CmpE352 – Spring 2021

Milestone 2 Report

Group 1

Group Members

Elif Akalın

Berke Argın

Hakan Balık

Emrah Doğan

Muhammed Göktepe

Cem Kaya

Emir Hıfsı Öztoprak

Bilal Tekin

Yunus Emre Topal

1. Executive Summary

1.1) Summary of project and overall status

1.1.1)Project Description

We are designing a social media platform that enables its users to create and share information in a community by creating posts in specific formats. Each user can create or join a community of their liking and create posts in some predetermined post format of their community. These predetermined post formats, which are called post templates, are created by community moderators and used in categorizing posts in that community. Each post template contains several compulsory or optional data fields to be filled, which can be of many types including location, image, video, text, date or a selection. Users can also filter posts based on their post templates and use queries on several data fields on that post template. Post template feature aims to provide an effective and accessible information sharing by providing a well-defined and characteristic outline for each piece of information in that community. Each post must belong to a community and put into that community's post feed. Users can also add other users as a friend after mutual consent and see their friends posts in their own user feed. Communities can be public or private, which in turn, will also define the privacy setting of the posts in that community. Currently, we aim to develop this platform both as a web and mobile application.

In summary, we aim to create a digital platform that enables its users to be a part of a purposeful community with each having their own specialized language for sharing information. In our platform, each community can define distinct and well-defined blocks of information which paves way for users to create specialized communities around a purpose.

1.1.2)Project Status

We developed the first version of our application. Basic features of application added. People can register the app by filling the necessary fields in the registration page. Users can login with their name and lastname. Users can create, join and leave communities. Users can share posts in communities. Users can create customized post templates in communities by their choice. In addition, we also created a Gannt chart for effort tracking, produced our second milestone report and we are ready to move forward through the development.

1.1.3)Moving Forward

Now we are ready to implement full features in our application. We learned how to develop a web application using Django in the backend, how to design pages in frontend and how to store datas in databases. We learned usage of git, pull requests and merging branches. With this knowledge we can move forward and implement all features with much better design.

1.2) URI of the Practice App

- The application is accessible at
  http://54.164.7.166:8080/
  It was deployed using docker and amazon AWS.

- The release v0.1 can be found at
  https://github.com/bounswe/2021SpringGroup1/releases/tag/v0.1

- Docker link is here :
  https://hub.docker.com/repository/docker/4teko7/practice-app

- Link to our Dockerfile :
  https://github.com/bounswe/2021SpringGroup1/blob/MainApp/practice-app/Dockerfile

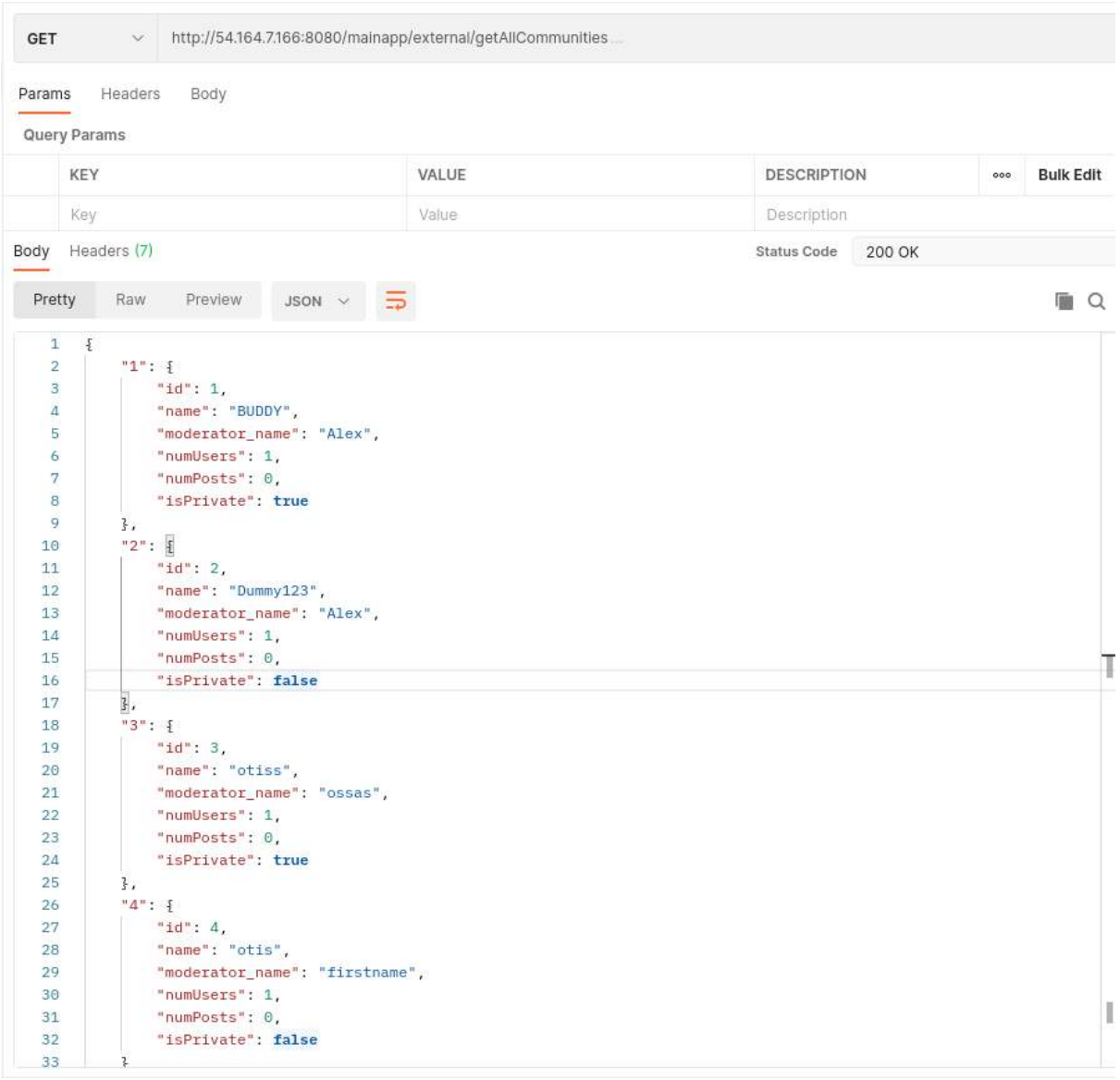
1.3) Documentation and URI of our RESTful API

These are the functions we have exposed to be used by third parties. The rest of the functions are designed for internal usage only, and documented in the

@Cem Kaya

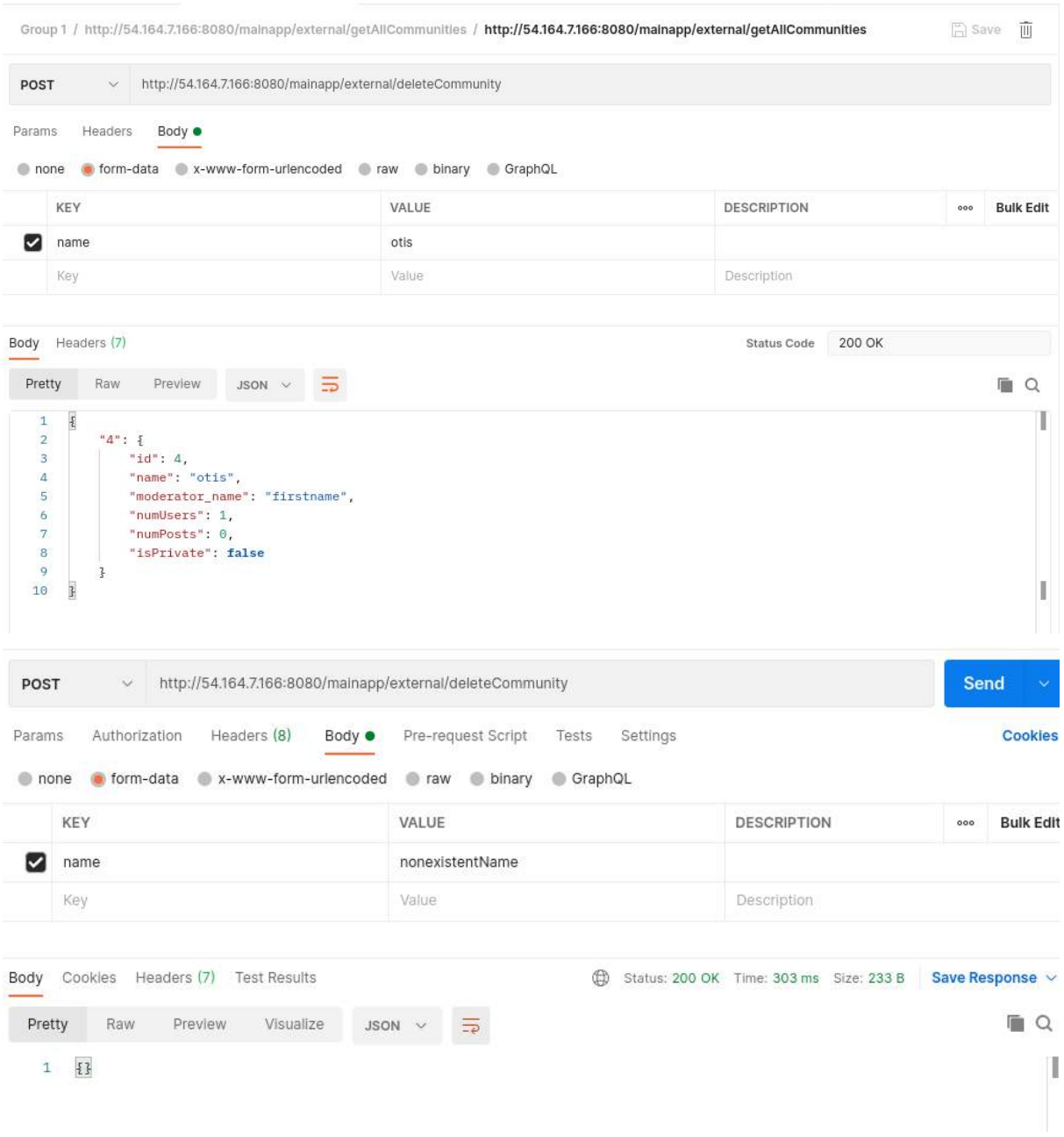
- http://54.164.7.166:8080/mainapp/external/getAllCommunities

Usage: Simple GET request is sufficient. A list of all communities are returned in JSON format.



