

NodeJs Web Programlama Hafta-7 Raporu

Bugünkü derste geçen haftaki koddan devam ettik. Amacımız hava durumu uygulaması geliştirmek. Şu ana kadarki yaptığımız uygulamaları konsol üzerinden yapmıştık bu ders web üzerine geçtik.

Bugünkü öğreneceğimiz yeni konu: object property shorthand, destructuring. Bu iki yeni özelliğin object-test.js dosyasında nasıl uygulandığını gördük.

```
//Object property shorthand  
//Destructuring
```

```
const userName = "Can";  
const userAge = 25;
```

```
const user = {  
  name: userName,  
  age: userAge,  
  location: "Bursa",  
};
```

```
console.user(user);
```

İlk olarak kullanıcı adı ve yaş bilgisini girdik. Ardından user adında obje tanımlayıp atamasını gerçekleştirdik. Fakat tanımlamadığımız location bilgisini de girdik. Ardından konsola yazdırdık ve çıktı olarak şunu aldık:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js  
  { name: 'Can', age: 25, location: 'Bursa' }  
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

Ardından kod tanımlamada obje içerisinde birkaç değişikliğe gittik.

```
//Object property shorthand  
//Destructuring
```

```
const userName = "Can";  
const userAge = 25;
```

```
const user = {  
  userName,  
  age: userAge,  
  location: "Bursa",  
};
```

```
console.log(user);
```

Kodu yine çalıştırdık ve çıktı olarak:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
  { userName: 'Can', age: 25, location: 'Bursa' }
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> 
```

Yine aynı sonucu aldık. Bunu sağlayan <object property shorthand> dır. Aynı işlemi age değişkeninde uygulayamayız çünkü tanımladığımız değişken adı ile (userAge) aynı isimde olmadığından ötürü hata verecektir.

Ardından başka bir örnek daha yaptık:

```
//Object destructing

const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined,
};
```

Bu kodda product adındaki obje içerisinde değişken tanımladık ve değer verdik. Bu kod içerisinde label ve stock bilgilerini çekmek için aşağıdaki yöntemi kullanabilirdik.

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

Fakat bu şekilde yapmak yerine daha basit olan aşağıdaki yöntemi kullandık:

```
const { label, stock } = product;
```

Bu kod yukarıdaki işlemle aynıdır. Bu kullanıma <object destructing> denir.

Ardından kodumuza ekledik ve şu hali aldı:

```
//Object destructing

const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined,
};

const { label, stock } = product;

console.log(label);
console.log(stock);
```

Çıktı olarak şunu aldık:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

Ardından olmayan değişken olan reting bilgisini de ekleyip çalıştırdık.

```
25
26 const { label, stock, reting } = product;
27
28 console.log(label);
29 console.log(stock);
30 console.log(reting);
31
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
undefined
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

Herhangi bir hata almadık.

Değişken tanımlanırken aynı zamanda değişken ismi değiştirebilir miyiz ona baktık:

```
const { label: etiket, stock, reting } = product;

console.log(etiket);
console.log(stock);
console.log(reting);
```

Çıktısı da şu şekilde oldu:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
undefined
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
● 20
undefined
PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

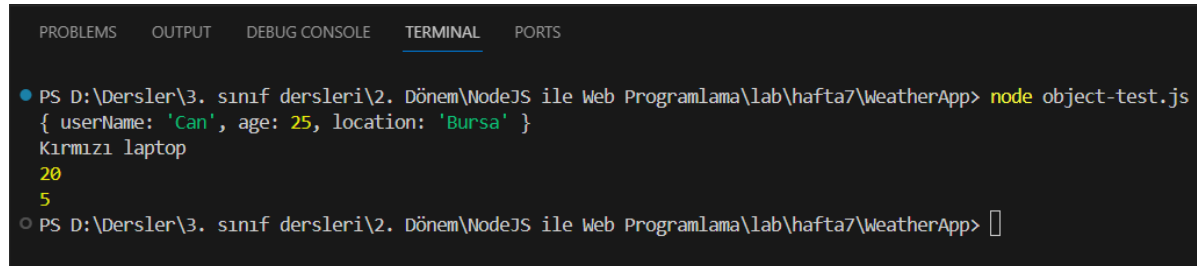
Kodumuz hala başarılı bir şekilde çalışıyor.

Object destructing olan kod parçasında değişken ismi değiştirebildiğimiz gibi değer de verebiliriz:

```
const { label: etiket, stock, rating = 5 } = product;

console.log(etiket);
console.log(stock);
console.log(rating);
```

Sonuç olarak çıktımız:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
5
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

Object destructing içerisinde verdiğimiz değişken değeri çıktımızda da doğru olarak verildi.

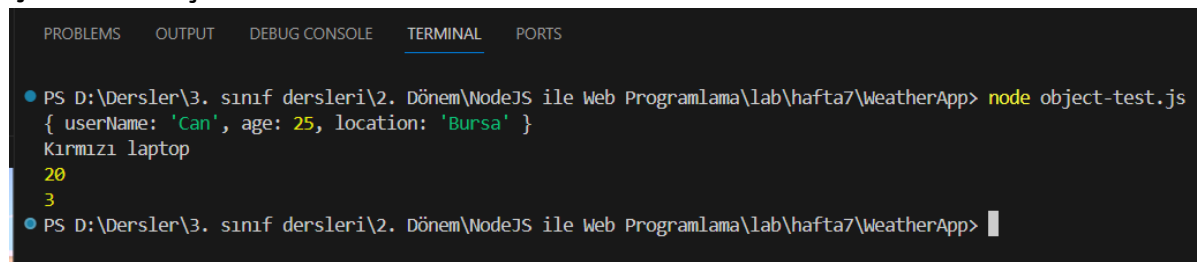
Ardından hem product içerisinde tanımlayıp değer verip hem de object destructing içerisinde değer verdiğimiz değişkeni yazdırdık.

```
const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined,
  rating: 3,
};

const { label: etiket, stock, rating = 5 } = product;

console.log(etiket);
console.log(stock);
console.log(rating);
```

Çıktı olarak şu sonucu verdi:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
{ userName: 'Can', age: 25, location: 'Bursa' }
Kırmızı laptop
20
3
• PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> █
```

Sonuç olarak obje içerisinde tanımladığımız değeri döndürdü.

Daha sonra transaction adında fonksiyon oluşturduk.

