

BURSA TEKNİK ÜNİVERSİTESİ

**Mühendislik ve Doğa Bilimleri Fakültesi – Bilgisayar
Mühendisliği Bölümü**



BLM0470 NodeJS İle Web Programlama

2023-2024 Bahar Dönemi

5.Hafta Raporu

Berkan SERBES – 22360859353

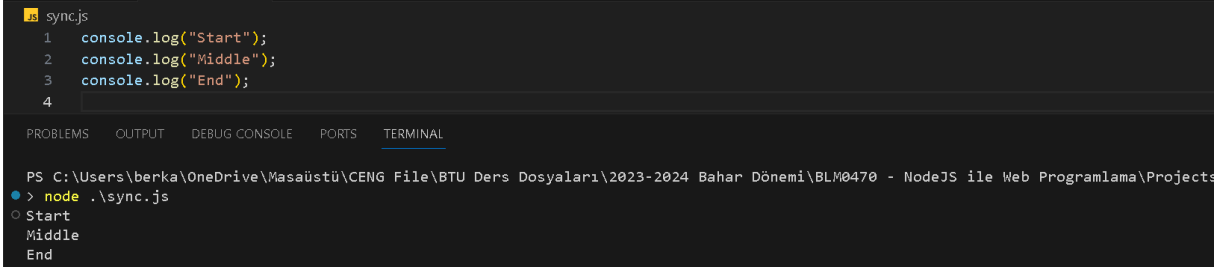
İçindekiler

1.Senkron Programlama	3
2.Asenkron Programlama	3
2.1 Asenkron Kod Örnekleri	4
3.API Nedir?.....	6
4. Hava Durumu Uygulaması Geliştirme	7
5.Kaynakça	17

1.Senkron Programlama

Senkron programlama, işlemlerin sırayla ve adım adım gerçekleştirildiği bir programlama yaklaşımıdır. Bu yaklaşımda, bir işlem tamamlanmadan diğerine geçilmez ve her işlem bir sonraki işlemin başlaması için bekler. Senkron programlama modeli, işlemlerin ardışık bir şekilde yürütülmesini sağlar.

NodeJS'te senkron programlama, varsayılan olarak işlemlerin senkron olarak yürütülmesine dayanır. Bu, işlemlerin birbirini takip eden sıralarda gerçekleştiği anlamına gelir.



```
1 console.log("Start");
2 console.log("Middle");
3 console.log("End");
4
```

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects> node .\sync.js

Start
Middle
End

Yukarıdaki görselde yer alan kod parçası, sırayla üç farklı mesajı konsola yazdırır.

İlk olarak, "Start" mesajı konsola yazdırılır. Daha sonra, "Middle" mesajı yazdırılır. Son olarak, "End" mesajı konsola yazdırılır. Bu kod, her bir console.log() ifadesinin ardışık olarak çalıştığı ve programın sona erdiği basit bir senkron programı temsil eder. Her bir console.log() ifadesi, önceki ifadenin tamamlanmasını bekler ve sırayla çalışır.

2.Asenkron Programlama

Asenkron programlama, bir işlemin tamamlanmasını beklemeksizin, programın diğer kısımlarının çalışmaya devam etmesine izin veren bir programlama modelidir. Bu modelde, işlemler sırayla ve senkronize bir şekilde yürütülmez; bunun yerine, bir işlem başlatıldığında diğer işlemlerin aynı anda yürütülmesine izin verilir. Asenkron programlama, özellikle uzun süren işlemler, ağ istekleri veya dosya okuma/yazma gibi girdi/çıkı (I/O) operasyonları için yaygın olarak kullanılır çünkü işlem sırasında diğer işlemler devam eder ve beklenmez.

NodeJS, asenkron programlama modelini varsayılan olarak benimser ve bu model, JavaScript dilinde bulunan callback, Promise ve async/await gibi mekanizmalarla desteklenir. Bu mekanizmalar, asenkron işlemlerin yönetilmesini sağlar.

Gerçek dünya örneklerinden biri, web tarayıcılarının asenkron olarak çalışma şeklidir. Bir web tarayıcısı, kullanıcının bir web sayfasını yüklediğinde, birden fazla kaynağı (HTML dosyaları, CSS stilleri, JavaScript dosyaları, görseller, vs.) eşzamanlı olarak indirir ve işler. Tarayıcı, bu kaynakları tek tek indirmek yerine, aynı anda birden fazla isteği gönderir ve bu kaynakların indirilmesini beklerken diğer işlemlere devam eder. Böylece, bir kaynağın indirilmesi tamamlanmadan diğer kaynakların indirilmesini beklemek zorunda kalmaz.

Asenkron programlamayı Formula 1 yarışlarındaki pit stop örneğiyle de açıklayabiliriz. Formula 1 yarışlarında pit stoplar araçların lastik değişimi, yakıt ikmali ve gerektiğinde araç ayarlarının yapılması gibi önemli bakım işlemlerinin yapıldığı duraklardır. Bu süreç, yarışı etkileyen kritik bir unsurdur ve hızlı bir şekilde gerçekleştirilmelidir.

Asenkron programlama modeliyle pit stop örneğini anlamak için şu adımları düşünebiliriz:

Pit Stop İsteği: Araçlar yarış sırasında pit stop ihtiyacı olduğunda, takım bu isteği iletişim cihazları aracılığıyla alır. Bu istek, aracın yarışa devam ederken yapılan bir işlem olup, aracın hızını etkilemez.

Eşzamanlı İşlemler: Pit stop sırasında, lastik değişimi, yakıt ikmali ve gerekli ayarlamalar eşzamanlı olarak gerçekleştirilir. Yani, bir ekip lastikleri değiştirirken diğer ekip yakıt ikmalini yapabilir ve bir başka ekip araç ayarlarını kontrol edebilir.

Bekleme Süresi Yoktur: Asenkron programlama modelinde, bir işlem diğerinin tamamlanmasını beklemek zorunda değildir. Yani, lastik değişimi tamamlanmadan yakıt ikmali yapılmaya başlanabilir ve araç ayarları kontrol edilirken diğer işlemler devam edebilir.

