

BURSA TEKNİK ÜNİVERSİTESİ

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



BLM0470 NodeJS İle Web Programlama

2023-2024 Bahar Dönemi

6.Hafta Raporu

Berkan SERBES – 22360859353

İçindekiler

1. Callback Fonksiyonlar.....	3
1.1 Senkron Biçiminde Çalışan Callback Fonksiyonları	3
1.1.1 Senkron Callback Fonksiyon Örnekleri	4
1.2 Asenkron Biçiminde Çalışan Callback Fonksiyonları	6
1.2.1 Asenkron Callback Fonksiyon Örnekleri	6
1.3 Callback Chain	8
2. Hava Durumu Uygulaması Geliştirme	9
3. Kaynakça	15

1.Callback Fonksiyonlar

Callback fonksiyonları, JavaScript'te yaygın olarak kullanılan bir programlama kavramıdır ve asenkron programlamada sıklıkla karşılaşılır. Bir fonksiyonun, diğer bir fonksiyonun argümanı olarak geçtiği ve daha sonra çağrılacağı bir yapıyı ifade ederler.

Asenkron programlama, işlemlerin sırasını beklemeksizin devam ettirilmesini sağlayan bir programlama yaklaşımıdır. Örneğin, bir dosyanın okunması, bir ağ isteğinin tamamlanması veya bir zamanlayıcı tarafından tetiklenen bir işlem gibi uzun süren işlemler asenkron olarak gerçekleştirilir. Callback fonksiyonları, bu tür asenkron işlemlerin sonuçlarını ele almak, verileri işlemek ve hata durumlarını yönetmek için kullanılır. Callback fonksiyonların hepsi asenkron işlemler için değil senkron işlemler için de kullanılabilir.

1.1 Senkron Biçiminde Çalışan Callback Fonksiyonları

Senkron callback fonksiyonları, JavaScript'te işlemlerin ardışık ve senkronize bir şekilde gerçekleştirilmesini sağlayan callback fonksiyonlardır. Bu fonksiyonlar, genellikle bir işlemin tamamlandığında çağrılacak bir geri çağırma işlevini içerir. İşlemlerin sıralı ve ardışık olarak gerçekleştiği senkronize bir ortamda kullanılan senkron callback fonksiyonları, bir fonksiyon çağrıldığında, bir sonraki işlem sırayla gerçekleştirilir ve bu işlem tamamlandığında geri çağırma işlevi çalıştırılır. Bu fonksiyonlar, bir işlem tamamlanana kadar beklemeyi gerektirebilirler. Örneğin, bir dosyanın okunması veya bir dizi üzerinde işlem yapılması gibi işlemler, sonuç elde edilene kadar beklemeyi gerektirir ve bu süreç boyunca geri çağırma işlevi bekletilir. İşlem sırasında blokaj oluşturabilirler ve işlemlerin tamamlanmasını beklerler. Bu durumda, bir işlem tamamlanana kadar diğer işlemler duraklatılır ve sıradaki işlem gerçekleştirilir. Hata yönetimi açısından, senkron callback fonksiyonları, işlem sırasında hataları yakalamak ve işlem sonucunu işlemek için kullanılabilir. Bir işlem başarısız olursa, hata durumu geri çağırma işlevine iletilir ve uygun şekilde ele alınabilir. Özetle, senkron callback fonksiyonları, işlemlerin sıralı ve senkronize bir şekilde gerçekleştirilmesini sağlar. Bu fonksiyonlar, işlemlerin sonuçlarını işlemek ve hata yönetimini sağlamak için kullanılır. Ancak, büyük veri işlemleri gibi uzun süren işlemlerde kullanıldığında uygulamayı bloke edebilirler, bu nedenle dikkatli kullanılmalıdır.

JavaScript'te senkron callback ile çalışan dizi metotları, `forEach()`, `map()`, `filter()` ve `reduce()` gibi fonksiyonlardır. Bu metotlar, bir dizi içindeki her bir eleman üzerinde işlem yapar ve belirli bir işlevi gerçekleştirir. Örneğin, `forEach()` metodu dizi elemanlarını döngüyle işlerken, `map()` metodu her eleman üzerinde belirli bir dönüşüm işlemi gerçekleştirir ve yeni bir dizi oluşturur. Benzer şekilde, `filter()` metodu belirli bir koşulu sağlayan elemanları filtreler ve `reduce()` metodu ise dizi elemanları arasında bir kümülatif işlem yapar. Bu metotlar, işlevlerini tamamlamak için bir geri çağırma işlevi (callback function) kullanır ve işlemleri ardışık bir şekilde gerçekleştirir. Bu sayede, dizi üzerindeki her eleman üzerinde işlem yapılırken program akışı bloke edilmez ve işlemler sırayla gerçekleştirilir.

```
index.js  callback-test.js X
callback-test.js > ...
1  const sayHello = (callback) => {
2    console.log("Hello");
3    callback();
4  };
5
6  const callbackFunction = () => {
7    console.log("Callback fonksiyon çalıştı");
8  };
9
10 sayHello(callbackFunction);
11

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\RaporKodlari\Hafta6_RaporKodlari
> node .\callback-test.js
Hello
Callback fonksiyon çalıştı
```

Bu kod, bir callback fonksiyonunu kullanarak "Hello" mesajını yazdıran ve ardından başka bir fonksiyonu tetikleyen basit bir JavaScript örneğidir. İlk olarak, sayHello adında bir fonksiyon tanımlanır. Bu fonksiyon, bir callback fonksiyonunu parametre olarak alır. Fonksiyon içinde "Hello" mesajı konsola yazdırılır ve ardından callback fonksiyonu çağrılır. Ardından, callbackFunction adında başka bir fonksiyon tanımlanır. Bu fonksiyon, konsola "Callback fonksiyon çalıştı" mesajını yazdırır. Son olarak, sayHello fonksiyonu çağrılır ve callbackFunction fonksiyonu argüman olarak geçirilir. Bu şekilde, "Hello" mesajı yazıldıktan sonra callback fonksiyonu çağrılarak ek bir işlem gerçekleştirilir.