```
const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined,
  rating: 3,
};

const { label: etiket, stock, rating = 5 } = product;

console.log(etiket);
console.log(stock);
console.log(rating);

////////

const transaction = (type, myProduct) => {
  const { label } = myProduct;
};

transaction("order", product);
```

Fonksiyon içerisinde yazdığımız product nesnesini çağırdık. Ardından tüm bilgileri göndermek yerine sadece ihtiyacımız olanları gönderebilmek için kodu şu şekilde düzenledik:

```
const product = {
  label: "Kırmızı laptop",
  price: 300,
  stock: 20,
  salePrice: undefined,
  rating: 3,
};

const { label: etiket, stock, rating = 5 } = product;

console.log(etiket);
console.log(stock);
console.log(rating);

const transaction = (type, { label, stock }) => {
  //const { label } = myProduct;
  console.log(type, label, stock);
};

transaction("order", product);
```

Kodda parametre olarak product göndermek yerine ihtiyaç olan bilgileri girdik. Girilen bilgileri de konsola yazdırdık. Çıktı olarak da sunu aldı:

<order Kırmızı laptop 20>

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node object-test.js
○ { userName: 'Can', age: 25, location: 'Bursa' }
  Kırmızı laptop
  20
  3
  order Kırmızı laptop 20
  PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> |
```

Burada öğrendiğimiz özelliği app.js dosyasında uyguladık.

```
const address = process.argv[2];

if (!address) {
  console.log("Lütfen bir konum bilgisi girin");
} else {
  geocode(address, (error, { longitude, latitude, location }) => {
    if (error) {
      return console.log(error);
    } else {
      forecast(latitude, longitude, (error, data) => {
        console.log(location);
        if (error) {
          return console.log(error);
        }
        console.log(data);
      });
    }
  });
}
```

Geocode içerisinde tüm bilgileri göndermek yerine yalnızca longitude, latitude ve locaiton bilgilerini gönderdik. Çıktı olarak da:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

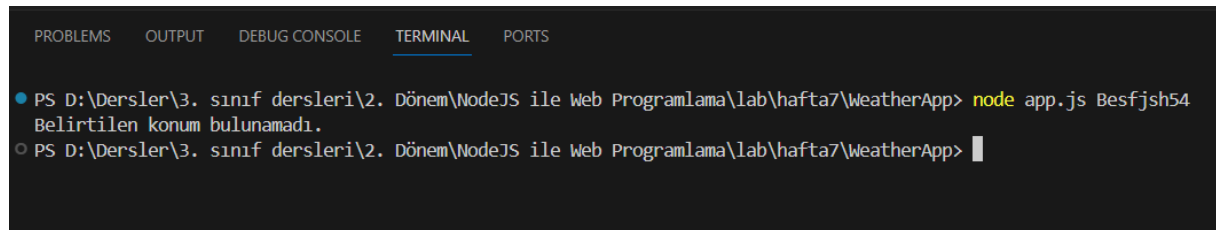
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node app.js Bursa
Bursa, Bursa, Türkiye
Hava şu anda: Light Rain
Hava sıcaklığı şu anda: 4 derece
● Hissedilen sıcaklık: 0 derece
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> |
```

Doğru sonuç aldık.

Hata almamak adına konuma yanlış girme durumunda; geocode içerisindeki parametremizi boş objeye atadık.

```
if (!address) {
  console.log("Lütfen bir konum bilgisi girin");
} else {
  geocode(address, (error, { longitude, latitude, location } = {}) => {
    if (error) {
      return console.log(error);
    }
  })
}
```

Artık yanlış değer girme durumunda şu çıktıyı alacak:



The screenshot shows a terminal window with the following text:

```
PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node app.js Besfjsh54
Belirtilen konum bulunamadı.
PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp>
```

Bu işlemi bitirince forecast.js dosyasına geçtik. Burada da bir takım değişikliğe gittik. O da şu oldu:

```
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 şu anda: " +
        body.current.weather_descriptions[0] +
        "\nHava sıcaklığı şu anda: " +
        body.current.temperature +
        " derece \nHissedilen sıcaklık: " +
        body.current.feelslike +
        " derece"
    );
  }
})
```

Response üzerinden değişkenlere uzun bir şekilde ulaşıyorduk. Kodumuza bakarsak tüm bilgiler body kısmındaydı. Bizde request fonksiyonunda ona göre tanımladık ve artık bir bilgi ulaşırken <response.body.değişken_adi> yazmak yerinde response yazmaktan kurtulduk. Böylece kodda sadeleştirmeye gitmiş olduk.

Aynı durum geocode.js için de geçerliydi. Ardından onu da değiştirdik. O da şu şekilde oldu:

```

const geocode = (address, callback) => {
  const url =
    "https://api.mapbox.com/geocoding/v5/mapbox.places/" +
    encodeURIComponent(address) +
    ".json?access_token=pk.eyJ1Ijoiz3ptYXZjMCIsImEiOiJjbHU0NDh3M3UxZWZuMmpuMHZ3eWRzaHdhIn0.TOJnU1NyRoRbIepClJMs7Q";

  request({ url: url, json: true }, (error, { body }) => {
    if (error) {
      callback("Geocoding servisine bağlanamadı.", undefined);
    } else if (body.features.length === 0) {
      callback("Belirtilen konum bulunamadı.", undefined);
    } else {
      const longitude = body.features[0].center[0];
      const latitude = body.features[0].center[1];

      callback(undefined, {
        longitude: body.features[0].center[0],
        latitude: body.features[0].center[1],
        location: body.features[0].place_name,
      });
    }
  });
};

module.exports = geocode;

```

Yine response yerine body olarak response yazan tüm değişkenlerde değiştirme yaptık. Ardından kodu çalıştırdık:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp> node app.js Bursa
Bursa, Bursa, Türkiye
● Hava şu anda: Light Rain
Hava sıcaklığı şu anda: 4 derece
Hissedilen sıcaklık: 0 derece
○ PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp>

```

Kod hala çalışır durumdaydı yani iki dosyada da yaptığımız değişiklikler doğruydü.

Ardından önemli bir modül olan express modülünün kullanımına baktık. Express bir frameworktür.

Yeni klasör oluşturduk: web-server

Terminalde <cd web-server> yazarak oluşturduğumuz klasöre gittik ve yine terminale <npm i -y> yazarak package.json dosyasını yükledik.

