How to use this api?

First install a fresh virtual environment and activate it.

Then open the folder named task in the assignment folder with your code editor.

Enter the code below.

$pip install -r requirements.txt

All necessary packages have been installed. Let's do the migrations

$python .\manage.py makemigrations

$python .\manage.py migrate

And create user. (Use real email when signing up to test Celery implementation)

$python manage.py createsuperuser

Looks like everything is ready, let's get our api up.

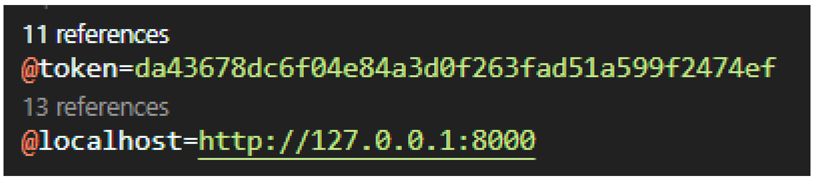
$python manage.py runserver

I wrote a file called test.http that contains all the unit tests.

To use it, install the extension named rest client in vs code.

Then we can start our unit tests.

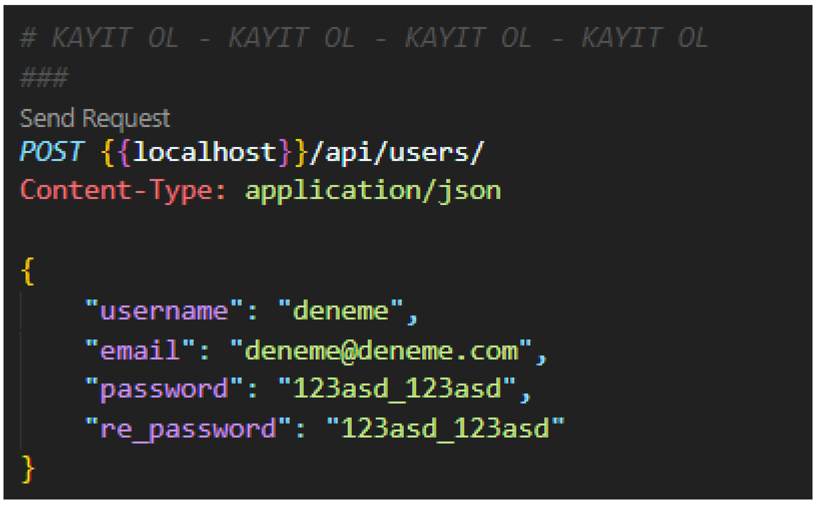
At the very beginning of the unit test file, there are two variables, where localhost is the local ip and port where you run the project, and the token is the token returned to the user after login.



```
11 references
@token=da43678dc6f04e84a3d0f263fad51a599f2474ef
13 references
@localhost=http://127.0.0.1:8000
```

All services have an IsAuthenticated check, so we have to login first and get our token.

To register as a user in the system

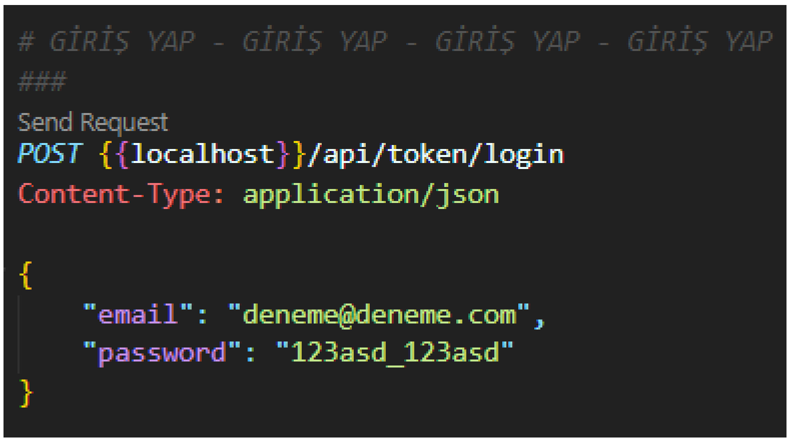


```
# KAYIT OL - KAYIT OL - KAYIT OL - KAYIT OL
###
Send Request
POST {{localhost}}/api/users/
Content-Type: application/json

{
    "username": "deneme",
    "email": "deneme@deneme.com",
    "password": "123asd_123asd",
    "re_password": "123asd_123asd"
}
```

