

Node.js Section 7 - 8

Merhabalar bugün node.js serimize devam ediyoruz ve 7. ve 8. Bölümüne bakacağız.

Oncelikle auxiliary topics klasoöründe ObjectPropertyShorthandAndDestructing.js adında bir dosya oluşturup. İnceleyelim.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructing.js
const name = "Andrew"
const userAge = 27
const user = {
  name,
  age: userAge,
  location: "Bursa"
}
```

Normalde nesnelerimizi bu şekilde oluştururuz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructing.js
// * Shorthand syntax
const name = "Andrew"
const userAge = 27
const user = {
  name,
  age: userAge,
  location: "Bursa"
}
```

Bu şekilde de oluşturabiliriz. Buna shorthand syntax denir.

name variable'ı olduğu için user nesnesinin name özelliğine bir üst scopedaki name variable'ını atamaktadır.

age için shorthand syntax avantajından yararlanamıyoruz sebebi ise age variable'nın olmamasıdır.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructing.js
const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined
}
const label = product.label
const stock = product.stock
```

Bir nesnenin özelliklerini kullanmak için bir değişkene atayacağımız zaman böyle yaparız bu biraz yazarken zaman kaybına yol açabilir. Onun yerine şu şekilde de kullanabiliriz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined
}
const { label, stock } = product
console.log(label)
console.log(stock)
```

Bu da aynı sonucu verecektir. Temel olarak label ve stock variable isimlerini product içinden bulup getiriyor bir değişken olarak. Eğer içinde olmayan bir variable ismini denersek undefined olarak dönecektir.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined
}
const { label, stock ,rating} = product
console.log(label)
console.log(stock)
console.log(rating) // return undefined
```

Cıktı:

```
Red notebook
201
undefined
```

Product nesnesinden gelen variable'ları oradaki isimleriyle kullanmak istemeyebiliriz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

// ! We can rename the variable.
const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined
}
const { label: productLabel, stock } = product
console.log(productLabel)
console.log(stock)
```

Product nesnesinden gelen label variable'ını productLabel olarak adlandırip kullanabiliyoruz burada.

Yeni bir değişken tanımlayıp default değer atayabiliriz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

// ! We can define a new variable.
const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined
}
const {label: productLabel, stock, rating=5} = product
console.log(productLabel)
console.log(stock)
console.log(rating)
```

Red notebook
201
5

Eğer nesnemizde eşleşirse rating default değer kullanılmaz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined,
  rating: 4.2
}
const { label: productLabel, stock, rating = 5 } = product
console.log(productLabel)
console.log(stock)
console.log(rating)
```

Red notebook
201
4.2

Şimdi bizim buradan öğrendiğimiz şey sudur : Bir gelen verinin belirli bir kısmını alabiliriz.Hepsini almamıza gerek yok.Bunu bir de fonksiyonla beraber yapmaya çalışalım.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined,
  rating: 4.2
}

const transaction = (type, myProduct) => {
  const { label } = myProduct
  console.log(label);
}
transaction("order", product)
```

Red notebook

Bu yapıda transaction adında bir fonksiyon vardır ve parametrelerinden myProduct içinden labelı yazdırmaktadır.

Peki bize myProduct nesnesinin hepsine gerek yokki label ve stock variable'ları yeterli.O zaman şuna dönüştürebiliriz.

```
My-NodeJS-Notes - ObjectPropertyShorthandAndDestructuring.js

const product = {
  label: "Red notebook",
  price: 3,
  stock: 201,
  salePrice: undefined,
  rating: 4.2
}

const transaction = (type, {label,stock}) => {
  console.log(label)
  console.log(stock);
}
transaction("order", product)
```

Red notebook
201

Şimdi bu öğrendiğimiz mantığı index.js ,forecast.js ve geocode.js dosyalarında uygulayalım.

Section 5-6 yi Section 5-6-7- 8 olarak değiştirelim. Ve bu klasör içine girelim.

INDEX.JS

Section5-6-7-8 - index.js

