

# ***BURSA TEKNİK ÜNİVERSİTESİ***

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



**BLM0470 NodeJS İle Web Programlama**

**2023-2024 Bahar Dönemi**

## **7.Hafta Raporu**

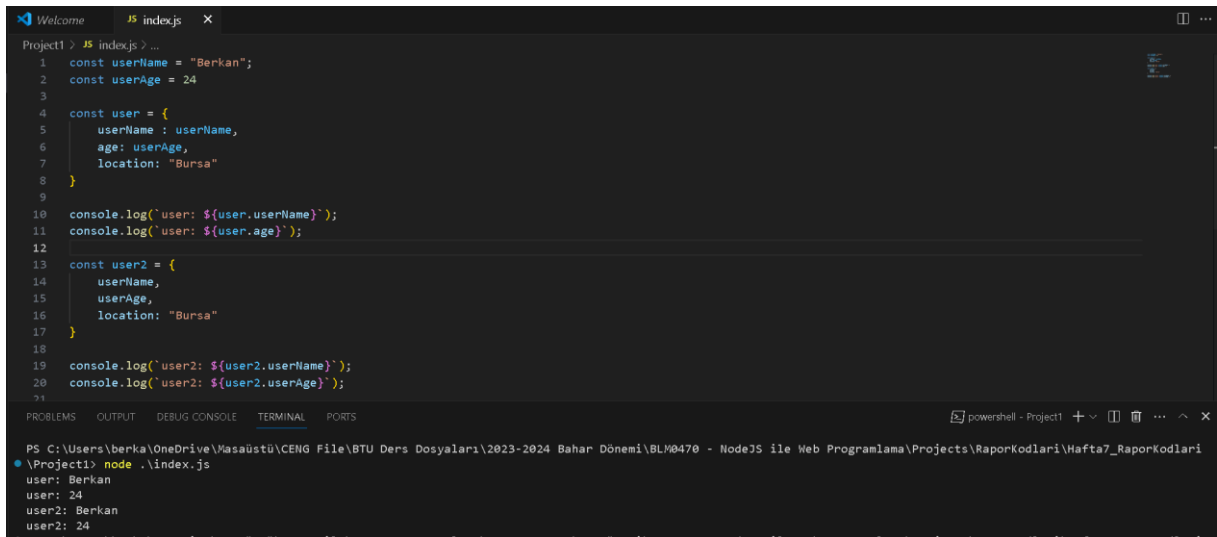
**Berkan SERBES – 22360859353**

## İçindekiler

1. Object Property Shorthand .....	3
2. Destructuring Assignment .....	4
2.1 Array Destructuring .....	4
2.2 Object Destructuring .....	7
3. Hava Durumu Uygulaması .....	10
4. Express.js .....	14
4.1 Hava Durumu Uygulamasına Express.js'i Dahil Etme .....	14
4.2 Statik HTML Sayfaları Oluşturma .....	17
4.3 Handlebars Paketi .....	21
5. Kaynakça .....	25

# 1. Object Property Shorthand

Object Property Shorthand, JavaScript'te nesne oluştururken kısa ve okunaklı bir şekilde kod yazmayı sağlayan bir özelliktir. Bu özellik, bir nesne oluştururken, nesnenin özelliklerini ve değerlerini belirtirken daha az kod yazmamıza olanak tanır. Geleneksel olarak, bir nesne oluştururken her bir özelliğin adını ve değerini belirtmek için tekrar tekrar aynı isimleri kullanmamız gerekir. Ancak Object Property Shorthand kullanılarak, bir değişkenin adı aynı zamanda bir nesne özelliği olarak kullanılabilir. Yani, bir değişken adı ile aynı isimde bir nesne özelliği oluşturulabilir ve bu değişkenin değeri nesne oluşturulurken otomatik olarak atanır. Bu şekilde, daha az kod yazarak nesneler oluşturabiliriz ve kodumuzu daha okunaklı hale getirebiliriz. Object Property Shorthand, özellikle büyük ve karmaşık nesneler oluştururken kod tekrarını azaltmaya ve kodun daha temiz ve anlaşılır olmasını sağlamaya yardımcı olur.



```
Project1 > JS index.js > ...
1  const userName = "Berkan";
2  const userAge = 24
3
4  const user = {
5      userName : userName,
6      age: userAge,
7      location: "Bursa"
8  }
9
10 console.log(`user: ${user.userName}`);
11 console.log(`user: ${user.age}`);
12
13 const user2 = {
14     userName,
15     userAge,
16     location: "Bursa"
17 }
18
19 console.log(`user2: ${user2.userName}`);
20 console.log(`user2: ${user2.userAge}`);
21
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodlari\Hafta7_RaporKodlari
.\Project1> node .\index.js
user: Berkan
user: 24
user2: Berkan
user2: 24
```

Bu kod örneğinde, Object Property Shorthand konsepti kullanılarak JavaScript nesneleri oluşturuluyor. İlk olarak, **userName** ve **userAge** adında iki değişken tanımlanıyor ve bu değişkenlere değerler atanıyor. Daha sonra, bu değişkenler kullanılarak iki farklı nesne oluşturuluyor: **user** ve **user2**. İlk nesne olan **user**'da, nesne özellikleri geleneksel bir şekilde belirtiliyor. Her bir özellik için ayrı ayrı değişkenler kullanılıyor ve değerleri elle atanıyor. **userName** için **userName** değişkeni, **age** için ise **userAge** değişkeni kullanılıyor. İkinci nesne olan **user2** ise Object Property Shorthand kullanılarak oluşturuluyor. Burada, değişken adları doğrudan nesne özellikleri olarak kullanılıyor ve değişkenlerin değerleri otomatik olarak atanıyor. Yani, **userName** değişkeninin değeri **userName** özelliğine atanıyor, aynı şekilde **userAge** değişkeninin değeri de **userAge** özelliğine atanıyor. Her iki nesne de oluşturulduktan sonra, her birinin özellikleri ayrı ayrı konsola yazdırılıyor. Bu şekilde, Object Property Shorthand kullanarak daha az kod yazılarak aynı sonuç elde edildiği görülüyor.

## 2. Destructuring Assignment

JavaScript'te Destructuring assignment, bir dizi (array) veya nesne (object) içindeki değerleri ayrıştırarak değişkenlere atamayı sağlayan bir yapıdır. Bu yapı, bir dizi veya nesne içindeki öğeleri tek tek değişkenlere atama işlemini tek bir adımda gerçekleştirir.

Destructuring assignment, iki tür veri yapısı üzerinde kullanılabilir.

### 2.1 Array Destructuring

Array destructuring, JavaScript'te bir dizinin elemanlarına kolayca erişmek ve bu elemanları değişkenlere atamak için kullanılan bir tekniktir. Bu yapı, bir dizi içindeki elemanları ayıklamak ve bu elemanları tek tek veya toplu olarak değişkenlere atamak için kullanılır.

Array destructuring'i kullanırken, atamak istediğimiz değişkenleri köşeli parantezler içine alırız. Bu atama işlemi, dizi içindeki elemanların **sırasına** göre yapılır. Dizideki eleman sayısı, atama yapılacak değişken sayısından fazla ise, fazla olan elemanlar yok sayılır. Eğer atama yapılacak değişken sayısı, dizideki eleman sayısından fazlaysa, eksik olan değişkenler undefined olarak atanır.

Örnek olarak, bir dizi içindeki elemanları farklı değişkenlere atama işlemi şu şekilde gerçekleştirilir:

```
Project1 > JS index.js > ...
1 // Array Destructuring
2 let [a, b] = [10, 20];
3
4 console.log("a: ", a);
5 console.log("b: ", b);
6
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - Node

● \Project1> node index.js

○ a: 10  
b: 20

Array destructuring aynı zamanda atama işlemi sırasında belirli elemanları atlamak veya geri kalan elemanları tek bir değişkene toplamak için de kullanılabilir. Bu durumda, atlanacak elemanlar için boşluk bırakılır veya bir rest operatörü (...) kullanılır.

```
Project1 > JS index.js > ...
1 // Array Destructuring
2 let [a, b, ...c] = [10, 20, 30, 40, 50];
3
4 console.log("a: ", a);
5 console.log("b: ", b);
6 console.log("c: ", c);
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

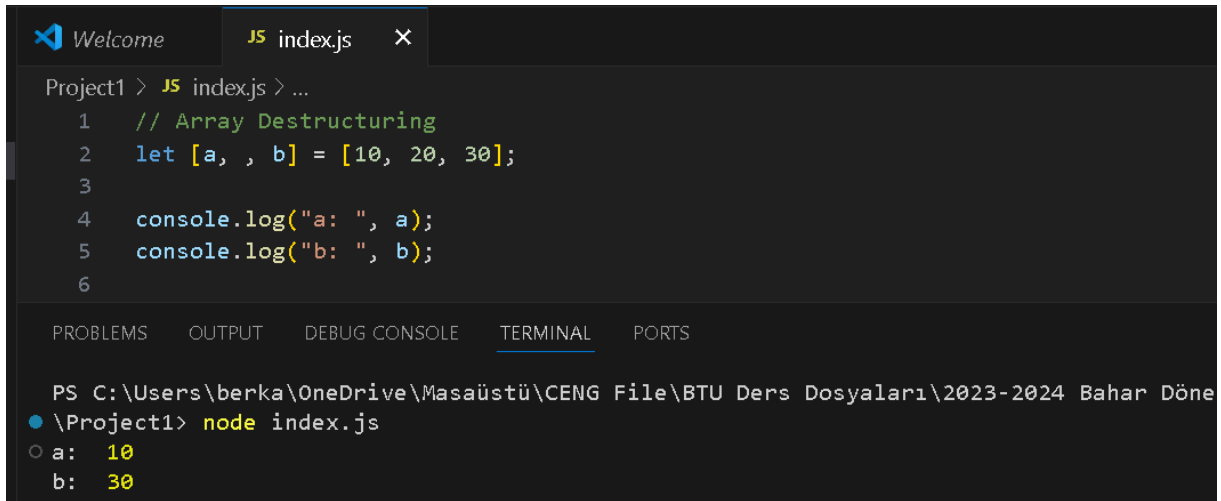
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024

● \Project1> node index.js

○ a: 10  
b: 20  
c: [ 30, 40, 50 ]

Şekil 1: Array Destructuring yönteminde Rest operatörünün kullanımı

Aşağıdaki kod örneğinde, [10, 20, 30] dizisindeki elemanlar, soldaki destructuring deseni ile eşleştirilir. Dizideki ilk eleman a değişkenine, ikinci eleman atanır ve üçüncü eleman b değişkenine atanır.