Geri Bildirim: Pit stop tamamlandığında, araç sürücüsüne veya takım yönetimine bir geri bildirim verilir. Bu geri bildirim, pit stop işleminin başarıyla tamamlandığını veya herhangi bir sorun olduğunu belirtir.

Bu örnekte, pit stoplar asenkron programlama modelinin bir örneğini oluşturur. İşlemler eşzamanlı olarak gerçekleştirilir ve bir işlemin tamamlanmasını diğerinin beklemesi gerekmez. Bu sayede, yarış takımları araçlarını mümkün olan en kısa sürede bakıma alabilir ve yarışa devam edebilirler, bu da yarışın genel performansını artırır.

2.1 Asenkron Kod Örnekleri

Aşağıdaki görselde yer alan kod bloğu, asenkron programlamanın bir örneğini göstermektedir. İlk olarak, "Start" mesajı ekrana yazdırılır. Daha sonra, setTimeout fonksiyonu kullanılarak bir zamanlayıcı başlatılır. Asenkron programlama modelinde, setTimeout fonksiyonu bir zamanlayıcı başlatır ve belirtilen süre sonunda bir işlemi yürütür. Ancak, zamanlayıcı başlatıldıktan sonra diğer kodlar hemen çalışmaya devam eder. Yani, "End" mesajı, zamanlayıcı çalışana kadar beklemeden hemen ekrana yazdırılır.

```
JS async.js > ...
1 console.log("Start");
2
3 setTimeout(() => {
4   console.log("0ms timeout");
5 }, 0);
6
7 console.log("End");
8
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS
> node .\async.js
Start
End
0ms timeout
```

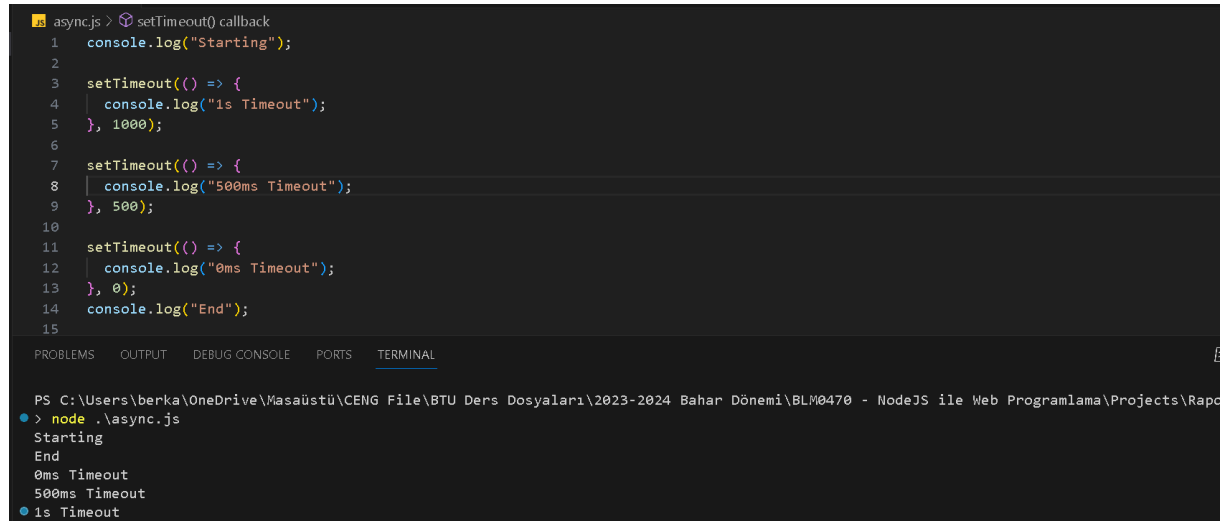
Şekil 1: Asenkron Kod Örneği 1

Şekil 2 de yer alan kod parçasında, öncelikle senkron işlemler (synchronous tasks) yürütülür. Bu işlemler, "Starting" ve "End" mesajlarını ekrana yazdırmak gibi zamanında tamamlanabilen görevlerdir. Daha sonra asenkron görevler (asynchronous tasks) sırayla yürütülür.

İlk olarak, zamanlayıcılar (setTimeout) ayarlanır. Bu zamanlayıcılar, belirli bir süre sonra tetiklenecek olan görevleri içerir. Örneğin, 1 saniye sonra tetiklenecek bir zamanlayıcı "1s Timeout" mesajını ekrana yazdıracak bir işlemi içerir.

Asenkron görevlerin yürütülmesi, Event Loop tarafından kontrol edilir. Event loopta iki farklı kuyruk bulunur bunlar mikro görev kuyruğu(micro task queue) ve makro görev kuyruğu(macro task queue). setTimeout, setInterval gibi makro görevler, macrotask queue'ye eklenirken, Promise.then, process.nextTick gibi işlemler mikro görev kuyruğuna eklenir. Event Loop, önce mikro görev kuyruğundaki (micro task queue) tüm mikro görevleri işler. Bu durumda, mikro görevlerimiz yoktur, çünkü zamanlayıcılar bir mikro görev değildir. Ardından, Event Loop, makro görev kuyruğundaki (macro task queue) zamanlayıcıları işler. Her bir zamanlayıcı belirli bir süre sonra tetiklenecek ve ilgili işlemi yürütecektir.