```
geocode(address, (error, data) => {
  if (error) {
    console.log("Error :", error)
    return
  }

  forecast(data.longitude, data.latitude, (forecastError, forecastData) => {
    if (forecastError) {
      console.log(chalk.red.inverse("Error : " + forecastError))
      return
    }
    console.log(chalk.green.inverse(data.location + forecastData))
  })
})
```

Burada geocode fonksiyonuna gelen data'dan bana latitude,longitude ve location dışında olanlar işimize yaramıyor. O yüzden şöyle bir hale çevirmek daha iyi olacaktır.

Section5-6-7-8 - index.js

```
geocode(address, (error, { latitude, longitude, location }) => {
  if (error) {
    console.log("Error :", error)
    return
  }

  forecast(longitude, latitude, (forecastError, forecastData) => {
    if (forecastError) {
      console.log(chalk.red.inverse("Error : " + forecastError))
      return
    }
    console.log(chalk.green.inverse(location + forecastData))
  })
})
```

FORECATS.JS

```
Section5-6-7-8 - forecast.js

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
    )
  }
})
```

Bu kısımda bakarsanız response'un body kısmı bize yeterlidir ve url parametresini yukarıda tanımladığımızdan oradan bulacaktır.O yüzden şu hale evirelim.

```
Section5-6-7-8 - forecast.js

request({ 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
    )
  }
})
```

GEOCODE.JS

```
Section5-6-7-8 - geocode.js

request({ url: geocodeURL, json: true }, (error, response) => {
  // ! bursa yerine saçma sapan bir şeyler girersek response'un features dizisi boş gelmektedir.
  if (error) {
    callback("Geocoding servisine bağlanamadı", undefined)
  } else if (response.body.features.length == 0) {
    callback("Belirttiğiniz konum bulunamadı", undefined)
  } else {
    const longitude = response.body.features[0].center[0]
    const latitude = response.body.features[0].center[1]
    const location = response.body.features[0].place_name

    callback(undefined, {
      longitude,
      latitude,
      location
    })
  }
})
```

Burada response'un body si gerekmektedir diğer datalar kullanılmamaktadır ve yukarıda tanımladığımız(burada gözükmeyen) geocodeURL variable ismini url olarak değiştirdikten sonra şu hale çevirebiliriz.

```
Section5-6-7-8 - geocode.js

request({ url, json: true }, (error, {body}) => {
  // ! bursa yerine saçma sapan bir şeyler girersek response'un features dizisi boş gelmektedir.
  if (error) {
    callback("Geocoding servisine bağlanamadı", undefined)
  } else if (body.features.length == 0) {
    callback("Belirttiğiniz konum bulunamadı", undefined)
  } else {
    const longitude = body.features[0].center[0]
    const latitude = body.features[0].center[1]
    const location = body.features[0].place_name

    callback(undefined, {
      longitude,
      latitude,
      location
    })
  }
})
```

Uygulamamıza command line dan etkileşime geçmek yerine express kütüphanesini kullanarak url lerden etkileşime geçeceğiz.
Html,css yada http json tabanlı api barındırabiliriz.

Aynı proje içinde web-server adında klasör oluşturalım ve o pathe gidip npm başlatalım(npm init -y).

Express library'sini indirelim.

npm i express.

web-server klasörünün içinde src adında bir klasör oluşturalım ve src klasörü içinde index.js adında bir dosya oluşturalım.

Bu dosyada express i dahil edelim.

```
Section5-6-7-8 - index.js
const express = require("express")
const app = express()
```

app.com , app.com/help ve app.com/about sayfalarına sahip olduğumuzu varsayıyalım.

```
Section5-6-7-8 - index.js
const express = require("express")
const app = express()

app.get("", (req, res) => {
  res.send("Hello express!") //send a response back to the requester
})
//port number. Default http port number is 80. 3000 is used for development for now.
app.listen(3000, () => {
  console.log("Server 3000 portunu dinliyor.")
})
```

Programı çalıştırıp browserden localhost:3000 i açalım.

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a project structure named "SECTION5-6-7-8" containing "node_modules", "utils", "web-server", and "src". Inside "src", there are "index.js", "package-lock.json", and "package.json".
- EDITOR**: Displays the file "index.js" with the following code:

```
1 const express = require("express")
2 const app = express()
3
4 app.get("", (req, res) => {
5   res.send("Hello express!") //send a response back to the requester
6 }
7 //port number. Default http port number is 80. 3000 is used for development for now.
8 app.listen(3000, () => {
9   console.log("Server 3000 portunu dinliyor.")
10 })
11
```
- TERMINAL**: Shows the command "node index.js" being run, with the output "Server 3000 portunu dinliyor."
- STATUS BAR**: Shows the path "node index.js", the host "iamburakgul@Buraks-MacBook-Air", the port "3000", and the message "Server 3000 portunu dinliyor."



Böyle bir ekran gelecektir. Yeni sayfalar ekleyelim ama yeni sayfa eklediğimizde server'ı kapat tekrar çalıştır durumları olacaktır bu yüzden nodemon kullanalım. Önceden nodemon indirmiştik bu yüzden nodemon src/index.js çalıştırıralım sonra yeni ekranlar ekleyelim.

Yeni sayfalar ekledikten sonra index.js şu halde olacaktır.

```
Section5-6-7-8 - index.js

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

app.get("", (req, res) => {
    res.send("Hello express!") //send a response back to the requester
})

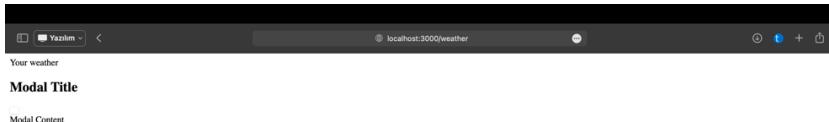
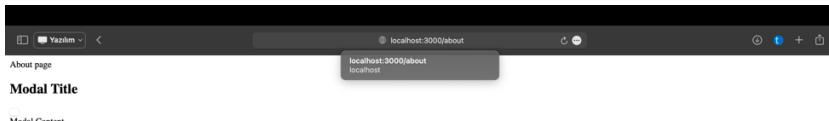
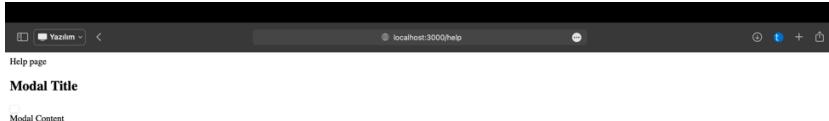
app.get("/help", (req, res) => {
    res.send("Help page")
})

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

app.get("/weather", (req, res) => {
    res.send("Your weather")
})

//port number. Default http port number is 80. 3000 is used for development for now.
app.listen(3000, () => {
    console.log("Server 3000 portunu dinliyor.")
})
```

help, about ve weather sayfalarını çalıştıralım.



Gördüğümüz üzere çalışıyor.

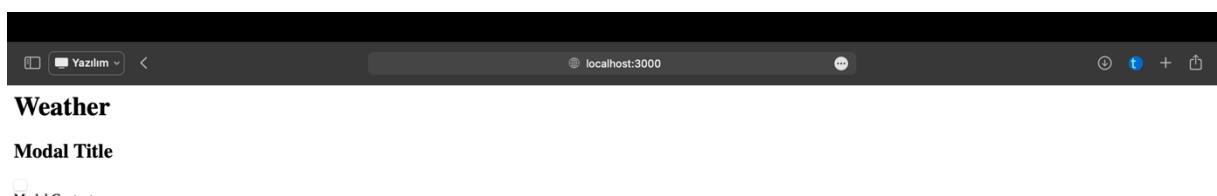
String olarak veri sunduğumuz gibi hmtl ve json verisi de sunabiliriz/gönderebiliriz.



```
Section5-6-7-8 - index.js

// ! html response
app.get("", (req, res) => {
  res.send("<h1>Weather</h1>") //send a response back to the requester
})
```

Ana sayfa için olan get methodunu bu hale getirelim.



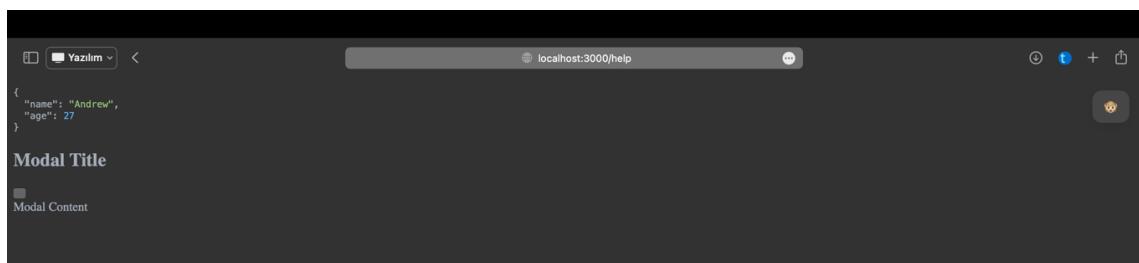
Gördüğümüz üzere html olarak veri sunduk.

Şimdi de help sayfasını değiştirerek json verisi sunalım.



