Spring 2021
GROUP 4

Milestone Report 2

Amateur Sport Events Platform

10.05.2021

Berkay Gümüş
Ece Dilara Aslan
İhsan Mert Atalay
Mehmet Hilmi Dündar
Muhammed İrfan Bozkurt
Salih Furkan Akkurt
Tolga Kerimoğlu
Yağmur Selek
Yiğit Sarıoğlu

,

Contents

1. **Executive Summary**

   Introduction - What is this report about?
   Description - What are we building?
   Project Status - Where are we?
   - What has been done?
   - Reflections

   Moving Forward
   - What will be improved?
   - What new things are coming next?

2. **List and Status of Deliverables**
3. **Evaluation of Deliverables**
4. **Evaluation of Tools and Processes**

   GitHub
   Django
   SQLite
   Docker
   Visual Studio Code
   Postman
   Google Meet & Whatsapp

5. **Summary of Work Done Individually**
6. **Deliverables**

1. Executive Summary

Introduction - What is this report about?

We are a group of students taking the CMPE 352 course at Boğaziçi University. As part of the coursework, we are building an amateur sports events platform where people can both organize or find sport events to participate in. This report marks the second milestone in our project and documents everything that our team has done since the first milestone report.

Description - What is this project about ?

We are building an application that's going to be on both Android and the Web. The premise is building a community around amateur sports and creating a platform for people to organize amateur sport events and find nearby events they can attend to, after which they can comment on the event, share opinions, interact with other participants and overall be a part of the community for that sport. In addition to that, we also provide a side service where users can sell equipment and buy them, basically an in-app marketplace. After signing up and logging in to the system, the user can search for events based on location, sport type, skill level, date etc. to find a suitable one if he/she wishes to join or the user can choose to join events as a spectator just to meet people and enjoy the games. On the other side, if the user wants to create an event and find people to join the event, he/she can easily do that just by specifying the date and location and the type of the event. Of course additional descriptions can also be provided if desired. After joining a mutual event, the user can grant badges to other users depending on the impression they got from them and further, they can follow other users to see get notified of events they create. Finally, all users can access the marketplace where they can find and purchase equipment they might need for a particular sport and sell the equipment they already have

Project Status - Where are we?

In the first part of the semester, we have dived into important software engineering practices and its subparts such as requirements engineering, modeling using the Unified Modeling Language standards and working as a team overall and we marked those with the first Milestone Report. Afterwards, we have dived deeper into the implementation aspects, learned about APIs, containers and CI/CD. We were tasked with creating a RESTful API, building a web application around it, dockerizing it and deploying it on an Amazon AWS EC2 instance.

What has been done?

Over the past 3 weeks, everyone has created an API that supports the GET and POST methods which makes use of an external API which are documented below.
We have written unit tests for testing the functioning of our APIs and then created a Web application using the Django framework, reviewed the code written by our

teammates, requested code review and got hands-on with GitHub's pull requests. This app brought together and packaged all the different APIs created by different teammates. Afterwards we have dockerized our application and deployed it using Amazon Web Services. It can be accessed here: http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/
The documentation of our APIs can be accessed here (also included in the report): https://github.com/bounswe/2021SpringGroup4/blob/dev/practice-app/API-docs.md

Reflections

This assignment brought on many challenges that we haven't encountered before and overall it was the most challenging one. Fiddling with APIs was pretty fun but also allowed us to understand the inner workings of API design. We also had no idea about how to create and deploy a web application so we had to choose a framework and learn from scratch which was educational for everyone. Learning about Docker and containers is also a must for any software engineer going into the industry so we are happy overall that we got to experience and use these technologies.
Perhaps the most difficult and educative part for us though, was actually using a VCS i.e. Git in a proper manner, dealing with multiple branches, pull requests. We have had to resolve many conflicts, deal with many bugs but we came through in the end. Lots of hair pulling but also lots of fun and success!

Moving Forward

This Milestone marks actually the end of our semester and the course. Although we have been learning a lot and doing a lot of work, we haven't actually started building our application yet. Next semester, we will pick off from right here with full speed and we will hopefully create a fully working application from scratch. We are happy with what we have learned this semester and we are looking forward to what interesting challenges (and bugs :) the next semester will bring!

2. List and Status of Deliverables

All deliverables listed below are complete. Completion date implies the date of the last change made on them

DELIVERABLE	COMPLETION DATE
Code Documentation	08.06.2021
API Documentation	09.06.2021
Project Plan	09.06.2021
Practice-app Implementation	08.06.2021

3. Evaluation of Deliverables

- **Code Documentation**

Code documentation is perhaps the process we are most familiar with from our past experiences even though we realized it's deeper than how we learned it. It was helpful for us when we were doing code reviews and it will also be helpful when we look back at the code and try to understand what we thought when coding it. It's also helpful for any non-members reviewing the code to understand it easier.

- **API Documentation**

This was the deliverable containing information about the functionalities, return types and arguments of the API endpoints. It's very essential since people trying to use our API will refer firstly to this document to understand the endpoints. It's described in detail later in the report.

- **Project Plan**

Project plan is helpful for understanding what tasks everyone should be handling and overall very useful for planning and time management overall. We have used the plans from the previous milestone report and updated certain parts as needed.

- **Practice-App Implementation**

This Practice-App had many parts that needed to work together to create the overall app. First of all, everybody got hands-on experience in dealing with APIs and created their own API endpoints. We have had the chance to think about proper unit test design and different unit testing tools. We have learned about the Django framework for creating web applications and the rest framework. On the other side, we had lots of documentation to do and we have definitely got intimate with Git, dealt with Pull request, reviewed each others code, dealt with annoying merge conflicts and overall got a much better idea of the real-life software development workflow and ecosystem. We have used new technologies such as Docker and fiddled with Amazon Web Services to deploy our application, very important processes in the software engineering pipeline. Finally, we (although not too much) had to learn a few things about frontend development to actually have something that's easy on the eyes.

4. Evaluation of Tools and Processes

Github

GitHub is a Git repository hosting service platform for software development and version control with the features that make it particularly easy to work on the same project simultaneously, especially as a team, during project creation and development. Since Git is compatible with all Operating Systems and as Git's non-linear development behavior, we were able to progress by working in different branches and without missing the current codes pushed in our main branch, thanks to the fetch command, even though we were all doing different parts of the project simultaneously. Being able to pull requests before merging our codes with Github's merge command allowed us to comment and advise on each other's codes, and to merge all the codes in the most correct form after the necessary updates were made.

Django

Django is a high-level Python Web framework. Among the frameworks we reviewed, we chose this framework, which is convenient for all of us, because its main goals are simplicity, flexibility, reliability and scalability, and we have members in our team who have not generated code for such an application before so it was easy to work with and learn . We have implemented our code based on Django's Model-View-Template architecture. The model served as the interface of our data and here we wrote the general structure needed to store the data we want to protect in SQLite database. View is a user interface, we used this interface to create the output we wanted to see in the browser and it is actually the main part where we gathered all the api features and reflected it as an output product. And we used the Template to combine the static parts and dynamic content of the HTML output of the page we want to finally achieve. Although most of our team members did not work with this framework before, they learned it during the implementation of our Practice-App . Since we believe that the most effective way to learn these tools is to use these tools directly, and because those who have not used these tools before have had this opportunity in this project, we see these tools as one of the most important things that the project contributed to us.

SQLite

SQLite is one of the most widely used databases in the world. It can be used as an open alternative to writing XML, JSON, CSV. We found it to be a recommended database for small to medium websites, as it is easy to be configured and smoothly stores the file in an ordinary disk file.

Also since it supports terabyte-sized databases and gigabyte-sized strings even though maybe all these features were not strictly necessary for our practice app; because of all these reasons, we thought it is a database that we can use in our Amateur Sport Events Platform project, which we plan to implement.

Visual Studio Code

Visual Studio Code is a highly recommended IDE, especially for front-end applications. It helped us to use our time more efficiently due to its suggestions to install which plugins and libraries according to errors we get and also its auto fixing errors feature was really helpful. It was also easy to work between terminal, command line and git bash by using IDE's command line interface.

Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. Instead of creating a full operating system, a Docker Container has just the minimum set of operating system software needed for the application to run and rely on the host Linux Kernel itself. Also there is a integration between Docker and Amazon ECS, so you could set up an AWS context in one Docker command. For our project , every group member also created docker hub account where we could reach our repository using Docker.

Google Meet & Whatsapp

Google Meet is a video-communication service developed by Google. We usually held group and sub-group meetings through this platform. We worked with our team members using this platform.

WhatsApp is a free, multiplatform messaging app that lets you make video and voice calls, send text messages, and more. We used this app for quick communication between group members.

Postman

Postman is an API (application programming interface) development tool which helps to build, test and modify APIs. Almost any functionality that could be needed by any developer is encapsulated in this tool. We used Postman frequently when developing our endpoint. Postman is the easiest and common tool used to test API. While testing the endpoints, it makes it very easy to get responses without using frontend by simulating requests.

5. Summary of Work Done Individually

MEMBER	CONTRIBUTION
Berkay Gümüş	Attended group meetings. Studying Django, Git and its commands and watching tutorials. Checking other API implementations Creating an API (team API) showing NBA teams in a choice window with GET request, taking an NBA team after a user selects with POST request, and displaying the information about the team. Reviewing other APIs (Formula1 and Search User). Creating unit tests. Documenting my API.
Ece Dilara Aslan	Studied lecture slides about APIs and unit testing. Studied git by watching PS videos. Studied Django by watching this video series https://www.youtube.com/watch?v=SlyxjRJ8VNY&list=PLsyeobzWxl7r2ukVgTqIQcl-1T0C2mzau . Researched free APIs to use in this assignment. Created the Formula 1 API of the practice-app by using Ergast F1 API, its HTML pages and unit tests. Created a pull request (https://github.com/bounswe/2021SpringGroup4/pull/67) for my changes in the code base. Reviewed "Holidays api" and "Yagmurselek" pull requests. Added my API documentation into the https://github.com/bounswe/2021SpringGroup4/blob/master/practice-app/API-docs.md .
İhsan Mert Atalay	İnstalling django with its requirements and struggling with errors Attended group meetings Studying Django watching lessons and examples and learning about github usage Searching about API projects Creating project which shows weather condition of the Istanbul. Used Get request to manage my API from view.py Preparing front-end index.html for homepage but it is not necessary so it is not used.

Mehmet Hilmi Dündar	-Attending group meetings. It was important for organizing this comprehensive project. -Studying about django, html files, unit tests , code environments, and some useful technologies. It was hard, because it was the first time for me in these technologies and they were time consuming. -Searching about public apis and finding a proper api to use. It was also hard because finding free api is really hard. -Creating django api which is holidays api and unit tests which try to handle the all unwanted cases and they also handle some cases which has also restricted by html file because of modularity -Overviewing the team member's code and giving some feedback. We use the overviewing feature of github effectively and solve the some problems in the github. -Documenting the code and preparing the summary of my efforts.
Muhammed İrfan Bozkurt	Studied Django & created small example projects Got even deeper in Git usage Studied efficient API implementation Created the "Search User API" Created unit tests for it Helping teammates when they are stuck in implementation Applied minor changes to merge everyone's branches A little html & css work for the main page Documenting what I've done
Salih Furkan Akkurt	● Attended group meetings regarding the report ● Studied Django & Git ● Added 'random_article' and 'eq_post' APIs and their tests to the practice-app and their documentation on the corresponding md file ● Used Wiki's external api to get the random article in the api ● Added both APIs' template htmls to the app folder ● Reviewed other members' code and gave comments ● Created a pull request for my code contribution ● Merged my code to the 'dev' branch
Tolga Kerimoğlu	● I have created and set up the initial Django app so everyone could easily integrate their APIs. Also integrated the Django Rest Framework to the app since we were asked to create RESTful APIs.

	● Made connections on the app to utilize a MySQL server instead of the default SQLite serverless database but this change was reverted later. ● Created the register API which returns a registration page (HTML) on a GET request and creates the record of a user on the database for the POST request. It handles various cases and returns appropriate responses. I have also written 8 unit tests for testing different input cases. ● Created the places API which return a search page (HTML) on a GET request and for a POST request, takes a location and keyword as input and returns an HTML page of places near the given location that match with the entered keyword, sorted by distance and also displays the exact addresses and ratings of the places. It first converts the location provided by the user into coordinate values using Google's Geodesic API and afterwards does a search using Google's Nearby_Places API on those coordinates. Finally, it filters the output to extract the rating and address information inside the HTML with the Jinja2 syntax. ● I have step by step merged all branches into a single branch to create the complete app, fixing the conflicts in the process. Dockerized the application and uploaded the image to a Dockerhub repository. Afterwards, using the Amazon Elastic Beanstalk CLI Tool and a json configuration file, deployed the dockerized application. ● In the process of deployment, I have fixed various bugs on the frontend and kept updating and fixing as per warning of team members. ● Written the executive summary for this report. ● Created the submission report for the deliverables assignment which was about the application. ● Created 2 pull requests and reviewed 4 of them. Pointed out certain bugs/suggested better coding practices.
Yağmur Selek	I studied a django tutorial, I watched a tutorial series from youtube that was about an hour long, mostly I learned it from cmpe321 django-ps. And I was not good with git commands, the 352 ps was helpful about it.

