

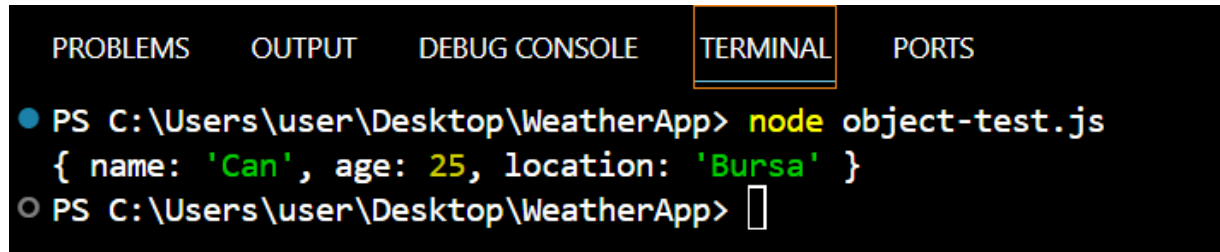
BURSA TEKNİK ÜNİVERSİTESİ NODE.JS ile WEB PROGLAMA DERSİ 7.HAFTA

TEORİ DERSİ RAPORU

Bu hafta ilk olarak object property shorthand ve destructing konularını öğreneceğiz. Kodlarımızı object-test.js dosyasında yazalım.

```
// değişkenler tanımlayalım
const userName = "Can";
const userAge = 25;
// tanımladığımız değişkenleri obje içine özellik olarak verelim.
const user = { // obje tanımlası
  name: userName,
  age: userAge,
  location: "Bursa",
};
console.log(user);
```

Çıktısı:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS C:\Users\user\Desktop\WeatherApp> node object-test.js
  { name: 'Can', age: 25, location: 'Bursa' }
○ PS C:\Users\user\Desktop\WeatherApp> █
```

Bu çıktı bizim beklediğimiz bir çıktıydı şimdi bu kodu shorthande göre düzenleyelim. User objesi içinde özellikleri tanımlarken farklı isimler vermiştik şimdi name kısmını silelim ve tekrar çalıştıralım.

```
// değişkenler tanımlayalım
const userName = "Can";
const userAge = 25;
// tanımladığımız değişkenleri obje içine özellik olarak verelim.
const user = { // obje tanımlası
  userName,
  age: userAge,
  location: "Bursa",
};
console.log(user);
```

Çıktısı:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
PS C:\Users\user\Desktop\WeatherApp>
```

Yine yanı çıktığı elde ettik. Kod kendi içine yukarıda tanımlanan aynı isimli userName değişkeninden bilgiyi alır ve atama yapar. Bu işlemi object property shorthand ile gerçekleştirmiştir. Age için bu özelliği kullanamayız çünkü userAge ile age değişkenlerinin isimleri birbirinden farklıdır. Birebir aynı isimli değişken olmalıdır.

Bu kısımda ise object destructing konusuna bakalım. Yeni bir obje tanımlayalım ve özelliklerini atayalım.

```
//Object destructing

const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined // satış fiyatını tanımlamayalım
};
// obje üzerinden değişkenlere erişim yapalım.
const label = product.label;
const stoce = product.stock;
```

bu bizim eskiden kullandığımız bir kod yazım tarzı ve oldukça da uğraştırıcı iken şimdi Object destructing ile yapalım.

```
//Object destructing
const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined // satış fiyatını tanımlamayalım
};
// obje üzerinden değişkenlere erişim yapalım eski yöntemdir.
//const label = product.label;
//const stoce = product.stock;

const { label, stock } = product; // değişken tanımlama işlemi yapar
mevcut objeden özellikleri çeker
console.log(label);
console.log(stock);
```

Çıktısı:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
● Kırmızı laptop
● 20
○ PS C:\Users\user\Desktop\WeatherApp>
```

Kısacası object property shorthand mevcutta sizde bulunan değişkeni kullanarak eğer o değişkenin ismi objenin özelliklerinden aynı isme sahipse tekrardan: ile değer atamasına gerek yoktur. Destructing ise tam tersidir elimizde bir obje var objenin içindeki özellikleri kullanarak değişken tanımlama işlemidir.

Peki obje içinde bulunmayan bir değişkeni (rating) çekmeye çalışalım.

```
const { label, stock, rating } = product; // değişken tanımlama işlemi
yapar mevcut objeden özellikleri çeker
console.log(label);
console.log(stock);
console.log(rating);
```

Çıktısı:

```
PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
undefined
PS C:\Users\user\Desktop\WeatherApp>
```

Bulunmayan değişkeni undefined olarak ekrana basar.

Burada kullanabileceğimiz bir diğer özellik ise değişken oluştururken ismini de değiştirebiliriz label değişkenin ismini değiştirelim.

```
const { label: etiket, stock, rating } = product; // değişken tanımlama
işlemi yapar mevcut objeden özellikleri çeker
console.log(etiket);
console.log(stock);
console.log(rating);
```

Çıktısı:

```
PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
undefined
PS C:\Users\user\Desktop\WeatherApp>
```

Objenin içinde bulunan özelliğin ismi label idi ama yaptığımız değişiklik ile etikete çevirmiş olduk.

Kodumuz içinde rating bilgisi bulunmuyordu rating ifadesini oluştururken değer atama işine bir bakalım.

```
const { label: etiket, stock, rating = 5 } = product; // değişken tanımlama işlemi yapar mevcut objeden özellikleri çeker
console.log(etiket);
console.log(stock);
console.log(rating);
```

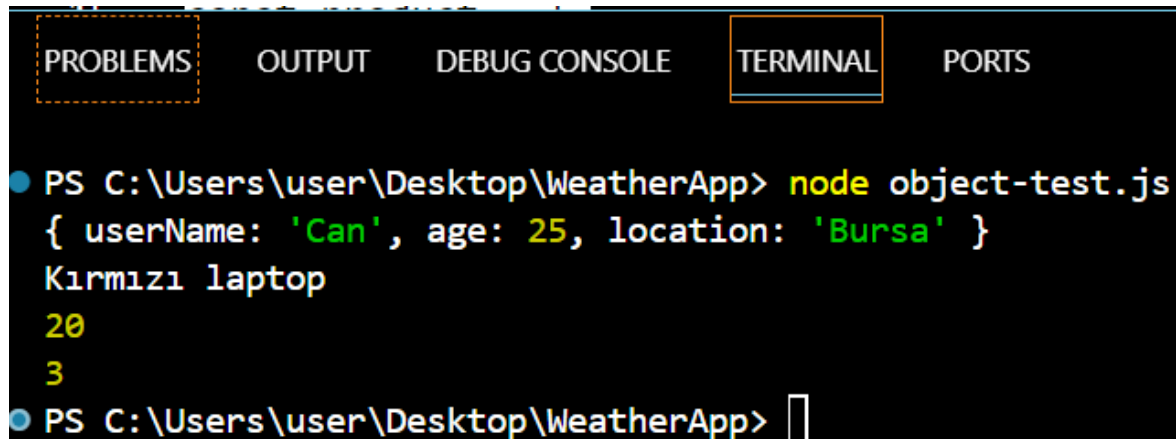
```
PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
5
PS C:\Users\user\Desktop\WeatherApp>
```

Peki rating bilgisi objenin iççinde değişken olarak değer ataması yapılmış olsaydı ve biz de Destructing ile obje tanımlaması yaparken değer ataması yapsaydık eğer geçerli bir eşleşme varsa o zaman Destructing ile tanımlanan değer default değer olarak kabul edileceği için burada tanımlanan değer kullanılmaz obje içinde değer ataması yapılırken atanan değer kullanılır.

```
//Object destructing
const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined , // satış fiyatını tanımlamayalım
  rating: 3
};
// obje üzerinden değişkenlere erişim yapalım eski yöntemdir
```

```
//const label = product.label;
//const stoce = product.stock;

const { label: etiket, stock, rating = 5 } = product; // değişken tanımlama işlemi yapar mevcut objeden özellikleri çeker
console.log(etiket);
console.log(stock);
console.log(rating);
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\user\Desktop\WeatherApp> node object-test.js
  { userName: 'Can', age: 25, location: 'Bursa' }
  Kırmızı laptop
  20
  3
● PS C:\Users\user\Desktop\WeatherApp> 
```

İki tane değer alan bir fonksiyon tanımlayalım.

```
const transaction = (type, myProduct) => {
  const { label } = myProduct; // bu şekilde tüm product bilgisini aktarır ama sadece 1 özellik kullanılacak

};
// fonksiyonu çağıralım
transaction("order", product);
```

Bu kısımda product nesnesinin label ve stock bilgine ihtiyacımız varsa sadece bu bilgileri gönderelim.