Bu nedenle, çıktıda "Starting" ve "End" mesajları hemen ekrana yazdırılır, çünkü bunlar senkron görevlerdir. Daha sonra, zamanlayıcılar belirlenen sürelerin ardından tetiklenir ve "0ms Timeout", "500ms Timeout" ve "1s Timeout" mesajları sırasıyla ekrana yazdırılır.



```
1 console.log("Starting");
2
3 setTimeout(() => {
4   console.log("1s Timeout");
5 }, 1000);
6
7 setTimeout(() => {
8   console.log("500ms Timeout");
9 }, 500);
10
11 setTimeout(() => {
12   console.log("0ms Timeout");
13 }, 0);
14 console.log("End");
15
```

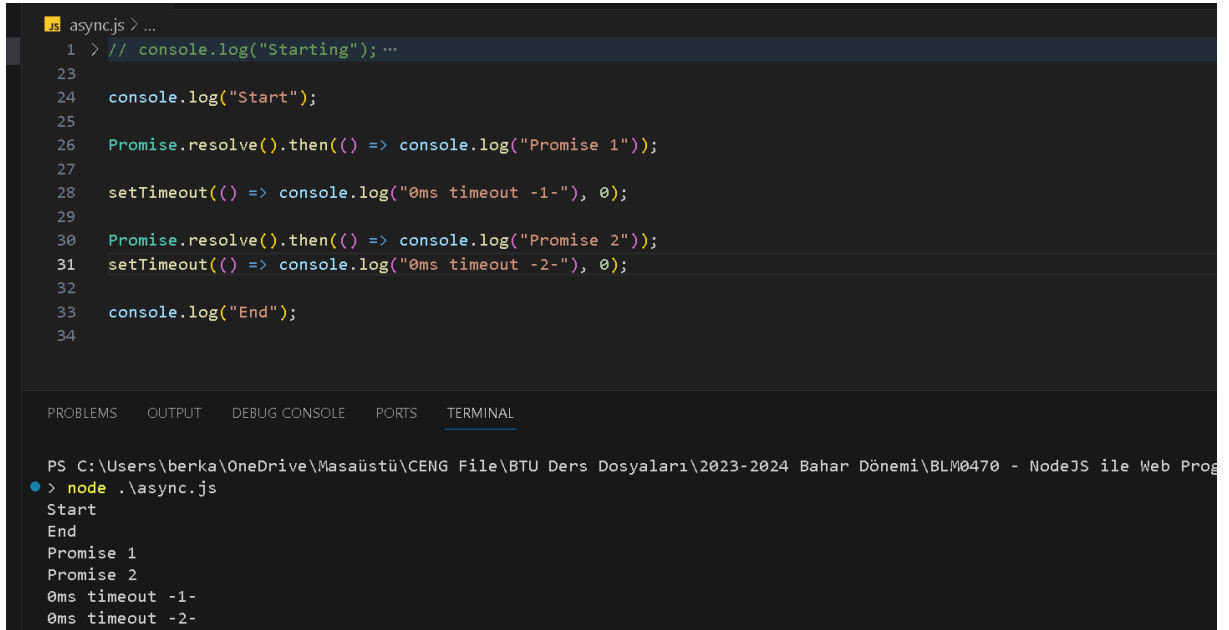
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\Rapc
> node .\async.js
Starting
End
0ms Timeout
500ms Timeout
1s Timeout
```

Şekil 2: Asenkron Kod Örneği 2

Şekil 3'te yer alan kod parçasında, öncelikle senkron işlemler (synchronous tasks) olan "Start" ve "End" ifadeleri ekrana yazdırılır. Ardından asenkron görevler (asynchronous tasks) sırayla yürütülür. Asenkron görevlerin içinde, öncelikle mikro görev kuyruğunda yer alan promise tabanlı kodlar (Promise 1 ve Promise 2) işlenir. Daha sonra, makro görev kuyruğunda bulunan zamanlayıcı olaylar (timer events) sırayla işlenir. Başlangıçta, Event Loop senkron görevleri çağrı yığınının (call stack) iter ve tüm senkron görevler tamamlandığında, çağrı yığını boşaltılır ve senkron görevlerin tamamı yürütülür. Ardından Event Loop, öncelikle mikro görev kuyruğundaki tüm mikro görevleri (Promise 1 ve Promise 2) tamamlamaya odaklanır ve sonra makro görev kuyruğundaki zamanlayıcı olayları işler. Sonuç olarak, çıktıda "Start" ve "End" ifadeleri hemen ekrana yazdırılır çünkü bunlar senkron işlemlerdir. Daha sonra,

mikro görevler (Promise 1 ve Promise 2) sırasıyla işlenir ve son olarak makro görevler (0ms timeout -1- ve 0ms timeout -2-) sırayla işlenir ve ekrana yazdırılır.