Express modülünü de yükledik: <npm i express>

Web-server içerisinde src klasörü oluşturduk ve içerisine de app.js dosyası oluşturduk. Dosya içerisinde ilk işlem olarak express modülünü ekledik. Ek olarak da express modülü bir fonksiyon olarak geldiğinden onu kullanmak için app adında bir değişken tanımladık.

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

Bundan sonraki amacımız app.com, app.com/about ve app.com/help adından sayfalar oluşturmak. İlk adım olarak anasayfamızı oluşturduk. Bunu app.get fonksiyonu ile sağladık. Parametre olarak ilk sayfa bilgisi alıyordu. Onu boş bıraktık. Bu şekilde de anasayfa olarak algılanabiliyordu. İkinci parametre olarak arrow fonksiyonu yazdık. Onunda req ve res adında parametresi vardı. Çıktı olarak da res.send() kullandık ve içerisinde yazdığımız bilgileri web sayfasına yazdırmayı sağladık. Bu kodun da localhost:3000 de çalışması adında app.listen() içerisinde parametre olarak 3000 değerini verdik ve çalışma sırasında konsola sunucunun çalıştığına dair bilgiyi çıktı olarak verdik.

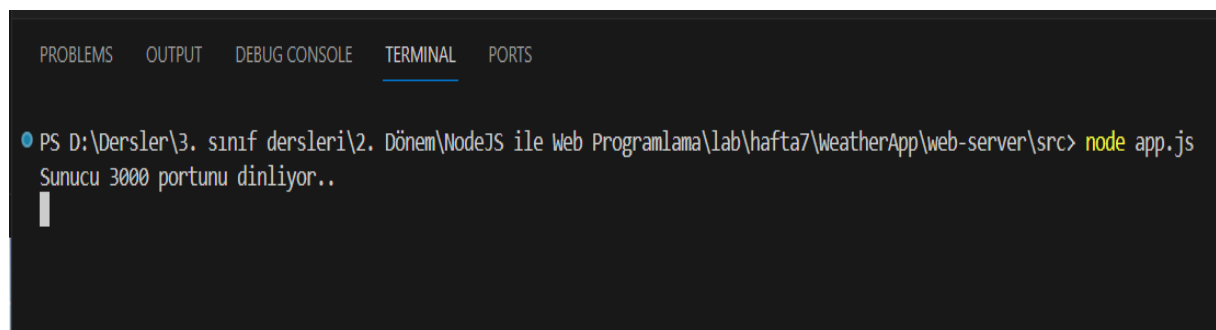
```
const express = require("express");
const app = express();

//app.com : app.com , app.com/about , app.com/help

app.get("", (req, res) => {
  res.send("Hello Express");
});

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

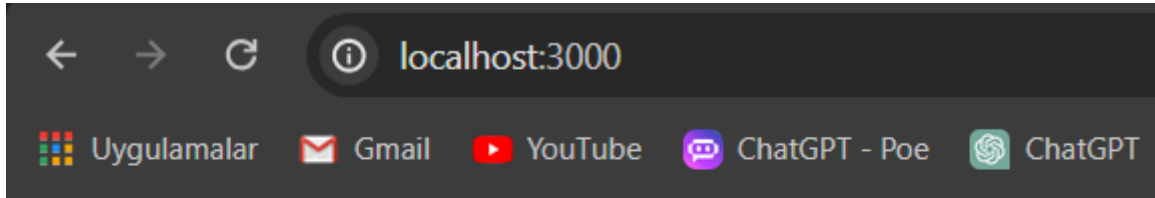
İlk olarak terminalde çalıştırdık ve çıktı olarak şunu aldık:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp\web-server\src> node app.js
Sunucu 3000 portunu dinliyor..
```

İstediğimiz gibi 3000 port da çalışıyordu. Ardından tarayıcıyı açtık ve <localhost:3000> yazıp sayfaya girdik. Sayfa olarak karşımıza şu çıktı:



Hello Express

Kod içerisinde kullandığımız `res.send()` komutu içerisinde yazdığımız bilgi sayfamızda yazdı.

Ardından tıpkı anasayfayı oluşturduğumuz gibi `help` ve `about` sayfalarını oluşturup küçük bir bilgi gönderdik.

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

//app.com : app.com , app.com/about , app.com/help

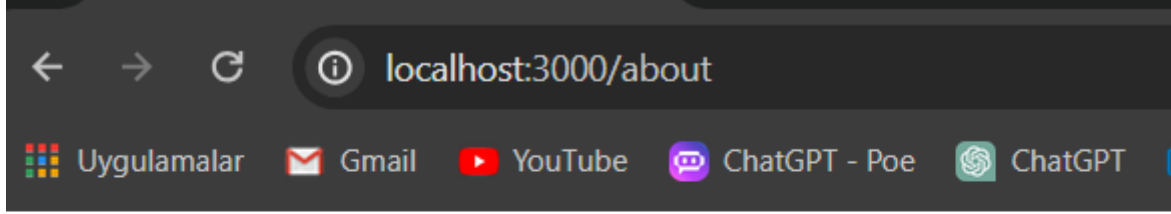
app.get("", (req, res) => {
  //home page
  res.send("Hello Express");
});

app.get("/help", (req, res) => {
  //help
  res.send("Yardım sayfasına hoşgeldiniz.");
});

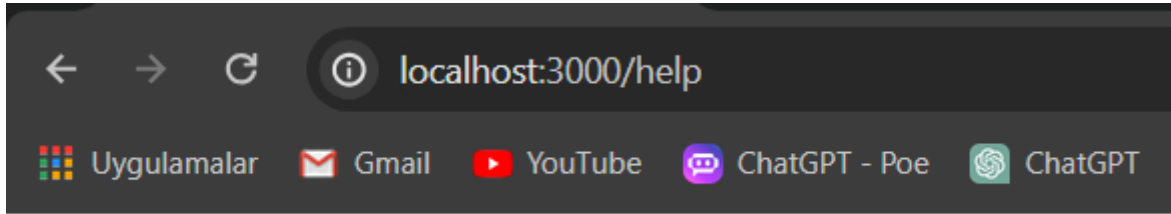
app.get("/about", (req, res) => {
  //about
  res.send("Hakkımızda");
});

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

Terminalde kodumuzu <ctrl+c> diyerek durdurup tekrardan çalıştırdık. Oluşturduğumuz sayfaların adını tarayıcıya girdik, çıktı olarak şunları aldık:



Hakkımızda



Yardım sayfasına hoşgeldiniz.