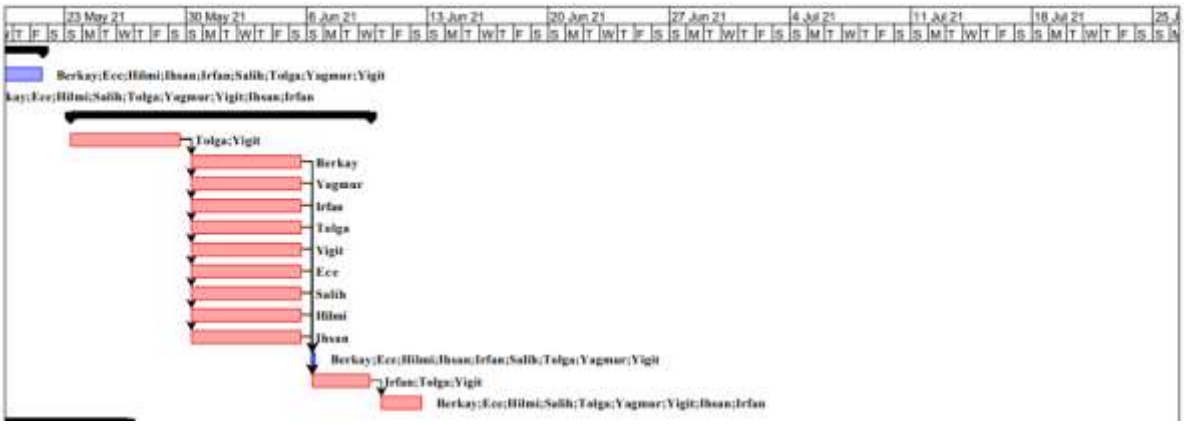
	I created the event post API. My API returns an HTML page to create an event post on a GET request. For POST request it creates the information about the event on the database . It checks for validity of the information and returns the different responses accordingly and I created 5 unit tests. I opened a google docs file for the report and made the general layout. I wrote the Evaluation of Tools and Processes. I reviewed 1 pull request.
Yiğit Sarıoğlu	● In the first week of the project, we have created and set up the initial Django app and integrated the Django Rest Framework to the app with Tolga. ● I implemented an API which returns a covid19 case reports by using an external API. There are 2 functions in API such as covid_api and covid_country_api. covid_api funcion is used to get the data (rankings top 20 country etc.) from the external API and shows when 'GET' method called. When 'POST' method called, same function takes the country code as a parameter calls the the covid_country_api function. It retrieves the specific country data from the external API using country code. The validity of the country code also checked before the json format data requested from the external API. If user enters nonvalid entry , httpResponse sends to the user. ● I also worked on the development of front-end side of the project. I prepared css file and main.html file which is served as application home page.Also I integrated this template(html-css) to all files to ensuring the visual integrity. ● I have worked with Tolga, when docker file prepared and docker image created, and amazon aws deployment.I helped him in this process. ● I worked on the creation and documentation on Milestone Report 2. ● I have opened a lot of issues on github to document our works, and commented other issues. ● I have created 2 pull requests and reviewed 4 of them.

	• I have reviewed : “Practice-app: Register API” and “ Practice-app: Formula 1 API”, “ Berkaygumus” and “salih” then gave feedback based on the errors I could find in their branch. • I documented the API endpoint which I implemented. It can be found at our GitHub repository under “API-Docs.md”

Deliverables

Project Plan

		Name	Duration	Start	Finish	Resource Names
69		**RESEARCH BEFORE PRE IMPLEMENTATION**	**5 days**	**5/17/21 8:00 AM**	**5/21/21 5:00 PM**	
70		research about Django, API and Git	5 days	5/17/21 8:00 AM	5/21/21 5:00 PM	Berkay;Ece;Hilmi;Ihsan;Ir...
71		preimplementation meting	0.5 days	5/17/21 8:00 AM	5/17/21 1:00 PM	Berkay;Ece;Hilmi;Salih;Tol...
72		**PRE IMPLEMENTATION**	**18 days**	**5/23/21 8:00 AM**	**6/9/21 5:00 PM**	
73		Server and API template	7 days	5/23/21 8:00 AM	5/29/21 5:00 PM	Tolga;Yigit
74		NBA Teams API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Berkay
75		Event Post API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Yagmur
76		Searh User API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Irfan
77		Register and Places APIs	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Tolga
78		Covid19 API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Yigit
79		Formula1 API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Ece
80		Random Article and Equipment Post API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Salih
81		Holiday API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Hilmi
82		Hava API	7 days	5/30/21 8:00 AM	6/5/21 5:00 PM	Ihsan
83		Pre Implementation Meeting	0.5 days	6/6/21 8:00 AM	6/6/21 1:00 PM	Berkay;Ece;Hilmi;Ihsan;Ir...
84		Deployment and Frontend	4 days	6/6/21 8:00 AM	6/9/21 5:00 PM	Irfan;Tolga;Yigit
85		MILESTONE 2 REPORT	3 days	6/10/21 8:00 AM	6/12/21 5:00 PM	Berkay;Ece;Hilmi;Salih;Tol...



API Documentation

Register API

- **Code Documentation**

The main script

```
"""
Created on May 23rd, 2021

This script handles the GET and POST requests to the register API
endpoint http://localhost:8000/api/register/

    'GET':
        Returns the registration page.
    'POST':
        Validates the information and registers the user into the
database if valid, returns a descriptive error if not
        Use the following JSON format to issue POST requests to this
endpoint
        JSON Format : { 'username': "",                 string, the
username selected by the user
```

```
                        'password': "",                 string, the
password selected by the user
                        'email':"",                     string, the
email address the user would like to register with
                        'fullname':"",                  string, full
name of the user
                        'description':"",               string, a
description about the user, can be NULL
                        'age':"",                       int, age of the
user, can be NULL
                        'location':"",                  string,
location provided by the user, can be NULL
                        'phone':"",                     string, phone
number provided by the user, can be NULL
                    }

@author: Tolga Kerimoğlu
"""

import copy, hashlib, random, json
from django.shortcuts import render
from rest_framework import serializers
from rest_framework.response import Response
from api.serializers import UserSerializer
from .forms import RegistrationForm

def register_api(request):
    """
    Process the GET and POST requests sent to the register API.

    This function processes the GET and POST requests seperately.
Returns the registration page for a GET request. For a POST request,
    it validates the information, encrypts the valuable fields such as
password and saves the user to the database and returns an HttpResponse
containing
    the meta-information of the newly registered user. If any errors
occur, returns a response describing what went wrong.

    Arguments:
    request (HttpRequest): django.http.HttpRequest object representing
the incoming request

    Returns(for POST requests):
```

```
    response (Response): rest_framework.Response object representing
the outgoing response

    """
    if request.method == 'GET':
        form = RegistrationForm() # Insantiate a registration form,
defined in forms.py
        return render(request, 'register.html', { 'form': form }) #
Render the HTML page, using the template in templates folder under the
root directory
                                                                    # The
form is accesible within the HTML using jinga2 syntax

    elif request.method == 'POST':
        response = Response() # Create a rest_framework.Response object
        response['Content-type'] = 'application/json' # Set it up as a
json response
        data = request.data
        # Extract user information from the request
        (username, password, email, description, age, location,
fullname, phone) = (data.get('username'),

data.get('password'),

data.get('email'),

data.get('fullname'),

data.get('description'),

data.get('age'),

data.get('location'),

data.get('phone'))

        if len(password) == 0:
            response.status_code = 400
            response.data = { 'password': 'A password must be
provided.'}
            return response
        hashed_pw = hashlib.sha256(password.encode()).hexdigest() #
Hash the password
```

```
        form = copy.copy(data) # Copy the data to a new dict
        form['hashed_pw'] = hashed_pw # Insert the hashed password
        serializer = UserSerializer(data = form) # Create a serializer
for the database entry of the user
        if serializer.is_valid(): # If valid, save user to the database
and return a success response
            response.status_code = 201
            response.data = { 'status': 'registration SUCCESSFUL'}
            serializer.save()
        else: # If not valid, return the appropriate page
            response.status_code = 400
            response.data = serializer.errors
            # TODO: Instead of returning the errors in JSON format,
return a proper HTML file displaying the error

    return response
```

Unit tests

```
from api.models import User
from api.serializers import UserSerializer
from rest_framework.test import APITestCase
from django.urls import reverse
from rest_framework import status

class RegistrationTests(APITestCase):
    def test_valid_registration(self):
        """
        Ensure we can register a new user with valid information.
        """
        data = { 'username': 'test_user', 'password': 'test_password',
'email': 'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50' }
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_valid_registration_with_extra_fields(self):
        """
        Ensure we can register a new user with additional fields appended to
the request data.
        """
```

```
        data = { 'username': 'test_user', 'password': 'test_password',
'email': 'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', 'useless': 'useless field'}
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_username_conflict(self):
        """
        Ensure the system detects when a user is already registered with the
username provided.
        """
        data = { 'username': 'test_user', 'password': 'test_password',
'email': 'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        self.client.post('/api/register/', data, format='json')
        data_conflict = { 'username': 'test_user', 'password':
'test_password', 'email': 'test_conflict@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        response = self.client.post('/api/register/', data_conflict,
format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_email_conflict(self):
        """
        Ensure the system detects when a user is already registered with the
email provided.
        """
        data = { 'username': 'test_user', 'password': 'test_password',
'email': 'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        self.client.post('/api/register/', data, format='json')
        data_conflict = { 'username': 'test_user_conflict', 'password':
'test_password', 'email': 'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        response = self.client.post('/api/register/', data_conflict,
format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

```
    def test_null_username(self):
        """
        Ensure we don't allow registrations without a username.
        """
        data = { 'username': '', 'password': 'test_password', 'email':
'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_null_email(self):
        """
        Ensure we don't allow registrations without an email address.
        """
        data = { 'username': 'test_user', 'password': 'test_password',
'email': '',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_null_password(self):
        """
        Ensure we don't allow registrations without a password.
        """
        data = { 'username': 'test_user', 'password': '', 'email':
'test@email.com',
        'description': 'I am a test description', 'location': 'test_location',
'age': '50', }
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_incomplete_registration(self):
        """
        Ensure that users can register without entering location, age or a
description.
        """
        data = { 'username': 'test_user', 'password': 'test_pw', 'email':
'test@email.com',
        'description': '', 'location': '', 'age': '' }
```

```
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
```

- **API Documentation**

Register

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/register/
http://localhost:8000/register/ **(If you are running the app locally)**

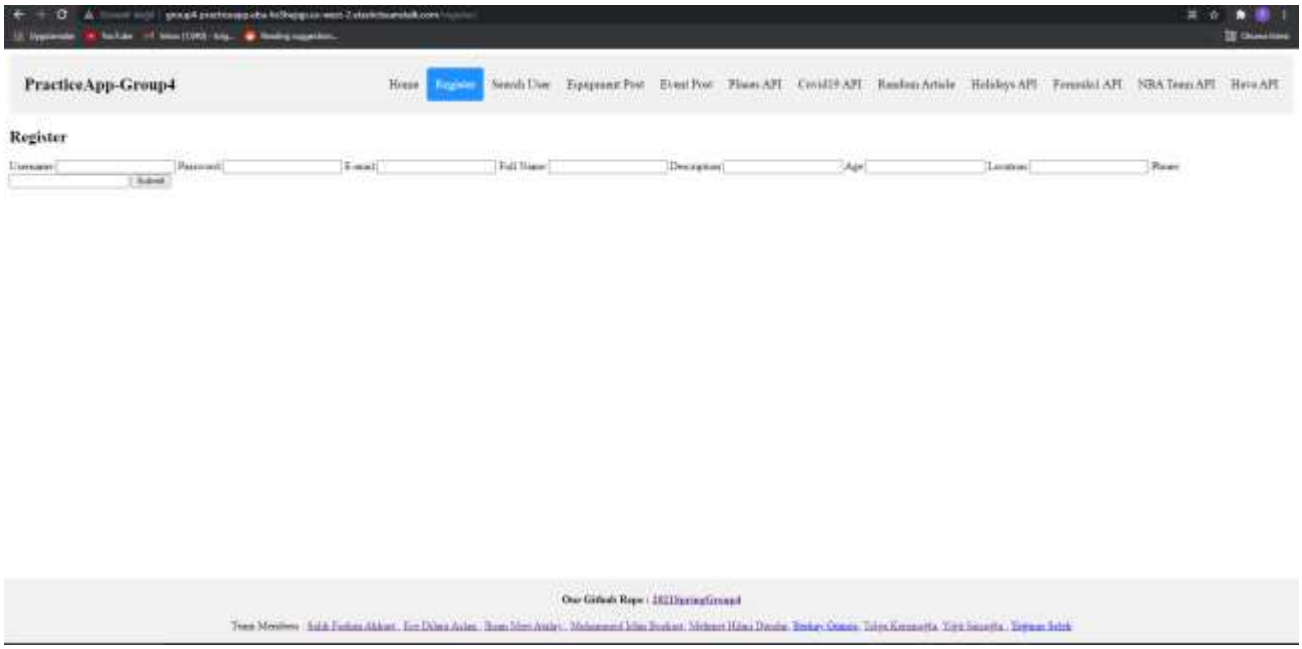
```
This endpoint is the registration interface to the system, running on
the default Django SQLite backend. Returns the registration page for a
GET request. For a POST request, it validates the information, encrypts
the valuable fields such as password and saves the user to the database
and returns an HttpResponse containing the meta-information of the newly
registered user. If any errors occur, returns a response describing what
went wrong.
```

```
'GET':
    Returns the registration page.

