Kadir ÇAPKIN 21360859023

NodeJS ile Web programlama Raporu Hafta – 9

Weather app – 4

Başlangıçta public klasörü altında templates adlı klasörümüzü oluşturduk. Bu klasör altında partials ve views adlı iki farklı klasör oluşturduk. Bu dosyalar bizim genel şablon dosyalarımızı ve sayfalarımızı barındıracak. (header,body şablonları) Bu klasörler oluşturulduktan sonra src klasörü altındaki app.js dosyamızda bu dosyaları kullanılabilir hale getirmemiz gerekiyor bu yüzden aşağıdaki kodu yazmamız gereklidir.

```
const hbs = require('hbs');
const partialPath = path.join(__dirname,"../public/templates/partials");
hbs.registerPartials(partialPath);
```

Bu kodu yazdıktan sonra aktifleşmiş olacak.Bunu yazdıktan sonra ise normalde mevcut views klasörümüzü kopyalayıp templates klasörü altındaki views içerisine atmamız gerekiyor. Bunu yaptıktan sonra view engine kodunu şu şekilde düzenlemeliyiz.

```
const viewsPath = path.join(__dirname,'../public/templates/views');
app.set('views',viewsPath);
```

Bu kodu yazmamızdaki amacımız mevcut web sitesinde genel bir taslak oluşturup her sayfada bu mevcut taslağı kullanarak kalan kısımları özelleştirmek için kullanıyoruz. Örnek vermek gerekirse her sayfada (anasayfa,about,help) tüm bu sayfalarda en üst tarafta bir navbar bulunur. Bu navbar kodunu tekrar tekrar her sayfada yazmak yerine bir kez yazıp diğer sayfalarımızda bu bir kere yazdığımız kodu kullanıp sayfa bazında özelleştirebiliyoruz amacımız bundan ibaret.

Şu an için her bir sayfada kullanmak üzere header bilgisini şu şekilde yazalım. Yukarıda dediğim gibi bu yazılan kodlar belirli sayfalarda doğrudan istediğimiz gibi kullanılabilir hale gelicek.

```
<header>
    <h1>{{title}}</h1>

    <div>
        <a href="/">Weather</a>
        <a href="/about">About</a>
        <a href="/help">help</a>
    </div>
</header>
```

Bu yazdığımız kodda h1 etiketi içerisinde bulunan title dinamik olarak src altındaki index.js içerisinden gelecek. Şimdi bu header dosyamızı başka bir view dosyası olan index.js içerisinde kullanalım.

```
    <h1>{{>header}}</h1>
    <div>
        <img src="./img/resim.png" alt="">
        <p>{{name}} Tarafindan gelistirilmistir.</p>
    </div>
```

{{<partialFileName}} şeklinde mevcut partial dosyamızı index.hbs dosyamıza ekledik. Bu kodun çıktısı şöyledir.

Görüldüğü gibi başarılı bir şekilde eklendi ve index.js içerisinde dinamik olarak gönderilen name parametresinin de header.hbs içerisindeki {{name}} de kullanabildik. Çünkü bu header sayfası index.hbs içerisinde kullandık. Şimdi kalan sayfalarımızda da oluşturmuş olduğumuz header.hbs'i kullanalım.

