CMPE 352

MILESTONE REPORT 2

Edited By

Abdulkadir Elmacı
Yağız Efe Şabanoğlu
Erencan Uysal
Hamza Akyıldız
İsmail Ata İnan
Merve Rabia Barın
Onur Can Avcı
Ramazan Bulut
Umut Kocasarı

Table of Contents

Executive Summary

---

1. Summary of the Project & Overall Status

Introduction:

We are Bogazici University Computer Engineering students. This report mainly contains our group work in CMPE-352 (Fundamentals of Software Engineering) course.

In this report, we documented our works for practice app project. This project is a practice project as stated in the name. We had prepared requirements, diagrams and scenarios for our “Crowdsourced Location Stories” project. In practice app, we tried to stay as connected as possible to the previous works. This practice will help us in the future.

We were expected to build a RESTful API, dockerize our application and deploy our application to a server. Each team member had to write a post and a get method. We should also use a third party API to fetch some content and process what it returns. In this report, you can see the work done by each team member in detail.

We designed an application like a social media platform. You can see functionality of our application in the “Basic Functionality of the Project” part of this report.

We used:

- Git as team software development,
- Python as programming language,
- Flask as python-based web framework,
- React for user interface design,
- MongoDB for our database,
- And swagger for documentation.

You can see a detailed evaluation report for these tools in “Evaluation of Tools and Processes”

Overall Status:

- All team members contributed to the API with at least one post and one get method.
- Everyone used at least one third-party API and processed what it returns in JSON format.
- We pushed all our work to the practice-app folder in our repository.
- All team members wrote unit tests for their codes.

- We dockerized our application and deployed it to AWS EC2.
- We provided a user interface which you can reach with URI

This project was a great chance to gain experience in software development. We had the opportunity to work with our team. We learned to use Git commands, web frameworks, database management systems and interface description languages.

2. URI of Deployed Application

At this part, we will be explaining how we deployed our app and the URI of the deployed application. At first, created an Amazon EC2 instance. Opened the port 80 for HTTP, 22 for SSH and 3000 for our API. Then connected the instance using SSH. Installed docker and docker-compose to the instance. Then ran the docker compose file. After some time, the application was running.

Here is the URI of our deployed application:
http://ec2-35-157-237-50.eu-central-1.compute.amazonaws.com

3. API URL

35.157.237.50:5000

4. Basic Functionality of the Project

Columbus is a social media platform which allows users to share their photos related to a specific location and shows the photos of these locations in a timeline. It is similar to other photo-sharing social media platforms however the main focus is on memories of a location. Different users in the system could add photos of the same place from different dates and by collective effort, one can see all the photos of a location in a timeline. This could lead to very different and creative experiences.

Columbus is a crowdsource platform, which means users shape all the photos and locations in the system. There are admins however they only care about the reports of users. All of the actions are based on the users. They interact with other users by adding photos for a location, interacting with a photo sent by another user or reporting other users or posts.

System also shows recommendations for a searched location based on geographical information of the location given by users. Moreover, it is possible that if the name of the location is entered mistakenly, the system shows recommended locations based on the text similarity with the all locations in the database.

5. Challenges

As can be expected, a frequently encountered problem in group projects is the creation of a collaborative working environment. We have 9 members in our team. We used git as a version control system. Sometimes it can be a challenging task to merge our work with the work of teammates. To overcome this challenge, firstly we used separate branches for our work. Of course we designed our branch within some rules like locations of directories or names of files. Then we merged these branches to “development” branch one by one. In this way, we prevent conflicts which can occur while working in the same branch.

Selecting tools was another challenge for us. We had to decide which framework and database we should use. Between Django and Flask, we decided to use Flask in our project. We decided that way because we had some experience from lectures and it would be easier to implement compared to Django.

Another challenge was learning the tools that we used in our project. We used git, flask docker, mongoDB, react, swagger and some other programming tools. Some of our teammates had experience in these tools and they helped us learn to use them. We also shared the tutorials we found with each other.

We also had difficulties in unit tests. There are some rules for unit tests that must be followed when working with a database. We used mock to avoid interactions with our database. In this way, we were able to perform our tests with sample data and we could easily see outputs of functions.

