
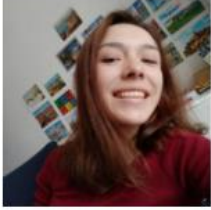

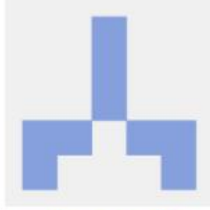
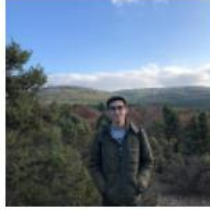
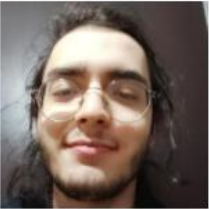






MILESTONE REPORT 2

13/06/2021

Group 2

 <p>Doğukan Akar</p>	 <p>Şefika Akman</p>	 <p>Ömer Arslan</p>	 <p>İbrahim Kağan Bayat</p>	 <p>Berkay Döner</p>
 <p>A. Necip Görgülü</p>	 <p>Ege Can Kaya</p>	 <p>Bengisu Özaydın</p>	 <p>Mehmet Saraçoğlu</p>	 <p>Kerem Zaman</p>

Contents:

Executive Summary	5
Brief Summary	5
How to run our Project	5
Basic Functionality of Our Project	5
Status of Deliverables	7
Project Plan	8
Summary of Work Done by Each Team Member	8
Evaluation of Tools and Processes	15
Google Drive	15
Discord	15
Whatsapp	15
Miro	16
Paint	16
GitHub	16
Project Libre	17
Visual Studio Code	17
Doodle	17
React.js	18
Flask	Error! Bookmark not defined.
Webstorm	19
Postman	19
Atom	19
Git	20
PyCharm	20
AWS	20
Docker	21
Insomnia	21
PostgreSQL	21
SQLAlchemy	22
Geocoding API of Google	22
Geolocation API of Google	22
RandomAPI	22
Google Search API	23
PIPL GETPERSON API (https://pipl.ir/v1/getPerson)	23

MetaWeather Weather States API	23
GeoNames API	23
Covid19 API	24
MapQuest API	24
CountriesNow API	24
Decathlon Sports API	24
API Documentation	24
Pages in Frontend	25
Events	25
Event Page	25
Create Event Page	26
Equipments	26
Equipment Page	26
Notification Page	26
Users Page	27
User	27
Get Users	27
Get User by ID	28
Post User	29
Event	31
Get Events	31
Get Event by ID	35
Post Event	36
Get Weather	38
Get Sport Types	39
Get Players	40
Spectator	41
Post Spectator	41
Get Spectator	42
Player	43
Apply as Player	43
Get Players by ID	44
Comment	45
Get Comments	45
Post Comment	46
Get Comment By ID	47

Answer	48
Get Answers	48
Post Answer	49
Equipment	50
Get Equipment By Id	50
Get Equipment	51
Equipment Search	52
Post Equipment	54
Follow	55
Get Followers	55
Follow User	56
Block	57
Post Block	57
Get Blocked Users	58
Notification	58
Post Notification	58
Get Notification	59

Executive Summary

Brief Summary

In this milestone report we are going to document how we have started implementing our project. You are going to find out how we have distributed the tasks among us, implemented API endpoints, created our database, created user interface, dockerized our project and deployed our docker to an Amazon server.

Overall Status of the Project

Overall we have completed our RESTful API and database. We also dockerized our project and deployed our docker using an Amazon server. Although we have completed our API, our frontend doesn't cover all our API functions. You can see the API functions accessible through our front end at the "Pages in Frontend" chapter in this report.

How to run our Project

You can run our project by entering the commands below to your machine's terminal:

```
git clone https://github.com/bounswe/2021SpringGroup2.git
cd 2021SpringGroup2/practice-app
sudo docker-compose build
sudo docker-compose up -d
```

Basic Functionality of Our Project

Through our user interface, users can:

- Post new events
- Search events
- Search for equipment by type, location
- Post new Equipment
- Apply as a player for an event
- Apply as a spectator for an event
- Get notifications
- Learn about the weather at a specific date
- See the players of an event
- See the spectators of an event
- Follow, block users
- List the users
- Comment under posts

- See the comments under posts

With the help of our RESTful API, one can:

- Retrieve all the users from the database - `/api/v1.0/users` GET
- Retrieve a user with a specific ID - `/api/v1.0/users/<string:userid>` GET
- Retrieve all the events in the database - `/api/v1.0/events` GET
- Retrieve an event with a specific ID - `/api/v1.0/events/<int:event_id>` GET
- Insert an event to the database - `/api/v1.0/events` POST
- Retrieve the weather at a specific date -
`/api/v1.0/weather/string:city/int:year/int:month/int:day` GET
- Add a spectator to the spectator list of an event -
`/api/v1.0/events/<int:event_id>/spectators` POST
- Retrieve spectators of an event -
`GET/api/v1.0/events/<int:event_id>/spectators`
- Add a randomly generated player to the players list of a specific event -
`POST /api/v1.0/events/<int:event_id>/players`
- Retrieve players of a specific event -
`/api/v1.0/events/<int:event_id>/players` GET
- Retrieve comments of a specific event -
`GET /api/v1.0/events/id:event_id/comments`
- Add a comment to a specific event -
`/api/v1.0/events/<int:event_id>/comments` POST
- Retrieve a comment from a specific event -
`/api/v1.0/events/<int:event_id>/comments/<int:comment_id>` GET
- Retrieve answers from a specific comment -
`/api/v1.0/<int:post_id>/comments/<int:comment_id>/answers` GET
- Add an answer under a specific comment -
`/api/v1.0/<int:post_id>/comments/<int:comment_id>/answers` POST
- Retrieve an equipment by Id - `/api/v1.0/equipment/<equipmentId>` GET
- Retrieve a randomly generated equipment - `GET /api/v1.0/equipments/`
- Search an equipment by equipment type - `/api/v1.0/search-equipment-type/<string:equipmentType>` GET
- Search an equipment by location - `/api/v1.0/search-equipment-location/<string:location>` GET
- Insert an equipment to the database - `/api/v1.0/equipments` POST
- Retrieve followers of a user - `/api/v1.0/users/<id:user_id>/followers` GET

- Add a follower to a specific user- /api/v1.0/users/<int:user_id>/followers POST
- Add a blocked user to the database - /api/v1.0/<id:user_id>/blocked-users POST
- Retrieve blocked users of a user - /api/v1.0/users/<int:user_id>/blocked-users GET
- Insert a notification to the database - /api/v1.0/notification POST
- Retrieve notifications - /api/v1.0/notification GET

Challenges We Met

1. We spent an entire week to find useful 3rd party API's to use in our functions. Since every one of us needed to find a unique API for themselves this part of our implementation took too much time.
2. Creating a roadmap for our project was probably the hardest obstacle to get over. Even though we managed to create a project plan from the first meeting after the assignment was given, new instructions given by teaching staff every week delayed our API Implementation part of our project. This ultimately put us into unbelievable time trouble which caused some parts of our project to be incomplete.
3. Distributing the tasks was a problem since not everyone had the same experience in the field, we could not really distribute the tasks equally. More experienced members of our team had to do more work in order to finish the project by deadline. Even with the heroic efforts shown by our experienced and hard working members, we could not complete the front end and we have some missing pages.
4. Most of us had to learn to work with new tools and maybe new languages which made the implementation process excruciating. Thanks to our experienced team members they were there whenever somebody needed help with their work.