1.1.1 Senkron Callback Fonksiyon Örnekleri

```
index.js  callback-test.js X
callback-test.js > ...
1  const numbers = [1, 2, 3, 4, 5];
2
3  console.log("Start");
4
5  numbers.forEach((number) => {
6    console.log(number);
7  });
8
9  console.log("Finish");
10

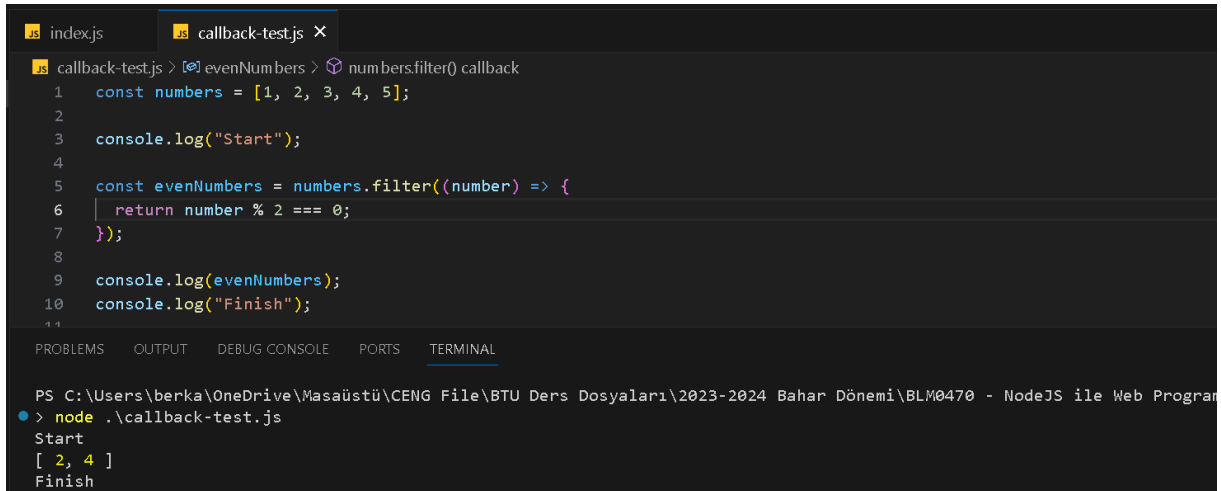
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodlari\Hafta6_RaporKodlari
> node .\callback-test.js
Start
1
2
3
4
5
Finish
```

Bu örnekte, forEach() metodu kullanılarak dizi içindeki her bir sayı konsola yazdırılır. forEach() metodu senkron bir şekilde çalışır. Dizi elemanları üzerindeki işlem tamamlandıktan sonra "Finish" mesajı konsola yazdırılır.

```
callback-test.js > numbers.map() callback
1  const numbers = [1, 2, 3, 4, 5];
2
3  console.log("Start");
4
5  numbers.map((number) => {
6    console.log(number * 4);
7  });
8
9  console.log("Finish");
10

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\Proje
> node .\callback-test.js
Start
4
8
12
16
20
Finish
```

Bu kod, bir dizi olan numbers içindeki her bir elemanı dört ile çarparak sonucunu ekrana yazdırmaktadır. İlk olarak "Start" mesajı konsola yazdırılır. Daha sonra map() metodu kullanılarak her bir eleman dört ile çarpılır, ancak bu işlem sırasında her bir sonuç ekrana yazdırılmaz. Bunun nedeni, map() metodu senkron bir callback fonksiyonu kullanmasına rağmen, içerideki console.log() işleminin her bir eleman için ayrı bir işlem olarak değerlendirilmemesidir. Son olarak, "Finish" mesajı konsola yazdırılır. Bu nedenle, ekrana yazdırılan değerler arasında herhangi bir bekleme veya gecikme olmaz.

