Node.JS ile Web Programlama
Rapor - 3

Mahire Zühal Özdemir

İÇİNDEKİLER

ŞEKİLLER DİZİN

Not Uygulaması / Not Ekleme İşlemleri

Yargs modülünü kullanarak basit bir not uygulaması yapmaya başlamıştık. Bu uygulamayı geliştirmek için çeşitli araçlar ve yine yargs modülünü kullanacağız.

Önceki haftada yargs command komutunun handler işlevini görmüştük. Handler; farklı komutları işlemek için kullanılarak modülerliği sağlayan bir işlevdir. Geçen haftalarda handler ile sadece ekrana çıktı yazdırmıştık. Bu hafta handler içerisinde fonksiyon tanımlayarak json dosyaya not ekleme ve silme işlemi yapacağız.

Bu işlemler sırasında ;

- dosya işlemleri için ***fs*** modülünü,
- renkli çıktılar için ***chalk*** modülünü kullanacağımız için import işlemlerini gerçekleştiriyoruz.

```
const fs = require('fs')
const chalk = require('chalk')
```

Şekil 1: Import İşlemleri

İlk olarak ***note_command*** adında js dosyası oluşturuyoruz. Bu dosya içerisinde; notları json dosyaya ekleyebilmek ve notları görüntülemek için ilgili fonksiyonları oluşturuyoruz.

Klasörde bulunan ***notes.json*** dosyamızın içerisine notları eklemek için ***saveNotes*** fonksiyonunu aşağıdaki gibi oluşturuyoruz.

Konsoldan alınan diziyi JSON formatına dönüştürerek kaydediyoruz.

```
const saveNotes = function (notes){
    const dataJson = JSON.stringify(notes)
    fs.writeFileSync('notes.json',dataJson)
}
```

Şekil 2: Not Kaydetme Fonksiyon

Dosyamızdaki verileri okumak için ***loadNotes*** fonksiyonunu oluşturuyoruz.

```
const loadNotes = function(){
    try{
    const data_buffer = fs.readFileSync('notes.json')
    //json formatına çevir: toString
    const data_json =  data_buffer.toString()
    //parse the string
    return JSON.parse(data_json)
}
    catch(e){
        return []
    }

}
```

Şekil 3: Not Okuma Fonksiyon

Oluşturmuş olduğumuz fonkiyonları handler içerisinde kullanmak için tek bir fonksiyon ile toparlamamız gerekir. Bunun için ***addNote*** fonsiyonunu oluşturuyoruz.

```
const addNote = function(title,body){
    const notes = loadNotes()

    // we filtered input for unique titles
    // duplicateNotes = notes that has same title
    const duplicateNotes = notes.filter(function (note){
        // run for each note of the notes array
        return note.title === title
    })

    if (duplicateNotes.length === 0){
        notes.push({
            title:title,
            body:body
        })
        console.log(chalk.green.inverse('Note Added'),notes)
        saveNotes(notes)
    }
    else {
        console.log(chalk.red.inverse('Note Title Found'))
    }
}
```

Şekil 4: Not Ekleme Fonksiyon

Bu fonkiyon ile;

- konsoldan title ve body değerlerini alıyoruz.
- ***loadNotes*** fonksiyonu ile json dosyasına eklemiş olduğumuz girdileri notes değişkenine alıyoruz.
- Title değerimizin unique olmasını istiyoruz.
- Konsoldan alınan title değeri ile önceden eklemiş olduğumuz verilerin title değerinin aynı olmamasını sağlamak için notes dizisi üzerinde filter fonksiyonunu oluşturuyoruz.
- Bu fonksiyon notes içerisindeki notlar üzerinde döngü oluşturarak filtreleme yapıyor.
- Daha önce eklenmiş olan ***title*** bulunduğunda ,not json verisini, ***duplicateNotes*** değişkeni ile tutuyoruz.
- Eğer title daha önce kullanılmamışsa if bloğu içerisinde notes dizisine yeni elemanı ekleyerek ***saveNotes*** fonksiyonunu çağırırız.
- Ekleme işleminin başarılı bir şekilde gerçekleştiğini göstermek içinde chalk kütüphanesini kullanarak ekrana 'Note Added' yazdırıyoruz.
- Eğer title önceden kullanılmışsa else bloğu içerisinde notun eklenmediğine dair ekrana mesaj yazdırıyoruz.

