7.hafta

Furkan Kazım Çam

21360859036

Bu haftaya Object Property Shorthand and Destructing konusu ile başlayalım.

Bir dosya açın ve aşağıdaki kodları yazın.

```
1   const userName='Can'
2   const userAge=25
3
4   const user={
5       userName:userName,
6       userAge:userAge,
7       location:'Bursa'
8   }
9
```

console.log(user) yazarsanız user nesnesinin bütün değişkenleri konsola yazdırılır.

```
[Running] node "c:\Users\furka\OneDrive\Masaüstü\Bah
{ userName: 'Can', userAge: 25, location: 'Bursa' }
```

Bunu kısaltabiliriz.

```
const userName='Can'
const userAge=25

const user={
    userName,
    userAge:userAge,
    location:'Bursa'
}
```

Bu şekilde console.log(user) ile çalıştırırsanız yine aynı sonucu almalısınız.

Product nesnesi tanımlayarak pekiştirmeye devam edelim.

```
const product ={
    label:'Red notebook',
    price:3,
    stock:201,
    salePrice:undefined,
}
```

Bu nesnenin özelliklerine erişmek için

```
const label=product.label
const stock=product.stock
```

Gibi kod satırlarını kullanabiliriz fakat bunu daha verimli hale getirmek istersek tabi ki bir yöntemi var.

```
const {label, stock} = product
console.log(label)
console.log(stock)
```

Bu syntax ile product nesnesinden erişmek istediğimiz değişkenlerin isimlerini süslü parantezler içinde vererek kullanılmayan değişkenlerin arka planda çekilmeyerek işlemi hızlandırmış oluruz.

```
[Running] node "c:\Users\
Red notebook
201
```

Sonucunda bu çıktıyı alırız fakat bir terslik var undefined değişken var ve sorun çıkmıyor. Neden?

```
const {label, stock, rating} = product
console.log(label)
console.log(stock)
console.log(rating)
```

Product nesnesinde rating diye bir değişken ve değer ataması yapmadık buna rağmen çıktımızda undefined görüyoruz.

```
[Running] node "c
Red notebook
201
undefined
```

İstersek değişkenleri çağırırken onlara başka isim verebilir veya değere verebiliriz.

```
const {label: productLabel, stock, rating=5} = product
console.log(productLabel)
console.log(stock)
console.log(rating)
```

Çalıştırdığımızda herhangi bir problem olmadan çalışacaktır.

Peki ratingi ilk önce productı tanımlarken tanımlayıp değer ataması yapsaydık. Konsola yazdırılırken hangi değer yazdırılır.

```
const product ={
    label:'Red notebook',
    price:3,
    stock:201,
    salePrice:undefined,
    rating:4.2,
}

// const label=product.label
// const stock=product.stock

const {label: productLabel, stock, rating=5} = product
console.log(productLabel)
console.log(stock)
console.log(rating)
```

Çıktı ise;

```
[Running] node "c:\Users\
Red notebook
201
4.2
```

Göründüğü gibi ilk atanan değer alınır.

Callback fonksiyonlar ile entegre ederek örnek çözelim.

```
const transaction = (type, { label, stock }) => {
    console.log(type, label, stock)
    }
    transaction('order', product)
```

Çıktısı ise;

```
[Running] node "c:\Users\furka\
Red notebook
201
4.2
order Red notebook 201
```

Şimdi bu öğrendiklerimizi yine önceki haftalarda yaptığımız hava durumu uygulamasında bazı yerleri değiştirerek pekiştirelim.

App.js dosyası ile başlayalım.

```
geocode('Bursa', (error, {latitude, longitude, location} = {} ) => {
    if (error) {
    return console.log(error)
    }
    Complexity is 3 Everything is cool!
    forecast(latitude, longitude, (error, forecastData) => {
    if (error) {
    return console.log(error)
    }
    console.log(location)
    console.log(forecastData)
    })
    })
```

Bu şekilde düzenlememiz gerekmektedir çünkü data nesnesinin sadece latitude longitude ve location metodlarını kullanıyoruz. Kısaca datanın kullanılmayan çok fazla metodu olduğu için gereksiz.

Forecast.js dosyası ile devam edelim.

```
Complexity is 3 Everything is cool!
request( {url, json:true}, (error, { body }) => {
    if (error) {
    callback('Unable to connect to weather service', undefined)
    } else if (body.error) {
    callback('Unable to find location', undefined)
    }
    else {
    callback(undefined, 'Hava şu anda: ' +
   body.current.weather_descriptions[0] + ' Hava sıcaklığı şu anda: ' +
   body.current.temperature + ' derece ve hissedilen sıcaklık: ' +
   body.current.feelslike + ' derece')
    }
    })
```

Böyle değişmesi gerekir burda da response yerine sadece body metodunu kullandığımız için böylesi daha verimli olucaktır.

