

# ***BURSA TEKNİK ÜNİVERSİTESİ***

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



**BLM0470 NodeJS İle Web Programlama**

**2023-2024 Bahar Dönemi**

## **8.Hafta Raporu**

**Berkan SERBES – 22360859353**

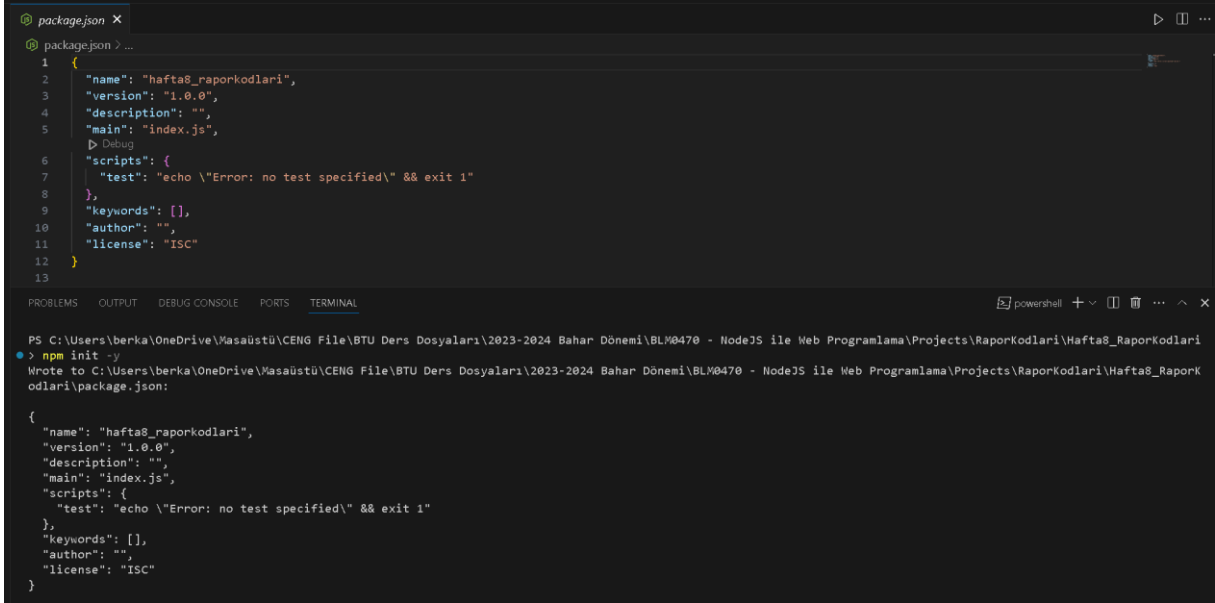
## İçindekiler

1. Dinamik Sayfalarla Hava Durumu Uygulaması Geliştirme.....	3
1.1 Uygulamanın Initialize Edilmesi ve Package.json Dosyasının Oluşturulması .....	3
1.2 Klasör yapılarının oluşturulması .....	4
1.3 Partials Sayfaları .....	6
1.4 Custom Error Sayfası .....	12
2. KAYNAKÇA .....	15

