



Node.js ile Web Programlama Rapor - 2

Mahire Zühal Özdemir

İÇİNDEKİLER

ŞEKİLLER DİZİN	3
Modül Nedir?	4
CommonJS Nedir?	4
ECMAScript Nedir?	4
CommonJS Kullanımı	5
ECMAScript Kullanımı	5
Core Modülleri	7
1-FS Modülü:	7
2-OS Modülü:	7
3-HTTP Modülü:	8
4-Event Modülü:	8
package-lock.json Dosyası Nedir?	9
Komut Satırından Argüman Almak	9
Yargs Kütüphanesi	10
Yargs Kütüphanesi Command Komutu	12

ŞEKİLLER DİZİNİ

Şekil 1: math.js dosyası	5
Şekil 2:index.js dosyası	5
Şekil 3: index.js çıktısı	5
Şekil 4: package.json dosyası	6
Şekil 5: export işlemi	6
Şekil 6: index.js dosyası	6
Şekil 7: index.js dosyası çıktısı	6
Şekil 8: fs modülü	7
Şekil 9: Txt dosyası veri okuması	7
Şekil 10: Hostname öğrenme işlemi	8
Şekil 11: Hostname	8
Şekil 12: Http Modülü	8
Şekil 13: Events Modülü	8
Şekil 14: Process Argv İşlemi	9
Şekil 15: Argv Kullanımı	10
Şekil 16: Argv - Argüman Kullanımı	10
Şekil 17: Birden fazla argüman alma işlemi	10
Şekil 18: Argüman Alma İşlemi	11
Şekil 19: Komutlar ile if-else bloğu	11
Şekil 20: Yargs ile if-else bloğu	11
Şekil 21: Yargs Command Komutu	12
Şekil 22: demandOption seçeneği	13
Şekil 23: demandOption Komutu	13
Şekil 24: Komut Detayları	13
Şekil 25: Versiyon Bilgisi	14
Şekil 26: json_deneme dosyası	14
Şekil 27: Json İşlemleri	15
Şekil 28: deneme_json Dosyası	15

Modül Nedir?

JavaScript'te bir modül, bir veya daha fazla fonksiyon, sınıf veya değişken içerebilen bir dosyadır. Başka bir dosya içinden bu kod parçalarına erişmek için modül sistemi kullanılır. Modüller; JavaScript kodunu modüler parçalara böler ve bu parçalar arasında verimli bir şekilde bağımlılıklar oluşturmayı sağlar.

JavaScript'te modül sistemi, CommonJS veya ECMAScript Modülleri (ESM) ile uygulanabilir. İkisi de farklı söz dizimleri sağlar.

CommonJS Nedir?

- CommonJs; özellikle sunucu tarafında kullanılan modül sistemidir.
- ***require*** ve ***module.exports*** fonksiyonları kullanarak modüller arası bağımlılıkları yönetir.
- Senkronize I/O işlemlerini gerçekleştirir.
- Modülleri yüklemek için ***require*** komutu kullanılır.
- Değerleri export etmek için ***module.exports*** komutunu kullanır.

ECMAScript Nedir?

- ES6, tarayıcı ve sunucu tarafı JavaScript uygulamalarında kullanılabilir.
- ***import*** ve ***export*** komuları ile modüller arası bağımlılıkları yönetir.
- Asenkron I/O işlemlerini gerçekleştirir.
- Modülleri yüklemek için ***import*** komutunu kullanır.
- Değerleri içe aktarmak için ***export*** komutunu kullanır.

CommonJS Kullanımı

CommonJS; **require** ve **module.exports** komutlarını kullanır demiştik.

Math.js dosyası oluşturarak içerisine toplama işlemi yapan basit bir **add** fonksiyonu yazalım.

```
function add(x,y){  
  return x+y;  
}  
  
module.exports = {add};
```

Şekil 1: math.js dosyası

Ana dosyamız olan index.js üzerinden **math.js** dosyasını çağıralım ve toplama işlemi gerçekleştirelim.

```
const math = require('./math.js')  
console.log(math.add(5,2))
```

Şekil 2:index.js dosyası

Index.js dosyamızın çıktısı aşağıdaki gibi olur.

```
C:\Users\zuhal\Desktop>node index.js  
7
```

Şekil 3: index.js çıktısı

ECMAScript Kullanımı

ES6 modülünü kullanmak için ilk olarak projemizi initialize etmemiz gerekiyor.

npm init komutu ile projemizi başlatalım.