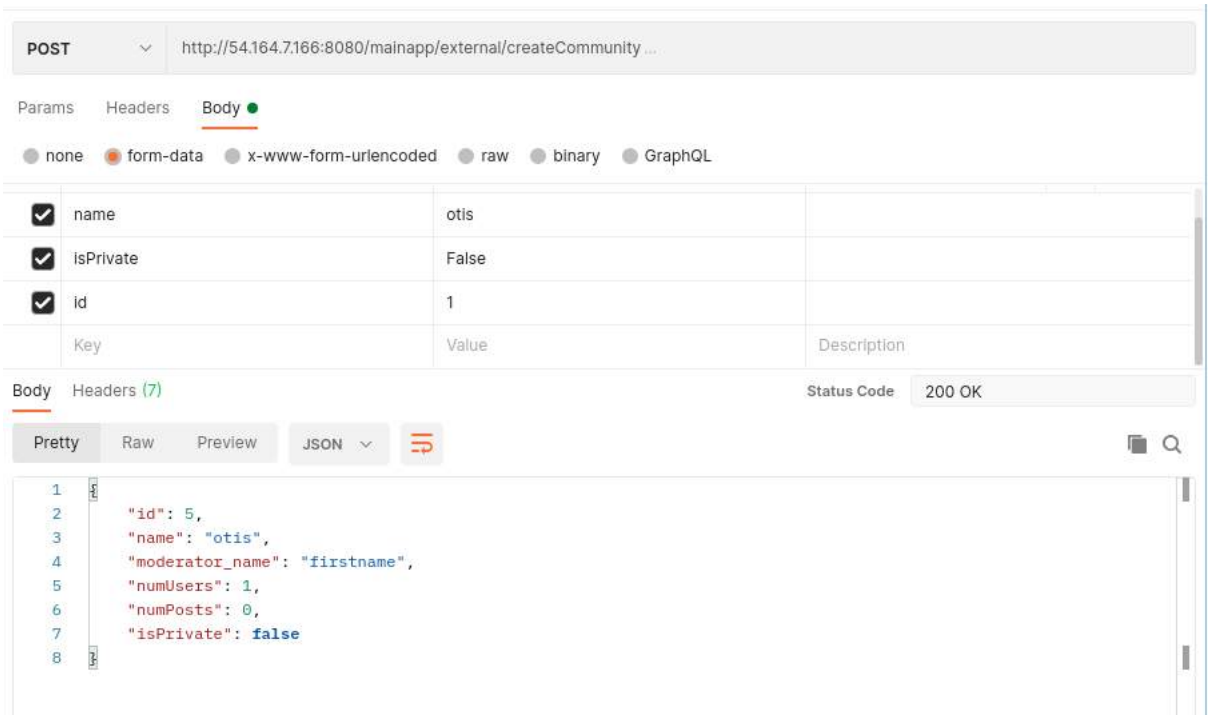
- http://54.164.7.166:8080/mainapp/external/deleteCommunity

Usage: A POST request is used. The parameter “name” must be supplied. All communities with the given name will be deleted as a result. The list of all deleted communities will be returned as JSON. If no communities are deleted, an empty JSON response will be returned.



- http://54.164.7.166:8080/mainapp/external/createCommunity

Usage: A POST request is used. The parameters “name”, “isPrivate” and “id” must be supplied. ‘Name’ is the name of the community to be created. ‘isPrivate’ can either be true or false. ’id’ is the id of the user that is to be the moderator of the created community.

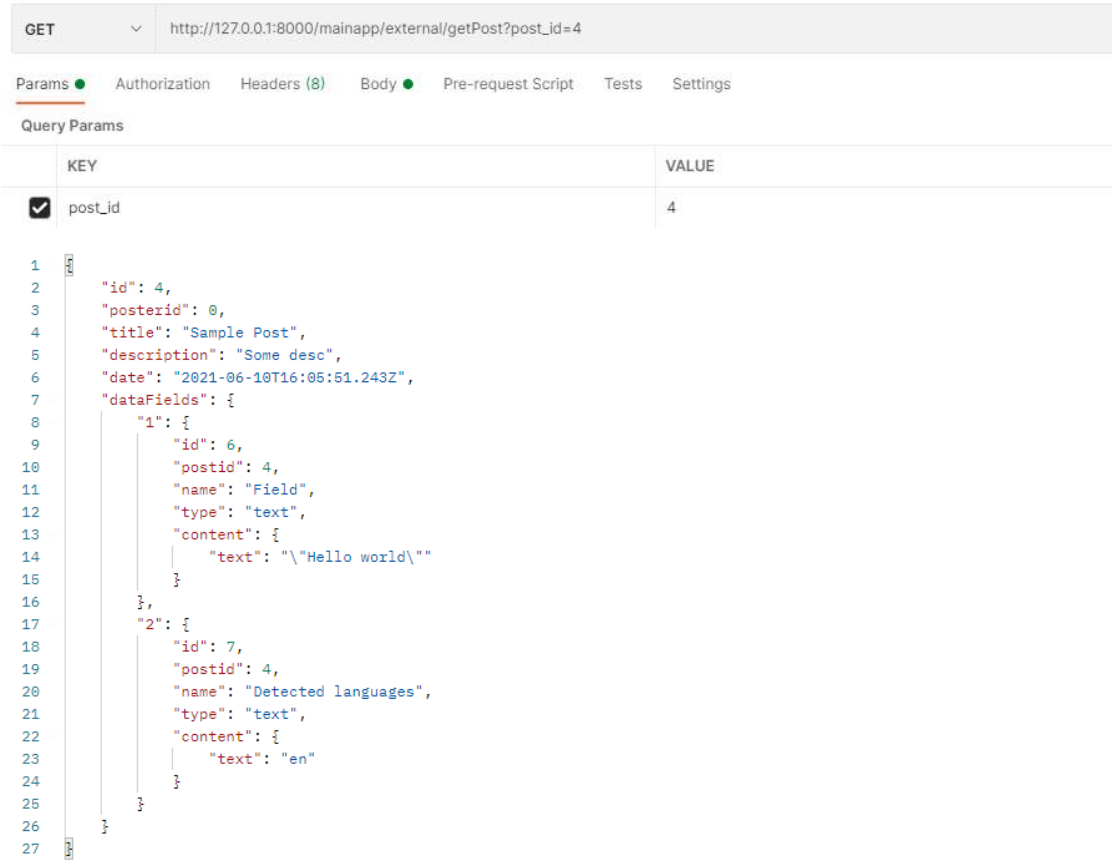


In addition to the postman screenshots attached, django unit tests also test the functionality of these API functions.

@Berke Argın

- http://54.164.7.166:8080/mainapp/external/getPost

Usage: Requires GET request and post_id parameter. Returns Post object in JSON format if post is not in a private community. Otherwise, returns {}



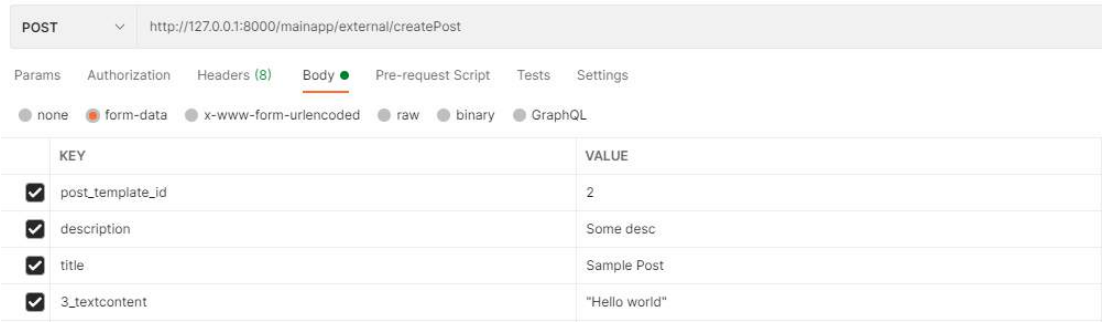
- http://54.164.7.166:8080/mainapp/external/getPostTemplate