Yine başarılı bir şekilde çalıştı.

Fakat her seferinde sunucuyu kapatıp tekrar tekrar başlatmak pek de kullanışlı değildi. Bu yüzden yeni modül olan nodemon modülünü kurduk.

<npm i nodemon>

Artık kodda değişiklik yapma durumunda kodu kaydedeceğiz ve sunucu otomatik güncellenecek.

Bunu da yine terminal üzerinden görmek istersek şu şekilde deneyebiliriz:

İlk olarak terminalde <nodemon app.js> yazarak kodu çalıştırırız. Ardından kod sayfaları üzerinden herhangi bir değişiklik yaparız. Değişiklik sonundan değişiklik yapılan sayfayı <ctrl+s> yaparak kaydederiz. Bu işlem sonunda terminalde şu şekilde bir değişiklik meydana gelir:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\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..
█
```

Sunucu portu kaydedilme sonrası güncellenip bir kez daha çalışır.

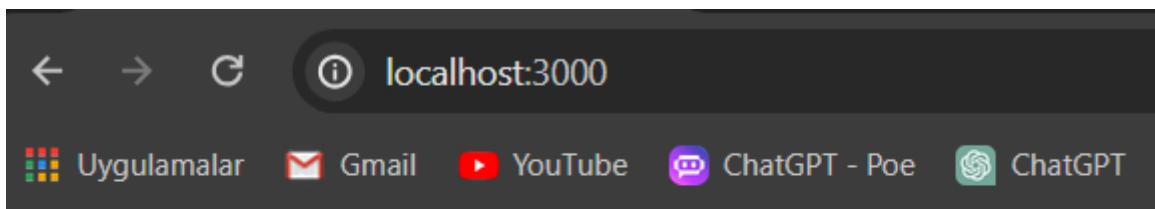
```
app.get("/weather", (req, res) => {
  //help
  res.send("Hava durumu bilgisi");
});
```

Artı olarak yukarıdaki kodda hava durumu bilgisi olan bir sayfa oluşturduk.

Anasayfada res.send() içerisinde html kodu oluşturmayı denedik:

```
app.get("", (req, res) => {
  //home page
  res.send("<h1>Hava Durumu</h1>");
});
```

Çıktı olarak da şunu aldık:

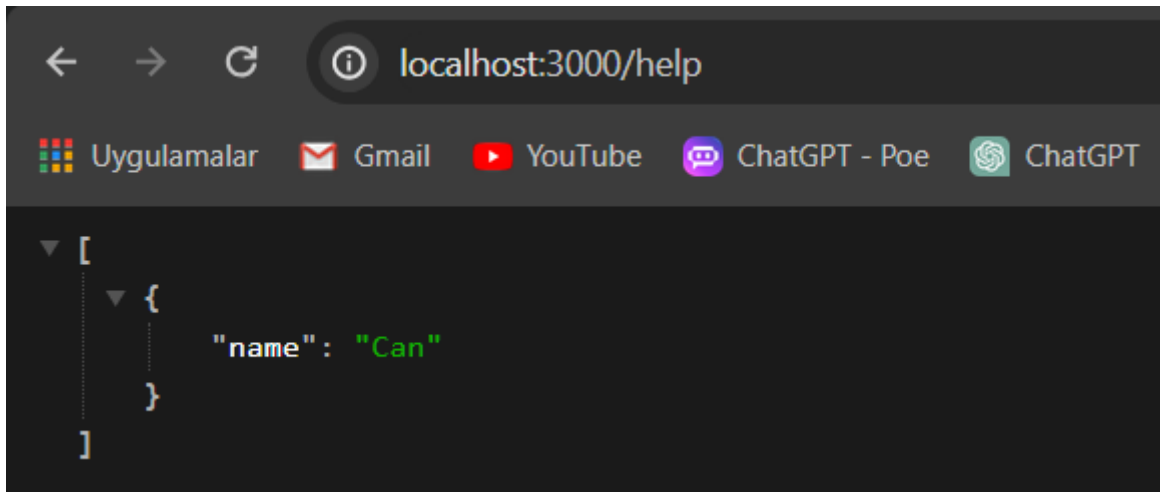


Hava Durumu

Yazdığımız gibi h1 başlığı şeklinde bize çıktı verdi. Ardından bu sefer de /help sayfasında json formatında bir çıktı göndermeyi denedik.

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

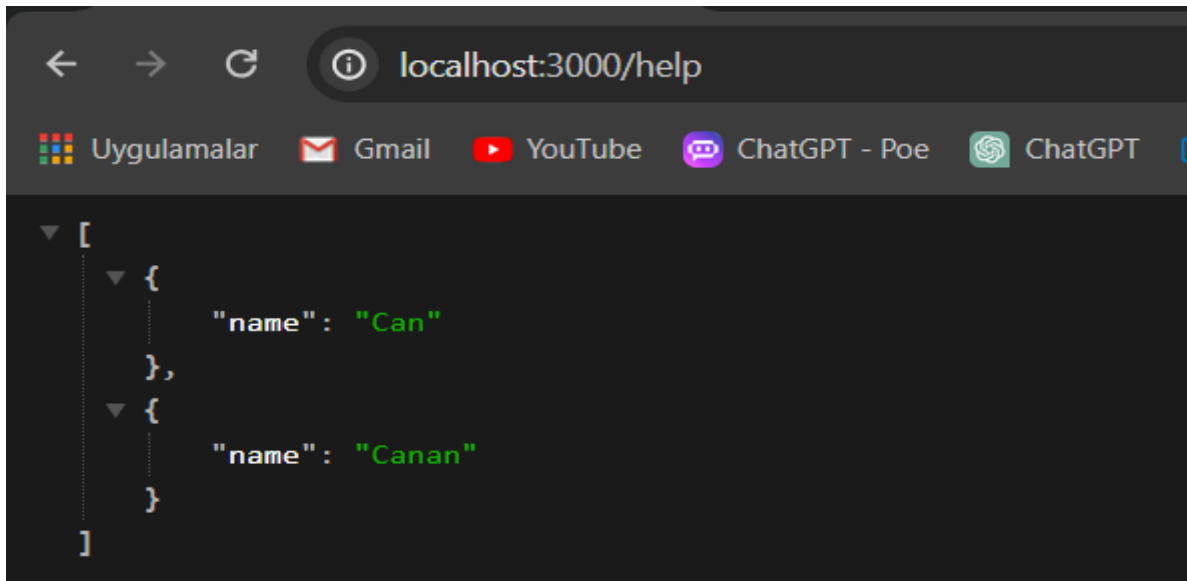
Çıktısı şu şekilde oldu:



Ek olarak aynı değişken adından yeni bir isim bilgisi ekledik.

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

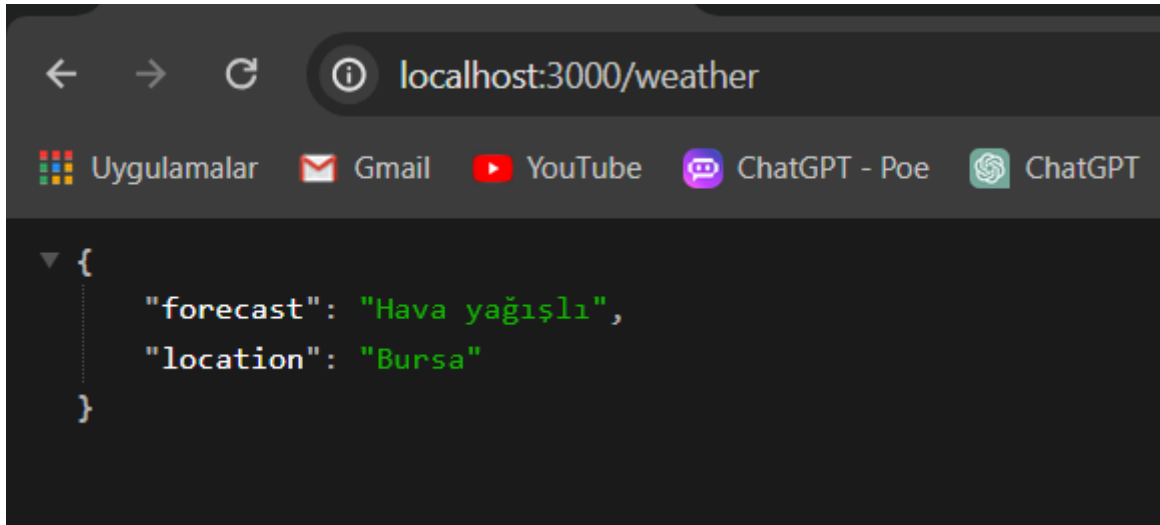
Bunun da çıktısı şu şekilde oldu:



Ardından weather sayfasını da json formatında yazdırdık:

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

Sayfanın çıktısı da şu şekilde oldu:



Bu işlemlerden sonra web-server klasörü içerisinde public adında yeni bir klasör açtık. O klasörün içine de index.html adında bir dosya açtık.

Dosya içerisinde klasik html kodlarını yazdık. <! Yazıp TAB tuşuna tıkladık> Artı olarak body kısmına h1 formatında başlık yazdırdık.