Initialize işlemi sonrasında package.json dosyamız oluşur.

Bu dosya içerisine **"type": "module"** satırı eklenir ve ES6 modülü aktif edilir.

```
sers > zuhal > Desktop > {} package.json > ...
{
  "name": "desktop",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "type": "module",
  "author": "",
  "license": "ISC"
}
```

Şekil 4: package.json dosyası

ES6 kullanarak import işlemini gerçekleştirmek için **math.js** dosyamız içerisinde add fonksiyonunu aşağıdaki gibi oluşturalım.

```
export function add(x,y){
  return x+y;
}
```

Şekil 5: export işlemi

Daha sonra index.js dosyamızda import komutu ile math.js dosyamızı çağıralım ve add fonksiyonumuzu çalıştıralım.

```
import {add} from './math.js';
console.log(add(5,7));
```

Şekil 6: index.js dosyası

Komut çıktımız aşağıdaki gibi olacaktır.

```
C:\Users\zuhal\Desktop>node index.js
12
```

Şekil 7: index.js dosyası çıktısı

Core Modülleri

Node.js platformunun bir parçası olan ve Node.js kurulumu ile birlikte gelen birçok yerleşik modülü bulunmaktadır. Bu modüller, **require** fonksiyonu kullanılarak programa yüklenir.

Bu modüllerden 4 tanesi aşağıdaki gibidir:

1-FS Modülü:

Fs (file system) modülü, Node.js'in dosya sistemi işlemleri için sağladığı yerleşik bir modüldür. Bu modül, dosyaları oluşturma, okuma, yazma, güncelleme, silme ve diğer dosya işlemlerini gerçekleştirmek için kullanılır.

ES6 yerine CommonJs kullanarak **require** ile fs modülünü çağırarak çalıştırabiliriz. core_modules.js dosyasını aşağıdaki gibi oluşturduktan sonra aynı klasör içerisinde bulunan sample.txt dosyasının içeriğini görüntüleyebiliriz.

```
const fs = require('fs')
fs.readFile('sample.txt', 'utf-8', (err,data) => {
  if (err) throw err;
  console.log(data);
})
```

Şekil 8: fs modülü

```
C:\Users\zuhal\Desktop>node core_modules.js
this file is just a sample txt file
```

Şekil 9: Txt dosyası veri okuması

2-OS Modülü:

Os (Operating System) modülü, Node.js'in işletim sistemiyle ilgili bilgilere erişim sağlayan yerleşik bir modüldür. Bu modül, işletim sistemi hakkında bilgiler almak ve işletim sistemiyle etkileşimde bulunmak için bir dizi fonksiyon içerir.

```
const os = require('os');
console.log(`hostname: ${os.hostname()}`);
```

Şekil 10: Hostname öğrenme işlemi

```
C:\Users\zuhal\Desktop>node core_modules.js
hostname: DESKTOP-
```

Şekil 11: Hostname

3-HTTP Modülü:

Http modülü, Node.js'in HTTP sunucuları oluşturmak ve HTTP istekleri göndermek için sağladığı yerleşik bir modüldür. Bu modül, HTTP protokolü üzerinde iletişim kurmak için bir dizi fonksiyon içerir.

```
const http = require('http')
const server = http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
console.log(server.address)
```

Şekil 12: Http Modülü

4-Event Modülü:

event modülü, olay işleme (event handling) için temel altyapı sağlayan yerleşik bir Node.js modülüdür. Bu modül, olayların tanımlanması, tetiklenmesi ve dinlenmesi için bir mekanizma sunar. Node.js, asenkron ve olaya dayalı bir mimariye sahip olduğundan, bu tür bir olay işleme mekanizması oldukça önemlidir.

```
const events = require('events')
const EventEmitter = new events.EventEmitter();
```

Şekil 13: Events Modülü

package-lock.json Dosyası Nedir?

package-lock.json dosyası, Node.js projesinde kullanılan paketlerin bağımlılıklarını ve bağımlılıkların sürüm bilgilerini içeren bir JSON dosyasıdır. Bu dosya, proje bağımlılıklarının belirli sürümlerini kilitlemek ve tekrarlanabilir bir kurulum sağlamak için kullanılır.