Usage: Requires GET request and template_id parameter. Returns PostTemplate object in JSON format if post template is not in a private community. Otherwise, returns an error message or empty JSON object.

```
{
    "id": 3,
    "name": "Sample Template",
    "description": "Some description",
    "data_field_temps": {
        "1": {
            "id": 4,
            "name": "Sample video",
            "type": "video",
            "form_content": {}
        },
        "2": {
            "id": 5,
            "name": "Sample text",
            "type": "text",
            "form_content": {}
        },
        "3": {
            "id": 6,
            "name": "Sample image",
            "type": "image",
            "form_content": {}
        }
    }
}
```


- http://54.164.7.166:8080/mainapp/external/createPost

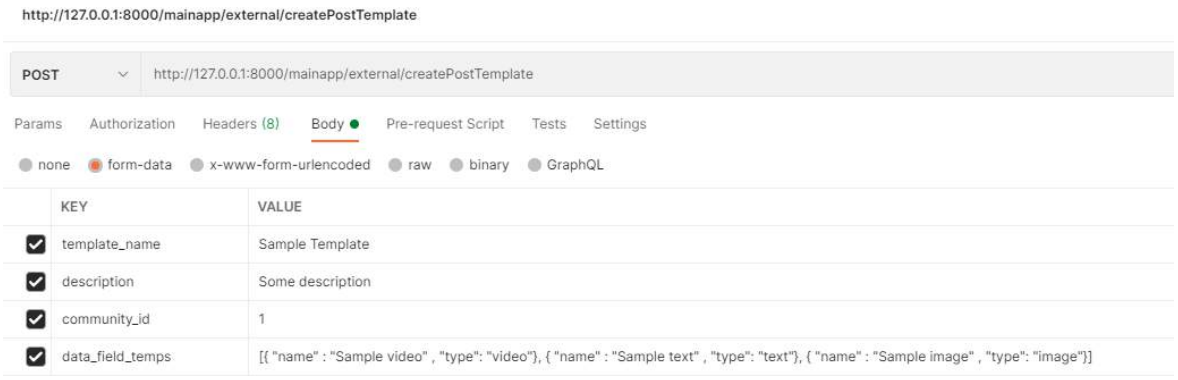
Usage: Requires POST request and fields “title” , “description” , “post_template_id” in the request body. It also requires fields of “i_textcontent” or “i_urlcontent” where i is the id of a DataFieldTemp of text type or video image type respectively. For video fields, urlcontent must be a Youtube video id. Returns Post object in JSON format if post template is not in a private community and fields are correct. Otherwise, returns {}



```
{
    "id": 4,
    "posterid": 0,
    "title": "Sample Post",
    "description": "Some desc",
    "date": "2021-06-10T16:05:51.243Z",
    "dataFields": {
        "1": {
            "id": 6,
            "postid": 4,
            "name": "Field",
            "type": "text",
            "content": {
                "text": "\"Hello world\""
            }
        },
        "2": {
            "id": 7,
            "postid": 4,
            "name": "Detected languages",
            "type": "text",
            "content": {
                "text": "en"
            }
        }
    }
}
```


- http://54.164.7.166:8080/mainapp/external/createPostTemplate

Usage: Requires POST request and fields “template_name” , “description” , “data_field_temps” and “community_id” in the request body. “data_field_temps” field must be of form “[{“name”:<dataFieldName>, “type”: <dataFieldType>}, …] where type can be “text”,”image or “video”. Returns PostTemplate object in JSON format if post template is not in a private community and fields are correct. Otherwise, returns {}



```
{
    "id": 3,
    "name": "Sample Template",
    "description": "Some description",
    "data_field_temps": {
        "1": {
            "id": 4,
            "name": "Sample video",
            "type": "video",
            "form_content": {}
        },
        "2": {
            "id": 5,
            "name": "Sample text",
            "type": "text",
            "form_content": {}
        },
        "3": {
            "id": 6,
            "name": "Sample image",
            "type": "image",
            "form_content": {}
        }
    }
}
```

@ Yunus

- http://54.164.7.166:8080/mainapp/external/getUserCommunities

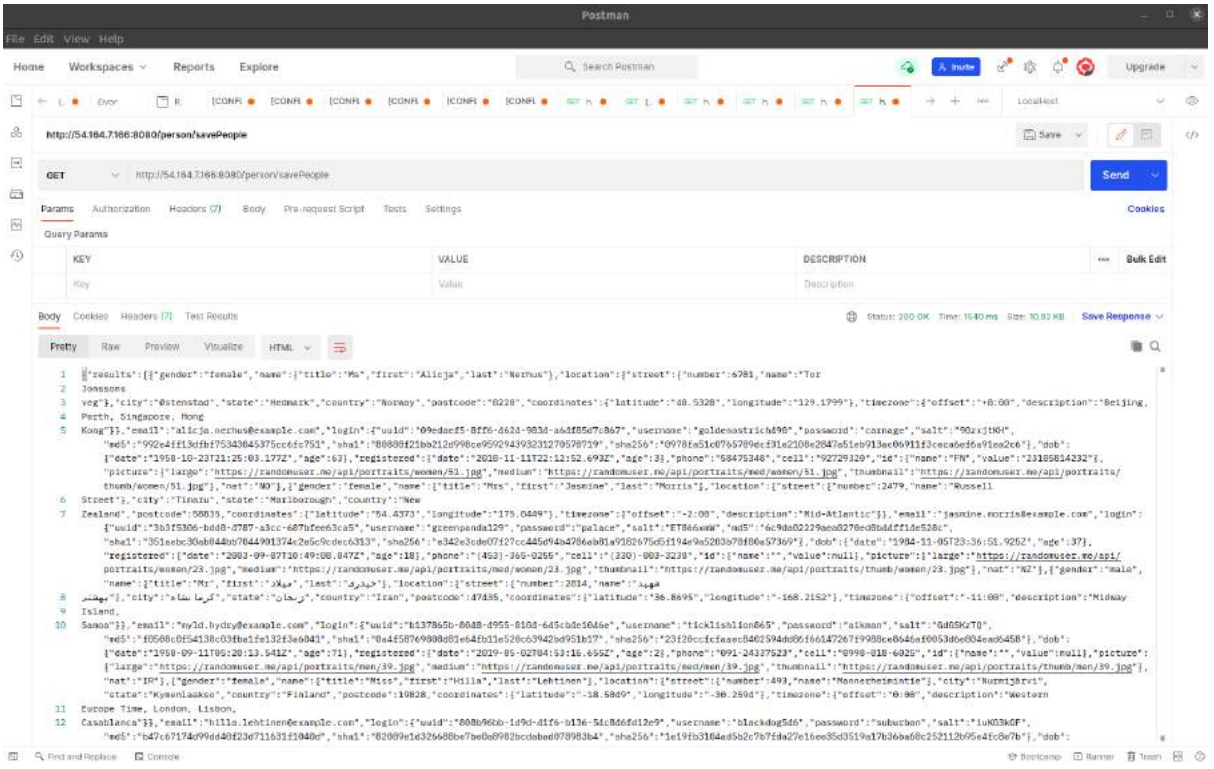
Usage: Requires GET request and field “user_id”. Returns communities of that user in JSON format. If that user has no communities or if a user with that id does not exist, return {}. If you do not provide a user_id, it will return {“Error”: “no user id provided.”}. If your request method is not GET, then it will return {“Error”: “Bad request.”}

- http://54.164.7.166:8080/mainapp/external/createUser

Usage: Requires POST request and fields “firstname”, “lastname”, “location”, “email”, “age”, “phone”, “imageUrl”. Only “firstname” and “lastname” fields are mandatory. Other fields have default values if you do not pass them ({“location”:”unknown”, ”email”,”{firstname}@mail.com”, “age”:0,” phone”:”05554443322”, “imageUrl”: “image.url”}). Creates a user with those properties in database and returns that user in JSON format. If you do not provide “firstname” or “lastname” or if your request method is not POST then it will return {“Error”: “Bad request.”}

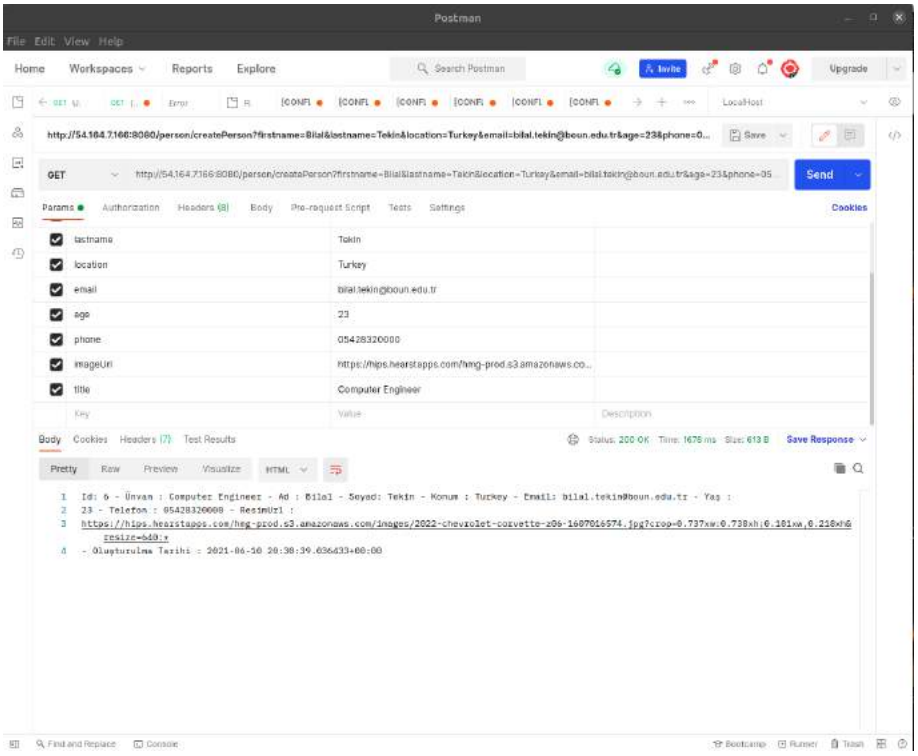
@BilalTekin

- USAGE: this url can be requested via postman. This is a get request. This will save some people to the database to use them for other functionalities. http://54.164.7.166:8080/person/savePeople