Help

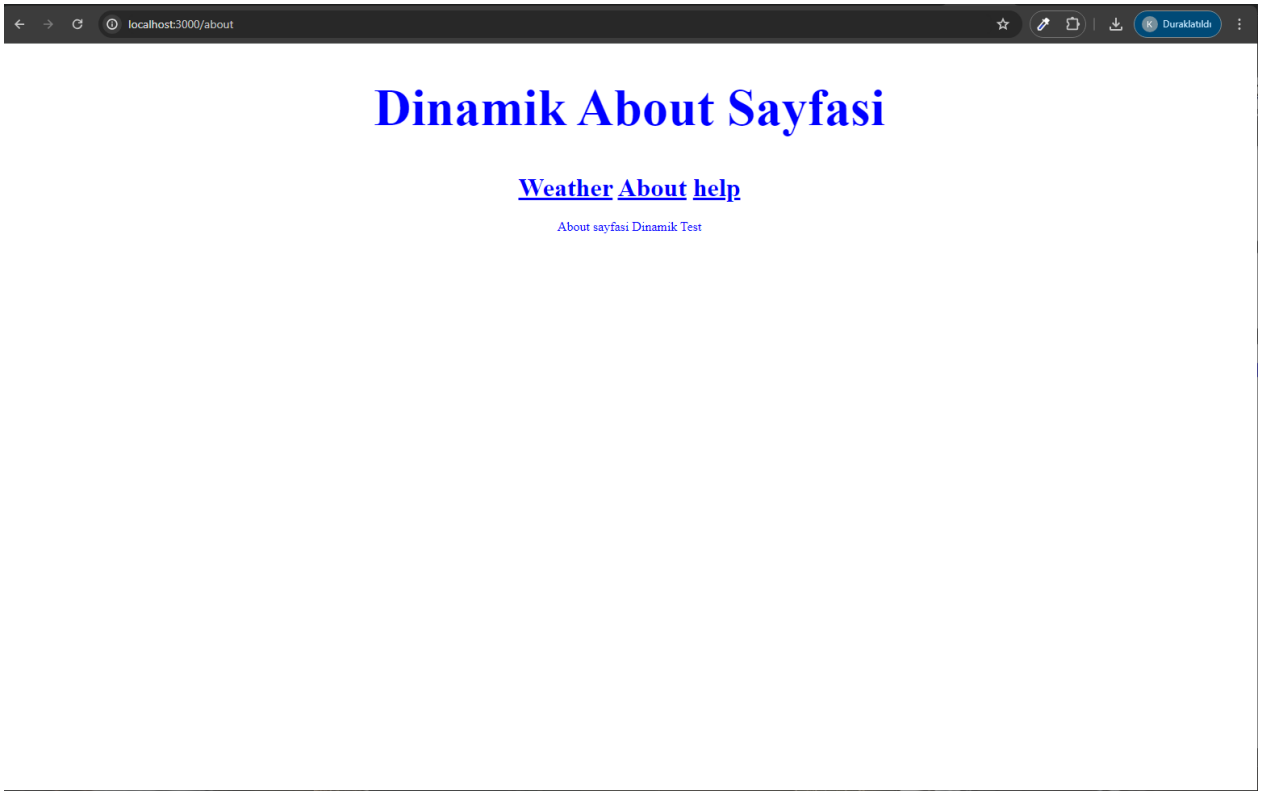
```
    {{!-- help sayfası --}}
    <h1>{{>header}}</h1>
    <p>{{name}}</p>
    <p>{{helpText}}</p>
```

About

```
    {{!-- about sayfası --}}
    <h1>{{>header}}</h1>
    <p>{{name}}</p>
```

Sayfaların çıktılarına bakacak olursak.


localhost:3000/help
Duraklatıldı
Yardim Sayfasi Dinamik Baslik
Weather About help
Yardim sayfasi dinamik icerik
HelpText Test Dinamik Yazısıdır.


localhost:3000/about
Duraklatıldı
Dinamik About Sayfasi
Weather About help
About sayfasi Dinamik Test

Şimdi alttaki yazı için de bir footer.hbs adında bir partial dosyası oluşturalım. Bu dosyada en altta gördüğümüz p etiketi içerisindeki bilgi yazısını içersin tek tek yazmak yerine tek seferde yazmış olalım.

```
<footer>
    <p>{{name}} tarafindan gelistirilmistir.</p>
</footer>
```

Şeklinde kodumuzu yazdık şimdi bu kodu az önceki gibi diğer sayfalar içerisinde kullanalım.