'POST':
    Validates the information and registers the user into the database if valid,
returns a descriptive error if not
    Use the following JSON format to issue POST requests to this endpoint
    JSON Format : { 'username': "",                 string, the username
selected by the user
                    'password': "",                 string, the password
selected by the user
                    'email':"",                     string, the email address
the user would like to register with
                    'description':"",               string, a description about
the user, can be NULL
                    'age':"",                       int, age of the user, can be
NULL
                    'location':"",                  string, location provided by
the user, can be NULL
                   }

 RESPONSE STATUS CODES
    GET:
        HTTP_200_OK : Successfully returns the registration webpage.
    POST:
        HTTP_201_CREATED : Successfully created the user in the database.
        HTTP_400_BAD_REQUEST : Something was wrong with the provided information
and no user was created. This could be due to the username already existing in
the database,
        empty fields etc.
```

GET Request

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/register/
http://localhost:8000/register/ **(If you have the application running on your local)**

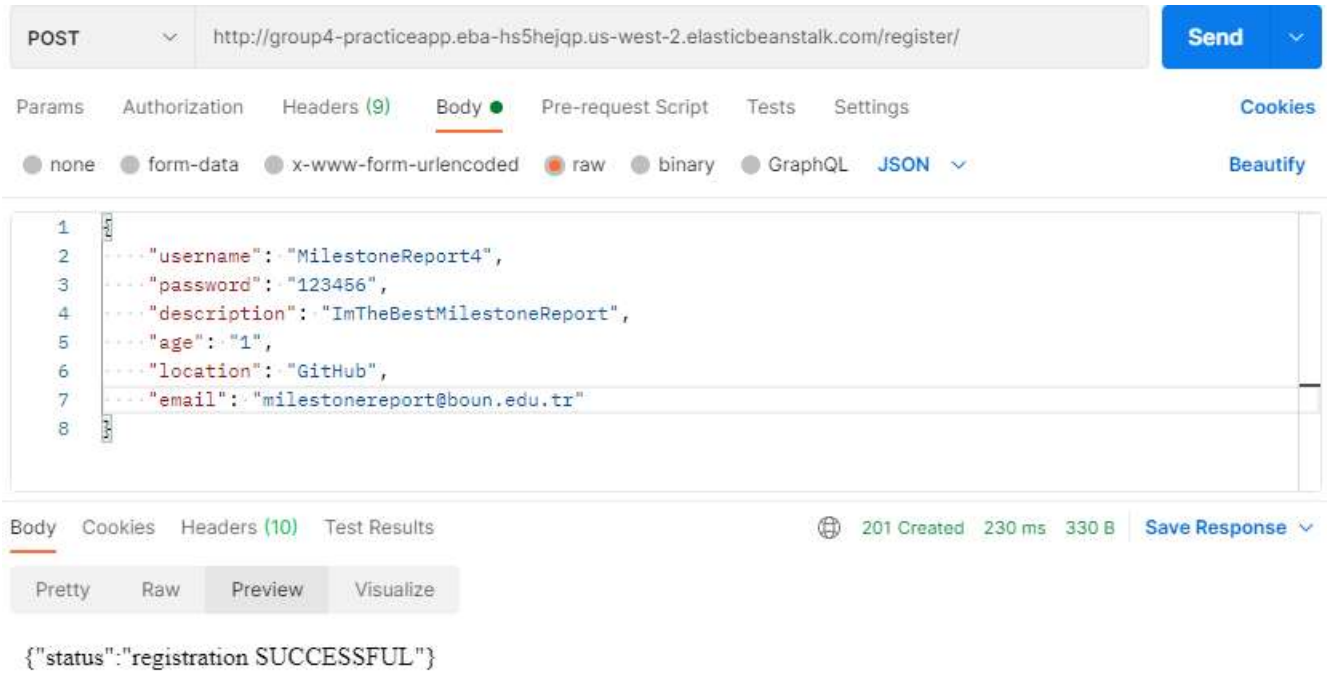
Returns an HTML page that contains the registration form



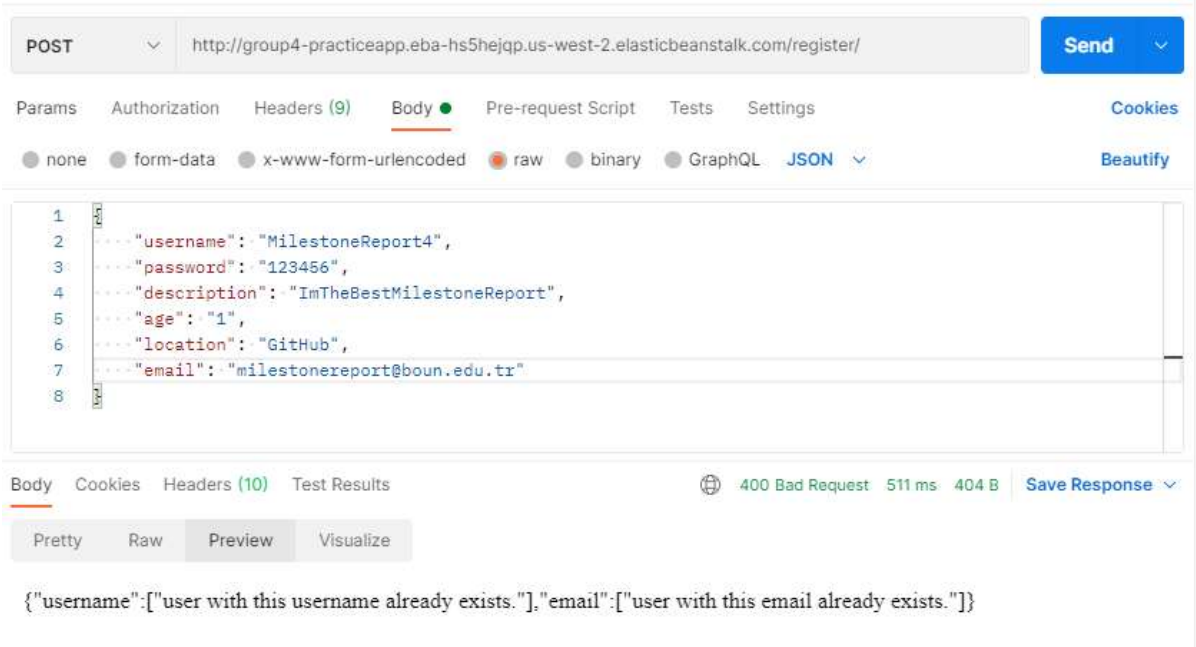
POST Requests

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/register/
http://localhost:8000/register/ **(If you have the application running on your local)**

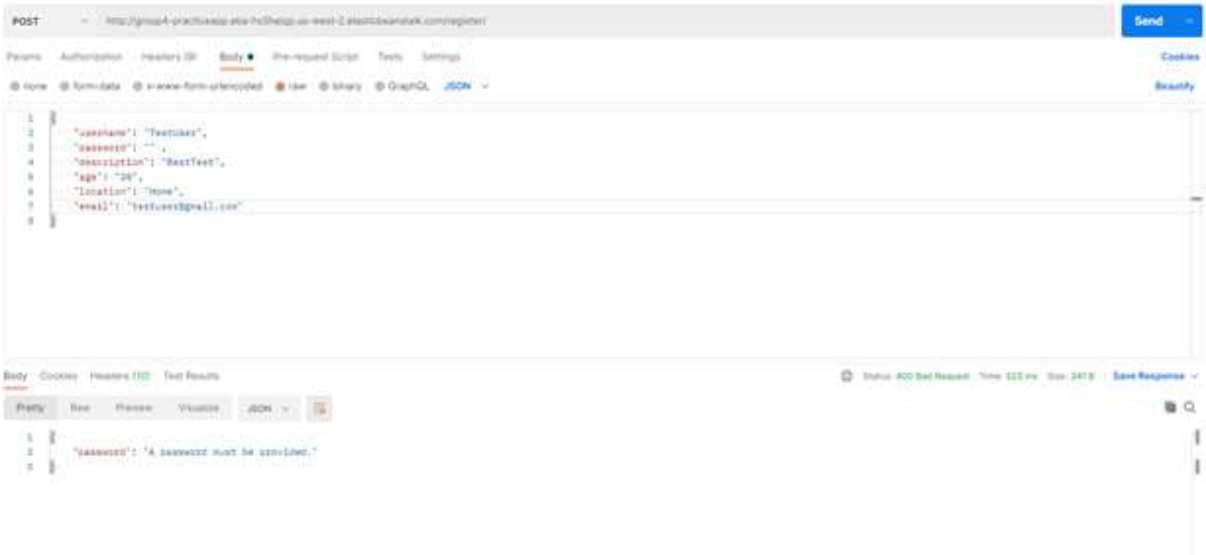
1. **Valid Request**



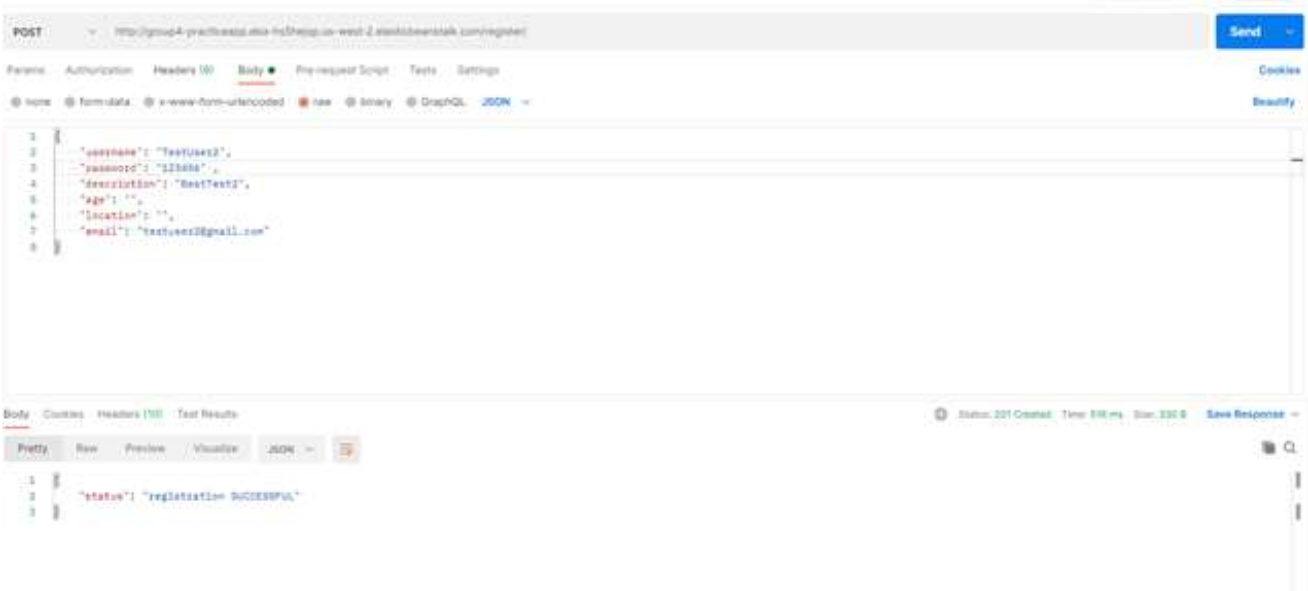
2. **Example Invalid Request (User Already Exists)**



3. **Example Invalid Request (Null Password)**



4. **Example Valid Request (User Doesn't Have To Provide Age or Location)**



Note that there are many other invalid and valid cases. You can find them documented in the unit tests code.

Places API

- ## Code Documentation

The main script

```
"""
Created on May 23rd, 2021

This script handles the GET and POST requests to the register API endpoint
http://localhost:8000/api/places/

    'GET':
        Returns the html for the search form.
    'POST':
        Using the location information provided by the user, first connects to
the Google's Geocode API
        to transform this location in text format into coordinates.
Afterwards, using these coordinates
        and the keywords provided, retrieves nearby information of nearby
locations and passes it to the
        Django template 'search_places.html' where the data is processed and
displayed to the user.

        JSON Format : { 'location': "",                 string, identifies the
location
```

```
                        'keyword': "",                  string, contains
relevant keywords about what type of place to search for

@author: Tolga Kerimoğlu
"""

import googlemaps
from django.shortcuts import render
from rest_framework.response import Response
from .forms import SearchPlacesForm


def places_api(request):
    """
    Process the GET and POST requests sent to the places API.

    This function processes the GET and POST requests seperately. Returns the
search page for a GET request. For a POST request,
    it first connects to the Google's Geocode API to transform this location
in text format into coordinates. Afterwards, using these coordinates
    and the keywords provided, retrieves nearby information of nearby
locations and passes it to the Django template
    'search_places.html' where the data is processed and displayed to the user
with address and ratings information displayed.

    Arguments:
    request (HttpRequest): django.http.HttpRequest object representing the
incoming request

    Returns(for POST requests):
    response (Response): an HttpResponse along with a rendered html file
    """
    if request.method == 'GET':
        form = SearchPlacesForm() # Initialize the search form
        return render(request, 'search_places.html', { 'form': form}) # Return
the form

    if request.method == 'POST':
        location, keyword = request.data['location'], request.data['keyword']
# Store information received from the user
```

```
        gmaps =
googlemaps.Client(key='AIzaSyBTjZQUnMQtaGDI_M_6Zrv0tHTh2sY767c') # Connect to
the Google Maps API
        loc = gmaps.geocode(location)[0]['geometry']['location'] # Convert the
entered location into coordinates using the geocode API
        address = gmaps.geocode(location)[0]['formatted_address'] # Store the
address
        lat, lng = int(loc['lat']), int(loc['lng']) # Store the latitute and
longitude information
        search = gmaps.places_nearby(location=(lat, lng), rank_by='distance',
keyword=keyword) # Use coordinate information to search for places nearby with
given keywords
        if (location == "" or keyword == ""): # Check for empty requests
            return render(request, 'search_places.html', {'fail': True},
status=400)
        return render(request, 'search_places.html', search, status=200) #
Return the response to be processed inside the html template
```

Unit Tests

```
from api.models import User
from api.serializers import UserSerializer
from rest_framework.test import APITestCase
from django.urls import reverse
from rest_framework import status

class SearchPlacesTests(APITestCase):
    def test_valid_search(self):
        """
        Ensure that we get a response after sending a non-blank request
        """
        data = { 'location': 'beşiktaş', 'keyword': 'üniversite'}
        response = self.client.post('/api/register/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_valid_search(self):
        """
        Ensure that the system doesn't accept an empty search.
        """
        data = { 'location': '', 'keyword': ''}
        response = self.client.post('/api/register/', data, format='json')
```

