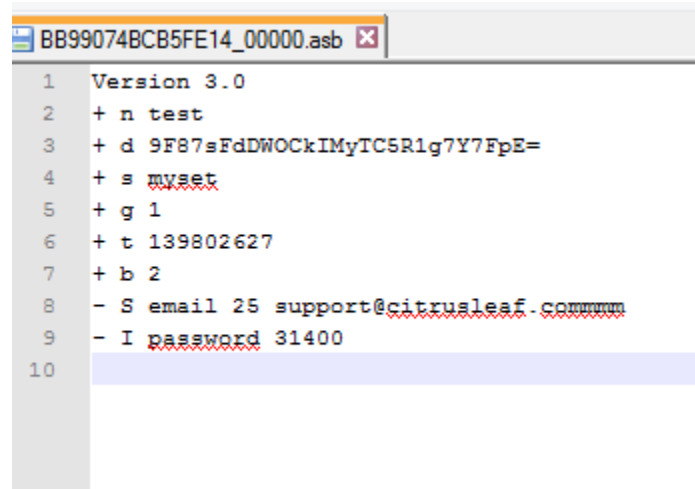
The **challenges faced** by us in using Aerospike are :

1. After successful installation, the major problem was that every time the server shuts down one of the folder (namely 'Aerospike') used for starting the Aersopike server formed in '/etc/var/run ' was deleted every time. On restarting the system, the folder was deleted automatically. This problem was found out after going through the logs.

The **solution** to the problem was found out that we changed the config file, changed the path of the folder to '/home/Aerospike' and gave appropriate rights to it.

2. Since Aerospike is an In-memory database, the stored data was wiped out in case the system is restarted. The **solution** to this we found out was that we wrote a script such that every time anything is written to the Aerospike database, we were saving the data in a file. In case of restarting the system, the database was restored from that file. The script used was :
 - **backup command** - *clbackup -h 127.0.0.1 -p 3000 -d cl_backup -F*
 - **restore command** - *clrestore -h 127.0.0.1 -p 3000 -d cl_backup*

The backup file looked like:



```
BB99074BCB5FE14_00000.asb
1 Version 3.0
2 + n test
3 + d 9F87sFdDWOckIMyTC5R1g7Y7FpE=
4 + s myset
5 + g 1
6 + t 139802627
7 + b 2
8 - S email 25 support@citrusleaf.commm
9 - I password 31400
10
```

3. We started and start developing the project. The next challenge was that in spite Aerospike v3 provides the storage of list and maps, due to the lack of documentation, examples and time crunch we were not able to find how to put list/map data in database in python.

We found out that *"Underneath the covers, Aerospike uses the C client functionality which is exposed using swig. For a working python client, we need swig software, python-devel libraries and a compiled C client library (included in the package)."* We found the C-client methods to put and get But we were not able to find the python methods which will map to these C methods.

We tried writing C method too to implement the put and get methods for list and maps but weren't successful.

We asked question on stack overflow too but due to time crunch, we were not able to research more over it. So we decided to move on to couch DB, as the whole class was implementing it and our too much of time was already invested in dealing with all the Aerospike's challenges.

Learnings

The major learning from implementing Aerospike was that how in-memory database differs from other databases. Till now, we only read about the in-memory databases but never implemented one. So, in spite of the fact that we are not able to complete our project using Aerospike and submitting our project using Couch DB, still the learnings in the process were huge. We got to know about benefits and the issues of In-memory database.