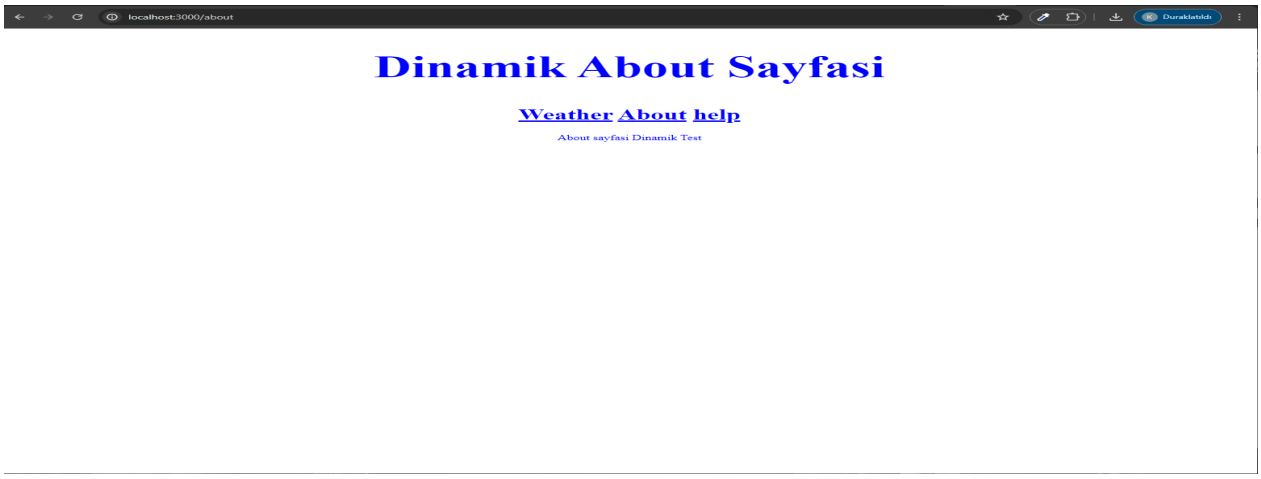
```
    {{!-- index.hbs sayfası --}}
    <h1>{{>header}}</h1>
    <div>
        <img src="./img/resim.png" alt="">
        {{>footer}}
    </div>

    <img src="./img/rapor6.png" alt="">
```

```
    {{!-- help sayfası --}}
    <h1>{{>header}}</h1>
    {{>footer}}
    <p>{{helpText}}</p>
```

```
    {{!-- about sayfası --}}
    <h1>{{>header}}</h1>
    {{>footer}}
```

Çalıştırdığımızda karşımıza yine aynı sayfa gelecektir.



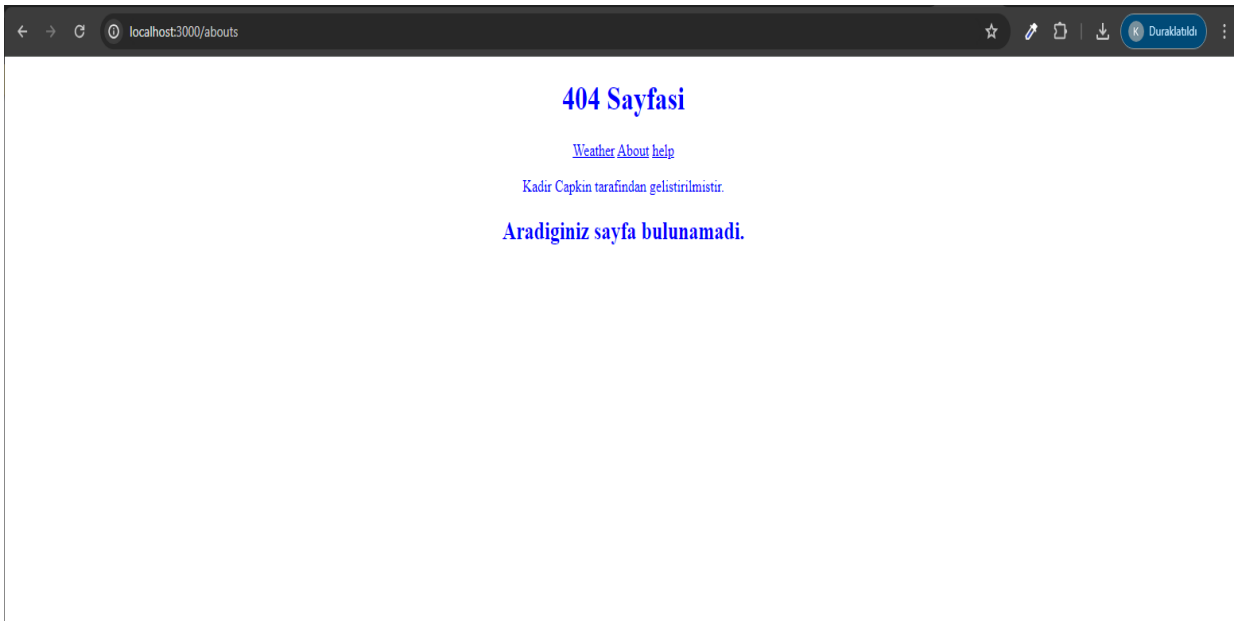
Şu an yazdığımız routerlarda istenmeyen bir url yazıldığında sayfa çalışmıyor. Sayfa çalışmaması yerine kendimiz bir hata sayfası oluşturup bunu vermemiz web sitemiz açısından daha mantıklıdır. Şimdi bunu yazalım.

```
app.get("*",(req,res)=>{
    res.render('404.hbs',{
        title:"404 Sayfasi",
        name:"Kadir Capkin",
        errorMessage:"Aradiginiz sayfa bulunamadi."
    });
})
```

