Node Js Ödevi

8th Week

Farhan Ahmad

20360859096

Geçen hafta **Object Property Shorthand, Destructuring ve Expressjs** gibi konuları incelemiştik. Bu **hafta ise Express.js ile Statik Web Sayfalarının Servis Edilmesi ve Dinamik İçeriklerin Hazırlanması ve Handlebars Partialları Nodemon** gibi konuları ele alacağız.

Statik Sayfaların Servis Edilmesi:

Öncelikle, proje klasöründe bulunan "public" adlı klasöre erişin ve içerisinde yeni bir klasör oluşturalım. Bu yeni klasörü "css" olarak adlandıralım. Ardından, oluşturduğumuz bu "css" klasörü içerisinde "styles.css" adında yeni bir dosya açalım.

Bu dosyanın içine, sayfanızın stilini belirlemek için gerekli CSS kodlarını yazalım. Örneğin, başlık etiketlerinin rengini gri yapmak için aşağıdaki gibi bir kod kullanabiliriz:

```
h1 {
    color: grey;
  }
```

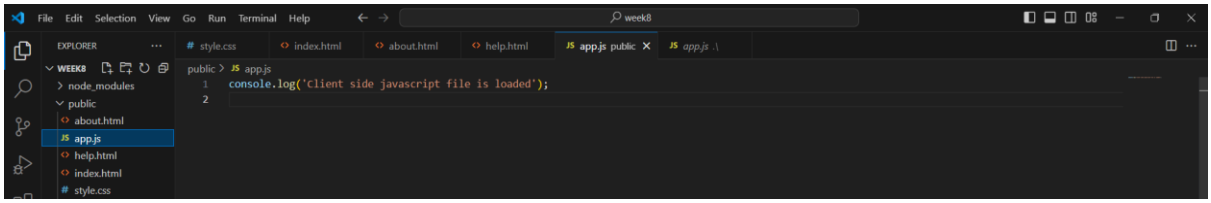
Bu adımları tamamladıktan sonra, index.html dosyasının head bölümüne aşağıdaki gibi bir satır ekleyerek "styles.css" dosyasını bağlayalım:

```
    <link rel="stylesheet" href="style.css">
```

Bu sayede, tarayıcı "styles.css" dosyasını yükleyerek başlık etiketlerinin rengini gri olarak görüntüleyecektir. Bu işlemi "help" ve "about" sayfaları için de aynı şekilde uygulayabilirsiniz.

Index sayfasını yeniledikten sonra, başlık etiketlerinin renginin gri olduğunu göreceksiniz. Aynı işlemi "help" ve "about" HTML dosyaları için de yapmanız gerekmektedir.

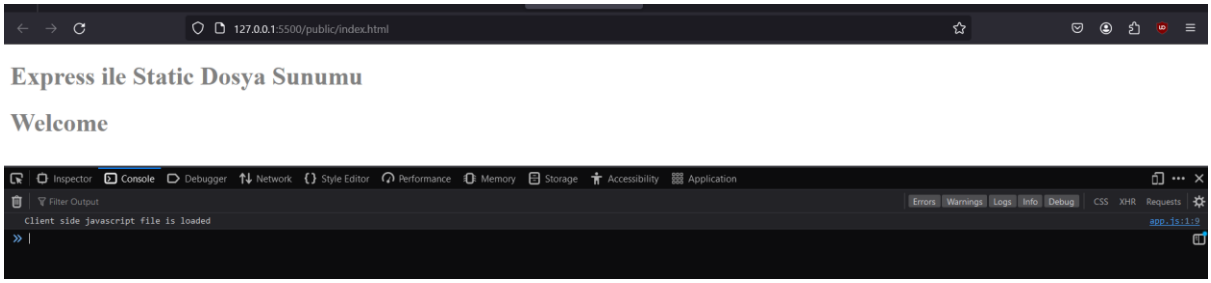
Bunun için, "public" klasörü altında yeni bir klasör oluşturun ve adını "js" olarak belirleyin. Ardından, "js" klasörü içinde "app.js" adında yeni bir dosya oluşturun. Bu dosyanın içine, tarayıcı tarafında çalışacak olan JavaScript kodlarını yazın. Bu kodlar genellikle sayfanın davranışlarını kontrol eder veya kullanıcı etkileşimlerini yönetir.