Status of Deliverables

Deliverable	Status
App implementation	Completed
Dockerization	Completed
Deployment	Completed

Doğukan Akar	<ul style="list-style-type: none"> • Creating a react project with material-ui framework for the frontend of practice-app. Creating navigation using react-router-dom. • Creating front-end pages for searching and adding events. • Creating sqlalchemy classes for Comment and Answer tables on the database. • Creating GET and POST functions for Answer using a flask blueprint named answer_api. • Creating a GET function for spectators, it is getting the list of ids of spectators of an event from Eventpost table. • Creating a GET function for getting a list of sport types from decathlon api. • Creating a template for Milestone Report 2 using Google • Creating a frontend page for viewing a single event • Creating a frontend page for viewing a single equipment • Creating a frontend page for listing equipments • Creating a frontend page for listing users • Creating a frontend page for showing all notifications. • Reviewing some pull requests on github repo (They are listed on my individual report.)
Şefika Akman	<ul style="list-style-type: none"> • Created get_notifications function which gets notifications of a user by looking its user_id. • Created post_users function which posts a user and returns it. • Created get_players function which gets players of an event and returns them. • Created get_weather function which uses a third party API and returns selected weather conditions of a city which an event will happen. • Created pull requests for the functions that I have created. • Made API documentation for the API I have implemented. • Added frontend table explanations on the API Documentation page. • Made reviews of pull requests. • Contributed creating Milestone Report 2

Ömer Arslan	<ul style="list-style-type: none"> • Created Get Equipments function which uses a third party api to get a random equipment.(#60) • Created Search Equipments function that search the equipments from database according to their type.(#60) • Created another Search Equipments function that search the equipments from database according to their location.(#60) • Created Post Blocked Users function.(#66) • Created the “Blocking” database table.(#128) • Created a pull request for my branch (origin/post-blocked-users).(#111) • Reviewed pull request about getting blocked users and approved it. (#83) • Contributed to the Milestone 2 group report.
İbrahim Kağan Bayat	<ul style="list-style-type: none"> • First of all I connected my IDE with my github and I learned how to use git. I watched videos about git from the internet. • I was very new to web development, I don't event know the means of backend and frontend. So, I made some research about these topics. • We decided to use Flask in our API's, so I watched videos on internet about Flask and how to write an API with Flask . • I searched some free API's to use in our projects and they will be applicable to our functions. They should be related with our concept. • I am assigned to write postspectator function for our project. This function simply add a spectator to the spectator list of the event. It takes a user parameter as body and event parameter as path, and adds this user to the spectator list of the event. If user doesn't exist, function creates a user with the third party api: https://pipl.ir/v1/getPerson and adds this user with this user id given as body parameter. • We used SQLAlchemy in our project. So, I had to make some research about it. Also, I made some research about docker and how to use it. Working with database system and running our docker container in my computer was very hard for me, they takes many hours. • Later I adjusted my post function to our database. Post

	<p>spectator function is in the spectator.py file. Interacting with database was very hard for me, so while writing my functions I got help from my friends Berkay, Ege ,Doğukan and Kerem.</p> <ul style="list-style-type: none"> • Later we learned that everybody should write at least a post and a get method. Then we added more functionality to our project and I am assigned to write getcomments function. Function simply gets all the comments with given event id. You can find getComment method in comment.py file. • It was very hard for me to write unit test for my API, and I didn't find any function to write unit test without API in my project part. So, I write a simple function to apply unit test on that for making practice about unit tests. I created a simple function and I wrote unit tests for that function. You can find my unit tests and simple function in test.py in test directory. TestPlayers class is written by me in test.py file. • I opened three pull requests seperately for my postspectator, getcomments and my unit test functions(#82, #86, #103). Before doing that I made some research on the internet about pull requests.I adjusted my functions according to reviews I got, until every reviewers approve my pull request. • I reviewed my friends pull requests and I gave feedback to their pull requests.(#71, #76, #79, #87, #89, #90, #114). I made a short research about the reviewing pull requests on the internet. • I filled the documentation page for API's written by me. You can see it in the wiki page in our github repository. • I opened three issues for postspectator, getcomments and unittests written by me. (#53, #81, #112) • Besides attending many group meetings, I was always in Discord and interacting my friends about how to write my functions and I asked questions them about my challenges (For example writing my functions, running docker and our database tables etc.).
Berkay Döner	<ul style="list-style-type: none"> • Studied Flask and basic Rest API concepts before writing my own functions. • Searched external API's with some other team members in a meeting. • Created Get Events function that filters events with respect to various criteria, including a location criterion that ensures only nearby events are returned,

	<p>which uses Geocoding and Geolocation API of Google.</p> <ul style="list-style-type: none"> • Created Post Comment function that creates a comment linked to an event after checking some conditions. • Created Get Comment By ID function that takes event ID and comment ID to return the corresponding comment. • Separated the functions to various files with respect to their functionalities, with Doğukan. • Helped with the front end part after I noticed that the frontend part was not complete. However, since I hadn't known React before, I could only provide some limited help for Create Event Page and Create Equipment Page. • Created the Post and Eventpost tables for the database. Post table is an abstract parent table that is inherited by both Eventpost and Equipmentpost tables. • Created unit tests for the get event, post comment and get comment by ID functionalities. Also helped other team members to write their unit tests. • Wrote API documentation about the functions I implemented, get events, post comment, get comment by ID. • Created #63, #79, #93, #97, #105, #106, #113 pull requests for the functions I implemented. • Reviewed #61, #62, #64, #80, #90, #109, #114 pull requests that are mostly about GET functionalities. • Contributed to the Milestone 2 report.
Ahmet Necip Görgülü	<ul style="list-style-type: none"> • Took the meeting notes for the duration of the project. • Researched 3rd party API's after a meeting with Doğukan Akar, Ege Can Kaya, Kerem Zaman, Berkay Döner, Mehmet Saraçoğlu and İbrahim Kağan Bayat. • Found a Google Search API and used it to Get Equipment. • Implemented Get Equipment function for our RESTful API. • Implemented Post Notification function for our RESTful API. • Created the "Notification" table for our database. • Wrote unit tests for the helper function 'results' which I used in Get Equipment function. • Wrote the API documentation for my endpoints • Created pull requests with #65, #80, #88, #118