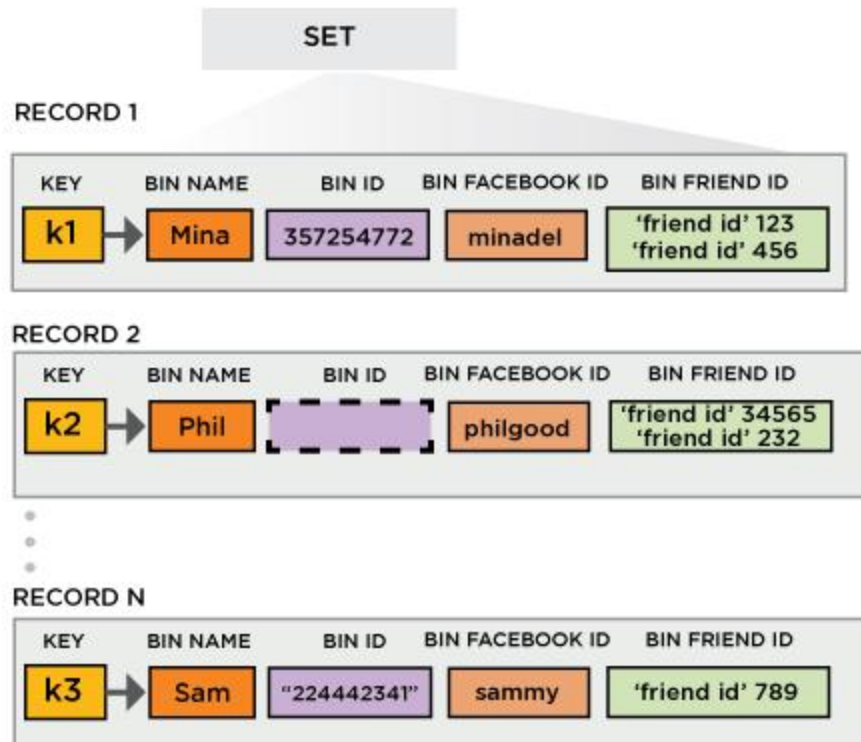


Figure : Structure of the records saved in Aerospike Database

- The figure above shows the structure of the Aerospike that how the data could be saved in it. The structure is like, namespaces have sets, sets have records in one record there can be several bins associated with keys. The database was appropriate for the kind of data we have to save. For eg :
 - UserID would be the key. All the details associated with it such as user details, array of board ids could be various bins saved in one record.
 - Board ID would be the key. All the details of the board, array of pinIDs would be various bins.
- We learnt that it is not same as permanent storage. So we learnt to deal with in-memory database that how the data cease to exist if the server is restarted. So, we should store the data in another persistent database or save it in files. But, in spite of its such disadvantage we use it for faster access.

COUCHDB

After facing the challenge in Aerospike we switched over to CouchDB. CouchDB's a few features that make it stand out:

1. It has no read locks.
2. Replication is *easy* and can be bidirectional.
3. Any record (row, document, etc) can participate in any index any number of times.
4. You can back up a database with 'cp' command without shutting it down.

There are few approaches that we analyzed for our database structure :

1. Each record(pin/board etc) can be saved as a separate document.
 - The **advantage** of this approach is that any document can participate in any index any number of times. So, indexing can be done in this case which will lead to faster search.
 - **Disadvantage** being all pins, boards and user documents will be saved all at one place. Searching for one pin/board will lead to searching in all the documents irrespective of whether they are pin, board or user documents. Moreover, indexing has to be done on the basis of kind of entity which has been saved in the document.
2. For each kind of entity i.e. pins/user/board/comment a different database can be made.
 - **Advantage** : This approach would make search faster and help in keeping the documents separate. seemed inappropriate.
 - **Disadvantage** : The design of making 4 different database for one application and related entities seems inappropriate.
3. Only one document each for users, pins, boards and comments. We found this approach better in design and will be good performance wise as compared to above 2. Moreover, we also found a way to overcome its demerits. Therefore, we implemented our database using this approach.
 - **Advantage** : This would make one DB for one application but will keep the entities separate. It makes the design clear and readable. Moreover, searching for one particular kind of entity would result in search in only its own kind of entities and not the whole DB.
 - **Disadvantage** : Since, indexing cannot be done within the document, searching for one pin/ board etc, would take more time and its time complexity will be $O(n)$. To **overcome this disadvantage**, our approach is as follows :
 - i. Suppose we have to find a pin with pin_id "xxx".
 - ii. All the details of pin are saved by using pin_id as the key and all other details of the pin as the value of that key.
 - iii. Now, to search for a pin among all the pins, all the pins are copied in python dictionary, which will have a complexity of $O(1)$.