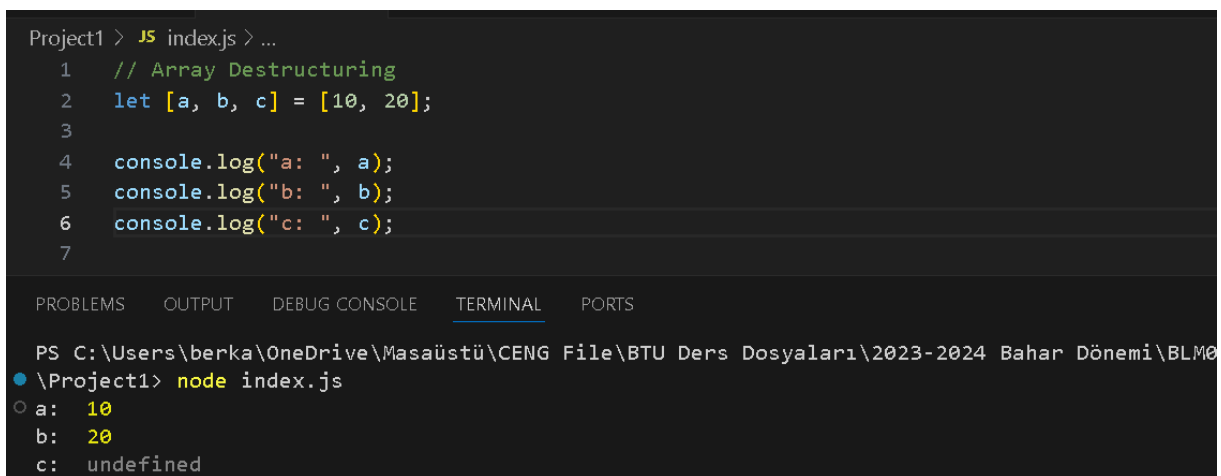


```
Project1 > JS index.js > ...
1 // Array Destructuring
2 let [a, , b] = [10, 20, 30];
3
4 console.log("a: ", a);
5 console.log("b: ", b);
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0
● \Project1> node index.js
○ a: 10
  b: 30
```

Aşağıdaki kullanım senaryosunda ise [10, 20] dizisinde bulunan elemanlar, soldaki destructuring deseni ile eşleştirilir. Ancak, desende üç değişken tanımlanmışken dizi sadece iki elemana sahiptir. Bu durumda, a değişkeni 10'a, b değişkeni 20'ye atanırken, c değişkeni için atama yapılamaz. c değişkeni için herhangi bir atama yapılmadığı için undefined değerini alır.



```
Project1 > JS index.js > ...
1 // Array Destructuring
2 let [a, b, c] = [10, 20];
3
4 console.log("a: ", a);
5 console.log("b: ", b);
6 console.log("c: ", c);
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0
● \Project1> node index.js
○ a: 10
  b: 20
  c: undefined
```

Bu durumda, undefined değeri almasını istemiyorsak, değişkenlere başlangıç değeri atayabiliriz. Örneğin, [10, 20] dizisinden c değişkenine atama yapamadığımızda, c değişkenine başlangıçta bir değer atayarak bu durumu önleyebiliriz. Bu şekilde, undefined yerine istediğimiz bir değer kullanılır. Bu sayede, atama yapılamayan durumlarda programın beklenmedik şekilde çalışmasını önleyebiliriz.

Aşağıdaki kod örneğinde, dizinin ilk iki değeri sırasıyla 'a' ve 'b' değişkenlerine atanırken, üçüncü bir değer olmadığı için 'c' değişkeninin undefined değerini almasını önlemek için 'c' değişkenine varsayılan olarak başlangıç değeri olarak 30 atanır. Sonuç olarak, 'a' değişkeni 10, 'b' değişkeni 20, ve 'c' değişkeni 30 değerini alır.

```
Project1 > JS index.js > ...
1  // Array Destructuring
2  let [a, b, c = 30] = [10, 20];
3
4  console.log("a: ", a);
5  console.log("b: ", b);
6  console.log("c: ", c);
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\
● \Project1> node index.js
a: 10
b: 20
c: 30
```

Aşağıdaki kod, bir dizi içindeki ilk üç değeri 'a', 'b' ve 'c' adlı değişkenlere sırasıyla atar. Ancak dizi içindeki fazla değerler (40 ve 50) kullanılmaz. Sonuç olarak, 'a' değişkeni 10, 'b' değişkeni 20, ve 'c' değişkeni 30 değerlerini içerir.

```
Welcome JS index.js X
Project1 > JS index.js > ...
1  // Array Destructuring
2  let [a, b, c] = [10, 20, 30, 40, 50];
3
4  console.log("a: ", a);
5  console.log("b: ", b);
6  console.log("c: ", c);
7
```

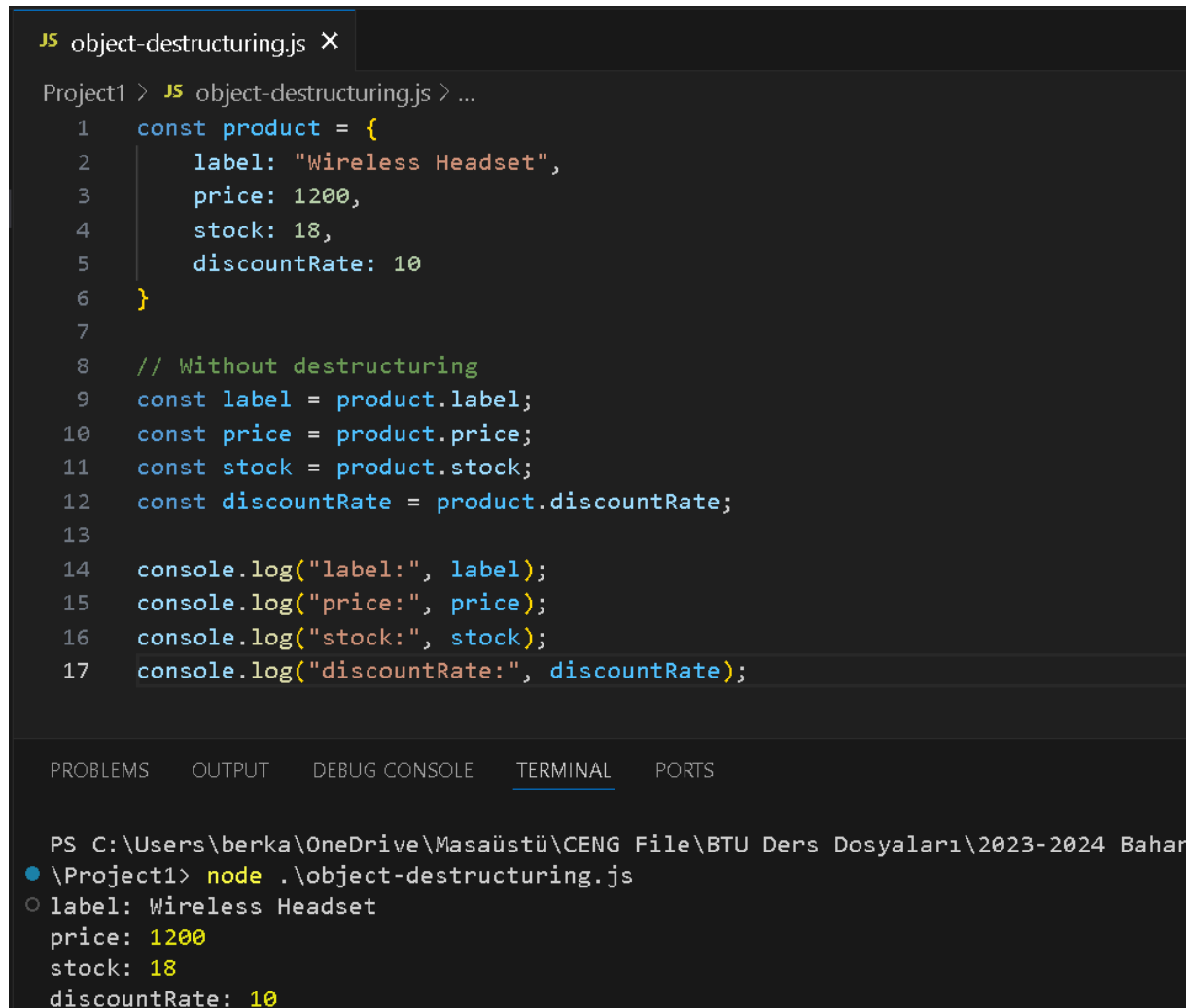
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - I
● \Project1> node index.js
○ a: 10
b: 20
c: 30
```

## 2.2 Object Destructuring

Object destructuring, JavaScript'te bir nesnenin içindeki özellikleri veya değerleri, özelliklerin isimlerine göre çözme veya çıkarma işlemidir. Bu yöntem, bir nesnenin içindeki özellikleri tek tek tanımlamak yerine, bir desen eşleştirmesi kullanarak hızlı bir şekilde erişmeyi sağlar. Object destructuring kullanırken, bir nesnenin özelliklerini çözmek istediğimiz değişkenlere atanır. Bu atama işlemi süslü parantezler `{}` içinde yapılır ve özelliklerin adlarına göre eşleştirilir. Özelliklerin adlarına göre eşleştirme yapılırken, nesnenin içindeki özelliklerin adları ile aynı adlara sahip değişkenler tanımlanır ve bu değişkenlere nesnenin ilgili özellikleri atanır. Destructuring işleminde değişkenlerin isimleri uyuşmaz ise destructuring işlemi yapılmaz.

Aşağıdaki görsel, bir nesnenin elemanlarını ayrı değişkenlere atamaktadır. Geleneksel yöntemle, her bir özelliğe nesne notasyonu kullanılarak erişilir ve ayrı değişkenlere atanır. Sonra, atanan değişkenler kullanarak değerler konsola yazdırılır. Destructuring kullanılmadan, bu işlem tekrarlayıcı ve uzun olmaktadır.