Yeni bir proje oluşturulduğunda veya bağımlılıklarda bir güncelleme yapıldığında, Node.js, package-lock.json dosyasını günceller ve projede kullanılan paketlerin tam sürüm bilgilerini içerir.

Package-lock.json içerisinde dependencies bulunur. Dependencies; ana bağımlılıkların altında yüklü olan alt bağımlılıkların listesini bulundurur.

Komut Satırından Argüman Almak

process.argv, Node.js ortamında çalışan bir programın komut satırı argümanlarını içeren bir diziye işaret eden global değişkendir. Bu dizi, programın çalıştırıldığı komut satırıyla ilişkili argümanları içerir.

```
console.log(process.argv);
```

Şekil 14: Process Argv İşlemi

index.js dosyamız içerisinde yukarıdaki komutları çalıştırdıktan sonra; aşağıdaki gibi çıktı elde ederiz.

Çıktıya baktığımızda ilk satır; executable dosya bilgisini [0],

İkinci satır; executable dosyanın head bilgisini [1],

3.satır ve sonrası alınan argümanları içerir [2], [3],

Aşağıdaki örnekte argüman girilmeden dosyayı çağırdığımızda 2 satırdan oluşan dosya bilgilerini döndürür.

```
C:\Users\zuhal\Desktop>node app.js
[
  'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\zuhal\\Desktop\\app.js'
]
```

Şekil 15: Argv Kullanımı

Aşağıdaki örnekte argüman girilerek dosyayı çağırdığımızda 2 satır ve ek olarak girilen argüman sayısı kadar satırdan oluşur.

```
C:\Users\zuhal\Desktop>node app.js zuhal
[
  'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\zuhal\\Desktop\\app.js',
  'zuhal'
]
```

Şekil 16: Argv - Argüman Kullanımı

Yargs Kütüphanesi

Yargs kütüphanesi; interaktif komut satırı aracıdır. Bu kütüphane; komut satırı argümanlarını analiz etmek, işlemek için kullanılır. Komut satırı arayüzü oluşturmayı sağlar.

Yargs kütüphanesi ile komut satırından argüman almak için ***process.argv*** komutu kullanılır.

Alınan argümanlar değişken içerisinde tutularak yazılır.

```
const yargs = require('yargs') //import yargs
yargs.version('1.1.0') //adding library version

const arg1 = process.argv[2]; //take a argument
const arg2 = process.argv[3]

console.log('Argument-1:', arg1)
console.log('Argument-2:', arg2)
```

Şekil 17: Birden fazla argüman alma işlemi

Yukarıda içeriği verilen app.js dosyasını argümanla birlikte çağırarak içeriğini yazdırabiliriz.

```
C:\Users\zuhal\Desktop>node app.js zuhal özdemir
Argument-1: zuhal
Argument-2: özdemir
```

Şekil 18: Argüman Alma İşlemi

Alınan değişkenler ile basit bir not uygulaması yapalım. Girdilerin değerlerine göre if-else bloğu oluşturarak hangi işlemlerin yapılacağını ekrana yazdırabiliriz.

!!! Üç eşittir kullanarak hem değer hem tip kontrolü yapılır.

```
const command = process.argv[2]

if (command === 'add'){
  console.log('adding new note')
}
else if (command === 'remove'){
  console.log('delete the note')
}
else if (command === 'list'){
  console.log('list the notes')
}
else{
  console.log('error argument')
}
```

Şekil 19: Komutlar ile if-else bloğu

App.js dosyasını farklı girdiler ile çalıştırarak if-else bloğunun çalışıp çalışmadığının kontrolünü yapabiliriz.

```
C:\Users\zuhal\Desktop>node app.js add
adding new note

C:\Users\zuhal\Desktop>node app.js remove
delete the note

C:\Users\zuhal\Desktop>node app.js list
list the notes
```

Şekil 20: Yargs ile if-else bloğu

Yargs Kütüphanesi Command Komutu

yargs.command() yargs kütüphanesinin bir yöntemidir, yeni bir komut tanımlamak için kullanılır. Bu komut, kullanıcının komut satırında belirli bir işlemi gerçekleştirmek için kullanabileceği bir komuttur. Her komut, bir komut adı (command), bir açıklama (describe), bir argüman yapısı (builder) ve bir işlem (handler) içerir.

!!! yargs command'in çalışması için dosya içerisine yargs.parse() satırı eklenir.