Bu kodu yazdığımızda url üzerinde her hangi rastgele bir şey yazdığımızda gelebilicek muhtemel sayfadır. Örneğin yanlışlıkla localhost:3000/about yerine localhost:3000/abouts yazarsak doğrudan burada renderladığımız 404.hbs dosyası karşımıza çıkar. Şimdi bu 404.hbs dosyasını ve içeriğini views klasörü altında oluşturalım.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="./css/style.css">
    <script src="./js/app.js"></script>
</head>
<body>
    {{!-- 404.hbs dosyası --}}
    {{>header}}
    {{>footer}}
    <h2>{{errorMessage}}</h2>
</body>
</html>
```

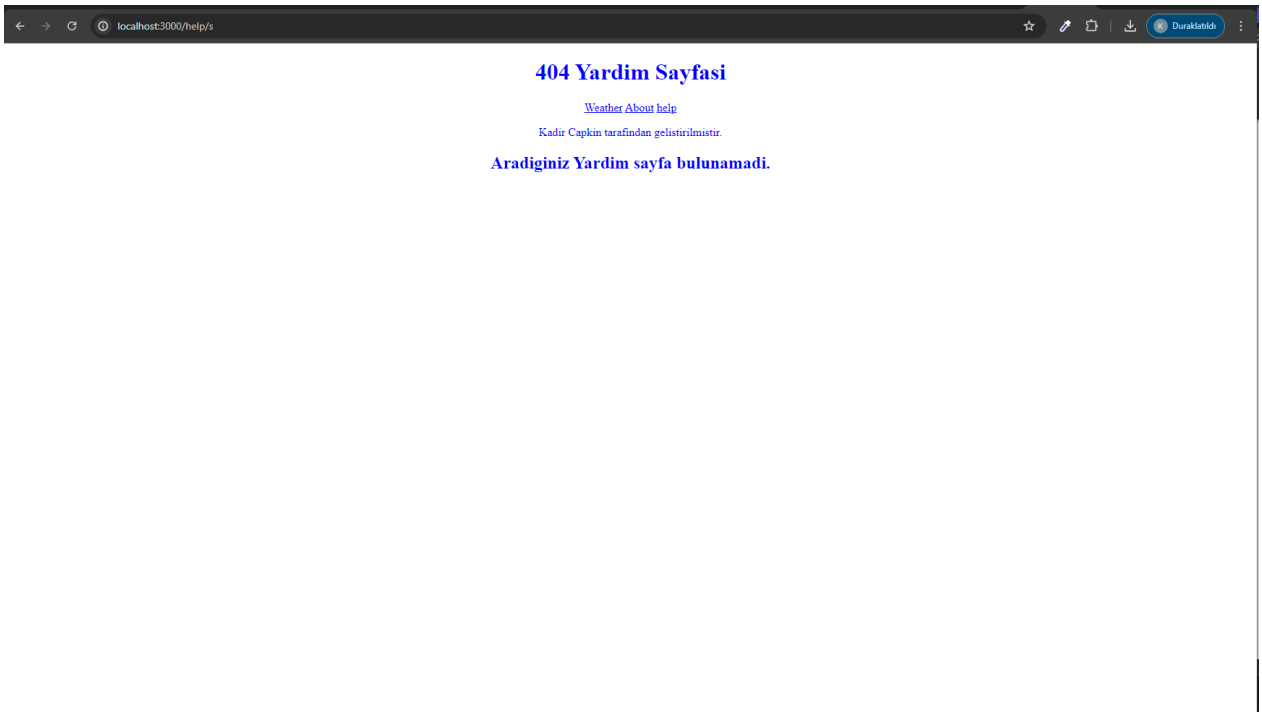
Şeklinde oluşturup gerekli script ve css dosyalarımızı head içerisinde ekledikten sonra uygulamayı çalıştırıp localhost:3000/abouts yazarak denersek karşımıza şu sayfa çıkar.



Şimdi web sitelerinde belirli url'ler üzerinden belirli başka url'lere geçişler yapabiliyoruz. Örneğin localhost:3000/help/helpText -> bu url'e gitmek istediğimiz help üzerindeki bir url'den yazmış olduk. Eğer böyle bir url'i yazarken hata yapmamıza karşın bu url'inde kendine özel bir hata sayfasını hazırlamak istersek şu şekilde yazarız.