```
const transaction = (type, {label, stock}) => {
  //const { label } = myProduct; // bu şekilde tüm product bilgisini aktarır ama sadece 1 özellik kullanılacak
  console.log(type, label, stock);
};
// fonksiyonu çağıralım
transaction("order", product);
```

```

PS C:\Users\user\Desktop\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
3
order Kırmızı laptop 20
PS C:\Users\user\Desktop\WeatherApp>

```

Fonksiyona objenin tamamını geçirmek yerine {} içlerine kullanmak istediğin nesne özelliklerini yazarak yapabileceğimizi gördük. Şimdi bu öğrendiğimiz özellikleri kendi projemizde uygulayalım bunun için app.js dosyamızı açalım.

Kodumuzun en son yazılmış hali şu şekildeydi.

```

const request = require('postman-request');

const geocode= require('./utils/geocode')

const forecast=require("./utils/forecast")

const address=process.argv[2]
if(!address){
  console.log("Lütfen adres bilgisini giriniz!!!")
}
else{
  geocode(address, (error, data)=>{
    if(error){
      return console.log(error)
    }
    forecast(data.latitude, data.longitude, (error, forecastData)=>{
      if(error){
        return console.log(error)
      }
      console.log(data.location)
      console.log(forecastData)
    })
  })
}
}

```

Kodu incelediğimizde data.latitude, data.longitude gibi ifadeler bulunmaktadır. Fonksiyona data nesnesinin tüm özelliklerini göndermek yerine gerekli olan özellikleri göndermeyi deneyelim ve değişkenleri çağırırken data.location ile belirtmemize gerek yok sadece location dememiz yeterlidir.

```

const address=process.argv[2]
if(!address){
  console.log("Lütfen adres bilgisini giriniz!!!")
}

```

```

}
else{
  geocode(address, (error, { longitude, latitude, location
})=>{ //sırların önemi yok isimlerin eşleşmesi önemli

    if(error){
      return console.log(error)
    }
    forecast(latitude,longitude, (error, forecastData)=>{
      if(error){
        return console.log(error)
      }
      console.log(location)
      console.log(forecastData)
    })
  })
}
}

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\user\Desktop\WeatherApp> node app.js bursa
Bursa, Bursa, Türkiye
Hava sıcaklığı: 31 Hissedilen: 29
PS C:\Users\user\Desktop\WeatherApp>

```

Bu kısımda ise forecast.js dosyamıza gidelim ve öğrendiğimiz özellikleri burada uygulayalım. Obje ve özelliği olan kısımlara odaklanalım eskiden bir objenin özelliğini çağırırken obje.adı.özellikadı şeklinde çağırma işlemi yapardık ama artık bu şekilde yapmıyoruz. Response nesnesinin tamamını almamıza gerek yok çünkü biz sadece response'nin sadece body özelliğini kullanıyoruz. Bu işlemler geçmeden önce uygulamamıza başka bir özellik daha ekleyelim diyelim ki terminalden konum bilgisine saçma bir değer girilmiş olsun.

```

PS C:\Users\user\Desktop\WeatherApp> node app.js 646rgr
C:\Users\user\Desktop\WeatherApp\app.js:14
  geocode(address, (error, { longitude, latitude, location })=>{ // sırların önemi yok isimlerin eşleşmesi önemli
                        ^
TypeError: Cannot destructure property 'longitude' of 'undefined' as it is undefined.
    at C:\Users\user\Desktop\WeatherApp\app.js:14:30
    at Request._callback (C:\Users\user\Desktop\WeatherApp\utils\geocode.js:14:4)
    at self.callback (C:\Users\user\Desktop\WeatherApp\node_modules\postman-request\request.js:311:12)
    at Request.emit (node:events:518:28)
    at Request.<anonymous> (C:\Users\user\Desktop\WeatherApp\node_modules\postman-request\request.js:1577:10)

```

Bu şekilde hata alınır bunu önlemeye çalışalım. Kodumuzda geocode fonksiyonunu çağırdığımız zaman bize error ve obje döndürür ama bunlardan yalnızca bir tanesi dolu olmalıdır. Ya error dolu olmalı ya da obje dolu olmalıdır. App.js içinde süslü parantez ile boş atama yapılarak bu hatanın önüne geçilir.

```

else{
  geocode(address, (error, { longitude, latitude, location }= {} )=>{
// sırların önemi yok isimlerin eşleşmesi önemli
    if(error){
      return console.log(error)
    }
  }
}

```

Çıktısı:

```

PS C:\Users\user\Desktop\WeatherApp> node app.js 646rgr
● Belirttiğiniz konum bilgisi bulunamadı
○ PS C:\Users\user\Desktop\WeatherApp> 

```

Sorgumuz yanlış ise hata almamızı önledik. Forecast.js dosyamıza geri gelelim ve az önce bahsettiğimiz değişiklikleri yapalım. Forecast.js dosyasının eski hali bu şekildedir.

```

const request= require("postman-request")

const forecast=(longitude, latitude, callback)=>{
  const url =
    "http://api.weatherstack.com/current?access_key=9c35cb63c25e03adfb73ae1ac35fa762&query="
    + latitude+
    ","+
    longitude
  request({ url: url, json:true }, (error, response) => {

    if (error) {
      callback("Hava durumu servisine bağlantı kurulamadı.", undefined)
    }
    else if(response.body.error){
      callback("Girilen konum bilgisi bulunamadı.",undefined)
    }
    else {
      callback(undefined,"Hava sıcaklığı: "
      +response.body.current.temperature+
      " Hissedilen: "+ response.body.current.feelslike)
    }
  })
}
module.exports=forecast

```


Biz bu kısımda request fonksiyonu içinde response'nin tamamını göndermek yerine sadece body özelliğini göndereceğiz ve özellikleri çağırırken response.body demek yerine sadece body dememiz yeterli olacaktır.

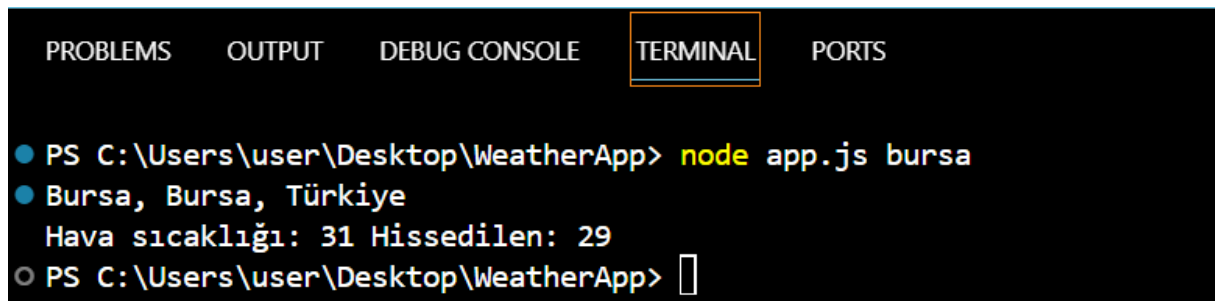
```
const request= require("postman-request")

const forecast=(longitude, latitude, callback)=>{
  const url =
    "http://api.weatherstack.com/current?access_key=9c35cb63c25e03adfb73ae1ac35fa762&query="
    + latitude+
    ","+
    longitude
  request({ url: url, json:true }, (error, {body}) => {

    if (error) {
      callback("Hava durumu servisine bağlantı kurulamadı.", undefined)
    }
    else if(body.error){
      callback("Girilen konum bilgisi bulunamadı.",undefined)
    }
    else {
      callback(undefined,"Hava sıcaklığı: "
      +body.current.temperature+
      " Hissedilen: "+ body.current.feelslike)
    }
  })
}

module.exports=forecast
```

Çıktısı:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\user\Desktop\WeatherApp> node app.js bursa
● Bursa, Bursa, Türkiye
  Hava sıcaklığı: 31 Hissedilen: 29
○ PS C:\Users\user\Desktop\WeatherApp> 
```

Goecode.js dosyamıza gidelim yaptığımız bu işlemleri bu dosyada da gerçekleştirelim. Bu dosyadaki kodların eki hali şu şekildedir:

```
const request=require("postman-request")
```

```

const geocode =(address,callback)=>{
const url =
"https://api.mapbox.com/geocoding/v5/mapbox.places/"+encodeURIComponent
(address) +
".json?access_token=pk.eyJ1Ijoib3psZW1hcnNsYW5ubiIsImEiOiJJbHUzZWYyMDAw
eDN2MmtwZGlvMzQzeG8wIn0.SB2-Lj5qXuozz89bPnWUUQ";

request({ url: url, json: true }, (error, response) => {

  if(error){

    callback("Geocoding servisine bağlanamadı",undefined)
  }
  else if(response.body.features.length===0){
    callback("Belirttiğiniz konum bilgisi bulunamadı",undefined)
  }
  else{
    callback(undefined,{
      longitude: response.body.features[0].center[0],
      latitude: response.body.features[0].center[1],
      location:response.body.features[0].place_name})
  }
});
}
module.exports=geocode