	<ul style="list-style-type: none"> • Reviewed pull requests #63, #89, #106, #111, #113, #115 • Wrote the Brief Summary of Milestone Report 2 • Wrote the 'Basic Functionality of Our Project' part of Milestone Report 2. • Wrote 'Challenges we met' part of Milestone Report 2 • Delivered our practice-app submission file to moodle.
Ege Can Kaya	<ul style="list-style-type: none"> • Created apply as player function (HTTP POST) method which lets a player apply to an event. This function also uses a 3rd party API, randomapi, where I wrote a custom API which returns the appropriate data to generate a new user in the case where an applying user does not exist. • Created the Get Followers function (HTTP GET) which gets the followers of a specified user. • Created the User table for the database, which holds the data for the users in the system. • Created pull requests #95 and #76 • Reviewed #79, #85, #109, #119 • Wrote API documentation for the functions I implemented • Resolved some conflicts in merges • Wrote the unit tests in test_apply.py • Contributed to the Milestone 2 group report
Bengisu Özaydın	<ul style="list-style-type: none"> • Created Post Event API. (Issue: #59, PR: #107) • Created Get Blocked Users API. (Issue: #83, PR: #84, #96) • Created unit tests for Post Event API. (Issue: #125, PR: #120) • Created unit tests for Get Blocked Users API. (Issue: #123, PR: #126) • Wrote documentation for the above APIs. #122 • Implemented frontend for equipment post page and equipments page (where equipments are shown in lists), by reference to the corresponding event pages. #49 • Reviewed pull request for equipment post class in the database. #71 • Reviewed pull request for post equipment method connection with database. #89 • Reviewed pull request for post blocked users, get equipments, filter equipments functions and addition of blocking table to the database. #111 • Reviewed pull request for post equipment unit tests.

	#115
Mehmet Saraçoğlu	<ul style="list-style-type: none"> • Created Post Equipment API. Issue: #54 PR: #71 • Created Get User API. Issue: #108 PR: #90 • Created Unit tests for Get User and Post Equipment APIs. Issue: #116, PR: #114, #115 • Created EquipmentPost database class. Issue: #58 • Wrote documentation about these APIs. Issue: #77 • Reviewed pull requests of my teammates. PR: #118, #88, #86, #82, #80, #107, #111 • Expanded the project-plan. Issue: #127
Kerem Zaman	<ul style="list-style-type: none"> • Created API endpoints for getting event by id endpoint using Covid19 and GeoNames APIs to serve the coronavirus information in the location of user and wrote unit tests for this API endpoint • Created general project folder structure with baseline codes for both API and tests • Created API endpoint for following users and wrote tests for it • Created Following table in database • Created and managed Docker-related files to dockerize application and database • Splitted main code into multiple files and made connection among them using Blueprint structure • Created baseline unit test code for database test to be used by other team members • Fixed various tiny bugs appeared after merges into development • Reviewed #82, #86, #95, #96, #103, #104, #105 • Created PRs #85, #109, #119 • Wrote documentations of the API endpoints that I wrote • Deployed application on AWS along with NGINX installation • Contributed to Milestone 2 Group report and wrote Milestone 2 Individual report

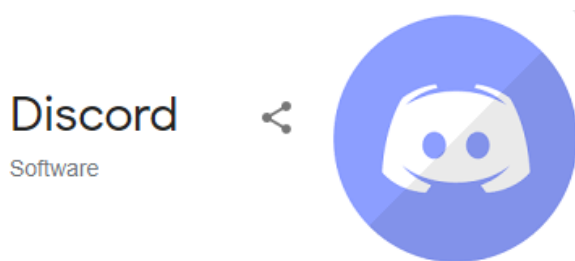
Evaluation of Tools and Processes

Google Drive



Google Drive is a file storage service. It has the google docs app to create and edit documents and spreadsheets. We created docs and spreadsheets for deliverables and milestone report. Drive helps us to be synchronized while working simultaneously.

Discord



We use discord for several purposes. Our main communication is on discord. We use discord to both chat and do online meetings. Also we use it for task distribution. We list the tasks and each member reacts to the task s/he will take. We created several chat channels to collect related texts on the same page like user requirements, system requirements. Also we have several voice channels for subgroups of works.

Whatsapp



Whatsapp is the easier way of communication. We just use it to reach other members quickly. Usually we don't take important decisions on whatsapp instead we use discord.

Miro



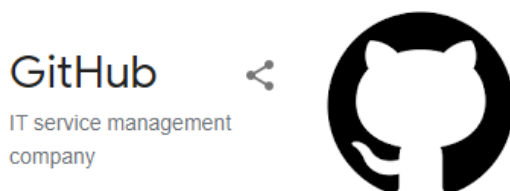
Miro is an app that allows all team members to work on the same board simultaneously. We have a board that we create most of the deliverables for example sequence diagrams, mockups and class diagram. It helped us on the team meeting because we can view and change diagrams simultaneously on the board. We could export sequence diagrams and mockups as an image. While exporting use case and class diagrams we encountered image resolution limit problem. So we should have used another program for this situation.

Paint



We used paint to merge small screenshots of the diagrams that we couldn't export as images from Miro.

GitHub



Everything we do is collected in github.

In our wiki page, we open separate pages for all works and upload what we have done. It is easy to update pages also it is easy to keep track who has made changes.

Another nice feature of github is that we can create an issue and make progress through issues. For example if we create an issue, we can add labels according to its importance, type, status etc., we can update and when it is done we can close it.

Project Libre



Project Libre is a project management software. We used it to create the project plan. It does not allow us to create online plans so we had to merge parts of the plan manually. Also, one of the team members observed that the program stops responding occasionally, which caused data loss during work. While exporting the project plan, it didn't include non ASCII characters, also it doesn't have export as an image feature. This made the process hard for us.

Visual Studio Code



Since the writing area in github is narrower, we sometimes prefer to write markdown text in visual studio code. You can preview the markdown while writing on VS Code.

Doodle



Doodle is a tool for scheduling meetings in which we can create polls. We used it to schedule customer and team meetings.

React.js



React is a javascript framework for web apps. We used react with Material-ui toolkit. It has good components for frontend. React was a good choice for visuality but most of the team didn't know how to use it and we had difficulties on the frontend. We should do a better organization while we develop using react.

Flask



We used Flask in our project app, which is a Python framework for web development. We used Flask since it is very easy to learn and provides various functionalities. We separated our functions into many files with respect to their functionalities using Blueprints and provided get and post methods in each file.