CAUTION! PLEASE REGISTER WITH YOUR REAL EMAIL ADDRESS FOR CELERY TESTS!

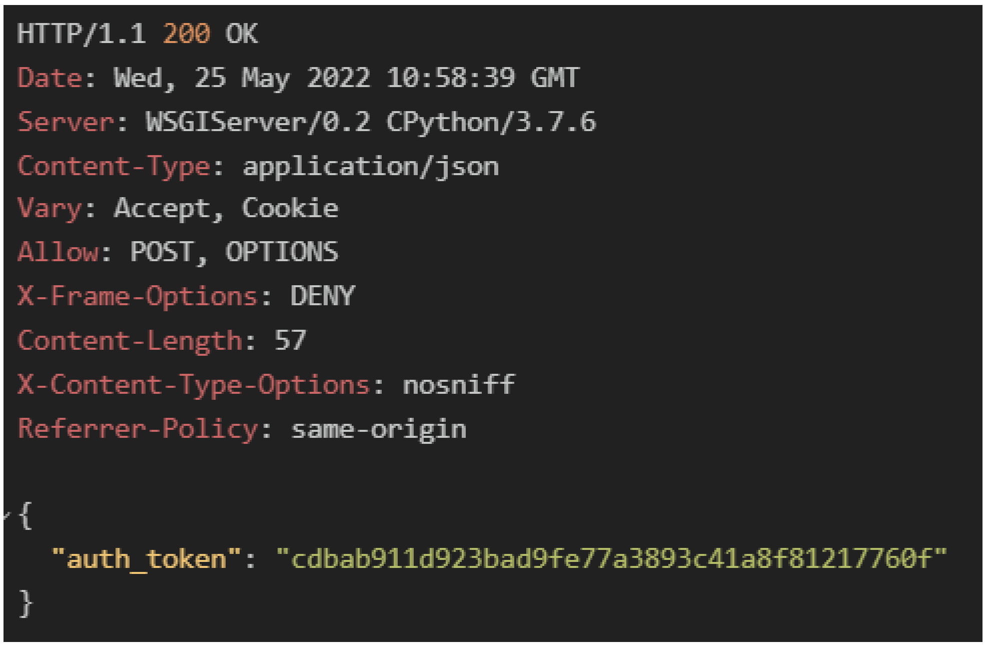
When we click on the Send request button, the service will run. If successful, it will return data with a 201 created response.

To log in and get the token



```
# GİRİŞ YAP - GİRİŞ YAP - GİRİŞ YAP - GİRİŞ YAP
###
Send Request
POST {{localhost}}/api/token/login
Content-Type: application/json

{
    "email": "deneme@deneme.com",
    "password": "123asd_123asd"
}
```

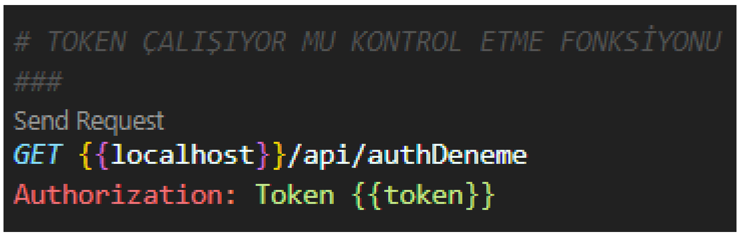
After clicking send request, the response will be like this:



```
HTTP/1.1 200 OK
Date: Wed, 25 May 2022 10:58:39 GMT
Server: WSGIServer/0.2 CPython/3.7.6
Content-Type: application/json
Vary: Accept, Cookie
Allow: POST, OPTIONS
X-Frame-Options: DENY
Content-Length: 57
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin

{
  "auth_token": "cdbab911d923bad9fe77a3893c41a8f81217760f"
}
```

We need to copy this token and assign it to the token variable in the first line.