*** I have forget to change GET request to POST request for this API call

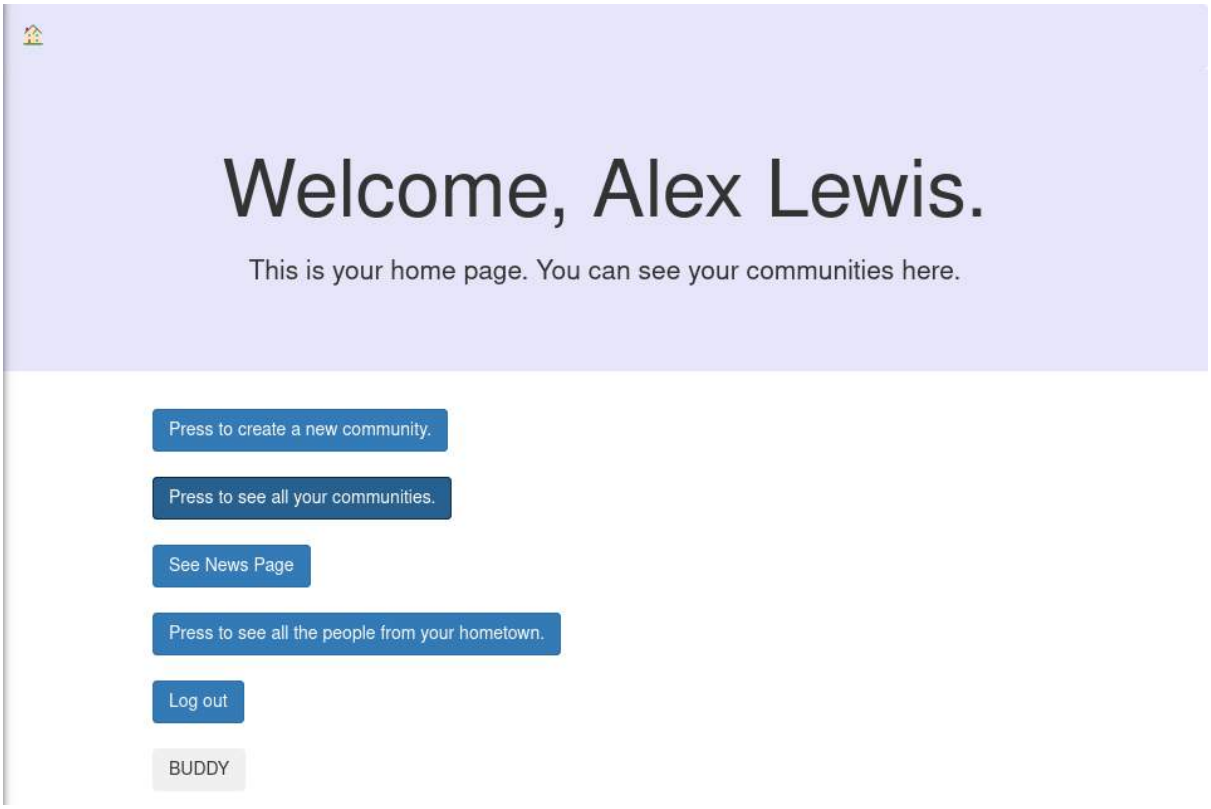
- USAGE: via postman we should put this url and select the get request. http://54.164.7.166:8080/person/createPerson?firstname=Bilal&lastname=Tekin&location=Turkey&email=bilal.tekin@boun.edu.tr&age=23&phone=05428320000&imageUrl=https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/2022-chevrolet-corvette-z06-1607016574.jpg?crop=0.737xw:0.738xh;0.181xw,0.218xh%26resize=640:*&title=Computer Engineer



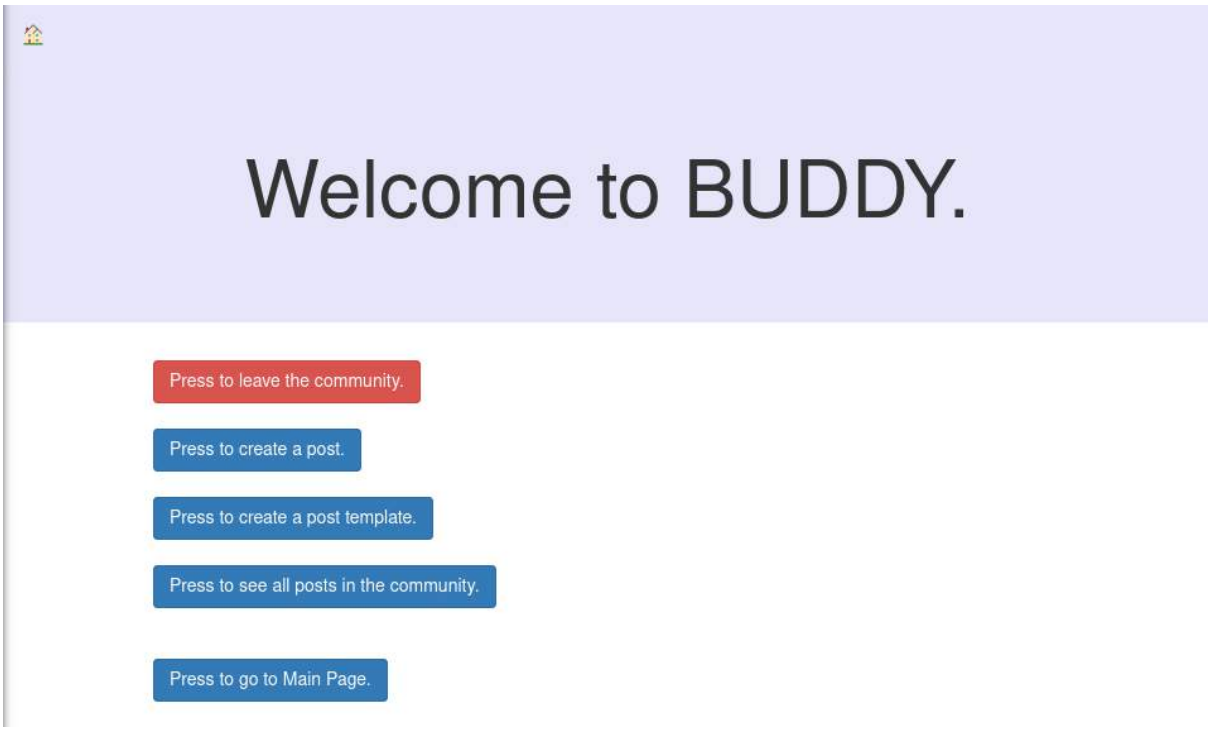
1.4) Functionality of the Project

In this assignment, we created a simple platform that allows users to create and join communities. Each community contains several, user created post templates, which can be used to create posts with specific format. Here below, you can find a manual that illustrates this,

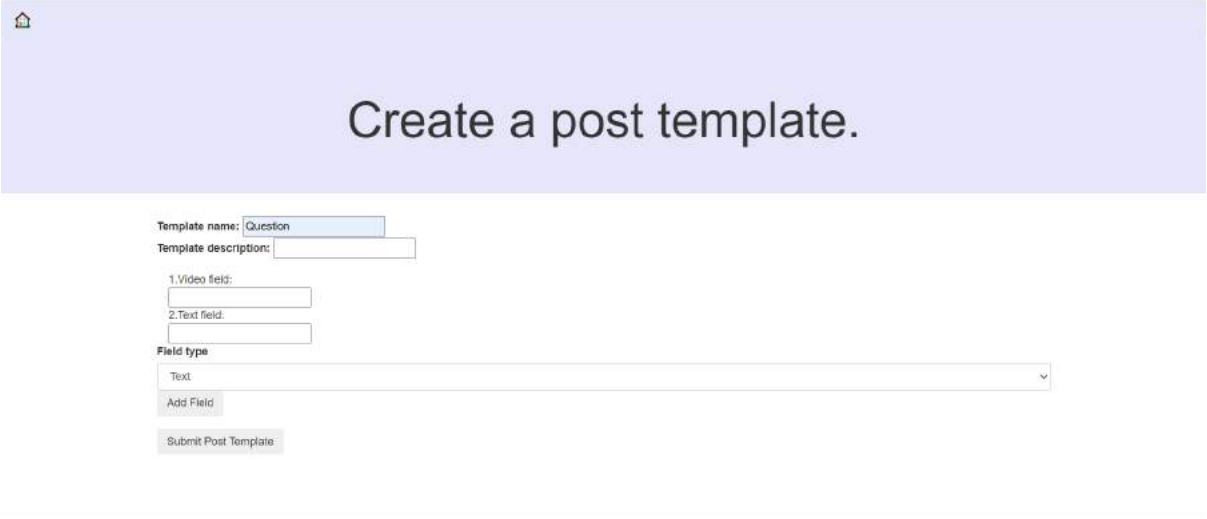
- Use the register button to register to the website.
- You will be redirected back to the login page.
- Use your name and firstname to login.
- You will see a screen like this:



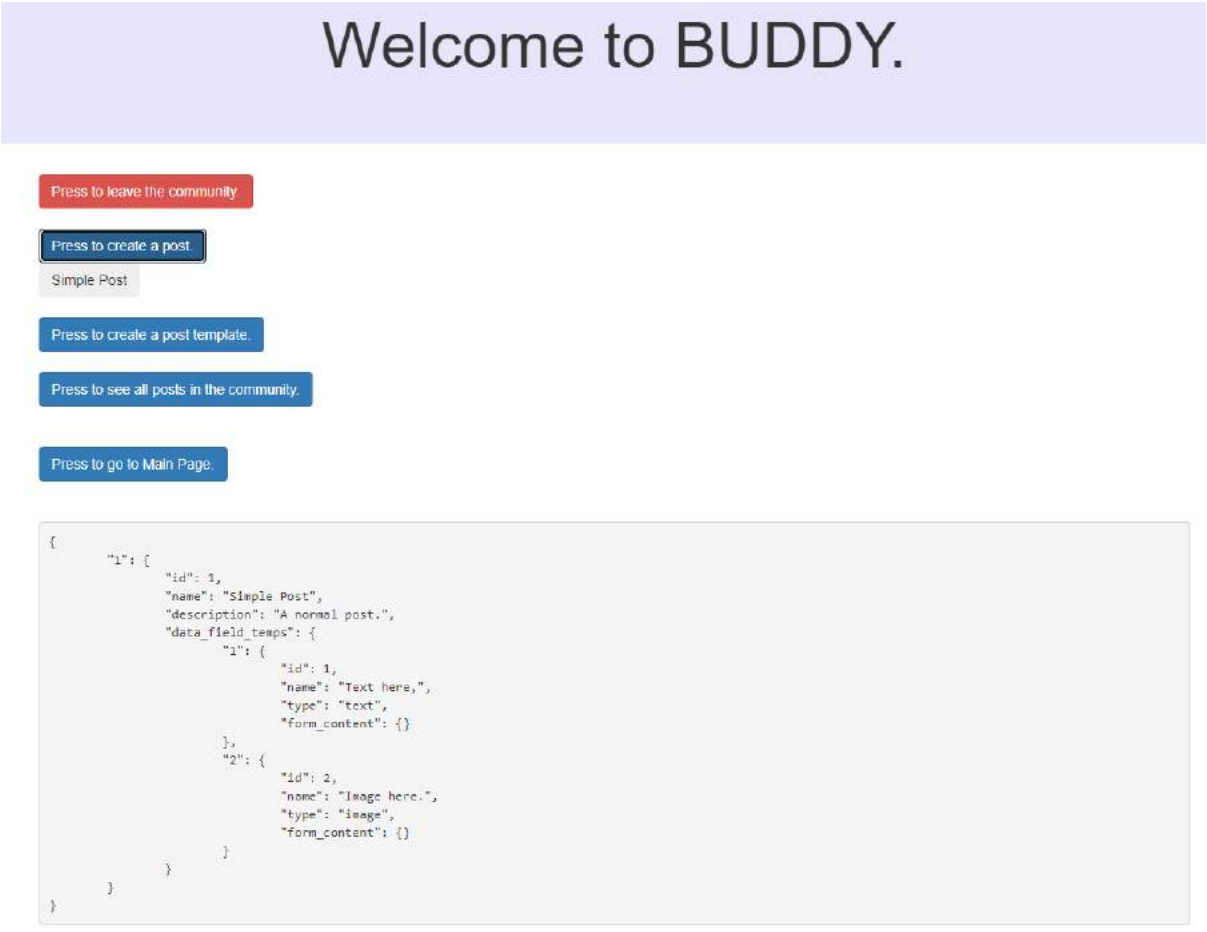
- Use the buttons to interact with the app. Their names are self-explanatory. You can press the home button on the upper left corner at any time to go back to your home page. Upon clicking a community (BUDDY in this case), you will see the community's homepage.



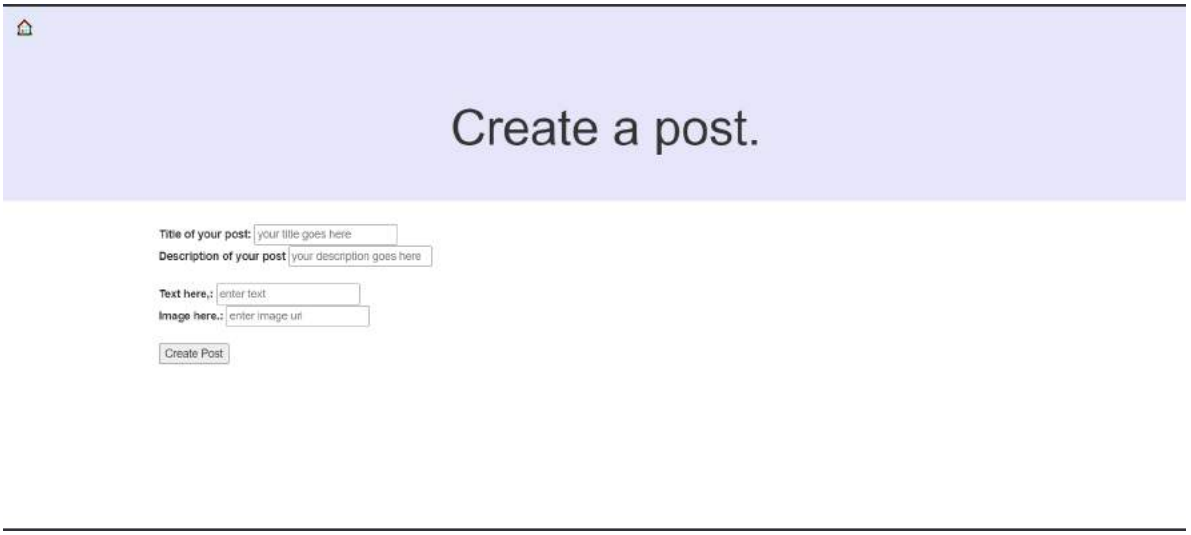
- To create a post template, you should click on button with "Press to create a post template"



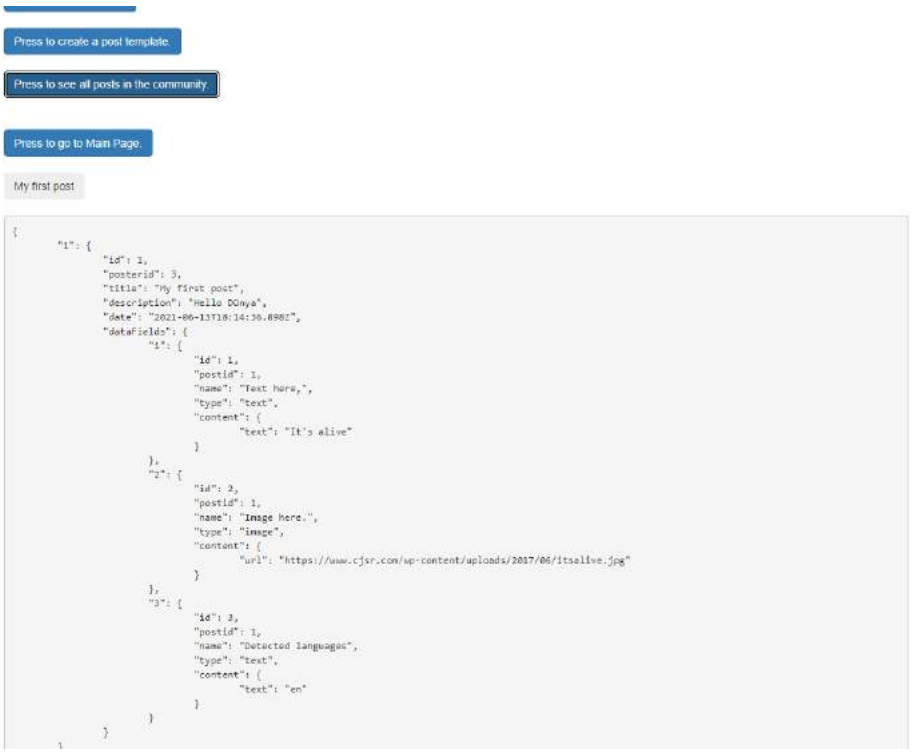
- You can add data fields to a post template by pressing the add field button. Currently, text, image and video fields are supported. Since a post in every community must make use of a template, at least one post template must be created by the moderator after creating a community. Once post templates exist in a community, they can be used in the ‘create a post’ part. Each data field should have a distinct name. In the image below, a post template with the title ‘Simple Post’ can be used to create a post in the community.



- You can see your post templates under the create post button. If there are no templates it will print “You need to create a post template first!”
- You will select the post templates and click it. it will redirect you to create a post page.



- You can create a post by filling the fields in that post template.
- If it is a text field, you should enter a text.
- If it is a image field, you should add an image url
- If it is a video field, you should enter a Youtube video ID. You should note that our internal API expects a public, embeddable video that is shorter than 5 minutes.
- After pressing the Create Post button, it will return a JSON object. If post creation is successful, it will be the Post object in JSON. If not it will be an error message.
- Either way, you can go back to the community page.