Geocode.js ile devam edelim.

```
request( {url, json: true}, (error, { body }) => {
    if (error) {
    callback('Unable to connect to location services!', undefined)
    } else if (body.features.length === 0) {
    callback('Unable to find location. Try another search',
   undefined)
    } else {
    callback(undefined, {
    latitude: body.features[0].center[1],
    longitude: body.features[0].center[0],
    location: body.features[0].place_name
    })
    }
    })
```

Yine forecast.js dosyası ile aynı şekilde olduğu için bu şekilde değiştirebiliriz.

Express

Bu zamana kadar yazdığımız uygulamaları hep konsol üzerinden çalıştırdık. Bunu web üzerinden de yapabiliriz. Yeni bir klasör açıp ismine web-server ismini verin.

Konsola npm init -y , npm i Express kodlarını sırayla girin.

Web-server klasörünün içine bir klasör daha açıp ismine src verin. Src klasörünün içinde ise app.js adlı bir dosya açınız.

```
const express =require("express")
const app=require("path")
```

Bu kodları içine yazın. Devamında ise

```
app.get('', (req, res) => {
    res.send('Hello express!')
    })

    app.listen(3000, () => {
    console.log('Server is uo on port 3000.')
    })
```

Yazarak devam ediniz. İlk açılan sayfaya Hello Express! Yazdırır ve 3000 numaralı portta çalıştırılır. Porta her hangi bir değer verilmez ise default olarak 80 değerini alır.

node src/app.js ile çalıştırınız. Tarayıcınızı açıp localhost:3000 urlsini girerseniz kodun çıktısını görebilirsiniz.

Yeni sayfa ekleyerek devam edebiliriz.

```
app.get('/help', (req, res) => {
    res.send('Help page')
    })
```

localhost:3000/help url'si ile bu sayfaya ulaşabilirsiniz.

About sayfasını da aynı şekilde ekleyelim.

```
app.get('/about', (req, res) => {
    res.send('About');
    })
```

Ve weather sayfasını da.

```
app.get('/weather', (req, res) => {
    res.send('Your weather')
    })
```

Sayfalara HTML JSON gibi formatlarda girdi gönderebilir ve çıktı alabilirsiniz.

HTML örneği;

```
app.get('', (req, res) => {
    res.send('<h1>Weather</h1>')
    })
```

JSON örneği;

```
app.get('/help', (req, res) => {
    res.send({
    name: 'Andrew',
    age: 27
    })
    })
```

Dizi şeklinde de girdi gönderebiliyorsunuz.

```
app.get('/help', (req, res) => {
    res.send([{
    name: 'Andrew',
    }, {
    name: 'Sara'
    }])
    })
```

About ve weather sayfalarını düzenleyerek devam edelim.

```
app.get('/about', (req, res) => {
    res.send('<h1>About</h1>')
    })

app.get('/weather', (req, res) => {
    res.send({
    forecast: 'It is snowing',
    location: 'Bursa'
    })
})
```

Express CSS, HTML, JavaScript gibi diller ile entegre çalışır ve bu dillerde yazılan sayfaları web sunucusuna girdi olarak alabilir.

Web-server klasörü altına bir public adında klasör açın ve bir index.html dosyası açıp statik bir sayfa yazalım.

Sayfamızın ana şeması;