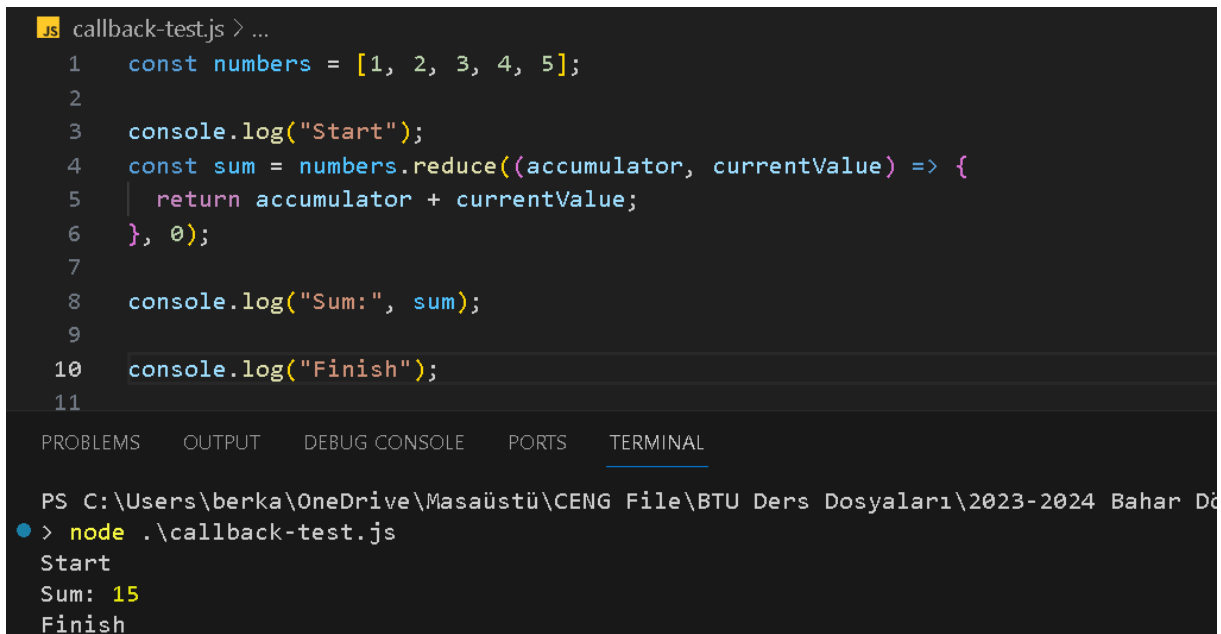


```
index.js | callback-test.js X
callback-test.js > [?] evenNumbers > [?] numbers.filter() callback
1  const numbers = [1, 2, 3, 4, 5];
2
3  console.log("Start");
4
5  const evenNumbers = numbers.filter((number) => {
6    return number % 2 === 0;
7  });
8
9  console.log(evenNumbers);
10 console.log("Finish");
11

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 Program
> node .\callback-test.js
Start
[ 2, 4 ]
Finish
```

Bu kod, numbers adında bir dizi içinde bulunan sayılardan yalnızca çift olanları filtreleyerek yeni bir dizi oluşturur. İlk olarak "Start" mesajı konsola yazdırılır. Daha sonra filter() metodu kullanılarak, dizi içindeki her bir eleman kontrol edilir ve sadece çift sayılar (number % 2 === 0 koşulu sağlayanlar) evenNumbers adlı yeni bir diziye eklenir. Bu işlem senkron bir şekilde gerçekleşir ve ekrana yazdırılan evenNumbers dizisi, yalnızca çift sayıları içerir. Son olarak "Finish" mesajı konsola yazdırılır.



```
callback-test.js > ...
1  const numbers = [1, 2, 3, 4, 5];
2
3  console.log("Start");
4  const sum = numbers.reduce((accumulator, currentValue) => {
5    return accumulator + currentValue;
6  }, 0);
7
8  console.log("Sum:", sum);
9
10 console.log("Finish");
11

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar D
> node .\callback-test.js
Start
Sum: 15
Finish
```

Bu kod, numbers adında bir dizide bulunan sayıların toplamını hesaplar. İlk olarak "Start" mesajı konsola yazdırılır. Daha sonra reduce() metodu kullanılarak, dizi elemanları üzerinde bir döngü yapılır ve her bir eleman toplam değişkenine eklenir. Başlangıçta accumulator

değeri 0 olarak belirlenir ve her adımda bu değere dizi elemanları eklenir. Sonuç olarak, sum değişkeninde dizi elemanlarının toplamı elde edilir. Bu işlem senkron bir şekilde gerçekleşir ve toplam değeri ekrana yazdırılır. Son olarak "Finish" mesajı konsola yazdırılır.

1.2 Asenkron Biçiminde Çalışan Callback Fonksiyonları

Asenkron callback fonksiyonları, JavaScript'te işlemlerin asenkron ve non-blocking bir şekilde gerçekleştirilmesini sağlayan önemli yapı taşlarından biridir. Bu tür fonksiyonlar, bir işlem tamamlandığında çağrılacak olan bir geri çağırma işlevini içerir ve genellikle asenkron işlemleri yönetmek için kullanılır. Örneğin, dosya okuma, ağ isteği yapma veya zamanlayıcı işlemleri gibi işlemler asenkron olarak gerçekleştirilebilir. Bu sayede, bir işlem tamamlanana kadar beklemeksizin diğer işlemler devam edebilir. Asenkron callback fonksiyonları, işlemin sonucunu veya hatasını ele almak için de kullanılır ve hata yönetimini sağlar. Ayrıca, belirli bir zamana bağlı olarak işlemleri planlamak için de kullanılabilirler.

JavaScript'teki `setTimeout()` ve `setInterval()` fonksiyonları asenkron callback fonksiyonlarına örnek olarak kabul edilir çünkü belirli bir zaman aralığı geçtikten veya belirli bir olay gerçekleştikten sonra çağrılan işlevler, asenkron olarak çalışır ve JavaScript'in non-blocking yapısını korur.