```
Section5-6-7-8 - index.js

// ! Json response:
app.get("/help", (req, res) => {
  res.send({
    name: "Andrew",
    age: 27
  })
})
```



JSON verimiz de geldi burada düzenli göstermesini JSON Viewer gibi eklentiler ile hallediyoruz.

JSON verisi sunarken json dizisi de sunabiliriz.

Şimdi veri sunarken html css gibi verileri de sunabiliriz. Bunlar send içinde hepsini yazarak yapmak mantıksız olur bu yüzden farklı dosyalar oluşturup onları sunacağız.

Web-server klasörünün içinde public klasörü oluşturup içinde de index.html oluşturalım.

```
Section5-6-7-8 - index.html

<!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>From a static file</h1>
</body>
</html>
```

Web-server içindeki src doyasındaki index.js dosyasına gidelim ve

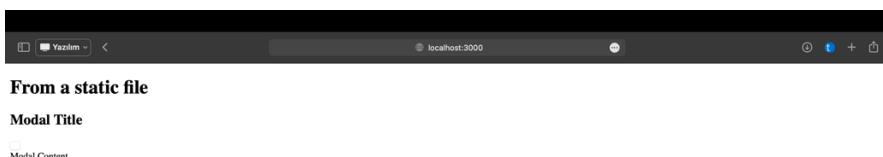
index.html dosyasına ulaşmak için __filename , __dirname kullanacağız. Path işlemleri için Path modülünü içe aktarmamız gerekiyor.

Web server içindeki index.js dosyasına şu kodları ekleyelim ve app.get("",...)(anasayfa) olan kodu yorum satırına alalım.

```
Section5-6-7-8 - index.js

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

Daha sonra localhost:3000 e gidelim



Gördüğümüz üzere static bir dosyayı da sunabildik.

publicDirectoryPath konumuna index.html dosyasını eklemedik çünkü default olarak index.html i arar engine'lar

Artık root(index.html) için statik bir sayfamız olduğundan hiçbir zaman çalışmayaçak olan root için req res komutunu da kaldırabiliriz.

Eski route handler'ları kaldırıp yerine static sayfalar koyacağız.
Public klasörü içinde help.html ve about.html oluşturacağız.

```
Section5-6-7-8 - index.js

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

Web-server içindeki src dosyasındaki index.js dosyam bu hali haldi.

```
Section5-6-7-8 - help.html

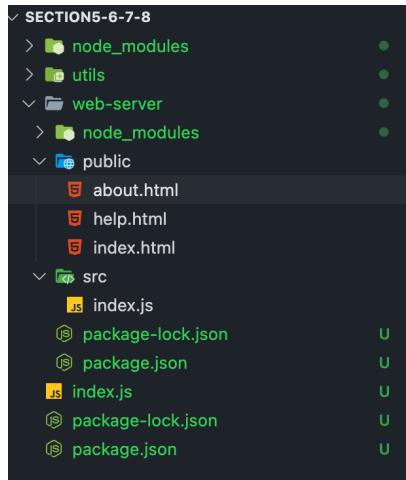
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>From a static file,Help</h1>
</body>
</html>
```

Public klasörü içindeki help.html

```
Section5-6-7-8 - about.html

<!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>From a static file,About</h1>
</body>
</html>
```

Public klasörü içindeki about.html



Projemin klasör yapısı bu oldu şu anda.

Şimdi de css dosyalarını eklemek için public klasörünün içinde css adında klasör oluşturup içine styles.css adında dosya oluşturalım.