```
<!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 statik bir web sayfasıdır.</h1>
</body>
</html>
```

Kodu yazdık fakat bu kodu app.js üzerinden ulaşmamız gerekiyor. Koda ulaşmak içinse yol bilgisini vermeliyiz. Fakat bu yol bilgisi bilgisayardan itibaren olmalı. Bunu yapmak da zor olduğundan konum bilgisini görmek için iki yeni özellik öğrendik:

→ __dirname: O esnada icra edilen dosyanın kök dizine olan yolunu belirtir.

→ __filename: O esnada icra edilen dosyanın mutlak yolunu içerir.

Kullanımları daha iyi anlamak adına web server klasöründeki src klasöründe bulunan app.js adındaki dosyada şu kodu yazıp çalıştırdık:

```
console.log(__dirname)
console.log(__filename)
```

Çıktı şunu aldık:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Sunucu 3000 portunu dinliyor..
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp\web-server\src
D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp\web-server\src\app.js
Sunucu 3000 portunu dinliyor..
```

__dirname için klasöre kadar yazdırırken, __filename için dosyaya kadar yazdırdı.

Bunun kullanımını kavradıktan sonra dosyanın yolunu vermesi için path modülünü öğrendik.

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

Ardından kodumuza modülü dahil ettik.

Aşağıdaki kod ile şunu sağladık:

- İlk olarak __dirname yazdık ve bulunan klasöre kadar yazdırdık.
- İkinci olarak ../ kullandık ve bulunan klasörden çıktık yani src klasöründeyken web-server klasörüne(bir üst klasör) geçiş sağladık.
- Son olarak da public yazarak web-server klasöründe bulunan public klasörüne geçiş yapmış olduk.

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

Aynı zamanda yazdığımız kodu çalıştırıp kontrol ettik:

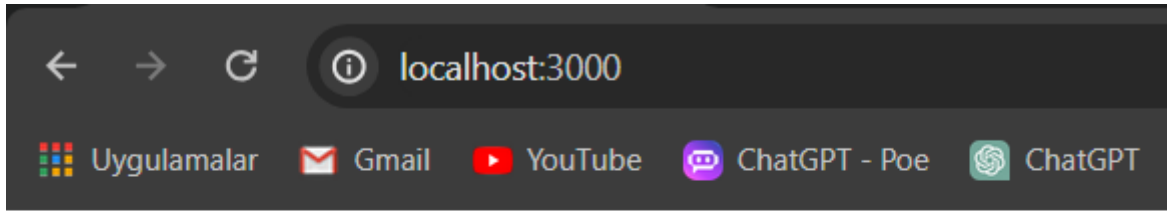
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Sunucu 3000 portunu dinliyor..
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp\web-server\src
D:\Dersler\3. sınıf dersleri\2. Dönem\NodeJS ile Web Programlama\lab\hafta7\WeatherApp\web-server\public
Sunucu 3000 portunu dinliyor..
```

İstedığımız sonuç çıktı. Yani yazdığımız yol doğruydı.