- iv. Since the pin details are saved against the pin_id, searching for a pin will again be of $O(1)$ task in python dictionary.

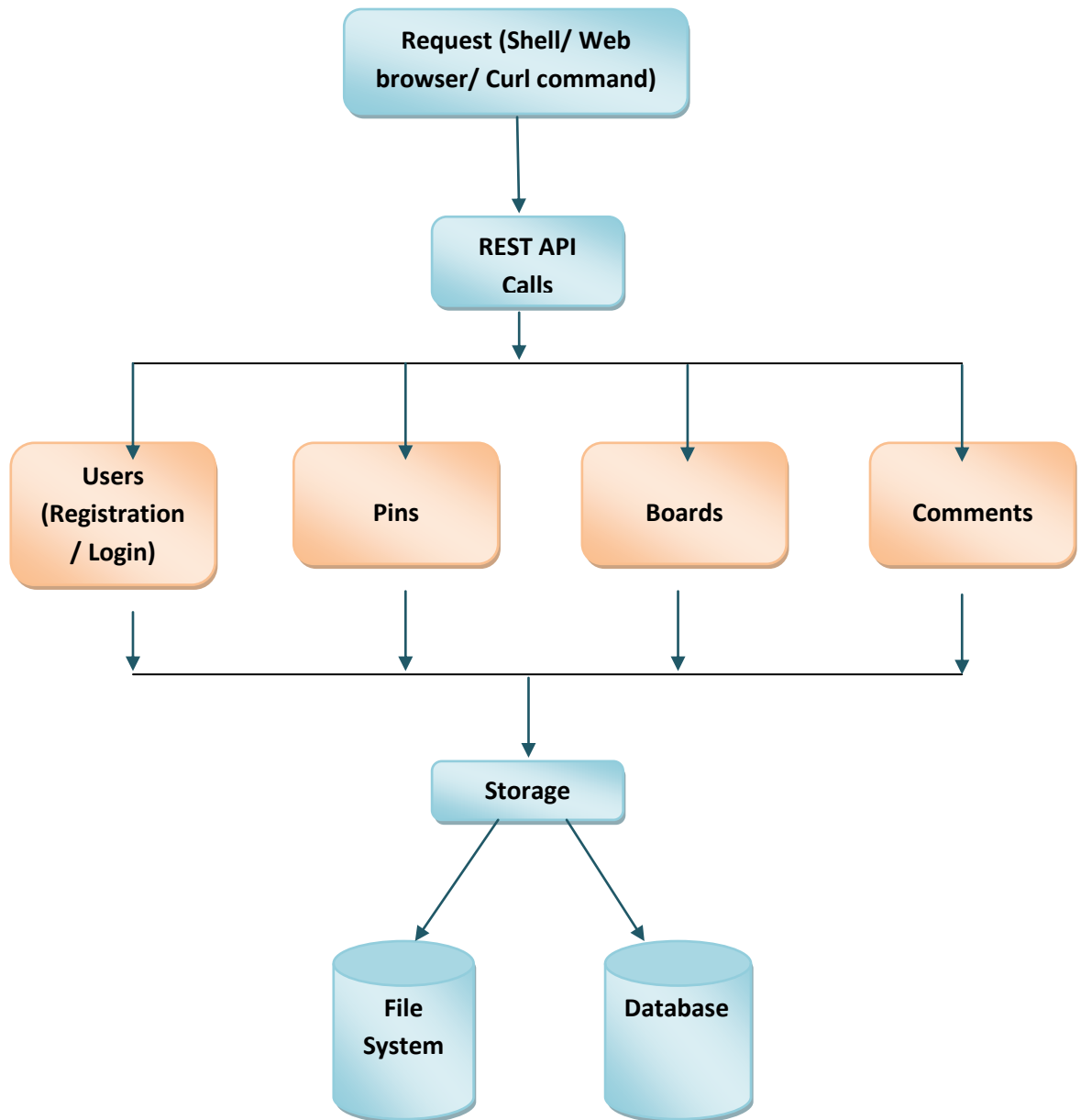


Figure : Represents how a request reaches to database

File System

The images in the pinterest like project 2 has been saved in a file system. The database is using for saving the path of the image in the file-system.