```
1 > // console.log("Starting"); ...
23
24 console.log("Start");
25
26 Promise.resolve().then(() => console.log("Promise 1"));
27
28 setTimeout(() => console.log("0ms timeout -1-"), 0);
29
30 Promise.resolve().then(() => console.log("Promise 2"));
31 setTimeout(() => console.log("0ms timeout -2-"), 0);
32
33 console.log("End");
34
```

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Prog
● > node .\async.js
Start
End
Promise 1
Promise 2
0ms timeout -1-
0ms timeout -2-

Şekil 3: Asenkron Kod Örneği

3.API Nedir?

API (Application Programming Interface), yazılım uygulamalarının ve sistemlerin birbiriyle iletişim kurmasını sağlayan bir arayüzdür. API'ler, uygulamaların ve sistemlerin birbirleriyle veri alışverişi yapmasını, işlemleri gerçekleştirmesini veya hizmetlere erişmesini sağlar.

API'ler, genellikle iki yazılım bileşeni arasındaki iletişimi kolaylaştırır. Bir yazılım bileşeni, başka bir bileşene belirli bir işlem yapması için bir istekte bulunur ve API, bu isteği alır, işler, gerekli işlemleri gerçekleştirir ve sonucu gönderir. Bu işlem sırasında, API kullanıcıları belirli bir protokole veya kuralla uygun olarak API'ye istekler gönderir ve API, bu isteklere uygun olarak yanıtlar verir.

API (Application Programming Interface), günlük hayatta sıkça karşılaştığımız bir kavramdır ve bir restoran analogisi ile açıklayabiliriz. Bir restoranda, müşteriler ve garsonlar arasındaki etkileşim, bir tür API'ye benzetilebilir. Müşteriler, menüdeki yemekleri ve içecekleri seçerler ve bu siparişleri garsona iletmek için menüyü kullanırlar. Garson, müşterinin taleplerini alır, mutfağa iletir, yemeği hazırlatır ve sonunda müşteriye sunar.

İşte bu noktada, müşteriler ve garson arasındaki ilişki, bir API'ye benzetilebilir. Müşteriler, garsona belirli bir "istek" veya "sipariş" iletmek için menüyü kullanır. Garson da bu isteği alır, bunu restoranın "hizmetlerine" (mutfak, bar vb.) iletir ve sonuç olarak müşteriye bir "yanıt" veya "cevap" sunar.

Bir başka örnek olarak, telefon rehberimizi düşünebiliriz. Telefon rehberi, telefon numaralarını, kişilerin adlarını ve diğer iletişim bilgilerini saklar. Bir kullanıcı, belirli bir kişinin adını aradığında, telefon rehberi bu bilgileri sağlar. Bu durumda, telefon rehberi bir tür API'ye benzetilebilir. Kullanıcı, belirli bir "istek" (kişinin adı) ile rehberi "sorgular" ve rehber de bu isteği alır, uygun "verileri" bulur ve kullanıcıya "cevap" olarak sunar.

API'ler, yazılım geliştirme sürecinde de aynı mantıkla çalışır. Bir uygulama, başka bir uygulamadan veya hizmetten veri almak veya işlem yapmak istediğinde, API'ler aracılığıyla bu işlemleri gerçekleştirir. Örneğin, bir hava durumu uygulaması, hava durumu verilerine erişmek için bir hava durumu API'sini kullanır. Bu API, uygulamanın hava durumu verilerine "istek" göndermesine ve API'nin bu isteğe uygun "yanıt"ı sunmasına olanak tanır.

4. Hava Durumu Uygulaması Geliştirme

Bu başlık altında, bir hava durumu uygulaması geliştireceğiz. Bu uygulama basitçe kullanacağımız dış bir API servisinden GET istekleri gerçekleştirecek.

İlk olarak weather-api adında bir boş klasör açalım. Klasörü açtıktan sonra editör ekranımızda bu klasör yoluna gidip **npm init -y** diyerek package.json dosyamızı oluşturalım.

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodlari\Hafta5_RaporKodlari\weather-api> npm init -y
Wrote to C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodlari\Hafta5_RaporKodlari\weather-api\package.json:

{
  "name": "weather-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Sonrasında gerekli paketlerimizi yükleyelim. Bu uygulamada kullanacağımız paketler **dotenv** ve **postman-request** paketleridir.

dotenv paketi, uygulama geliştirme sürecinde çevresel değişkenleri yönetmek için kullanılır. Çevresel değişkenler, geliştirme ve dağıtım süreçlerinde kullanılan hassas bilgileri, özellikle gizli anahtarlar veya veritabanı bağlantı bilgileri gibi verileri uygulama kodundan ayırmak için kullanılır. dotenv paketi, .env adında bir dosyada çevresel değişkenleri depolar ve bu değişkenlere kolayca erişim sağlar. Bu sayede, uygulama kodu daha güvenli hale gelir ve çevresel değişkenlerin yönetimi kolaylaşır.

postman-request paketi ise HTTP isteklerini yapmak için kullanılır. Bu paket, Postman uygulamasının özelliklerini kod içine taşır ve kolayca HTTP istekleri oluşturmaya ve yönetmeye olanak tanır. Özellikle, dış API'lere istek yapmak veya sunucudan veri almak için postman-request paketi oldukça kullanışlıdır. Bu paket sayesinde, uygulamanızın dış dünyayla etkileşim kurması ve veri alışverişi yapması sağlanır.

```
weather-api > package.json > {} dependencies > postman-request
1
2 {
3   "name": "weather-api",
4   "version": "1.0.0",
5   "description": "",
6   "main": "index.js",
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "dotenv": "^16.4.5",
15    "postman-request": "^2.88.1-postman.33"
16  }
17 }

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL powershell - weather-api

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları>
.\weather-api> npm i dotenv postman-request
npm WARN deprecated har-validator@5.1.5: this library is no longer supported

added 56 packages, and audited 57 packages in 15s

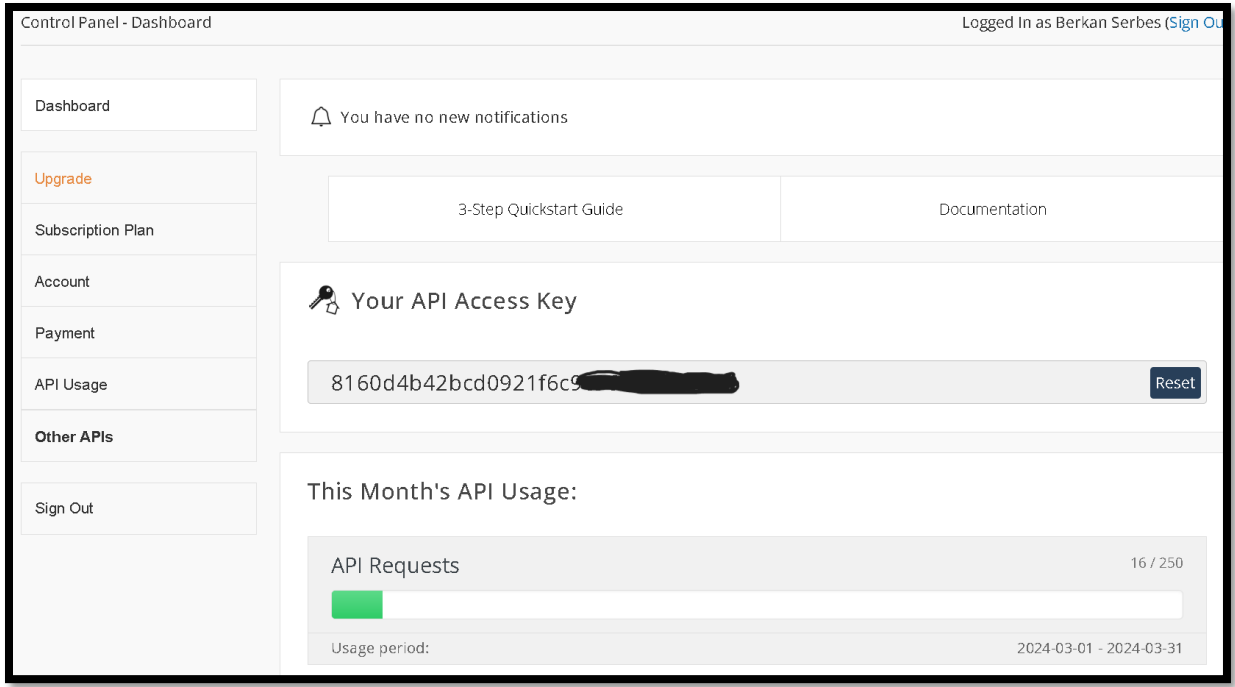
4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Yukarıdaki görseldeki gibi kullanacağımız paketleri **npm i dotenv postman-request** komutuyla toplu bir şekilde yükleyelim.