# 1. Dinamik Sayfalarla Hava Durumu Uygulaması Geliştirme

## 1.1 Uygulamanın Initialize Edilmesi ve Package.json Dosyasının Oluşturulması

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

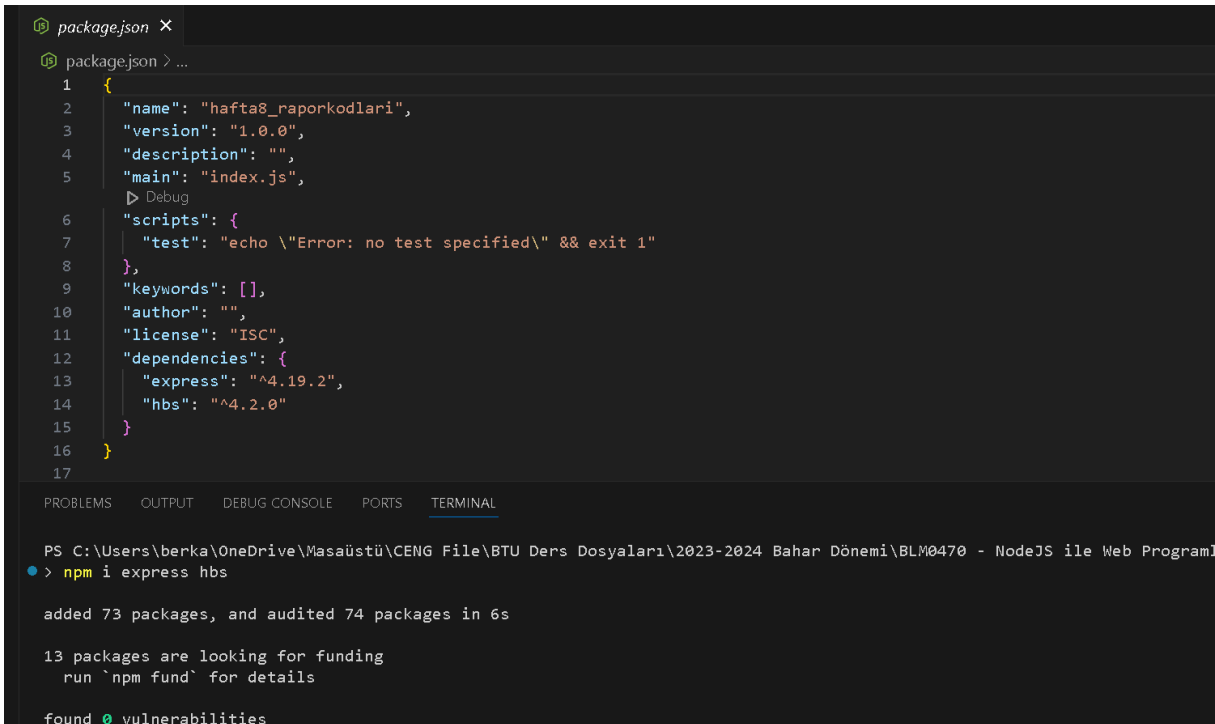


The screenshot shows the VS Code editor with a file named `package.json` open. The file contains the following JSON structure:

```
{
  "name": "hafta8_raporkodlari",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Below the editor, the terminal window shows the command `npm init -y` being executed. The output indicates that the file was written to the correct path and shows the same JSON structure as the editor.

Şimdi de uygulama boyunca kullanacağımız paketleri **npm install** komutunun yardımıyla uygulamamıza dahil edelim. Uygulamamızda kullanacağımız paketler **express** ve **hbs** paketleridir.



The screenshot shows the VS Code editor with the `package.json` file updated to include dependencies:

```
{
  "name": "hafta8_raporkodlari",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.19.2",
    "hbs": "^4.2.0"
  }
}
```

The terminal window shows the command `npm i express hbs` being executed. The output indicates that 73 packages were added and 74 packages were audited in 6 seconds. It also shows that 13 packages are looking for funding and that no vulnerabilities were found.

## 1.2 Klasör yapılarının oluşturulması

Sonrasında projemizin bulunduğu dizine **src** adında bir klasör oluşturup içerisine **index.js** adında bir dosya oluşturalım. Bu dosya projemizin başlatılacağı ana dosya olacaktır.

