Node Js Ödevi

9th Week

Farhan Ahmad

20360859096

Geçen hafta **Express.js ile Statik Web Sayfalarının Servis Edilmesi ve Dinamik İçeriklerin Hazırlanması ve Handlebars Partialları Nodemon** gibi konuları incelemiştik. Bu hafta ise **Tarayıcı Üzerinden Express API'leri Oluşturma ve Kullanma Rehberi** gibi konuları ele alacağız.

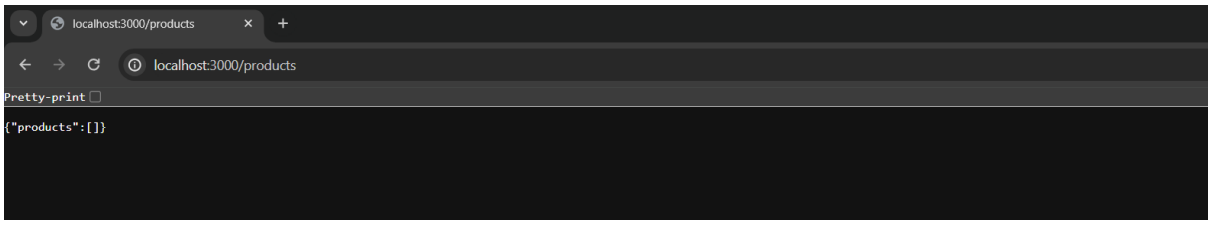
Tarayıcıdan API Erişimi:

Hava durumu sayfasını bir API oluşturacak şekilde değiştireceğiz. Konum bilgisini sağlamak için sorgu dizgisini (query string) kullanacağız. Daha önce geocode.js dosyasında da kullandık. app.js dosyasına gidin ve weather rotası altında test için yeni bir url oluşturun.

Bu, boş bir ürünler dizisi JSON nesnesi döndürür:

```
app.get('/products', (req, res) => {
  res.send({
      products: []
  })
})
```

nodemon app.js



Sonuç statiktir ve asla değişmez. Bir arama yapmak ve belirli bir ürünü döndürmek istiyorsak sorgu dizgisini kullanmamız gerekir. Sorgu dizesi, url'in sonunda ?key=value şeklinde ve birden fazla key-value çiftini geçmek için & kullanırız, örneğin ?search=oyunlar&rating=5

Ürünler rotasında req.query'i yazdırın:

```
app.get('/products', (req, res) => {
  console.log(req.query)
  res.send({
      products: []
  })
})
```

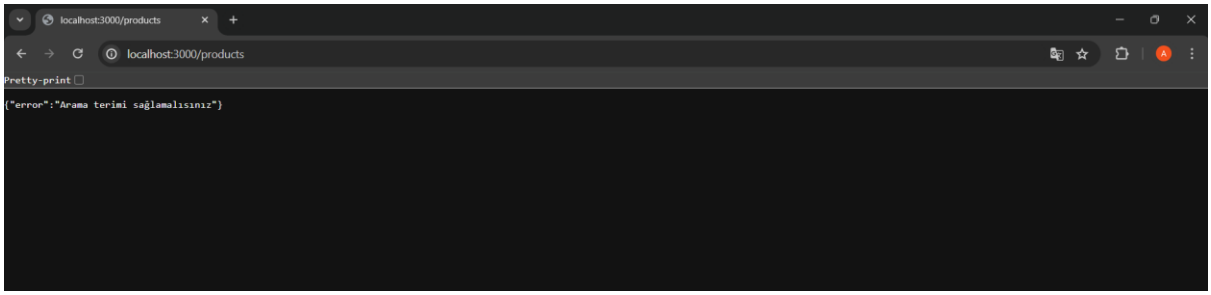
Dosyayı kaydedin ve localhost:3000/products?search=oyunlar&rating=5 adresine gidin. Sonuç hala aynıdır ancak VS Code çıktısında sorgu dizgilerinin listesini görürüz. Belirli bir sorgu dizesine erişmek için.

```
app.get('/products', (req, res) => {
  console.log(req.query.search)
  res.send({
      products: []
  })
})
```

Dosyayı kaydedin ve yenileyin.

Belirli sorgu dizgileri zorunlu yapılabilir. Arama zorunlu, derecelendirme ise isteğe bağlı olacak şekilde değiştirin.

```
app.get('/products', (req, res) => {
  if (!req.query.search) {
      res.send({
          error: 'Arama terimi sağlamalısınız'
      })
  }
  console.log(req.query.search)
  res.send({
      products: []
  })
})
```