```

Düzenleme yapalım ve response ile ilgili kısımları değiştirelim.

```

const request=require("postman-request")
const geocode =(address,callback)=>{
const url =
"https://api.mapbox.com/geocoding/v5/mapbox.places/"+encodeURIComponent
(address) +
".json?access_token=pk.eyJ1Ijoib3psZW1hcnNsYW5ubiIsImEiOiJJbHUzZWYyMDAw
eDN2MmtwZGlvMzQzeG8wIn0.SB2-Lj5qXuozz89bPnWUUQ";

request({ url: url, json: true }, (error, {body}) => {

  if(error){

    callback("Geocoding servisine bağlanamadı",undefined)
  }
  else if(body.features.length===0){
    callback("Belirttiğiniz konum bilgisi bulunamadı",undefined)
  }
}

```

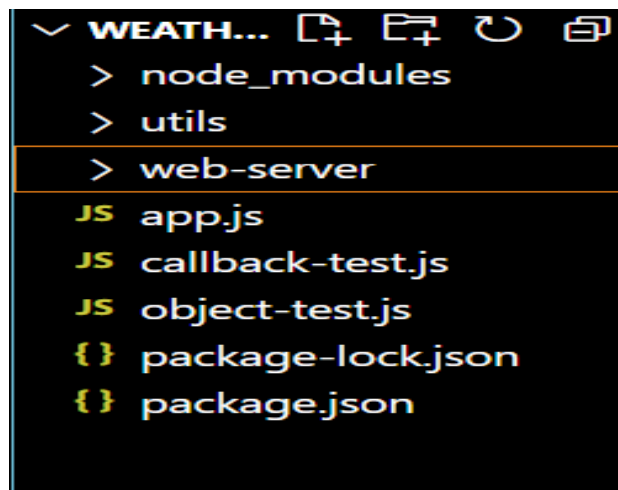
```
else{
  callback(undefined,{
    longitude: body.features[0].center[0],
    latitude: body.features[0].center[1],
    location: body.features[0].place_name})
}
});
}
module.exports=geocode
```

Kaydedip çalıştıralım.

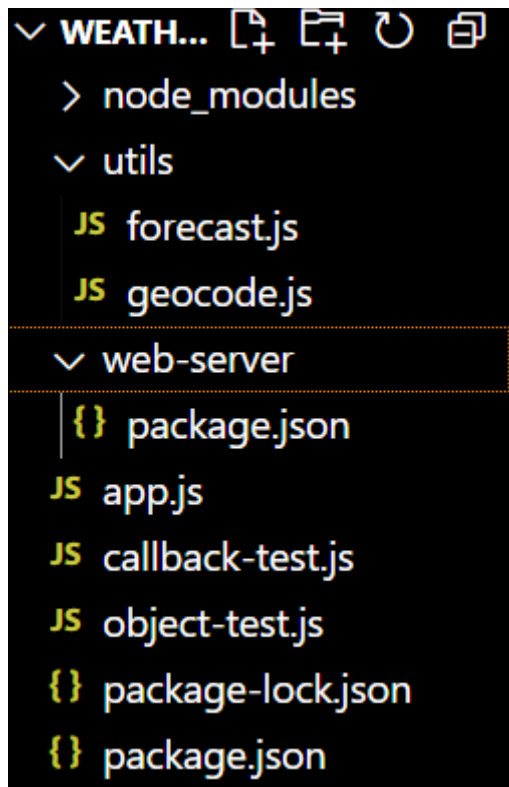
```
PS C:\Users\user\Desktop\WeatherApp> node app.js bursa
Bursa, Bursa, Türkiye
Hava sıcaklığı: 31 Hissedilen: 29
PS C:\Users\user\Desktop\WeatherApp>
```

Express modülünün kullanımına geçelim Express bir frameworkdür. Şu ana kadar kodlarımızı hep terminal üzerinden çalıştırıyorduk artık web üzerinde uygulamamızı çalıştırmaya çalışacağız.

Web-server isimli yeni bir klasör açalım. Express ile kodumuzun içine css ve html kodları ekleyebileceğiz.



Terminal üzerinde web-server klasörüne gidelim bunu için terminale `cd web-server` yazalım. Daha sonra terminale `npm init -y` diyelim ve bu sayede klasör içinde package.js dosyasını dahil eder.



Express kütüphanesini kurmak için terminale `npm i express` yazarız.

```
PS C:\Users\user\Desktop\WeatherApp\web-server> npm i express

added 64 packages, and audited 65 packages in 5s

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

found 0 vulnerabilities
PS C:\Users\user\Desktop\WeatherApp\web-server> 
```

```
web-server > {} package.json > ...
3 | "version": "1.0.0",
4 | "description": "",
5 | "main": "index.js",
  | ▸ Debug
6 | "scripts": {
7 |   | "test": "echo \"Error: no test specified\" && exit 1"
8 | },
9 | "keywords": [],
10 | "author": "",
11 | "license": "ISC",
12 | "dependencies": {
13 |   | "express": "^4.19.2"
14 | }
15 | }
16 |
```

Gördüğümüz üzere package.json dosyasının içinde dependencies kısmında Express kütüphanesini dahil etmiştir. Source dosyalarını ekleyebilmek için web-server klasörü içinde src klasörü oluşturalım ve bu src klasörü içinde app.js dosyası oluşturalım.

```
✓ web-server
  > node_modules
    ✓ src
      | JS app.js
      | {} package-lock.json
      | {} package.json
```

App.js içine Express modülünü dahil edelim.

```
// express modülünü dahil edelim

const express = require("express")
const app = express();
```

biraz web sitemizin nasıl olması gerektiğini düşünelim ve web sitemizin app.com adında bir anasayfamız olsun. Bu web sitemizin içinde app.com/help ve app.com/about sayfalarımız bulunmalıdır. App.get fonksiyonumuz ile oluşturduk.

```
// express modülünü dahil edelim

const express = require("express")
const app = express(); // Constructors çağırma işlemidir.
```

```
// app.com, app.com/help ve app.com/about sayfalarını oluşturma  
başlayalım.  
  
app.get("", (req, res) => { //home page olmasını "", ifadesi sağlar.  
    res.send("Hello Express"); // isteğe karşı gelen cevabı oluşturma  
işlemi  
});
```

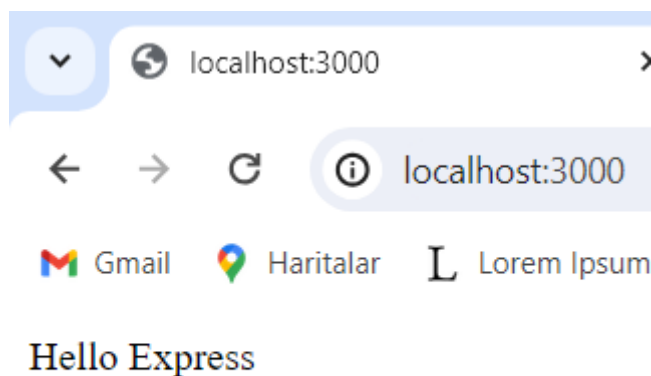
Normalde http sayfaları 80 numaralı porttan çalışır ama bunu istersek app.listen ile değiştirebiliriz. Bunun için kodumuzun devamına şu kısımları ekledik.

```
app.listen(3000, () => {  
    console.log("Sunucu 3000 portunu dinliyor..");  
});
```

İlk olarak terminalden kodumuzu çalıştıralım.

```
PS C:\Users\user\Desktop\WeatherApp\web-server> cd src  
PS C:\Users\user\Desktop\WeatherApp\web-server\src> node app.js  
Sunucu 3000 portunu dinliyor..  
█
```

Daha sonra tarayıcı açalım ve arama kısmına localhost:3000 ifadesini yazalım.



Şimdi help ve about sayfalarını app.get ifadeleri ile oluşturalım.

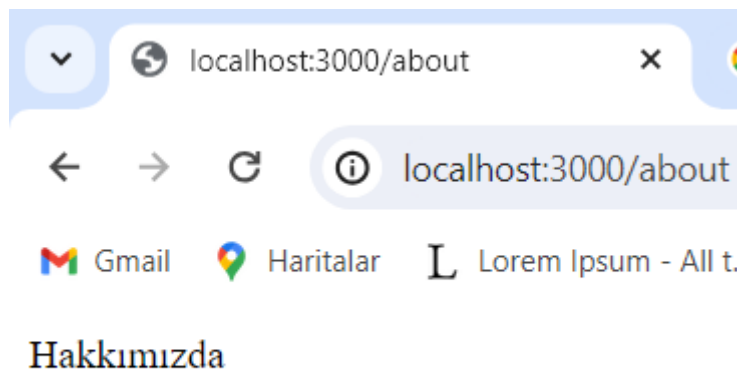
```
// express modülünü dahil edelim  
  
const express = require("express")  
const app = express(); // Constructors çağırma işlemidir.
```

```
// app.com, app.com/help ve app.com/about sayfalarını oluşturmaya
başlayalım.

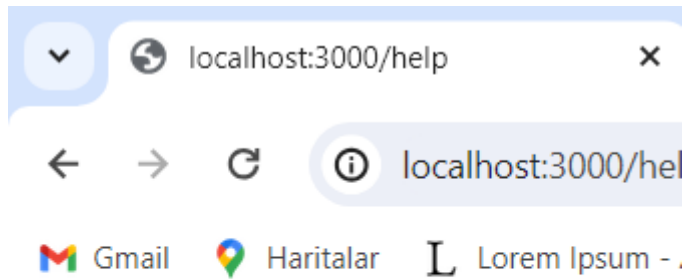
app.get('', (req, res) => { //home page olmasını "", ifadesi sağlar.
    res.send("Hello Express"); // isteğe karşı gelen cevabı oluşturma
    işlemi
});
// help sayfasını oluşturalım
app.get("/help", (req, res) => {
    res.send("Yardım sayfasına hoşgeldiniz.");
});
// about sayfasını oluşturalım
app.get("/about", (req, res) => {
    res.send("Hakkımızda");
});

// 3000. porttan dinleme işlemi yapalım.
app.listen(3000, () => {
    console.log("Sunucu 3000 portunu dinliyor..");
});
```

Kodları kaydedelim sunucuyu kapatıp tekrar açmamız lazım kapatmak için ctrl+c ile kapatırız daha sonra terminale node app.js diyerek sunucumuzu tekrardan başlatırız. Daha sonra tarayıcıda arama kısmına localhost:3000/about yazalım.