```
JS index.js ×
src > JS index.js > ...
1  const express = require("express");
2  const app = express();
3
4  const hbs = require("hbs");
5
6  app.set("view engine", "hbs");
7
8  app.listen(3000, () => {
9    console.log("Server is running on port 3000");
10 });
11
```

Bu kod, Express.js ve Handlebars (hbs) modülünü kullanarak bir web sunucusu oluşturur. İlk olarak, express modülü çağrılır ve bir uygulama oluşturulur. Ardından, hbs modülü çağrılarak Handlebars şablon motoru eklenir. **app.set("view engine", "hbs")** ifadesi, Express uygulamasında Handlebars'ın kullanılacağını belirtir. Son olarak, **app.listen(3000)** ifadesiyle sunucu belirtilen port numarasında dinlemeye başlar ve konsola "Server is running on port 3000" yazısı yazdırılır.

Şimdi de uygulamamızda kullanacağımız html dosyalarını oluşturalım. İlk olarak src klasörünün içerisine **public > views** klasörlerini oluşturalım. Views klasörünün içerisine **index.hbs** adında bir dosya oluşturalım.

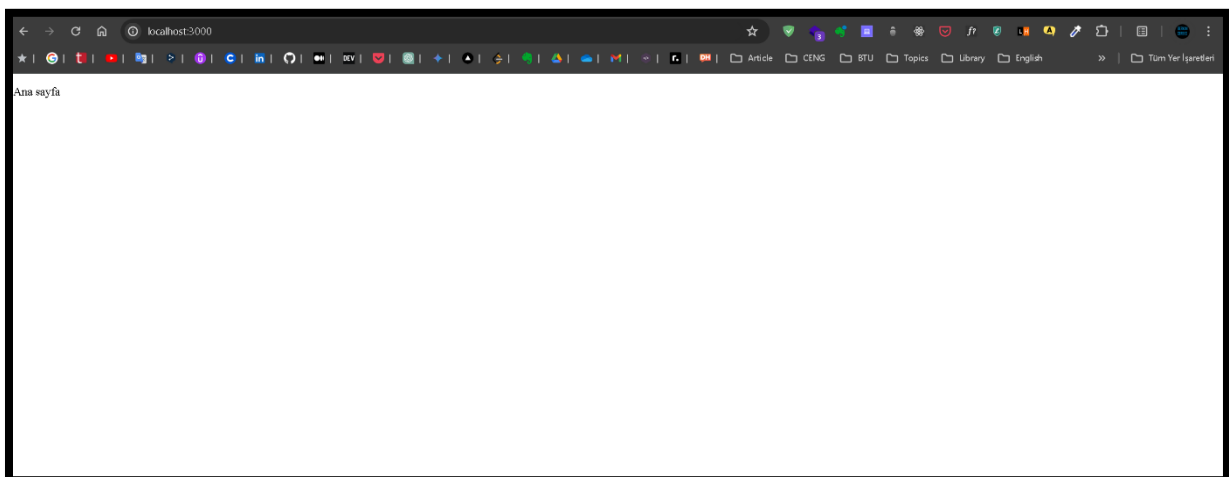
```
JS index.js  index.hbs ×
src > public > views > index.hbs > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Index</title>
7  </head>
8  <body>
9    <p>Ana sayfa</p>
10 </body>
11 </html>
```

Bir sonraki adım, oluşturduğumuz bu sayfayı kullanıcı "/" kök route'a istek attığında render etmek olacaktır.

```
JS index.js • index.hbs
src > JS index.js > ...
1  const express = require("express");
2  const app = express();
3
4  const hbs = require("hbs");
5  const path = require("path");
6
7  const publicDirectory = path.join(__dirname, "./public");
8  const viewsDirectory = path.join(__dirname, "./public/views");
9
10 app.use(express.static(publicDirectory));
11
12 app.set("view engine", "hbs");
13 app.set("views", viewsDirectory);
14
15 app.get("/", (req, res) => {
16   res.render("index");
17 });
18
19 app.listen(3000, () => {
20   console.log("Server is running on port 3000");
21 });
22
23
```

Yukarıdaki kod, public klasörünün ve views klasörünün klasör yolları belirtilerek değişkenlerde saklanmıştır. **app.use(express.static(publicDirectory))** ifadesiyle, Express'e statik dosyaların bulunduğu dizini belirterek istemcilere sunulmasını sağlar. **app.set("views", viewsDirectory)** ifadesi handlebars şablon motorunun kullanacağı görünüm dosyalarının kullanılacağı dizini belirtir. **app.get("/", (req, res) => {res.render("index")})** ifadesiyle "/" rotası belirlenir ve tarayıcıya "index.hbs" şablonu render edilir.

Tarayıcıda **localhost:3000/** url'sine istek attığımızda aşağıdaki çıktıyı almaktayız.



## 1.3 Partial Sayfaları

Partials, web uygulamalarında tekrar eden yapıların ayrı dosyalarda saklanmasını sağlayan ve bu yapıları farklı sayfalarda kullanmayı kolaylaştıran bir Handlebars özelliğidir. Bu yapılar genellikle başlık, altbilgi, yan menü gibi sayfa bileşenleridir. Örneğin, bir başlık veya altbilgiyi değiştirmek istediğimizde, sadece ilgili partial dosyasını güncellememiz yeterlidir. Bu değişiklik, tüm sayfalarda otomatik olarak yansıtılacaktır, böylece her sayfada aynı değişiklikleri tek tek yapmak zorunda kalmayız. Partials, web uygulamalarının daha modüler ve yönetilebilir olmasını sağlar. Büyük ölçekli projelerde, farklı ekipler tarafından çalışılan ve farklı parçaları geliştirilen uygulamalarda özellikle önemlidir.

Uygulamamızda 2 farklı partials bileşeni kullanacağız. Bu bileşenler **header** ve **footer** olacaktır. Header, genellikle uygulamanın üst kısmında bulunan ve genel navigasyon bağlantıları, logo ve kullanıcı giriş/çıkış işlevleri gibi bileşenleri içerir. Footer ise uygulamanın alt kısmında yer alır ve genellikle site haritası, iletişim bilgileri, sosyal medya bağlantıları gibi bilgileri içerir.

İlk olarak views klasörünün altına partials adında bir klasör açalım. Bu klasörün içerisine **header.hbs** ve **footer.hbs** adında iki dosya oluşturalım.

header.hbs dosyamızın html içeriği ve css içeriği aşağıdaki görsellerdeki gibi olacaktır.