6. List and the Status of Deliverables

Deliverable	Status	Link
Practice-App	*Completed*	https://github.com/bounswe/2021SpringGroup7/tree/development/practice-app

7. Evaluation of the Status of Deliverables

- Practice-App

Practice app was our first implementation deliverable as a team. Testing, dockerization, and deployment were also a part of it. Since there was a whole application waiting to be developed and deployed, effective time management was crucial to deliver Practice-App on time. In this state of mind, we were quite strict about our deadlines and we split the app into two parts: backend and frontend.

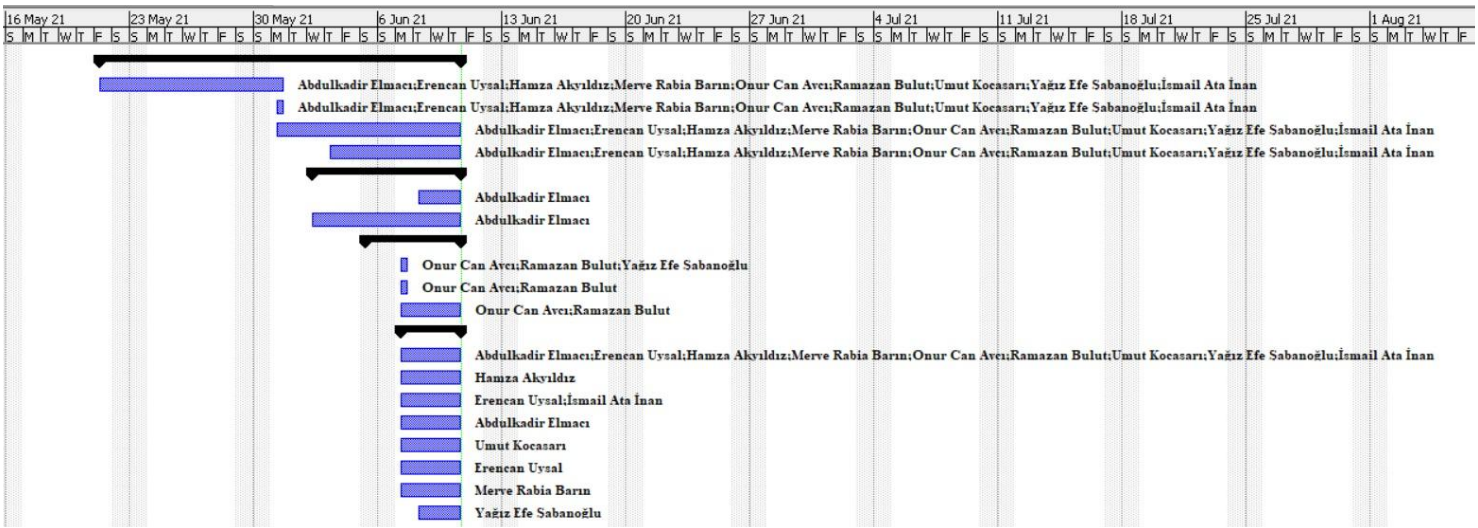
In the backend part, we shared API functionalities that we had decided to use in Practice-App. Everyone had written unit tests for their parts of functionalities and documented them in a careful manner. The team knew that the proper documentation of endpoints was crucial for the frontend part. We had moved to the frontend part after testing the backend. Two of us handled the frontend part, one of us dockerized the application and the other one of us deployed the application in AWS.

While working with the team, we were careful about working in different branches for different issues to prevent great conflicts. After completing the code for an issue, we merged the related branch with the development branch after resolving conflicts. At least 3 members reviewed the pull requests.

Completing the process of developing, documenting and deploying a whole application has been a remarkable experience for the team before we move on to our actual project. We have clarified our way of working in Github and become aware of possible challenges that we can face in our actual project.