```
JS object-destructuring.js X
Project1 > JS object-destructuring.js > ...
1  const product = {
2      label: "Wireless Headset",
3      price: 1200,
4      stock: 18,
5      discountRate: 10
6  }
7
8  // Without destructuring
9  const label = product.label;
10 const price = product.price;
11 const stock = product.stock;
12 const discountRate = product.discountRate;
13
14 console.log("label:", label);
15 console.log("price:", price);
16 console.log("stock:", stock);
17 console.log("discountRate:", discountRate);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar
\Project1> node .\object-destructuring.js
label: Wireless Headset
price: 1200
stock: 18
discountRate: 10
```

Şekil 2: Destructuring kullanılmadan değer atama

Aynı işlem, destructuring yöntemi ile yapılmak istenirse aşağıdaki gibi daha kısa ve okunabilir bir şekilde gerçekleştirilebilir.

```
JS object-destructuring.js X
Project1 > JS object-destructuring.js > ...
1  const product = {
2    label: "Wireless Headset",
3    price: 1200,
4    stock: 18,
5    discountRate: 10
6  }
7
8  const {label, price, stock, discountRate} = product;
9
10 console.log("label:", label);
11 console.log("price:", price);
12 console.log("stock:", stock);
13 console.log("discountRate:", discountRate);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024
● \Project1> node .\object-destructuring.js
○ label: Wireless Headset
  price: 1200
  stock: 18
  discountRate: 10
```

Object destructuring'de, nesne özelliklerini alırken, bu özelliklere farklı isimler vermek mümkündür, yani nesnedeki özellik adlarıyla aynı isimleri kullanmak zorunda değiliz. Object destructuring'de isim değiştirmek, bir nesnedeki özelliklerin değerlerini alırken, bu değerleri farklı isimlere atama işlemidir. Yani, nesnenin özelliklerini alırken, bu özelliklere farklı isimler vererek kullanabiliriz.

```
16 const product = {
17   label: "Wireless Headset",
18   price: 1200,
19   stock: 18,
20   discountRate: 10
21 }
22
23 const {label: productLabel, price: productPrice, stock: productStock, discountRate: productDiscountRate} = product;
24
25 console.log("label:", productLabel);
26 console.log("price:", productPrice);
27 console.log("stock:", productStock);
28 console.log("discountRate:", productDiscountRate);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\
● \Project1> node .\object-destructuring.js
○ label: Wireless Headset
  price: 1200
  stock: 18
  discountRate: 10
```



Object destructuring'de, bir özelliğe değer atanmazsa, o özelliğin varsayılan bir değeri olabilir. Bu varsayılan değeri belirtmek için destructuring işlemi sırasında özellik adının yanına eşittir işareti ve varsayılan değer yazılır. Örneğin, `{ label = "Default Label" } = {}` şeklinde tanımlanan bir destructuring işleminde, eğer nesnede "label" adında bir özellik yoksa, bu özelliğin değeri "Default Label" olur. Bu şekilde, destructuring işlemi sırasında tanımlanan varsayılan değer, nesnede ilgili özellik bulunamadığında kullanılır.

Aşağıdaki kod örneği, bu durumu ele almaktadır.

```
16 const product = {
17
18 }
19
20 const {label: productLabel = "Headset", price: productPrice = "100", stock: productStock = "1", discountRate: productDiscountRate = "5"} = product;
21
22 console.log("label:", productLabel);
23 console.log("price:", productPrice);
24 console.log("stock:", productStock);
25 console.log("discountRate:", productDiscountRate);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta7\_RaporKo  
● \Project1> node .\object-destructuring.js  
label: Headset  
price: 100  
stock: 1  
discountRate: 5

**İç içe object destructuring, bir nesnenin içindeki başka bir nesnenin özelliklerini ayıklamak için kullanılır.** Örneğin, verilen bir user nesnesinde, skills adında başka bir nesne bulunabilir. Bu durumda, skills nesnesinin özelliklerine erişmek için iç içe object destructuring kullanılabilir. Kod örneğinde, user nesnesinin skills özelliğine destructuring uygulanmıştır. Bu sayede, skills özelliği içindeki frontEnd, backEnd ve mobile özellikleri doğrudan ayrıştırılmış ve ayrı değişkenlere atanmıştır. Sonuç olarak, frontEnd, backEnd ve mobile değişkenleri, user nesnesinin skills özelliğine erişmek için kullanılabilir. Bu şekilde, iç içe object destructuring kullanarak, derinleşen nesnelerin özelliklerine kolayca erişilebilir ve kod daha okunabilir hale gelir.

```
27 const user = {
28   firstName: "Berkan",
29   skills: {
30     frontEnd: ["React.js", "Next.js", "Angular"],
31     backEnd: ["NodeJS", ".NET Core"],
32     mobile: [".NET MAUI", "React Native"]
33   }
34 }
35
36 const {skills: {frontEnd, backEnd, mobile}} = user;
37
38 console.log("Front-end:", frontEnd);
39 console.log("Back-end:", backEnd);
40 console.log("Mobile:", mobile);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470  
● \Project1> node .\object-destructuring.js  
Front-end: [ 'React.js', 'Next.js', 'Angular' ]  
Back-end: [ 'NodeJS', '.NET Core' ]  
Mobile: [ '.NET MAUI', 'React Native' ]

Aşağıdaki görseldeki kod parçası, hem array destructuring'i hem de object destructuring'i aynı anda kullanarak, bir dizi içindeki nesnelerin özelliklerine erişmeyi sağlar. Kodun amacı, props adlı bir dizinin içindeki üçüncü nesnenin name özelliğine erişmektir. Bunun için array destructuring kullanılarak props dizisinden üçüncü nesne ayrıştırılır, ancak bu nesnenin içindeki özelliklere object destructuring uygulanır. { name: text } ifadesiyle, üçüncü nesnenin name özelliği text adlı bir değişkene atanır. Sonuç olarak, console.log(text) ifadesiyle text değişkeni ekrana yazdırılır, bu da üçüncü nesnenin name özelliği olan "FizzBuzz" değerini görüntüler. Bu şekilde, kod hem array hem de object destructuring'i kullanarak dizinin içindeki bir nesnenin belirli bir özelliğine erişir.

```
40 const props = [
41   { id: 1, name: "Fizz" },
42   { id: 2, name: "Buzz" },
43   { id: 3, name: "FizzBuzz" },
44 ]
45
46 const [, , { name: text }] = props;
47
48 console.log(text);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\Project1> node .\object-destructuring.js  
FizzBuzz

### 3. Hava Durumu Uygulaması

Bu kısımda basitçe bir hava durumu uygulaması geliştireceğiz. Bu uygulama basitçe kullanacağımız dış bir API servisinden GET istekleri gerçekleştirecek. Bu GET isteklerini gerçekleştirirken callback fonksiyonlardan yararlanacağız. Ayrıca fonksiyonları yazarken bir önceki bölümde öğrendiğimiz object destructuring yöntemini kullanacağız.

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

```
package.json
WeatherApp > package.json > ...
1 {
2   "name": "weatherapp",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
12 }
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

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

Şimdi de uygulama boyunca kullanacağımız paketleri **npm install** komutunun yardımıyla uygulamamıza dahil edelim. Kullanacağımız paketler **dotenv** ve **postman-request** paketleridir.

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

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - Node.js\WeatherApp> npm i dotenv postman-request
npm WARN deprecated har-validator@5.1.5: this library is no longer supported

added 56 packages, and audited 57 packages in 15s

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

Verileri çekmek için kullanacağımız sitelere üye olup API erişim anahtarlarını uygulamamızın .env dosyasına dahil etmemiz gerekmektedir. Bunun için aşağıdaki sitelere üye olmamız gerekmektedir.

- <https://weatherstack.com/signup/free>

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

Her iki siteye de başarılı bir şekilde üye olduktan sonra API erişim anahtarlarını aşağıdaki görseldeki gibi .env dosyamıza dahil edelim.

```
WeatherApp > .env
1 WEATHER_API_KEY="8160d4b42bcd0921f6c9c96"
2 MAPBOX_API_KEY="pk.eyJ1IjoieWVya2Fuc2VyYmVzIiwiaYSI6ImNsdHpoYjd4dTAWMWEybXBkNWgwOWE2NW0ifQ.rL_r0AAc"
```

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

utils klasörünün içerisine forecast.js adında bir dosya oluşturalım. Bu dosyada olacak fonksiyon, weatherstack API'sini kullanarak hava durumu verilerini bir callback fonksiyonu aracılığıyla çekecek.