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ana Sayfa</title>
    </head>
    <body>
        <h1>Bu statik bir web sayfasıdır!</h1>
    </body>
</html>
```

App.js dosyasına gidip requireların hemen altına

console.log(__dirname)

console.log(__filename)

bu kod satırlarını yazın. Bunlar dosya üst dizini ve direkt olarak dosyanın dizinini verirler.

Biz dirname olanı kullanacağız.

Önceki iki satırı silip bunları yazın

const path = require('path')

console.log(__dirname)

console.log(path.join(__dirname, '../public'))

dirname dizininden bir önceki dizine gidip ordan public adlı bir klasöre girebiliyor mu diye kontrol ediyoruz aslında. Burdan çıktı alır isek bunları silip alttaki adımlara devam edebiliriz.

App.js dosyasının en tepesindeki kodların böyle olması gerekmektedir.

const app = express()

const publicDirectoryPath = path.join(__dirname, '../public')

app.use(express.static(publicDirectoryPath))

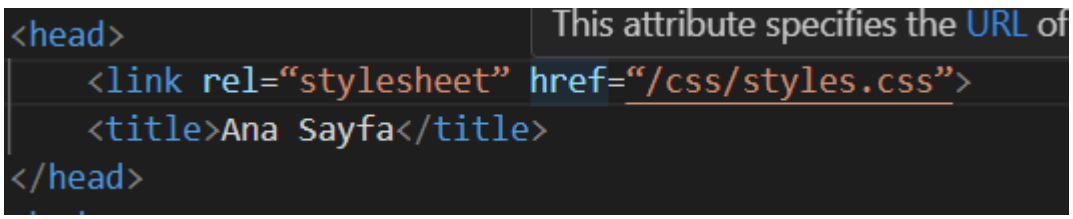
Sayfaları get ile tanımlamıştık bunları silebiliriz ve onların yerine ise o sayfaları html olarak yazmalıyız. Yazınız ve çalıştırınız aynı şekilde çalışmaya devam edecektir.

8.Hafta

Web sitemize devam edelim.

Public klasörünün altına CSS adlı bir klasör açın ve styles.css adlı bir dosya açıp içine css kodlarınızı yazın.

Bu kod sayesinde h1 etiketinin rengi kırmızı olarak değişecektir. Ama tabi ki bu dosyayı html dosyamıza entegre etmemiz gerekmektedir.



Head etiketleri arasına bu şekilde tanımlayarak styles.css dosyamızı entegre etmiş oluyoruz.

Sayfanızı yeniden başlatırsanız yazının renginin kırmızı olduğunu göreceksiniz.

Bir de JavaScript dosyası entegre edelim.

Public klasörü altına Js klasörü açıp içine app.js adlı bir dosya açın.

```
Click here to ask Blackbox to help you code faster
console.log("Bu bir Js dosyasıdır ve client tarafında çalışır");
```

Bu kodu yazın ve html dosyasına bu sayfayı entegre ederek devam edin.

```
<script src="/js/scripts.js"></script>
```

Bu kodu da yine css'te olduğu gibi head etiketleri arasına yazmalısınız.

Sayfayı yenilediğinizde herhangi bir değişiklik yok çünkü sayfada değilişklik yapmadık.

Js dosyamızda sadece konsola yazı yazdırdık.

HTML sayfamıza resim eklerek devam edelim.

```
<body>
    <h1>Bu statik bir web sayfasıdır!</h1>
    <img src="/img/abc.png">
</body>
```

İmg etiketi içinde src değişkenine resmin dizinini vererek sayfamızda resmi görebilirsiniz.

İmg etiketine css dosyamızda özellikler verebiliriz.

```
img {
    width: 250px;
}
```

Resmin genişliği olarak 250px verildi.

Bu zamana kadar yaptığımız web değişikleri statik idi. Bunları dinamik yapabiliriz.

Bunun için npm modüllerine ihtiyacımız vardır.

Nodemon modülünü kapatıp onun yerine hbs modülünü indireceğiz.

npm i hbs ile indirin.

App.js dosyamızın başına app.set('view engine', 'hbs') kod satırı ekleyerek modülü kullanabiliriz.

İndex.hbs dosyasın açınız ve index.html dosyasında bütün kodları kopyala yapıştır yapın. İndex.html dosyasını ise silin.

App.js'e

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

res.render('index')

})
```

Bu kodu ekleyerek index.hbs dosyasını entegre ediyoruz. Çalıştırıp sayfada bir şey değişmediğini görebilirsiniz.

Şu an hala her şey statik bunu dinamik yapalım.

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

Title'ın ve name'in dışardan gönderileceğini gösterir. App.js dosyasından göndereceğiz.

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

Göründüğü gibi title Weather App, name'in ise Cem Can olarak gönderiliyor.

About.html dosyası içinde bir about.hbs dosyası açın ve html den kopyalayıp hbs uzantılı dosyaya kopyalayın ve html uzantılı olanı silin.

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

Body etiketini böyle düzeltin ve app.js dosyasını ise;

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

Bu şekilde düzenleyerek aynı sonucu elde edebilirsiniz.

Son olarak help sayfası içinde htmli hbsye kopyalayıp htmlyi silin. Body etiketini düzenleyin.

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

App.js'te ise;

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

Şeklinde düzenleyin.

Helptext'i dışardan isteyen hbs uzantılı dosyaya app.js'ten girdi gönderiliyor.