Arama kısmına localhost:3000/help yazalım.



Yardım sayfasına hoşgeldiniz.

Uygulama geliştirme süresi boyunca sunucumuzu sürekli açıp kapatmak yerine global olarak nodemon modülünü kuralım bunun için terminale npm i -g nodemon yazalım.

```
PS C:\Users\user\Desktop\WeatherApp\web-server\src> npm i -g nodemon

changed 33 packages in 5s

4 packages are looking for funding
  run `npm fund` for details
PS C:\Users\user\Desktop\WeatherApp\web-server\src>
```

Bu sayede kodda değişiklik yaptığımızda kodu sadece kaydedip çalıştırmamız yeterli olacaktır.

Nodemon u komut gibi kullanmaya başlayacağız çünkü -g ile global modda kurduk.

```
run npm fund for details
PS C:\Users\user\Desktop\WeatherApp\web-server\src> nodemon app.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Sunucu 3000 portunu dinliyor..
```

Kodumuzda ufak bir değişiklik yapalım ve ctrl-s ile kaydedelim kendisi yenilme işlemi yapar. About sayfası açıldığında yazan ifadeyi değiştirelim ve kaydedelim.

```
app.get("/about", (req, res) => {
  res.send("Hakkımızda sayfasına hoşgeldiniz");
});
```

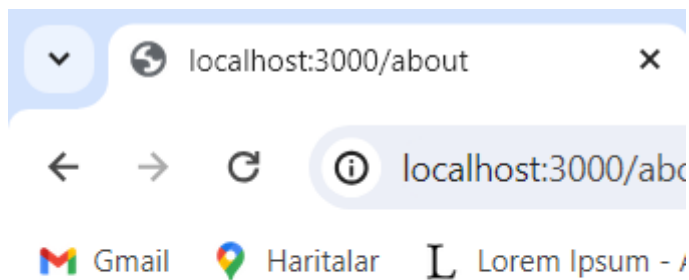
Terminalde yenileme işlemi meydana gelir.


```

PS C:\Users\user\Desktop\WeatherApp\web-server\src> nodemon app.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Sunucu 3000 portunu dinliyor..
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Sunucu 3000 portunu dinliyor..

```

Tarayıcıda ise değişiklik yansır.



Hakkımızda sayfasına hoşgeldiniz

Biz hava durumu uygulaması geliştirdiğimiz için bir tane de hava durumu bilgisi içeren bir sayfa oluşturalım.

```

// hava durumu bilgisi içeren sayfa
app.get("/weather", (req, res) => {
  res.send("Hava durumu bilgisi");
});

```

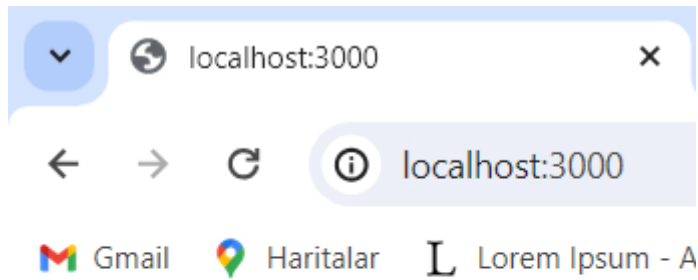
Daha sonra Express bize html kodu yazmamızı ve bunu server etmemize izin vermektedir o zaman anasayfamızı değiştirelim eskiden Hello Express yazıyordu onun yerine html kodu ekleyelim.

```

app.get('/', (req, res) => { //home page olmasını "", ifadesi sağlar.
  res.send("<h1>Hava Durumu</h1>"); // isteğe karşı gelen cevabı
  oluşturma işlemi
});

```

<h1> bir html etiketidir ve başlık formatlama etiketidir. Kodu kaydedip çalıştıralım ve tarayıcıda çıktısına bakalım.

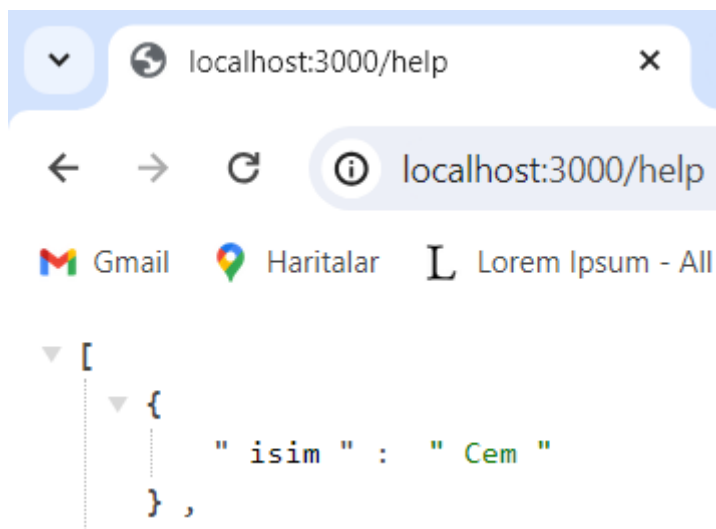


Hava Durumu

Bu kısımda ise json formatında bir response işlemi nasıl gerçekleştirilir ona bakalım. Helps sayfasına gidelim ve bazı değişiklikler yapalım. Obje arrayi döndürelim.

```
// help sayfasını oluşturalım
app.get("/help", (req, res) => {
  res.send([{ name: "Cem" }]);
});
```

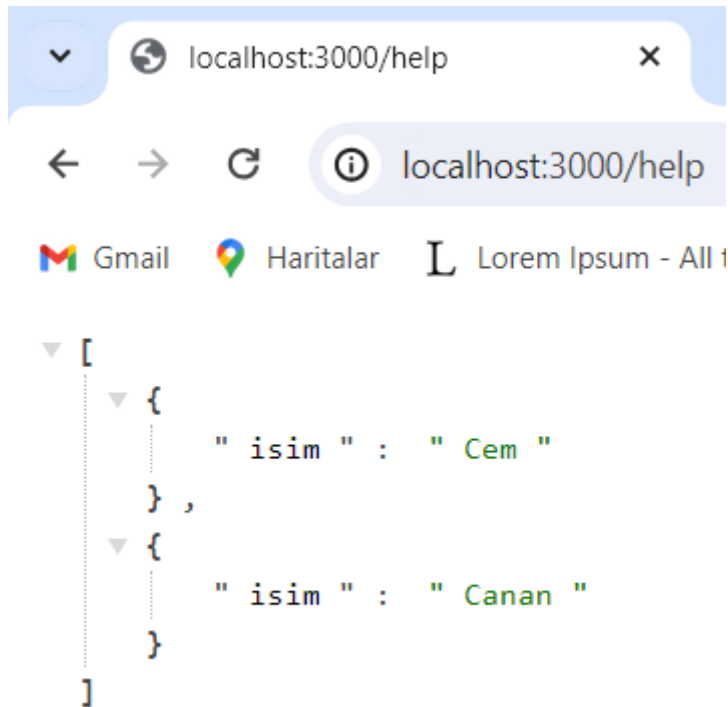
Kaydedelim ve help sayfasına gidelim.