1.2.1 Asenkron Callback Fonksiyon Örnekleri

```
js callback-test.js > ...
1 console.log("Start");
2
3 setTimeout(() => {
4   console.log("1s Timeout");
5 }, 1000);
6
7 console.log("End");
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\Bl
● > node .\callback-test.js
Start
End
○ 1s Timeout
```

"setTimeout" metodu, belirli bir süre sonra bir işlevi çağırmaq için kullanılır. İlk parametre olarak çağrılacak işlevi, ikinci parametre olarak ise çağırılma zamanını (milisaniye cinsinden) alır. Bu zaman dilimi geçtikten sonra, belirtilen işlev asenkron olarak çağrılır.

```
callback-test.js > setInterval() callback
1 console.log("Start");
2
3 setInterval(() => {
4   console.log("Is Interval");
5 }, 1000);
6
7 console.log("End");
8
9 // const sayHello = (callback) => {
10 //   console.log("Hello");
11 // }

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\Proje
> node .\callback-test.js
Start
End
1s Interval
1s Interval
1s Interval
1s Interval
1s Interval
1s Interval
```

"setInterval" metodu, belirli aralıklarla bir işlevi çağırmak için kullanılır. İlk parametre olarak çağrılacak işlevi, ikinci parametre olarak ise aralığı (milisaniye cinsinden) alır. Belirtilen aralık geçtikten sonra, işlev asenkron olarak çağrılır ve bu işlem belirtilen aralıklarla tekrarlanır.

```
callback-test.js > callbackFunction
1 const add = (num1, num2, callback) => {
2   const sum = num1 + num2;
3   callback(sum);
4 };
5
6 const callbackFunction = (result) => {
7   return console.log("Sum:", result);
8 };
9
10 add(5, 10, callbackFunction);
11

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024
> node .\callback-test.js
Sum: 15
```

Bu kodda, "add" adında bir fonksiyon tanımlanır. Bu fonksiyon, iki sayı alır ve bu sayıların toplamını hesaplar. Ardından, üçüncü bir parametre olarak aldığı "callback" fonksiyonunu çağırır ve toplamı bu geri çağırma işlevine iletir. Daha sonra, "callbackFunction" adında başka bir fonksiyon tanımlanır. Bu fonksiyon, bir parametre alır ve bu parametreyi konsola yazdırır. "add" fonksiyonu çağrılarak 5 ve 10 sayıları toplanır ve sonuç, "callbackFunction" fonksiyonuna iletildikten sonra konsola yazdırılır.

1.3 Callback Chain

Callback zinciri (callback chain), JavaScript'te asenkron işlemleri sıralı olarak yönetmek için kullanılan bir desendir. Bu desen, bir işlemin tamamlanmasının ardından bir sonraki işlemin başlatılmasını sağlar ve işlemler arasında bağımlılıkları belirtmek için kullanılır. Özellikle dosya okuma, veritabanı sorguları veya ağ istekleri gibi işlemlerde sıkça karşımıza çıkar. Bu zincir, bir işlemin sonucuna bağlı olarak diğer işlemlerin sıralı olarak gerçekleştirilmesini sağlar.

Örneğin, bir dosyanın okunması işlemi için bir callback fonksiyonu belirlenir ve bu işlem tamamlandığında çağrılır. Ardından, dosya okunduktan sonra yapılacak işlemler için başka bir callback fonksiyonu belirlenir ve bu şekilde zincirleme devam edilir. Her bir callback fonksiyonu, işlemin sonucunu ele alır ve gerekirse bir sonraki işlemi başlatır. Bu sayede, asenkron işlemler sıralı ve düzenli bir şekilde gerçekleştirilebilir.

Callback zinciri, kodun daha okunabilir ve sürdürülebilir olmasını sağlar, çünkü işlemler arasındaki bağımlılıklar açıkça belirtilir. Ancak, derinleşen callback zincirleri, kodun karmaşık hale gelmesine ve "callback cehennemi (callback hell)" sorununa yol açabilir. Bu nedenle, modern JavaScript uygulamalarında Promise, async/await gibi alternatifler kullanılarak callback zinciri yerine daha temiz ve okunabilir kod yapıları oluşturulabilir.

```
callback-test.js > readFile("example.txt") callback
1 // Asenkron olarak bir dosya okuma işlemi
2 const readFile = (fileName, callback) => {
3   setTimeout(() => {
4     const content = "Dosya içeriği...";
5     callback(null, content); // Hata olmadığını ve içeriği geri döndürür
6   }, 1000);
7 };
8
9 // İkinci bir işlem için callback zinciri
10 const processFile = (content, callback) => {
11   setTimeout(() => {
12     const processedContent = content.toUpperCase();
13     callback(null, processedContent); // Hata olmadığını ve işlenmiş içeriği geri döndürür
14   }, 500);
15 };
16
17 // Dosya okuma işlemi başlatılır
18 readFile("example.txt", (err, fileContent) => {
19   if (err) {
20     console.error("Dosya okuma hatası:", err);
21     return;
22   }
23
24   // Okunan dosya içeriği işlenir
25   processFile(fileContent, (err, processedContent) => {
26     if (err) {
27       console.error("Dosya işleme hatası:", err);
28       return;
29     }
30
31     // İşlenmiş dosya içeriği yazdırılır
32     console.log("İşlenmiş dosya içeriği:", processedContent);
33   });
34 });
35
```

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\Proje\callback-test.js

● > node .\callback-test.js

○ İşlenmiş dosya içeriği: DOSYA İÇERİĞİ...