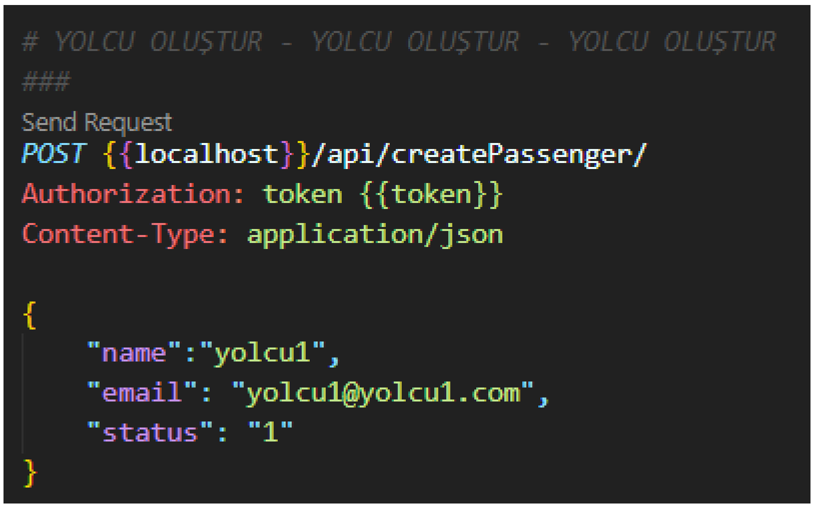
Token test service



```
# TOKEN ÇALIŞIYOR MU KONTROL ETME FONKSİYONU
###
Send Request
GET {{localhost}}/api/authDeneme
Authorization: Token {{token}}
```

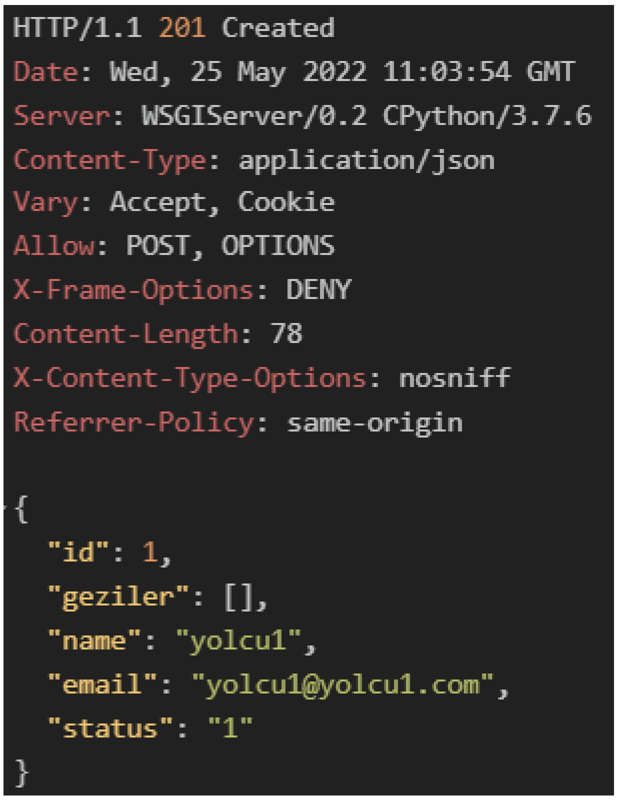
If everything is correct it should read “TOKEN BAŞARILI BİR ŞEKİLDE ÇALIŞIYOR”, if the token is missing or incorrect it will print “geçersiz simge”.