```
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

- ## API Documentation

Places

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/places/
http://localhost:8000/places/ **(If you are running the app locally)**

```
This endpoint is used for searching places nearby a given location
related to the given keyword. Returns the search page for a GET request.
For a POST request, it first connects to the Google's Geocode API to
transform this location in text format into coordinates. Afterwards,
using these coordinates and the keywords provided, retrieves nearby
information of nearby locations and passes it to the Django template
'search_places.html' where the data is processed and displayed to the
user with address and ratings information displayed.
URL: to be added.
```

```
'GET':
    Returns the html for the search form.
'POST':
    Using the location information provided by the user, first connects to the
Google's Geocode API
    to transform this location in text format into coordinates. Afterwards,
using these coordinates
    and the keywords provided, retrieves nearby information of nearby locations
and passes it to the
    Django template 'search_places.html' where the data is processed and
displayed to the user.

    JSON Format : { 'location': "",                 string, identifies the
location
                    'keyword': "",                 string, contains relevant
keywords about what type of place to search for
                    }
 RESPONSE STATUS CODES
    GET:
        HTTP_200_OK : Successfully conducted the search and returned matching
places.
    POST: .
        HTTP_400_BAD_REQUEST : Something was wrong with the provided
information. Most likely an empty search.
```

```
@author: Tolga Kerimoğlu
```

GET Request

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/places/

http://localhost:8000/places/ (If you have the application running on your local)

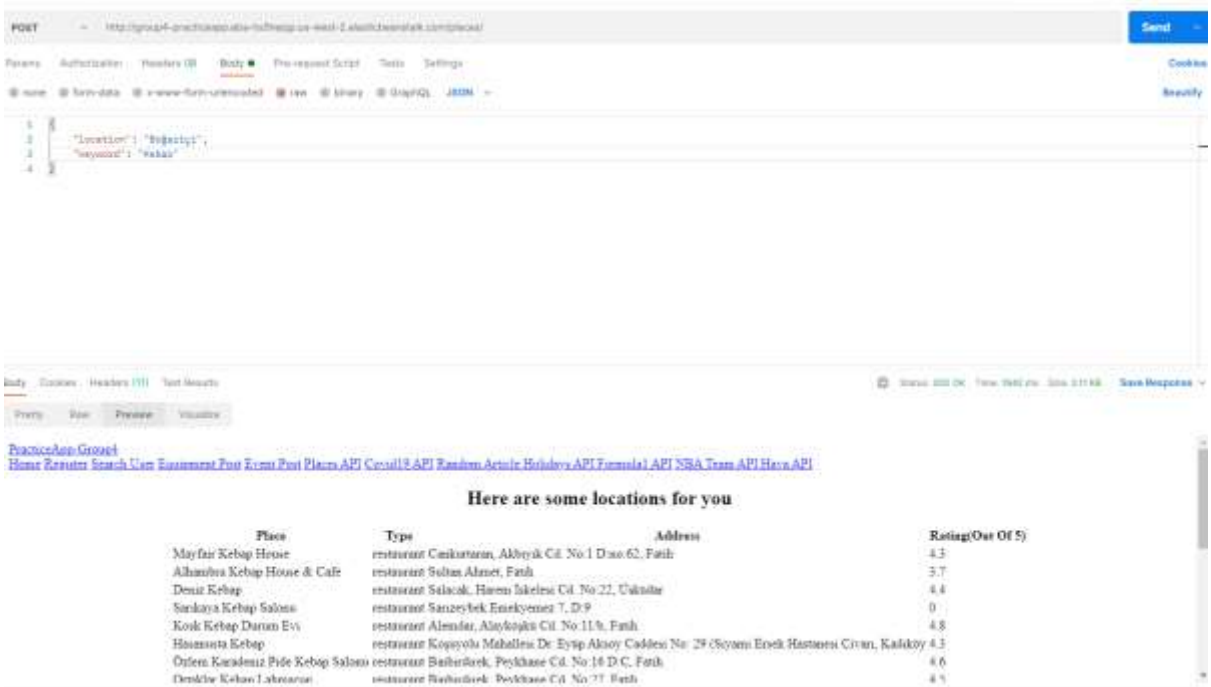
Returns an HTML page that contains the search places form.

POST Requests

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/register/

http://localhost:8000/register/ (If you have the application running on your local)

1. **Valid Request** (Returns an HTML page containing matching places with ratings)



eq_post API

place in repo:

https://github.com/bounswe/2021SpringGroup4/tree/dev/practice-app/api/eq_post

related template file:

https://github.com/bounswe/2021SpringGroup4/blob/dev/practice-app/templates/eq_post.html

code:

in main.py:

```
def eq_post_api(request):
    if request.method == 'GET':
        eq_post = EquipmentPost()
        return render(request, 'eq_post.html', { 'post': eq_post })

    elif request.method == 'POST':
        response = Response()
        response['Content-type'] = 'application/json'
        data = request.data
        (username, title, description, location) = (data.get('username'),
data.get('title'), data.get('description'), data.get('location'))
        if len(title) == 0:
            response.status_code = 400
            response.data = { 'title': 'A title must be provided.'}
            return response
        form = copy.copy(data)
        serializer = PostSerializer(data = form)
        if serializer.is_valid():
            response.status_code = 201
            response.data = { 'status': 'post SUCCESSFUL'}
            serializer.save()
        else:
            response.status_code = 400
            response.data = serializer.errors
    return response
```

in posts.py:

```
class EquipmentPost(forms.Form):
    username = forms.CharField(label='Username', max_length=30)
    title = forms.CharField(label='Title', max_length=50)
    description = forms.CharField(label='Description', max_length=250)
    location = forms.CharField(label='Location', max_length=60)
```

in test_eq_post.py

```
class PostTests(APITestCase):
    def test_valid_post(self):
```

```
        """
        Ensure post is valid.
        """
        data = { 'username': 'test_user', 'title': 'test_title', 'description': 'I
am a test description', 'location': 'test_location'}
        response = self.client.post('/api/eq_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_valid_post_with_extra_fields(self):
        """
        Ensure post is valid with additional fields appended to the request data.
        """
        data = { 'username': 'test_user', 'title': 'test_title', 'description': 'I
am a test description', 'location': 'test_location', 'useless': 'useless field'}
        response = self.client.post('/api/eq_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

    def test_null_username(self):
        """
        Ensure we don't allow posts without a username.
        """
        data = { 'username': '', 'title': 'test_title', 'description': 'I am a test
description', 'location': 'test_location'}
        response = self.client.post('/api/eq_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_null_title(self):
        """
        Ensure we don't allow registrations without a title.
        """
        data = { 'username': 'test_user', 'title': '', 'description': 'I am a test
description', 'location': 'test_location',}
        response = self.client.post('/api/eq_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_incomplete_registration(self):
        """
        Ensure that users can post equipment without entering location or a
description.
        """
        data = { 'username': 'test_user', 'title': 'test_title', 'description': '',
'location': ''}
        response = self.client.post('/api/eq_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
```

documentation:
Equipment Post

This endpoint is the equipment posting interface on the system. It returns the equipment posting page for a GET request. For a POST request, it checks if the required fields are filled. Then, it adds the equipment post to the database.

URL: http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/eq_post/

'GET':
Returns the equipment posting page.

'POST':
Checks the fields and adds the equipment post in the database, returns an error if not
Use the following JSON format to issue POST requests to this endpoint
JSON Format : { 'username': "", string, the username selected by the user
'title': "", string, the title selected by the user
'description':"", string, a description about the user, can be NULL
'location':"", string, location provided by the user, can be NULL
}

RESPONSE STATUS CODES
GET:
HTTP_200_OK : Successfully returns the equipment post page.
POST:
HTTP_201_CREATED : Successfully added the post in the database.
HTTP_400_BAD_REQUEST : Something was wrong with the provided information and no post was added.

@author: Salih Furkan Akkurt

example run:
[09/Jun/2021 16:02:03] "GET /api/eq_post/ HTTP/1.1" 200 995
[09/Jun/2021 16:02:04] "GET /api/eq_post HTTP/1.1" 301 0

example tests:
py manage.py test api.eq_post
Creating test database for alias 'default'...

System check identified no issues (0 silenced).
.....

Ran 5 tests in 0.030s

OK
Destroying test database for alias 'default'...

random_article

place in repo:
https://github.com/bounswe/2021SpringGroup4/tree/dev/practice-app/api/random_article
related template file:
https://github.com/bounswe/2021SpringGroup4/blob/dev/practice-app/templates/rand_wiki.html
code:
in main.py:

```
def ra_api(request):
    if request.method == 'GET':
        f=open('api/random_article/arena_list.txt','r')
        a_l = f.read().split('\n')
        f.close()
        r_n = random.randrange(len(a_l))
        r_url =
'https://en.wikipedia.org/w/api.php?action=query&titles={}&prop=extracts&format=jso
n'.format(a_l[r_n])
        req_page = requests.get(r_url)
        page_text = json.loads(req_page.text)
        page_text = list(page_text['query']['pages'].values())[0]['extract']
        soup = BeautifulSoup(page_text, 'html.parser')
        page_text = soup.get_text()
        if 'References' in page_text:
            idx = page_text.index('References')
            page_text = page_text[:idx]
        if 'See also' in page_text:
            idx = page_text.index('See also')
            page_text = page_text[:idx]
        if 'External links' in page_text:
            idx = page_text.index('External links')
            page_text = page_text[:idx]
        if page_text == '':
            response = Response()
```

```
            response['Content-type'] = 'application/json'
            response.status_code = 400
            return response
        return render(request, 'rand_wiki.html', {'rand_article' : page_text})
```

in test_ra.py:

```
class rand_article_tests(APITestCase):
    def test_article_empty(self):
        # Ensure fetched article not empty
        response = self.client.get('/api/random_article/', format='json')
        if len(response.content) == 0:
            self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

there is also a file 'arena_list.txt' I constructed, including sports arena ids I got from Wikipedia previously.

documentation:

Random Article

This endpoint is for getting an article off Wikipedia on a sports arena. It only has a GET request. It uses a predefined list of arenas and shows a random article.

URL: http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/random_article/

'GET':
Returns the random article on the page.

RESPONSE STATUS CODES
GET:
HTTP_200_OK : Successfully returns the article.
HTTP_400_BAD_REQUEST : The returned article was empty somehow.

@author: Salih Furkan Akkurt

example run:

[09/Jun/2021 16:02:51] "GET /api/random_article HTTP/1.1" 301 0
[09/Jun/2021 16:02:52] "GET /api/random_article/ HTTP/1.1" 200 347

example tests:

py manage.py test api.random_article
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

.

Ran 1 test in 0.912s

OK
Destroying test database for alias 'default'...

Create Event Post API

- **Code Documentation**

The script

-File Hierarchy

main.py

```
"""
 GET and POST requests to the  http://localhost:8000/api/event_post/

    'GET':
        Returns the  event creation page.
    'POST':
        Validates the necessary informations are filled
        and check for validity of that informations to record the event to our database if it is valid
        JSON format : { 'eventName': "",                string, the eventname entered by the event creator, also a primary key for db table
                        'sportType': "",                string, the sporttype entered by the user
                        'numOfPlayers ':"",             int, the number of players wanted for that event given by the event creator
                        'description':"",               string, a description about the event created by event creator, this can be NULL
                        'date':"",                      date(year:int, month:int, day: int) , a valid date that can be selected
                        'time':"",                      time (hour: int, min: int), a valid time that can be selected

@author: Yağmur Selek
"""
```

```python
from django.http.response import JsonResponse
import copy
from django.shortcuts import render

import time
import pytz
from datetime import datetime
from rest_framework.response import Response
from api.serializers import EventPostSerializer
from .forms import EventPost


def event_post_api(request):
    """  Process the GET and POST requests sent to the Create Event Post API.

    This function returns the Create Event Post Page for a GET request.
    For a POST request,
    it validates the information entered as 'event name' should be different than the events that currently
    residing in our SQLite database, 'date' and 'time' fields should be selected as future,
    then saves the event to the database and returns an HttpResponse containing
    the message says 'Event creation is successfull', if the request was valid.
    If any errors occur, returns a response default response page describing the error.

    Arguments:
    request (HttpRequest): django.http.HttpRequest object representing the incoming request

    RESPONSE STATUS CODES
      GET:
            HTTP_200_OK : Successfully returns the event post page.
      POST:
            HTTP_202_CREATED : Successfully added the event post desired to the database.
            HTTP_400_BAD_REQUEST : Serializer ERROR
            HTTP_101_BAD_REQUEST : SportType field is not provided
            HTTP_100_BAD_REQUEST : An eventName must be provided.
            HTTP_102_BAD_REQUEST : You can not enter a past date or time
```

```python
    todayDateTime= datetime.today()
    todayDate=todayDateTime.strftime('%Y-%m-%d')
    todayTime=todayDateTime.strftime('%H.%M')


    if request.method == 'GET':
        event_post = EventPost() #Initiliaze event post form defined in event_post/forms.py
        return render(request, 'event_post.html', { 'post': event_post }) #render the event_post.html page

    elif request.method == 'POST':
        #create the json response object
        response = Response()
        response['Content-type'] = 'application/json'
        data = request.data

        #get the data fields according to given order
        (eventName, sportType, numOfPlayers,description,date,time) = (data.get('eventName'), data.get('sportType'),
         data.get('numOfPlayers'), data.get('description'),data.get('date'),data.get('time'))

       #if event name field is not entered then give status code 400
        if len(eventName) == 0 :
            response.status_code = 100
            response.data = { 'eventName': 'A eventName must be provided.'}
            return response
         #if sportType field is not provided then give status code 401
        if len(sportType) == 0 :
           response.status_code = 101
           response.data = { 'sportType': 'A sportType must be provided.'}
           return response
        #if date field is provided as a past date then give status code 402
        if date < todayDate :
           response.status_code = 102
           response.data = { 'date': 'You can not enter a past date'}
           return response
        if date==todayDate and time<todayTime :
           response.status_code = 102
           response.data = { 'time': 'You can not enter a past time'}
           return response

        form = copy.copy(data)
```

```python


        serializer = EventPostSerializer(data = form)
        response.status_code = 202
        if serializer.is_valid(): # If valid
            response.status_code = 202 # return success status code 201
            response.data = { 'status': 'EVENT POST SUCCESSFUL'}
            serializer.save()  #save event to the database
        else: # If not valid
            response.status_code = 400
            response.data = serializer.errors
            return HttpResponse("<h1>Serializer ERROR </h1>")


    return response
```

forms.py

```python
"@author: Yağmur Selek"
from django import forms
class DateInput(forms.DateInput):
    input_type='date'
class TimeInput(forms.TimeInput):
    input_type='time'

class EventPost(forms.Form):
    eventName = forms.CharField(label='EventName', max_length=30)
    sportType = forms.CharField(label='SportType', max_length=50)
    numOfPlayers = forms.IntegerField(label='NumOfPlayers')
    description = forms.CharField(label='Description', max_length=250)
    date= forms.DateField(widget=DateInput)
    time= forms.DateField(widget=TimeInput)

```

test_event_post.py

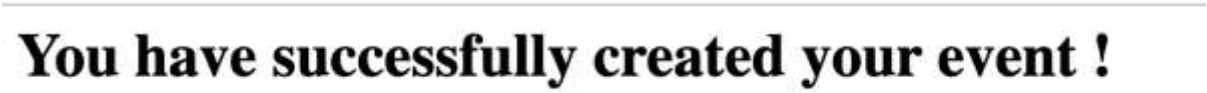
```python
"@author: Yağmur Selek"
from models import EventPost
from serializers import EventPostSerializer
from rest_framework.test import APITestCase
from django.urls import reverse
from rest_framework import status
import datetime

class EventPostTests(APITestCase):