```
src > public > views > partials > header.hbs > 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/header-style.css" />
8    </head>
9    <body>
10     <div class="navbar-container">
11       <div>
12         <h1 class="app-name">Weather App</h1>
13       </div>
14       <div class="link-container">
15         <a href="/">Home</a>
16         <a href="/about">About</a>
17         <a href="/weather">Weather</a>
18       </div>
19       <div>
20         
21       </div>
22     </div>
23   </body>
24 </html>
25
```

src > public > css > header-style.css > .link-container > a > &:hover

```
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 }
45
```

Bu dosyalarımızı oluşturduktan sonra hbs motoruna bu partials klasörümüzü tanıtmamız gerekmektedir. Index.js dosyamızı aşağıdaki gibi düzenleyelim.

```
src > js index.js > ...
1  const express = require("express");
2  const app = express();
3
4  const hbs = require("hbs");
5  const path = require("path");
6
7  const publicDirectory = path.join(__dirname, "./public");
8  const viewsDirectory = path.join(__dirname, "./public/views");
9  const partialsDirectory = path.join(__dirname, "./public/views/partials");
10
11 app.use(express.static(publicDirectory));
12
13 app.set("view engine", "hbs");
14 app.set("views", viewsDirectory);
15
16 hbs.registerPartials(partialsDirectory);
17
18 app.get("/", (req, res) => {
19   res.render("index");
20 });
21
22 app.listen(3000, () => {
23   console.log("Server is running on port 3000");
24 });
25
```

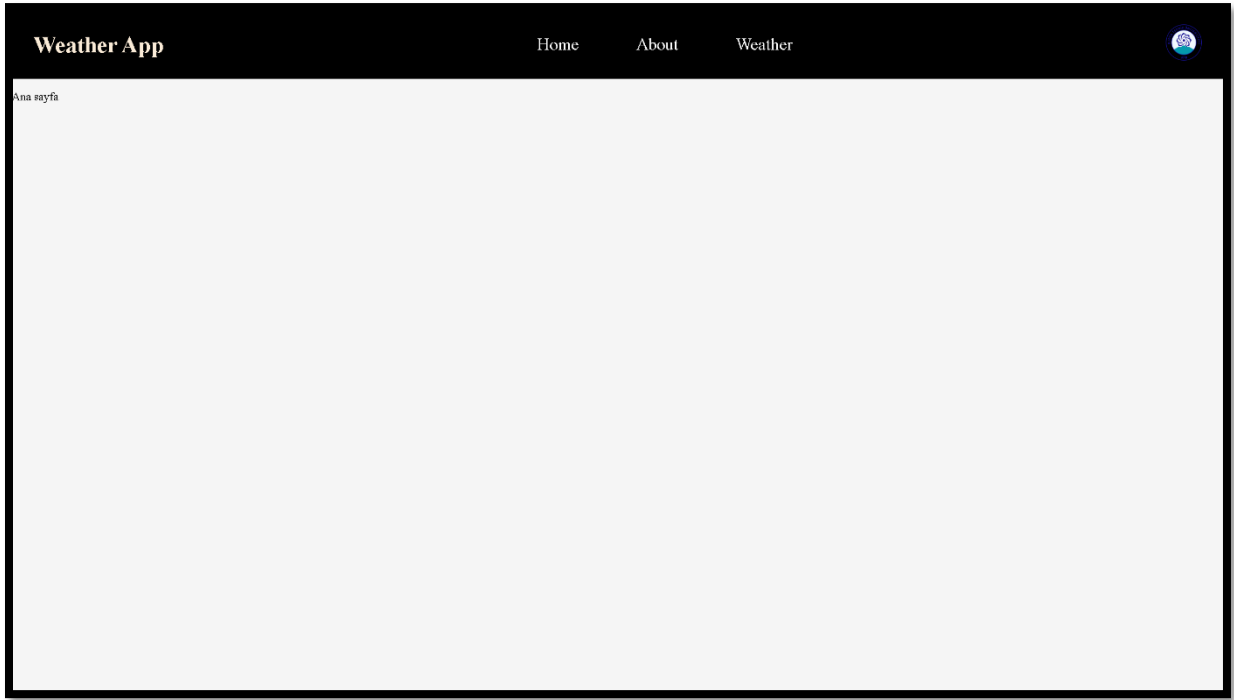
Yukarıdaki kodda, partials klasörünün dizin yolunu partialsDirectory adında bir değişkene atadık ve hbs.registerPartials(partialsDirectory) ifadesini kullanarak bu dizini partials görünümüleri için kullanılabilir hale getiriyoruz.

Şimdiki adımımız, oluşturduğumuz bu header partial view'ını index.hbs dosyamızda kullanmak olacak. Aşağıdaki şekilde kullanımı gösterilmiştir.