The database is empty, let's create a passenger first.



```
# YOLCU OLUŞTUR - YOLCU OLUŞTUR - YOLCU OLUŞTUR
###
Send Request
POST {{localhost}}/api/createPassenger/
Authorization: token {{token}}
Content-Type: application/json

{
    "name":"yolcu1",
    "email": "yolcu1@yolcu1.com",
    "status": "1"
}
```

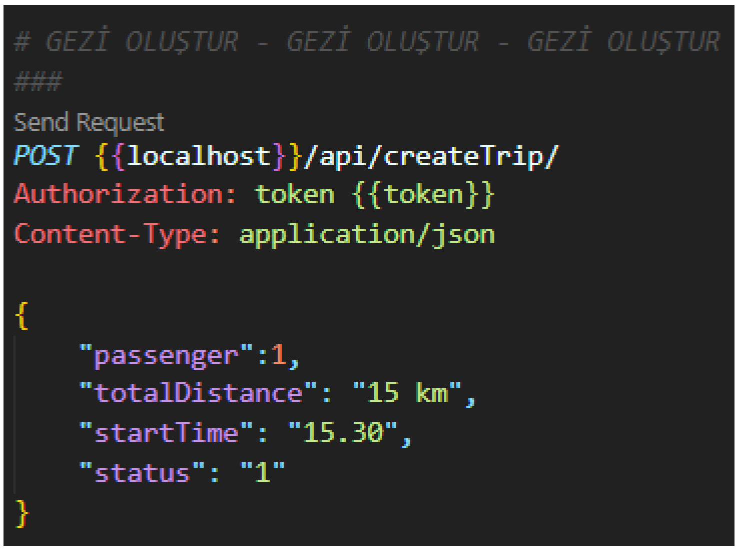
After clicking Send Request, the response will be like this:



```
HTTP/1.1 201 Created
Date: Wed, 25 May 2022 11:03:54 GMT
Server: WSGIServer/0.2 CPython/3.7.6
Content-Type: application/json
Vary: Accept, Cookie
Allow: POST, OPTIONS
X-Frame-Options: DENY
Content-Length: 78
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin

{
  "id": 1,
  "geziler": [],
  "name": "yolcu1",
  "email": "yolcu1@yolcu1.com",
  "status": "1"
}
```

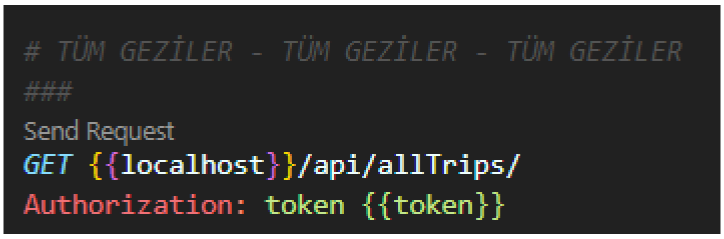
The id is important for us here, since trips and passengers tables are connected with a foreign key, we cannot insert a trip without this id information.

Now let's add a trip for the user whose id is 1.