Project Plan

		Name	Duration	Start	Finish	Pr...	Resource Names
47		⊟API	15 days?	**5/21/21 8:00 AM**	**6/10/21 5:00 PM**		
48		Researching about API	7 days?	5/21/21 8:00 AM	5/31/21 5:00 PM		Abdulkadir Elmacı;Erencan Uysal;Hamza Akyıldız;Merve Rabia Barın;Onur Can Avcı;Ramazan Bulut;Umut Kocasarı;Yağız Efe Şabanoğlu;İsmail Ata İnan
49		Meeting for API Design	1 day?	5/31/21 8:00 AM	5/31/21 5:00 PM		Abdulkadir Elmacı;Erencan Uysal;Hamza Akyıldız;Merve Rabia Barın;Onur Can Avcı;Ramazan Bulut;Umut Kocasarı;Yağız Efe Şabanoğlu;İsmail Ata İnan
50		Implementation of API	9 days?	5/31/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı;Erencan Uysal;Hamza Akyıldız;Merve Rabia Barın;Onur Can Avcı;Ramazan Bulut;Umut Kocasarı;Yağız Efe Şabanoğlu;İsmail Ata İnan
51		Testing the API	6 days?	6/3/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı;Erencan Uysal;Hamza Akyıldız;Merve Rabia Barın;Onur Can Avcı;Ramazan Bulut;Umut Kocasarı;Yağız Efe Şabanoğlu;İsmail Ata İnan
52		⊟Deployment	7 days?	**6/2/21 8:00 AM**	**6/10/21 5:00 PM**		
53		Tried A Dummy Deploy, 8/6/21	3 days?	6/8/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı
54		Creating a Deployment Plan	7 days?	6/2/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı
55		⊟UI Design	4 days?	**6/5/21 8:00 AM**	**6/10/21 5:00 PM**		
56		Meeting	1 day?	6/5/21 8:00 AM	6/7/21 5:00 PM		Onur Can Avcı;Ramazan Bulut;Yağız Efe Şabanoğlu
57		Sketching	1 day?	6/6/21 8:00 AM	6/7/21 5:00 PM		Onur Can Avcı;Ramazan Bulut
58		Designing UI	4 days?	6/6/21 8:00 AM	6/10/21 5:00 PM		Onur Can Avcı;Ramazan Bulut
59		⊟Milestone 2	4 days?	**6/7/21 8:00 AM**	**6/10/21 5:00 PM**		
60		Preparing Brief and clear summary of work done b	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı;Erencan Uysal;Hamza Akyıldız;Merve Rabia Barın;Onur Can Avcı;Ramazan Bulut;Umut Kocasarı;Yağız Efe Şabanoğlu;İsmail Ata İnan
61		Evaluation of tools and processes you have used t	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Hamza Akyıldız
62		Updating Summary of project and overall status	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Erencan Uysal;İsmail Ata İnan
63		Preparing URI of your deployed application	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Abdulkadir Elmacı
64		Preparing Basic functionality of your project	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Umut Kocasarı
65		Preparing Challenges you met as a group	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Erencan Uysal
66		Preparing List the status of deliverables.	4 days?	6/7/21 8:00 AM	6/10/21 5:00 PM		Merve Rabia Barın
67		Updating Project Plan	3 days?	6/8/21 8:00 AM	6/10/21 5:00 PM		Yağız Efe Şabanoğlu



Summary of Individual Works

Member	Work
Abdulkadir Elmacı	- I implemented the follow.py file which is follow actions for this API - In follow.py, I wrote the follow(usernameOfFollower, usernameToFollow) as a POST method. It takes two path parameter and follows or sends a follow request for the given usernames if they are present. If they are not present a 404 error is produced. - I also wrote 3 GET methods. Each of them returns a JSONArray which has the username of followers, followings or follow requests. They all take a path parameter which is the username. If there is no such user, 404 error is produced - I wrote some logging methods for the exception handlers. Rabia mainly wrote the exception handlers and I added some logging support. - I integrated Sentry to our application to track our errors. The Sentry integration packages the exceptions, checks if they are handled, which IP produced the error, when did error happen etc. Simply all the good information we need to understand what caused the error. We can check which part of the code is problematic by checking the Sentry dashboard. - I added a 3rd party API call to our root of the API. It counts the hits to our API by simply calling a get method for given URI. For more information: https://countapi.xyz - I wrote unit tests for the follow.py file functions which are in test_follow.py and test_followDetails.py. Test_follow.py file tests the follow action and test_followDetails.py file tests the GET methods. - I helped Umut to integrate Swagger into our app. We wrote the first examples to give some idea to other members.
Yağız Efe Şabanoğlu	- I implemented the location.py file which is adding new locations for this API . In follow.py, I wrote the location(address) as a POST method. It takes an address parameter and create a location if address are present on map. If address is not present a 404 error is produced. ● Swagger paths for POST method: Swagger Link ● URI of get comment functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/locations/{address}**. address is string. You can test it with setting address to Hostapark Hotel.· ● Finally this method creates a new location: ● {'location_name' : 'Hostapark Hotel , 'latitude' : '36.8041565', 'longitude': '34.6244029', 'relatedLocations': ['Mersin','Hostapark Hotel','Kardelen