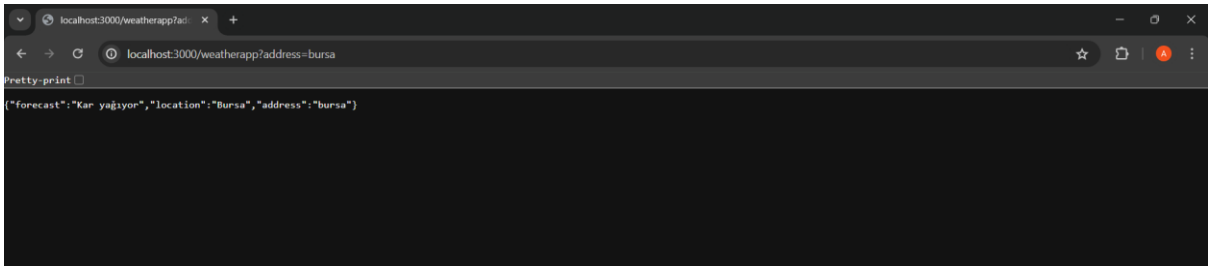
Dosyayı kaydedin ve localhost:3000/products adresine gidin

Beklenildiği gibi tarayıcıda hata görüntülenir. Öte yandan, VS Code'da, client'a gönderildikten sonra başlık ayarlanamıyor mesajı görünür. Bu, bir yanıtı gönderdikten sonra kodun devam edip başka bir yanıt göndermeye çalıştığını belirtir. Bir istek için iki kez yanıt gönderemeyiz. Bu nedenle, return veya else koşulunu kullanmamız gerekmektedir. Genellikle return kullanılır.

```
app.get('/products', (req, res) => {
  if (!req.query.search) {
      return res.send({
          error: 'Arama terimi sağlamalısınız'
      })
  }
  console.log(req.query.search)
  res.send({
      products: []
  })
})
```

Şimdi hava durumu endpoint'ini güncelleyeceğiz. Bir adres kabul edeceğiz. Adres yoksa, hata mesajı döndürün.

```
app.get('/weather', (req, res) => {
  if (!req.query.address) {
      return res.send({
          error: 'Bir adres sağlamalısınız'
      })
  }
  res.send({
      forecast: 'Kar yağıyor',
      location: 'Bursa',
      address: req.query.address
  })
})
```



Dosyayı kaydedin ve localhost:3000/weather?address=bursa adresine gidin, ayrıca İzmir için de test edin

Zaten geocoding ve hava durumu için bazı kodlarımız var. Bu kodları buraya kopyalayın. İki js dosyası ile tüm utils dizinini kopyalayın ve web-server/src dizinine yapıştırın.

Modüllerin Yüklenmesi: İlk olarak, web sunucusu dizinindeki utils dizininde kullanılan bazı npm modüllerinin yüklenmemiş olabileceğini fark ettik. Bu nedenle, nodemon'ı durdurarak (Ctl C) ve eksik modülleri yükleyerek başlamamız gerekiyor.

npm i request

Geocode ve Forecast Kullanımı: Artık geocode ve forecast işlevlerini kullanabiliriz. Bunun için, app.js dosyasında en üst kısıma gereksinimleri eklememiz gerekiyor.

```
const geocode = require('./utils/geocode')
const forecast = require('./utils/forecast')
```

Weather Endpoint Güncelleme: Hava durumu endpoint'ini güncelleyerek, geocode işlevini kullanarak iki argüman almasını sağlıyoruz. Bu işlemi, önceki adımlarda yaptığımız gibi yeniden yapılandırma kullanarak gerçekleştiriyoruz.

```
  geocode(req.query.address, (error, { latitude, longitude, location }) => {
      if (error) {
          return res.send({ error })
      }
      forecast(latitude, longitude, (error, forecastData) => {
          if (error) {
              return res.send({ error })
          }
          res.send({
              forecast: forecastData,
              location,
              address: req.query.address
```

```
          })
      })
  })
})
```

Varsayılan Parametreler: ES6'da varsayılan parametreleri kullanarak, bir fonksiyonun parametrelerine varsayılan değerler atayabiliriz. Bu özelliği, bir örnek üzerinde görelim.

```
const greeter = (name = 'kullanıcı', age) => {
  console.log('Merhaba ' + name)
}
greeter('Can')
greeter()
```

Nodemon'u tekrar çalıştırın ve localhost:3000/weather?address=! adresine gidin. Bir hata mesajı görünür ve sunucu çöker.

Sorun, geocode çağrısında, başarılı veri için (latitude, longitude, location) bir değer geçirmememizdir, çünkü hata döndürülür ve hala nesneyi yeniden yapılandırmaya çalışırız. Bunu düzeltmek için, bir varsayılan değer sağlayın.

Bu adımları takip ederek, Express ile basit bir web sunucusu oluşturabilir ve API'ler oluşturabilirsiniz.

Frontend Güncelleme: Öncelikle, frontend'i güncellememiz gerekiyor. Bir form sağlayacağız. Sadece index dosyasında başlık kısmında js kodu var. Verileri istemci tarafındaki javascript dosyasında alacağız.