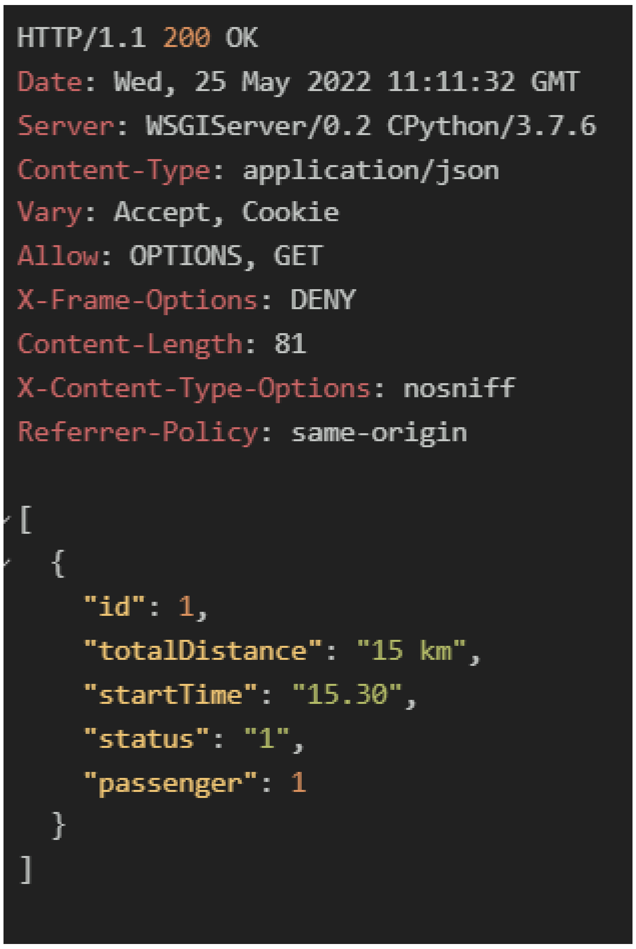
```
# GEZİ OLUŞTUR - GEZİ OLUŞTUR - GEZİ OLUŞTUR
###
Send Request
POST {{localhost}}/api/createTrip/
Authorization: token {{token}}
Content-Type: application/json

{
    "passenger":1,
    "totalDistance": "15 km",
    "startTime": "15.30",
    "status": "1"
}
```

Now let's look at the data in the trips table with the allTrips service. , the information we have inserted should appear.



```
# TÜM GEZİLER - TÜM GEZİLER - TÜM GEZİLER
###
Send Request
GET {{localhost}}/api/allTrips/
Authorization: token {{token}}
```

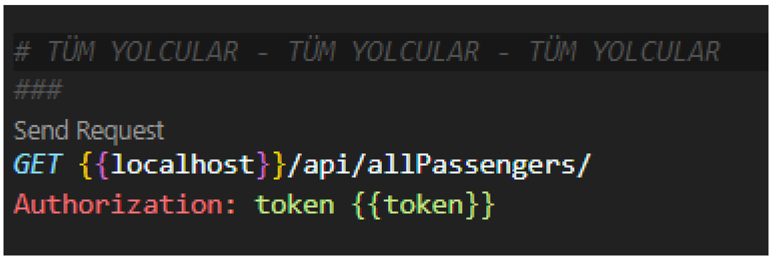
When we throw the request, the response is as follows