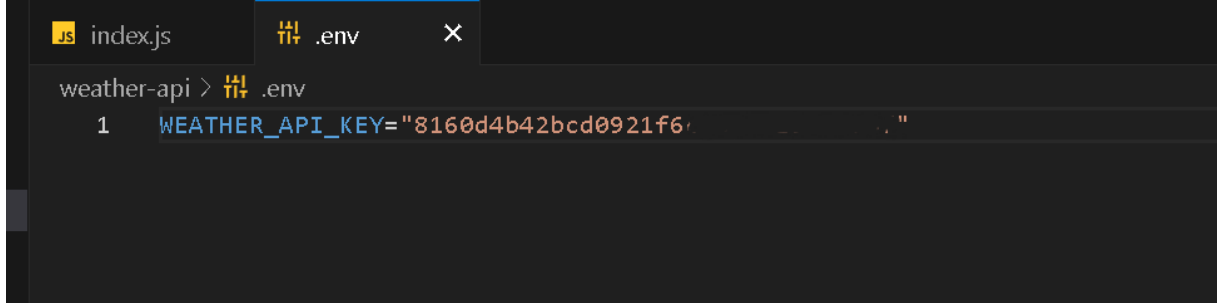
API olarak kullanacağımız siteye ilk olarak üye olmamız gerekmektedir. Yandaki linkten siteye üye olunuz → <https://weatherstack.com/signup/free>

Başarılı bir şekilde üye olduktan sonra kontrol panelinde bulunan API erişim anahtarını uygulamamıza dahil etmemiz gerekmektedir. API erişim anahtarı, API sağlayıcısının sunucularına yapılan isteklerde bir kimlik doğrulama mekanizması olarak kullanılır. Bu anahtar, istek başlıklarında veya URL parametrelerinde bulunabilir. API sağlayıcısı, gelen istekleri bu anahtarlarla kimlik doğrular ve API'ye erişim izni olan isteklere hizmet verir. Bu sayede, API sağlayıcısı istenmeyen erişimlerden korunabilir ve güvenli bir hizmet sunabilir. API erişim anahtarları, genellikle gizli tutulması gereken bilgilerdir çünkü bu anahtarlara sahip olan kişi veya uygulama, API'ye erişim hakkına sahiptir ve bu erişimle işlemler yapabilir. Bu nedenle, API erişim anahtarlarının güvenli bir şekilde saklanması ve paylaşılması önemlidir.



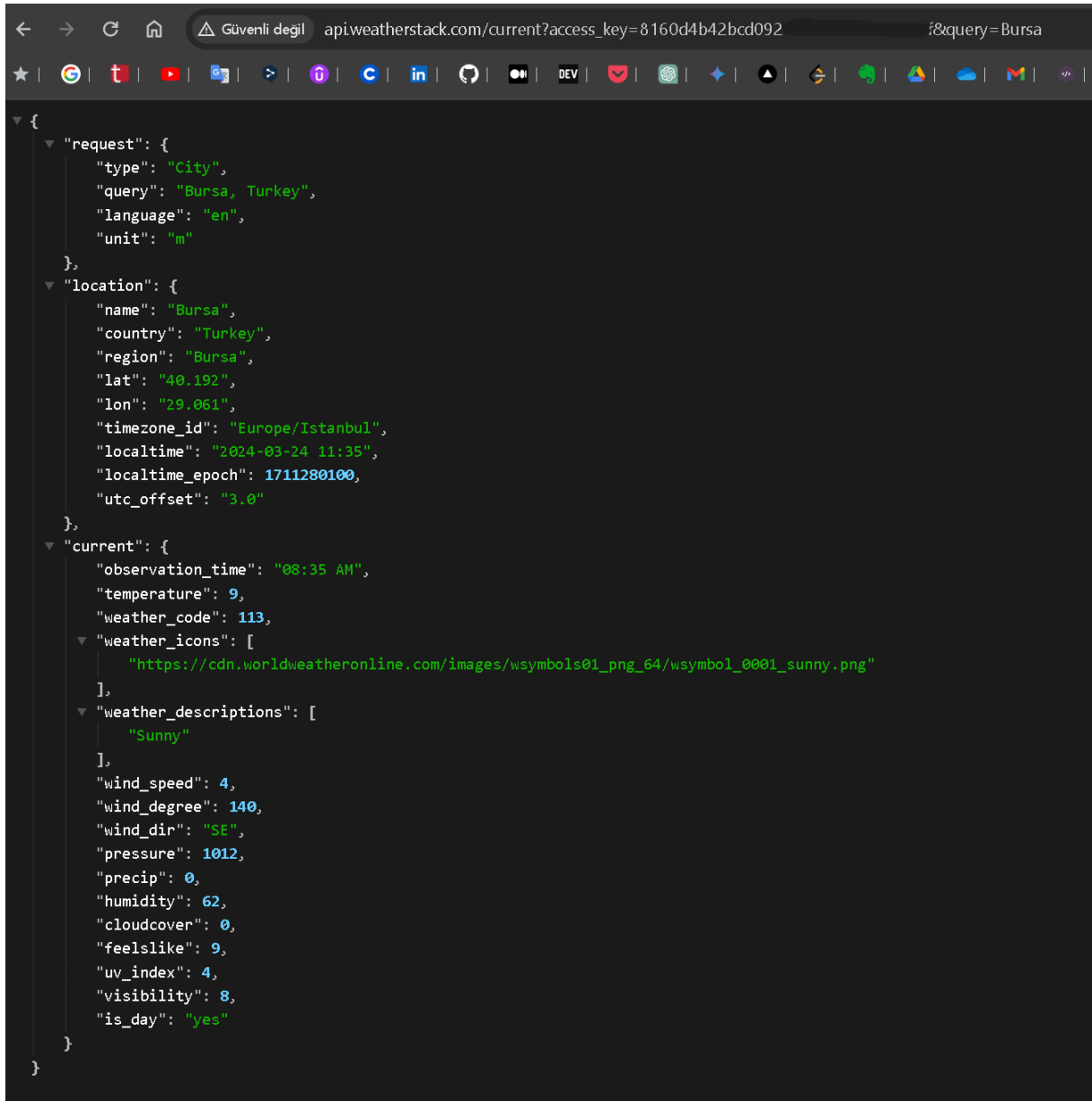
Yukarıdaki görseldeki gibi kendi API erişim anahtarınızı kontrol panelinden edinebilirsiniz. API erişim anahtarını kopyalayalım ve uygulamamıza dahil edelim.