```
const publicDirectoryPath = path.join(__dirname, "../public") //herkese açık
klasör bilgisi
```

```
app.use(express.static(publicDirectoryPath))
app.use() kullanarak tarayıcıda erişime izin verdik.
```



Bu statik bir web sayfasıdır.

Artık home page : index.html sayfası oldu.

Ardından public içerisinde about.html ve help.html sayfalarını da oluşturup router handlerlere olan ihtiyacımızı ortadan kaldırdık.

İlk olarak help.html sayfası:

```
<!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>
```

Ardından about.html sayfası:

```
<!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>Hakkımızda</h1>
  </body>
</html>
```

Şeklinde klasik html sayfalarını oluşturduk.

Public içinde css klasörü ve onun da içerisine style.css oluşturduk. Bu css dosyası içerisine şu kodu yazdık:

```
h1 {  
  color: pink;  
}
```

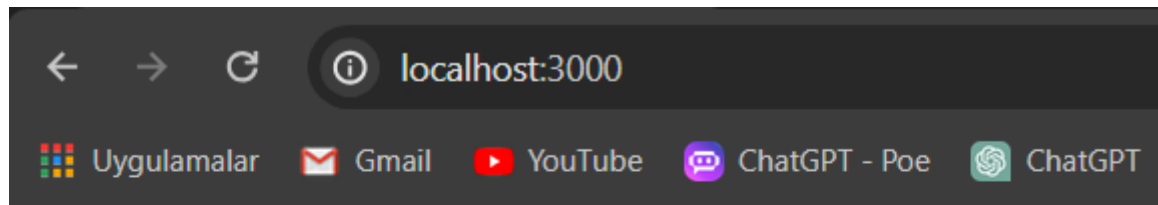
Bu sayede dahil ettiğimiz yerlerdeki h1 başlık kullanılan yazılar pembe renkli olacaktı.

Ardından anasayfamıza yani index.html sayfasına css dosyamızı dahil ettik:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <link rel="stylesheet" href="./css/styles.css" />  
    <title>Document</title>  
  </head>  
  <body>  
    <h1>Bu statik bir web sayfasıdır.</h1>  
  </body>  
</html>
```

Kodda head içerisine rel ve href komutları ile dosya yolunu belirttik. Ardından kodu kaydedip sayfamıza döndük.

Sayfayı güncellediğimizde şöyle bir sonuç aldık:

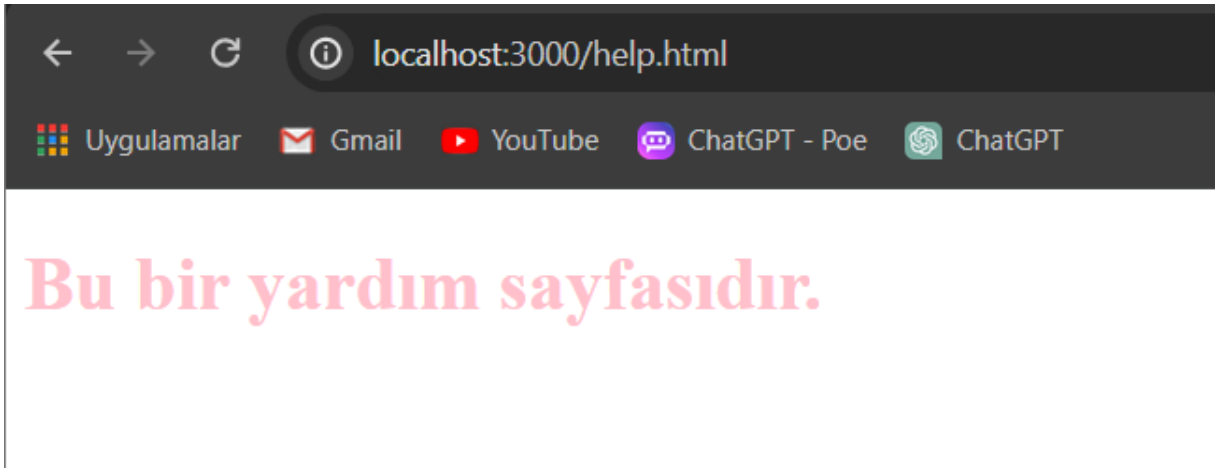


Bu statik bir web sayfasıdır.

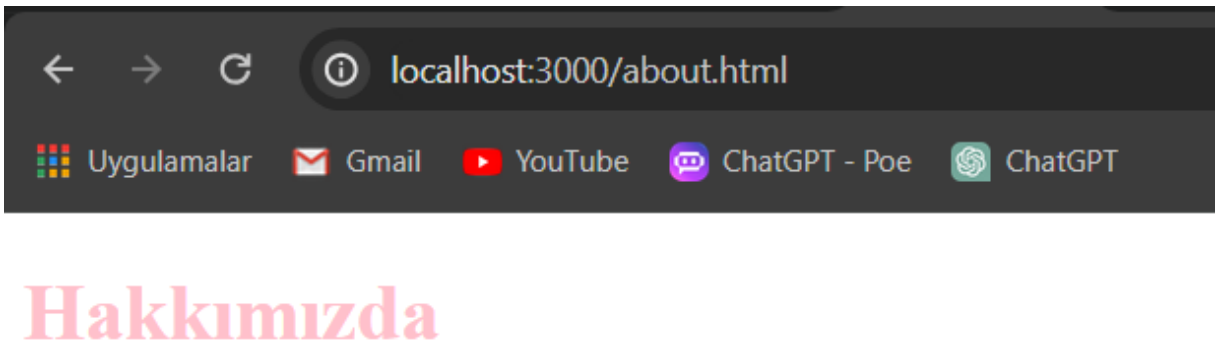
Aynı işlemi about.html ve help.html için de uyguladık. Kod kısmında head bloğu arasına şu kodu yazdık:

```
<head>  
  <link rel="stylesheet" href="./css/styles.css" />  
  <title>Document</title>  
</head>
```

Çıktı olarak da help.html de:



about.html de:



Çıktılarını aldık. Artık app.js dosyasında yazdığımız app.get() komutu ile yazdığımız kod bloklarına ihtiyacımız kalmadı. Kodu bu sayfalarla devam ettireceğiz.