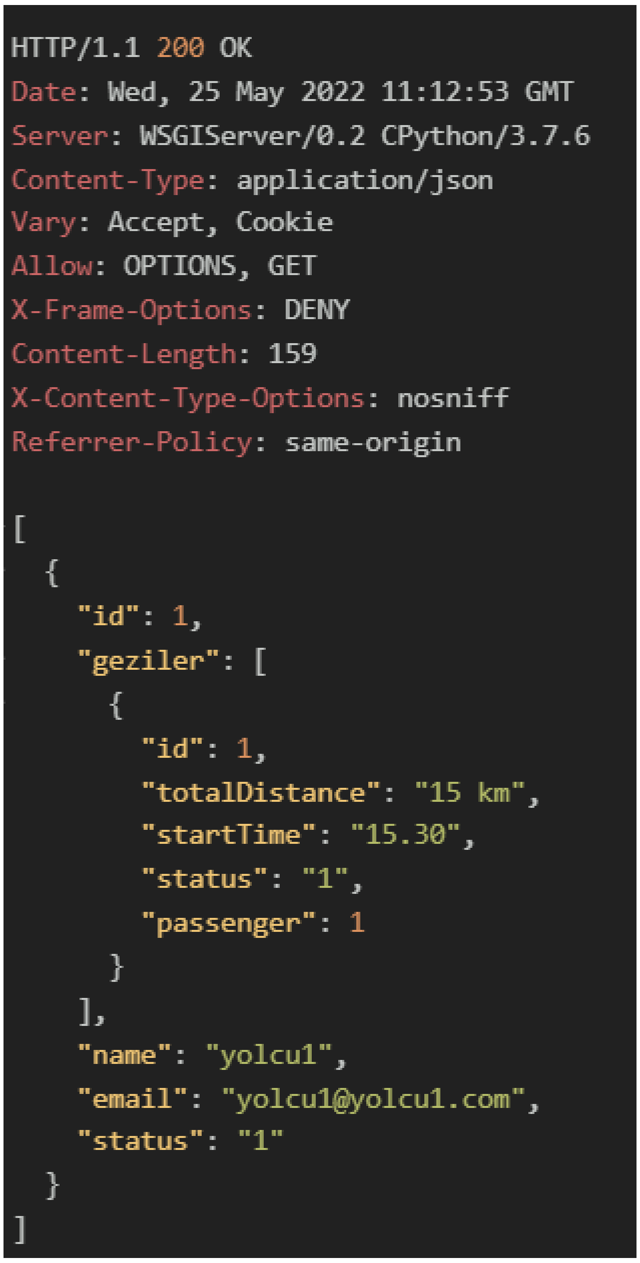
```
HTTP/1.1 200 OK
Date: Wed, 25 May 2022 11:11:32 GMT
Server: WSGIServer/0.2 CPython/3.7.6
Content-Type: application/json
Vary: Accept, Cookie
Allow: OPTIONS, GET
X-Frame-Options: DENY
Content-Length: 81
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin

[
  {
    "id": 1,
    "totalDistance": "15 km",
    "startTime": "15.30",
    "status": "1",
    "passenger": 1
  }
]
```

Here id is the trip id, and passenger holds the passenger id information.

Let's call the passengers to check if the foreign key is working.



```
# TÜM YOLCULAR - TÜM YOLCULAR - TÜM YOLCULAR
###
Send Request
GET {{localhost}}/api/allPassengers/
Authorization: token {{token}}
```

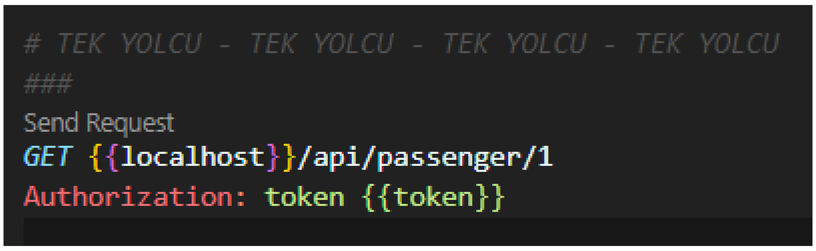


```
HTTP/1.1 200 OK
Date: Wed, 25 May 2022 11:12:53 GMT
Server: WSGIServer/0.2 CPython/3.7.6
Content-Type: application/json
Vary: Accept, Cookie
Allow: OPTIONS, GET
X-Frame-Options: DENY
Content-Length: 159
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin

[
  {
    "id": 1,
    "geziler": [
      {
        "id": 1,
        "totalDistance": "15 km",
        "startTime": "15.30",
        "status": "1",
        "passenger": 1
      }
    ],
    "name": "yolcu1",
    "email": "yolcu1@yolcu1.com",
    "status": "1"
  }
]
```

As we can see everything works fine.

Let me explain the other services one by one.