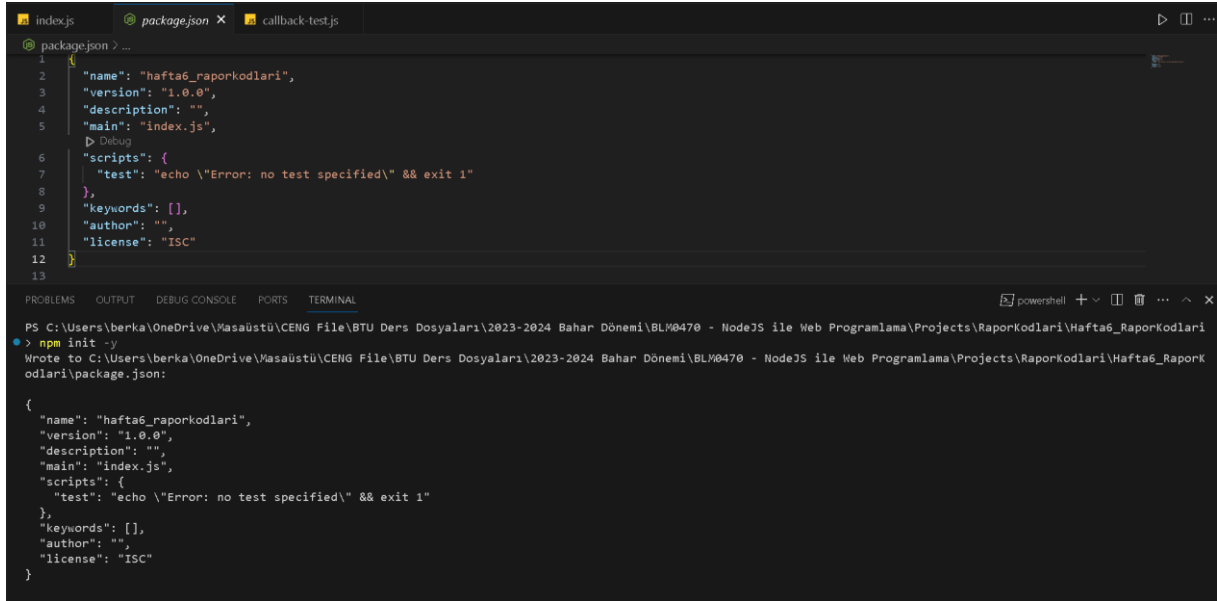
Bu örnekte, readFile fonksiyonu önce bir dosya okuma işlemi simüle eder, ardından processFile fonksiyonu bu dosya içeriğini alır ve işler. Her iki işlem de asenkron olarak gerçekleştirilir ve bir sonraki adıma geçmek için bir callback fonksiyonu kullanılır. Bu, bir

callback zinciri oluşturur: Dosya okuma işlemi tamamlandığında, işlenmiş dosya içeriğini almak için bir sonraki işlem başlatılır.

2. Hava Durumu Uygulaması Geliştirme

Bu başlık altında, bir hava durumu uygulaması geliştireceğiz. Bu uygulama basitçe kullanacağımız dış bir API servisinden GET istekleri gerçekleştirecek. Bu GET isteklerini gerçekleştirirken callback fonksiyonlardan yararlanacağız.

İlk olarak projemizin olduğu dizinde komut satırı yardımıyla **npm init -y** komutunu kullanarak package.json dosyamızı oluşturalım.

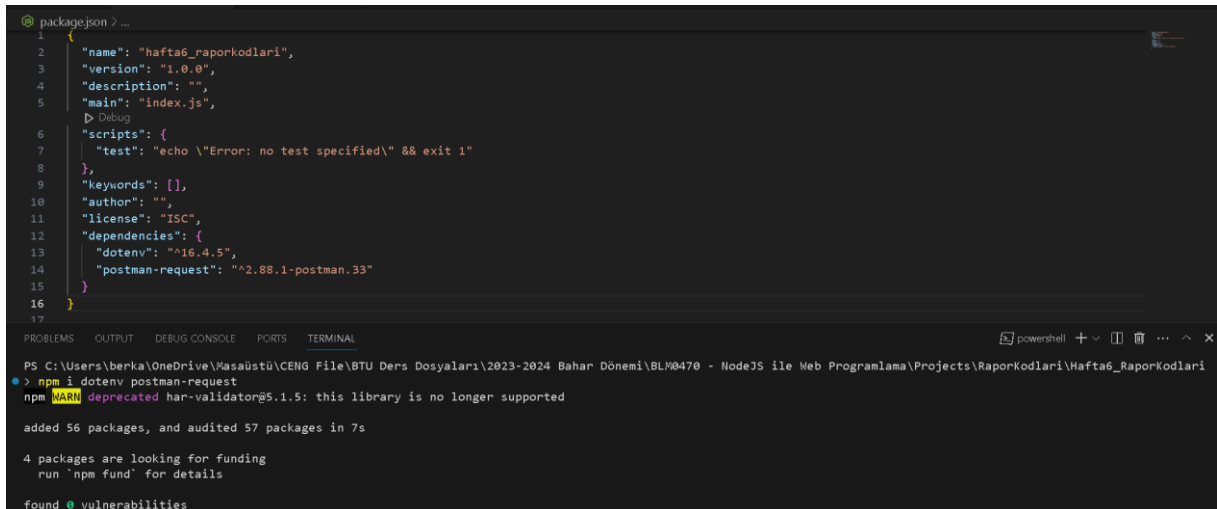


```
index.js package.json x callback-test.js
package.json > ...
1 {
2   "name": "hafta6_raporkodlari",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
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\Projects\RaporKodlari\Hafta6_RaporKodlari
> 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\Hafta6_RaporKodlari\package.json:

{
  "name": "hafta6_raporkodlari",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Sonrasında kullanacağımız paketleri npm install komutunun yardımıyla uygulamamıza dahil edelim. Kullanacağımız paketler dotenv ve postman-request paketleridir.