- You can see all of the posts in a community by pressing the button highlighted above.
- For each post in this community, there will be a button for each post that you can use to view that post in more detail.

Post Id: 1

Post owner: 3

Post Title: My first post

Post description: Hello Dünya

Text here,

It's alive

Image here.

Detected languages

en

- As seen above, a post is created with a text field and image field.
- This can be done with any post template.

1.5) Challenges met as a group

There were a lot of challenges in this development process. Learning and research phase took a lot of time as the assignment required the utilization of a lot of different tools and had many different aspects.

AWS:

AWS was very complicated in the beginning because There are many machines, their features, types, security, terminal commands etc. we again made a lot of searches to learn it. We bought a free tier machine and installed docker on it using docker and ubuntu notes. Then security is another problem because we don't know why our application didn't work. We searched to solve the problem, and we realized that the problem was about security of EC2 instances. We should have opened the corresponding ports to reach our working app via browser.

DOCKER:

After that, we should have learned about Docker to dockerize our project. It was another hard task to complete. It was very complicated at first, It took about 3 days to learn it enough to use it in our project. In this process, again we watched lots

of youtube videos and udemy videos, and looked at official documentation as well as Docker hub. Took lots of notes and shared them with the team via discord.

DJANGO:
As a team, learning Django took a significant amount of time as most of us were newcomers to web development. For that reason, we had some challenges meeting some deadlines at the early stages due to inexperience. In addition, as Django contains many features and middleware such as session, CSRF tokens, we had to make some practice beforehand to use them correctly.

JAVASCRIPT
We also had to use Javascript to implement some frontend functionalities as well as to send messages to backend. Since most of us had no prior experience, we had to learn it from scratch and had some troubles implementing several methods. As an example, we couldn't figure out a way to send a POST request with the appropriate body format that Django accepts. There were also a lot of bugs in most of the scripts in some html files, which took a considerable amount of time to fix.
GITHUB
We initially had some trouble organizing issues and merging branches as some branches had unnecessary conflicts in irrelevant files which should be in .gitignore. We also lost some files due to a bad merge operation in the early stages. However, as the project moved on, we gained a deeper understanding of these Github features and learned effective usage of these systems.

1.6) List and Status of deliverables

Deliverable	Status	Frequency	Description
Practice App	Complete	Completed	This project is a basic implementation of our main project. It includes basic features of a purposeful community app.
Milestone Report 2	Complete	Completed	This Document

2. Project Plan

2.1) Project Communication plan

Participants	Place	Purpose	When
Team Members	Zoom	General Discussion	Every Friday at 20.00

Team Members	Slack	Weekly Progress Review	When Needed
Team Members	Discord	General Discussion	When Needed
Team Members	WhatsApp	Urgent Communication	When Needed

2.2) Roadmap

47		**Preimplementation Research**	7,667 günl...	**10.05.2021 08:00**	**10.06.2021 00:00**	
48		Research on API and web technologies	3,333 günler?	10.05.2021 08:00	13.05.2021 16:00	Everyone
49		Research on Django, Springboot, Nodejs	3,333 günler?	10.05.2021 08:00	13.05.2021 16:00	Everyone
50		Meeting Notes 8	3,375 günler?	10.05.2021 08:00	13.05.2021 17:00	Cem Kaya
51		Searching necessaary and purposeful APIs	3,375 günler?	10.05.2021 08:00	13.05.2021 17:00	Everyone
52		Research on Mongodb and Sqlite	3,333 günler?	10.05.2021 08:00	13.05.2021 16:00	Everyone
53		**Pre Implementation**	2,333 günl...	**13.05.2021 08:00**	**22.05.2021 17:00**	
54		Watching Django Tutorials	9,375 günler?	13.05.2021 08:00	22.05.2021 17:00	Everyone
55		Creating Basic Django Project	9,375 günler?	13.05.2021 08:00	22.05.2021 17:00	Everyone
56		Creating Community API Template	2,333 günler?	13.05.2021 08:00	15.05.2021 16:00	Cem Kaya
57		Creating Person Template	2,333 günler?	13.05.2021 08:00	15.05.2021 16:00	Bilal Tekin
58		Creating Post API Template	2,333 günler?	13.05.2021 08:00	21.05.2021 17:00	
59		Meeting Notes 9	9,375 günler?	13.05.2021 08:00	22.05.2021 17:00	Bilal Tekin
60		**Implementation**	4,333 günl...	**22.05.2021 17:00**	**10.06.2021 00:00**	
61		Homa Page	4 günler?	22.05.2021 17:00	26.05.2021 17:00	Cem Kaya
62		Login	0,333 günler?	22.05.2021 17:00	24.05.2021 17:00	Cem Kaya;Yunus Emre Topal
63		Register	4 günler?	22.05.2021 17:00	26.05.2021 17:00	Yunus Emre Topal
64		**Community API**	0,771 günl...	**24.05.2021 08:00**	**26.05.2021 10:30**	
65		Create Community	0,724 günler?	24.05.2021 08:00	26.05.2021 09:22	Berke Argın;Cem Kaya
66		View Community	0,724 günler?	24.05.2021 08:00	26.05.2021 09:22	Berke Argın;Cem Kaya
67		Join Community	0,667 günler?	24.05.2021 08:00	26.05.2021 02:19	Berke Argın;Cem Kaya;Muhammed Göktepe
68		Leave Community	0,667 günler?	24.05.2021 08:00	26.05.2021 02:19	Berke Argın;Cem Kaya;Muhammed Göktepe
69		Delete Community	0,771 günler?	24.05.2021 08:00	26.05.2021 10:30	Berke Argın;Cem Kaya
70		**Person API**	0,667 günl...	**24.05.2021 08:00**	**26.05.2021 08:00**	
71		Create Person	0,667 günler?	24.05.2021 08:00	26.05.2021 08:00	Bilal Tekin;Yunus Emre Topal
72		**Post API**	0,266 günl...	**22.05.2021 17:00**	**24.05.2021 15:23**	
73		Create Post	0,266 günler?	22.05.2021 17:00	24.05.2021 15:23	Berke Argın;Yunus Emre Topal
74		View Post	0,266 günler?	22.05.2021 17:00	24.05.2021 15:23	Berke Argın;Yunus Emre Topal
75		Create Post Template	0,266 günler?	22.05.2021 17:00	24.05.2021 15:23	Berke Argın;Yunus Emre Topal
76		Get All Posts	0,266 günler?	22.05.2021 17:00	24.05.2021 15:23	Berke Argın;Yunus Emre Topal
77		**Deployment**	0,333 günl...	**09.06.2021 08:00**	**10.06.2021 00:00**	
78		Dockerizing	0,667 günler?	09.06.2021 08:00	10.06.2021 00:00	Bilal Tekin
79		Pushing AWS	0,667 günler?	09.06.2021 08:00	10.06.2021 00:00	Bilal Tekin
80		Writing Test Case	2,375 günler?	07.06.2021 08:00	09.06.2021 17:00	Everyone
81		Meeting Notes 10	1 gün?	05.06.2021 08:00	06.06.2021 08:00	Cem Kaya
82		Meeting Notes 11	1 gün?	07.06.2021 08:00	08.06.2021 08:00	Hakan Balık

83		**Third Party API**	4 günler?	**22.05.2021 08:00**	**08.06.2021 17:00**	
84		News API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Emrah Doğan
85		Youtube Data API and Youtube IFrame API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Berke Argın
86		GIPHY API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Muhammed Göktepe
87		Detect Language API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Emir Hıfsı Öztoprak
88		Detect Country from IP	4 günler?	22.05.2021 08:00	08.06.2021 17:00	Elif Akalın
89		Google Image Suggestion API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Cem Kaya
90		Cat Facts API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Yunus Emre Topal
91		Person API	17,375 günler?	22.05.2021 08:00	08.06.2021 17:00	Bilal Tekin

3. Summary of Work Done

Member Name	Contributions
Elif Akalın	• Made research on Django and RESTful APIs. • Made research on HTML, JavaScript. • I opened 2 issues. • I created a simple country API where the user can view the users that are in the same country as they are in.