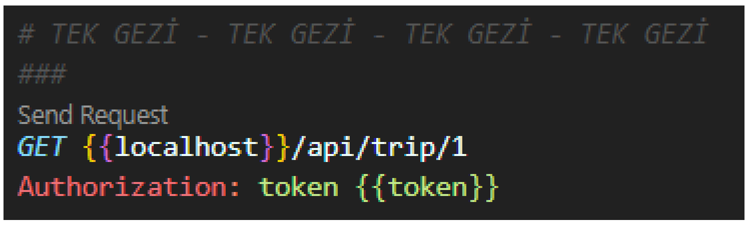
To fetch the information of a single passenger, the passenger ID is sent to the following service:



```
# TEK YOLCU - TEK YOLCU - TEK YOLCU - TEK YOLCU
###
Send Request
GET {{localhost}}/api/passenger/1
Authorization: token {{token}}
```

Passenger information with 1 id returns as response

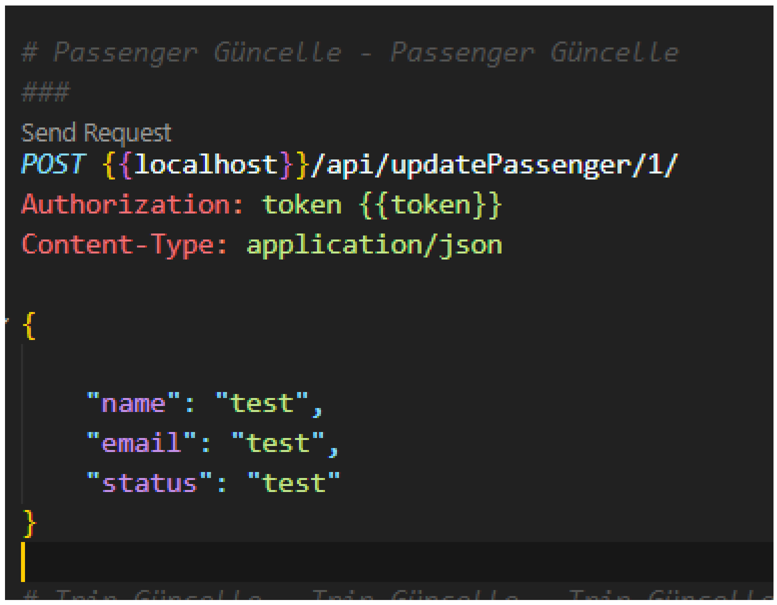
For a single travel information, let's send the travel id to the following service:



```
# TEK GEZİ - TEK GEZİ - TEK GEZİ - TEK GEZİ
###
Send Request
GET {{localhost}}/api/trip/1
Authorization: token {{token}}
```

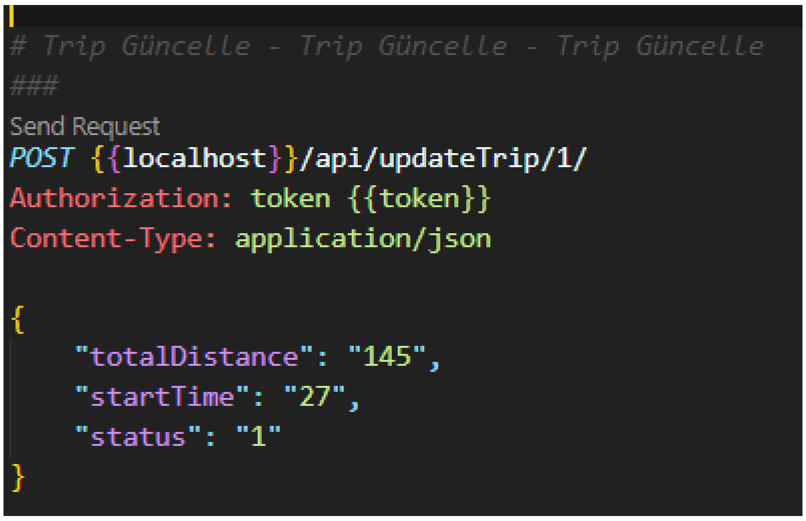
The information of the trip with 1 id is returned to the user as a response.