```
package.json > ...
1 {
2   "name": "hafta6_raporkodlari",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "dotenv": "^16.4.5",
14    "postman-request": "^2.88.1-postman.33"
15  }
16 }
17

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\RaporKodlari\Hafta6_RaporKodlari
> 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 7s

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

found 0 vulnerabilities
```

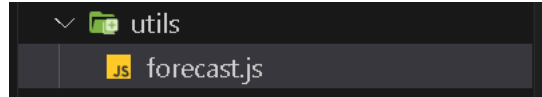
Dış servis olarak kullanacağımız sitelere üye olup API erişim anahtarlarını edinmemiz gerekmektedir. Bunun için aşağıdaki sitelere üye olmamız gerekmektedir.

- <https://weatherstack.com/signup/free>
- <https://account.mapbox.com/auth/signup/>

Başarılı bir şekilde üye olduktan sonra, her iki sitede de aldığımız API erişim anahtarlarını projemizde bir .env dosyası oluşturup bu anahtarları eklememiz gerekiyor aşağıdaki görseldeki gibi.

```
.env
1 WEATHER_API_KEY="8160d4b42bcd0921f8..."
2 MAPBOX_API_KEY="pk.eyJ1IjoiYmVya2Fuc2VyYmVzIiwiaSI6ImNsdHpoYjd4dTANmWEybXBkNWgwOWE2NW0ifQ..."
```

Daha sonrasında projemizde kullanacağımız callback fonksiyonlarını oluşturmamız gerekmektedir. Bunun için utils adında bir klasör açıp fonksiyonlarımızı bu klasörün içerisinde tutacağız.



Forecast.js adında bir klasör oluşturalım. Bu dosyada olacak fonksiyon, weatherstack API'sini kullanarak hava durumu verilerini bir callback fonksiyonu aracılığıyla çekecek.

Aşağıdaki kod parçası, bir hava durumu API'si olan weatherstack'i kullanarak hava durumu verilerini çekmek için tasarlanmış bir fonksiyon olan getForecastData'yı içerir. Bu fonksiyon, bir konumun boylam ve enlem bilgilerini alır ve bu bilgilere göre ilgili hava durumu verilerini API'den çeker.

Fonksiyonun içeriği şu adımları izler: Öncelikle, parametrelerden alınan boylam ve enlem bilgileri kullanılarak bir URL oluşturulur. Bu URL, API'ye istek göndermek için kullanılacak olan adresi içerir. Daha sonra, request modülü aracılığıyla API'ye istek gönderilir. Bu istek, oluşturulan URL ile birlikte yapılır.

API'den gelen yanıt değerlendirilir. Eğer bir hata varsa veya istek başarısız olursa, uygun hata mesajlarıyla birlikte callback fonksiyonuna bilgi gönderilir. Eğer herhangi bir hata yoksa ve istek başarılıysa, API'den gelen verilerden sıcaklık ve hissedilen sıcaklık bilgileri alınarak, bu bilgilerle birlikte callback fonksiyonuna başarılı bir şekilde veri gönderilir.

Son olarak, bu fonksiyon module.exports ile dışarıya açılarak, başka dosyalardan erişilebilir hale getirilir. Bu sayede, hava durumu verilerini çekmek için bu fonksiyon başka dosyalarda kullanılabilir.

```
index.js forecast.js X package.json .env callback-test.js
utils > forecast.js > ...
1 require("dotenv").config();
2 const request = require("postman-request");
3
4 const getForecastData = (longitude, latitude, callback) => {
5   const url = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${longitude},${latitude}`;
6   try {
7     request({ url, json: true }, (error, response) => {
8       if (error) {
9         callback("Unable to connect to weather service", undefined);
10      } else if (response.body.error) {
11        callback("Unable to find location", undefined);
12      } else {
13        const temperature = response.body.current.temperature;
14        const feelslike = response.body.current.feelslike;
15        callback(undefined, { temperature, feelslike });
16      }
17    });
18  } catch (err) {
19    callback(err.message, undefined);
20  }
21 };
22
23 module.exports = getForecastData;
24
```

index.js dosyası, "./utils/forecast" modülünden getForecastData fonksiyonunu içeri aktarır. Bu fonksiyon, belirtilen boylam ve enlem koordinatlarına sahip bir konum için hava durumu verilerini almak için kullanılır.

getForecastData fonksiyonu çağrılırken, belirtilen koordinatlarla birlikte bir callback fonksiyonu da sağlanır. Callback fonksiyonu, hava durumu verileri alındığında işlenecek ve kullanılacak olan fonksiyondur. Fonksiyon, belirtilen koordinatlarla birlikte bir API isteği gönderir. API'den gelen yanıt, error ve data parametreleri aracılığıyla geri çağırma fonksiyonuna iletilir. Eğer istekte herhangi bir hata oluşursa, bu hata error parametresi aracılığıyla yakalanır ve konsola yazdırılır. Aksi halde, hava durumu verileri data parametresi aracılığıyla alınır ve kullanılır.