```

WeatherApp > utils > JS forecast.js > forecast > request() callback
1  require("dotenv").config();
2  const request = require("postman-request");
3
4  const forecast = (longitude, latitude, callback) => {
5      const weatherBaseUrl = `http://api.weatherstack.com/current?access_key=${process.env.WEATHER_API_KEY}&query=${longitude},${latitude}`;
6
7      try {
8          request({url: weatherBaseUrl, json: true}, (error, {body}) => {
9              if(error) {
10                 callback("Unable to connect to weather services!", undefined);
11             } else if(body.error) {
12                 callback("Location cannot found", undefined);
13             } else {
14                 const {temperature, feelslike} = body.current;
15                 callback(undefined, {temperature, feelslike});
16             }
17         })
18     }
19     catch(err) {
20         callback(err.message, undefined);
21     }
22 }
23
24 module.exports = forecast;

```

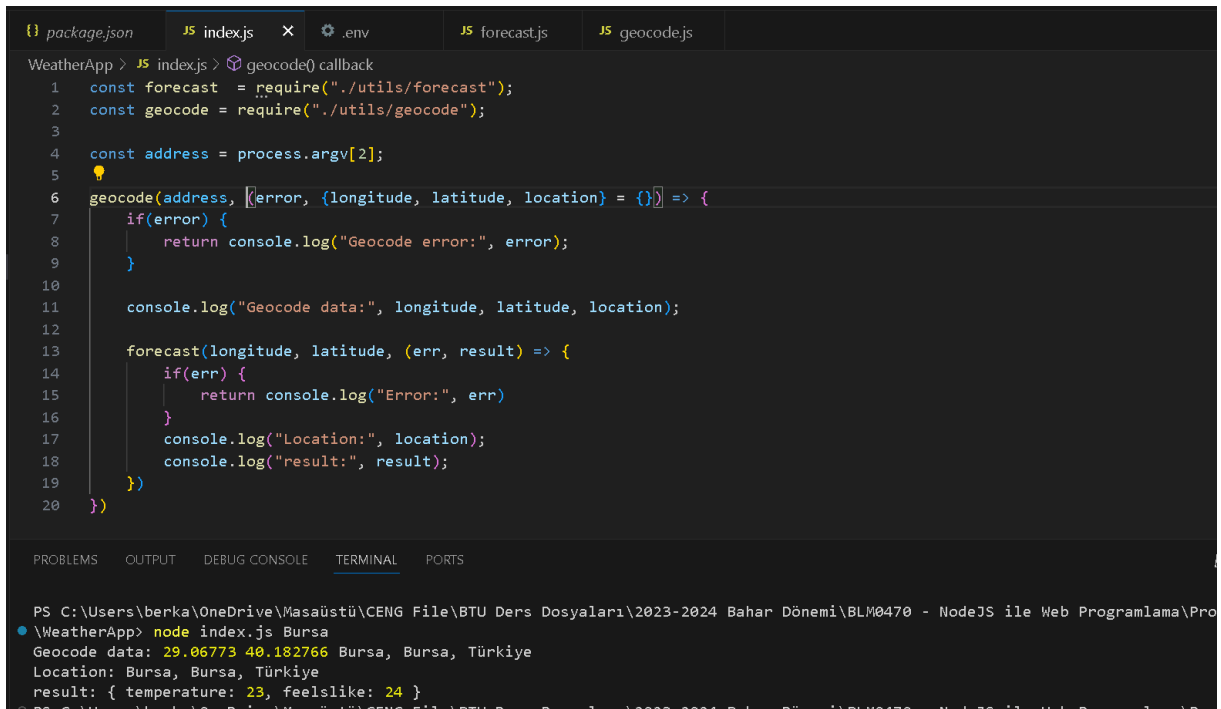
Yukarıdaki kod parçası, bir hava durumu API'si olan weatherstack'i kullanarak hava durumu verilerini çekmek için tasarlanmış bir fonksiyon olan forecast fonksiyonunu içerir. Bu fonksiyon, bir konumun boylam ve enlem bilgilerini alır ve bu bilgilere göre ilgili hava durumu verilerini API'den çeker. Fonksiyonun içeriği şu adımları izler: Öncelikle, parametrelerden alınan boylam ve enlem bilgileri kullanılarak bir URL oluşturulur. Bu URL, API'ye istek göndermek için kullanılacak olan adresi içerir. Daha sonra, request modülü aracılığıyla API'ye istek gönderilir. Bu istek, oluşturulan URL ile birlikte yapılır. API'den gelen yanıt değerlendirilir. Eğer bir hata varsa veya istek başarısız olursa, uygun hata mesajlarıyla birlikte callback fonksiyonuna bilgi gönderilir. Eğer herhangi bir hata yoksa ve istek başarılıysa, API'den gelen verilerden sıcaklık ve hissedilen sıcaklık bilgileri alınarak, bu bilgilerle birlikte callback fonksiyonuna başarılı bir şekilde veri gönderilir. Son olarak, bu fonksiyon module.exports ile dışarıya açılarak, başka dosyalardan erişilebilir hale getirilir. Bu sayede, hava durumu verilerini çekmek için bu fonksiyon başka dosyalarda kullanılabilir. Aynı zamanda 8 ve 14. satırlarda object destructuring kullanılarak objenin ilgili özelliklerine kolayca erişilmiştir.

```

WeatherApp > utils > JS geocode.js > ...
1  require("dotenv").config();
2  const request = require("postman-request");
3
4  const geocode = (address, callback) => {
5      const mapBoxBaseUrl = `https://api.mapbox.com/geocoding/v5/mapbox.places/${encodeURIComponent(
6          address
7      )}.json?access_token=${process.env.MAPBOX_API_KEY}`;
8
9      request({url: mapBoxBaseUrl, json: true}, (error, {body}) => {
10         try{
11             if(error) {
12                 callback("Unable to connect to location services!", undefined);
13             } else if(body.features.length === 0) {
14                 callback("Location cannot found", undefined);
15             }
16             const [longitude, latitude] = body.features[0].center;
17             callback(undefined, { longitude, latitude, location: body.features[0].place_name});
18         }
19         catch(err) {
20             callback(err.message, undefined);
21         }
22     })
23 }
24 module.exports = geocode;

```

Yukarıdaki kod parçası, bir adresin coğrafi konumunu belirlemek için Mapbox API'sini kullanır. İlk olarak, "dotenv" ve "postman-request" modülleri projeye dahil edilir. "dotenv" modülü, çevresel değişkenlerin projede kullanılmasını sağlar. "postman-request" modülü ise HTTP isteklerini yapmak için kullanılır. Ardından, "geocode" adında bir fonksiyon oluşturulur. Bu fonksiyon, bir adres ve bir callback fonksiyonu alır. Adres, Mapbox API'sine gönderilecek olan sorguyu oluşturmak için kullanılır. Oluşturulan URL, Mapbox API'sine gönderilen bir GET isteği yapmak için kullanılır. encodeURIComponent fonksiyonu, belirli karakterleri URL içinde güvenli bir şekilde kullanılabilir hale getirmek için kullanılır. URL'lerde özel anlama sahip olan bazı karakterler, doğrudan kullanıldıklarında hatalara veya istenmeyen sonuçlara neden olabilir. Örneğin, URL'lerde boşluk yerine "+" işareti veya "%20" gibi bir işaretle temsil edilir. Bu istek, belirtilen adrese göre bir coğrafi konum döndürür. İstek gönderildiğinde, bir callback fonksiyonu kullanılır. Bu callback fonksiyonu, API'den gelen yanıtı işlemek için kullanılır. Eğer istekte herhangi bir hata oluşursa, bu hata "error" parametresi aracılığıyla yakalanır ve geri çağırma fonksiyonuna iletilir. Eğer istek başarılı olursa, API'den gelen yanıt object destructuring yardımıyla "body" nesnesi aracılığıyla işlenir. Yanıtta hata yoksa ve belirtilen adrese karşılık gelen coğrafi konum bilgileri bulunuyorsa, bu bilgiler callback fonksiyonuna iletilir. Son olarak, "geocode" fonksiyonu module.exports ile dışarıya açılır ve başka dosyalarda kullanılabilir hale gelir.