```
src > public > views > index.hbs > 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>Index</title>
7  </head>
8  <body>
9    {{> header}}
10   <p>Ana sayfa</p>
11 </body>
12 </html>
```



Uygulamamızı çalıştırdığımızda aşağıdaki görüntüyü elde edeceğiz.



Header partials görünümümüzü bu şekilde oluşturduktan sonra sıra footer partials görünümünü oluşturmakta. Aşağıdaki görsellerdeki gibi kodumuzu yazalım.

```
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     <link rel="stylesheet" href="../../css/footer-style.css" />
7
8   </head>
9   <body>
10    <div class="footer-container">
11      <p>{{name}} tarafından geliştirilmiştir</p>
12    </div>
13  </body>
14 </html>
15
```

```

1  body {
2      background-color: #f5f5f5;
3      margin: 0;
4      padding: 0;
5      box-sizing: border-box;
6  }
7  .footer-container {
8      position: absolute;
9      bottom: 0;
10     color: white;
11     background-color: black;
12     width: 100%;
13     display: flex;
14     justify-content: center;
15 }
16

```

Şimdide oluşturduğumuz footer partial'ını index.js dosyamızda kullanalım. Dikkat ettiyseniz footer.hbs dosyasındaki içerikte p etiketleri arasına **name** değişkeni render edilmiş. Bu değişkeni render edebilmek için bu partial görünümünü kullanan ilgili view sayfasına bu değişkeni göndermemiz gerekmektedir. Aşağıdaki şekilde yer alan 19. satırda değişken gönderme gösterilmiştir.

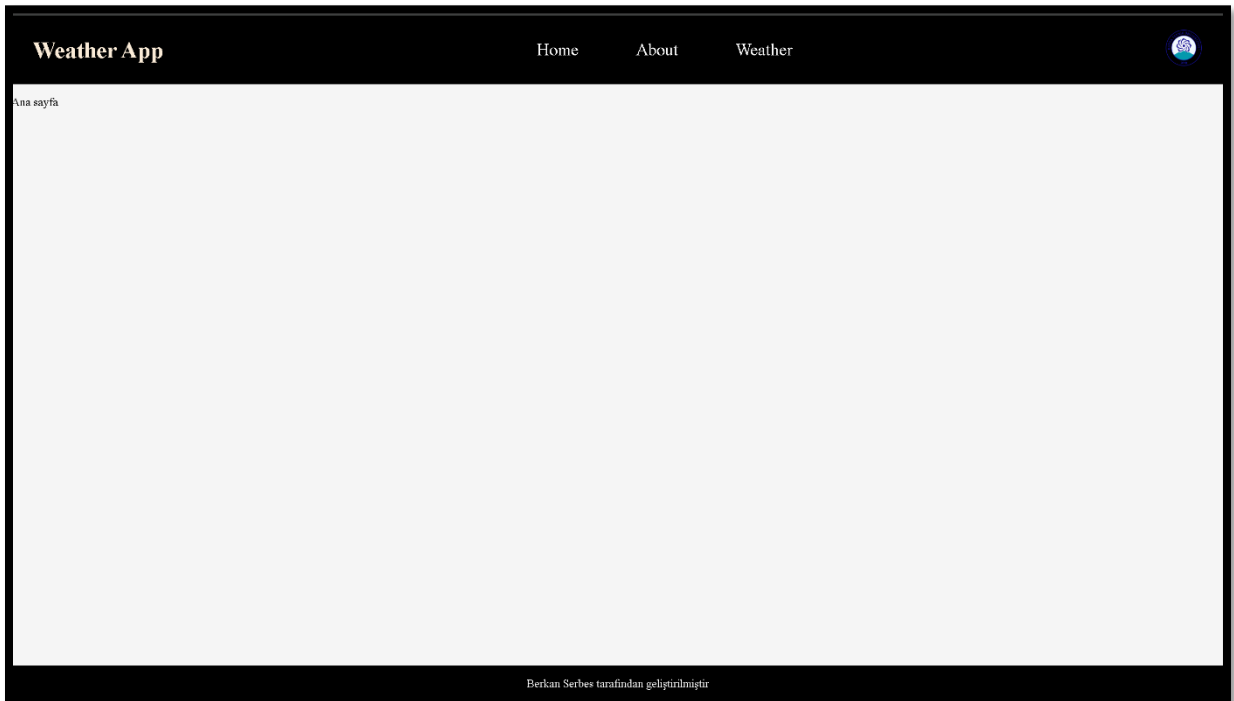
```