İkinci bir eleman ekleyelim.

```
// help sayfasını oluşturalım
app.get("/help", (req, res) => {
  res.send([{ name: "Cem" }, { name: "Canan" }]);
});
```

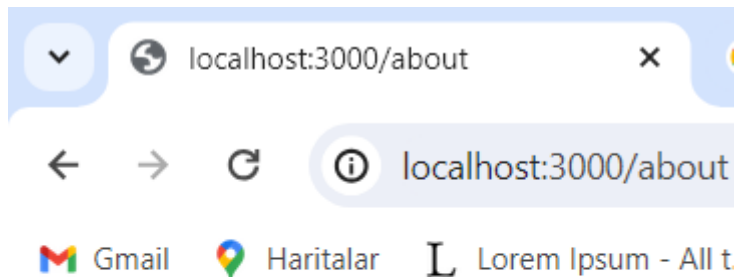
Tarayıcıdaki çıktısı:



About sayfasına da html etiketi ekleyelim.

```
// about sayfasını oluştalım
app.get("/about", (req, res) => {
  res.send("<h1>Hakkımızda sayfası</h1>");
});
```

Tarayıcıdaki çıktısı:



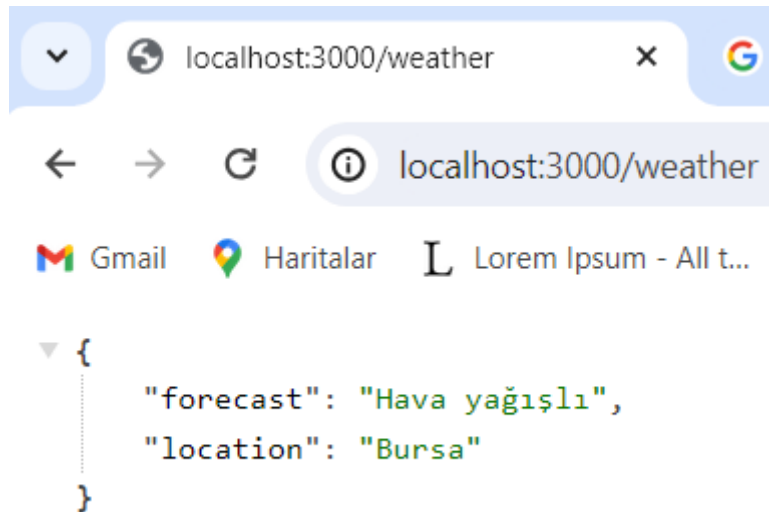
Hakkımızda sayfası

Weather sayfasına geelim ve burada json objesi oluşturup iki özellik verelim hava durumu tahmin bilgisi ve lokasyon bilgisi verelim bunu şu an statik olarak oluşturuyoruz daha sonra ise api ile çekmeye bakacağız.

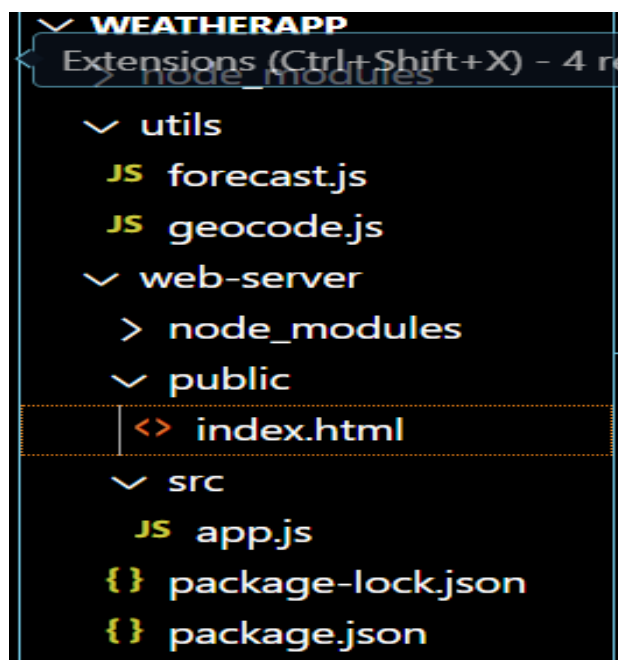
```
// hava durumu bilgisi içren sayfa
```

```
app.get("/weather", (req, res) => {  
  //weather  
  res.send({  
    forecast: "Hava yağışlı",  
    location: "Bursa",});  
});
```

Tarayıcıdaki çıktısı:



Html kodlarını src klasörü içinde yazıp Express ile kodun konumu gösterip dahil edeceğiz. Bunun için web-server klasörü içinde public isminde yeni bir klasör açalım. Bu klasör içinde index.html dosyası açalım.



Bu sayfaya html kodu yazalım. Sayfa içinde ! yazıp tab tuşuna basarsanız klasik bir html kodu yazar hadi deneyelim.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

Daha sonra body blokları arasına şu kodu ekleyelim.

```
<h1>BU statik bir web sayfasıdır.</h1>
```

Bir sonraki adımda src içindeki app.js dosyasına gitmemiz gerekli ama public klasörün konum bilgisini öğrenmemiz gereklidir. Bunun için app.js içine şu kodları ekleyim:

```
// express modülünü dahil edelim

const express = require("express")
const app = express(); // Constructors çağırma işlemidir.

// dosyanın konum bilgisini öğrenebilmek için.
console.log(__dirname) // src ye kadar olan kısmın yolunu getirir
console.log(__filename) // app.js bilgisini de getirir.
```

Çıktısı:

```
Sunucu 3000 portunu dinliyor..
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
C:\Users\user\Desktop\WeatherApp\web-server\src
C:\Users\user\Desktop\WeatherApp\web-server\src\app.js
Sunucu 3000 portunu dinliyor..
█
```

Path modülü ile uygulamamızı geliştirelim. Path bir core modüldür. App.js içine

```
// express modülünü dahil edelim

const express = require("express")
const app = express(); // Constructors çağırma işlemidir.
// path modülünü dahil edelim
const path = require("path");
```

Daha sonra şu kodu ekleyelim

```
console.log(__dirname);  
console.log(path.join(__dirname, "../public"));
```

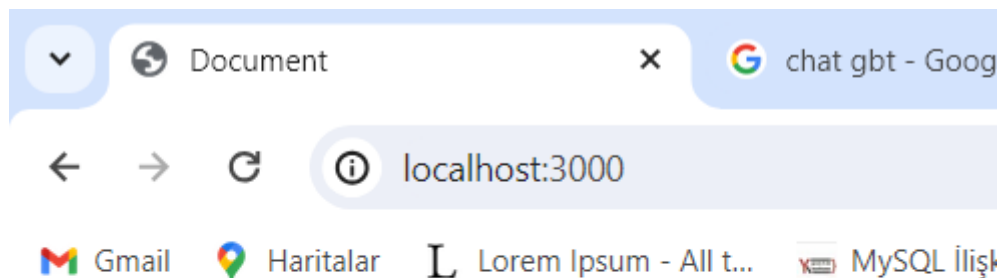
public klasörünün konum bilgisini bulmaya çalışıyoruz bunun için bir konum yukarı çık bu işlemi ../ sağladık artık web-server klasöründeyiz daha sonra web-server klasörü içindeki public klasörüne geçmiş olduk. Path.join aldığımız klasör bilgisini al __dirname ile daha sonra public bilgisini ekle şu an src kalsörü içinde bir üst konuma geç yaptığımız işlem kısaca budur ekran çıktısına bakalım.

```
C:\Users\user\Desktop\WeatherApp\web-server\src  
C:\Users\user\Desktop\WeatherApp\web-server\public  
Sunucu 3000 portunu dinliyor..  
|
```

Bu kısımda path.join ile elde ettiğimiz klasör bilgisini bir değişkene atayalım ve az önce yazdığımız kodları yorum satırına alalım.

```
const publicDirectoryPath = path.join(__dirname, "../public")  
app.use(express.static(publicDirectoryPath))
```

app.use bu klasörü kullan anlamına geliyor. Herkese açık olan kalsörün konumu tespit edip ardında gönderme işlemi yaptık.



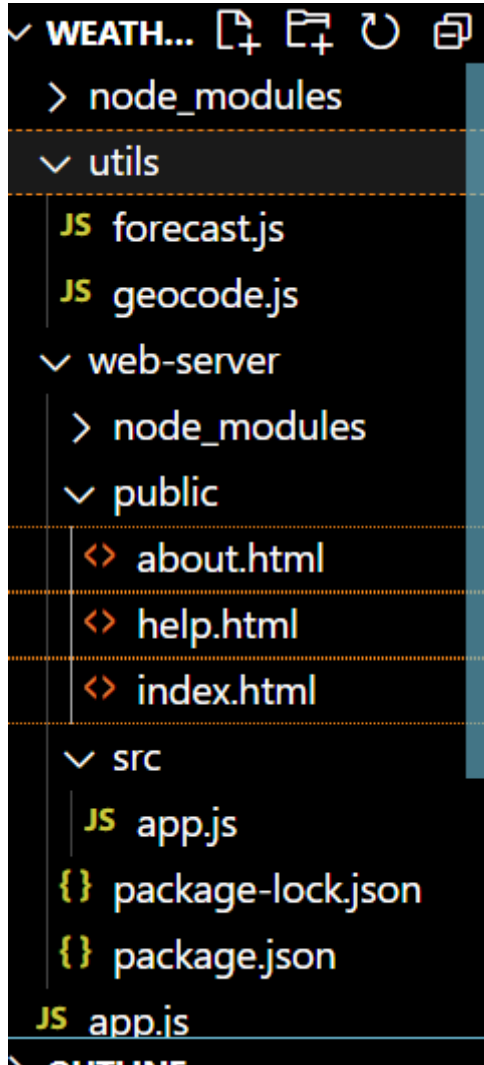
BU statik bir web sayfasıdır.