- i. When the client requests for upload of an image, the image is downloaded from the internet to the server's file system.
- ii. The path of the image in server is saved in the database with respective pin_id. While attaching a pin to the board, simple an association is made of the pin_id with the board_id.
- iii. Now, when the user requests for get_pin, user will get all the pin details including pin_url.

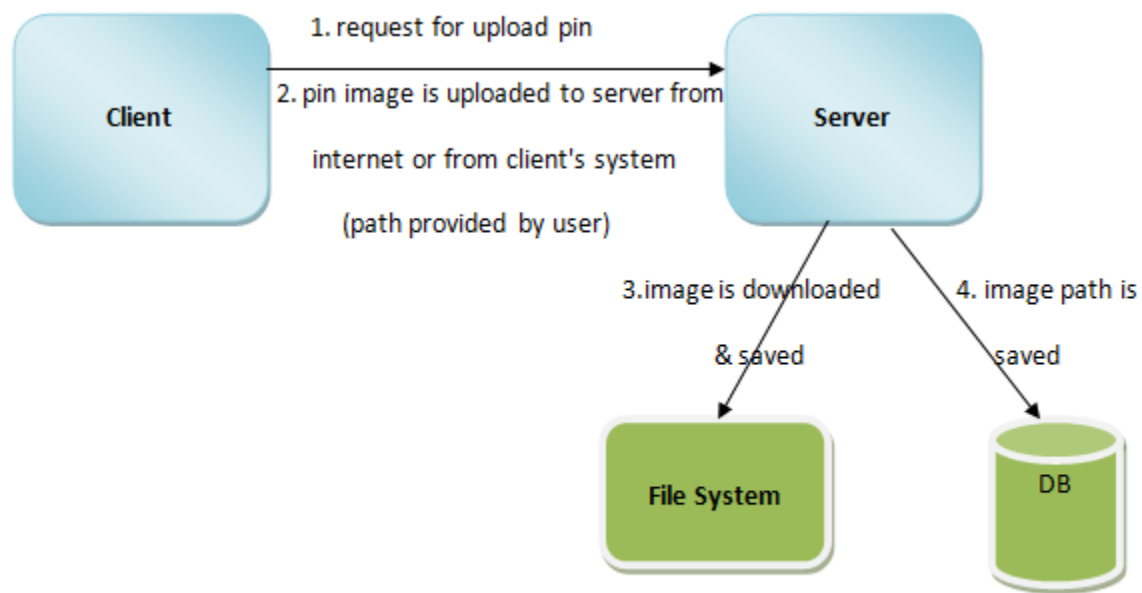


Figure : Request for Pin Upload

Reason for Implementation

We implemented image storage on the server and not on cloud because of following reasons :

1. **Security** : The security of our server would be in our hands. We can implement the layers of security according to our choice. If personal images are loaded, then clients would like to have the images with the hosting company and not with some third party.
2. **Dependency on Vendor** : Uploading images on cloud will make dependency on vendor in great terms. If our files are with us, we would not have to be dependent on anybody else.

Common REST APIs

Assumptions

A few assumptions which are made in the project development are :

1. No Private Boards or pins.
2. Pins can never be deleted by any user.
3. User who created board can only delete it.
4. If user comments on a pin, it will be associated with all the boards.
5. ':user_id' in the REST API URL refers to the token provided by the server during registration or logging in. This token needs to be provided for accessing certain functions which only a logged user have rights to perform.

Implemented APIs

1. **Get All Pins** : This function is used to get all the pins that are present in the database. Since our assumption is that there are no private pins. So anybody can see all the pins present in the database.
Request URL : /v1/pins
Method : Get
Request Payload : None
Response Body : pins[{ pin_id, pin_name, pin_url }]
2. **Get All Boards** : This function is used to get all the boards that are present in the database. Since our assumption is that there are no private boards. Therefore, anybody can see all the boards present in the database.
Request URL : /v1/boards
Method : Get
Request Payload : None
Response Body : boards[{ board_id, board_name }]
3. **Get Board** : This function is used to get one particular board details. Since our assumption is that there are no private boards, anybody can view any board which are present in the database.
Request URL : /v1/boards/:board_id
Method : Get
Request Payload : None
Response Body : pins[{ pin_id, pin_name, pin_url }]

4. **Get Pin** : This function is used to get one particular pin details. Since our assumption is that there are no private pins of a user, anybody can view any pin which are present in the database.