App.js dosyası içerisinde komut adını **command**,

komut açıklamasını **describe** ile tanımlarız.

Komutun yapısını tanımlamak için **builder** kullanılır. Builder içerisinde komutun yapısında kullanılan **title** ve **body** seçenekleri bulunur.

Komutun zorunlu olup olmadığını belirtmek için **demandOption** seçeneği ayarlanır.

```
yargs.command({
  command: 'add', //command
  describe: 'adding new note', //command description
  builder: {
    title: {
      describe: 'Note Title',
      demandOption: true,
      type: 'string',
    },
    body: {
      describe: 'Note Content',
      demandOption: true,
      type: 'string',
    }
  },
  handler: function (argv) {
    console.log('Title: ', argv.title);
    console.log('Body: ', argv.body);
  },
})
```

Şekil 21: Yargs Command Komutu

Title ve body değerlerinde **demandOption** seçeneğini true belirlediğimiz için title ve body olmadan girdilerde hata alırız.

```
C:\Users\zuhal\Desktop>node app.js add
app.js add

adding new note

Options:
  --help      Show help                                [boolean]
  --version   Show version number                      [boolean]
  --title     Note Title                                [string] [required]
  --body      Note Content                              [string] [required]

Missing required arguments: title, body
```

Şekil 22: demandOption seçeneği

Title ve Body değerlerini doldurarak aşağıdaki gibi add komutu çağrılabilir.

```
C:\Users\zuhal\Desktop>node app.js add --title="Title-1" --body="Body-1"
Title:  Title-1
Body:   Body-1
```

Şekil 23: demandOption Komutu

Komut detaylarına ulaşmak için **help** anahtar sözcüğü aşağıdaki gibi kullanılır.

```
C:\Users\zuhal\Desktop>node app.js add --help
app.js add

adding new note Describe

Options:
  --help      Show help                                [boolean]
  --version   Show version number                      [boolean]
  --title     Note Title                                [string] [required]
  --body      Note Content                              [string] [required]
               Title Describe
               Body Describe
```

Şekil 24: Komut Detayları

Komut versiyon bilgisine erişmek için **--versiyon** terimi kullanılır.

```
C:\Users\zuhal\Desktop>node app.js add --version 1.1.0
```

Şekil 25: Versiyon Bilgisi

json_deneme.js adında yeni bir dosya oluşturalım. Bu dosya içerisinde nesne oluşturup yazdıralım.

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

const book = {
  title: 'Note Title',
  author: 'Author of note'
}

const bookJSON = JSON.stringify(book)

fs.writeFileSync('denemeJson.json', bookJSON)
const dataBuffer = fs.readFileSync('denemeJson.json')
const dataJSON = dataBuffer.toString()
console.log(dataJSON)
const data = JSON.parse(dataJSON)
console.log(data)
```

Şekil 26: json_deneme dosyası

İlk olarak **title** ve **author** dan oluşan bir book nesnesi oluşturalım.

Oluşturulan book nesnesini **stringify** ile JSON formatına dönüştürelim.

Fs modülü ile denemeJson.js dosyası oluşturalım. Bu dosya içerisine Json verimizi yazdırıyoruz.

Oluşturulan dosyayı fs modülü ile okuyoruz.

Okunan veriyi string'e dönüştürüp ekrana yazdırıyoruz.

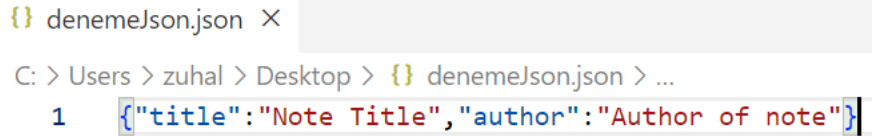
Veriyi JS nesnesine dönüştürmek için **JSON.parse** kullanıyoruz.

Dosyayı çalıştırdıktan sonra çıktı aşağıdaki gibi olur.

```
C:\Users\zuhal\Desktop>node json_deneme.js
{"title":"Note Title","author":"Author of note"}
{ title: 'Note Title', author: 'Author of note' }
```

Şekil 27: Json İşlemleri

Deneme_json dosyası içeriği aşağıdaki gibi olur.



```
{ } denemeJson.json ×
C: > Users > zuhal > Desktop > { } denemeJson.json > ...
1  { "title": "Note Title", "author": "Author of note" }
```

Şekil 28: deneme_json Dosyası