```
//Help error router
app.get("/help/*",(req,res)=>{
    res.render('404.hbs',{
        title:"404 Yardim Sayfasi",
        name:"Kadir Capkin",
        errorMessage:"Aradiginiz Yardim sayfa bulunamadi."
    });
})
```

Şu an mevcut 404.hbs dosyamızı kullandık fakat dinamik olarak farklı bir parametre gönderdik. Çıktımız şu şekildedir.



Görüldüğü gibi localhost:3000/help/random bir şey yazarsak bu hata sayfasını almış oluruz.

Şimdi mevcut kodlarımıza biraz css eklemek adına index.hbs dosyamızı şu şekilde:

```
<body>
  <div class ="main-content">
    <h1>{{>header}}</h1>
  </div>
  {{>footer}}
</body>
```

Help.hbs dosyamızı şu şekilde:

```
<body>
  <div class ="main-content">
  {{>header}}
  <p>{{helpText}}</p>
  </div>

  {{>footer}}
</body>
```

About.hbs dosyamızı şu şekilde:

```
<body>

  <div class ="main-content">
    <h1>{{>header}}</h1>
    <img class="portrait" src="/img/resim.png" />
```

```
    </div>

   {{>footer}}
  </body>
```

404.hbs dosyamızı ise şu şekilde düzenleyelim:

```
  <body>
    <div class="main-content">
      <h1>{{>header}}</h1>
      <p>{{errorMessage}}</p>
    </div>
    {{>footer}}
  </body>
```

Bunu yapmamızdaki amaç tüm çalışmalarımızın main-content adı altında bir blokta olması ve bloğun css özelliklerini daha kolay bir şekilde ayarlayabilmektir. Şimdi css kodlarımızı inceleyecek olursak.

```
*{
    text-align: center;
    color: blue;
}

img {
    width: 500px;
    margin-top: 10px;
}
```

Normalde şu şekildeydi tüm elementler mavi renkli ve ortalanmış , resimler ise sabit 500px genişliğinde ve üstteki element ile mesafesi 10px olacak şekildeydi ve bunlar test amaçlıydı şimdi bunları silip sırasıyla css kodlarını ekleyelim.

```
body {
    color: #1ecb49;
    font-family: Arial;
    max-width: 650px;
    margin: 0 auto;
    padding: 0 16px;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
  }
```

Tüm body içerisindeki etiketleri etkileyecek şekilde bu kodları yazdık. Color:rengi, font-family:yazı tipimizi,max-width:maxiumum genişliği,margin:body etiketinin diğer elementlerinin uzaklığını, padding:iç dolgu olarak tanımlanabilir, iki değer verilmiş üst ve

alttan 0 sağ ve soldan 16px , display:flex bu ise sayfanın responsive özellik taşımasını sağlayan flexbox içerir mevcut sayfa boyutuna göre sayfa içeriği düzenlenir.

```
.main-content {
  flex-grow: 1;
}
```

Main -content class'ini kullandığımız div içerisinde etki eder sayfadaki konum için kullanılır.

```
footer {
  color: coral;
  border-top: 1px solid #1e46cb;
  margin-top: 16px;
  padding: 16px 0;
}
```

Olusturdugumuz footer.hbs içerisindeki <footer></footer> etiketine etki eder. Coral isimli bir renk, ardından sadece etiketin üst kısmında olucak bir çerçeve, üzerindeki etiketten 16 px olacak uzaklıkta ve iç dolgusu üstten ve alttan 16px iken sağ ve soldan 0 px'dir.

```
header {
  margin-top: 16px;
  margin-bottom: 48px;
}
```

Aynı şekilde header.hbs içerisinde oluşturmuş olduğumuz <header></header> etiketine etki eder. Üstündeki elementten 16px , altındaki elementten ise 48px mesafe bırakır.

```
h1 {
  font-size: 64px;
  margin-bottom: 16px;
}
```

Mevcut h1 etiketinin içeriğini düzenler ve default font size yerine 64px'lik bir yazı boyutu oluşturur ve altına gelen elemente 16px mesafe bırakır.