```
Section5-6-7-8 - styles.css

h1{
    color: blue;
}
```

İçine bu kodları ekleyelim. Bu styles.css i kullananlar h1 etiketine sahip olanların rengini mavi yapacaktır.

Index.html içine head kısmına bu kodu ekleyip localhost:3000 den alakalı sayfaya(anasayfa) giderek mavi olduğunu görelim.

```
Section5-6-7-8 - index.html

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



Şimdi public klasörü altında js adında bir klasör oluşturup içine app.js adında bir dosya oluşturalım.

İçine de çalıştığını görmek için bir şeyler yazalım.



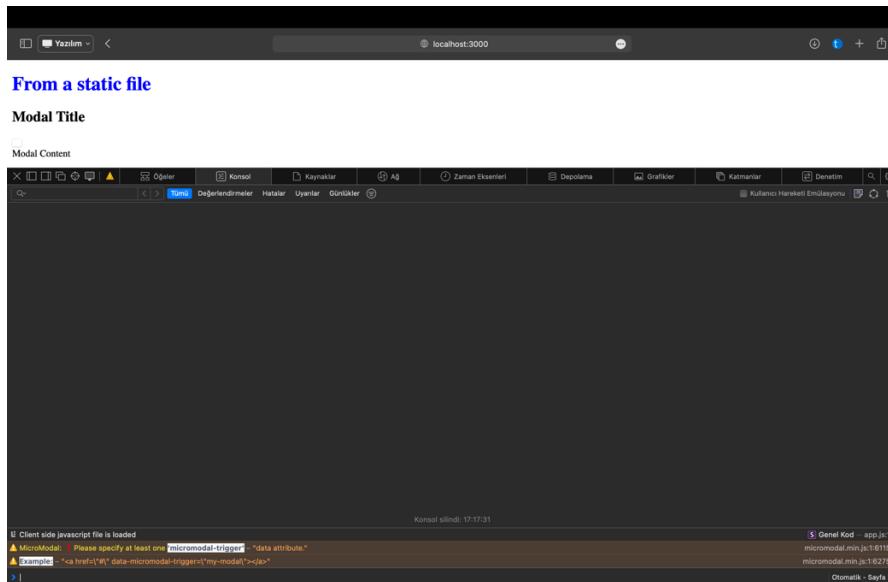
```
Section5-6-7-8 - app.js
console.log("Client side javascript file is loaded")
```

Şimdi index.html de head kısmına şu kodu da ekleyelim.



```
Section5-6-7-8 - index.html
<script src="/js/app.js"></script>
```

Ve çalışıralım localhost:3000 de



Gördüğümüz üzere js klasörünün içindeki app.js dosyası çalıştı.

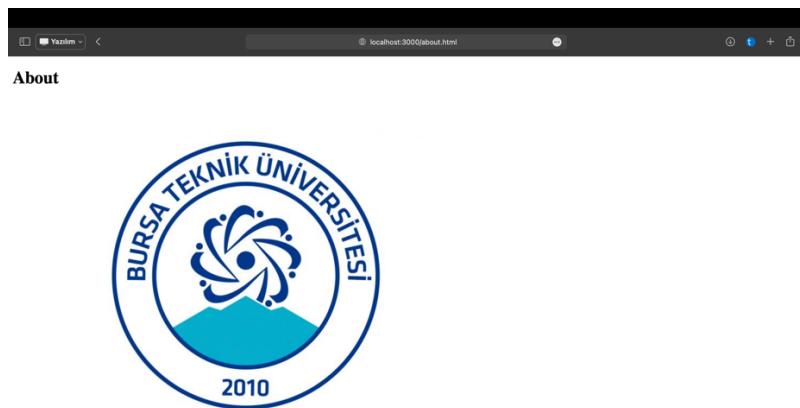
Bir resim göstermek için public klasörünün içinde img klasörü oluşturup içine bir resim ekleyelim.Ben btü adında tutacağım resmi.