Fetch API Kullanımı: Verileri almak için fetch API'sini kullanacağız. Fetch API, node.js'nin bir parçası değildir ancak modern tarayıcılarda sağlanmaktadır. Bu js dosyası sunucu üzerinde değil, istemci tarafında çalıştırılacaktır. Bu nedenle, burada kullanmak oldukça uygundur. then ise ileri kurslarda göreceğimiz promise'ların bir parçasıdır.

```
fetch('http://puzzle.mead.io/puzzle').then((response) => {
    response.json().then((data) => {
        console.log(data)
    })
})
```

Form Oluşturma: Arayüzde bir arama formu oluşturacağız. index.hbs dosyasında, kapanış div'inden önce aşağıdaki gibi bir form ekleyin:

```
<form>
    <input placeholder="Konum">
    <button>Ara</button>
</form>
```

Form İşlemleri: Forma veri göndermek ve bu veriyi işlemek için aşağıdaki adımları takip edin:

```
const weatherForm = document.querySelector('form')
weatherForm.addEventListener('submit', (e) => {
    e.preventDefault()
    const location = document.querySelector('input').value
    console.log(location)
})
```

Bu kod, formun submit olayını dinler ve sayfanın yenilenmesini önler. Ardından, formdaki girdiyi alır ve konsola yazdırır.

Fetch Kullanarak Sunucuya Veri Gönderme: Formdan alınan veriyi sunucuya göndermek için fetch API'sini kullanabiliriz. Bu işlemi, form olay dinleyicisi içinde gerçekleştirebiliriz.

```
const weatherForm = document.querySelector('form')
const search = document.querySelector('input')

weatherForm.addEventListener('submit', (e) => {
    e.preventDefault()
    const location = search.value

 fetch('http://localhost:3000/weather?address=' + location).then((response) =>
{
        response.json().then((data) => {
            if (data.error) {
```

```
                console.log(data.error)
            } else {
                console.log(data.location)
                console.log(data.forecast)
            }
        })
    })
})
```

Bu kod, formu dinler, veriyi sunucuya gönderir ve sunucudan gelen yanıtı konsola yazdırır.

Arama Terimini Almak: Kullanıcının arama yapacağı terimi almak için, HTML formu ve JavaScript kodunu kullanıyoruz. JavaScript, formun gönderilmesini dinler ve kullanıcının girdiği konumu alır.

```
const weatherForm = document.querySelector('form')
const search = document.querySelector('input')
weatherForm.addEventListener('submit', (e) => {
    e.preventDefault()
    const location = search.value
    console.log(location)
})
```

Fetch İşlemini Form Dinleyicisine Taşımak: Formun submit olayını dinleyen kısımda, sunucuya veri gönderme işlemini gerçekleştiriyoruz. Fetch API'si, sunucudan veri almak için kullanılır. Bu durumda, kullanıcının girdiği konumu sunucuya gönderiyoruz ve sunucudan gelen yanıtı işliyoruz.

```
const weatherForm = document.querySelector('form')
const search = document.querySelector('input')
weatherForm.addEventListener('submit', (e) => {
    e.preventDefault()
    const location = search.value

    fetch('http://localhost:3000/weather?address=' + location).then((response)
=> {
        response.json().then((data) => {
            if (data.error) {
                console.log(data.error)
            } else {
                console.log(data.location)
                console.log(data.forecast)
```

```
            }
        })
    })
})
```

Kullanıcı Arayüzünü Güncelleme: Kullanıcı arayüzünde bazı değişiklikler yaparak, JavaScript ile içeriği güncelliyoruz. Kullanıcıya işlem sırasında bilgi vermek için, önce "Yükleniyor..." mesajını gösteriyoruz. Sonra, sunucudan gelen verilere göre bilgileri güncelliyoruz.

```
<p id="message-1"></p>
<p id="message-2"></p>
const messageOne = document.querySelector('#message-1')
const messageTwo = document.querySelector('#message-2')

// Event listener içine ekleyin
e.preventDefault()
const location = search.value
messageOne.textContent = 'Yükleniyor...'
messageTwo.textContent = ''

fetch('http://localhost:3000/weather?address=' + location).then((response) =>
{
    response.json().then((data) => {
        if (data.error) {
            messageOne.textContent = data.error
        } else {
            messageOne.textContent = data.location
            messageTwo.textContent = data.forecast
        }
    })
})
```

Stil Ekleme: Son olarak, buton için yeni bir stil ekliyoruz. Bu stil dosyası, input alanı ve buton için belirli bir görünüm sağlar.

```
input {
    border: 1px solid #cccccc;
    padding: 8px;
}
button {
    cursor: pointer;
    border: 1px solid #888888;
    background: #888888;
```

```
    color: white;
    padding: 8px;
}
```

Bu adımlar, kullanıcının arama yapması için bir form oluşturmayı, bu formun verilerini sunucuya göndermeyi ve sunucudan gelen yanıtları kullanarak kullanıcı arayüzünü güncellemeyi içerir.