    #Test_1 Ensures post is valid.

    def test_1(self):
        data = { 'EventName': 'test_EventName1', 'SportType': 'SportType1','NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2022,2,22) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)


    #Test_2 Ensures that you can not create an event for past dates

    def test_2(self):
        data = { 'EventName': 'test_EventName2','SportType': 'SportType2', 'NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2020,9,8) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_402_CREATED)


    #Test_3 Ensures that you can not create an event without providing an event name

    def test_3(self):
        data = { 'EventName': None, 'SportType': 'SportType3','NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2022,9,8) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_CREATED)



    #Test_4 Ensures that you can not create an event without providing an sport type

    def test_4(self):
        data = { 'EventName': 'test_EventName4','SportType': None, 'NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2022,9,8) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
```

```python
    #Test_4 Ensures that you can not create an event without providing an sport type

    def test_4(self):
        data = { 'EventName': 'test_EventName4','SportType': None, 'NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2022,9,8) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_401_CREATED)



    #Test_5 Ensures that you can not create an event without providing an sport type

    def test_5(self):
        data = { 'EventName': 'test_EventName4','SportType': None, 'NumOfPlayers': 3 ,
        'Description': 'test_Description' , 'Date' : datetime.date(2022,9,8) , 'Time': datetime.time(2, 3)}

        response = self.client.post('/api/event_post/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_401_CREATED)
```

GET Request :

POST Requests:

Valid Request
Returns the following html page



Invalid request:

NBA Team API

-Code Documentation

The Main Script

```python
"""
Created on 01.06.2021

This api gets an NBA team name and gives information about the team.

This endpoint is used for checking information for selected NBA team. Returns a
selection page for a GET request. For a POST request, when a user selects a team
and submit, it takes information using NBA-api and shows them at a new page.
Also, POST request can be used by adding the abbreavion of the desired team to
the url at selection page (/team/cle) to take information without selecting a
team and submitting at the page.

'GET':
    Returns the html page with a choice field showing all available teams and a submit
    button to select a team.
'POST':
    It connects to NBA-api and takes the data about desired NBA team. Then, this data
    is displayed at new page.

JSON Format : { 'team_code': "", string, identifies the desired team for searching }

@author: Berkay Gümüş
"""
from django.shortcuts import render
from .choice_team import ChoiceTeam
from rest_framework.response import Response
from nba_api.stats.static import teams

def team_api(request):
    if request.method == 'GET':

        context = {}
        context['form'] = ChoiceTeam(teams=teams.get_teams())
        return render(request=request, template_name='team.html', context=context)

def list_team_api(request, team_code=None):
    if request.method == 'GET':
        response = Response()    # Create a rest_framework.Response object
        response['Content-type'] = 'application/json'   # Set it up as a json response
        team_found = teams.find_team_by_abbreviation(abbreviation=team_code)
        if team_found is None:
            response.status_code = 400
            response.data = { 'NOT VALID ABBREVIATION' }
            return response
        else:
            return  render(request, 'list_team.html', team_found)

    elif request.method == 'POST':
        abbreviation:str = request.data.get('team_field')
        team_found = teams.find_team_by_abbreviation(abbreviation=abbreviation)

        return  render(request, 'list_team.html', team_found)
```

Form

```python
from django import forms

class ChoiceTeam(forms.Form):
    team_field = forms.ChoiceField(label='Team', choices=[])

    def __init__(self, teams=None, *args, **kwargs):
        super(ChoiceTeam, self).__init__(*args, **kwargs)
        if teams:
            self.fields['team_field'].choices = [
                (str(v['abbreviation']), str(v['full_name']))
                for k, v in enumerate(teams)
            ]
```

Unit Test

```python
from rest_framework import status
from rest_framework.test import APITestCase, APIClient

class NBATeamsTests(APITestCase):
    def test_valid_page(self):
        """
        ensure  that GET request is valid for endpoint http://localhost:8000/api/team/
        """
        client = APIClient()
        response = client.get('/api/team/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        print("OK")


    def test_valid_team(self):
        """
        ensure that we get a response after selecting a team
        """
        data = { 'team_code': 'cle'}
        response = self.client.post('/api/team/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        print("OK")
```

-API Documentation

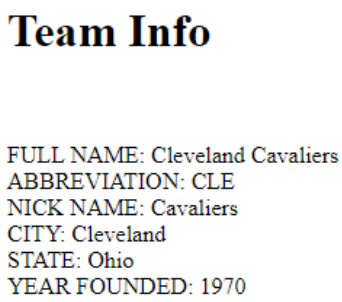
http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/team/

http://localhost:8000/team/ (If you are running the app locally)

```
This endpoint is used for checking information for selected NBA team. Returns a
selection page for a GET request. For a POST request, when a user selects a team
and submit, it takes information using NBA-api and shows them at a new page.
Also, POST request can be used by adding the abbreavion of the desired team to
the url at selection page (/team/cle) to take information without selecting a
team and submitting at the page.

'GET':
    Returns the html page with a choice field showing all available teams and a submit
    button to select a team.
'POST':
    It connects to NBA-api and takes the data about desired NBA team. Then, this data
    is displayed at new page.

JSON Format : { 'team_code': "", string, identifies the desired team for searching }

@author: Berkay Gümüş
```

GET Request

POST Request



Formula 1 API

Code Documentation

- The Main Script

```python
"""
Created on June 4th, 2021

This script handles the GET requests to the formula1 API endpoint http://localhost:8000/formula1/ ,
                and the GET and POST requests to the formula1 API endpoint http://localhost:8000/formula1/driver_info/

    'GET' - endpoint http://localhost:8000/formula1/:
        Returns the html for the current formula 1 driver standings and
        a search option of information about a driver from the upper table.

    'GET' - endpoint http://localhost:8000/formula1/driver_info/:
        Returns the html of the results page without a data,
        so shows a message to user and a link for the standings page.

    'POST' - endpoint http://localhost:8000/formula1/driver_info/:
        Returns the html for the information about the driver that is provided by the user.

        JSON Format : { 'driver_name': "",      string, identifies the name of the driver}

@author: Ece Dilara Aslan
"""

from django.shortcuts import render
import requests

# Returns the needed part of the API
def main():
    response = requests.get("http://ergast.com/api/f1/current/driverStandings.json")
    data = response.json()['MRData']['StandingsTable']['StandingsLists'][0]
    return data

# Creates a list of lists which consists of the ranks and corresponding informations
                                                    # such as driver name, constructor name,
                                                    # points and wins
def formula1_api(request):
    data = main()
    if request.method == 'GET':
        table = []
        for i in range(len(data['DriverStandings'])):
            table.append([i+1,
                          data['DriverStandings'][i]['Driver']['givenName']+" "+data['DriverStandings'][i]['Driver']['familyName'],
                          data['DriverStandings'][i]['Constructors'][0]['name'],
                          data['DriverStandings'][i]['points'],
                          data['DriverStandings'][i]['wins']])
        return render(request, "formula1_standings.html", {'year':data['season'], 'table':table}, status=200)
```

```python
def driver_info_api(request):
    data = main()
    table = []
    # Sends the information of the page is requested without a data
    if request.method == 'GET':
        return render(request, "driver_information.html", {'redirect':True}, status=400)
    # Sends the information of the driver provided by the user
    # There can be three cases: no driver name is provided,
                                # a driver which is not in the table is provided (invalid driver name),
                                # a valid driver name is provided
    elif request.method == 'POST':
        driver_name = request.data['driver_name']
        d_name = driver_name.lower().split() # name, surname or name and surname is accepted, not case-sensitive
        search_key = ""
        no_result = False
        status = 200
        if len(d_name) == 2:
            for i in range(len(data['DriverStandings'])):
                if d_name[0] == data['DriverStandings'][i]['Driver']['givenName'].lower() and d_name[1] == data['DriverStandings'][i]['Driver']['familyName'].lower():
                    temp = []
                    temp.append(data['DriverStandings'][i]['Driver']['givenName']+" "+data['DriverStandings'][i]['Driver']['familyName'])
                    temp.append(data['DriverStandings'][i]['Driver']['permanentNumber'])
                    temp.append(data['DriverStandings'][i]['Driver']['nationality'])
                    birth_date = data['DriverStandings'][i]['Driver']['dateOfBirth'].split('-')
                    temp.append(birth_date[2]+"/"+birth_date[1]+"/"+birth_date[0])
                    temp.append(data['DriverStandings'][i]['Driver']['url'])
                    table.append(temp)
                    search_key = temp[0]
        elif len(d_name) == 1:
            for i in range(len(data['DriverStandings'])):
                if d_name[0] == data['DriverStandings'][i]['Driver']['givenName'].lower() or d_name[0] == data['DriverStandings'][i]['Driver']['familyName'].lower():
                    temp = []
                    temp.append(data['DriverStandings'][i]['Driver']['givenName']+" "+data['DriverStandings'][i]['Driver']['familyName'])
                    temp.append(data['DriverStandings'][i]['Driver']['permanentNumber'])
                    temp.append(data['DriverStandings'][i]['Driver']['nationality'])
                    birth_date = data['DriverStandings'][i]['Driver']['dateOfBirth'].split('-')
                    temp.append(birth_date[2]+"/"+birth_date[1]+"/"+birth_date[0])
                    temp.append(data['DriverStandings'][i]['Driver']['url'])
                    table.append(temp)
                    search_key = d_name[0][0].upper() + d_name[0][1:]

        if driver_name == "":
            status = 400
        elif search_key == "":
            no_result = True
            status = 404
            for i in range(len(d_name)):
                search_key = d_name[i][0].upper() + d_name[i][1:]+ " "

        return render(request, "driver_information.html", {'redirect':False, 'no_result':no_result, 'search_key':search_key, 'table':table}, status=status)
```

- Unit Test

```python
from rest_framework import status
from rest_framework.test import APITestCase, APIClient

class Formula1Tests(APITestCase):
    def test_get_standings_page(self):
        """
        Ensure we can use GET request to endpoint http://localhost:8000/api/formula1/ properly
        """
        client = APIClient()
        response = client.get('/api/formula1/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_get_driver_info_page(self):
        """
        Ensure we cannot use GET request to endpoint http://localhost:8000/api/formula1/driver_info/
        """
        client = APIClient()
        response = client.get('/api/formula1/driver_info/')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_valid_input(self):
        """
        Ensure we can search a valid driver name/surname
        """
        data = {'driver_name':'Hamilton'}
        client = APIClient()
        response = client.post('/api/formula1/driver_info/', data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_valid_input_ignore_case(self):
        """
        Ensure we can search a valid driver name and surname, ignoring case
        """
        data = {'driver_name':'leWis HamiltoN'}
        client = APIClient()
        response = client.post('/api/formula1/driver_info/', data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_invalid_input(self):
        """
        Ensure we cannot search an invalid driver name
        """
        data = {'driver_name':'xxxxx'}
        client = APIClient()
        response = client.post('/api/formula1/driver_info/', data)
        self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)

    def test_no_input(self):
        """
        Ensure we cannot search without an input
        """
        data = {'driver_name':''}
        client = APIClient()
        response = client.post('/api/formula1/driver_info/', data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

