



# **Node.js ile Web Programlama**

## **Rapor - 9**

Mahire Zühal Özdemir

## İÇİNDEKİLER

ŞEKİLLER DİZİN .....	3
Browser ile API Erişimi .....	4
ES6 Varsayılan Parametreler .....	7
Node JS Fetch API .....	8
Arama Formu Oluşturma .....	9

## ŞEKİLLER DİZİNİ

Şekil 1: Boş Ürün Get İsteği	4
Şekil 2: Ürün değerini yazdırma	4
Şekil 3: Zorunlu req sorguları	5
Şekil 4: Return İşlemi	6
Şekil 5: Hava Durumu Uygulaması	6
Şekil 6: App.js Dosyası	7
Şekil 7:Fetch Api	9

## Browser ile API Erişimi

- app.js dosyasına gidilir ve hava durumu rotası altında test için yeni bir URL oluşturulur.
- Oluşturulan URL, boş bir ürünler dizisi içeren bir JSON nesnesi döndürür.

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


Şekil 1: Boş Ürün Get İsteği

- Çalıştırmak için; **nodemon src/app.js -e js,hbs** komutu çalıştırılır.
- Tarayıcı açılır ve adres çubuğuna **localhost:3000/products** URL'si yazılır.
- Bu, belirtilen URL'ye gidilerek sunucudan dönen sonuçların tarayıcıda görüntülenmesini sağlar.
- Sunucudan dönen sonuçların statik olduğu ve değişmediği belirtilir.
- Belirli bir ürünü aramak ve döndürmek için sorgu dizgisinin kullanılması gerektiği ifade edilir.
- Sorgu dizgisi, URL'nin sonunda **?key=value** şeklinde gelir ve birden fazla anahtar-değer çifti geçirmek için **&** işareti kullanılır.
- Gelen isteğin sorgu dizgisini konsola yazdırmak için aşağıdaki satırlar eklenir. **req.query**, isteğin sorgu dizgisini içeren bir nesnedir. Bu nesne, URL'de bulunan anahtar-değer çiftlerini temsil eder.

```
app.get('./products',(req,res)=>{  
  console.log(req.query.search)  
  //search değerini yazdırır  
  //sonucu görmek için ; konsola yazdırır(vs code konsol)  
  //yani client tarafında yazılan sorgu çekilebilir  
  res.send({  
    products:[]  
  })  
})
```

Şekil 2: Ürün değerini yazdırma

- Bu kod, "/products" rotasına gelen isteklerin işlenmesini sağlar ve gelen isteğin sorgu dizgisini konsola yazdırır. Sonrasında istemcilere boş bir ürünler dizisi döndürülür. Bu sayede, "/products" rotası üzerinde yapılan isteklerin işlevselliği ve gelen verilerin incelenmesi sağlanır.

 localhost:3000/products?search=games&rating=5

- Zorunlu sorgular için if-else blokları yazılabilir.
- **if (!req.query.search) { ... }**: Bu satır, gelen isteğin sorgu dizgisinde "search" parametresinin olup olmadığını kontrol eder. Eğer "search" parametresi yoksa, bir hata mesajı döndürür. Yani, "search" parametresi zorunlu hale gelir.
- **res.send({ products: [] })**: Son olarak, bu satır istemciye bir yanıt döndürür. Yanıt, bir JSON nesnesi içerir ve bu nesnenin içinde "products" adında bir boş dizi bulunur.

```
//req sorgularını zorunlu hale getirmek için;
app.get('./products',(req,res)=>{
  if(!req.query.search){
    res.send({
      error:'search girilmemiş'
    })
  }
  console.log(req.query.search)
  res.send({
    products:[]
  })
})
```

Şekil 3: Zorunlu req sorguları

- if bloğundan sonra kodun çalışmaması için **return** kullanılır.

```

//kodun devamı çalışmasın diye
//tek bir req için tek bir response döner
app.get('/products',(req,res)=>{
  if(!req.query.search){
    return res.send({
      error:'search girilmemiş'
    })
  }
  console.log(req.query.search)
  res.send({
    products:[]
  })
})
})

```

Şekil 4: Return İşlemi

- Hava durumu uygulamasına zorunlu sorguları eklemek için adres değişkeni eklenir.

```

app.get('/weather', (req, res) => {
  if (!req.query.address) {
    return res.send({
      error: 'Adres bilgisi eksik.' })
  }
  res.send({
    forecast: 'hava yağışlı',
    location: 'Bursa',
    address: 'Yıldırım'
  })
});

```

Şekil 5: Hava Durumu Uygulaması

- ***if (!req.query.address) { ... }:*** Bu satır, gelen isteğin sorgu dizisinde "address" parametresinin olup olmadığını kontrol eder. Eğer "address" parametresi yoksa, bir hata mesajı döndürür. Yani, "address" parametresi zorunlu hale gelir.
- ***res.send({ forecast: 'It is snowing', location: 'Bursa', address: req.query.address }):*** Eğer "address" parametresi varsa, bu satır istemciye bir yanıt döndürür. Yanıt, bir JSON nesnesi içerir ve bu nesnenin içinde hava durumu tahmini (forecast), konum bilgisi (location) ve istemciden gelen adres (address) bulunur.
- App.js dosyasında son değişiklikler aşağıdaki gibi eklenir.

```

app.get('/weather', (req, res) => {
  if (!req.query.address) {
    return res.send({
      error: 'You must provide an address'
    })
  }
  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
      })
    })
  })
})
})