```
1 const forecast = require("../utils/forecast");
2 const geocode = require("../utils/geocode");
3
4 const address = process.argv[2];
5
6 geocode(address, (error, {longitude, latitude, location} = {}) => {
7   if(error) {
8     return console.log("Geocode error:", error);
9   }
10
11   console.log("Geocode data:", longitude, latitude, location);
12
13   forecast(longitude, latitude, (err, result) => {
14     if(err) {
15       return console.log("Error:", err);
16     }
17     console.log("Location:", location);
18     console.log("result:", result);
19   })
20 })
```

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

Geocode data: 29.06773 40.182766 Bursa, Bursa, Türkiye

Location: Bursa, Bursa, Türkiye

result: { temperature: 23, feelslike: 24 }

Bu kod bizim ana fonksiyonumuzdur ve diğer iki yazdığımız fonksiyon burada kullanılır. Öncelikle, komut satırından girilen adres bilgisi process.argv[2] ile alınır ve address değişkenine atanır. Ardından, bu adres bilgisi geocode fonksiyonuna iletilir. geocode fonksiyonu, verilen adrese göre enlem ve boylam koordinatlarını ve konum adını belirlemek için harita servisine bir istek gönderir. Bu işlem asenkron olarak gerçekleşir, dolayısıyla bir geri çağırma fonksiyonu kullanılır. Eğer bir hata oluşursa, hata mesajı konsola yazdırılır. Başarılı bir şekilde konum bilgileri alınır, bu bilgiler longitude, latitude ve location değişkenlerine atanır ve hava durumu tahmini almak için forecast fonksiyonu çağırılır. Benzer

şekilde, forecast fonksiyonu da asenkron olarak çalışır ve bir geri çağırma fonksiyonu kullanır. Eğer bir hata oluşursa, hata mesajı konsola yazdırılır. Başarılı bir şekilde hava durumu tahmini alınırsa, konum ve tahmin bilgileri konsola yazdırılır.

## 4.Express.js

Express.js, web uygulamaları ve API'ler oluşturmak için kullanılan hızlı, esnek ve minimalist bir web uygulama çatısıdır. Node.js üzerinde çalışır ve modern web geliştirme projeleri için yaygın olarak tercih edilen bir çözümdür.

Express.js'in temel avantajlarından biri, minimalist yapısıdır. Basit ve özelleştirilebilir bir yapı sunar, bu da geliştiricilere ihtiyaçlarına göre uygulama geliştirme ve ölçeklendirme esnekliği sağlar. Basit bir HTTP sunucusundan karmaşık web uygulamalarına kadar çeşitli ihtiyaçları karşılayabilir.

Express.js ayrıca güçlü bir rota (route) yönetimi sağlar. URL rotalarını tanımlamak ve bu rotalara istenilen HTTP metodlarını (GET, POST, PUT, DELETE vb.) atamak kolaydır. Bu, isteklerin belirli işlemlere yönlendirilmesini ve istemciye yanıt verilmesini kolaylaştırır.

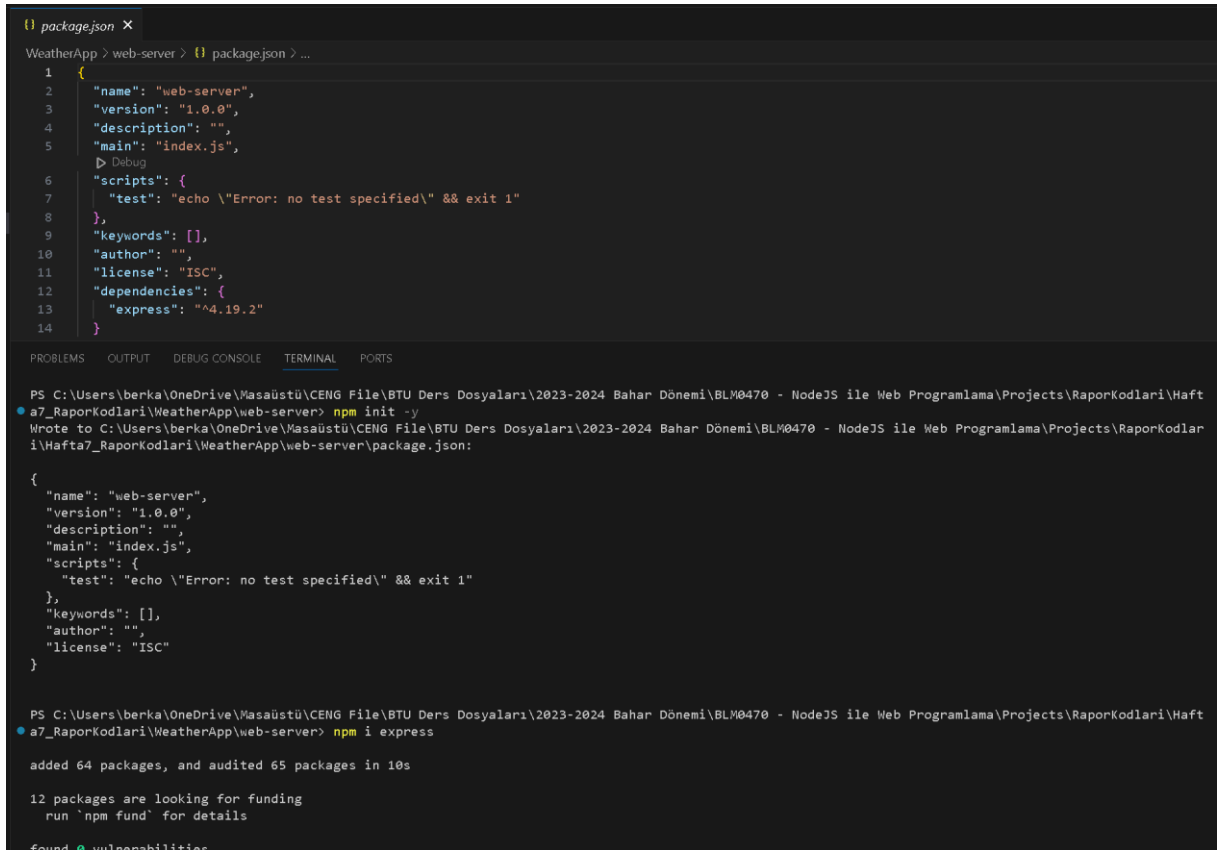
Express.js'in middleware sistemi, gelen istekleri işlemek ve yanıtlamak için esnek bir yapı sağlar. Middleware'ler, isteklerin işlenmesi veya yanıt verilmesi öncesinde veya sonrasında ek işlemler gerçekleştirmek için kullanılır. Örneğin, oturum yönetimi, kimlik doğrulama, güvenlik kontrolleri, günlük tutma gibi işlemler middleware'ler aracılığıyla kolayca eklenir.

Express.js'in geniş bir ekosistemi vardır. Birçok üçüncü taraf modül ve eklentisi bulunur, bu da geliştiricilere projelerine ek işlevsellikler eklemeleri için birçok seçenek sunar. Örneğin, veritabanı entegrasyonu için MongoDB, PostgreSQL gibi popüler veritabanlarına erişim sağlayan modüller mevcuttur.

### 4.1 Hava Durumu Uygulamasına Express.js'i Dahil Etme

Uygulamamız şu ana kadar komut satırı üzerinden alınan argümanlar veya kodlara yazılan çeşitli parametrelerle çalışıyordu. Ancak şimdi, Express.js'in gücünden faydalanarak, hem bir API hem de etkileşimli bir kullanıcı arayüzü oluşturmak için adımlar atabiliriz. Express.js, uygulamamızın gelişmiş özelliklerini kullanarak web sunucusu oluşturmayı sağlar ve HTTP isteklerini yönlendirir. Bu sayede, kullanıcılar web tarayıcıları aracılığıyla uygulamamıza erişebilir ve etkileşimli bir deneyim yaşayabilirler. Express.js'in sağladığı hızlı ve etkili yöntemlerle, kullanıcılarımıza daha iyi bir kullanıcı deneyimi sunabilir ve uygulamamızı daha da geliştirebiliriz.

Var olan projemize web-server adında bir klasör oluşturalım ve bu klasörün içine terminal yardımıyla gidip **npm init -y** komutunu kullanarak package.json dosyamızı oluşturalım. package.json dosyası oluştuktan sonra express paketini npm i express komutuyla projemize dahil edelim.



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

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta7_RaporKodları\WeatherApp\web-server> npm init -y
Wrote to C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta7_RaporKodları\WeatherApp\web-server\package.json:

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

PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\RaporKodları\Hafta7_RaporKodları\WeatherApp\web-server> npm i express

added 64 packages, and audited 65 packages in 10s

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

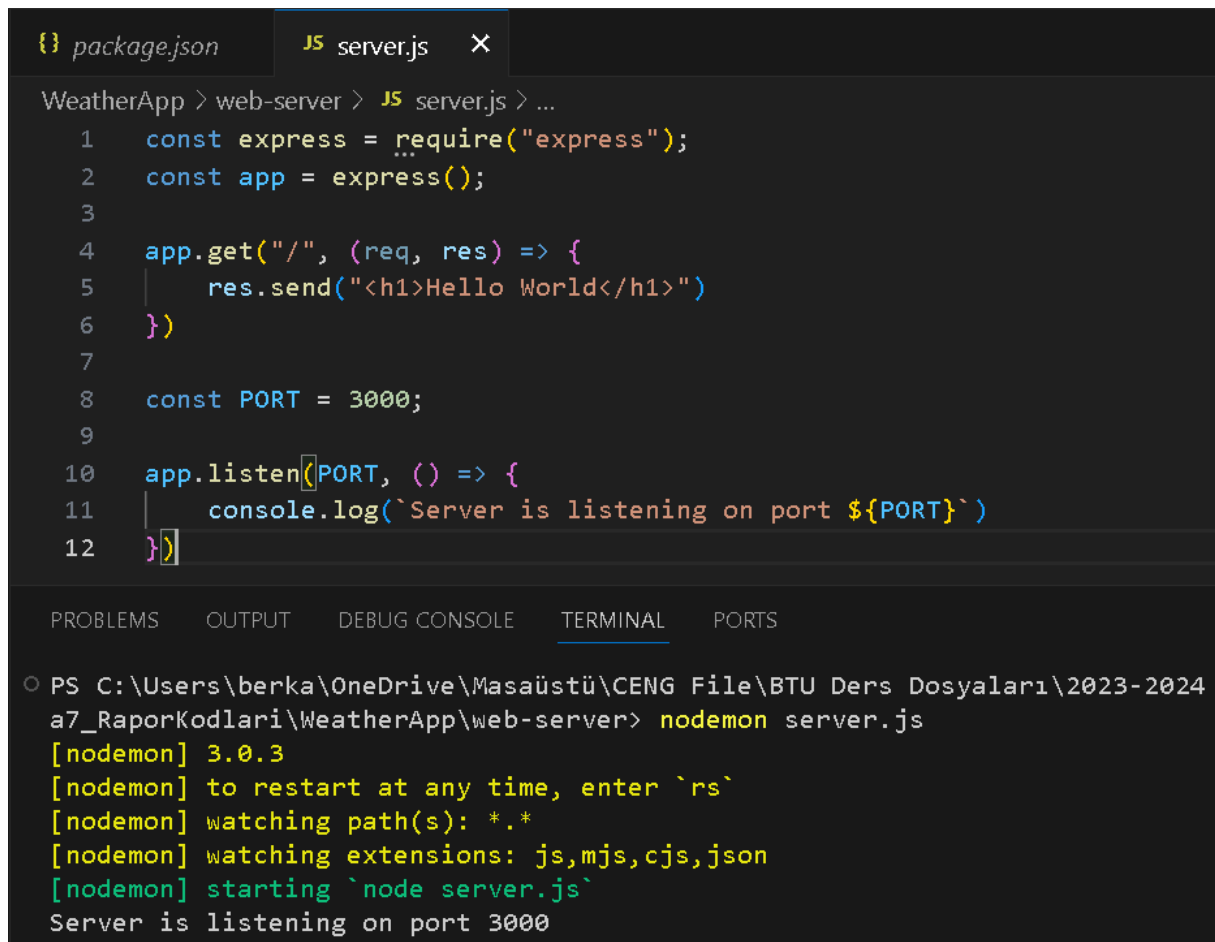
found 0 vulnerabilities
```

Daha sonra server.js adında bir dosya oluşturalım. Bu dosya bizim için uygulamanın ayağa kalktığı ana dosya olacak.

Aşağıdaki görselde bulunan kod, Express.js kullanarak basit bir web sunucusu oluşturmayı göstermektedir. İlk olarak, express modülünü projemize require fonksiyonuyla dahil ediyoruz ve bu modülü kullanarak bir Express uygulaması oluşturuyoruz. Ardından, app.get() metodunu kullanarak HTTP GET isteği geldiğinde ("/" rotası), bir işlevi çalıştırıyoruz. Bu işlev, istemciye "Hello World" başlığı içeren bir HTML yanıtı gönderir.