	Hotel’,’Othello Hotel in Mersin Mediterranean’, … too much hotels … , ‘Garanti Mersin Şubesi’, ’Beko’, ’Grand Ezel Hotel’, ‘Sok’], ‘status’:200 , ‘story_ids’:[]} ● I added a 3rd party API’s call to our root of the API. First one is Geocoding API which takes an address as a parameter and gives its geolocation informations as a return (latitude,longitude): https://developers.google.com/maps/documentation/geocoding/overview ● Then second one is Nearby Search API. This takes latitude and longitude of specific location and gives nearest locations on the map near as a return: https://developers.google.com/maps/documentation/places/web-service/search ● I wrote unit tests for the location.py file functions which are in test_location.py tests the add new location to database. ● Also, I updated the works done in the time between milestone 1 and milestone 2 in the project plan. I deleted the lines that were not done and added the ones that were done. I edited the time ranges, contacts and links between lines.
Erencan Uysal	- I did research about Flask, MongoDB, Docker and Swagger. - I did research about unit tests. I also learned how to use mock in unit tests. - I implemented comment functionality in this API. - In comment.py, I wrote the getComments(postId) function as a GET method. - URI of get comment functionality : ‘**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/{postId}/comments**’. PostId is an integer. You can test it by setting postId to 1. It should return a list of comments. If you test it with 777 (this post does not exist) you can see a 404 error message. - I wrote the postComment(post,username) function as a POST method. - This method creates a comment like : {‘username’ : ‘atainan’ , ‘comment’ : ‘A great story’, ‘date’: ‘11/11/2021 10:45’, ‘language’: ‘en’} - URI of post comment functionality : ‘**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/{postId}/comments/new/{username}**’. PostId is integer and username is string. You can test it by setting postId to 1 and username to atainan. You should also give comment text as form data. If you test it with a user or post id that does not exist in the database, it will return 404 code with a message. - Comment texts are received from users via form data. I used “detectlanguage” as third party api. I am sending a post request to URI (**https://ws.detectlanguage.com/0.2/detect**) with the comment text given by user. If it finds languages related to

	this text, it returns language codes with reliability rates. I take the most probable language and use it while creating comments. - I wrote unit tests for comment.py in test_comment.py. You can also run this file in docker and see test results. - I also tried to dockerize react for this project. I tried a lot but kept getting errors. Then I asked my teammate İsmail Ata and he managed to dockerize it. - Abdulkadir and İsmail Ata helped me a lot on swagger and mock issues. I had no previous experience but I learned these things thanks to them. - I wrote "summary&overall status" and "challenges" parts of this report
Hamza Akyıldız	- I implemented home.py file - In home.py, I wrote getHome(username) as a GET method, where it returns a json array containing the post that is related to the given username. - URI for home API swagger doc is Swagger for home API - URI of View Home functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/home/{username}**' , can be tested with setting {username} to 'atainan' - I implemented savePost.py file - In savePost.py, I wrote savePost(username) as a POST method where it adds the post id, from a request as a body parameter, to the user's savedPost lists in the db. - URI for savePost API swagger doc is Swagger for savePost API - URI of savePost functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/{username}/savings**' , can be tested with a post request setting {username} to 'atainan' with body parameter which is a post - For third party API i uses an endpoint which randomly returns advice - URL of advice generator: https://api.adviceslip.com/advice - I wrote unit tests for the APIs which I implemented. - In test_home.py, I tested the getHome(username) function which is a GET method. In test_savePost.py, I tested the savePost(username) function which is a post method. - To understand the boundaries of unit testing, I did research on the Internet. Kadir's unit tests were a good lead to start writing. Mocking the data is the main topic that i focused to learn during the research - Research about web frameworks. I started to learning about django with a youtube playlist Django - We decided to use Flask, therefore i made research about Flask. And Ata gave us a little introduction to Flask, which was very helpful.