Eskiden bizim home page sayfamız şu idi:

```
app.get('/', (req, res) => {  
  res.send('<h1>Hava Durumu</h1>')  
})
```

Artık bizim home page index.html olmuştur. Bu işlemi app.use ile yaptık ve index.html sayfası özel isimli bir sayfadır ana sayfa olarak açılır. Aslında yaptığımız bu işlemler help, about ve weather birer rotalar ve router handler olarak isimlendirilirler.

Bir sonraki aşamada ise artık bu routerlara ihtiyacımız yok web sayfaları oluşturalım. Public klasörü içinde help.html ve about.html sayfalarını oluşturalım ve içlerine index.html sayfasından kodları kopyala yapıştır yap.



Hepl.html içinde:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Bu bir yardım sayfasıdır.</h1>
</body>
```

```
</html>
```

About.html içinde:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Bu bir hakkında sayfasıdır.</h1>
</body>
</html>
```

Ve app.js içinde şu kısımları artık kapatabiliriz bunları yorum satırına alalım ihtiyacımız kalmadı.

```
// help sayfasını oluşturalım
// app.get("/help", (req, res) => {
//   res.send([ { name: "Cem" }, { name: "Canan" } ]);
// });

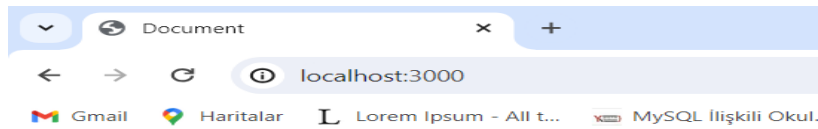
// // about sayfasını oluşturalım
// app.get("/about", (req, res) => {
//   res.send("<h1>Hakkımızda sayfası</h1>");
// });
```

Daha sonra public klasörü içinde css isminde klasör oluşturalım ve css klasörü içinde styles.css isminde bir dosya açalım içine şu kodları yazalım.

```
/* web sayfasında h1 etiketi varsa rengi pembe olarak gösterilsin
anlamına gelir. */

h1 {
  color: pink;
}
```

index.html dosyasına geçelim ve head kısmında bu h1 etiketi renk özelliğini dahil edelim. Kaydedelim ve tarayıcı da çıktısına bir bakalım.



BU statik bir web sayfasıdır.

Uyguladığımız bu adımları about ve help sayfaları içinde yapalım.

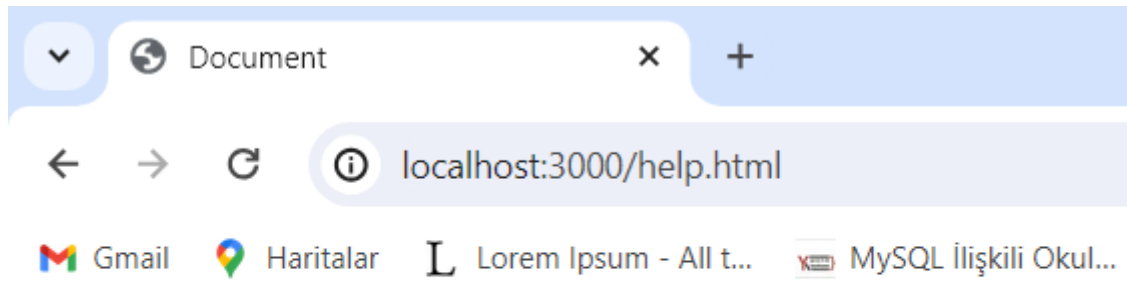
Help.html sayfasında yaz kodlar:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="./css/styles.css" />
  <title>Document</title>
</head>
<body>
  <h1>Bu bir yardım sayfasıdır.</h1>
</body>
</html>
```

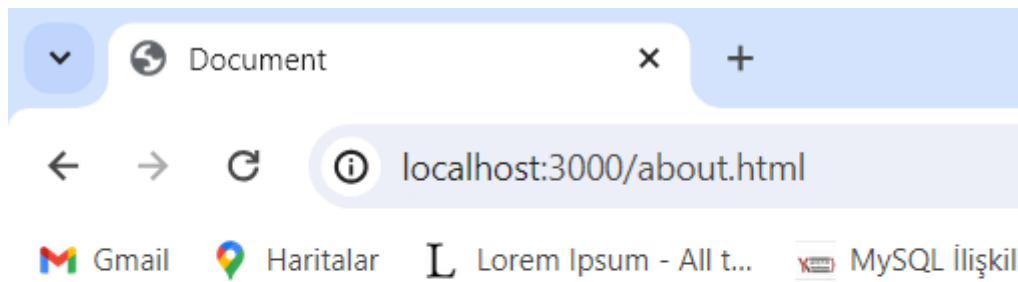
About.html sayfasında yazan kodlar:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="./css/styles.css" />
  <title>Document</title>
</head>
<body>
  <h1>Bu bir hakkında sayfasıdır.</h1>
</body>
</html>
```

Her iki sayfanın da tarayıcıda çıktıları:

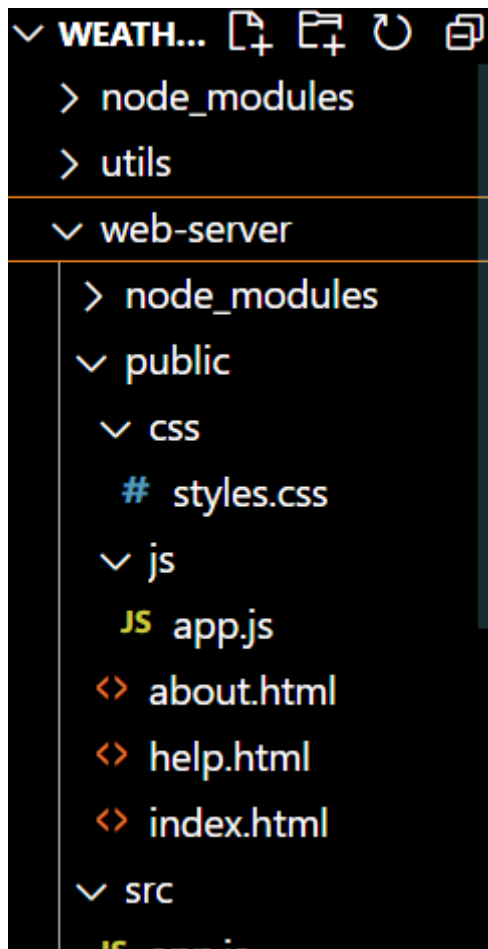


Bu bir yardım sayfasıdır.



Bu bir hakkında sayfasıdır.

Daha sonra public klasörü altında yeni klasör oluşturalım ve adına da js diyelim bunun içinde de app.js isminde dosya oluşturalım.



App.js içine şu kodu yazdık.

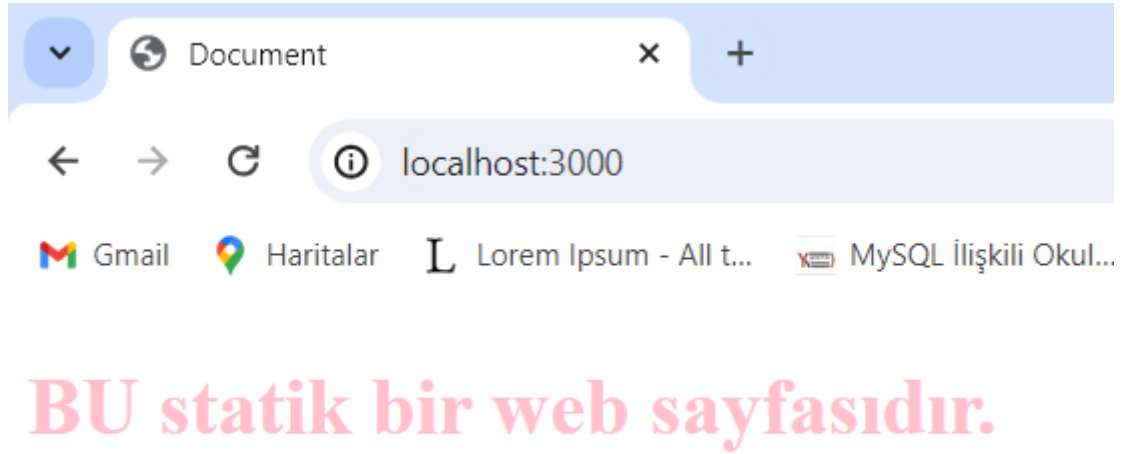
```
console.log("Bu bir js dosyasıdır ve client tabanlı çalışır");
```

index.html ye gidelim ve burada css linkinin hemen altında konum bilgisini vererek bu dosyayı dahil edelim.