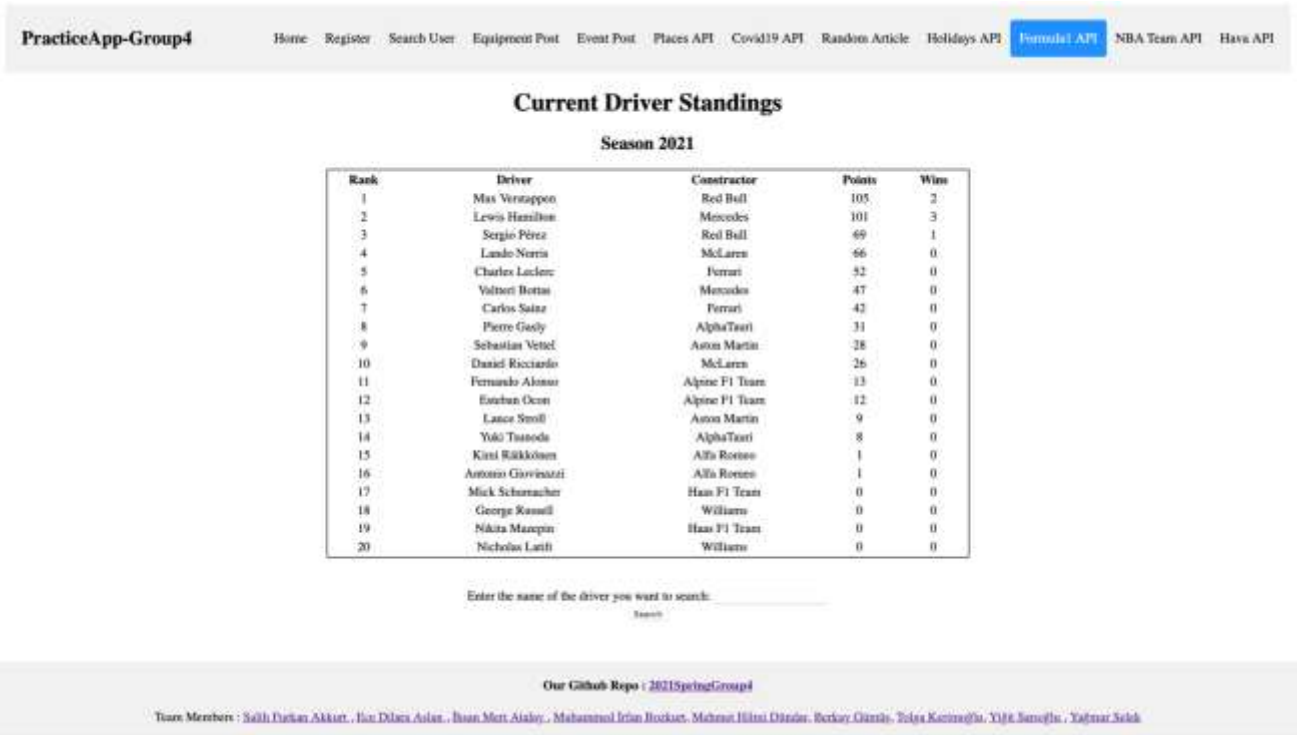
API Documentation

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/formula1/

http://localhost:8000/formula1/ (If you are running the app locally)

```
"""
Created on June 4th, 2021

This script handles the GET requests to the formula1 API endpoint http://localhost:8000/formula1/ ,
                and the GET and POST requests to the formula1 API endpoint http://localhost:8000/formula1/driver_info/

    'GET' - endpoint http://localhost:8000/formula1/:
        Returns the html for the current formula 1 driver standings and
        a search option of information about a driver from the upper table.

    'GET' - endpoint http://localhost:8000/formula1/driver_info/:
        Returns the html of the results page without a data,
        so shows a message to user and a link for the standings page.

    'POST' - endpoint http://localhost:8000/formula1/driver_info/:
        Returns the html for the information about the driver that is provided by the user.

        JSON Format : { 'driver_name': "",      string, identifies the name of the driver}

@author: Ece Dilara Aslan
"""
```

- GET Request to endpoint /formula1/



Rank	Driver	Constructor	Points	Wins
1	Max Verstappen	Red Bull	105	2
2	Lewis Hamilton	Mercedes	101	3
3	Sergio Pérez	Red Bull	69	1
4	Lando Norris	McLaren	66	0
5	Charles Leclerc	Ferrari	52	0
6	Valtteri Bottas	Mercedes	47	0
7	Carlos Sainz	Ferrari	42	0
8	Pierre Gasly	AlphaTauri	31	0
9	Sebastian Vettel	Aston Martin	28	0
10	Daniel Ricciardo	McLaren	26	0
11	Fernando Alonso	Alpine F1 Team	13	0
12	Esteban Ocon	Alpine F1 Team	12	0
13	Lance Stroll	Aston Martin	9	0
14	Yuki Tsunoda	AlphaTauri	8	0
15	Kimi Räikkönen	Alfa Romeo	1	0
16	Antonio Giovinazzi	Alfa Romeo	1	0
17	Mick Schumacher	Haas F1 Team	0	0
18	George Russell	Williams	0	0
19	Nikita Mazepin	Haas F1 Team	0	0
20	Nicholas Latifi	Williams	0	0

- POST Request to endpoint /formula1/driver_info/

Register Search User Equipment Post Event Post Places API Covid19 API Random Article Holidays API

You have searched for Verstappen

Name	Permanent Number	Nationality	Date of Birth	For more information
Max Verstappen	33	Dutch	30/09/1997	Click here!

Search User API

- **Code Documentation**

The main script (/practice-app/api/search_user/search_user.py)

```
"""

Created on May 23rd, 2021


This script handles the GET and POST requests to the user_profile API endpoint

http://localhost:8000/api/search_user/


    'GET':

        Returns an html page including a search bar to search a user by their

        username.


        An example GET request from terminal:
```

```
            curl http://127.0.0.1:8000/api/search_user/


    'POST':

        Checks if such a user exists.

        Returns a user profile page if so, and an error response if not.


        Use the following JSON format to issue POST requests to this endpoint

        JSON Format : { 'input': ""  string, the username you want to search }


        An example POST request from terminal:


            curl -X POST

                 -H "Content-type:application/json"

                 --data "{\"input\":\"<your search here>\"}"

                 http://127.0.0.1:8000/api/search_user/


@author: Irfan Bozkurt

"""

import copy, hashlib, random, json

from django.http.response import HttpResponse

from django.shortcuts import render

from rest_framework import serializers

from rest_framework.response import Response

from django.template import RequestContext
```

```python
from ..serializers import UserSerializer

from ..models import User




# First implement a search bar to retreive input from the API user.

# Its function will only be transmitting the search input from front-end to

# back-end,so we're using Django forms.

from django import forms


class SearchBar(forms.Form):

    input = forms.CharField(label='input', max_length=30)


"""

    Process the GET and POST requests sent to the search_user API.


    This function processes the GET and POST requests seperately. Returns

    a search bar for a GET request. For a POST request, it checks if the input

    is null. If not, checks if that user exists in database. If so, it returns

    profile information regarding that user.


        Arguments:

    request (HttpRequest): django.http.HttpRequest object representing the

    incoming request


        Returns(for POST requests):
```

```
    response (Response): rest_framework.Response object representing the

    outgoing response           , OR

    HttpResponse (request): django.http.HttpResponse object representing

    the outgoing response

"""


# For now, only able to find user with a complete username as input.

# Should implement a better search.

def search_user_api(request):


    if request.method == 'GET':


        # Insantiate a search bar, defined in bar.py

        bar = SearchBar() # Dynamic link for 'bar'


        # Render the HTML page, using the template in templates folder under

        # the root directory. The form is accesible within the HTML using
jinga2

        # syntax

        return render(request, 'search_user.html', { 'bar': bar })


    elif request.method == 'POST':


        response = Response() # Create a rest_framework.Response object

        response['Content-type'] = 'application/json' # Set it up as a json
response
```

```
        searchbar = request.data # We used a SearchBar object, and now
retrieving it

                                # to extract search input.

        # Check what is searched

        input = searchbar.get('input')



        # In case of empty input

        if not input:

            response.status_code = 400

            response.data = { 'error': 'You must enter at least one
character.'}

            return response



        # TODO: return a proper html instead of a JSON response

        # Control if input contains any character but numerical and
alphabetical ones:

        if not input.isalnum():

            response.status_code = 400

            response.data = { 'error': 'You must enter alphanumerical
characters.'}

            return response



        # TODO

        # Line below can well be changed with a well-designed search
algorithm.

        # For now, as we filter w.r.t the username, the list named "result"
will

        # contain 1 element only, which is supposedly the user we search for.
```

```
result = User.objects.filter(username = input)


# TODO: return a proper html instead of a JSON response
# If user not found:
if len(result) == 0:
    response.status_code = 400
    response.data = { 'error': 'User not found.'}
    return response


user = result[0]
# Now we have the user. Determine the empty fields and provide
# a dictionary to render along with the profile page.
userdict = user.__dict__


if not user.description:
    userdict['description']="No description provided."
if not user.phone:
    userdict['phone']="No phone number provided."
if not user.age:
    userdict['age']="No age provided."
if not user.location:
    userdict['location']="No location provided."
# If the user uploaded a pp, retrieve its url:
if user.profile_picture:
    userdict['profile_picture'] = str(user.profile_picture.url)
```

```
        # If the user hasn't provided a pp yet, use the UI Avatars API to

        # retreive a rounded picture with the user's initials on it:

        else:

            name = str(user.fullname).replace(" ","+")  # Format in a way that

            size = 128                                  # external API
desires.

            url = "https://ui-
avatars.com/api/?rounded=true&name={}&size={}&bold=true"



            # Put this URL to the dictionary to use in the <img> tag

            userdict['profile_picture'] = url.format(name, size)



        return render(request, 'search_result.html', {'userdict': userdict})
```

- **Unit tests (**/practice-app/api/search_user/test_search_user/test_search_user.py**)**

In this part, please note I've implemented my tests a while ago, and then we updated the mail URL of all the APIs.

It was ../api/search_user, or, ../api/register, but now the /api/ part is gone.

That's why one must change every request made in the code below from ../api/example to ../example.

```
from ...serializers import UserSerializer

from ...models import User
```

```
import unittest


from rest_framework.test import APITestCase

from rest_framework import status

import random, string


class Search_User_Test_Cases(APITestCase):


    if __name__ == '__main__':

        unittest.main()


    def test_typical_search(self):

        """

        Ensure we can search a user with valid input.

        First create a user with a random name, using register API

        """

        username = ''.join(random.choice(string.ascii_letters) for x in
range(5))

        password = ''.join(random.choice(string.ascii_letters) for x in
range(5))

        email = ''.join(random.choice(string.ascii_letters) for x in range(5))
+ '@' + ''.join(random.choice(string.ascii_letters) for x in range(5)) +
'.com'

        firstname = ''.join(random.choice(string.ascii_letters) for x in
range(5)).capitalize()

        lastname = ''.join(random.choice(string.ascii_letters) for x in
range(5)).capitalize()

        fullname = firstname + " " + lastname
```

```
        user_data = { 'username': username, 'password': password, 'email':
email, 'fullname': fullname}

        """

        Create a post request to the register API

        Control if registration is successful. Terminate test otherwise.

        """

        response = self.client.post('/api/register/', user_data,
format='json')

        if response.status_code != status.HTTP_201_CREATED:

            self.assertEqual(response.status_code, status.HTTP_201_CREATED)



        """

        Perform search using search_user API

        """

        data = { 'input' : username }

        response = self.client.post('/api/search_user/', data, format='json')

        self.assertEqual(response.status_code, status.HTTP_200_OK)


    def test_invalid_search(self):

        """

        Ensure we cannot search a user with poor input.

        Create a user with a bad username, using register API

        """

        username = ''.join(random.choice(string.ascii_letters) for x in
range(5)).join('#')
```

```
        password = ''.join(random.choice(string.ascii_letters) for x in
range(5))

        email = ''.join(random.choice(string.ascii_letters) for x in range(5))
+ '@' + ''.join(random.choice(string.ascii_letters) for x in range(5)) +
'.com'

        firstname = ''.join(random.choice(string.ascii_letters) for x in
range(5)).capitalize()

        lastname = ''.join(random.choice(string.ascii_letters) for x in
range(5)).capitalize()

        fullname = firstname + " " + lastname

        user_data = { 'username': username, 'password': password, 'email':
email, 'fullname': fullname}

        """

        Create a post request to the register API

        Control if registration is successful. Terminate test otherwise.



        For now, there's no barrier in front of creating a user with a poor
username.

        TODO:   sovle-implement the problem above.



        If the creation of the user fails, this test must fail:

        """

        response = self.client.post('/api/register/', user_data,
format='json')

        if response.status_code != status.HTTP_201_CREATED:

            self.assertEqual(response.status_code, status.HTTP_201_CREATED)



        """

        Perform search using search_user API
```

```
        The search should fail:

        """

        data = { 'input' : username }

        response = self.client.post('/api/search_user/', data, format='json')

        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)




    def test_user_not_found(self):

        """

        Ensure we cannot find a user that does not exist.

        Create a user using register API, but search something else.


        If that user exists, pick another username.

        """

        result = ["dummy string"]

        while (len(result) > 0) :

            username = ''.join(random.choice(string.ascii_letters) for x in
range(5))

            # TODO

            # Line below can well be changed with a well-designed search
algorithm.

            # For now, as we filter w.r.t the username, the list named
"result" will contain

            # 1 element only, which is supposedly the user we search for.

            result = User.objects.filter(username = username)

        """
```

```
        Now we have a username that does not exist.

        Perform search using search_user API

        The search should fail:

        """

        data = { 'input' : username }

        response = self.client.post('/api/search_user/', data, format='json')

        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

Screenshot of the results:

```
PS C:\Users\irfan\Desktop\REPORT\2021SpringGroup4\practice-app> python .\manage.py test .\api\search_user\
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
----------------------------------------------------------------------
Ran 3 tests in 0.182s

OK
Destroying test database for alias 'default'...
PS C:\Users\irfan\Desktop\REPORT\2021SpringGroup4\practice-app>
```

- **API Documentation**

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/search_user/

http://localhost:8000/search_user/ **(If you are running the app locally)**

```
This endpoint is the user search interface to the system, running
on the default Django SQLite backend. Returns an html page with a
search bar for a GET request. For a POST request, it checks if
input is valid and looks for the user in the database (username
and input must exactly be the same). Returns an error response if
user not found.

Returns a profile page of the user containing user information.
Blank sections say “Not provided”.

Likewise, if the user hasn’t provided a profile picture yet,
which they cannot for now, makes use of an external API named UI
Avatars to retrieve a rounded image with the user’s full name
initials in it.
```

```
'GET':

        Returns an html page including a search bar to search a user by
their

        username.


        An example GET request from terminal:


            curl http://127.0.0.1:8000/api/search_user/


'POST':

        Checks if such a user exists.

        Returns a user profile page if so, and an error response if not.