  Oluşturduğumuz fonksiyonları kullanmadan önce exports ve import işlemlerini gerçekleştiriyoruz.

  note_command dosyasında export işlemini aşağıdaki gibi gerçekleştiriyoruz.

```
module.exports = {
    addNote: addNote,
}
```

Şekil 5: Export İşlemi

Bu dosya ve fonksiyonları çağırmak için ***app.js*** dosyamız içerinde import işlemini gerçekleştiriyoruz.

```
const command = require('./note_command.js')
```

Şekil 6: Import İşlemi

AddNote fonksiyonuna title ve body değerlerini vererek çağırıyoruz.

```
    handler: function (argv) {
        command.addNote(argv.title, argv.body)
    },
```

Şekil 7: Handler Fonksiyon

Add fonksiyonu uygulaması görüntüleri:

```
C:\Users\zuhal\Desktop\Node.Js>node app.js add --title="Task-1" --body="Share Project"
Note Added [ { title: 'Task-1', body: 'Share Project' } ]
```

Şekil 8: Not Ekleme

Aynı title değerine sahip not eklemek istediğimizde çıktı aşağıdaki gibi olur.

```
C:\Users\zuhal\Desktop\Node.Js>node app.js add --title="Task-1" --body="Share Project"
Note Title Found
```

Şekil 9:Not Bulundu

Farklı title değerine sahip not ekleme;

```
C:\Users\zuhal\Desktop\Node.Js>node app.js add --title="Task-2" --body="Share Project"
Note Added [
  { title: 'Task-1', body: 'Share Project' },
  { title: 'Task-2', body: 'Share Project' }
]
```

Şekil 10:Not Eklendi

Oluşan notes.json dosyamızın içeriği de aşağıdaki gibidir.

notes.json

C: > Users > zuhal > Desktop > Node.Js > {} notes.json > ...

```
1   [{"title":"Task-1","body":"Share Project"},{"title":"Task-2","body":"Share Project"}]
2
3
```

Şekil 11: Notes.json Dosyası

Not Uygulaması / Not Silme İşlemleri

Not ekleme işlemlerini gerçekleştirdikten sonra ,dosyamız üzerinde not silme işlemini gerçekleştirmek için yeni fonksiyonlar oluşturmamız gerek. Bunun için ***remove_note*** adında fonksiyon oluşturuyoruz.

- İlk olarak konsol üzerinden silmek istediğimiz not ismini alıyoruz.
- loadNotes ile json dosyamızda bulunan notları, note dizisine atıyoruz.
- Bu dizi içerisinde notun bulunup bulunmadığını kontrol etmek için filtre uyguluyoruz.
- Dizi içerisindeki title değerleri ile girdimiz eşleşmediği durumda dizi elemanlarını ***notesToKeep*** dizisine ekliyoruz.
- Eğer ilk başta aldığımız not dizisi ile filtreleme işlemi sonrası oluşan dizi aynı uzunlukta ise title değeri ile eşleşen veri olmadığını ekrana yazdırıyoruz.
- Eğer uzunlukları farklı ise not silinme işlemi gerçekleştiğini ekrana yazdırarak, json dosyasına yeni diziyi kaydediyoruz.

```
const remove_note = function(title){
    const notes = loadNotes()
    const notesToKeep = notes.filter(function (note){
        // run for each note of the notes array
        return note.title !== title
    })

    if (notes.length == notesToKeep.length){
        console.log(chalk.red.inverse('Note not found'))
    }
    else{
        console.log(chalk.green.inverse('Note removed'))
        saveNotes(notesToKeep)

    }
}
```

Şekil 12: Remove Note Fonksiyonu

Bu fonkiyonu ana dosyamız içerisinde kullanmak için export işlemini gerçekleştiriyoruz.

```
module.exports = {
    addNote: addNote,
    remove_note:remove_note
}
```

Şekil 13: Modül Export İşlemi

Yargs.command içerisinde handler ile komuta title girdisini vererek çalıştırıyoruz.

```
    handler: function (argv) {
        command.remove_note(argv.title)
    }
})
```

Şekil 14: Handler İle Remove İşlemi

Not silme işlemine ait uygulama görüntüleri;

```
C:\Users\zuhal\Desktop\Node.Js>node app.js remove --title="Task-3"
Note not found
```

Şekil 15: Bulunmayan Title Değeri

```
C:\Users\zuhal\Desktop\Node.Js>node app.js remove --title="Task-1"
Note removed
```

Şekil 16: Not Silme İşlemi

Silme işlemleri sonrası Json dosyamız aşağıdaki gibi olur.

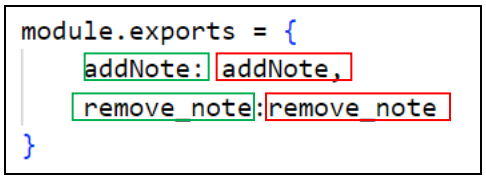
notes.json

C: > Users > zuhal > Desktop > Node.Js > {} notes.json > ...

```
1   [{"title":"Task-2","body":"Share Project"}]
```

Şekil 17: Notes.Json Dosyası Remove İşlemi Sonrası

!!! Module.exports komutu ile birden fazla fonksiyon export edilebilir. Bu komutu kullanırken dikkat etmemiz gereken ilk unsur fonksiyon isimleri ve export isimleridir. Alttaki görselde kırmızı renk ile çevrelenen isimler fonksiyonun kendi ismidir. Bu değer değiştirilmeden fonksiyonun ismi ne ise o yazılarak kullanılır. Yeşil kutucuk ile gösterilen değerler import işlemi ile bu fonksiyonu hangi isimle çağıracağımızı belirtir. Yeşil kutucuktaki değerleri farklı isimler ile değiştirmek mümkündür.