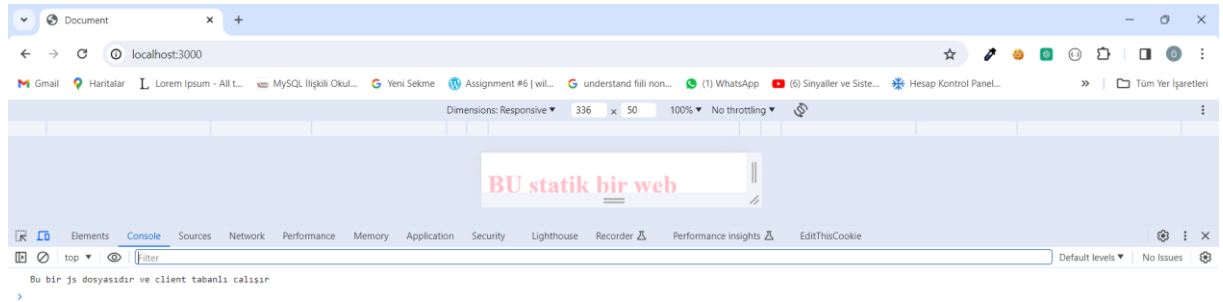
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <!-- href den sonra konum bilgisi gelir -->
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="./css/styles.css" />
  <script src="/js/app.js"></script>

  <title>Document</title>
</head>
<body>
  <h1>BU statik bir web sayfasıdır.</h1>
</body>
</html>
```

Kaydedelim ve anasayfaya gidelim.

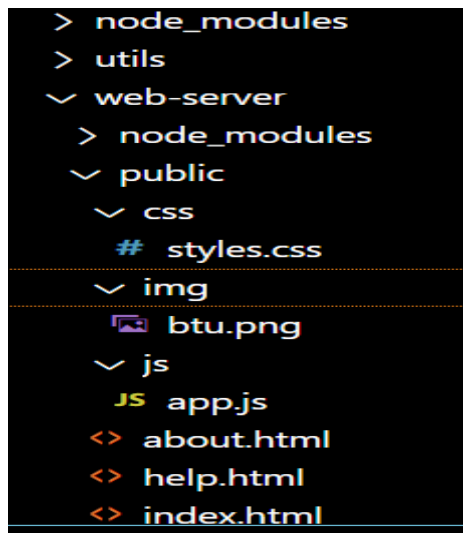


Bu sayfada sağa tıklayıp incele dedikten sonra console kısmında bizi bir yazı karşılayacaktır.



Js dosyanın client tarafında çalışması demek client tarafında disable edilebilir olmasıdır.

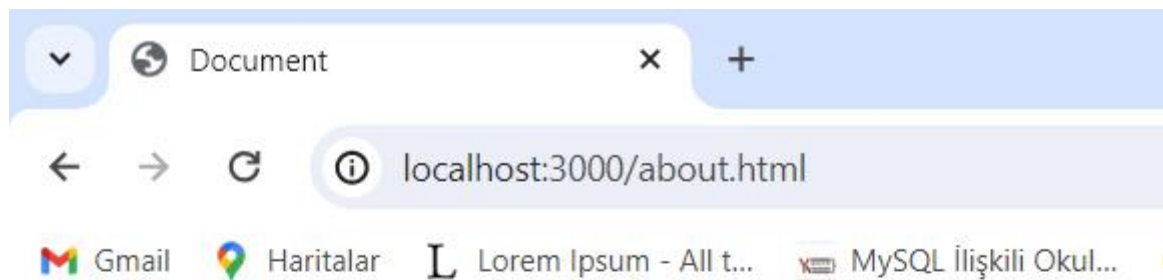
Resim nasıl kullanılıyor bunu öğrenelim. Tarayıcıya btu logo yazalım. Public klasörü altında img isimli bir klasör açalım ve belirlediğimiz resmi buraya taşıyalım.



Daha sonra about sayfasına gidelim ve resmi `` bu sayfaya yükleyelim.

About.html sayfasında yazan kodlar:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="./css/styles.css" />
  
  <title>Document</title>
</head>
<body>
  <h1>Bu bir hakkında sayfasıdır.</h1>
</body>
</html>
```

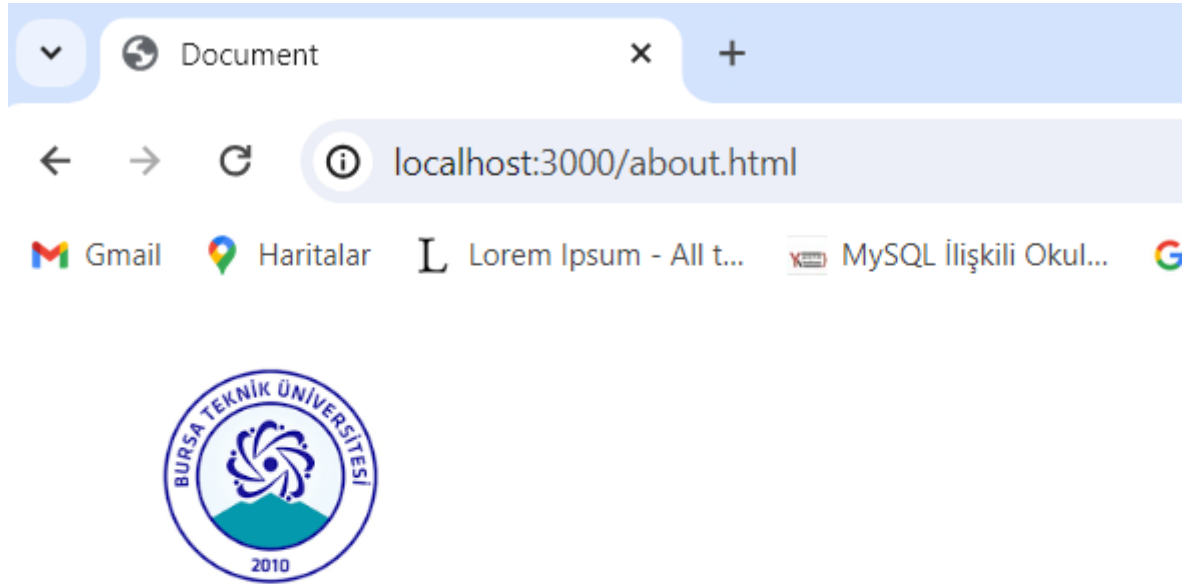


Bu bir hakkında sayfasıdır.

Resim çok büyük bunu styles.css dosyası içerisinde resim boyutunu düzenleyelim.

```
/* web sayfasında h1 etiketi varsa rengi pembe olarak gösterilsin anlamına gelir. */  
  
h1 {  
  color: pink;  
}  
/* resim boyutunu ayarlar.  
img {  
  width: 200px;  
  margin-top: 10px;  
}
```

Resmin yeni görüntüsü:



Bazı web sayfaları statiktir ve değişmez iken bazı web sayfaları dinamiktir ve içerikleri değişebilir.

Dinamik web sayfası yapabilmek için bazı araçlar kullanılır handlerbars bunlardan bazılarıdır. Hbs modülünü yüklememiz lazım bunun için sunucuyu durduralım. Terminal de npm i hbs yazalım ve modülü dahil edelim.

```
PS C:\Users\user\Desktop\WeatherApp\web-server\src> npm i hbs

added 9 packages, and audited 74 packages in 7s

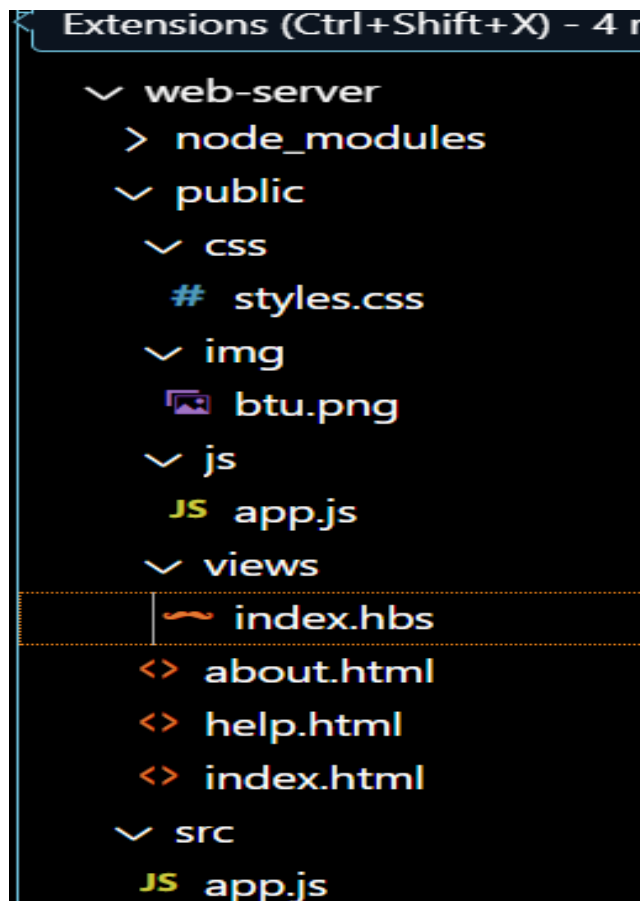
13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\user\Desktop\WeatherApp\web-server\src> |
```

Bu modülü kullanabilmek için src içindeki app.js de modülü eklememiz lazım. App.use den hemen önce hemen şu kodu ekleyelim.

```
app.set('view engine', 'hbs')
```

Kaydedelim ve public klasörü altında views isminde yeni bir klasör oluşturalım ve bunun içinde index.hbs isminde yeni bir dosya açalım.