	● I reviewed my teammate Emrah's code and after everyone's approval, merged it with our main project. ● I attended the PS sessions as much as I could. ● I made a pull request for my API. ● I attended the meetings and brainstormed and planned together with my teammates.
Berke Argın	● Researched Django Framework and RESTful APIs from the official documentations and links provided in the assignment description. ● Created a small app named post2 that features customizable post templates based on the works done by Yunus Emre Topal (post app). ● Merged community app and post2 app onto a single app mainapp with Cem Kaya over discussion in Discord. ● Made contributions to model structure of mainapp such as modifying Post model and adding PostTemplate, DataField and DataFieldTemp models. Also added many view functions such as createPost, createPostTemplate, getPost, getPostTemplate to improve on the post template feature. ● Learned HTML, Javascript through some internet sources including W3Schools and StackOverflow with helps from my teammate Cem Kaya. ● Designed viewPost.html, createPost.html, and createPostTemplate.html template pages for the frontend and added several Javascript functions to add UI features. ● Integrated Youtube Data API and Youtube Iframe API functionalities to our application which are used to determine video specifications and to create embedded players for video content. ● Wrote unit tests for Post and PostTemplate models that checked whether functionalities are working as expected. ● Added 4 API functions to our RESTful API, which are external_api_createPost, external_api_getPost, external_api_createPostTemplate, external_api_getPostTemplate. ● Tested both the API calls written by me and the oth Postman ● Learned about Docker and AWS with helps from the PS sessions and live tutorial sessions given by Bilal Tekin ● Used GitHub extensively, created issues, pull requests and reviewed codes in other pull requests. I tried to explain my work clearly through commits and create relevant issues to provide a project plan. ● Attended many meetings with my teammates and participated in discussions over implementation, project plan, documentation and work share.
Hakan Balık	●
Emrah Doğan	● I watched some tutorials about what an API is and what is RESTful API

	● I did some research about Django and script codes ● I did some research about HTML codes ● I did some research about Javascript ● I did some research on 3rd party APIs that can be used ● I added the News API in our application to get news top news from Turkey and create a news part. ● I added getTrNews, getFirst_news, getLast_news, createNews, createNews_ui functions ● I added createNews, newsPage HTML files ● I make a minor change on homePage HTML file ● I watched videos and read tutorials about Dockerizing and AWS ● I opened 2 issues ● I created Emrah branch from mainapp and then i implemented news API and then merged it to mainapp ● I attended all team meetings ● I attended all customer meetings ● I helped Bilal on Dockerizing and AWS ● I received the codes of my teammates ● I attended the PS sessions ● I made some research on Postman ● I learned Django forms for createNews HTML files and applied for saving created news from users in the database.
Muhammed Göktepe	● I watched some tutorials about what an API is and what is RESTful API ● I did some research about Django and Flask ● I did some research on Html and javaScript ● I created basic django app as a practice in my local ● I added leaveCommunity and joinCommunity functions ● I wrote leaveCommunity_ui.html in the front-end and added some javascript functions to handle clicks. ● I used GIPHY third-party API to show relevant leaving messages to our users ● I wrote test case for leaving and joining a community ● I did some research on postman ● I watched ps about Docker and Aws and made some practices ın these subjects ● I joined all weekly and urgent meetings ● I joined the meeting that Bilal Tekin talked about Aws and Docker ● I opened 2 issues about what functions that I am implementing. ● I made 3 pull requests and solved conflicts with Yunus, Cem and Berke. Then merged them into the MainApp branch. ● I reviewed many of the pull requests but just added one comment to pull requests from Emrah. ● I prepared a Project plan from the first milestone to now in project libre.
Cem Kaya	● I attended PS sessions that did not have a conflict with my schedule.

	● I did some research on Django, RESTful APIs, Javascript and Docker. ● I had no knowledge about databases. I researched databases before implementing a simple application in Django. ● I created a simple app named ‘community’ that implemented a minimal community model. ● I used the google custom image search API to serve image suggestions to the user during community creation. ● After the initial development phase that included simple users, posts, post templates and communities were completed, I worked with Berke to create the MainApp branch, where we merged all the features we have developed so far. We worked over discord chat and screen share. ● Did most of the initial frontend work (HTML, Bootstrap, JS). After this, many of my teammates used my html pages such as mainPage.html viewCommunity.html , createCommunity.html and the login page as a template to create new pages. ● As an overall overview, I created the Community model, related tests and backend functions, custom image search functionality, as well as most of the frontend. Please see github commits and relevant issues for a better overview. ● I created the MainApp-tests and MainApp-RESTful branches to develop unit tests and RESTful external API calls respectively. We worked together on these branches, after which they were merged to the MainApp branch. ● I reviewed many of my teammates’ pull requests, commented on them , as well as worked together over discord and zoom with them to solve merge conflicts. ● I have opened several issues and commented on others’ issues. ● I dockerized and tested the application.
Emir Hıfsı Öztoprak	● I attended some of the PS sessions, and also watched the recordings of the sessions I couldn’t attend ● I learned about Django since I didn’t have much experience with it before ● I joined our meetings on Discord and also joined the feedback meeting with our TA. ● I did some research on 3rd party APIs that can be used, and I also did research about Docker. ● I reviewed the pull requests of my teammates. ● I added the detect language part of the create user method using the Detect Language API. ● I created a branch named MainApp-emir and wrote my code in this branch. ● Then I did a pull request to the MainApp branch, my

	teammates reviewed my pull request and requested changes. ● I committed the necessary changes and then merged my branch into MainApp. ● I also helped Bilal with dockerizing our project. Then I also tested our deploy from my PC.
Bilal Tekin	● Researched about django and other requirements about it to show my team in a zoom meeting what I learned. ● I searched for a useful way to integrate it into our app. I was creating the first app in the project therefore I knew that the first app should have been very useful for all team members. I decided that a person API would be very effective for all other functionalities. As a result, I found an API which gives random people which have a lot of information in them. I took necessary information from them and saved it to the database. ● I created a fully working django app. We arranged a zoom meeting, and I showed all details of the working app to my team. For example, how to request to the django app, how requests are handled by it, how to create routers, return responses, create apps, use databases etc. This meeting took about 2 hours. Many details of a working application given to my team by me. With these, all members had the ability to create their own applications methods etc. ● I pushed the working code which has an app Person in it to the github master branch. By this, all members could continue on it and create their app easily. ● Created a requirements file for my team to be aware of the libraries that are necessary to run the application ● Test the application via postman to see if there is a problem or not. ● I checked all the codes created by my team and reviewed them by writing to them via whatsapp and discord. Sometimes we met on zoom and discord and discussed the codes written and pushed to git github. ● Opened more than 5 issues about requirements such as Dockerizing an application, EC2 instance, security, databases etc. ● While my team created their own apps and functionalities, I searched for Docker from various sources such as Udemy, youtube, official documentations stackoverflow etc. I took lots of notes and shared them with my team via discord. Some of the notes are: how to install docker on linux, how to create docker image, run images, push to dockerhub, pull from dockerhub, what parameters given as command do what etc. By these notes, all my team get much more familiar with docker. I created an example django app and dokerized it. I registered to the dockerhub and pushed the dockerized app to it. But there was a missing to run that app which is AWS-EC2. ● AWS was a new thing for me. To learn and tell about

	AWS and EC2, I searched from many sources like Docker. We decided to buy a free tier ubuntu machine. Then I searched how to connect to the machine, install Docker, run it in the background etc. I learned all of them. I pulled the dockerized app from Dockerhub and ran it in the background. But there was a problem, I couldn't reach my app. I searched to understand what the problem is. Finally I learned that there are some security issues about ports in the EC2 machines. To reach specific ports, we first of all open these ports as custom TCP. I knew that our app was working on 8080 port, therefore I opened that port in security which resulted in a completely working application. ● I edit the structure of github code. With this, the required folder structure was reached.
Yunus Emre Topal	● Attended every group and customer meeting. ● I did some research about the Django framework and learned how to create a login page and a couple other things thanks to my Cmpe 321 Assignment. ● Opened 5 new issues after the M1 report. ● Made 5 pull requests. ● Reviewed 10 pull requests. ● Fixed a bug in 1 pull request and helped 2 other pull requests with conflicts. ● Prepared unit tests to test creation of a Person. ● Made some research on Postman ● Created an external api call for creating a person with POST request that has some required and optional fields to fill in. Optional fields have some default values. If user does not provide required fields or request method is not POST, they will get json response that says Error:Bad request. ● Created an external api call for viewing communities of a person with GET request that has 1 required field to fill in which is user id.. If user does not provide required fields or request method is not GET, they will get json response that says Error:Bad request. ● I checked if my api calls are working as intended with Postman. ● Initially, I created Login and Post apps in our project. Post feature modified by my teammates later on. ● Post feature had these functionalities: create a post, view a post with a specific id, view all posts. ● Login feature lets a user log in if they are registered to the system, otherwise send them to the error page. ● On top of the Login feature, I added logout and registration features and created styled html files for them in the mainapp branch. ● Watched ps about Docker and made some dockerizing practices. ● I attended a ps session about Azure and cloud services. ● Used a third party api to show some cat facts on the registration page.

	• I wrote the Backend part of this Milestone report and filled personal parts of our reports. • In the “yunus” branch, I also implemented a feature to the posts but I couldn't manage to put it into our “MainApp” branch. • Also, i used a third party api, newsapi.org to create random posts with a title and description. • I added a ReadME file on that “yunus” branch because some of the features in that branch are not available in the “MainApp” branch. Link of the “yunus” branch: https://github.com/bounswe/2021SpringGroup1/tree/yunus

4. Evaluation of Tools and Processes Used

- **Github:** Github is the main platform where our project will be developed. We have only used the wiki and issue management features of github so far. It will be incredibly useful once we start coding, since it will let us keep record of our progress, and give the option to branch and/or merge as we try and add more features to our code base.
- **Slack:** We use slack due to its tight integration with other platforms. Channels and integration are the main features that make slack invaluable for us.
- **Zoom:** We use zoom for our weekly meetings, since everyone has zoom installed and configured on their computers for the lectures.
- **Discord:** We use discord for asynchronous communication. In addition to the channels, voice chat and livestream capabilities, we also use discord to communicate with our TA.
- **Google Docs:** Google docs was used to create this document. Similar to lucidchart, it allows multiple team members to work on the same document.
- **ProjectLibre:** We used the project management software ProjectLibre to produce a Gantt chart of our team effort until now. ProjectLibre is an open-source project management software that supports many features like Gantt charts, network diagrams and PERT graphs. While we are currently unaccustomed to using this software, we intend to learn and use it in the future.
- **Bootstrap:** Bootstrap is an open-source toolkit that facilitates quick design of responsive mobile-first sites. It is the world’s most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.
- **Django:** We chose to use Django Framework to develop our application as it provided many documentations and is relatively simple compared to SpringBoot. While there was a relatively involved learning process, we learned to utilize its many functionalities and gained a significant amount of knowledge. Designed primarily on the Model Template View pattern, Django provided us many easy to use features that had helped our development process greatly.
- **Docker:** Dockerizing an application means that if an application is working on your computer you can run it everywhere. This is done by creating an isolated container inside your local computer. All the required libraries and tools installed by docker in the created container according to commands you give in the Dockerfile. This created container is called image and this image can be deployed to any machine. If that machine has docker installed (it can be installed very easily), the dockerized app

can run without any problems. It will not care about hardware or software installed in that machine. Therefore, Docker is very important and we decided to dockerize our app to run in on an EC2 instance.