Sonrasında, bir port numarası belirleyerek sunucumuzu bu porta bağlıyoruz. app.listen() metodu, belirtilen porta gelen HTTP isteklerini dinlemeye başlar. Dinleme işlemi başladığında, bir geri çağırma işlevi çalıştırılır ve konsola "Server is listening on port 3000" şeklinde bir mesaj yazdırılır.

Bu kod, temel bir Express.js uygulamasını oluşturmak için kullanılan standart bir yapıyı göstermektedir. Bu yapı, HTTP isteklerine yanıt veren ve belirli rotalara göre işlem yapan bir web sunucusu oluşturmak için kullanılır.



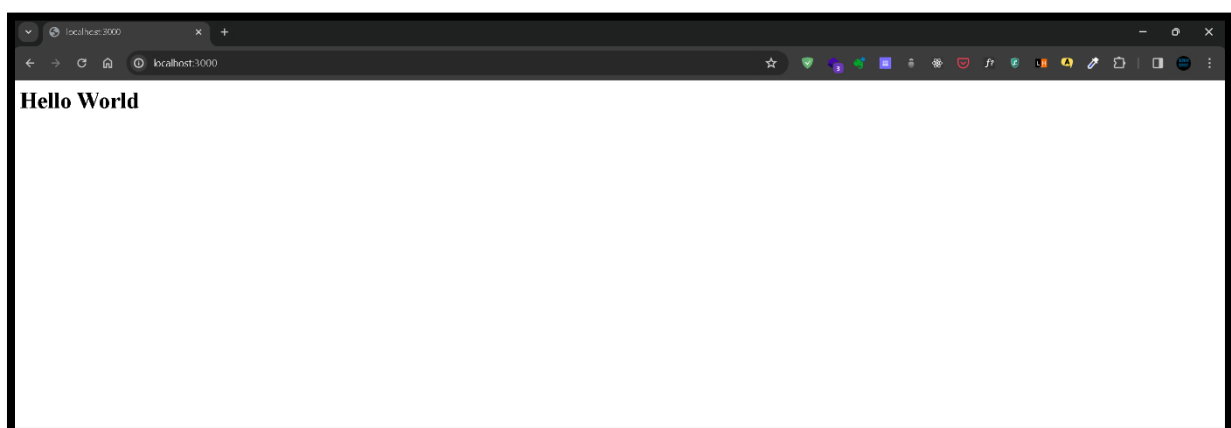
The image shows a VS Code editor window with two tabs: `package.json` and `JS server.js`. The `server.js` file contains the following code:

```
1  const express = require("express");
2  const app = express();
3
4  app.get("/", (req, res) => {
5    res.send("<h1>Hello World</h1>")
6  })
7
8  const PORT = 3000;
9
10 app.listen(PORT, () => {
11   console.log(`Server is listening on port ${PORT}`)
12 })
```

Below the code editor, the `TERMINAL` tab is active, showing the command `nodemon server.js` and its output:

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024
a7_RaporKodlari\WeatherApp\web-server> nodemon server.js
[nodemon] 3.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server is listening on port 3000
```

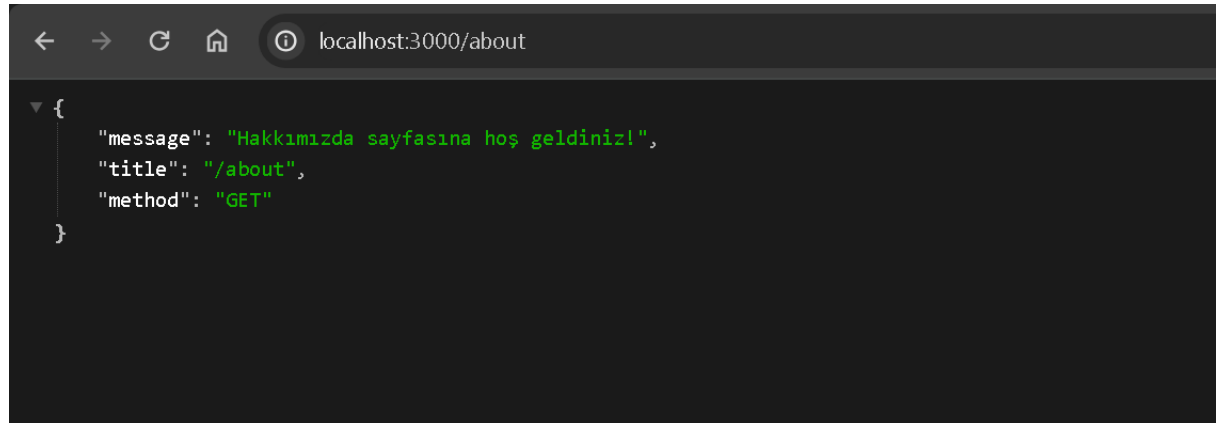
Tarayıcımızda localhost:3000 adresine gittiğimizde, Express uygulamamız tarafından sunulan "/" rotasına gönderdiğimiz istek sonucunda tarayıcıda "Hello World" başlığıyla bir HTML sayfası görüntülenir. Bu, Express uygulamamızın belirlediğimiz portta (3000) çalıştığını ve HTTP isteklerini doğru şekilde işlediğini gösterir. Başka rotalara yapılan istekler de benzer şekilde belirlenen işlevlere yönlendirilerek ilgili yanıtlar tarayıcıya gönderilir.





Benzer şekilde route'lara yalnızca düz bir yazı değil json formatında veriler de gönderebiliriz.

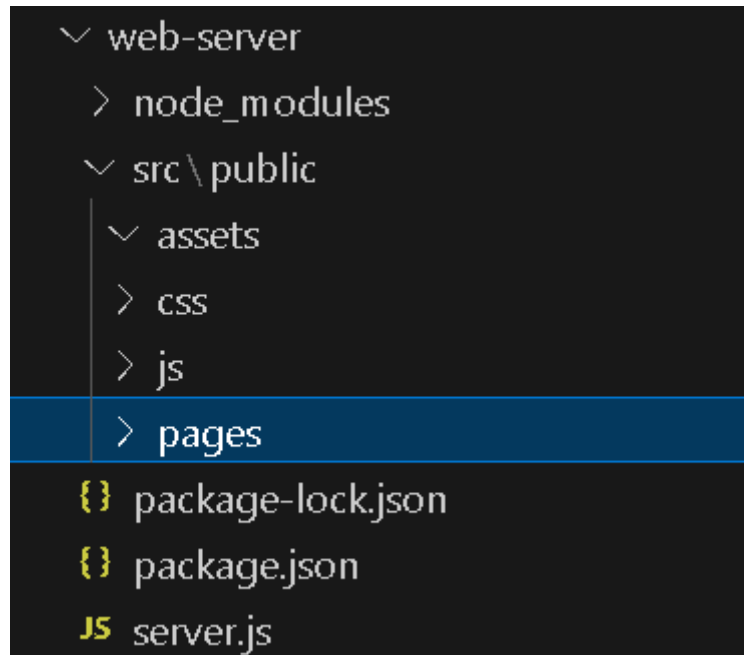
```
7
8 app.get("/about", (req, res) => {
9   res.status(200).json({message: "Hakkımızda sayfasına hoş geldiniz!", title: req.url, method: req.method });
10 });
11
12
```



## 4.2 Statik HTML Sayfaları Oluşturma

Statik HTML sayfaları oluşturma, web geliştirme sürecinde temel bir adımdır. Bu süreçte HTML, CSS ve JavaScript gibi web teknolojilerini kullanarak belirli bir tasarımı veya içeriği temsil eden sabit sayfalar oluşturulur. Bu sayfalar, web tarayıcısı tarafından doğrudan işlenir ve kullanıcıya sunulur. Sayfaların yapısı HTML ile tanımlanırken, görünüm ve stil CSS ile belirlenir. JavaScript ise sayfanın davranışlarını ve etkileşimlerini kontrol eder.

Projemizr “src” klasörü ekleyelim ve bu klasörün altına “public” adında bir klasör daha oluşturalım. Sonrasında public klasörünü aşağıdaki şekildeki gibi klasörlere ayıralım.



- **assets:** Bu klasör, genellikle resimler, videolar, fontlar ve diğer medya dosyalarını içerir. Bu dosyalar, web sayfasının içeriğini zenginleştirmek veya görsel olarak daha çekici hale getirmek için kullanılır.
- **pages:** Bu klasör, web sitesinin farklı sayfalarını içerir. Her bir HTML dosyası bir sayfaya karşılık gelir. Örneğin, anasayfa, hakkımızda sayfası, iletişim sayfası gibi farklı içeriklere sahip sayfalar bu klasör altında yer alabilir.
- **css:** CSS dosyalarının bulunduğu klasördür. CSS dosyaları, HTML içeriğinin stilini belirlemek için kullanılır. Sayfanın tasarımı, renkleri, yazı tipleri, düzeni ve diğer görsel özellikler bu dosyalar aracılığıyla kontrol edilir.
- **js:** JavaScript dosyalarının bulunduğu klasördür. JavaScript, web sayfalarına dinamik özellikler kazandırmak için kullanılır. Bu dosyalar, sayfanın davranışlarını kontrol etmek, kullanıcı etkileşimlerini işlemek veya sayfa içeriğini dinamik olarak değiştirmek için kullanılabilir.

Bu HTML belgesi, bir hava durumu uygulamasının ana sayfasını tanımlar. Başlık ve meta etiketleri belge özelliklerini belirtirken, bağlantılar aracılığıyla CSS ve JavaScript dosyaları eklenir. Sayfa içeriği, bir navigasyon çubuğu ve bir logo içeren bir div ile başlar. Son olarak, JavaScript dosyası belgeye bağlanarak sayfanın davranışları kontrol edilir.