        Use the following JSON format to issue POST requests to this
endpoint

        JSON Format : { 'input': ""  string, the username you want to search
}


        An example POST request from terminal:


            curl -X POST

                 -H "Content-type:application/json"

                 --data "{\"input\":\"<your search here>\"}"

                 http://127.0.0.1:8000/api/search_user/
```

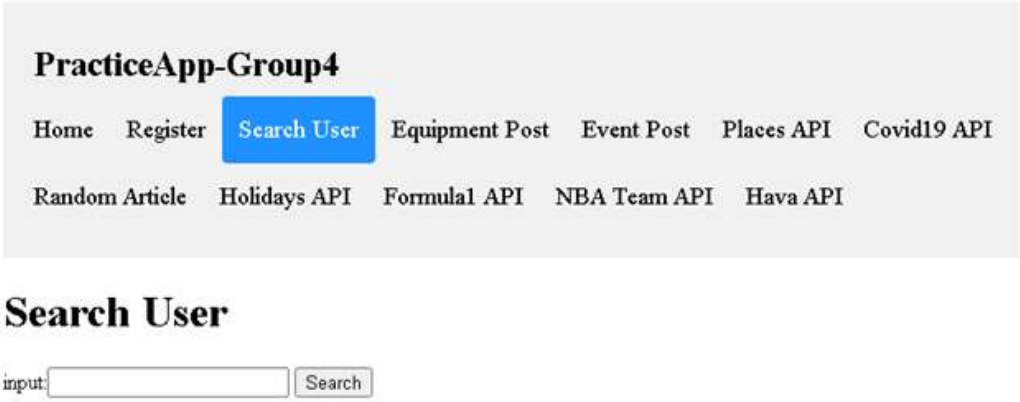
```
@author: Irfan Bozkurt
```

GET Request

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/search_user/

http://localhost:8000/search_user/ (If you are running the app locally)

Returns an HTML page that contains the registration form



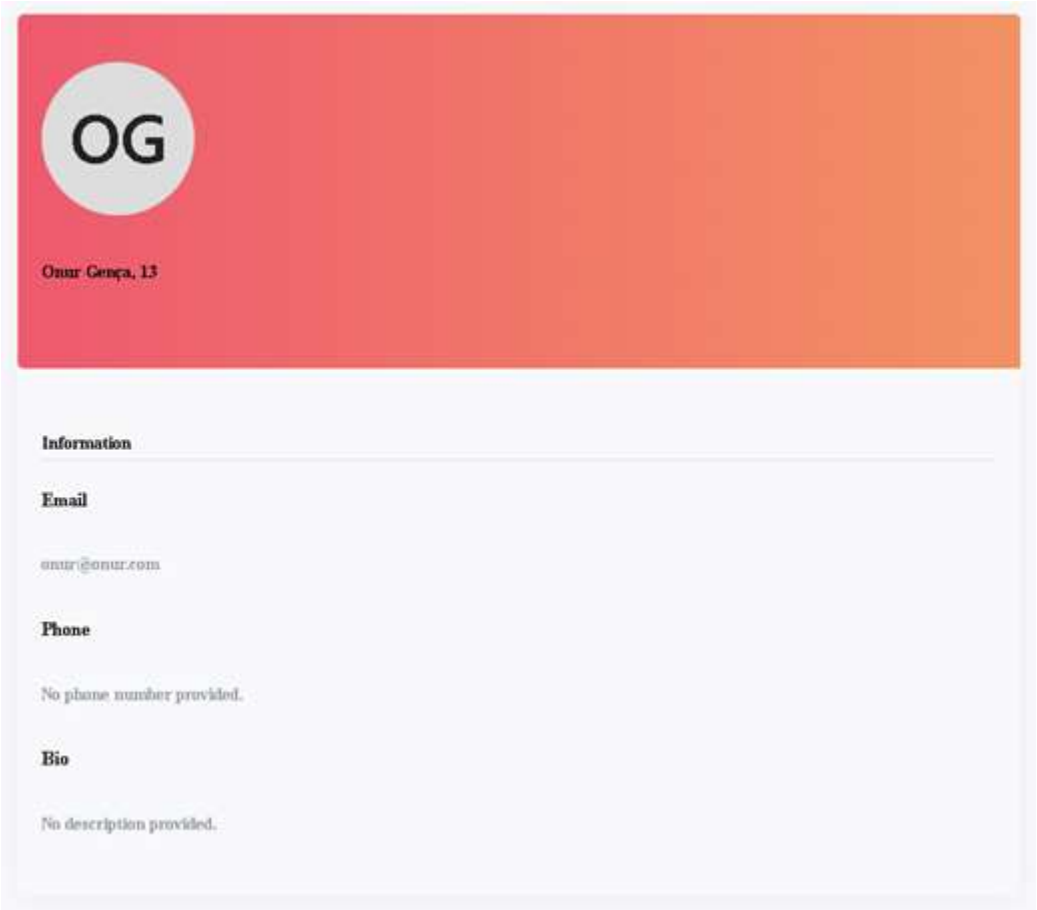
POST Requests

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/search_user/

http://localhost:8000/search_user/ (If you are running the app locally)

Example Valid Request

```
C:\Users\Irfan>curl -X POST -H "Content-type:application/json" --data "{\"input\":\"Onur\"}" http://group4-practiceapp.eba-fsihejgp.us-west-2.elasticbeanstalk.com/search_user/
<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="/static/search_result.css">
</head>

<body>
<div class="page-content page-container" id="page-content">
    <div class="padding">
        <div class="row container d-flex justify-content-center">
            <div class="col-xl-6 col-md-12">
                <div class="card user-card-full">
                    <div class="row m-l-0 m-r-0">
                        <div class="col-sm-4 bg-c-lite-green user-profile">
                            <div class="card-block text-center text-white">
                                <div class="m-b-25"> <img src="https://ui-avatars.com/api/?rounded=true&amp;name=Onur+Genca&amp;size=128&amp;bold=true" class="img-radius" alt="User-Profile-Image"> </div>
                                <h6 class="f-w-600">Onur Genca, 13</h6>
                                <i class=" mdi mdi-square-edit-outline feather icon-edit m-t-10 f-16"></i>
                            </div>
                        </div>
                        <div class="col-sm-8">
                            <div class="card-block">
                                <h6 class="m-b-20 p-b-5 b-b-default f-w-600">Information</h6>
                                <div class="row">
                                    <div class="col-sm-6">
                                        <p class="m-b-10 f-w-600">Email</p>
                                        <h6 class="text-muted f-w-400">onur@onur.com</h6>
                                    </div>
                                    <div class="col-sm-6">
                                        <p class="m-b-10 f-w-600">Phone</p>
                                        <h6 class="text-muted f-w-400">No phone number provided.</h6>
                                    </div>
                                </div>
                                <div class="row">
                                    <div class="col-sm-6">
                                        <p class="m-b-10 f-w-600">Bio</p>
                                        <h6 class="text-muted f-w-400">No description provided.</h6>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
</body>

</html>
C:\Users\Irfan>
```



Example Invalid Request (User Does Not Exists)

Note that there are other invalid cases. You can find them documented in the unit tests code.

##Holidays API

Code Documentation:

The main script (/practice-app/api/holidays/main.py)

api > holidays > main.py > ...

```python
"""
Created on Jun 5th, 2021

holidays api gets the date information by "datetime" and holiday information of the countries by google calender. Firstly
the api recieve the necessary informations by GET method in holiday_form=(day,month,year,country_code). Then,
country holiday information is taken from holidays as list. In this state, the api checks whether the input for date is
proper and if it is not the data_massage show this situtation. If the date format is correct, the program checks whether
the requestad date is holiday or not. If it is not holiday, The "date is proper" message should be showed. However, if it
is a holiday day, the program finds the first next day which is not holiday and the properr message should be showed.
In this state, the created massage is rendered with holidays.html by render method.


@author: Mehmet Hilmi Dündar
"""


from os import abort
from django.shortcuts import render
from .holiday_form import HolidayForm
from datetime import date, datetime, timedelta
import requests
import json


def holidays_api(request):

    #gathering data
    if request.method == 'GET':
        form = HolidayForm()
        return render(request, 'holidays.html', { 'form': form})

    #showing properr massage
    if request.method == 'POST':
        #gathered information
        day_input=request.data['day']
        month_input=request.data['month']
        year_input=request.data['year']
        country_code = request.data['country_code']
        country_code_lower = country_code.lower()

        error_case=False
        error_case_message=''

        if(day_input=='' or month_input=='' or year_input=='' or country_code==''):
            error_case_message='null element'
            error_case=True

        try:
            day = int(day_input)
            month = int(month_input)
            year = int(year_input)
        except:
            error_case_message='not integer'
            error_case=True

```

```
#checking error:
if error_case:
    return render(request, 'holidays.html', {'input_error': True, 'error_message': error_case_message}, status=400)


#necessary information
countries_dict={'turkey':'turkish','united states':'usa','germany':'german','azerbaijan':'az','france':'french','spain':'spain','italy':'italian','russia':'russian'}
google_key = "AIzaSyDyrUnwAI-XQgBwuDEhRjTiuo_cbbfw"


#checking for input
if (day<1 or day>31 or month<1 or month>12 or year<2021 or year >2022): # Check for inputs
    error_case_message='internal problem'
    error_case=True
elif((month==4 or month==6 or month==9 or month==11) and day==31) or (month==2 and ((year%4!=0 and day>=29) or  (year%4==0 and day>=30))):
    error_case_message='month-day disagreement'
    error_case=True
elif str(country_code_lower) in countries_dict.keys():
    error_case_message='wrong country'
    error_case=True


#checking error:
if error_case:
    return render(request, 'holidays.html', {'input_error': True, 'error_message': error_case_message}, status=400)



#determining the url

r_url="https://www.googleapis.com/calendar/v3/calendars/en.".+countries_dict[country_code_lower]+"%23holiday%40group.v.calendar.google.com/events?key="+google_key



#getting information from url
response=requests.get(r_url).json()
country_holidays_general=response['items']
holiday_dict={}

for x in country_holidays_general:
    holiday_dict.update({x["start"]["date"]:x["summary"]})


#determining correct date which is the first next day which is not holiday
requested_day=date(year, month, day)
requested_day_string=requested_day.strftime("%Y-%m-%d")
is_holiday=(requested_day_string in holiday_dict)
correct_date = requested_day
correct_date_string = correct_date.strftime("%Y-%m-%d")
while(correct_date_string in holiday_dict):
    correct_date += timedelta(days=1)
    correct_date_string = correct_date.strftime("%Y-%m-%d")
#preparing message
requested_day_string2=""
holiday_name=''
next_day=""
is_holiday_string=''

requested_day_string2=requested_day.strftime("%d/%m/%Y %a")+""
if(is_holiday):
    holiday_name=holiday_dict.get(requested_day_string)
    next_day=correct_date.strftime("%d/%m/%Y %a")+""
    is_holiday_string='True'
else:
    is_holiday_string='False'

legal_date_string=True

return render(request, 'holidays.html', {'input_error': False,'legal_date':legal_date_string, 'requested_date':requested_day_string2, 'is_holiday':is_holiday_string, 'holiday_name': holiday_name, 'next_day': next
```

The form (/practice-app/api/holidays/holiday_form.py)

```
from django import forms

country_names=(
    ("Turkey", "Turkey"),
    ("United States", "United states"),
    ("Germany", "Germany"),
    ("Azerbaijan", "Azerbaijan"),
    ("France", "France"),
    ("Spain", "Spain"),
    ("Italy", "Italy"),
    ("Russia", "Russia"),

)


class HolidayForm(forms.Form):
    day = forms.IntegerField(label='day',max_value=31, min_value=1, required=False)
    month = forms.IntegerField(label='month',max_value=12,min_value=1)
    year = forms.IntegerField(label='year',max_value=2022, min_value=2021)
    country_code = forms.ChoiceField(label='Country Name', choices=country_names)
```

Unit tests for holidays api

The test (/practice-app/api/holidays/test_holidays.py)

```python
from rest_framework.test import APITestCase
from rest_framework import status

class HolidaysTests(APITestCase):
    #testing whether the host answers requests with correct input

    def test_valid_input_correct(self):
        data = { 'day': '9', 'month': '4', 'year': '2021','country_code':'united states'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    # testing the wrong date format extending date limit
    def test_valid_interval_problem(self):
        data = { 'day': '39', 'month': '4', 'year': '2021','country_code':'united states'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    # testing the wrong date such as 30 febr
    def test_valid_month_day_disagreement(self):
        data = { 'day': '29', 'month': '2', 'year': '2021','country_code':'united states'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    # testing the country is not in the choices
    def test_valid_wrong_country(self):
        data = { 'day': '9', 'month': '4', 'year': '2021','country_code':'south korean'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    # testing empty input
    def test_valid_null_element(self):
        data = { 'day': '', 'month': '4', 'year': '2021','country_code':'turkey'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    # testing wrong type input
    def test_valid_null_element(self):
        data = { 'day': '2', 'month': 'april', 'year': '2021','country_code':'turkey'}
        response = self.client.post('/api/holidays/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

Documentation:

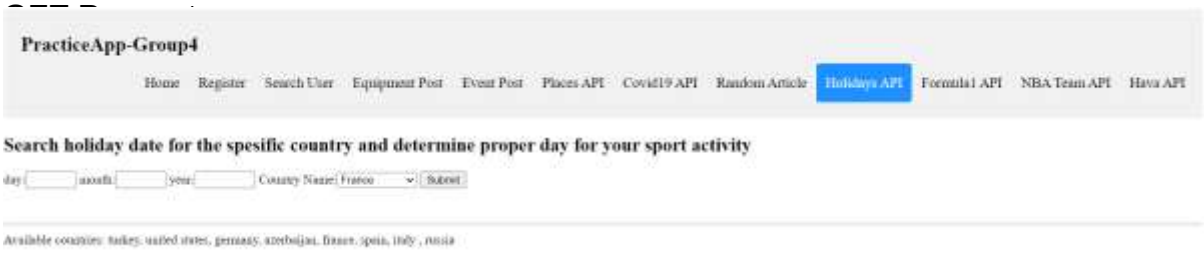
Holidays

This endpoint is for checking holidays for selected country. Gets the date and country information from user by GET request and it firstly tests the validity of this date. Then, Gets the holidays for the input country by google calender url as JSON format. Determines whether this date is a holiday and if it is, returns the holiday information and the next date which is not holiday by POST request. Also, if there is an wrong input. Determines source of the error and returns the proper error message by POST request. Finally,The returned data is displayed in html file.