```
index.js > ...
1 const getForecastData = require("../utils/forecast");
2
3 getForecastData(37.8267, -122.4233, (error, data) => {
4   if (error) {
5     console.error("Forecast error:", error);
6     return;
7   }
8
9   console.log("Forecast data:", data);
10 }
11
```

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta6_RaporKodları > node index.js
Forecast data: { temperature: 13, feelslike: 13 }

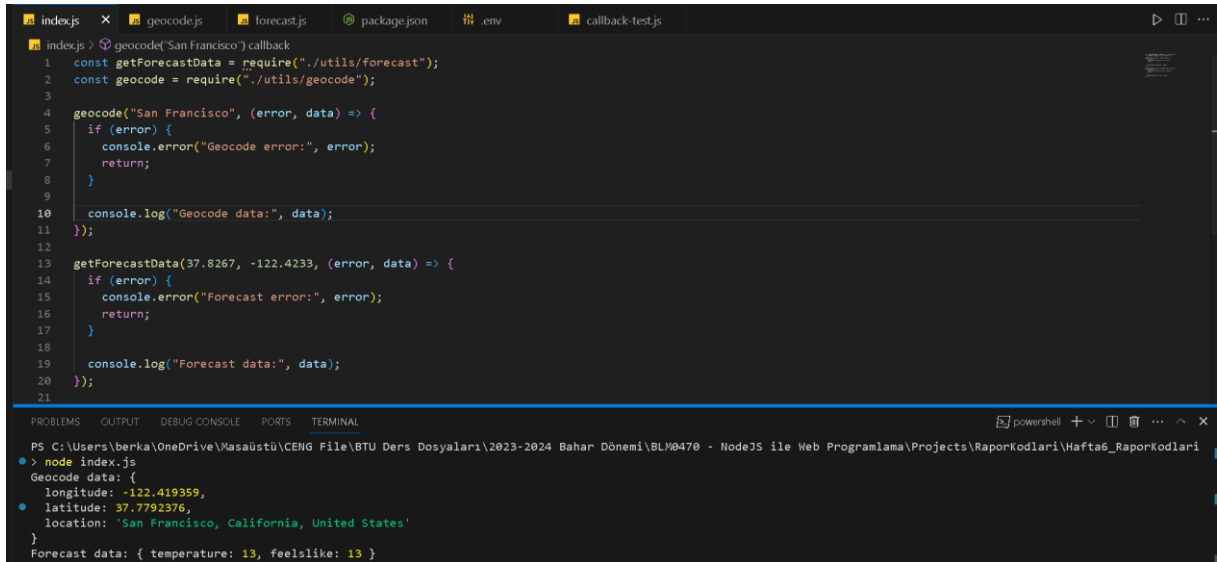
```
index.js geocode.js forecast.js package.json .env callback-test.js
utils > geocode.js > ...
1  require("dotenv").config();
2  const request = require("postman-request");
3
4  const geocode = (address, callback) => {
5    const url = `https://api.mapbox.com/geocoding/v5/mapbox.places/${encodeURIComponent(
6      address
7    )}.json?access_token=${process.env.MAPBOX_API_KEY}`;
8    try {
9      request({ url, json: true }, (error, response) => {
10        if (error) {
11          callback("Unable connect mapbox service", undefined);
12        } else if (response.body.features.length === 0) {
13          callback("Data cannot found", undefined);
14        } else {
15          const longitude = response.body.features[0].center[0]; // boylam
16          const latitude = response.body.features[0].center[1]; // enlem
17
18          callback(undefined, {
19            longitude,
20            latitude,
21            location: response.body.features[0].place_name,
22          });
23        }
24      });
25    } catch (err) {
26      callback(err.message, undefined);
27    }
28  };
29
30  module.exports = geocode;
31
```

Bu kod, bir adresin coğrafi konumunu belirlemek için Mapbox API'sini kullanır. İlk olarak, "dotenv" ve "postman-request" modülleri projeye dahil edilir. "dotenv" modülü, çevresel değişkenlerin projede kullanılmasını sağlar. "postman-request" modülü ise HTTP isteklerini yapmak için kullanılır.

Ardından, "geocode" adında bir fonksiyon oluşturulur. Bu fonksiyon, bir adres ve bir callback fonksiyonu alır. Adres, Mapbox API'sine gönderilecek olan sorguyu oluşturmak için kullanılır. Oluşturulan URL, Mapbox API'sine gönderilen bir GET isteği yapmak için kullanılır.

encodeURIComponent fonksiyonu, belirli karakterleri URL içinde güvenli bir şekilde kullanılabilir hale getirmek için kullanılır. URL'lerde özel anlama sahip olan bazı karakterler, doğrudan kullanıldıklarında hatalara veya istenmeyen sonuçlara neden olabilir. Örneğin, URL'lerde boşluk yerine "+" işareti veya "%20" gibi bir işaretle temsil edilir.

Bu istek, belirtilen adrese göre bir coğrafi konum döndürür. İstek gönderildiğinde, bir callback fonksiyonu kullanılır. Bu callback fonksiyonu, API'den gelen yanıtı işlemek için kullanılır. Eğer istekte herhangi bir hata oluşursa, bu hata "error" parametresi aracılığıyla yakalanır ve geri çağırma fonksiyonuna iletilir. Eğer istek başarılı olursa, API'den gelen yanıt "response" parametresi aracılığıyla işlenir. Yanıtta hata yoksa ve belirtilen adrese karşılık gelen coğrafi konum bilgileri bulunuyorsa, bu bilgiler callback fonksiyonuna iletilir. Son olarak, "geocode" fonksiyonu module.exports ile dışarıya açılır ve başka dosyalarda kullanılabilir hale gelir.