	- I tested the dockerized file before merging with the development and informed the team members if their parts have any problem. - Ata Inan dockerized the app. In a group meeting, he talked about how the dockerization works and related fields in the docker file. - I wrote the evaluation of tools and processes related to the implementation part for Milestone Report 2.
İsmail Ata İnan	- URI of View Profile functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/user/{username}**' , can be tested with setting username to 'atainan' for example. - This functionality utilizes GET request and the name of the function is **getProfile(username)** in **profile.py** - URI of Update Profile functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/user/{username}/update**' - This functionality utilizes POST request, so the request should have a suitable body to test. But it can be tested using Swagger documentation with again setting username to'atainan'. The name of the function is **updateProfile** in **profile.py** - For third party API, I used Places API from Google Maps API. I utilized **Place Search** and **Place Details** services from there. - URI of **Place Search** : '**https://maps.googleapis.com/maps/api/place/textsearch/json?query={location}&{APIkey}**', where location is a string describing any location and APIkey is the API key obtained from Google Cloud Platform - URI of **Place Details :** '**https://maps.googleapis.com/maps/api/place/details/json?place_id={placeid}&{APIkey}**' , where placeid is the place id obtained from the response taken from **Place Search** API request and APIkey is the API key obtained from Google Cloud Platform - For both of my Google Maps API requests, I utilize a GET request and all these requests are done in the function **getLocationInfo(location)** in **profile.py**. The responses taken from Google Maps API are also filtered in this function, and the filtered results are returned from this function to be used in **getProfile(username).** - Additionally, I took critical responsibility at the framework, database and Docker tasks. - I set up the Flask application from scratch, constructing an overall body for backend of the application and writing all necessary files to execute a simple application. Therefore, I found the server for the application. I also prepared a demo and presented it to the team to show how use and run it. I explained them the file structure so everyone can understand where to implement their codes and how to connect it to the main application.

	- I set up the whole database from scratch and connected it to the Flask application. I firstly connected a DynamoDB to the Flask application, but we later decided using MongoDB so I set up the environment which Flask and Mongo will work in accordance. I directly installed a Mongo Docker image and configured a docker compose file to run it in a container. Then I connected its port to our main Flask application. - I set up all the Docker system for our application. I dockerized every component of our application including backend, frontend and database. I dockerized whole Flask application by configuring docker compose file and Dockerfile designed specially for Flask. I implemented them from scratch and created some dependencies between containers with help of Kadir. In addition, I wrote the Dockerfile for React application and configured the docker compose file accordingly by making all necessary connections. In these parts, I got help from Onur since he knew how to run the frontend code. - After dockerization, I made the necessary changes in the code so that our backend and frontend components run in production mode from now on, not in development mode. This improvement made our whole application ready for deployment.
Merve Rabia Barın	- Researched how to install and run Flask. (https://flask.palletsprojects.com/en/2.0.x/installation/) Also learned Routing, handling HTTP requests such as GET and POST in Flask. (https://flask.palletsprojects.com/en/2.0.x/quickstart/#routing) - As a GET request I wrote View Post functionality in **viewPost.py**. Worked in the CB-5 branch before merging. I kept my commits up-to-date. - URI of **View Post** functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/viewPost/{postId}**'. To view details of the post, write the post's id in {postId}. - As a POST request I wrote Edit Post functionality in **editPost.py**. Worked in the CB-5 branch before merging. - URI of **Edit Post** functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/editPost/{ownerUsername}/{postId}**'. Give username of the post in {ownerUsername} and post id in {postId}, also send attributes that are desired to be edited in the json body. It updates the given post with given values in the database. Example json body is {'story': 'I changed it'}. - Researched and decided to use 'Similar Words API' from (https://rapidapi.com/wordgrabbag/api/similar-words/) as an