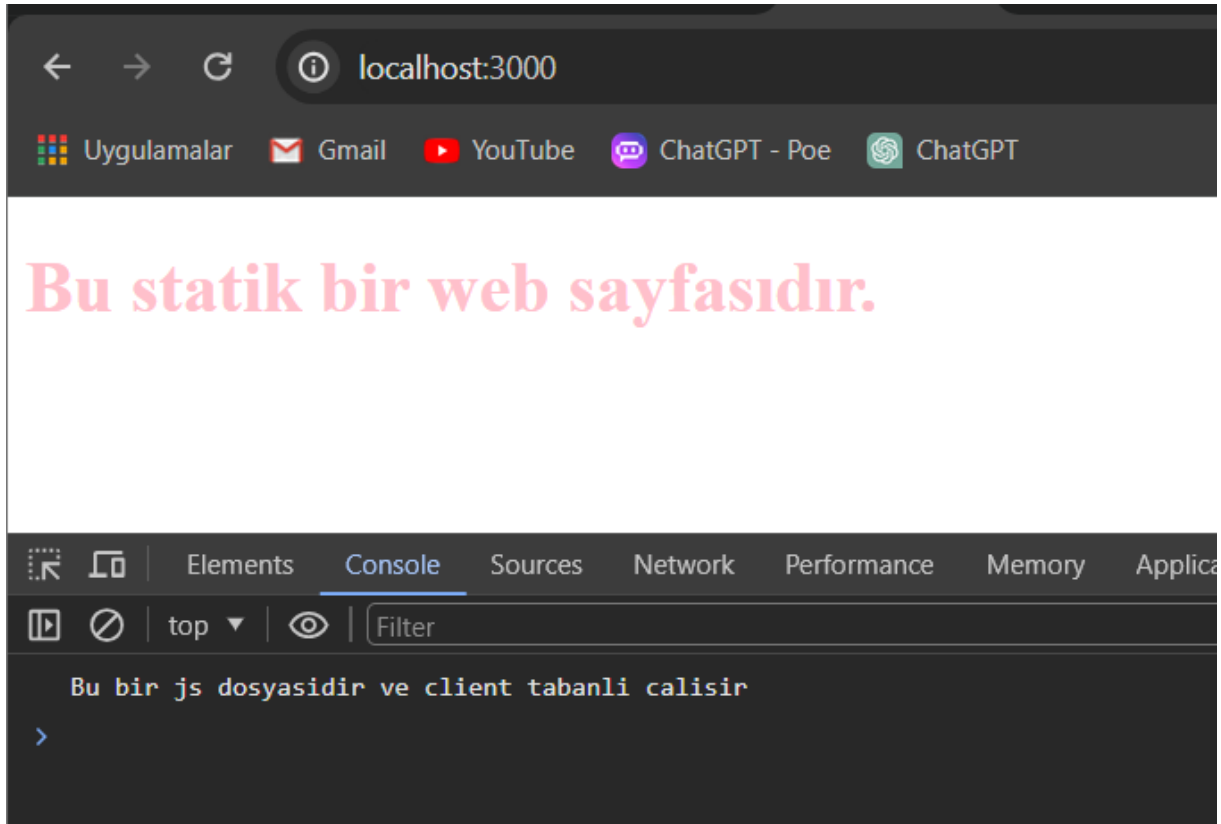
Ardından web-server klasörü içerisindeki public klasörü içine js adında başka bir klasör açıp içerisine de app.js adında dosya açtık. Bu dosya içerisine yalnızca şu kodu yazdık:

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

Ardından index.html dosyasına bu dosyayı ekledik. O da şu kodlar ile oldu:

```
<head>
  <link rel="stylesheet" href="./css/styles.css" />
  <script src="/js/app.js"></script>
  <title>Document</title>
</head>
```

Script komutu içinde src kullanarak dosyanın yolunu girdik. Ardından kodu kaydedip tarayıcıya döndüğümüzde şu çıktıyı gördük:



Ardından yine public klasörüne img adında başka bir klasör açtık. Bu klasöre fotoğraf yükleyip sayfaya ekleme işlemini gördük.

İlk olarak rastgele bir resim indirdik ve indirmiş olduğumuz resmi img klasörüne yerleştirdik.

Ardından about.html sayfasına girdik ve body kısmına şu kodu yazdık:

```
<body>
  <h1>Hakkımızda</h1>
  
</body>
```

Kaydedip tarayıcıya gittik. Sayfamız şu şekildeydi:

Hakkımızda

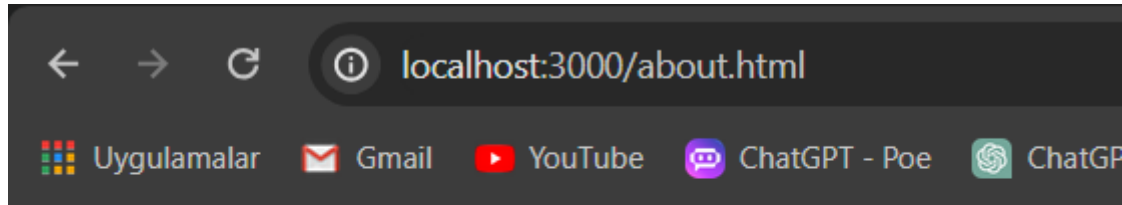


BURSA TEKNİK

Fotoğraf çok büyük olduğundan ekrana sığmadı. Bu yüzden styles.css dosyası içerisinde img boyutunu ayarlamak için ek kod yazdık. O da şu şekilde oldu:

```
img {  
  width: 200px;  
  margin-top: 10px;  
}
```

Bu sayede boyut düzenlendi. Sayfa da şu şekilde göründü:



Hakkımızda



BURSA TEKNİK ÜNİVERSİTESİ

Ardından kodumuzda handlerbars kullandık. Handlebars basit bir şablonlama dilidir. Geliştirdiğimiz web sitesi parçalara ayırmamıza ve parametrikleri kullanarak yönetmemize olanak sağlar.

Kurulumu için terminale `<npm i hbs>` yazdık. Ardından da src klasörü içerisindeki app.js dosyamıza gerekli eklemeyi yaptık:

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

Bu kodu yazıp public klasörü içerisine views adında yeni bir klasör açtık ve içerisine index.hbs dosyası oluşturduk.