index.js  x  btu_llogo.png  header.hbs  header-style.css  footer.hbs  footer-sty
src > index.js > ...
1  const express = require("express");
2  const app = express();
3
4  const hbs = require("hbs");
5  const path = require("path");
6
7  const publicDirectory = path.join(__dirname, "../public");
8  const viewsDirectory = path.join(__dirname, "../public/views");
9  const partialsDirectory = path.join(__dirname, "../public/views/partials");
10
11 app.use(express.static(publicDirectory));
12
13 app.set("view engine", "hbs");
14 app.set("views", viewsDirectory);
15
16 hbs.registerPartials(partialsDirectory);
17
18 app.get("/", (req, res) => {
19     res.render("index", { name: "Berkan Serbes" });
20 });
21
22 app.listen(3000, () => {
23     console.log("Server is running on port 3000");
24 });
25

```

```
src > public > views > 🍷 index.hbs > 📦 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>Index</title>
7  </head>
8  <body>
9    {{> header}}
10   <p>Ana sayfa</p>
11   {{> footer}}
12 </body>
13 </html>
```

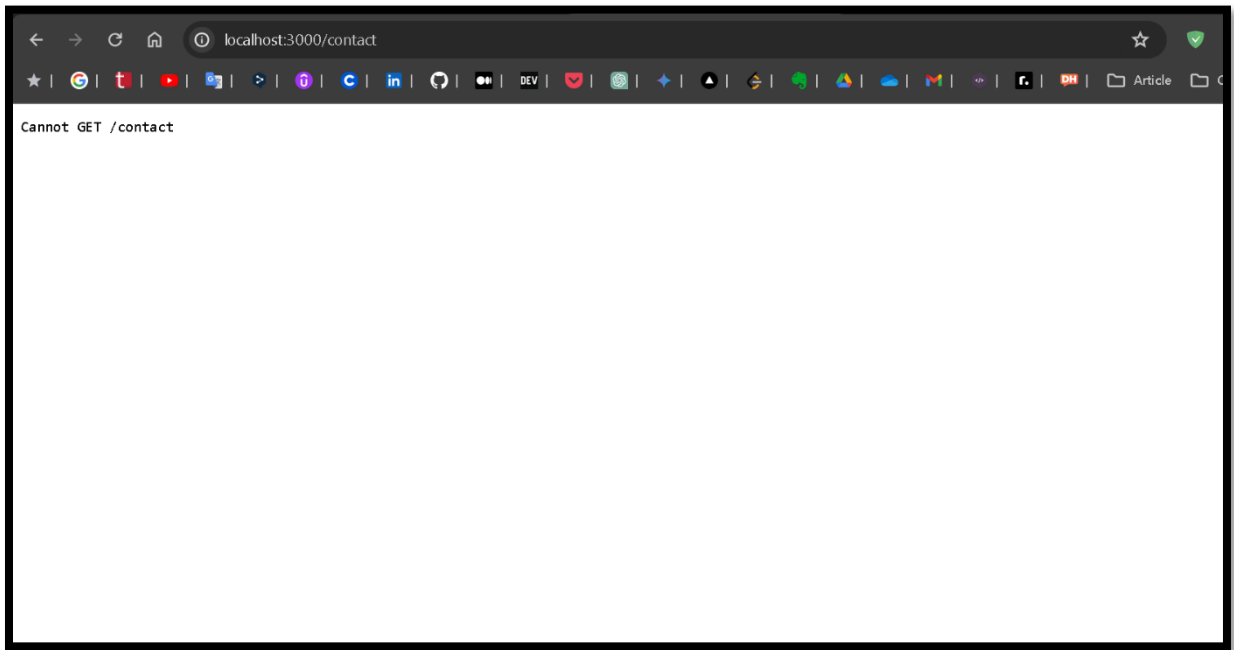
Kodlarımızı bu şekilde düzenledikten sonra index.hbs dosyamızın görüntüsü tarayıcıda aşağıdaki gibi olacaktır.



## 1.4 Custom Error Sayfası

Custom error sayfası, kullanıcıların var olmayan veya geçersiz bir URL'ye istek yaptıklarında gösterilen özel bir sayfadır. Bu sayfa genellikle "404 Not Found" hatasıyla ilişkilendirilir ve kullanıcıya "Sayfa Bulunamadı" veya benzeri bir mesaj iletilir. Bu tür bir sayfa, kullanıcıların karşılaştığı hataları daha anlaşılır hale getirmek