About html sayfasını şu şekilde düzenleyelim .

```
Section5-6-7-8 - about.html

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

Kaydedip localhost:3000/about.html i çalıştırırsak şu şekilde gözükar :



Modal Title

Modal Content

Şimdi de styles.css dosyasını şu şekilde düzenleyip çalışıralım

```
Section5-6-7-8 - styles.css

h1 {
  color: blue;
}
image {
  width: 250px;
}
```

Bu styles.css dosyasını kullanan yerde image genişlikleri 250 px olsun diyoruz tekrardan çıktıya bakalım.

Sonuç olarak bir şey değişmediğini görüyoruz.

Bu zamana kadar static web sayfaları kullanıyorduk ve static web sayfaları değişmez.

Template engine'lar kullanacağımız dinamik sayfalar yapmak için.

Npm modülüne ihtiyaç duyuyoruz burada handlebars ve hbs ye ihtiyacımız var.

Nodemon u kapatıp kuralım web-server klasöründe kuracağız.

Npm i hbs.

Src/index.js içinde app.use dan önce hangi template engine'ı kullanacağımızı söylemeliyiz.

```
Section5-6-7-8 - index.js

const express = require("express")
const app = express()
const path = require("path")
const publicDirectoryPath = path.join(__dirname, "../public")
const viewsPath = path.join(__dirname, "../public/views")
app.set("views", viewsPath)

app.use(express.static(publicDirectoryPath))
app.get("", (req, res) => {
  req.sendFile("index")
})
app.listen(3000, () => {
  console.log("Server 3000 portunu dinliyor.")
})
```

web-server/public/src/index.js

Public klasör altında views adında bir klasör oluşturalım.

İçinde index.hbs adında klasör oluşturalım ve içine index.html içindekileri kopyalayıp başlığı değiştirelim.

```
Section5-6-7-8 - index.hbs

<!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" />
    <script src="js/app.js"></script>
    <title>Document</title>
  </head>
  <body>
    <h1>Weather</h1>
  </body>
</html>
```

views/index.hbs

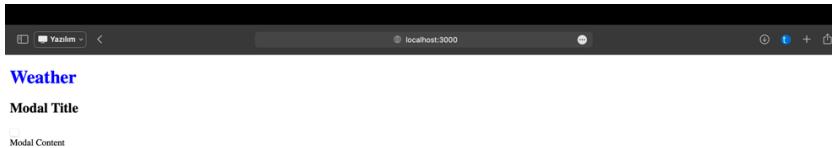
Artık index.html ile işimiz yok silebiliriz.

Bu sayfaya erişebilmek için src/index.js içi şöyle olmalı :

```
Section5-6-7-8 - index.js

const express = require("express")
const app = express()
const path = require("path")
const publicDirectoryPath = path.join(__dirname, "../public")
const viewsPath = path.join(__dirname, "../public/views")
app.set("views", viewsPath)
app.set("view engine", "hbs")
app.use(express.static(publicDirectoryPath))
app.get("", (req, res) => {
  res.render("index")
})
app.listen(3000, () => {
  console.log("Server 3000 portunu dinliyor.")
})
```

Şimdi de nodemon src/index.js çalıştırıralım. Ve localhost:3000 e gidelim.



Gördüğümüz üzere çalışmaya başladı.

Şimdi sayfamızı dinamik hale getirelim.

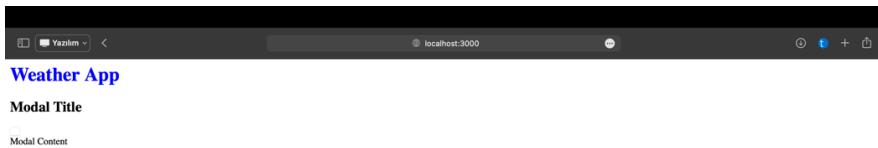
Index.hbs dosyasını şu hale getirelim :

```
Section5-6-7-8 - index.hbs

<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" />
    <script src="js/app.js"></script>
    <title>Document</title>
  </head>
  <body>
    <h1> {{title}}</h1>
  </body>
</html>
```

src/index.js içinde app.get i de şu hale getirelim ve localhost:3000 e gidelim.