```
header a {
  color: blueviolet;
  font-size: 16px;
  margin-right: 16px;
  text-decoration: none;
}
```

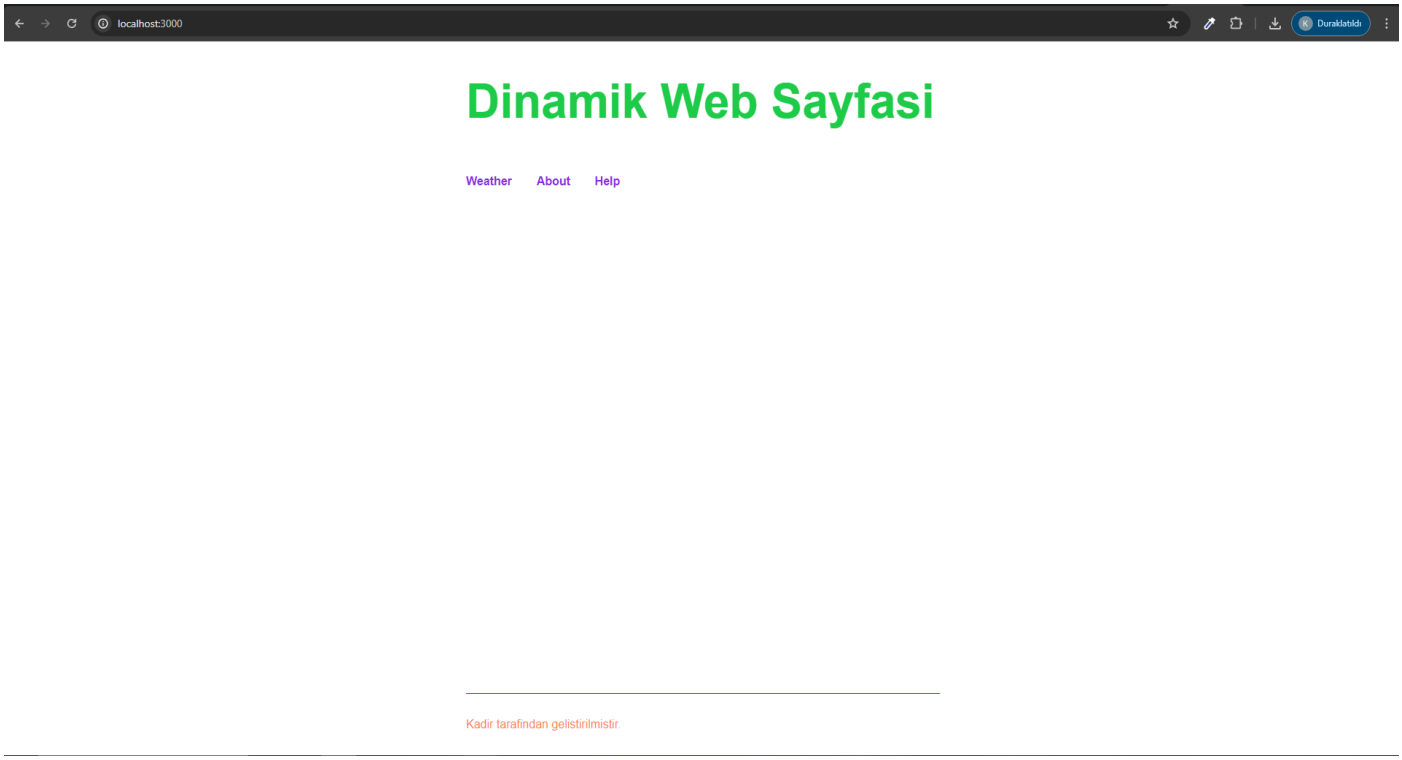
Sadece <header></header> etiketi içerisindeki a etiketlerini etkiler. Blueviolet adli renkte, 16px büyüklüğünde, sağındaki elementten 16px uzaklıkta, ve mevcut altı çizili a etiketinin altını çizmemek için tex-decoration:none kullanıldı.

Ardından web tarayıcısının sekme kısmında icon eklemek için img klasörümüz altına bir icon resmi koyduk. Bu icon resmini kullanmak içinse header kısmı içerisinde şu kodu kullandık.

```
<link rel="icon" href="/img/indir.png">
```
Bu şekilde mevcut icon görünür hale geldi.

Mevcut CSS kodlarımı ekledikten sonra hazırladığımız sayfaların görüntüsü şu şekildedir.

Anasayfa:



About Sayfası:

Help Sayfası:

Error Sayfası:

404 Sayfasi

Weather About Help

Aradiginiz sayfa bulunamadi.

Kadir Capkin tarafindan gelistirilmistir.