```

Şekil 6: App.js Dosyası

## ES6 Varsayılan Parametreler

- Dosyanın içine, greeter adında bir fonksiyon tanımlayalım. Bu fonksiyon, bir isim parametresi alır ve konsola "Hello" kelimesi ve verilen isim parametresini yazdırır.
- greeter('Can'): Fonksiyon, "Can" ismiyle çağrılır ve "Hello Can" çıktısı konsola yazdırılır.
- Ardından, fonksiyon çağrılırken herhangi bir argüman verilmez, yani greeter() şeklinde çağrılır. Ancak, fonksiyonun içinde varsayılan bir isim parametresi tanımlanmadığı için, bu durumda undefined olarak değer alır ve konsola "Hello undefined" çıktısı yazdırılır.

```

const greeter = (name) => {
  console.log('Hello ' + name)
}
greeter('Can')
greeter()

```

```

Hello Can
Hello undefined

```

- Sonrasında, fonksiyonun içinde varsayılan bir isim parametresi tanımlanır. Fonksiyon çağrıldığında isim parametresine verilen değer yoksa, "user" değeri kullanılır.

- greeter('Can'): Fonksiyon, "Can" ismiyle çağrılır ve "Hello Can" çıktısı konsola yazdırılır.
- greeter(): Fonksiyon, herhangi bir isim verilmeden çağrılır. Bu durumda varsayılan değer olan "user" kullanılır ve "Hello user" çıktısı konsola yazdırılır.

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

Hello Can  
Hello user

## Node JS Fetch API

- Fetch API'nin kullanımını ve sunucudan veri almayı işlemidir. Bu işlemler, JavaScript dosyalarının istemci tarafında yani tarayıcıda çalıştırılması gerektiğinden, sunucu tarafında değil istemci tarafında gerçekleştirilir.
- İlk olarak, Fetch API kullanılarak bir URL'den veri alınır. Bu kod, "http://puzzle.mead.io/puzzle" adresinden veri alır.
- Alınan yanıt, .then() yöntemi kullanılarak işlenir. Bu yöntem, bir Promise'in tamamlanmasını bekler ve ardından bir işlem gerçekleştirir.
- İkinci .then() yöntemi, yanıtı JSON formatına dönüştürür ve bu JSON verisini konsola yazdırır.
- Tarayıcının geliştirici araçları kullanılarak konsol sekmesi kontrol edilir ve verinin doğru şekilde alındığı doğrulanır.
- Ardından, sunucudan hava durumu bilgilerini almak için başka bir Fetch API isteği yapılır. Bu istek, belirli bir adresin hava durumu bilgilerini alır.
- Gelen yanıt, .then() yöntemi kullanılarak işlenir. Eğer hata varsa, hata konsola yazdırılır. Eğer hata yoksa, konum ve hava durumu bilgileri konsola yazdırılır.
- Dosya kaydedilir ve ana sayfa yenilenir. Bu, kodun çalıştırılmasını sağlar.
- Adres bilgisi "!" olarak değiştirilir ve dosya tekrar kaydedilir ve ana sayfa yenilenir.



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

Şekil 7:Fetch Api

## Arama Formu Oluşturma

HTML formu aracılığıyla kullanıcının konum bilgisini girmesine ve ardından bu bilgiyi sunucuya göndererek hava durumu bilgilerini almasına olanak tanıyan işlevsel form oluşturalım.

- İlk olarak, "index.hbs" dosyasına gidilir ve kullanıcıya bir arama formu eklenir. Formun içine bir metin girişi (input) ve bir arama düğmesi (button) eklenir. Metin girişi için bir yer tutucu (placeholder) eklenir.
- Ardından, "app.js" dosyasında form elemanı seçilir ve arama düğmesine bir olay dinleyici (event listener) eklenir. Bu dinleyici, form gönderildiğinde tetiklenir ve bir işlev çalıştırır.
- Olay dinleyicisi içinde, öncelikle varsayılan form gönderme davranışını (refresh) engellemek için e.preventDefault() kullanılır. Ardından, kullanıcının girdiği konum bilgisi alınır.
- Alınan konum bilgisi ile sunucuya bir Fetch API isteği gönderilir. Bu istek, "/weather" rotası altındaki sunucuya gönderilir ve konum bilgisi isteğin parametresi olarak eklenir.
- Sunucudan gelen yanıt işlenir ve eğer hata varsa hata mesajı konsola yazdırılır, yoksa konum ve hava durumu bilgileri konsola yazdırılır.
- "index.hbs" dosyasına, formun altına iki adet paragraf eklenir ve her birine birer id atanır.
- "app.js" dosyasında, bu iki paragraf elementi seçilir ve birinci paragrafa bir metin içeriği eklenir.
- Dosyalar kaydedilir ve tarayıcı yeniden yenilenir. Artık kullanıcı arama formunu kullanarak konum bilgisi girebilir ve gönder düğmesine tıkladığında, hava durumu bilgileri alınarak ekrana yazdırılır.