	external API. - I am sending a GET request to take similar words to a post's tags. I integrated this API call into the View Post endpoint under **callSimilarTags(post)** function. View Post returns 'similarTags' key with other post details. - Example API call URI of **Similar Words API :** send GET request to 'https://similarwords.p.rapidapi.com/moar?query={word}' by giving headers = {'x-rapidapi-key': {apiKey}} - Wrote Unit Tests for 'View Post' and 'Edit Post' functionalities in **test_viewPost.py** and **test_editPost.py**, respectively. Mocked the data for tests and tested each function of 'View Post' and 'Edit Post' endpoints. Before merging, I worked in the CB-17 branch. -Documented 'View Post' and 'Edit Post' endpoints in Swagger, wrote **viewPost.yml** and **editPost.yml** files respectively. Fixed a bug where I wrote 'int' instead of 'integer' and made Swagger documentation work properly. - Researched how to write Custom Error pages in Flask. Wrote a blueprint for global Error Handlers in **errorHandlers.py** to handle HTTP 400, 403 and 404 error codes and integrated it into **app.py**. I worked with Kadir and I edited my code based on the feedback I received from him. We worked in the CB-13 branch before merging. - I pulled the dockerized version of our app into my local, realized import errors in test cases and informed my teammates. We fixed import errors and made unit tests work without a problem.
Onur Can Avcı	- I researched and created Github projects with Abdulkadir for our projects because it facilitates and automates the development process and it enabled us to agile working. - URI of GET likes post functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/post/{postId}/likes**' , can be tested with setting postId to '2' for example. - URI of Like Post functionality : '**ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/post/{postId}/likes/{username}**' - This functionality utilizes POST request, so the request should have a suitable body to test. - Documented Get Likes of Post' and 'Like Post' endpoints in Swagger, wrote in likes folder **getLikes.yml** and **likePost.yml** files respectively. Fixed a bug where I wrote 'int' instead of 'integer' and made Swagger and schema of getLikes. Finally, documentation of getLikes and likePost api work properly. - Wrote Unit Tests for Like Post' and 'Get Like of Posts' functionalities in **test_likes.py**, respectively. Mocked the data for tests and tested each function of 'Get Likes' and Like Post' endpoints. Before merging, I worked in the

	CB-32 branch. - The name of the function is **likePost** in **likes.py** - For third party API, I used - I created React project with **create-react-app**(https://create-react-app.dev/docs/getting-started). - I create a structure for server side of react project. I add **axios**(https://www.npmjs.com/package/axios). - I create a structure for routing, I added **React-Router-Dom**(https://reactrouter.com/web/guides/quick-start) for routing in application. - I also, create a structure for ui design system and components. I added **Material UI** library(https://material-ui.com/). We use Material UI components in our projects. - I also worked in Flask and React at the same time with different ports. - I help us to Ata in dockerizing our React projects. After Ata completed dockerizing, he expalin us to how to dockerize whole projects and then I tested in my local projects. He change starting projects configuration and he automates the running app configs. - Fix cors problem in frontend project. I add cors config to Home api.
Ramazan Bulut	- I wrote POST request endpoint for post creating process that checks input firstly, then creates an post object in DB - I wrote GET request endpoint that returns object with id corresponds to give id input - I used a 3 party API called Weather-API that takes geolocation and returns an id field of this location. However this API needs another request to another endpoint to return weather information. So i made 2 request to this API to get exact weather forecast information of location - For documentation, we used swagger so I wrote a yml file to document my endpoints' input and output information. - For the front-end, I created a profile page for our mock user and designed some components like post with like and comment features that shows image and comments of post. We adopted component style design for the front end because we try to use the same components in different pages. - I created and reviewed some PR on Github. Some of them determine some code mistakes.
Umut Kocasarı	- The GET request I have written is in **search.py.** Its functionality is to search for a specific location and return recommendations and an exact match if there is one. - URI of search functionality: '**http://ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/search/{searchText}**', the search text should be a string and preferably a name of a location.

