Açık Kaynak Yazılım Teknolojileri

Öğr.Gör. M.Ersin AKAY

2020

Açık Kaynak Yazılım Teknolojileri

Konular

- Node.js
- Npm Paket Yönetimi
- Express

Açık Kaynak Yazılım Teknolojileri

Node.js

- 2009 yılında Ryan Dahl tarafından Google Chrome tarayıcısının JavaScript komutlarını çalıştırmak için kullandığı V8 JavaScript motoruna çeşitli eklemeler yaparak JavaScript'in sunucu tarafında çalışması için geliştirilmiştir.
- V8 motoru C/C++ ile geliştirilmiş JavaScript komutlarını makine diline çevirmek için kullanılan bir ara yazılımdır.
- Komutların makine koduna çevrilmesi JavaScript komutlarının daha hızlı ve performanslı çalışmasını sağlar.

Açık Kaynak Yazılım Teknolojileri

Node.js

- Node.js popüler olmasının sebebi hızlı ve performanslı olmasının yanında
  - JavaScript komutlarının esnek oluşu,
  - Komutların bloklanmadan işlenmesi,
  - Olay tabanlı çalışması,
  - Diğer sunucu taraflı çalışan programlama dilleri gibi ek bir web sunucusuna (Apache HTTP, IIS, Nginx vb.) ihtiyaç duymamasıdır.

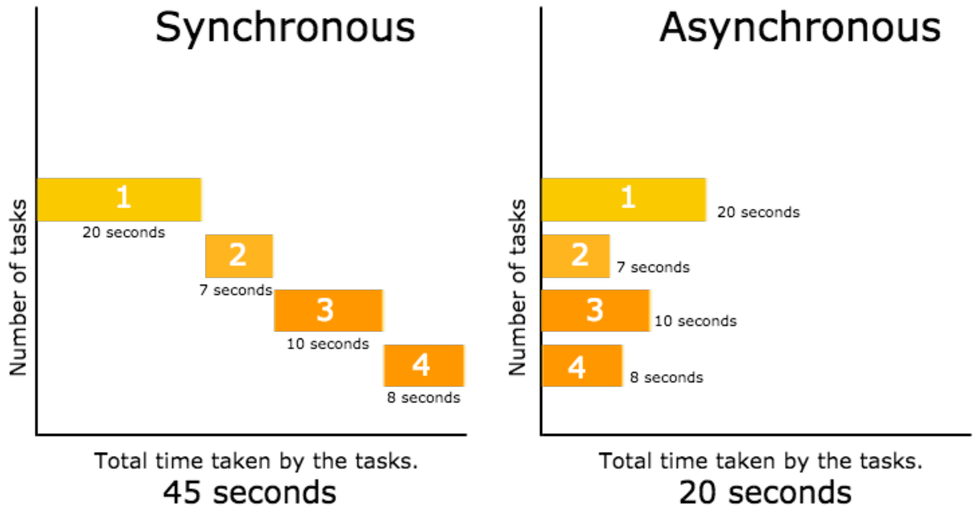
Açık Kaynak Yazılım Teknolojileri

Node.js

- Diğer sunucu taraflı çalışan programlama dilleri (PHP, ASP.NET vb.) ile yazılmış uygulamalar web sunucu (Apache HTTP, IIS, Nginx vb.) denilen istemci ve sunucu arasında bağlantıyı kuran ek yazılımlara ihtiyaç duyar.
- Web sunucusuna gelen istekler daha sonra sunucu taraflı çalışan programlama dillerine iletilir ve istenilen komutlar çalıştırılır.
- Node.js içerisinde gelen çekirdek modülleri sayesinde ek bir web sunucusuna ihtiyaç duymadan komutların çalıştırılmasını sağlar.

Açık Kaynak Yazılım Teknolojileri

Node.js - Eşzamansız (Asynchronous) ve Olay Tabanlı



- Eşzamanlı programlamada, bir işlem başlattığınızda tüm işlem bitinceye kadar diğer istekler gönderilemez. Örneğin, kullanıcı bir dosya yüklemeye başladığında bu dosyanın sonuna kadar yüklenmesini beklemek zorundadır.