Ardından index.html dosyasına olan kodu index.hbs içerisine yapıştırdık. Artık html sayfasına ihtiyacımız kalmadı. Tabii bunun için de app.js dosyamızdaki anasayfa bilgilerinde değişiklik yapmamız gerekli. O da şu:

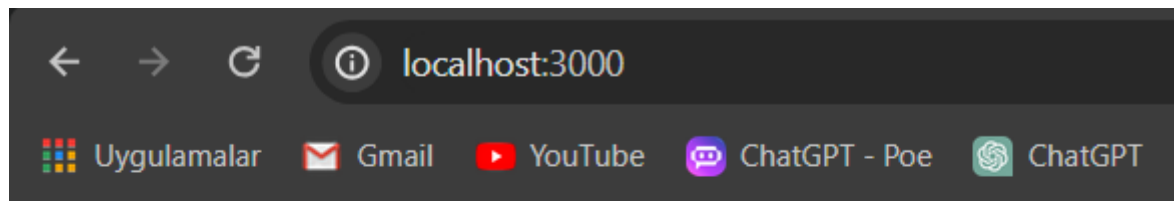
```
app.get("", (req, res) => {  
  //home page  
  // res.send("<h1>Hava Durumu</h1>");  
  res.render("index");  
});
```

Farklılık olarak; res.send() kullanımı yerine res.render() kullandık. İçerisine ise index.hbs yazmak yerine index yazabildik çünkü ikisi de aynı anlama geliyor.

Kontrol amaçlı sayfamıza bakmadan önce de son bir ekleme yapıp klasörün erişimi sağlayacağız:

```
app.set("view engine", "hbs");  
const viewsPath = path.join(__dirname, "../public/views");  
app.set("views", viewsPath);
```

Bu işlemten sonra sayfamıza bakabiliriz:



Bu statik bir web sayfasıdır.

Yaptığımız işlem başarılı bir şekilde çalıştı.

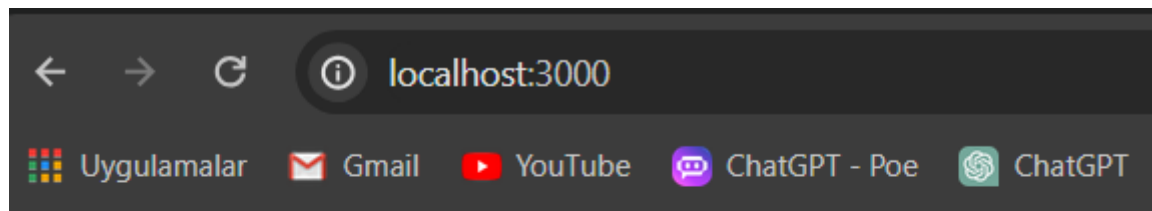
Ardından index.hbs dosyasının body kısmına şu kodları ekledik:

```
<body>
  <h1>{{title}}</h1>
  <p>{{name}} tarafından geliştirilmiştir.</p>
</body>
```

Bu sayede app.js dosyasından gönderdiğimiz bilgileri çekip sayfamızda yazdırabileceğiz. Tabii ilk önce dosyadan bilgi gönderelim:

```
app.get("", (req, res) => {
  //home page
  // res.send("<h1>Hava Durumu</h1>");
  res.render("index", {
    title: "Hava Durumu Uygulaması",
    name: "Gizem Avci",
  });
});
```

Şimdi kodumuzu kaydedip sayfamıza bakalım:



Hava Durumu Uygulaması

Gizem Avci tarafından geliştirilmiştir.

Bu işlem de başarılı bir şekilde çalıştı. Bu dosyamız için (index.hbs) işlemler şimdilik bitti. Yaptığımız bu işlemleri about ve help sayfaları için de gerçekleştireceğiz.

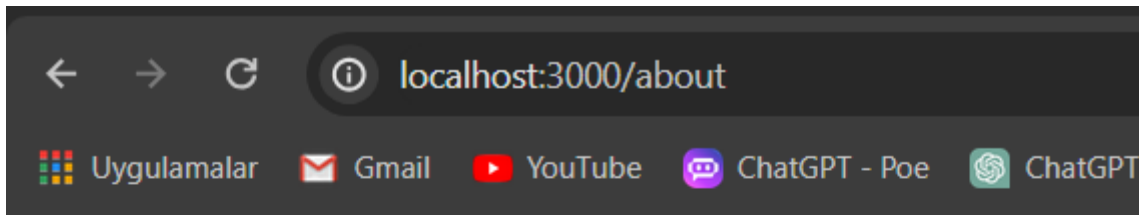
İlk olarak about sayfasından başladık ve diğer seferki gibi html sayfasında yazdığımız kodları içerisine aldık. Artı olarak body kısmına şunu ekledik:

```
<body>
  <h1>Hakkımızda</h1>
  
  <p>{{name}} tarafından geliştirilmiştir.</p>
</body>
```

Ardından app.js içerisinde about sayfası için şunları yazdık:

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

Kodu kaydettik ve sayfamız:



Hakkımızda



Gizem Avcı tarafından geliştirilmiştir.

Halini aldı.

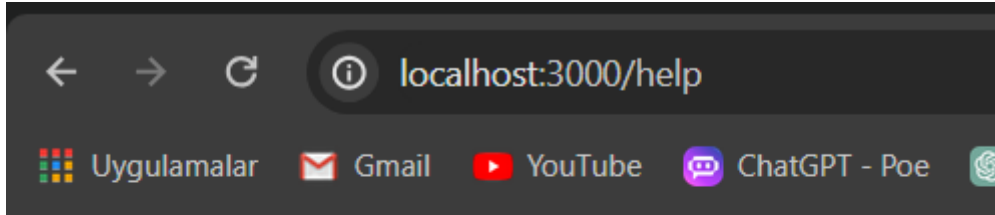
İkinci işlem olan help sayfasını düzenlemeye geçtik. Bu sayfada yine html den aldık ve artı olarak body kısmına şunları yazdık:

```
<body>  
  <h1>Yardım Sayfası</h1>  
  <p>{{helpText}}</p>  
</body>
```

Ardından app.js dosyasında da şu kodu yazdık:

```
app.get("/help", (req, res) => {  
  //help  
  //res.send([{ name: "Can" }, { name: "Canan" }]);  
  res.render("help", {  
    title: "Yardım sayfası",  
    helpText: "Bu bir deneme yazısıdır.",  
  });  
});
```

Ardından kaydedip yine sayfamızı kontrol ettik:



Yardım Sayfası

Bu bir deneme yazısıdır.

Son olarak bu kontrolle birlikte yazdığımız bütün html sayfalarına ihtiyacımız kalmadı ve artık hbs üzerinden devam edebiliriz.