Request URL : /v1/pin/:pin_id

Method : Get

Request Payload : None

Response Body : {pin_url, pin_name, comments[{ user, comment }]}

5. **Registration** : This function is used for signing up for the first time users. The user id given by the user is checked in the system for its uniqueness. The password entered is encrypted before saving in the database using *SHA256*. We are using *passlib.hash* module of python which contains all the password hash algorithms.

Request URL : /v1/reg

Method : Post

Request Payload : { name, username, password }

Response Body : { token }

6. **Login** : This function is used for signing in into the system. Once a user is signed in, then only the user can perform certain functions like create board, create pin, attach pin, delete board, add comment etc.

Request URL : /v1/login

Method : Post

Request Payload : { username, password }

Response Body : { token }

7. **Get User Info/ Boards** : This functionality is used for getting a user's information. Along with the user information, all the boards associated with the user can also be seen.

Request URL : /v1/user/:user_id

Method : Get

Request Payload : None

Response Body : { name, boards[{ board_id, board_name }]}

8. **Upload Pin** : This function is used for uploading or creating a pin into the system. While creating a pin, a user has to give the client URL i.e. the URL of his local system from where the server can download the image and can save it in its file system and its respective path in database.

Request URL : /v1/user/:user_id/pin/upload or /v1/user/:user_id/pin

Method : Post

Request Payload : {client_url: <<file path>>}

Response Body : {pin_id}

9. **Create Board** : This method will be used for creating a board for a particular user. Since, only a logged in user can create board, we will need to have a token in the URL.

Request URL : /v1/user/:user_id/board

Method : Post

Request Payload : {board_name}

Response Body : {board_id}

10. **Attach Pin** : This method is used for attaching a pin to a board. 'Upload pin' and 'Attach pin' are two different functionalities because a pin can be uploaded only once but a pin can be attached to any number of boards. While attaching uploading a pin won't be required as its details already exist in the database

Request URL : /v1/user/:user_id/board/:board_id

Method : Put

Request Payload : {pin_id }

Response Body : None

11. **Delete Board** : This function is used for deleting the board from the database. While deleting the board, only the pins attachment to that particular board will be removed. This will not delete the pins or pin's comments.

Request URL : /v1/user/:user_id/board/:board_id

Method : Delete

Request Payload : None

Response Body : None(only a success code :200)

12. **Add comment** : This function is used for adding a comments to the pin, The comments will be associated with the pin and not the board. Therefore deletion of board won't result in deletion of the comments.

Request URL : /v1/user/:user_id/pin/:pin_id/comment

Method : Post

Request Payload : {comment}

Response Body : None(only a success code :200)

Future Implementations

1. Feature enhancement
2. Aerospike can be further researched over and learnt. If we would have time, we would have tried testing and correcting more our C method to set list and maps through python. We tried writing the method but it did not work successfully. Time limit does not allowed us to invest more time in it.
3. REST APIs could have been made more robust against all types of formats to requests.

Learnings

1. Getting familiar with the Python language and micro framework Bottle.
2. Even though we were to able to implement project using new key-value database Aerospike, it was a tremendous learning experience.
3. We got to learn one more database and its structure, Couch DB. Brooding over the design of the database in the project was also a great learning point.
4. While facing certain challenges, the research required to overcome them made us familiar with some concepts like NoSql, Rest, Nested JSON Objects, JSON dumps.
5. We also learnt to encrypt and decrypt the password using SHA2. We got to know that python has its own library for all the password hash algorithms. we did not imagine that encryption of passwords in python would be so simple.

Conclusion

This project familiarized us with the basics of the most commonly used current web architecture. This project also provided us with an altogether difference experience with working in teams and designing the global standards that ensured successful communication across the distributed set up.