- Eşzamansız programlamada ise birden fazla işlem başlatılabilir. Buna eşzamansız veya engellenmemiş (non-blocking) programlama denir. İstemci sunucudan istekte bulunduğunda bir olay (event) bildirim mekanizması önceki istekten bir yanıt almasına yardımcı olur

Açık Kaynak Yazılım Teknolojileri

Node.js

- Bununla ilgili en güzel örnek yemek sipariş sistemi diyebiliriz.
- Klasik sunucu taraflı programlama dilleri bir yemek siparişi geldiğinde sırada duran diğer müşteriler siparişin hazırlanmasını bekler.
- Node.js ise herhangi bir yemek siparişi geldiğinde siparişi arka tarafa bildirir ve not alır ve daha sonra sıradaki müşterinin siparişini alır.
- Verilen yemek siparişlerinin hangisi daha önce hazırlanırsa o yemek siparişine cevap verir.
- Böylece müşteri daha önce ve hızlı hazırlanacak bir yemek için sırada fazladan beklemez.
- Bloklamadan işlem yapılmasına non-blocking I/O, ölçeklenebilir uygulama geliştirme gibi çeşitli isimler verilmiştir.
- Bu yapı sayesinde anlık mesajlaşma, oyun sistemleri gibi gerçek zamanlı uygulamalar kolaylıkla ve daha az maliyetle yapılır.

Node.js

Kurulum

- Resmi internet sitesi www.nodejs.org adresinden işletim sisteminize göre önerilen kurulum dosyasını indirip kurabilirsiniz

- Kurulumu gerçekleştirdikten sonra

  ```
  node –v
  ```

  komutuyla kurulan Node.js'in hem versiyonunu öğrenebilir hem de böylece çalışmasını test edebilirsiniz.

Node.js

Merhaba Dünya

- Javascript kodumuzu ders1.js dosyasına yazalım
- İlk kodumuz konsola “Merhaba Dünya” yazsın

```
console.log(“Merhaba Dünya”);
```

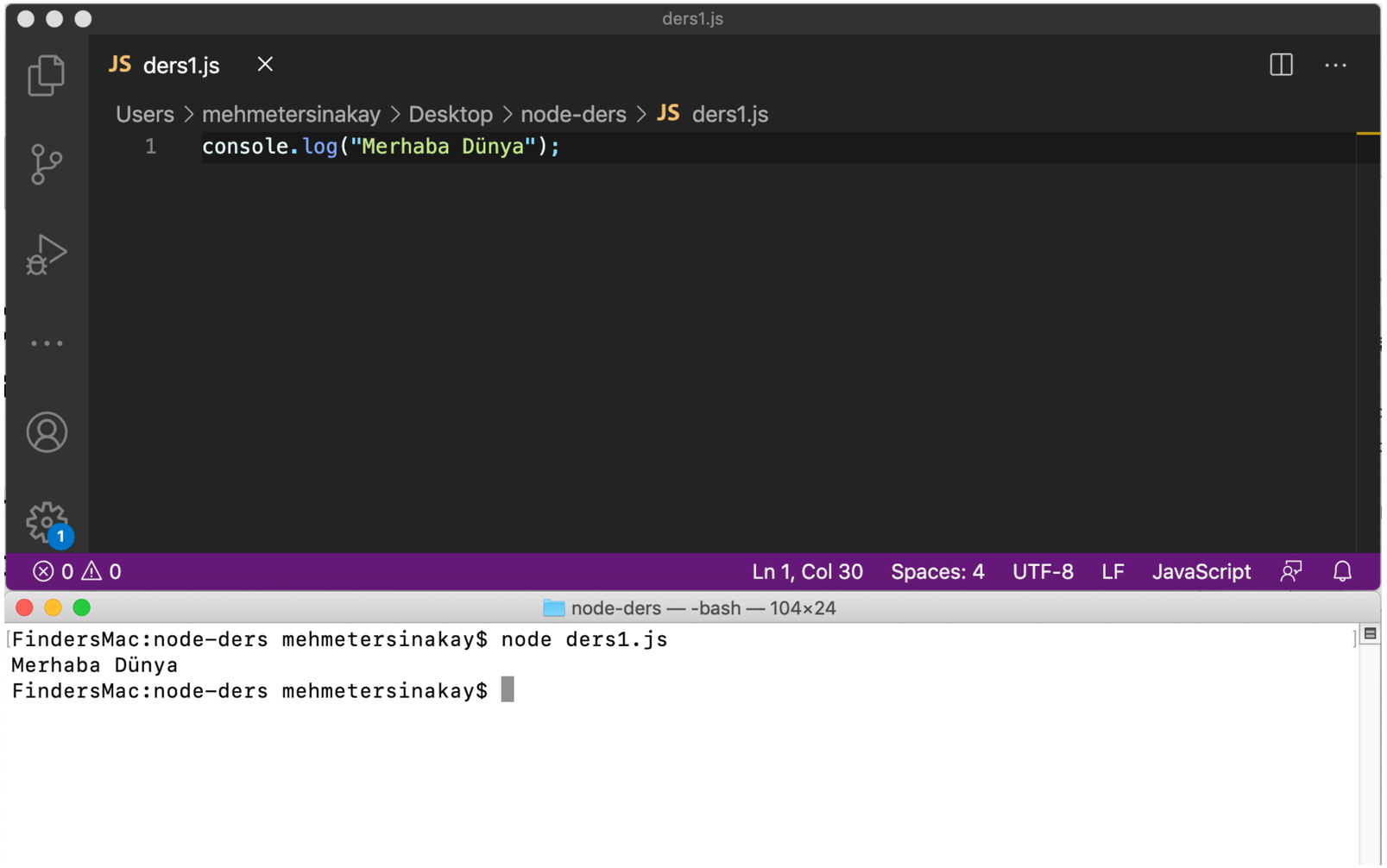
- Konsolda (Windows Komut Satırı) ders1.js dosyamızın bulunduğu dizine giderek

```
node ders1.js
```