```
Section5-6-7-8 - index.js
app.get("", (req, res) => {
  res.render("index",{
    title :"Weather App",
    name:"Burak Gül"
  })
})
```

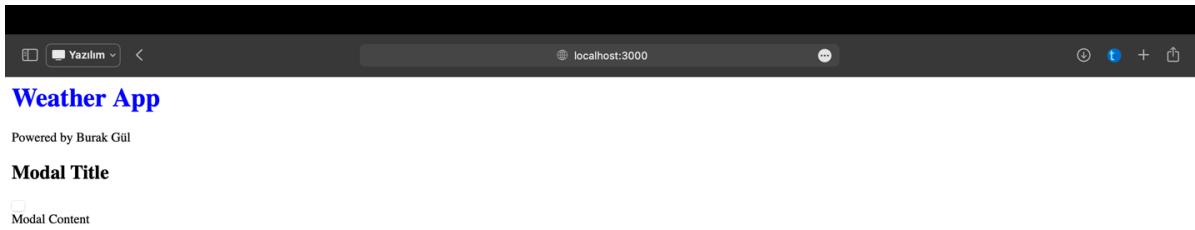


Gördüğümüz üzere çalıştı.

Şimdi de index.hbs dosyasında body kısmını da şu hale getirelim :

```
Section5-6-7-8 - index.hbs
<h1> {{title}}</h1>
<p>Powered by {{name}}</p>
```

Tekrardan çalışıralım.



Sayfamızı dinamik hale getirdik.

Şimdi bu yaptıklarımızı diğer sayfalar için de yapalım.

About.html i about.hbs ye yapıştırıp,about.html i silelim ve about.hbs yi modifiye edelim.

```
Section5-6-7-8 - about.hbs

<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> {{title}}</h1>
    
    <p>Powered by {{name}}</p>
  </body>
</html>
```

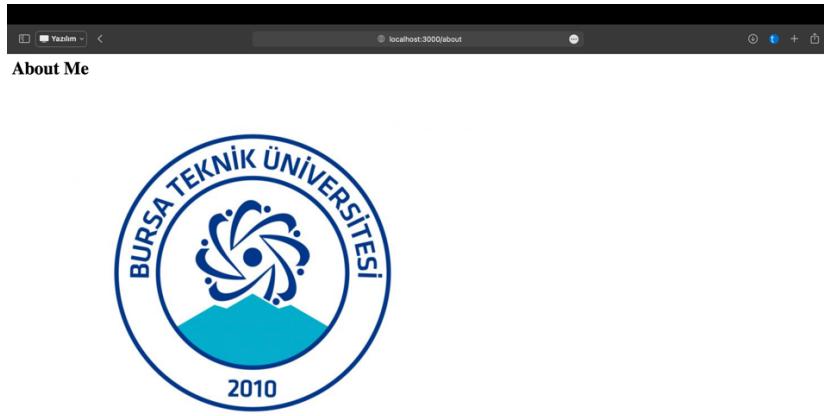
public/views/about.hbs

Web-server/src/index.js e de şu kodu ekleyelim

```
Section5-6-7-8 - index.js

app.get("/about", (req, res) => {
  res.render("About", {
    title: "About Me",
    name: "Burak Gül"
  })
})
```

Şimdi de localhost:3000/about a gidelim



Tamam bu da oldu.

Şimdi help.html dosyasının içeriğini de kopyalayıp help.hbs adında dosyaya yapıştırıyalım ve help.html i silelim işimiz kalmadı onunla daha sonra help.hbs yi de şu hale getirelim :

```
Section5-6-7-8 - help.hbs

<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>Help</h1>
    <p>{{helpText}}</p>
  </body>
</html>
```

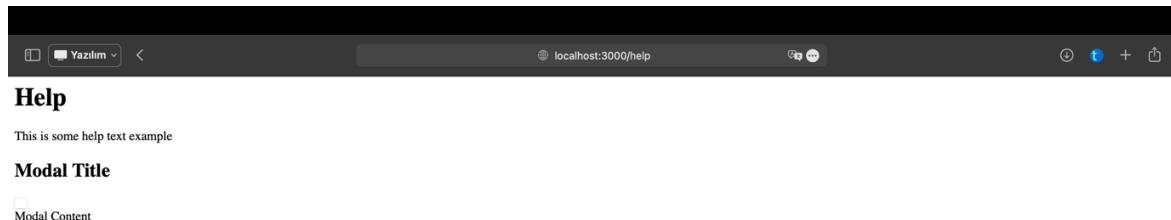
web-server/public/views/help.hbs

Daha sonra web-server/src/index.js e şu kodu ekleyelim :