Flask provides the necessary functionalities for now; however, we might want to choose more complex frameworks like Django in CMPE451, since we will implement much more complex methods.

Webstorm



Webstorm is an IDE by JetBrains for developing web applications. We used it while we were building the frontend app. Webstorm has good tools for node.js. It also has a good version control system. It makes the process simple to checkout branches.

Postman



Postman is a tool to test API methods with an user-friendly interface. It can be used to send both GET and POST requests to some URI's with different types of parameters. We used Postman to test both our own API methods and external API's.

Atom



Atom is a free, open-source text editor for macOS, Linux and Microsoft Windows. Some of our team members used it in the development of our application,

Git



Git is version control system that is currently the industry standard in software development. It allows users to track and view changes made in files along with their previous versions. It is essential for coordinating work among programmers collaboratively writing code during the development of software. We used Git during every step of our development in conjunction with GitHub to keep track of our progress and to be able to view each other's codes.

PyCharm



PyCharm is an IDE by JetBrains which allows convenient tools for Python code editing. Some of our team members used it in the development and test runs of our practice application. It provides many convenience features such as syntax showing, error correction, an integrated VCS that can be used in conjunction with GitHub, a convenient GUI and an integrated terminal.

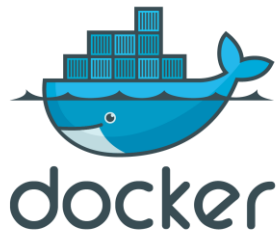
AWS



Amazon Web Servers (AWS) is a platform providing a collection of cloud computing services and APIs. We used AWS EC2 instances to deploy our application. We selected T2.micro instance type because of the 750 hours monthly free usage. It has 1 vCPU and 1 GB RAM which are enough for our application since our user base is very small and there is no high computational requirements. AWS provides an easy-to-use user interface that allows you to create an instance in minutes. It's also very easy to define network rules. Despite all the great

features, one should be very careful while using extra AWS features to prevent unexpected pricing.

Docker



Docker is a great tool for managing different services in an efficient and modular way. We dockerized our API and frontend and used a ready-to-use PostgreSQL docker image for the database. Using the PostgreSQL image was very convenient to configure. Also, we used docker-compose to manage our different services which are two different containers. It would be very hard to deploy our application without docker. Docker saves us to manage each dependency, running our application, and configuring the database each time.

Insomnia



Insomnia is a tool much like Postman to make interacting with and designing HTTP-based APIs convenient. It arguably provides users with a cleaner and simpler interface than Postman on mac OS. Some of our team members used it as an alternative to Postman.

PostgreSQL



We used PostgreSQL server to build our database for the practice app, following the advice of our instructor. It is very similar to SQL so we did not have many problems using it. PostgreSQL provided the functionalities we needed for now, but we might want to change our database server to a No-SQL server, which is more flexible. PostgreSQL also comes with a convenient GUI tool called pgAdmin to create databases and check their content, which came in handy during development.

SQLAlchemy



SQLAlchemy is an open-source Python module which serves as a toolkit for using database services and executing SQL queries using the Python language. We used SQLAlchemy ORM's convenient syntax by setting the base as the declarative base, which allows one to write very object-oriented-programming-like statements using classes and Python methods instead of pure SQL strings.

External APIs

Geocoding API of Google

Geocoding API of Google takes an address from a get request and returns the latitude and longitude values for this address in addition to some information about the location such as formatted address and geometry information. It is used to find the coordinates of the given address string for event search, those coordinates are then used to find distances between events and the address using Haversine distance.

Geolocation API of Google

Geolocation API of Google is used for the same purpose. It returns the latitude and longitude values that are then used as parameters in the Haversine distance formula. However, this API calculates coordinates using the IP address of the request sender not using an address. It also is accessed through post request, not a get request.

RandomAPI

This API provides the user with some useful modules like Faker, Deity and Moment which help in creating randomly generated data. Actually, RandomAPI does not provide the user with a ready-made API to use. Instead, it asks the user to write their own API using JavaScript and the provided modules. This allows the user to create exactly the random fields they need for usage in their application. RandomAPI then runs this user-made API permanently on their server, which the user can access anytime they want. However, a free user can only send 500 requests per day and a payment must be made for infinite usage.

Google Search API

This API provides different functions namely GET News, GET Crawl, GET Search, GET Images and GET Scholar. I decided to use GET Search functionality of this API. This function allowed me to retrieve search results relevant to the equipment we retrieve in GET Equipment by Id function.

PIPL GETPERSON API (<https://pipl.ir/v1/getPerson>)

This api generates a random person with many parameters including personal, educational information and working, marriage status. We used this third party API in postspectator function in spectator.py file. If there is not any user in our database with given user id, functions take data from here and creates a user. Function also adds user to the spectator list of the given event.

MetaWeather Weather States API

MetaWeather provides an API which can give us information about the weather by making location search. We can take weather conditions of a city for a date which is formed year/month/day. By using this API, someone can make decision about an event.

GeoNames API

This API gives comprehensive information about similar locations to the searched one. Its search results return administrative code, latitude, longitude, administrative name, population, country code, country name and many other information about the locations that are contained by the given location or similar to the given location. We used country name information to extract the country of a user when its location is known since users does not have to specify their country when entering location information. The user's country information is used for fetching coronavirus stats.

Covid19 API

The Covid19 API gives daily coronavirus stats by country along with confirmation status of cases, date, country code, latitude and longitude. We used this API to serve coronavirus risk status while showing events. We get daily cases in the country of the event owner from the day the event is showed to user to the 3 days before. If there is a constant increase in the number of daily cases for the period of 3 days, we decided that there is a high risk. Also, we show current daily cases as well along with the event details.

MapQuest API

MapQuest API is a free API that provides maps operations such as geocoding (converting a string of address into a tuple of coordinates) and reverse-geocoding (converting a tuple of coordinates into a string of address). In our project, the geocoding API is used when posting an event. The location of the event is converted into the latitude and longitude values associated with that location. The latitude and longitude values of a posted event are used to filter nearby events, where the area being searched is limited to a circular area of radius of user's choice.

CountriesNow API

CountriesNow API is a completely free API which can be used to derive the country names from the given city name. By request command, it returns a dictionary that includes all the keys and values. It requires a little modification and turning it into the json type. It works only via city names.

Decathlon Sports API

Decathlon provides an API for listing and searching different sports. It provides some information about sports such as their name, description, and icon. It gives information about other similar sports and which sport is popular in some areas. We used the API for getting sport names, descriptions and icons in our function "get sport types".