Aşağıdaki görseldeki gibi .env adında bir dosya oluşturalım. Atamanın sağ tarafına kendi API erişim anahtarını yazmanız gerekmektedir.



Bu kod, bir .env dosyasında bulunan ve WEATHER_API_KEY adında bir ortam değişkenine atanan bir değeri içerir. .env dosyaları genellikle bir uygulamanın yapılandırma ayarlarını saklamak için kullanılır ve hassas bilgileri (örneğin API anahtarları gibi) içerebilir. .env dosyasındaki değerler genellikle uygulamanın kodunda process.env nesnesi aracılığıyla erişilir. Örneğin, bu API anahtarına erişmek için JavaScript kodunda **process.env.WEATHER_API_KEY** ifadesi kullanılabilir.

API erişim anahtarın doğru bir şekilde çalışıp çalışmadığını anlamak için tarayıcımız üzerinden bir istekte bulunalım.

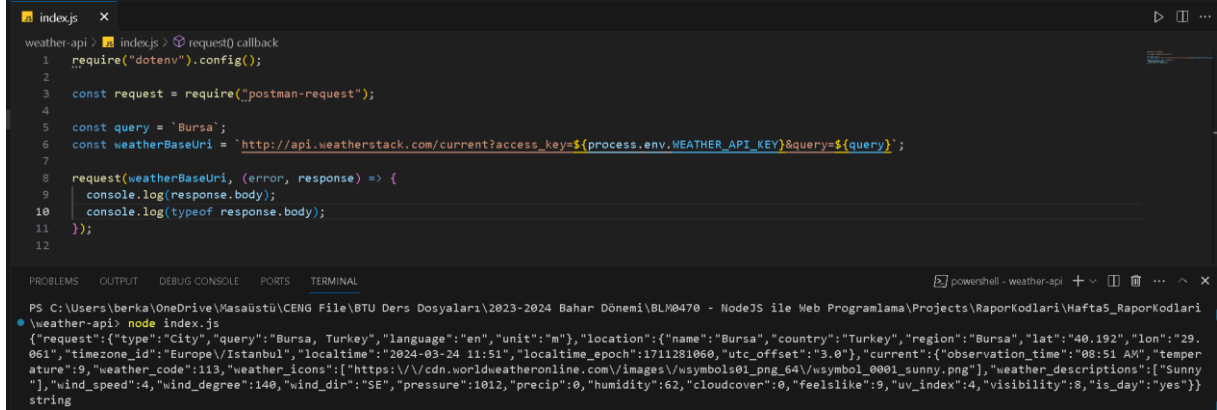


Yukarıda şekilde görüldüğü gibi attığımız istek sonucunda bize bir cevap dönmektedir. Bu, erişim anahtarının ve api servisimizin doğru bir şekilde çalıştığını göstermektedir.

Sırada yapacağımız işlem projemizin ana dosyasını oluşturmaktır. index.js adında bir dosya oluşturalım ve aşağıdaki görselde yer alan kodları yazalım.



Bu kodlar, .env dosyasından çevre değişkenlerini yükler ve HTTP istekleri göndermek için gerekli olan postman-request paketini projeye dahil eder. Bu şekilde, projenin yapılandırmasını ve HTTP isteklerini yapmasını sağlar.



```
index.js X
weather-api > index.js > request() callback
1 require("dotenv").config();
2
3 const request = require("postman-request");
4
5 const query = 'Bursa';
6 const weatherBaseUri = 'http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}';
7
8 request(weatherBaseUri, (error, response) => {
9   console.log(response.body);
10  console.log(typeof response.body);
11 });
12

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta5_RaporKodları
.\weather-api> node index.js
{"request":{"type":"City","query":"Bursa, Turkey","language":"en","unit":"m"},"location":{"name":"Bursa","country":"Turkey","region":"Bursa","lat":"40.192","lon":"29.061","timezone_id":"Europe/Istanbul","localtime":"2024-03-24 11:51","localtime_epoch":1711281060,"utc_offset":"+3.0"},"current":{"observation_time":"08:51 AM","temperature":9,"weather_code":113,"weather_icons":["https://cdn.worldweatheronline.com/images/wsymbols01_png_64/wsymb01_0001_sunny.png"],"weather_descriptions":["Sunny"]},"wind_speed":4,"wind_degree":140,"wind_dir":"SE","pressure":1012,"precip":0,"humidity":62,"cloudcover":0,"feelslike":9,"uv_index":4,"visibility":8,"is_day":"yes"}}
string
```

Bu kod parçası, hava durumu verilerini almak için bir HTTP isteği oluşturur ve bu isteği gönderir. İlk olarak, query değişkeni Bursa şehrinin adını temsil eder. Ardından, weatherBaseUri değişkeni, hava durumu API'sine yapılacak olan isteğin tam URI'sini oluşturur. Bu URI, hava durumu API'sine yapılan isteği tanımlar ve **process.env.WEATHER_API_KEY** ile .env dosyasından yüklenen hava durumu API anahtarını içerir.