URL: http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/holidays/

```
'GET':
    Returns an html page including a holidays form (day,month,year,country) to get input by user.


'POST':
    By spesific country name which is provided by the user,  determines the google calender url and gets holidays
    for this country as json format. By date which is also provided by the user, checks whether this date is in the
    holidays list. For holiday dates, Calculates the next not holiday date and displays the proper information.
    For the wrong inputs, It handles this cases and displays error message.


  JSON Format : { 'input_error': "",                boolean, Identifies whether there is an error
                  'legal_date': "",                 boolean, Identifies whether the input date is legal
                  'requested_date': "",             string, input date
                  'is_holiday': "",                 boolean, Identifies whether the input date is a holiday
                  'holiday_name': "",               string, holiday name for the input date, if it is not '-'
                  'next_day': "",                   string, next not holiday date, if it is not '-'
                  'error_message': "",              string, type of error, if it is not '-'
                  }
 RESPONSE STATUS CODES
    GET:
        HTTP_200_OK : Successfully gets the input of the date and the country.

    POST:
        HTTP_200_OK : Successfully proceeded the getting holidays information and input format is legal.
        HTTP_400_BAD_REQUEST : There is a problem in the input such as illegal date, not country in the list, or empty input
```

@author: Mehmet Hilmi DÜndar



POST Request: (for not holiday legal date)

POST Request: (for holiday legal date)

POST Request: (for not legal date)

Covid19 API

- **Code Documentation**

The main script

```python
"""
Created on 30.05.2021

This script handles the GET and POST requests to the covid19 API endpoint http://localhost:8000/api/covid19/
This api gets the latest covid19 data, shows the organized and sorted data.
Users also could search according to country code
    'GET':
        Returns the html page for the case reports all over the world
    'POST':
        Using the country code information , provided by the user, it connects to the CovidPy API,
        to take the country data. Retrieves the data and passes it to the Django template "covid_country_report.html"
        where the data is processed and displayed to the user.
        JSON Format : { 'countrycode': "", string, identifies the country code user search for it
@author: Yigit SARIOĞLU
"""
from django.shortcuts import render
import COVID19Py
import pycountry
from rest_framework.response import Response
from .form_search import SearchForm
from django.http import HttpResponse
#This method returns the data of the all over the world
#Also users could search specific country
def covid_api (request):
    covid19 = COVID19Py.COVID19()  #create a new instance of covid19
    if request.method == 'GET':
        form=SearchForm()       #initialize a form object

        latest = covid19.getLatest()   #Getting latest amount of total confirmed cases, deaths, and recoveries
        confirmed= latest["confirmed"] #latest amount of total confirmed case all over the world
        death = latest["deaths"]  #latest amount of total deaths all over the world
        recover=latest["recovered"]  #latest amount of total recover all over the world

        deathranks = covid19.getLocations(rank_by='deaths')          #  rank the results deaths .
        confirmedranks = covid19.getLocations(rank_by='confirmed')   #  rank the results confirmed

        #here list1 takes the name of the top 20 countries according to death
        list1=[]
        for x in range(20) :
            list1.append(deathranks[x]["country"])
```

```python
        #here list2 takes the name of the top 20 countries according to confirmed case
        list2=[]
        for x in range(20) :
            list2.append(confirmedranks[x]["country"])

        # Returns the search form for the user, and the latest amount of total confirmed cases, deaths, and recoveries information.
        # Also returns death rankings and confirmed case rankings of countries
        return render(request, 'covid_reports.html', {'confirm' : confirmed, 'recovery' : recover,
                                                      'death' : death, 'sform':form, "deathrank":list1, "confirmrank":list2 })

    elif request.method == 'POST':
        form = SearchForm(request.POST)    #initialize a form object
        if form.is_valid():
            country = form.cleaned_data.get('country')  #gets the country code from the form
            if len(country) != 2:  #if the country code length is not equal to 2, it is not a valid countrycode, so it returns HttpResponse
                return HttpResponse("<h1>Not Valid Country Code..Code length should be 2 .Please write valid code</h1>")

            #Using pycountry package, specific countries can be looked up by their various codes
            elif pycountry.countries.get(alpha_2=country):   #checks the country code, with using pycountry
                return covid_country_api(request, country) # if it is valid, it calls the covid_country_api method to show the specific country data
            #elseif that country code is not valid, HttpResponse needed
                return HttpResponse("<h1>Not Valid Country Code..Your Country Code is not in the table. Please look the country table. Then write a valid code</h1>")

        else:
            return HttpResponse("<h1>Not valid form request </h1>")

    else:

        return  HttpResponse("<h1>Not valid request </h1>")

#This method returns selected country covid19 data(confirmed,death,recover)
```

```python
#this method returns selected country covid19 data(confirmed,death,recover)

def covid_country_api (request,countrycode):

    covid19 = COVID19Py.COVID19() #Create a new instance of covid19
    # GET method : Users could search the country using : /covid19/countrycode
    # For example: /covid19/TR  calls the "GET" and returns the turkey covid data
    if request.method == 'GET':  #if GET method is called

        if len(countrycode) != 2:   #a valid country code should be 2.this statement check this condition
            return HttpResponse("<h1>Not Valid Country Code..Code length should be 2 .Please write valid code</h1>")

        elif pycountry.countries.get(alpha_2=countrycode): #checks the country code, with using pycountry

            locationdata = covid19.getLocationByCountryCode(countrycode)  #get the data according to specific country code
            countryname = locationdata[0]["country"]          # gets the country name data
            confirmed = locationdata[0]["latest"]["confirmed"]   # gets the country confirmed cases
            deaths = locationdata[0]["latest"]["deaths"]         # gets the country death data
            recovered = locationdata[0]["latest"]["recovered"]    # gets the country recovered  data
            updatetime = locationdata[0]["last_updated"]         #gets the update time

            #Returns the confirmed cases,deaths ,recovery , update time of data and country name
            return render(request, 'covid_country_report.html',
                      {'cname': countryname, 'confirm': confirmed, 'death': deaths, 'recover': recovered,
                       'time': updatetime})
        else:  #if that country code is not valid, HttpResponse sended
            return HttpResponse(
                "<h1>Not Valid Country Code..Your Country Code is not in the table. Please look the country table. Then write a valid code</h1>")


    # POST method : called when user post a request
    elif request.method == 'POST':

        locationdata = covid19.getlocationByCountryCode(countrycode)
        countryname = locationdata[0]["country"]          # gets the country name data
        confirmed = locationdata[0]["latest"]["confirmed"]  # gets the country confirmed cases
        deaths = locationdata[0]["latest"]["deaths"]        # gets the country death data
        recovered = locationdata[0]["latest"]["recovered"]  # gets the country recovered  data
        updatetime = locationdata[0]["last_updated"]        #gets the update time

        #Returns the confirmed cases,deaths ,recovery , update time of data and country name
        return render(request, 'covid_country_report.html',
                      {'cname': countryname, 'confirm': confirmed, 'death': deaths, 'recover': recovered,
                       'time': updatetime})
```

Unit Tests

```python
from rest_framework.test import APITestCase
from rest_framework import status


class Covid19Tests(APITestCase):
    def test_valid_searchform(self):
        """
        Ensure we could search a valid country
        """
        data = {'TR'}
        response = self.client.post('/api/covid19/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK, " not success status response code")
        print("Test completed")

    def test_unvalid_length_search(self):
        """
        Ensure we could not search a unvalid country
        """
        data = {'TRZ'}                    # valid country code length should be 2
        response = self.client.post('/api/covid19/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK , " not success status response code")
        print("Test completed")


    def test_unvalid_country_search(self):
        """
        Ensure we could not search a unvalid country
        """
        data = {'ZZ'}     # no any country code with ZZ
        response = self.client.post('/api/covid19/', data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK , " not success status response code")
        print("Test completed")

    def test_web_service(self):
        """
        Ensure we have web connection to take data from source
        """
        response=self.client.get('/api/covid19/')
        self.assertEqual(response.status_code, status.HTTP_200_OK, " not success status response code")
        print("Test completed")
```

- **API Documentation**

Covid19

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/covid19/
http://localhost:8000/covid19/ **(If you are running the app locally)**

Covid19 Case Reports

This endpoint is used for searching covid19 case reports all over the world. Returns a total confirmed case,deaths and recoveries of the world , rankings of confirmed cases using connection to the CovidPy API , also displays a search form for a GET request. For a POST request, it connects to the Covid API and it retrieves specific country data. Using the countrycode from the form, it retrieves confirmed,death and recovery data of the specific country and passes it to the Django template 'covid_country_report.html' where data is displayed to the user.

'covid_reports.html' where data is taken, used, processed and shown to the user 'covid_country_report.html' wher data processed and shown to the user

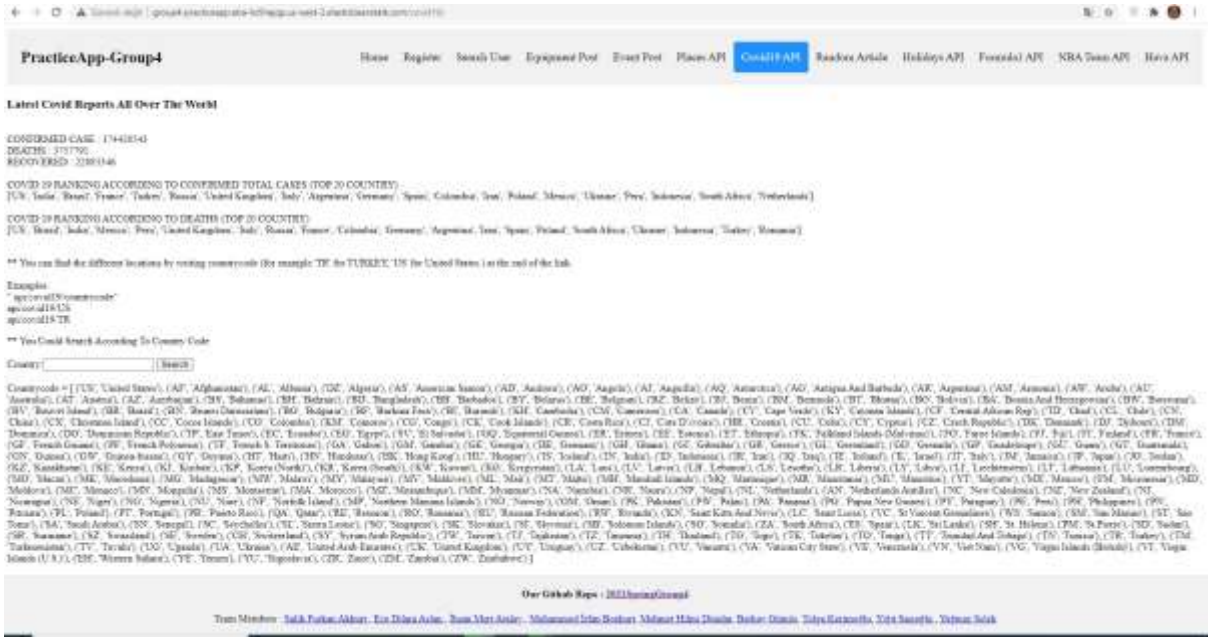
```
'GET':
    Returns the html page for the case reports (confirmed, death, recovered) all over the world,
    Returns the html page for top 20 rankings according to confirmed cases or deaths
    Displays a search form
'POST':
    Using the country code information , provided by the user, it connects to the CovidPy API,
    to take the country data. Retrieves the data and passes it to the Django template "covid_country_report.html"
    where the data displayed to the user.
    JSON Format : { 'countrycode': "", string, identifies the country code user search for it }
```

@author: Yiğit Sanoğlu

GET Request

http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/covid19/
http://localhost:8000/covid19/ **(If you have the application running on your local)**

Returns an HTML page that contains covid19 data from the world and rankings.



POST Requests

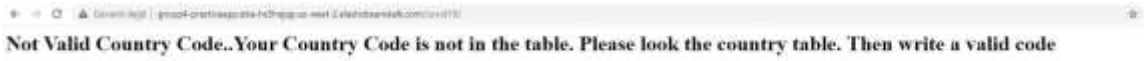
http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/covid19/
http://localhost:8000/covid19/ **(If you have the application running on your local)**

1. **Valid Request**

   For example user searches : "TR"

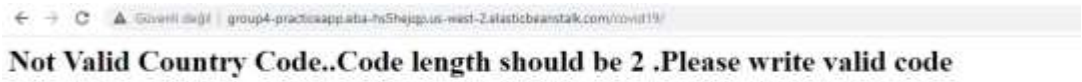
2. **Invalid Request Example (User searches Invalid Country Code)**

   For example user searches : "ZZ"



3. **Invalid Request (User searches Invalid Country Code)**

   For example user searches : " DDDD"



Weather Forecast Apı

Current Weather Condition of Istanbul

This project return the Current Weather Condition of Istanbul. This API get Temparature, Weather condution for example suny,rainy,clear sky ... and small image about weather condution from cities name.

URL: *to be added*

'GET' or 'POST':

Returns the Current Weather condition of Istanbul.

RESPONSE STATUS CODES

GET:

HTTP_400_BAD_REQUEST : It Can't return correctly

HTTP_200_OK : Successfully returns the Current Weather Condition of Istanbul.

@author: İHSAN MERT ATALAY

URL: http://group4-practiceapp.eba-hs5hejqp.us-west-2.elasticbeanstalk.com/hava/

Code Documentation

- The Main Script

```python
import requests
from django.shortcuts import render

# Create your views here.

def hava_api(request):

    url = 'http://api.openweathermap.org/data/2.5/weather?q={}&appid=70e1a9e29891d1988a4d585410a4a618'
    city = 'Istanbul'
    r = requests.get(url.format(city)).json()

    city_weather = {
        'city' : city,
        'temperature' : round( r['main']['temp'] -273,1) ,
        'description' : r['weather'][0]['description'],
        'icon': r['weather'][0]['icon'],
    }
    print(city_weather)
    context = {'city_weather' : city_weather}

    return  render(request,'hava.html ',context)
```

TERMINAL PROBLEMS 16 OUTPUT DEBUG CONSOLE

```
Quit the server with CTRL-BREAK.
C:\Users\ASUS\projects\hava\hava1\urls.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 05, 2021 - 17:43:44
Django version 2.2.13, using settings 'hava.settings'
Starting development server at http://127.0.0.1:8000/
```

What's the weather like?

Istanbul
25.5 C
clear sky