```
WeatherApp > web-server > src > public > pages > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Weather</title>
7      <link rel="stylesheet" href="../css/styles.css" />
8      <script src="../js/app.js"></script>
9    </head>
10   <body>
11     <div class="navbar-container">
12       <div>
13         <h1 class="app-name">Weather App</h1>
14       </div>
15       <div class="link-container">
16         <a href="/">Home</a>
17         <a href="/about">About</a>
18         <a href="/weather">Weather</a>
19       </div>
20       <div>
21         
22       </div>
23     </div>
24   </body>
25 </html>
26
```

styles.css dosyamızın içeriği de aşağıdaki gibidir.

```
package.json server.js index.html styles.css app.js
WeatherApp > web-server > src > public > css > styles.css > h1
1 body {
2   background-color: #f5f5f5;
3   margin: 0;
4   padding: 0;
5   box-sizing: border-box;
6 }
7 .navbar-container {
8   padding: 0px 30px;
9   background-color: black;
10  height: 10vh;
11  display: flex;
12  justify-content: space-between;
13  align-items: center;
14 }
15 .link-container {
16   display: flex;
17   gap: 5rem;
18   justify-content: center;
19   align-items: center;
20   height: 100%;
21   > a {
22     color: white;
23     font-size: 1.5rem;
24     text-decoration: none;
25     &:hover {
26       color: aqua;
27       text-decoration: underline;
28       transition: ease-in-out 0.3s all;
29     }
30   }
31 }
32 h1 {
33   color: brown;
34   text-align: center;
35 }
36
37 .app-name {
38   color: antiquewhite;
39 }
40
41 .logo {
42   width: 50px;
43   height: 50px;
44 }
```

Oluşturulan dosyalarımızı ana çalıştırma dosyamızda yani “**server.js**” dosyamıza tanımlamamız gerekmektedir. İlk olarak, “**path**” modülü require ile import edilir ve “publicDirectory” ve “pagesDirectory” adında iki değişken oluşturulur. publicDirectory, \_\_dirname (uygulamanın ana dizini) ve “src/public” dizinini birleştirerek statik dosyaların bulunduğu dizinin tam yolunu belirtir. pagesDirectory ise publicDirectory içindeki “pages” dizinine yol oluşturur.

Daha sonra, express.static middleware'i kullanarak publicDirectory dizinini sunucu tarafında erişilebilir hale getiririz. Bu sayede tarayıcıya gönderilen istekler, bu dizindeki dosyalara erişebilir.

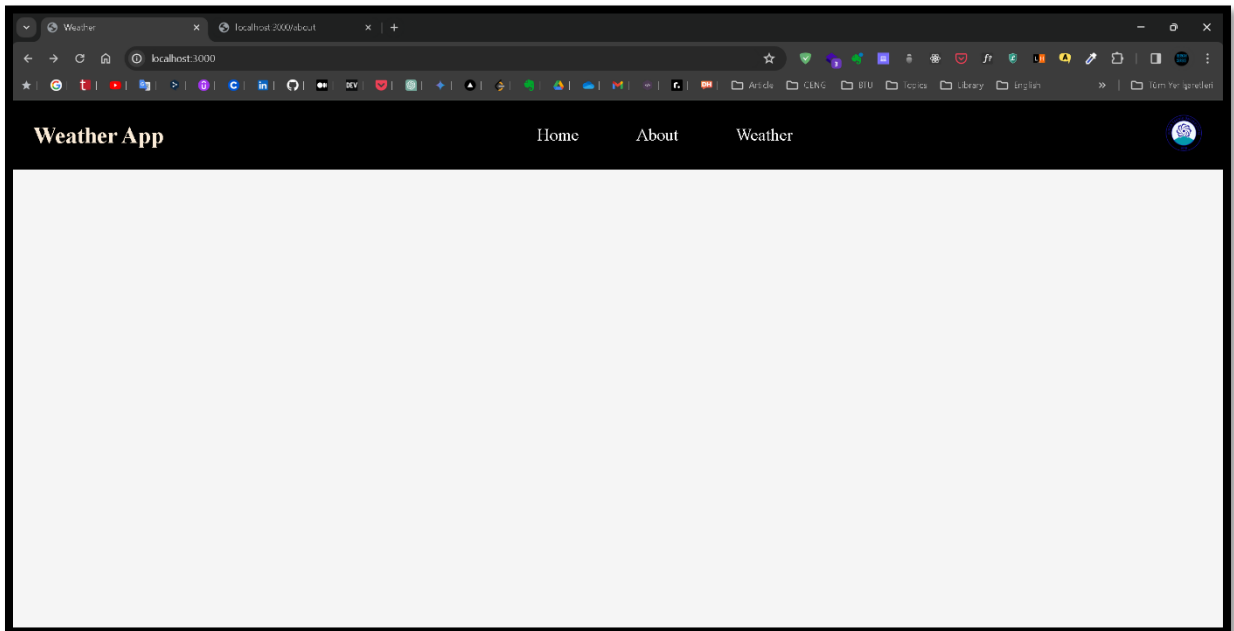
Son olarak, “/” rotası tanımlanır ve bu rotaya gelen istekler, pagesDirectory içindeki “index.html” dosyasını kullanıcıya göndermek için res.sendFile() yöntemiyle işlenir. Bu

sayede, tarayıcı "/"'a yoluna geldiğinde, index.html dosyası gönderilir ve ana sayfa görüntülenir.

Aşağıdaki görsel server.js dosyamızın güncel halini göstermektedir.

```
WeatherApp > web-server > .js server.js > ...
1  const express = require("express");
2  const app = express();
3  const path = require("path");
4
5  const publicDirectory = path.join(__dirname, "./src/public");
6  const pagesDirectory = path.join(publicDirectory, "./pages");
7
8  app.use(express.static(publicDirectory));
9
10 app.get("/", (req, res) => {
11   res.sendFile(path.join(pagesDirectory, "index.html"));
12 });
13
14 app.get("/about", (req, res) => {
15   res.status(200).json({
16     message: "Hakkımızda sayfasına hoş geldiniz!",
17     title: req.url,
18     method: req.method,
19   });
20 });
21
22 const PORT = 3000;
23
24 app.listen(PORT, () => {
25   console.log(`Server is listening on port ${PORT}`);
26 });
```

Uygulamayı çalıştırıp "/" sayfasına gittiğimizde ise aşağıdaki gibi bir sayfayla karşılaşacağız.



## 4.3 Handlebars Paketi

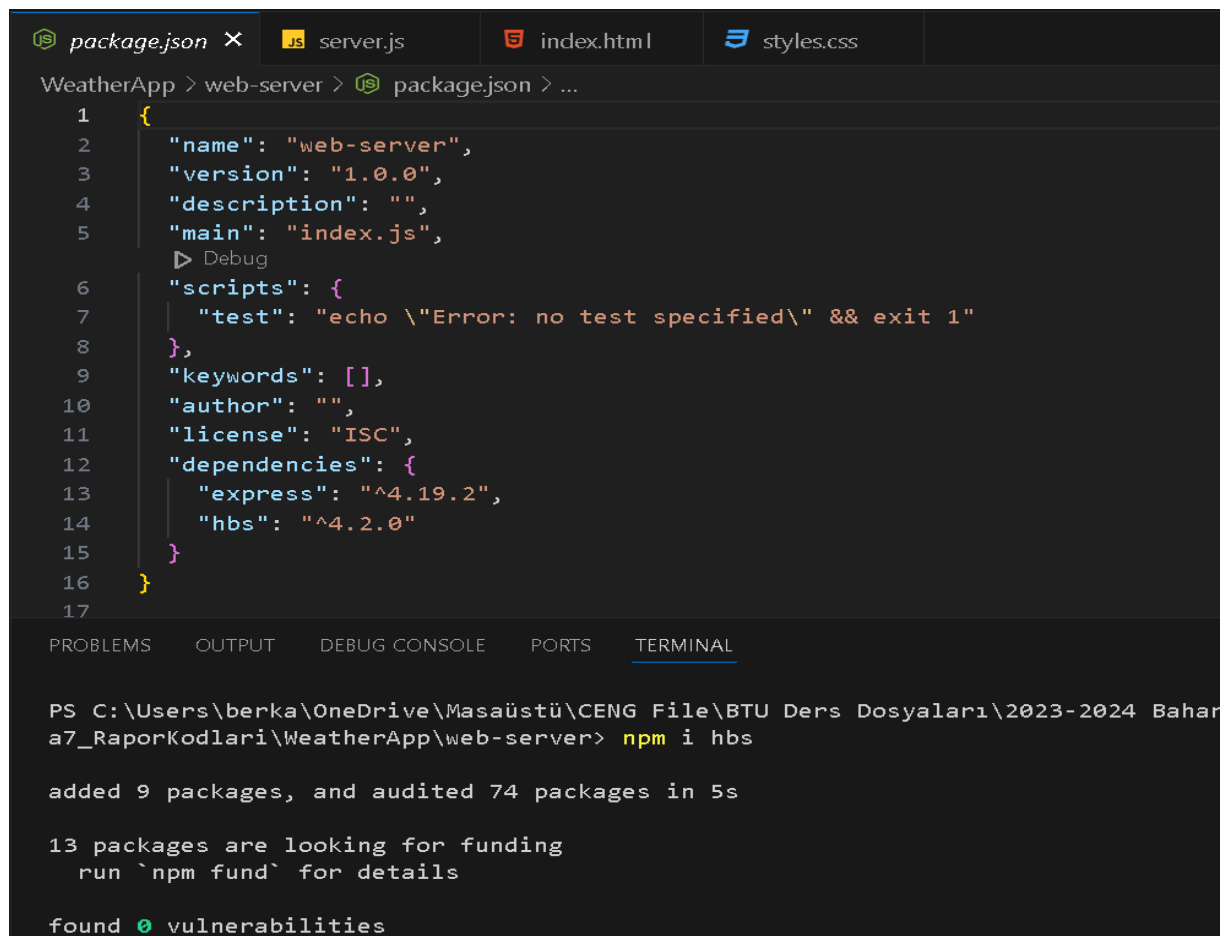
Şimdi de uygulamamızı bir adım öteye taşıyalım. Dinamik içerikler oluşturmak için uygulamamıza bir şablon motoru dahil edelim. Şablon motoru olarak kullanacağımız paketin adı hbs (Handlebars)'dir.

hbs (Handlebars) modülü, Express.js ile birlikte kullanılan ve dinamik HTML sayfaları oluşturmayı kolaylaştıran bir şablon motorudur. Bu modül, sunucu tarafında veri ile HTML dosyalarının birleştirilmesini sağlar ve böylece dinamik içerik oluşturulabilir.

Handlebars, statik HTML dosyalarınızın içine dinamik olarak değişkenleri, döngüleri, koşullu ifadeleri ve diğer JavaScript kodlarını eklemenizi sağlar. Bu sayede, sunucu tarafında belirli bir veriye göre HTML içeriği dinamik olarak oluşturulabilir ve istemciye gönderilebilir.

Handlebars, {{}} süslü parantezler içindeki değişkenler, blok ifadeleri ve yardımcı fonksiyonları kullanarak şablonları işler. Sunucu tarafında verilerle doldurulmuş bir Handlebars şablonu, istemciye HTML olarak gönderilir ve tarayıcıda render edilir.

Komut satırımızı açalım ve **npm i hbs** komutunu kullanarak hbs paketini uygulamamıza dahil edelim.