Let's update the information of Passenger whose id is 1:



```
# Passenger Güncelle - Passenger Güncelle
###
Send Request
POST {{localhost}}/api/updatePassenger/1/
Authorization: token {{token}}
Content-Type: application/json

{

    "name": "test",
    "email": "test",
    "status": "test"
}

# Trip Güncelle - Trip Güncelle - Trip Güncelle
```

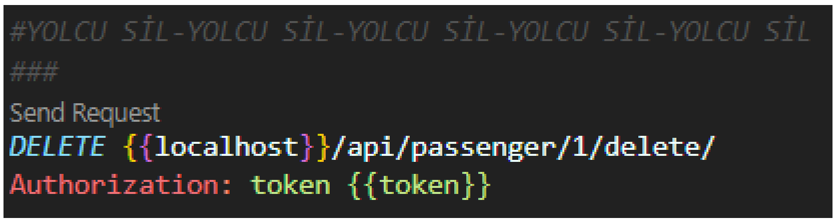
Let's update the information of the trip with an id of 1:



```
# Trip Güncelle - Trip Güncelle - Trip Güncelle
###
Send Request
POST {{localhost}}/api/updateTrip/1/
Authorization: token {{token}}
Content-Type: application/json

{
    "totalDistance": "145",
    "startTime": "27",
    "status": "1"
}
```

You can check that the update services are running correctly by going to /api/allPassengers/.

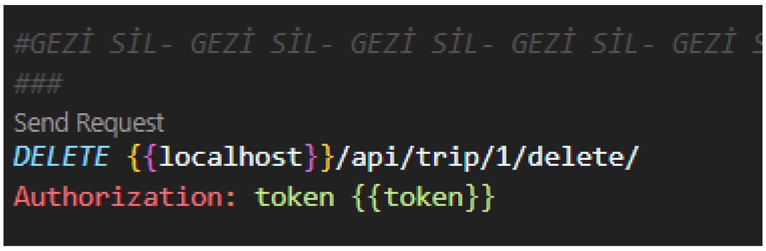
To delete everything about the passenger with id 1:



```
#YOLCU SİL-YOLCU SİL-YOLCU SİL-YOLCU SİL-YOLCU SİL
###
Send Request
DELETE {{localhost}}/api/passenger/1/delete/
Authorization: token {{token}}
```

It deletes the passenger with id 1 and their trip information from the database.

To delete a trip with id=1:



```
#GEZİ SİL- GEZİ SİL- GEZİ SİL- GEZİ SİL- GEZİ S
###
Send Request
DELETE {{localhost}}/api/trip/1/delete/
Authorization: token {{token}}
```

Deletes the trip with id=1.

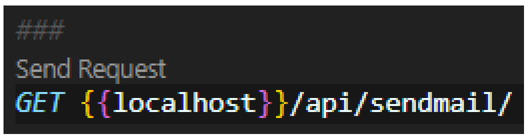
I used djoser which is a 3rd party application for middleware, it is very useful and safe.

Djoser is a REST implementation of Django authentication system. **djoser** library provides a set of Django Rest Framework views to handle basic actions such as registration, login, logout, password reset and account activation. It works with a custom user model.

As you know, you need to run redis service first for celery. I started the service using wsl on windows, this process is easier on linux devices.

$sudo apt-add-repository ppa:redislabs/redis
$sudo apt-get update
$sudo apt-get upgrade
$sudo apt-get install redis-server
$sudo service redis-server start

After Redis is up, we send a get request to the following link:



```
###
Send Request
GET {{localhost}}/api/sendmail/
```

The algorithm I wrote here will send all the travel information in the database to the e-mail address you registered at the beginning and to the e-mail addresses of the users who registered later.

In case of no mail, you can stop the service and enter the code below to access the celery logs.

$celery -A proje.celery worker --pool=solo --loglevel=info