Sonra, request fonksiyonu çağrılır. Bu fonksiyon, postman-request paketinden gelir ve HTTP isteği göndermek için kullanılır. İlk argüman, isteğin URI'sini, ikinci argüman ise bir callback fonksiyonunu içerir. Callback fonksiyonu, istek tamamlandığında veya bir hata oluştuğunda çağrılır.

Callback fonksiyonu içinde, error parametresi hata olup olmadığını belirtir ve response parametresi ise yanıtı içerir. Bu kod örneğinde, sadece yanıtın gövdesi console.log() ile yazdırılır. Yanıtın gövdesi, hava durumu API'sinden gelen verileri içerir.

Yukarıdaki görselde görüldüğü gibi, konsola response nesneminin body değerini konsola yazdırdığımızda bize JSON string'i geriye döndürmektedir.

Yapmamız gereken sıradaki işlem bu JSON string'ini **JSON.parse()** fonksiyonu yardımıyla JavaScript nesnesine dönüştürmektir.

```
weather-api > index.js > request() callback
1  require("dotenv").config();
2
3  const request = require("postman-request");
4
5  const query = `Bursa`;
6  const weatherBaseUri = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}`;
7
8  request(weatherBaseUri, (error, response) => {
9      const data = JSON.parse(response.body);
10     console.log(data);
11     console.log(typeof data);
12 });
13
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\
weather-api> node index.js
{
  request: { type: 'City', query: 'Bursa, Turkey', language: 'en', unit: 'm' },
  location: {
    name: 'Bursa',
    country: 'Turkey',
    region: 'Bursa',
    lat: '40.192',
    lon: '29.061',
    timezone_id: 'Europe/Istanbul',
    localtime: '2024-03-24 12:49',
    localtime_epoch: 1711284540,
    utc_offset: '3.0'
  },
  current: {
    observation_time: '09:49 AM',
    temperature: 12,
    weather_code: 113,
    weather_icons: [
      'https://cdn.worldweatheronline.com/images/wsymbols01_png_64/wsymbol_0001_sunny.png'
    ],
    weather_descriptions: [ 'Sunny' ],
    wind_speed: 4,
    wind_degree: 284,
    wind_dir: 'WNW',
    pressure: 1011,
    precip: 0,
    humidity: 54,
    cloudcover: 0,
    feelslike: 11,
    uv_index: 5,
    visibility: 10,
    is_day: 'yes'
  }
}
object
```

Yukarıdaki görselden görüldüğü üzere attığımız istekten gelen veriyi `JSON.parse()` metodu yardımıyla dönüştürdüğümüzde bize artık bir JavaScript nesnesi dönmektedir.

Request fonksiyonu, ikinci parametre olarak isteğe bağlı olarak bir dizi seçenek sunar. Bu seçenekler arasında, gönderilen isteğin nasıl biçimlendirileceğini belirlemek için kullanılan **"options"** parametresi bulunur. Eğer dönen verinin string formatında değil de JavaScript nesnesi formatında olmasını istiyorsak, options parametresine bir nesne geçiririz ve içine **"json: true"** ifadesini ekleriz. Bu sayede, istek sonucunda dönen veri otomatik olarak JavaScript nesnesi formatına dönüştürülür ve bize sunulur.

Aşağıdaki görsel bize bu parametrenin nasıl ekleneceğini göstermektedir.

```
weather-api > index.js > request() callback
1  require('dotenv').config();
2
3  const request = require('postman-request');
4
5  const query = `Bursa`;
6  const weatherBaseUri = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}`;
7
8  request(weatherBaseUri, { json: true }, (error, response) => {
9    const data = response.body;
10   console.log(data);
11   console.log(typeof data);
12 });
13
```

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Proje\weather-api> node index.js

```
{
  request: { type: 'City', query: 'Bursa, Turkey', language: 'en', unit: 'm' },
  location: {
    name: 'Bursa',
    country: 'Turkey',
    region: 'Bursa',
    lat: '40.192',
    lon: '29.061',
    timezone_id: 'Europe/Istanbul',
    localtime: '2024-03-24 12:56',
    localtime_epoch: 1711284960,
    utc_offset: '3.0'
  },
  current: {
    observation_time: '09:56 AM',
    temperature: 12,
    weather_code: 113,
    weather_icons: [
      'https://cdn.worldweatheronline.com/images/wsymbols01_png_64/wsymbol_0001_sunny.png'
    ],
    weather_descriptions: [ 'Sunny' ],
    wind_speed: 4,
    wind_degree: 284,
    wind_dir: 'WNW',
    pressure: 1011,
    precip: 0,
    humidity: 54,
    cloudcover: 0,
    feelslike: 11,
    uv_index: 5,
    visibility: 10,
    is_day: 'yes'
  }
}
object
```

Görselden görüldüğü üzere callback fonksiyonundan gelen response verisini parse etmeden istediğimiz çıktıyı almaktayız.

Şimdi, aldığımız hava durumu verisindeki gerçek sıcaklık ve hissedilen sıcaklık değerlerini konsola yazdıralım.

```
1  require('dotenv').config();
2
3  const request = require('postman-request');
4
5  const query = `Bursa`;
6  const weatherBaseUri = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}`;
7
8  request(weatherBaseUri, { json: true }, (error, response) => {
9    const data = response.body;
10   console.log("Hava sıcaklığı: ", data.current.temperature);
11   console.log("Hissedilen hava sıcaklığı: ", data.current.feelslike);
12 });
13
```

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\weather-api> node index.js

```
Hava sıcaklığı: 15
Hissedilen hava sıcaklığı: 14
```

Görselden görüldüğü üzere sıcaklık değerleri varsayılan olarak santigrat derece türünde yazdırılmaktadır.