API Documentation

This page contains list of pages and API documentation of our practice-app in CMPE352. We built an API named practice-app and we build also a web application that uses functions of the API. Both frontend and the RESTful API are served on the Amazon Web Services.

You can check our API documentation at our [Wiki](#).

It can be accessed on <http://3.19.67.188>.

Path for the API is <http://3.19.67.188/api/v1.0/>

Pages in Frontend

Events

/events

Definition: This page is for listing event posts. it has a table that each row represents an event.

API Endpoints

- /api/v1.0/events GET

Event Page

/event/:id

Definition: This page is for viewing a single event post. It has attributes of an event post such as location, date, covid risk status of that location, weather conditions... It lists users that are players or spectators of the event. It has also a discussion section that shows comments of the post and their answers.

API Endpoints

- /api/v1.0/events/<int:event_id> GET
- /api/v1.0/events/<int:event_id>/players GET

- /api/v1.0/events/<int:event_id>/spectators GET
- /api/v1.0/events/<int:event_id>/comments GET
- /api/v1.0/events/<int:event_id>/comments/<int:comment_id>/answers GET
- /api/v1.0/weather/<string:city>/<int:year>/<int:month>/<int:day> GET
- /api/v1.0/users/<int:user_id> GET

Create Event Page

/create-event

Definition: This page is for creating an event post. It has a form that takes necessary input and makes a post request for creating an event.

API Endpoints

- /api/v1.0/events POST

Equipments

/equipments

Definition: This page is for searching and listing equipment posts. It shows a list of equipment posts as a table.

API Endpoints

- /api/v1.0/search-equipment-type/<string:equipmentType> GET

Equipment Page

/equipment/:id

Definition: This page is for viewing a single equipment post. Page has attributes of a post and its creator.

API Endpoints

- /api/v1.0/equipments/<int:equipment_id> GET
- /api/v1.0/users/<int:user_id> GET

Notification Page

/notifications

Definition: This page is for viewing all notifications.

API Endpoints

- /api/v1.0/notifications GET

Users Page

/users

Definition: This page is for listing users. Users are viewed with their nickname and avatar.

API Endpoints

- /api/v1.0/users GET
- /api/v1.0/users/<int:userid> GET

User

Get Users

GET '/api/v1.0/users/'

- **Author:** [Mehmet Saraçoğlu](#)
- **Definition:** It returns all the users. Searches through the database and returns all the answers.
- **Example Response Value:**

```
[
  {
    "user_id" : 92581736,
    "nickname" = "oduncubaba",
    "first_name" = "keramettin",
    "last_name" = "tosuncuklar",
    "biography" = "Born, living. Not died yet",
    "birth_year" = "1923",
    "avatar" = "image.png",
    "location" = "cave",
    "fav_sport_1" = "football",
```

```
"fav_sport_2" = "soccer",
"fav_sport_3" = "ayaktopu",
"badge_1" = "Woodie",
"badge_2" = "best player eu",
"badge_3" = "oldest guy in eu",
"privacy" = Yes,
"country" = "not known"
}
]
```

- Response Messages
 - 200: Search successful.
 - 404: Not found.

Get User by ID

GET '/api/v1.0/users/<int:userid>'

- **Author:** [Mehmet Saraçoğlu](#)
- **Definition:** Returns the user according to the given user ID.
- **Example Response Value:**

```
{
  "user_id" : 92581736
  "nickname" = "oduncubaba"
  "first_name" = "keramettin"
  "last_name" = "tosuncuklar"
  "biography" = "Born, living. Not died yet"
  "birth_year" = "1923"
  "avatar" = "image.png"
  "location" = "cave"
  "fav_sport_1" = "football"
  "fav_sport_2" = "soccer"
  "fav_sport_3" = "ayaktopu"
  "badge_1" = "Woodie"
  "badge_2" = "best player eu"
  "badge_3" = "oldest guy in eu"
```

```
"privacy" = Yes
"country" = "not known"
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
nickname	Nickname of a user	Path	String

- **Response Messages**
 - 200: Search successful.
 - 404: Not found.

Post User

'POST /api/v1.0/users'

- **Author:** [Sefika Akman](#)
- **Definition:** This method is for adding a user. It returns the user.
- **Example Response Value:**

```
{
  "user_id" : 43525335646,
  "nickname" = "headscef",
  "first_name" = "cansu",
  "last_name" = "suyol",
  "biography" = "writer",
  "birth_year" = 1987,
  "avatar" = "soul.png",
  "location" = "Bursa",
  "fav_sport_1" = "football",
  "fav_sport_2" = "soccer",
  "fav_sport_3" = "basketball",
  "badge_1" = "",
  "badge_2" = "",
  "badge_3" = "",
  "privacy" = false
}
```

}

- Parameters

Parameter	Description	Parameter Type	Data Type
user_id	id of a user	Body	Biginteger
nickname	nickname of a user	Body	String
first_name	first name of a user	Body	String
last_name	last name of a user	Body	String
biography	biography of a user	Body	String
birth_year	birth year of a user	Body	Integer
avatar	avatar of a user	Body	String
location	location of a user	Body	String
fav_sport_1	favorite sport 1 of a user	Body	String
fav_sport_2	favorite sport 2 of a user	Body	String

fav_sport_3	favorite sport 3 of a user	Body	String
badge_1	one of the badges of a user	Body	String
badge_2	one of the badges of a user	Body	String
badge_3	one of the badges of a user	Body	String
privacy	privacy of a user	Body	Boolean

- **Response Messages**
 - 201: Successful.
 - 400: Not found.

Event

Get Events

GET /api/v1.0/events

- **Author:** [Berkay Döner](#)
- **Definition:** This method implements a search mechanism for events. It takes several filter parameters and returns the list of events that satisfy these filters. All filters are optional except the location filter, which is used to filter the events that are in the given radius of the given location. Location can be either given as an address string, which is converted into coordinates using the Geocoding API of Google or it can be calculated from the IP address of the user, which is converted to coordinates using Geolocation API of Google
- **Example GET Request**

/api/v1.0/events?ip=false&radius=300&empty=true&search=Everyone&ageGroup=20,30&dateBegin=06/06/2021 13:00:00&dateEnd=20/06/2021 15:00:00&skillLevel=Beginner&address=Boğaziçi Üniversitesi&orderby=title&order=desc&sport=Football

- **Example Response Value**