JavaScript dosyası kaydedildikten ve index sayfası yenilendikten sonra, tarayıcıda herhangi bir çıktı gözükmemesi normaldir çünkü bu dosya sadece tarayıcı konsolunda bir çıktı üretmektedir. Ancak, tarayıcı üzerinde bu çıktıyı görebilmek için, index.html dosyasının head bölümüne aşağıdaki gibi bir kod ekleyerek "app.js" dosyasını bağlayabiliriz:

```
    <link rel="stylesheet" href="style.css">
```

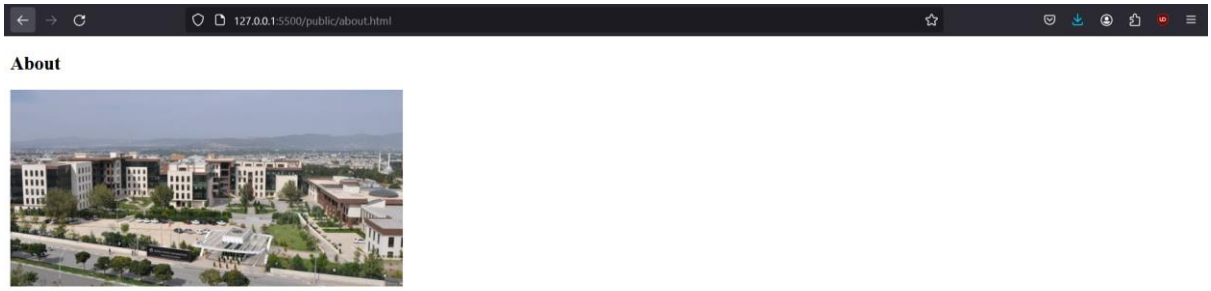
Bu işlemi gerçekleştirdikten sonra, tarayıcı konsolunda "Client side javascript file is loaded" gibi bir çıktı görebilmeliyiz. Bu adımları uyguladıktan sonra, JavaScript dosyasının başarıyla yüklendiğini ve çalıştığını görmeliyiz.



Şimdi ise, projenizin "public" klasörü altına "img" adında yeni bir klasör oluşturun. Bu klasör, projenizde kullanacağınız resim dosyalarını içerecektir. Daha sonra, "about.html" dosyasını açın ve içine aşağıdaki gibi bir kod ekleyin:

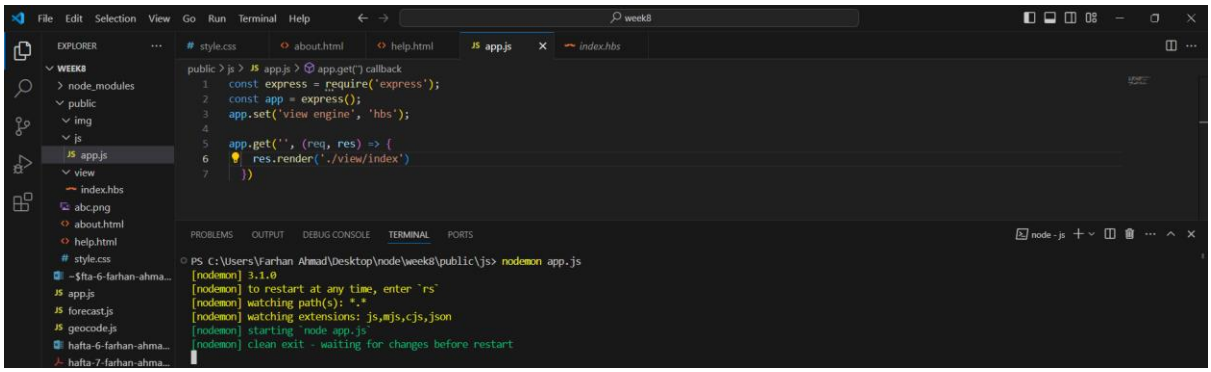
```
<body>

        <h1>About</h1>
        <img src="abc.png">


</body>
```

Bu kod, "about.html" sayfasında "img" klasörü altındaki "abc.png" adlı resmi gösterecektir. Resim dosyasının adını ve uzantısını projenize uygun şekilde değiştirebilirsiniz. Bu adımları tamamladıktan sonra, "about.html" sayfasını kaydedin ve tarayıcıda yenileyerek resmin görünüp görünmediğini kontrol edin.