- **AWS:** Aws is a very efficient platform for deploying the various applications and many more. There are EC2 instances and we are free to select any type. For exceple, Ubuntu nano, Ubuntu small, Ubuntu Free tier , Windows .. etc. Ubuntu Free tier is free for 1 year monthly and 750 hours can be utilized without any price. Moreover, there is a lot of documentation showing how to use and deploy applications. Therefore we choose AWS EC2 instances. Security can be done by us which is very important. We installed docker and deployed our dockerized application to the ubuntu free tier machine.

5. Code and Deployment of Practice App

5.1) Back-end

We were not familiar with backend development. As a result, we didn't know which framework would be the best for our application. To find that framework, everyone made some research about backend frameworks such as Flask, Django, Spring Boot, Node.js and so on. But in the end, we decided that Django is the one we should use. Of course Suzan Hoca's advice on Django was a big factor for this decision.

We opened an issue to make sure everyone made some practice by creating dummy projects before making a meeting. After that, we had a group meeting and talked about the basics of Django and we made our first push on github. This first push was done by Bilal and that Django project had only a single app: Person. This Person app had some functionalities such as creating a random Person from a third party api, creating a person with given fields, getting first or last Person's fields as a json response, getting every Person's fields as a json response and getting a specific Person's fields by giving an id as input.

Now we got the basic concepts but we needed much more than that. Luckily, I had an assignment from my Cmpe 321 course that required a web application as well, so I learned how to create a basic login page and a homepage with almost zero functionality from that assignment. Thus, I created two more apps: login and post. Login app had only one job and that was making sure that everyone registered to the system can login and the rest can't. On the other hand, the post app had functionalities such as creating random or manual posts, viewing posts as a whole or only one of them. I also created some html files but there was no styling at all. I pushed this in our repository and shared it with my groupmates.

After this improvement, we were ready for the main part of our project. Berke and Cem created Community and Post Template features for our project which gave us lots of features such as creating a new community with title, description, image and privacy options, creating post templates to bring new post types (posts with images, videos or additional text fields) to the communities, json responses of communities, post templates, posts and so on. Also, they add some tweaks to the

post app as well. With these features, we were almost set up to finish our project but there was one problem: merge. Since they worked in separate branches, merging tons of different features was painful. There were some bugs in our backend code and some functions were not connected to html files. In a timely manner, we managed to finish all bugs in the back end and Muhammed added some new features to the community app such as join community and leave community.

With these features, our app was almost completed. We added their unit tests to test various functionalities of our project. Also, we made some third party api calls from websites to add some tweaks to the project and finally, we put some external api call functions that returned json responses accordingly and tested all of them with Postman. And that is all about the backend part of our project.

5.2) Front-end

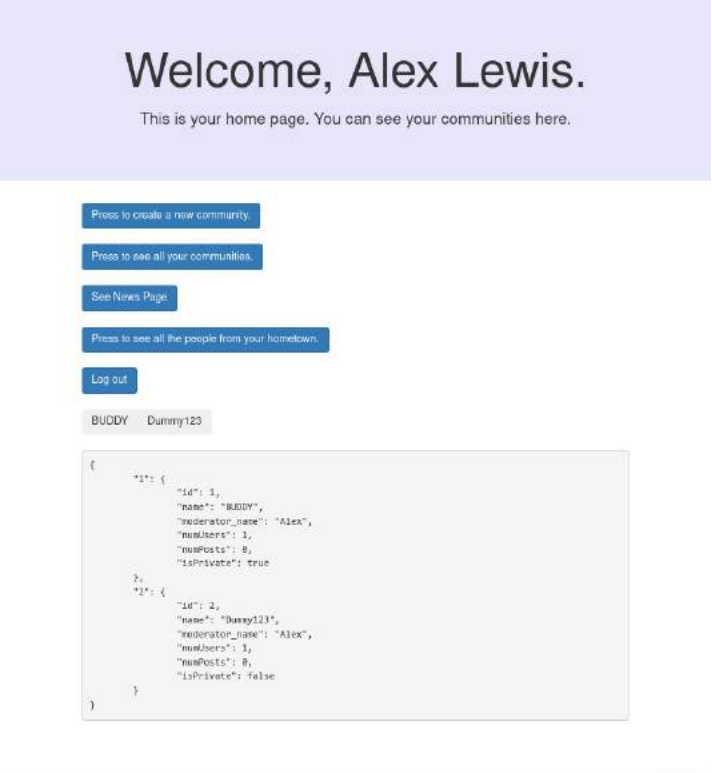
We were not familiar with frontend development. We decided that learning HTML, CSS and JS in detail would take too much time. Since it was stated that a simple frontend would be enough for the practice app, we focused on finding a platform agnostic library that facilitated swift and simple development. Due to its responsive grid system and easy integration capabilities, we decided to use bootstrap. Due to the grid system included in this library, the practice app works both on mobile applications and personal computers. The library was easily included using a content delivery network supplied by bootstrap in the HTML headers.

The ‘jumbotron’ class of bootstrap was used to create a lavender colored section at the top part of most pages, which displayed session-related data. For example, once a user named John Doe logs in, this section displays ‘Welcome John Doe’ in the home page. Once the user enters a community’s page, the title and description of said community is displayed.

The button styles were also imported from bootstrap, and classes such as ‘btn’, ‘btn-primary’, ‘btn-success’ and ‘btn-danger’ were used to apply css visuals to buttons. For example, the leave community button is red, while join community button is not.

A very simple navigation bar from this framework was also used to add a transparent navigation bar and a home button to the upper left edge of the pages. This home button redirects the user to his or her homepage for ease of use.

We decided to use simple JS instead of react.js or node.js since the practice app was a fairly simple app. Each page of the app was designed individually within the Django framework and placed in the ‘templates’ folder. In some pages, default button callbacks were prevented with JS to offer customized operability (such as the see all communities button, which spawns a list of communities of the user), while in other pages Django forms were used for simplicity. Javascript was also used to prettify and display the json responses received from several of our API’s functions, and an example where all communities of a user is returned can be seen below.



5.3) Dockerizing

We were not familiar with Docker and AWS. To handle these tasks, we researched Docker and AWS from many sources. Such as Youtube, Udemy, Google, Docker Documentation etc. It took nearly 3-4 days to have a good knowledge about these fields. To improve our skills in dockerization, we met on zoom, and dockerized an example django application to learn and see the process better. This meeting lasted about 2 hours but it was very efficient. Every team member now can dockerize an django application. In this process we took lots of notes and shared them via Discord. For example, if we are using ubuntu and we want to install docker, we should use these commands:

INSTALLATION OF DOCKER (Ubuntu)

****

```
sudo apt remove docker docker-engine docker.io containerd runc
sudo apt update
sudo apt install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
If Is is amd64 (dpkg --print-architecture):
echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io
****

We shared some very beneficial commands in Discord to use them in the dockerizing process. Here are some of them.

Docker without sudo
****
-Create the docker group.
sudo groupadd docker

-Add your user to the docker group.
sudo usermod -aG docker $USER

docker ps => See only working containers
docker ps --all => See all containers in computer

docker build -t 352 . => Build the app according to Dockerfile
docker run -it -p 8080:8080 352 => run it on pc

docker login => Login to docker to push and pull requests
docker images => See images of local computer
docker tag "image name" 4teko7/"tag name" => Tag the image to push to the Dockerhub
docker push "username"/352 => pushing to Dockerhub
docker pull 4teko7/352 => Pulling our app from Docherhub
docker run -it -d -p 8080:8080 352 => Run docker app, -d = background çalıştır,

5.4) AWS

AWS was a new thing for us, we didn't know what aws was. We heard some information about aws from the lecturer. Deploying an app to AWS was very complicated for us at that time. To overcome this complicated task, we searched

from many resources such as youtube, udemy etc. There were some decisions to take. For example, which operating system to use (Ubuntu, Windows etc.), Database (postgresql, mysql, sqlite). We decided to use Ubuntu because there are some friends in our team who have been using Ubuntu for a long time. Also, we decided to use sqlite, because its deployment is very easy in django and dockering it is not challenging. We buy a free tier ubuntu machine. We installed docker in it, but the applications didn't work. We searched to learn the problems, reasons. We realized that there should be some configurations about security of EC2 instances. To reach the instance from http, https, we should open 80, 443 ports. We run our application in 8080 ports, so we created a custom TCP port, and opened it to the public. By this process, our app worked as expected in http://54.164.7.166:8080/