	- This function, **search(searchText)**, takes an input string. It finds the closest location names from the database given the text **searchText** , then finds the closest 5 location names and returns them. It also returns the name of the closest location if there is an exact match. - The detail could be seen in CB-4 issue and pull request. - In this function, I use a third-party api. The link of the api website is https://detectlanguage.com/ and I send my GET request to https://ws.detectlanguage.com/0.2/detect with a query parameter '**q**' and **Bearer authorization key**. - This third party api returns the language of the query string and in my search functionality, I only accept query words from English and Turkish, while eliminating others and returning 'Wrong Input Language' in my api. - The detail could be seen in CB-4 issue and pull request. - The POST request I have written is in **reportedUser.py**. Its functionality is to add a userId to reports collection in the database . - URI of report user functionality: '**http://ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/api/report/{userId}' ,** userId should be an integer corresponding to the id of the reported user. - The function, **postReport(userId)** , adds userId to reports collection and returns the userId back, which means the operation is successful. - I have written api documentation for both of my apis using swagger and they can be reached from '**http://ec2-35-157-237-50.eu-central-1.compute.amazonaws.com:5000/apidocs/**' - I have written unit tests for my functionalities, which are located in **test_search.py** and **test_reportedUser.py**. I tested **search(searchText)** in **test_search.py** and **postReport(userId)** in **test_reportedUser.py.** - I have searched for Flask-Swagger, https://github.com/flasgger/flasgger , and how we could implement it easily in our code with Abdulkadir Elmacı. We prepared two samples for our functions and made other team members' jobs easier. - I have learned the details of unittest and read many parts from websites such as https://docs.python.org/3/library/unittest.html . I created my initial unittest code for my both **search(searchText)** and **postReport(userId),** which became beneficial for other team members. - I created a Gmail account and AWS account for the team. - Helping Onur Can Avcı while preparing the README.md file and commands inside of it.

Evaluation of Tools and Processes

GitHub-Git

GitHub is an interface for GIT version management systems with some extensions. Git and GitHub were the keystones of our development phase since we need concurrent access to our works. It helps us to implement the project concurrently. We can work with the latest version of the project in our locals and share the work with others through Git. In GitHub, we can review the collaborator's work and give feedback to them via pull requests. Acknowledging Git working systems was challenging for the ones who encountered Git for the first time. However, the group members with experience with Git guide the others via our communications channels. We made effective use of Git and GitHub.

Python

Python is an interpreted high-level general-purpose programming language. Python has lots of open-source libraries, which helps us improve the implementation. We had some troubles with relative imports, which is a common problem in python applications. We omitted the relative imports and used absolute imports.

Flask

Flak is an easy to use python-based web framework. Flask has a simple architecture that helps us understand the basics of software developments. Since it has lots of libraries, we can focus on development rather than implementation. We did not face any challenge based on Flask.

Unit Testing

Unit testing was a challenge. First, we should have to acknowledge the idea of unit-testing, which was our first mistake not to do it. We first implement testing scripts, but they are not unit tests but integration tests. After the review of the first test scripts and meeting with the TA, we tried to refactor the tests. We mocked the database connections, which is the second challenge in unit testing. Finally, we managed to write unit tests that can properly test our units.

Docker

Docker is OS-level virtualization to deliver software in packages called containers. We used it with a MongoDb image for the database. And before deployment, we dockerized our app. Dockerization causes the file system to change. Therefore, we need to refactor the imports in our app.

React

React is an open-source JavaScript library for user interface design. We used it to design a simple user interface. Since we did not dive into a complex design, we did not encounter any challenges using react.

MongoDB

We use MongoDB for our database. Python database library has nice built-in functions for database creation and query writing in mongo. Since mongo offers a nice architecture for databases, we did not encounter any problem with our relations.

Swagger

Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON. We used it to document our APIs. Since we do not design APIs with complex parameters, our swagger documents are simpler.