```
[
  {
    "content": "People who are looking for a tennis court can apply this event. Our court is for you if you are seeking for a clean and well-lit court.",
    "creationDate": "2021-06-05 13:02:37",
    "eventAgeGroup": "[20, 30]",
    "eventDate": "2021-06-05",
    "eventHours": "13:00:00",
    "eventLatitude": "41.0869173",
    "eventLongitude": "29.0321301",
    "eventPlayerCapacity": "3",
    "eventPlayers": "[1, 2]",
    "eventSkillLevel": "Beginner",
    "eventSpectatorCapacity": "10",
    "eventSpectators": "[1, 2, 3]",
    "eventSport": "Tennis",
    "location": "Etiler Tennis Club",
    "ownerID": "1",
    "postID": "1",
    "title": "Tennis Game for Everyone"
  }
]
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
ip	Whether or not to use the IP address of the user to calculate coordinates. Can be either "true" or "false".	Query	String

address	Address to find events nearby. Not necessary if the ip is "true".	Query	String
radius	Radius to find nearby events, unit is kilometers. Must be given.	Query	Integer
empty	Whether or not to show the full events. Full events are not displayed if empty is "true", are displayed if "false"	Query	String
sport	Sport type of searched events. Only the events with the given sport type are displayed.	Query	String
skillLevel	Skill level of the searched events. It is an enumeration of "Beginner", "Preintermediate", "Intermediate", "Advanced" and "Expert". Other values are not accepted.	Query	String
search	Keyword to be searched in the title and description of the events. Only those events that contain the keyword are displayed.	Query	String
ageGroup	A tuple containing the lower and upper bounds of the age group of events. Both bounds are inclusive. Tuple is given without parentheses, e.g. 20,30	Query	Integer Tuple

dateBegin	A datetime object with format "%d/%m/%Y %H:%M:%S". Must be given together with dateEnd to filter the events that will take place between two dates. Events are filtered with respect to date and time separately. For example, when 06/06/2021 13:00:00 and 20/06/2021 15:00:00 are given as dateBegin and dateEnd, filtered events will take place in between 13:00:00 and 15:00:00, 07/06/2021 12:00:00 would not be returned for this query, even though its date is between the given dates.	Query	Datetime
dateEnd	A datetime object with format "%d/%m/%Y %H:%M:%S". Must be given together with dateBegin.	Query	Datetime
orderby	Column name to sort the events. title, content, eventSport, location, eventSkillLevel, eventDate can be used for sorting.	Query	String
order	Direction of the sorting. Must be "asc" or "desc". "asc" results in ascending order and "desc" results in descending order.	Query	String

Since all parameters are query parameters, they are converted to string when request is sent.

- **Response Messages**

- 200: Search is successful, list of json objects are returned as above.
- 404: Address could not be converted into coordinates.

```
{'error': 'Address was not found.'}
```

- 400: Search parameters are incorrect, due to either missing parameters or type mismatch. Ex:

```
{'error': 'Date and time must be in format DD/MM/YYYY HH:MM:SS'}
```

Get Event by ID

GET /api/v1.0/events/<int:event_id

- **Author:** [Kerem Zaman](#)
- **Definition:** It returns all the event information, current number of daily coronavirus cases in the country of event owner and additional warning remarking risk status according to increase in number of daily cases for 3 days in a row.

- **Example Response Value**

```
{
  "covid_risk_status":false,
  "current_cases":188762,
  "event":{
    "content":"Very funny event",
    "creationDate":"2021-06-13 08:54:32.050785",
    "eventAgeGroup":"[5, 10)",
    "eventDate":"2021-06-13",
    "eventHours":"13:00:00",
    "eventLatitude":"39.95623",
    "eventLongitude":"32.66393",
    "eventPlayerCapacity":"2",
    "eventPlayers":"[]",
    "eventSkillLevel":"Beginner",
    "eventSpectatorCapacity":"5",
    "eventSpectators":"[]",
    "eventSport":"Tennis",
    "location":"Etiler",
    "ownerID":"55",
    "postId":"1",
    "title":"Tennis Game for Everyone"
  }
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	ID of the event of which the details requested	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Requested event not found

Post Event

POST /api/v1.0/events

- **Author:** [Bengisu Özaydın](#)
- **Definition:** It posts an event with necessary information: title, date, time, type of sport and player capacity, and with optional information: content, age group, spectator capacity, players, spectators, skill level and address as coordinates.
- **Example Response Value**

```
{
  "event":{
    "ownerID":"1",
    "postID":"1",
    "content":"Kickbox sparring training",
    "title":"Group kickbox training",
    "creationDate":"2021-05-06 00:00:00",
    "location":"Hisarustu",
    "eventDate":"2022-01-01",
    "eventHours":"02:00:00",
    "eventSport":"Kickbox",
    "eventAgeGroup":"None",
    "eventPlayerCapacity":"10",
    "eventSpectatorCapacity":"5",
    "eventPlayers":"None",
    "eventSpectators":"None",
    "eventSkillLevel":"Intremediate",
  }
}
```

- Parameters

Parameter	Description	Parameter Type	Data Type
ownerID	ID of the owner of event	Body	Integer
content	ID of the owner of event	Body	String
title	Title of event	Body	String
creationDate	Creation date of event post	Body	DateTime
location	Location of event	Body	String
eventDate	Date of event	Body	Date
eventHours	How many hours the event will take	Body	Time
eventSport	The sport that event is associated with	Body	String
eventAgeGroup	Target age group of event	Body	Integer Tuple

eventPlayerCapacity	Player capacity of event	Body	Integer
eventSpectatorCapacity	Spectator capacity of event	Body	Integer
eventPlayers	List of player users	Body	Array of Integers
eventSpectators	List of spectator users	Body	Array of Integers
eventSkillLevel	Target skill level of event	Body	Enum
eventLatitude	Latitude of event location coordinates	Body	Float
eventLongitude	Longitude of event location coordinates	Body	Float

- **Response Messages**

- 201: Post successful.
- 400: Event details not sufficient for proceeding.

Get Weather

'GET /api/v1.0/weather/string:city/int:year/int:month/int:day'

- **Author:** [Şefika Akman](#)

- **Definition:** It gets data from a third party API by looking city, year, month and day and selects some elements of this data.
- **Example Response Value**

```
{
  "humidity": "",
  "max_temp": 36, "min_temp": 20,
  "the_temp": 25,
  "weather_state_name": "",
  "wind_speed": 5
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
city	city that the weather condition asked	Path	String
year	year of date	Path	Integer
month	month of date	Path	Integer
day	day of date	Path	Integer

- **Response Messages**
 - 200: Success
 - 500: Internal server error

Get Sport Types

GET /api/v1.0/sportTypes

- **Author:** [Doğukan Akar](#)

- **Definition:** It returns a list of sport types with their description, name and icon. It would help user when creating a post. It uses Decathlon's REST API.