```
index.js | geocode.js | forecast.js | package.json | .env | callback-test.js
index.js > geocode('San Francisco') callback
1 const getForecastData = require("../utils/forecast");
2 const geocode = require("../utils/geocode");
3
4 geocode("San Francisco", (error, data) => {
5   if (error) {
6     console.error("Geocode error:", error);
7     return;
8   }
9
10  console.log("Geocode data:", data);
11 });
12
13 getForecastData(37.8267, -122.4233, (error, data) => {
14   if (error) {
15     console.error("Forecast error:", error);
16     return;
17   }
18   console.log("Forecast data:", data);
19 });
20
21
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\Hafta6_RaporKodları
> node index.js
Geocode data: {
  longitude: -122.419359,
  latitude: 37.7792376,
  location: 'San Francisco, California, United States'
}
Forecast data: { temperature: 13, feelslike: 13 }
```

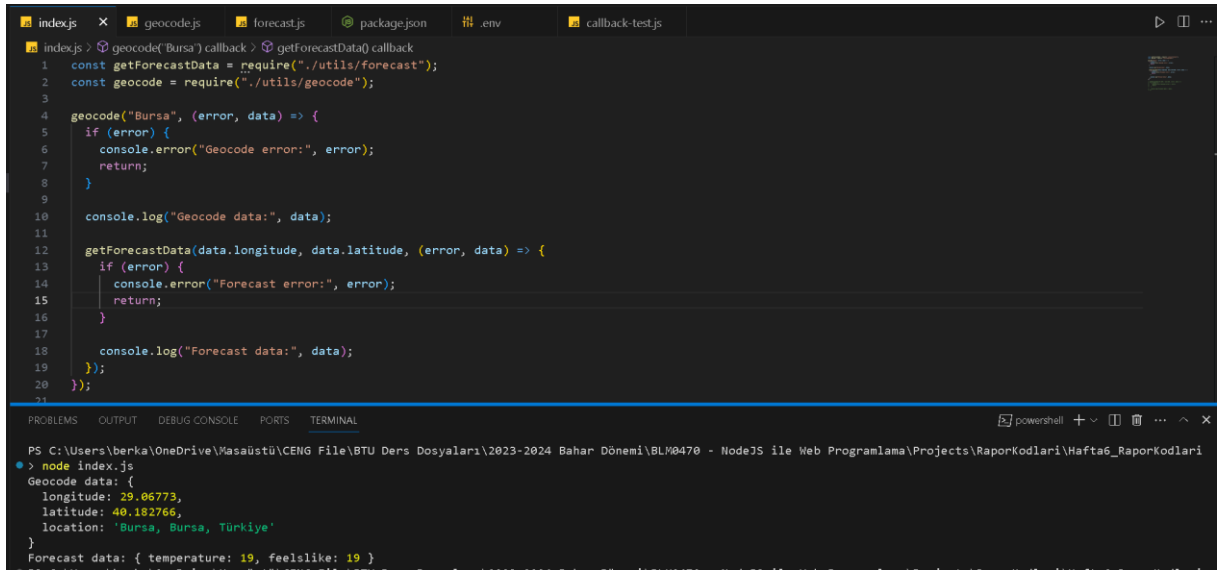
index.js adlı dosyamız yukarıdaki görseldeki gibi şekillenecektir. Bu ana kodda, hava durumu tahmin verilerini almak için iki farklı servisi kullanıyor. Birincisi, "geocode" servisi, belirli bir yer adına karşılık gelen coğrafi konum bilgilerini almak için Mapbox API'sini kullanıyor. İkincisi, "getForecastData" servisi, belirli bir enlem ve boylama sahip bir konum için hava durumu tahmin verilerini almak için Weatherstack API'sini kullanıyor.

Kod, önce "San Francisco" gibi bir yer adı vererek coğrafi konum bilgilerini alıyor. Eğer işlem sırasında herhangi bir hata olursa, konsola hata mesajı yazdırılıyor. Eğer hata olmazsa, coğrafi konum bilgileri konsola yazdırılıyor.

Ardından, belirli bir enlem ve boylama sahip bir konum için hava durumu tahmini verileri alınıyor. Yine, eğer işlem sırasında bir hata oluşursa, konsola hata mesajı yazdırılıyor. Eğer hata olmazsa, hava durumu tahmini verileri konsola yazdırılıyor.

Bu kod, asenkron bir şekilde çalışır, yani bir işlem diğerinin tamamlanmasını beklemek zorunda kalmaz. Bu sayede, hava durumu tahmini verilerini daha hızlı ve etkili bir şekilde alabiliriz.

Şimdi, ana fonksiyonumuzu yeniden düzenleyerek, bir fonksiyonun çıktısının diğerini etkileyecek şekilde yapılandıralım. Yani, bir fonksiyonun sonucu, diğerinin giriş parametresi veya çağırısı olarak kullanılacak şekilde ayarlayalım.



```
index.js x geocode.js forecast.js package.json .env callback-test.js
index.js > geocode('Bursa') callback > getForecastData() callback
1 const getForecastData = require("../utils/forecast");
2 const geocode = require("../utils/geocode");
3
4 geocode("Bursa", (error, data) => {
5   if (error) {
6     console.error("Geocode error:", error);
7     return;
8   }
9
10  console.log("Geocode data:", data);
11
12  getForecastData(data.longitude, data.latitude, (error, data) => {
13    if (error) {
14      console.error("Forecast error:", error);
15      return;
16    }
17
18    console.log("Forecast data:", data);
19  });
20 });
21
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\RaporKodlari\Hafta6_RaporKodlari
> node index.js
Geocode data: {
  longitude: 29.06773,
  latitude: 40.182766,
  location: 'Bursa, Bursa, Türkiye'
}
Forecast data: { temperature: 19, feelslike: 19 }
```

Bu kod, "geocode" ve "getForecastData" adlı iki asenkron fonksiyonu bir araya getirir. İlk olarak, "geocode" fonksiyonu "Bursa" gibi bir adresi alır ve bu adresin koordinatlarını (boylam ve enlem) döndürür. Ardından, bu koordinatlar "getForecastData" fonksiyonuna parametre olarak iletilir ve hava durumu verilerini almak için kullanılır. Sonuçlar, hata olup olmadığına bağlı olarak konsola yazdırılır. Bu şekilde, bir fonksiyonun sonucu diğerini etkileyecek şekilde yapılandırılmış olur.

3. Kaynakça

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