```
package.json x server.js index.html styles.css
WeatherApp > web-server > package.json > ...
1 {
2   "name": "web-server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.19.2",
14    "hbs": "^4.2.0"
15  }
16 }
17

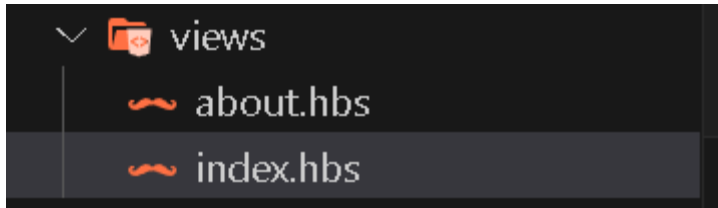
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar a7_RaporKodlari\WeatherApp\web-server> npm i hbs

added 9 packages, and audited 74 packages in 5s

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

found 0 vulnerabilities
```

Daha sonra views klasörü oluşturalım.



Sonrasında server.js dosyamıza bu paketi kullanması ve oluşturulan .hbs uzantılı dosyaları render etmesi için gerekli ayarlamaların yapılması gerekmektedir.

```
about.hbs  server.js  app.js  index.html  index.hbs  styles.css
WeatherApp > web-server > server.js > ...
1  const express = require("express");
2  const app = express();
3  const path = require("path");
4
5  const publicDirectory = path.join(__dirname, "./src/public");
6  const pagesDirectory = path.join(publicDirectory, "./pages");
7  const viewsDirectory = path.join(publicDirectory, "./views");
8
9  app.use(express.static(publicDirectory));
10
11 app.set("view engine", "hbs");
12 app.set("views", path.join(__dirname, "./src/public/views"));
13
14 app.get("/", (req, res) => {
15   res.render("index");
16 });
17
18 app.get("/about", (req, res) => {
19   res.render("about", {
20     title: "Hava Durumu Uygulaması",
21     developerName: "Berkan",
22   });
23 });
```

**app.set("view engine", "hbs"):** Bu kod, Express uygulamasında kullanılacak olan şablon motorunu belirtir. "hbs" değeri, Handlebars'ın kullanılacağını ifade eder.

**app.set("views", path.join(\_\_dirname, "./src/public/views")):** Bu kod, şablon dosyalarının bulunduğu dizini belirtir. Handlebars şablon dosyaları, genellikle "views" adlı bir klasörde saklanır.

**res.render("index")** ifadesi, "index" adlı Handlebars şablonunu kullanarak bir HTML sayfası oluşturur ve istemciye gönderir.

18. satırdaki kod, "/about" isteği geldiğinde çalışacak olan işlevi tanımlar. "about" adlı Handlebars şablonunu kullanarak bir HTML sayfası oluşturur. Ayrıca, bu sayfaya "title" ve "developerName" adında ekstra veriler gönderilir ve bu veriler şablon içinde `{{}}` ifadeleri arasında kullanılabilir.

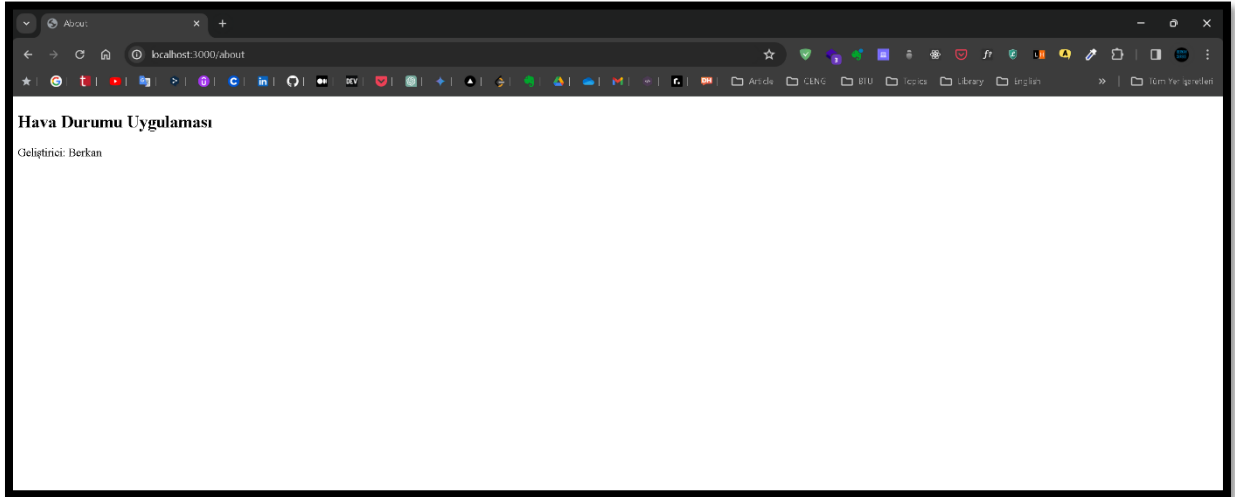
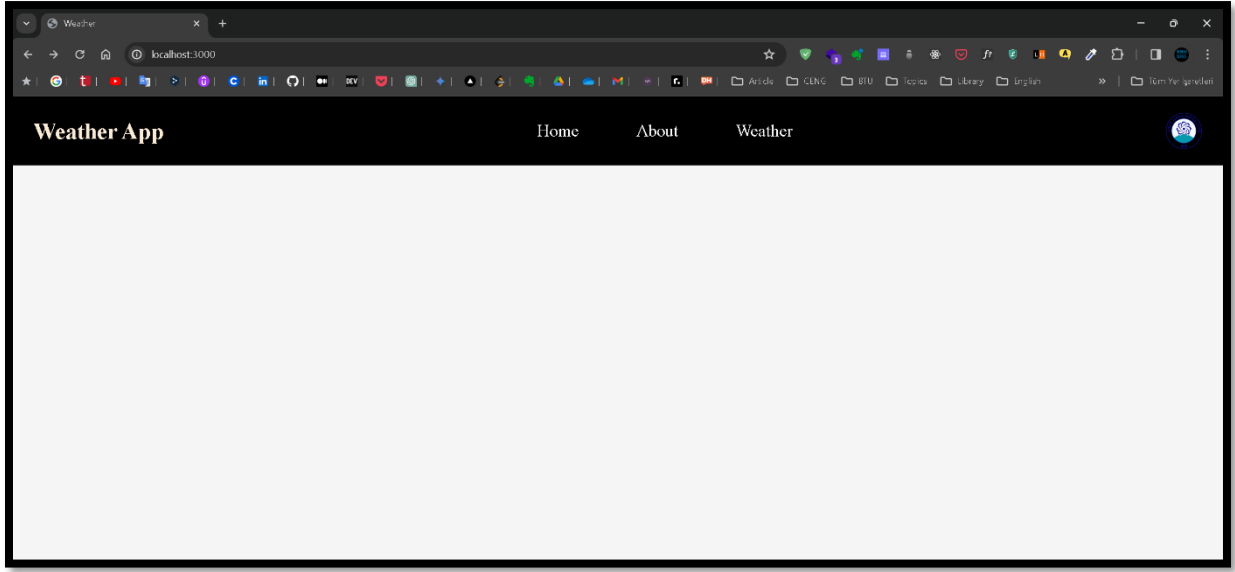
index.hbs dosyamız index.html dosyamızla şimdilik aynı içeriğe sahip olacaktır.

```
WeatherApp > web-server > src > public > views > index.hbs > html
1 <html lang="en">
2   <head>
3     <meta charset="UTF-8" />
4     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
5     <title>Weather</title>
6     <link rel="stylesheet" href="../css/styles.css" />
7   </head>
8   <body>
9     <div class="navbar-container">
10      <div>
11        <h1 class="app-name">Weather App</h1>
12      </div>
13      <div class="link-container">
14        <a href="/">Home</a>
15        <a href="/about">About</a>
16        <a href="/weather">Weather</a>
17      </div>
18      <div>
19        
20      </div>
21    </div>
22
23    <script src="../js/app.js"></script>
24  </body>
25 </html>
26
```

about.hbs adlı dosyamız aşağıdaki gibidir. Görüldüğü üzere server.js dosyamızdan gönderdiğimiz verileri aynı isimle {{}} ifadesi içerisine yazıp kullandık.

```
WeatherApp > web-server > src > public > views > about.hbs > html > body > p
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>About</title>
7 </head>
8 <body>
9   <h2>{{ title }}</h2>
10  <p>Geliştirici: {{developerName}}</p>
11 </body>
12 </html>
```

Sayfa çıktıları aşağıdaki görseldekiler gibi olacaktır.





## 5.Kaynakça

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

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

[6.Hafta Raporu - Berkan Serbes](#)