```
Section5-6-7-8 - index.js

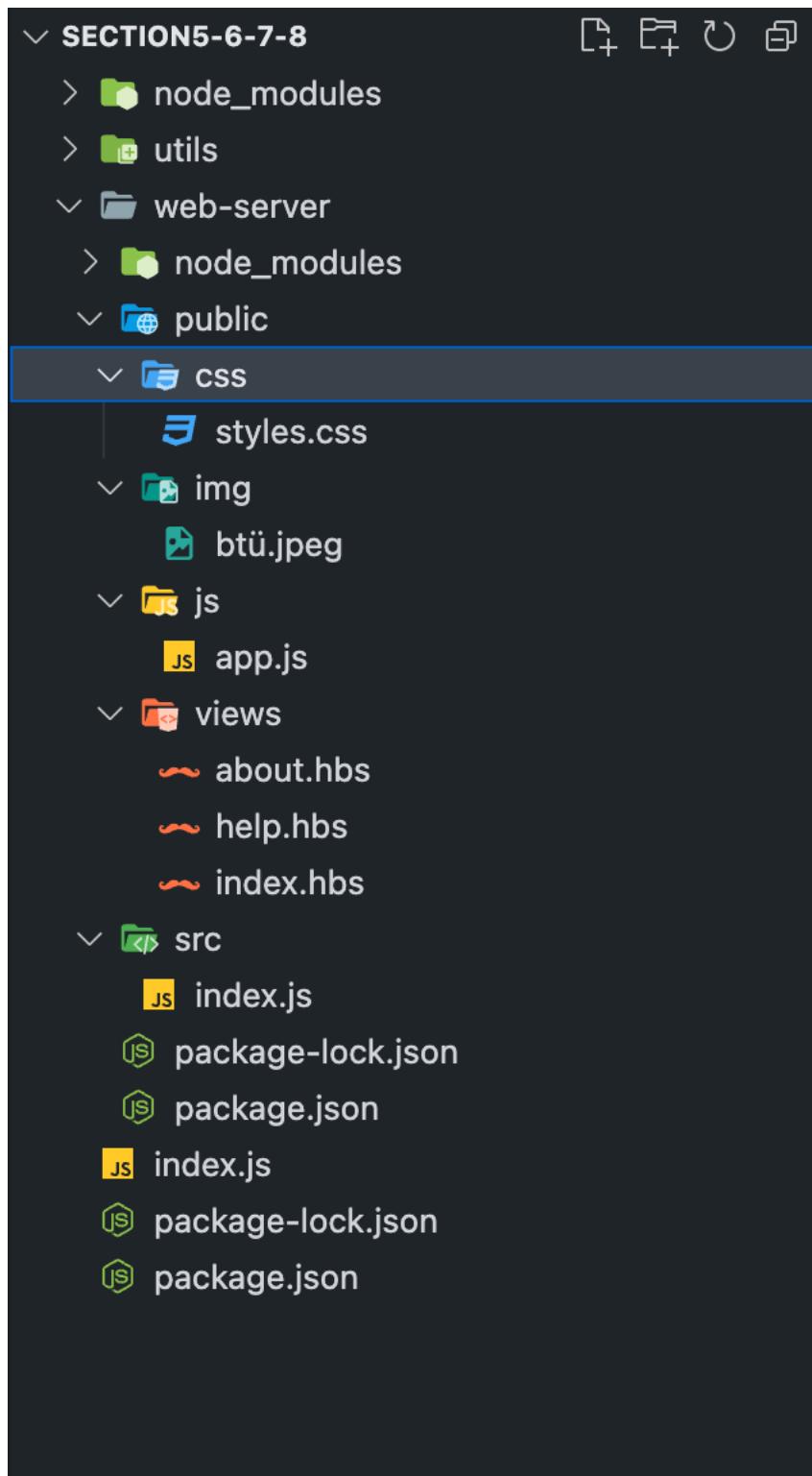
app.get("/help", (req, res) => {
  res.render("help", {
    helpText: "This is some help text example"
  })
})
```

localhost:3000/help e gidelim .



Bu da çalıştı.

Genel dosya yapımız sonunda şu şekilde olmuştur :



Bu haftalık bu kadar 😊 .