Statik web sayfaları değişmez, bu nedenle dinamik içerikler oluşturmak için bir template motoru kullanılması gerekmektedir. Template motoru olmadan, her HTML dosyasına başlık gibi tekrar eden kısımları eklemek gerekecektir. Bu projede handlebars ve hbs gibi express.js için görünüm motoru olan npm modülleri kullanılacaktır. Son olarak, Nodemon kapatılarak ve gerekli modüller yüklenerek bu adımlar tamamlanmalıdır.

npm install -g nodemon



Yeni dosyada, index.html dosyasının içeriğini kopyalayın ve başlığı değiştirerek aşağıdaki gibi güncelleyin:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="styles.css">
<script src="app.js"></script>
</head>
<body>

<h1>Weather</h1>
</body>
</html>
```

Artık index.html dosyasına ihtiyaç yoktur ve silebiliriz. Sayfaya erişim sağlayabilmek için app.js dosyasına aşağıdaki kodu ekleyin:

```
app.get('', (req, res) => {
  res.render('index')
})
```

nodemon src/app.js

Weather

Handlebars Partialları ve Nodemon Kullanımı:

Tarayıcıda kök sayfayı ziyaret ederek "Weather" başlığını görebilirsiniz. Devam edelim!

Şimdi içeriği dinamik hale getirelim. Hbs dosyasında şu kodu kullanın:

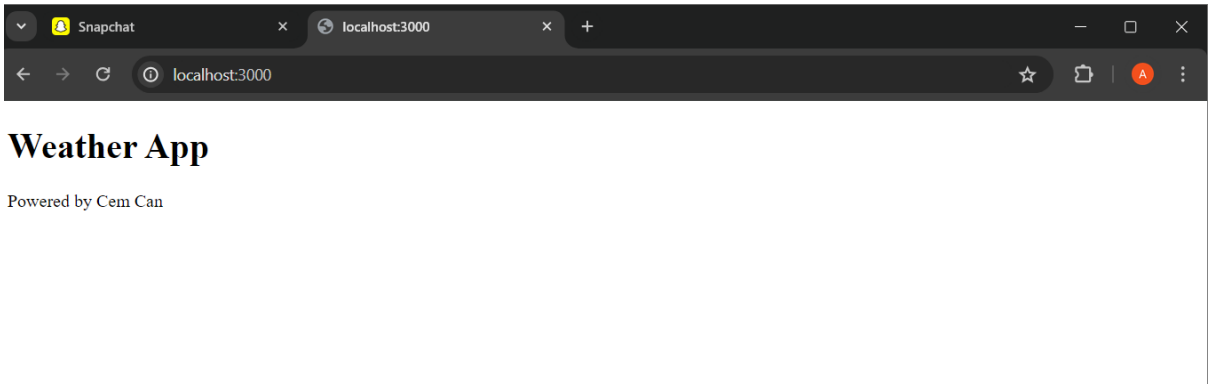
```
</html>
<body>
<h1>{{title}}</h1>
</body
```

app.js dosyasında ise aşağıdaki gibi güncelleyin:

```
app.get('', (req, res) => {
  res.render('index', {
    title: 'Weather App',
    name: 'Cem Can'
  })
})
```

Dosyayı kaydedin ve sayfayı yenileyin. Ardından, hbs dosyasını aşağıdaki gibi güncelleyin:

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



Dosyayı kaydedin ve sayfayı yenileyin. views klasörünün altında about.hbs dosyasını oluşturun, html içeriğini kopyalayıp yapıştırın ve statik about.html dosyasını silin.

app.js dosyasına şu kodu ekleyin:

```
app.get('/about', (req, res) => {
  res.render('about', {
    title: 'About Me',
    name: 'Cem Can'
  })
})
```

about.hbs dosyasını aşağıdaki gibi güncelleyin:

```
<body>
<h1>{{title}}</h1>
<img src="/img/abc.png">
<p>Powered by {{name}}</p>
</body>
```

Dosyaları kaydedin ve sayfayı yenileyin. views klasöründe help.hbs dosyası oluşturun, help.html dosyasından içeriği kopyalayıp yapıştırın. Hbs dosyasında şu kodu kullanarak içeriği dinamik hale getirin:

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

app.js dosyasına şu kodu ekleyin:

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

Dosyaları kaydedin ve sayfayı yenileyin.

Express, template görünümleri için belirli bir klasör adı arar. views klasörünü templates olarak değiştirin ve hatanın oluştuğunu görün. app.js dosyasında aşağıdaki kodu ekleyerek views klasörünün adını değiştirin:

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

Dosyayı kaydedin ve tekrar çalıştırın, çalıştığını göreceğiz.

404 Sayfaları

Var olmayan bir sayfaya gidin (örneğin: localhost:3000/me) ve express'ten gelen genel hata mesajını görün.

app.js dosyasına aşağıdaki kodu ekleyin:

```
app.get('*', (req, res) => {
  res.send('My 404 Page')
})
```

My 404 Page

Sayfayı kaydedin ve yenileyin.

Daha iyi bir 404 mesajı için app.js dosyasına aşağıdaki kodu ekleyin:

```
    {{header}}
    <p>{{errorMessage}}</p>
    {{footer}}
```

```
app.get('*', (req, res) => {
  res.render('404', {
      title: '404',
      name: 'Cem Can',
      errorMessage: 'Page not found'
  })
})
app.get('/help/*', (req, res) => {
  res.render('404', {
      title: '404',
      name: 'Cem Can',
      errorMessage: 'Help article not found'
  })
})
```

Page not found

Styling the app- I

Uygulamaya stil vermek için CSS kullanacağız ve sayfaların renkleri, yazı tipleri ve düzenini güncelleyeceğiz. Bu değişiklikler tüm sayfalara uygulanacak, böylece tutarlı bir görünüm elde edilecek.

1. **styles.css Dosyasını Güncelle:**
   Öncelikle styles.css dosyasını düzenleyeceğiz. Mevcut içeriği kaldırarak başlıyoruz. Body için metin rengini, yazı tipini, maksimum genişliği ve kenar boşluğunu ayarlayacağız. Footer'ı farklı bir metin rengi, kenarlık, kenar boşluğu ve dolgu ile stilize edeceğiz. Başlık için kenar boşluğu ayarları ve daha büyük bir yazı tipi boyutu ekleyeceğiz. Başlık içindeki bağlantıları (linkleri) aynı renkte, kenar boşluğu olan ve altı çizili olmayacak şekilde stilize edeceğiz.

```
/* styles.css */
body {
  color: #333333;
  font-family: arial;
  max-width: 650px;
  margin: 0 auto;
  padding: 0 16px;
}

footer {
  color: #888888;
  border-top: 1px solid #eeeeee;
  margin-top: 16px;
  padding: 16px 0;
}

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

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

```
header a {
  color: #888888;
  margin-right: 16px;
  text-decoration: none;
}
```

Sonrasında, footer.hbs dosyasını düzenleyerek footer içeriğini yapılandırıyoruz. Footer içeriğindeki oluşturanı belirtmek için basit bir metin ekliyoruz.

```
   <!-- footer.hbs -->
<footer>
    <p>Oluşturan: {{name}}</p>
</footer>
```

Styling part II adımında, uygulamanın stilini daha da iyileştireceğiz. İlk olarak, about.hbs dosyasına giderek profil resmi için img etiketini değiştireceğiz. Ardından, CSS dosyasına giderek profil resminin genişliğini belirleyeceğiz. Footer'ın sayfanın altında sabit kalmasını sağlayacağız. Bunun için, index.hbs dosyasının body bölümünü düzenleyerek main-content divini ekleyeceğiz ve CSS dosyasında bu div için gerekli stillendirmeleri yapacağız. Bu düzenlemelerin tüm sayfalara uygulanması için diğer sayfalarda da aynı değişiklikleri yapacağız. Son olarak, favicon ekleyerek uygulamanın simgesini belirleyeceğiz.

Öncelikle, about.hbs dosyasına giderek profil resmi için img etiketini değiştiriyoruz:

```
<!-- about.hbs -->
<div class="main-content">
    {{header}}
    <img class="portrait" src="/img/abc.png">
</div>
{{footer}}
```

Ardından, CSS dosyasına giderek profil resminin genişliğini belirliyoruz:

```
.portrait {
  width: 250px;
}
```

Footer'ın sayfanın altında sabit kalmasını sağlamak için index.hbs dosyasının body bölümünü düzenliyoruz:

```
    <div class="main-content">
        {{header}}
        <p>Use this website to get your weather!</p>
    </div>
    {{footer}}
```

CSS dosyasında, body için aşağıdaki stillemeleri ekliyoruz:

```
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh; /* viewport height 100 percent */
}
.main-content {
  flex-grow: 1;
}
```

Son olarak, favicon ekleyerek uygulamanın simgesini belirliyoruz. index.hbs dosyasının head bölümüne favicon eklemesi yapıyoruz:

```
<head>

    <title>Weather</title>
    <link rel="icon" href="/img/weather.png">
    <!-- diğer head içerikleri -->

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

Bu değişiklikleri diğer sayfalara da uygulamak için aynı adımları help.hbs, 404.hbs ve index.hbs dosyalarına da uygulayabiliriz.