<https://weatherstack.com/documentation>

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodu
\weather-api> node .\index.js
Hava sıcaklığı: 59
Hissedilen hava sıcaklığı: 57

```

Fahrenheit cinsinden geldi.

sistemleri (GIS) entegrasyonu, konum tabanlı analizler ve daha fazlası bulunmaktadır.

bağlantısında bulunan API erişim anahtarını kopyalamanız gerekmektedir.

Access tokens

You need an API access token to configure [Mapbox GL JS](#), [Mobile](#), and [Mapbox web services](#) like routing and geocoding. Read more about [API access tokens](#) in our documentation.

[+ Create a token](#)

Default public token

 Refresh

pk.eyJ1IjoieYmVya2Fuc2VyYmVzIiwiaSI6ImNsZHpoYjd4dTAwMWEybXB1IiwiaWQiOiJ1IiwiaWF0IjoiMTU1MjY0MjY0InQ=

Last modified: 4 days ago

URLs: N/A

Sonrasında bu API erişim anahtarını .env dosyamıza **MAPBOX_API_KEY** adında bir değişkene atayalım.

```
weather-api > .env
1 WEATHER_API_KEY="8160d4b42bcd0921f6c..."
2 MAPBOX_API_KEY="pk.eyJ1Ijo1YmVya2Fuc2VyYmVzIiw1YSI6ImNsdHpoYjYjd4dTAwMWEybXBkLnng..."
```

Sonrasında hangi bölgenin verilerini çekmek istiyorsak bunu bir değişkende tutmamız gerekmektedir.mapBoxQuery adında bir değişken tanımlıyoruz ve içerisine 'bursa.json' değerini yazıyoruz dilerseniz başka bir şehrin adını da yazabilirsiniz.

```
weather-api > index.js > request() callback
1 require("dotenv").config();
2
3 const request = require("postman-request");
4
5 const query = 'Bursa';
6 const mapBoxQuery = 'bursa.json';
7
8 const units = "f";
9 const weatherBaseUrl = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}&units=${units}`;
10 const mapBoxBaseUrl = `https://api.mapbox.com/geocoding/v5/mapbox.places/${mapBoxQuery}?access_token=${process.env.MAPBOX_API_KEY}`;
11
12 request(mapBoxBaseUrl, { json: true }, (error, response) => {
13   const data = response.body;
14   console.log(data);
15 });
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
• \weather-api> node index.js
{
  type: 'FeatureCollection',
  query: [ 'bursa' ],
  features: [
    {
      id: 'place.1935588',
      type: 'Feature',
      place_type: [Array],
      relevance: 1,
      properties: [Object],
      text: 'Bursa',
      place_name: 'Bursa, Bursa, Türkiye',
      bbox: [Array],
      center: [Array],
      geometry: [Object],
      context: [Array]
    },
    {
      id: 'region.247012',
      type: 'Feature',
      place_type: [Array],
      relevance: 1,
      properties: [Object],
      text: 'Bursa',
      place_name: 'Bursa, Türkiye',
      bbox: [Array],
      center: [Array],
      geometry: [Object],
      context: [Array]
    },
    {
      id: 'poi.188978602777',
      type: 'Feature',
      place_type: [Array],
      relevance: 1,

```

Yukarıdaki kod parçasının çıktısı konsolda görünmektedir.

Bizim erişmek istediğimiz veriler, features dizisinde bulunmaktadır. O yüzden aşağıdaki şekildeki gibi features dizisine ulaşmamız gerekmektedir.

```
6 const mapBoxQuery = 'bursa.json';
7
8 const units = "f";
9 const weatherBaseUri = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}&units=${units}`;
10 const mapBoxBaseURL = `https://api.mapbox.com/geocoding/v5/mapbox.places/${mapBoxQuery}?access_token=${process.env.MAPBOX_API_KEY}`;
11
12 request(mapBoxBaseURL, { json: true }, (error, response) => {
13   const data = response.body;
14   console.log(data.features);
15 });
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL powershell - weather-api

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Ha
\weather-api> node .\index.js
[
  {
    id: 'place.1935588',
    type: 'Feature',
    place_type: [ 'place' ],
    relevance: 1,
    properties: { mapbox_id: 'dXJu0m1ieHBsYzpIWMpr', wikidata: 'Q40738' },
    text: 'Bursa',
    place_name: 'Bursa, Bursa, Türkiye',
    bbox: [ 28.664827, 39.995556, 29.259995, 40.346381 ],
    center: [ 29.06773, 40.182766 ],
    geometry: { type: 'Point', coordinates: [Array] },
    context: [ [Object], [Object] ]
  },
  {
    id: 'region.247012',
    type: 'Feature',
    place_type: [ 'region' ],
    relevance: 1,
    properties: {
      mapbox_id: 'dXJu0m1ieHBsYzpB0FRn',
      wikidata: 'Q43690',
      short_code: 'TR-16'
    },
    text: 'Bursa',
    place_name: 'Bursa, Türkiye',
    bbox: [ 28.100619, 39.596634, 29.975333, 40.7805414 ],
    center: [ 29.0675481, 40.182737 ],
    geometry: { type: 'Point', coordinates: [Array] },
    context: [ [Object] ]
  },
]
```

```
weather-api> index.js> request() callback
1 require("dotenv").config();
2
3 const request = require("postman-request");
4
5 const query = 'Bursa';
6 const mapBoxQuery = 'bursa.json';
7
8 const units = "f";
9 const weatherBaseUri = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${query}&units=${units}`;
10 const mapBoxBaseURL = `https://api.mapbox.com/geocoding/v5/mapbox.places/${mapBoxQuery}?access_token=${process.env.MAPBOX_API_KEY}`;
11
12 request(mapBoxBaseURL, { json: true }, (error, response) => {
13   const data = response.body;
14   console.log(data.features[0].center[0]); // Boylam
15   console.log(data.features[0].center[1]); // Enlem
16 });
17
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL powershell - weath

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKod
\weather-api> node index.js
29.06773
40.182766
```

Yukarıdaki kod parçası Bursanın bulunduğu kordinatları enlem ve boylam cinsinden yazdırmaktadır.

5.Kaynakça

[linkedin.com/posts/javascript-interview-questions](https://www.linkedin.com/posts/javascript-interview-questions)

<https://chat.openai.com/>