Index.html dosyasını açalım oradaki kodu kopyalayıp index.hbs içine yapıştıralım. Index.html sayfasını artık kullanmayacağız. Dolayısı ile bu sayfayı silelim.

```

web-server > public > views > index.hbs > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <!-- href den sonra konum bilgisi gelir -->
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="./css/styles.css" />
8   <script src="/js/app.js"></script>
9
10  <title>Document</title>
11 </head>
12 <body>
13 | <h1>BU statik bir web sayfasıdır.</h1>
14 </body>
15 </html>

```

Peki biz index.hbs dosyasına nasıl erişim sağlayacağız? Bunun için src klasörü içindeki app.js içinde bazı eklemeler yapalım.

```

const publicDirectoryPath = path.join(__dirname, "../public")
app.use(express.static(publicDirectoryPath))

app.get("", (req, res) => {
  // res.send("<h1>Hava Durumu</h1>");
  res.render("index"); // res.render ifadesi ile index.hbs sayfasına ulaş
});

```

Sunucumuzu tekrardan çalıştıralım.

```

found 0 vulnerabilities
PS C:\Users\user\Desktop\WeatherApp\web-server\src> nodemon app.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Sunucu 3000 portunu dinliyor..

```

Home page e gidelim.

Bu şekilde bir hata aldık bunu önlemek için bazı ifadeler ekleriz.

```
const express = require("express")
const app = express();
const path = require("path");

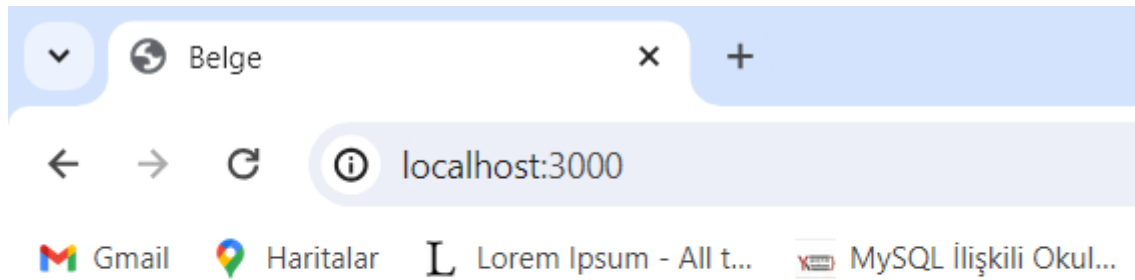
app.set('view engine', 'hbs')

// hata almayı önlemek için bu ifadeyi yazalım
const viewsPath = path.join(__dirname, "../public/views");
app.set("views", viewsPath);

const publicDirectoryPath = path.join(__dirname, "../public")
app.use(express.static(publicDirectoryPath))

app.get("", (req, res) => {
  res.render("index");
});

app.get('/', (req, res) => {
  res.send("<h1>Hava Durumu</h1>");
});
app.get("/weather", (req, res) => {
  //weather
  res.send({
    forecast: "Hava yağışlı",
    location: "Bursa",});
});
app.listen(3000, () => {
  console.log("Sunucu 3000 portunu dinliyor..");
});
```



BU statik bir web sayfasıdır.

Yaptığımız işlem index.hbs ile içeriği dinamik olarak tasarlayacağız. Index.hbs dosyasında title kısmını dinamik hale getirelim.

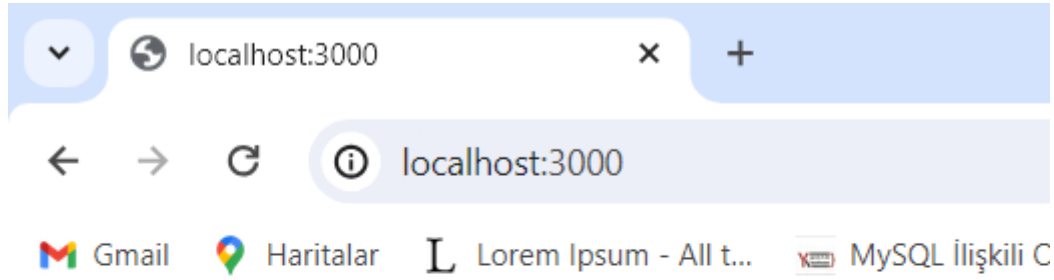
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <!-- href den sonra konum bilgisi gelir -->
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="./css/styles.css" />
  <script src="/js/app.js"></script>
</head>
<body>
  {{!-- dinamik hale getirmek için --}}

  <h1>{{title}}</h1>
  <p>{{name}} tarafından geliştirilmiştir.</p>
  {{!-- <h1>BU statik bir web sayfasıdır.</h1> --}}
</body>
</html>
```

Daha sonra src klasörü altındaki app.js dosyasından gönderdiğimiz title ve name bilgisini alıp anasayfamızda yazdırabileceğiz. Şimdi bilgi gönderme kısmına bakalım.

```
app.get('', (req, res) => { //home page olmasını "", ifadesi sağlar.
  // res.send("<h1>Hava Durumu</h1>");// isteğe karşı gelen cevabı
  oluşturma işlemi
  res.render("index",{
    title: "Hava Durumu Uygulaması",
    name: "Özlem Arslan",
  }); // res.render ifadesi ile index.hbs sayfasına ulaş

});
```



Hava Durumu Uygulaması

özlem arslan tarafından geliştirilmiştir.

Daha sonra views klasörü altında about.hbs ve help.hbs dosyaları açalım. About.html ve help.html dosyaları ile işimiz yok silebiliriz.

Aynı işlemleri about ve help sayfalarında uygulayım bunun için about.hbs dosyasında şu kodlar yer almaktadır.

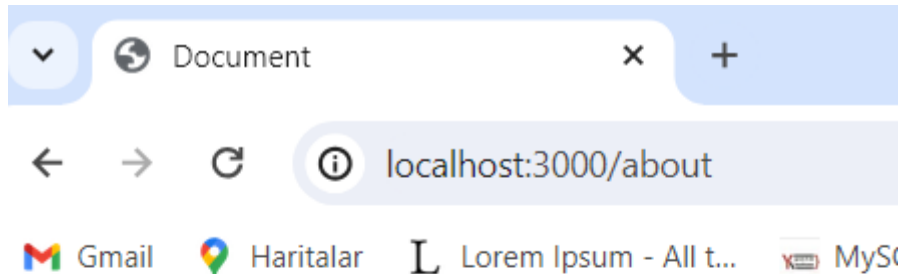
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <link rel="stylesheet" href="../css/styles.css" />
  <title>Document</title>
</head>
<body>
  <h1> {{title}} </h1>
  
  <p>{{name}} tarafından geliştirilmiştir.</p>
</body>
</html>
```

Daha sonra src klasörü altındaki app.js içinde şu kodu ekleyelim.

```
app.get("/about", (req, res) => {
  //res.send("<h1>Hakkımızda</h1>");
  res.render("about", {
    title: "Hakkımızda",
```

```
name: "Özlem ARSLAN",  
});  
});
```

Tarayıcıdaki çıktısı:



Hakkımızda



Özlem ARSLAN tarafından geliştirilmiştir.

Help.hbs dosyasında ise şu kodlar yer almaktadır.

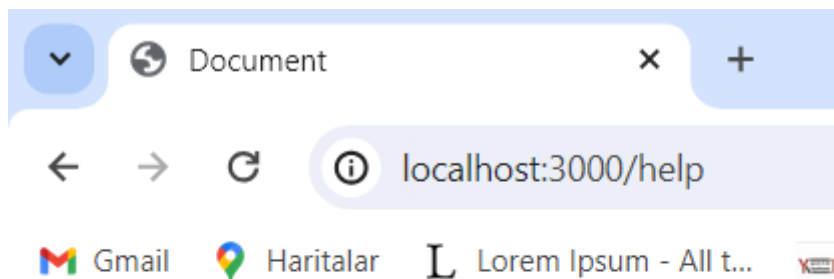
```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
  <title>Document</title>  
  <link rel="stylesheet" href="../css/styles.css" />  
</head>  
<body>  
  <h1> {{title}} </h1>  
  <p> {{helpText}} </p>  
</body>
```

```
</html>
```

App.js içinde şu kodu ekledik.

```
app.get("/help", (req, res) => {  
  
  res.render("help", {  
    title: "Yardım Sayfasıdır",  
    helpText: "Bu bir deneme yazısıdır."  
  });  
});
```

Çıktısı:



Yardım Sayfasıdır

Bu bir deneme yazısıdır.