ve kullanıcı deneyimini geliştirmek için kullanılır. Ayrıca, custom error sayfası, web uygulamasının marka kimliğine uygun olarak tasarlanabilir ve hata durumunda kullanıcıya yönlendirici veya yardımcı bilgiler sunabilir. Bu sayede, kullanıcılar hatayla karşılaştıklarında daha olumlu bir deneyim yaşarlar ve uygulama ile etkileşimlerini sürdürebilirler.

Şuan ki uygulamamızda herhangi bir custom error sayfası bulunmamaktadır. Kullanıcı tanımlanmayan bir url'e istek attığında aşağıdaki görüntü karşımıza gelecektir.



Bu aşamada ise custom error sayfası oluşturacağız. Öncelikle views klasörümüzün altına **404.hbs** adında bir dosya oluşturalım.

```
src > public > views > 404.hbs > ...
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>Error</title>
7      <link rel="stylesheet" href="../css/404-style.css" />
8    </head>
9    <body>
10     <div class="container">
11       <h1><span class="uri">{{baseUrl}}</span> <span class="arrow">=></span> 404 Not Found</h1>
12     </div>
13   </body>
14 </html>
```

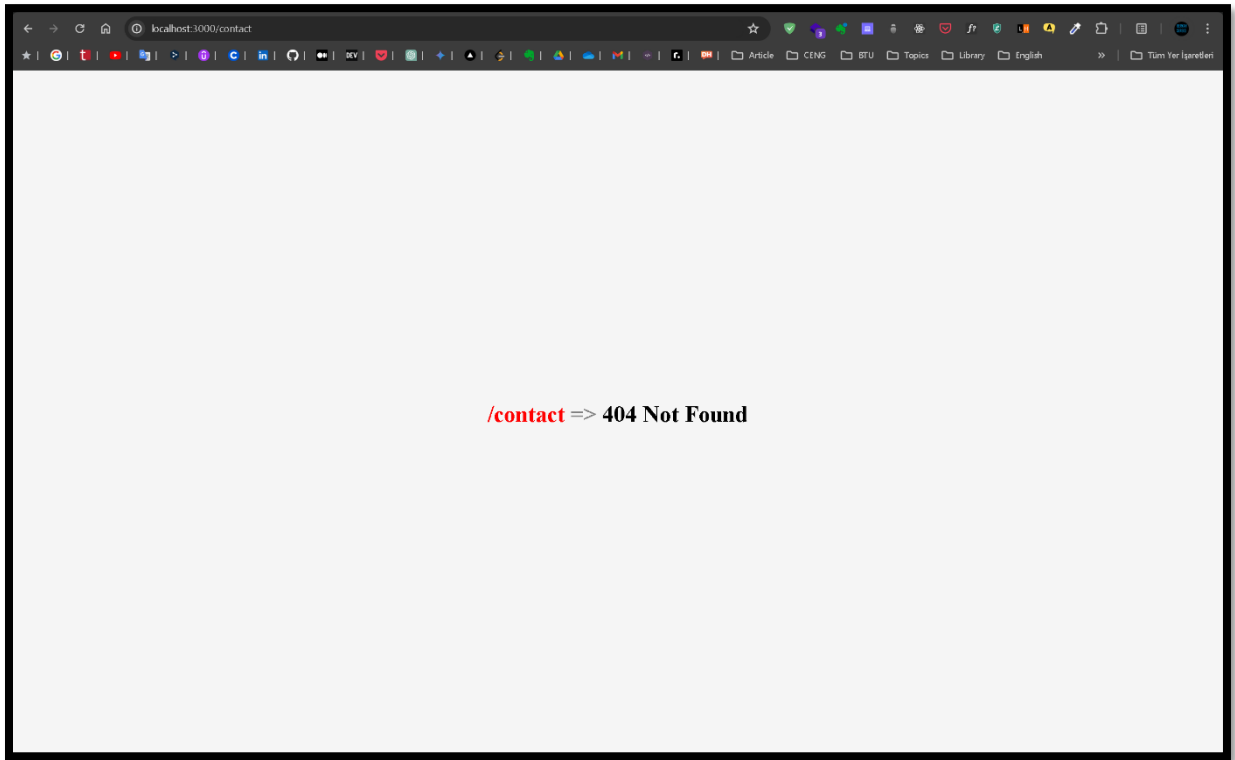
```
src > public > css > 404-style.css > ...
1  body {
2      background-color: #f5f5f5;
3      margin: 0;
4      padding: 0;
5      box-sizing: border-box;
6  }
7  .container {
8      display: flex;
9      justify-content: center;
10     align-items: center;
11     height: 100vh;
12 }
13
14 .uri {
15     color: red;
16 }
17
18 .arrow {
19     color: gray;
20 }
```

Dosyalarımızı bu şekilde oluşturduktan sonra yapmamız gereken şey bu sayfayı ilgili route'da render etmek olacak.