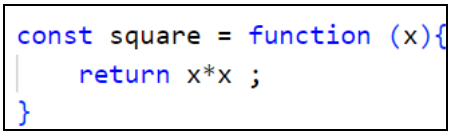


Şekil 18:Exports İşlemleri İsimlendirme

Arrow Function

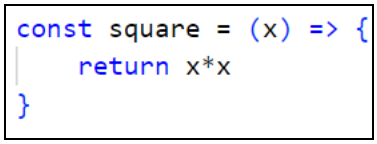
Arrow fonksiyonları, JavaScript'te daha kısa ve daha net bir şekilde fonksiyonlar tanımlamanıza olanak tanıyan bir syntax şeklidir. İlk olarak JS ES6 ile tanıtılmış ve kullanılmaya başlanmıştır.

Klasik yöntem ile fonksiyon tanımlarken fonksiyon ismi ve parametre değerleri aşağıdaki gibi doldurulur.



Şekil 19: Klasik Yöntem İle Fonksiyon Tanımlama

Arrow ile fonksiyon tanımlamak için => işareti , fonksiyon ismi kullanılır. Arrow ile fonksiyonu kısaca aşağıdaki gibi tanımlayabiliriz.



Şekil 20: Arrow Function İle Fonksiyon Tanımlama

Arrow function ile tanımladığımız fonksiyonları biraz daha kısaltmak mümkündür. Basit bir return işlemi içeren fonksiyon aşağıdaki gibi kullanılabilir.

```
const square = (x) => x*x;
console.log(square(4))
```

Şekil 21: Arrow Function Kısa Hali

Nesne Tanımlama ve This Komutu

Node.js'te bir nesne, JavaScript'te olduğu gibi, veri ve davranışları bir araya getiren yapıdır. Nesneler, özellikler (properties) ve yöntemler (methods) içerebilirler. Özellikler, nesnenin veri parçalarını temsil ederken, yöntemler ise nesnenin davranışlarını belirtir.

Aşağıdaki gibi ***my_event*** adında bir nesne tanımlayalım. Bu nesneye ait ***name*** değişkeni ve ***printGuestList*** fonksiyonu bulunsun. Bu fonksiyon içerisinde nesneye ait olan name değişkenine ***this*** anahtar kelimesi ile erişebildiğimizi görüyoruz.

```
const my_event = {
    name:'xx baby Birthday Party',
    printGuestList: function(){
        console.log(this.name + ' İçin Katılımcı Listesi')
    }
}

my_event.printGuestList()
```

Şekil 22: Event Tanımlama

Yukarıdaki dosyayı çalıştırdığımızda aşağıdaki gibi bir çıktı alırız. This ile name değişkenine eriştiğimizi görürüz.

```
C:\Users\zuhal\Desktop\Node.Js>node arrow_deneme.js
xx baby Birthday Party İçin Katılımcı Listesi
```

Şekil 23: Event This Komutu

Fakat nesneye ait fonksiyonu arrow function ile tanımladığımızda name değişkenine erişilemediğini görürüz. Arrow fonksiyonları sadece tanımlandığı yerde bulunan değişkenlere erişebilir. Bu yüzden arrow function ve this kullanımına dikkat edilmelidir.

Aşağıdaki kodun çıktısına baktığımızda ***undefined*** hatası aldığımızı görürüz.

```
const my_event = {
    name:'xx baby Birthday Party',
    printGuestList: () => {
        console.log(this.name + ' İçin Katılımcı Listesi')
    }
}

my_event.printGuestList()
```

Şekil 24: Arrow This Komutu

Kodun çıktısı aşağıdaki gibidir.

```
C:\Users\zuhal\Desktop\Node.Js>node arrow_deneme.js
undefined İçin Katılımcı Listesi
```

Şekil 25: Arrow This Komutu Çıktısı

Nesne fonksiyonları kısa bir şekilde tanımlanmak istenirse aşağıdaki gibi bir yapı kullanılabilir. Bu şekilde hem this komutu ile nesne değişkenlerine erişilebilir hem de kısa bir fonksiyon tanımlaması yapılmış olur.

```
const my_event = {
    name:'xx baby Birthday Party',
    guestList: ['Ayşe', 'Mehmet', 'Ali'],
    printGuestList(){
        console.log(this.name + ' İçin Katılımcı Listesi')
        this.guestList.forEach((guest) => {
            console.log(guest + ' , ' + this.name + 'ne katılıyor')
        })
    }
}

my_event.printGuestList()
```

Şekil 26: Fonksiyon Tanımlama ve This Komutu

```
C:\Users\zuhal\Desktop\Node.Js>node arrow_deneme.js
xx baby Birthday Party İçin Katılımcı Listesi
Ayşe , xx baby Birthday Partyne katılıyor
Mehmet , xx baby Birthday Partyne katılıyor
Ali , xx baby Birthday Partyne katılıyor
```

Şekil 27: Komut Çıktısı