- **Example Response Value**

```
[
  {
    description: "Recreational and touring activity, but also a very... Will you have enough power for the final sprint?",
    icon: "https://sports-api-production.s3.amazonaws.com/uploads/sport/icon/282/282.svg",
    name: "Road cycling"
  }
]
```

- **Response Messages**

- 200: Successful

Get Players

GET /api/v1.0/events/<int:event_id>/players

- **Author:** [Şefika Akman](#)
- **Definition:** It returns players of an event.
- **Example Response Value**

```
[
  {
    "event_id": 52,
    "players": ""
  }
]
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	id of an event	Path	Integer
players	players of an event	Body	List

- Response Messages
 - 200: Successful
 - 404: Not found

Spectator

Post Spectator

POST /api/v1.0/events/<int:event_id>/spectators

- **Author:** [İbrahim Kağan Bayat](#)
- **Definition:** This method takes a parameter event id for a user to apply to this event, also this method takes another parameter as body which is the user id of the spectator. Method checks if there is an event exist with the given parameter. If there is no event with given parameter it returns an error. Functions checks if there is a user with given user id. If user exists, function adds it to the spectator list of the event. If user doesn't exists, a random user will be generated with a third party api and added to both user table and spectator list of the given event.
- **Example Response Value**

```
{
  "eventId": 48,
  "eventTitle": "Tennis Game for Everyone",
  "applicantId": 12,
  "applicantNickname": "Florina_Marsh",
  "applicationServerTime": "08/06/2021 00:39:12"
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer
user_id	Id of an applying user	Body	Integer

- **Response Messages**
 - 201: Created
 - 404: Not found

Get Spectator

GET /api/v1.0/events/<int:event_id>/spectators

- **Author:** [Doğukan Akar](#)
- **Definition:** It gets the id of an event as request parameter and returns a list of ids of spectators of the event.
- **Example Response Value**

```
{
  "event_id": 4520,
  "spectators": [
    21385,
    12544
  ]
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Not found

Player

Apply as Player

POST /api/v1.0/events/<int:event_id>/players

- **Author:** [Ege Can Kaya](#)
- **Definition:** It gets the ID of an event as request parameter and the ID of an applying user from the request body and adds that user to the list of players for that event. In the case of a non-existing user, a new randomly-generated user is created using a 3rd party API, randomAPI. This randomly-generated user is added to the database table for users and then added to the list of players for the event.
- **Example Response Value**

```
{
  "eventId": 1,
  "eventTitle": "Tennis Game for Everyone",
  "applicantId": 128,
  "applicantNickname": "ila_karabulut54",
  "applicationServerTime": "07/06/2021 16:57:25"
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer
user_id	Id of an applying user	Body	Integer

- **Response Messages**
 - 201: Created
 - 404: Not found

Get Players by ID

GET /api/v1.0/events/<int:event_id>/players

- **Author:** [Şefika Akman](#)
- **Definition:** This function gets an ID of event as request parameter and gets players of that event.
- **Example Response Value**

```
{
  "user_id" = 423452658562341
  "nickname" = "salim"
  "first_name" = "salih"
  "last_name" = "akyel"
  "biography" = "from Izmir"
  "birth_year" = 1977
  "avatar" = "yellow.png"
  "location" = "Istanbul"
  "fav_sport_1" = "football"
  "fav_sport_2" = "volleyball"
  "fav_sport_3" = "socket"
  "badge_1" = ""
}
```

```
"badge_2" = ""  
"badge_3" = ""  
"privacy" = true  
}
```

- Parameters

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer

- Response Messages
 - 404: Not found

Comment

Get Comments

`GET /api/v1.0/events/id:event_id/comments

- Author: [İbrahim Kağan Bayat](#)
- **Definition:** This method takes a parameter post id and looks to the comment table in the database if there is a comment with the given ID. If there is no comment with the given ID it returns error 404 with an error message **There is no comment with this postid**. If there is a comment with given ID it returns success code with the content of the comment
- **Example Response Value**

```
{  
"comment": "Thisisacomment"  
}
```

- Parameters

Parameter	Description	Parameter Type	Data Type
event_id	Id of the event	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Not found

Post Comment

POST /api/v1.0/events/<int:event_id>/comments

- **Author:** [Berkay Döner](#)
- **Definition:** This method takes an event_id as a path parameter, user_id and comment as post parameters and adds the given comment to the database. First, event_id and user_id are checked to ensure that there is a post and a user with given id's. If this check is satisfied, the comment is added to the database and it is returned as a json object.
- **Example Response Value**

```
{
  "comment": "Was a very good match, thanks!",
  "commentDate": "10/06/2021 17:44:28",
  "ownerID": "1",
  "postID": 1
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer

user_id	ID of a user	Body	Integer
comment	Comment String	Body	String

- **Response Messages**

- 201: If the comment was posted successfully, the json object is returned as above.
- 404: If the either event_id or user_id is invalid. Ex:

```
{'error': 'There is no post with given ID'}
```

Get Comment By ID

GET /api/v1.0/events/<int:event_id>/comments/<int:comment_id>

- **Author:** [Berkay Döner](#)
- **Definition:** This method takes event_id and comment_id parameters from the path, and returns the comment associated with these ID's. It applies three checks for the parameters, it checks if the event_id and comment_id are valid ID's associated with some event and comment. Also, it checks if the comment is a comment of the given event. If so, the comment is returned as a json object.
- **Example GET Request**

api/v1.0/events/1/comments/2

- **Example Response Value**

```
{
  "comment": "Was a nice match.",
  "commentDate": "2021-09-06 22:46:46",
  "commentID": "2",
  "ownerID": "1",
  "postID": "1"
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
event_id	Id of an event post	Path	Integer
comment_id	Id of the comment of post	Path	Integer

- **Response Messages**
 - 200: If the comment was fetched successfully, the json object is returned as above.
 - 404: If event_id or comment_id is invalid, or the comment is not a comment of the given event. Ex:

```
{'error': 'There is no comment with given ID'}
```

Answer

Get Answers

GET /api/v1.0/<int:post_id>/comments/<int:comment_id>/answers

- **Author:** [Doğukan Akar](#)
- **Definition:** It gets a post id and a comment id as path parameters. Firstly it checks if post and comment with given ids are existing in the database, aborts 404 error if they are not. It returns a json that includes comment id and a list of answers. Answer contains the answer text, answer id and id of the user who wrote the answer.
- **Example Response Value**

```
{
  "comment_id": 2455,
```



```

"answers": [
  {
    "answer_id": 452,
    "answer": "Yes, you can",
    "owner_id": 543
  }
]
}

```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
post_id	Id of an event post	Path	Integer
comment_id	Id of the comment of post	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Not found

Post Answer

POST /api/v1.0/<int:post_id>/comments/<int:comment_id>/answers

- **Author:** [Doğukan Akar](#)
- **Definition:** It gets a post id and a comment id as path parameters and comment and answer as request body. Firstly it checks if post, comment and user with given ids are existing in database, aborts 404 error if they are not. It posts the answer to the database and it returns a json that includes success status and id of the answer.
- **Example Response Value**

```

{
  "success": true,
  "answer_id": 5432
}