```
src > index.js > ...
1  const express = require("express");
2  const app = express();
3
4  const hbs = require("hbs");
5  const path = require("path");
6
7  const publicDirectory = path.join(__dirname, "../public");
8  const viewsDirectory = path.join(__dirname, "../public/views");
9  const partialsDirectory = path.join(__dirname, "../public/views/partials");
10
11 app.use(express.static(publicDirectory));
12
13 app.set("view engine", "hbs");
14 app.set("views", viewsDirectory);
15
16 hbs.registerPartials(partialsDirectory);
17
18 app.get("/", (req, res) => {
19     res.render("index", { name: "Berkan Serbes" });
20 });
21
22 app.get("*", (req, res) => {
23     res.render("404", { baseUrl: req.url });
24 });
25
26 app.listen(3000, () => {
27     console.log("Server is running on port 3000");
28 });
29
```

Yukarıdaki görselde yer alan 22-24.satırlar arasındaki kod parçası, Express uygulamasında tanımlanan bir yönlendirme işlemini temsil eder. **app.get("\*", ...)** ifadesi, kullanıcıların herhangi bir URL'ye istekte bulunduğunda çalışacak olan bir yönlendirme işlemidir. Yani, bu kod, kullanıcıların tanımlanmış diğer tüm route'lara (yönlendirmelere) uymayan herhangi bir URL'ye istekte bulunduğunda devreye girer. **res.render("404", { baseUrl: req.url })** ifadesi ise, bu durumda kullanıcılara gösterilecek olan sayfayı oluşturur. **res.render()** metodu, bir Handlebars (hbs) şablonunu render etmek için kullanılır. Burada **"404"** adında bir şablon dosyası render edilir ve bu dosya, kullanıcılara gösterilecek özel bir sayfayı temsil eder. İkinci parametre olarak **{ baseUrl: req.url }** objesi verilir ve bu, 404 sayfasının içinde kullanılacak olan veri veya değişkenleri temsil eder. Bu örnekte, **baseUrl** adında bir değişken tanımlanır ve istenen URL'yi (**req.url**) içerir. Bu, 404 sayfasında kullanıcılara hangi URL'ye istekte bulunduklarını göstermek için kullanılır.

“/” dışında bir url’ye istek attığımızda aşağıdaki sayfa karşımıza çıkacaktır.



## 2. KAYNAKÇA

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