```

}

- Parameters

Parameter	Description	Parameter Type	Data Type
post_id	Id of an event post	Path	Integer
comment_id	Id of the comment of post	Path	Integer
owner_id	Id of the user who own the answer	Body	Integer
answer	Content of the answer	Body	string

- Response Messages
 - 200: Successful
 - 404: Not found

Equipment

Get Equipment By Id

GET /api/v1.0/equipment/<equipmentId>

- Author: [Ahmet Necip Görgülü](#)
- Definition: It gets the equipmentId of an equipment as parameter and returns the equipment's attributes.
- Example Response Value

```
{  
  "equipments": {
```

```
"content": "I have a pair of shoes in good condition that i want to sell.",
"equipment type": "Shoes",
"equipmentId": 2,
"link": "letgo.com/245323",
"ownerId": 1,
"title": "Tennis rackets for sale!",
"website name": "letgo"
}
}
.
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
equipmentId	Id of an equipment	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Not found

Get Equipment

GET /api/v1.0/equipments/

- **Author:** [Ömer Arslan](#)
- **Definition:** It gets randomly generated equipment from a third party api ([randomapi](#)) and returns some of the selected equipment attributes.
- **Example Response Value**

```
{
  "equipmentId": 3,
  "ownerId": 5,
  "title": "old racket",
  "location": "Arizona"
}
```

- **Response Messages**
 - 200: Successful
 - 500: Internal Server Error

Equipment Search

GET /api/v1.0/search-equipment-type/<string:equipmentType>

- **Author:** [Ömer Arslan](#)
- **Definition:** It searches the equipment that has the specific equipment type and returns that equipment's attributes.
- **Example Response Value**

```
{
  "equipments": {
    "content": "I have a pair of shoes in good condition that i want to sell.",
    "equipment type": "Shoes",
    "equipmentId": 2,
    "link": "letgo.com/245323",
    "ownerId": 1,
    "title": "Tennis rackets for sale!",
    "website name": "letgo",
    "location": "Istanbul"
  }
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
equipmentType	Type of an equipment	Path	String

- **Response Messages**
 - 200: Successful
 - 404: Not found

GET /api/v1.0/search-equipment-location/<string:location>

- **Author:** [Ömer Arslan](#)
- **Definition:** It searches the equipment according to equipment location and returns that equipment's attributes.
- **Example Response Value**

```
{
  "equipments": {
    "content": "I have a pair of shoes in good condition that i want to sell.",
    "equipment type": "Shoes",
    "equipmentId": 2,
    "link": "letgo.com/245323",
    "ownerId": 1,
    "title": "Tennis rackets for sale!",
    "website name": "letgo"
    "location": "Ankara"
  }
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
location	Location of an equipment	Path	String

- Response Messages
 - 200: Successful
 - 404: Not found

Post Equipment

POST '/api/v1.0/equipments'

- **Author:** [Mehmet Saraçoğlu](#)
- **Definition:** It creates a new equipment post according to the given information.
- **Example Response Value:**

```
{
  "content": "I have a pair of shoes in good condition that i want to sell.",
  "equipment type": "Shoes",
  "equipmentId": 2,
  "link": "letgo.com/245323",
  "ownerId": 1,
  "title": "Tennis rackets for sale!",
  "website name": "letgo"
  "location": "Istanbul"
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
content	user ID	Body	String

equipment_type	Type of the equipment	Body	String
equipmentId	ID of the equipment	Body	String
link	link of the website	Body	String
ownerId	ID of the owner	Body	String
title	title	Body	String
website_name	website name of the link	Body	String
location	location	Body	String

- **Response Messages**
 - 201: Created with Success
 - 400: Bad Request

Follow

Get Followers

GET /api/v1.0/users/<id:user_id>/followers

- **Author:** [Ege Can Kaya](#)

- **Definition:** It gets the ID of a user as parameter and returns the followers of that user.
- **Example Response Value**

```
{
  "followerId": [6, 9, 20, 23, 30, 74, 77, 128]
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
user_id	Id of a user	Path	Integer

- **Response Messages**
 - 200: Successful
 - 404: Not found

Follow User

POST /api/v1.0/users/<int:user_id>/followers

- **Author:** [Kerem Zaman](#)
- **Definition:** It gets the IDs of followed and following users as parameter and adds the new follow relation to the database.
- **Example Response Value**

```
{
  "follower_id":1,
  "following_id":1
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type

user_id	Id of a user	Path	Integer
follower_id	Id of new follower	Body	Integer

- **Response Messages**
 - 201: Created
 - 404: Not found

Block

Post Block

POST /api/v1.0/<id:user_id>/blocked-users

- **Author:** [Ömer Arslan](#)
- **Definition:** It posts the blocked IDs of the user and adds the blocked IDs to the database.
- **Example Response Value**

```
{
  "blocked_id":2,
  "blocking_id":1
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
user_id	Id of blocking user	Path	Integer

blocked_id	Id of blocked user	Body	Integer
------------	--------------------	------	---------

- **Response Messages**
 - 201: Successful
 - 404: Not found

Get Blocked Users

GET /api/v1.0/users/<int:user_id>/blocked-users

- **Author:** [Bengisu Özaydın](#)
- **Definition:** Returns the ID's of blocked user by the user having the given ID.
- **Example Response Value**

```
{
  "blocked_id":2,
  "blocking_id":1
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
user_id	ID of blocking user	Body	Integer

- **Response Messages**
 - 200: Successful

Notification

Post Notification

POST /api/v1.0/notification

- **Author:** [Ahmet Necip Görgülü](#)
- **Definition:** It checks if the recipientId is valid. If it is not it returns 404. If it is it inserts the new notification into the Notification database table.
- **Example Response Value**

```
{  
  "ID": 3,  
  "date": "22.10.2022",  
  "description": "You have been approved!",  
  "isRead": False,  
  "recipientID": 1,  
}
```

- **Parameters**

Parameter	Description	Parameter Type	Data Type
ID	ID of the notification	body	BigInteger
description	description of notification	body	String
recipientId	Id of the recipient	body	BigInteger

- **Response Messages**
 - 201: Created
 - 404: Not found

Get Notification

GET /api/v1.0/notification

- **Author:** [Şefika Akman](#)
- **Definition:** It gets notifications of a user by looking his/her user_id and return notifications.
- **Example Response Value**

```
{  
  "ID": 54,  
  "date": "12.02.2030",  
  "description": "",  
  "isRead": True,  
  "recipientID": 4  
}
```

- **Parameters**

There is no parameter.

- **Response Messages**
 - 200: Successful