komutunu çalıştıralım, konsol ekranında çıktı olarak “Merhaba Dünya” yazacaktır

Node.js

Merhaba Dünya



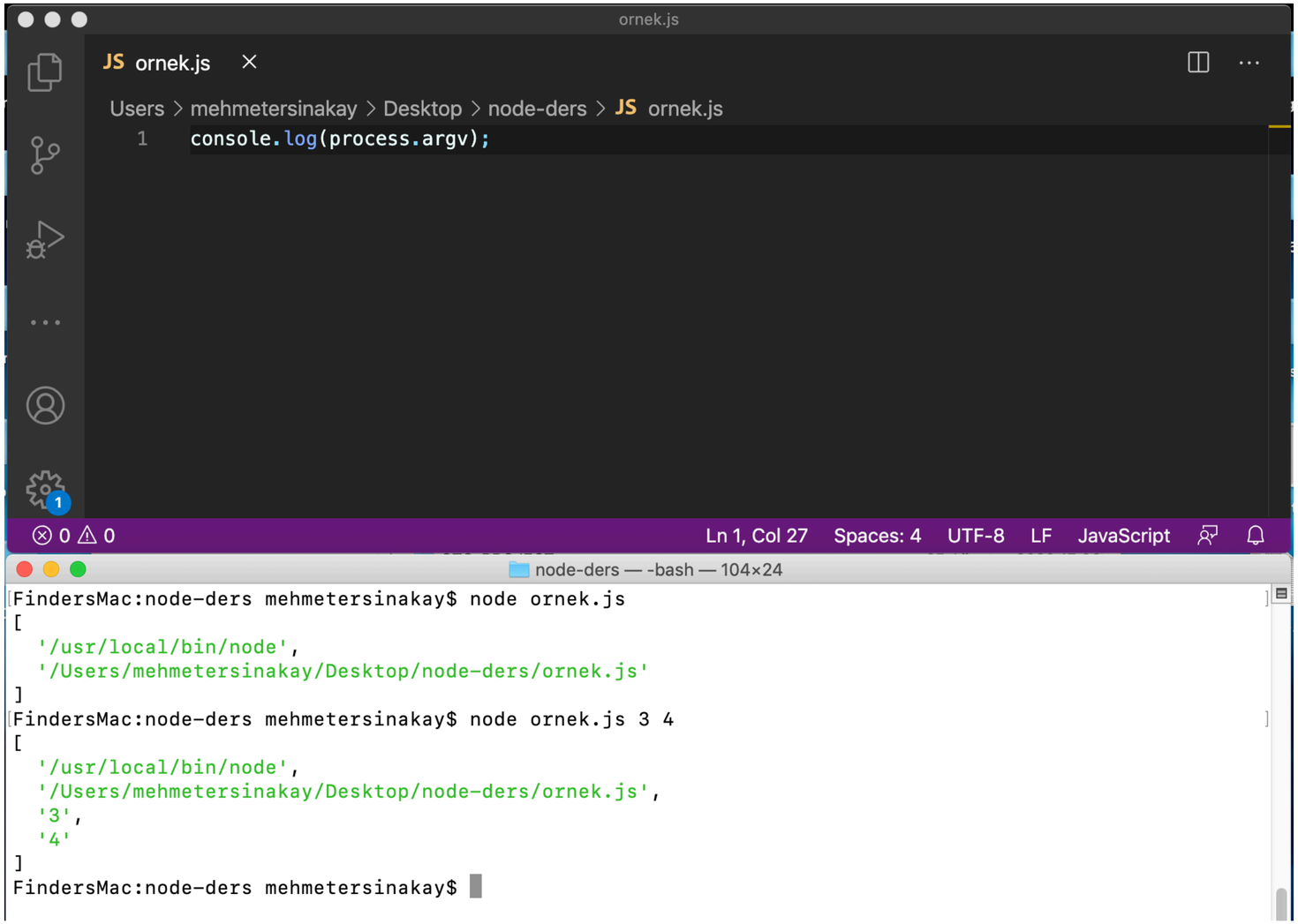
Node.js

Parametre Kullanımı

- nodejs sisteminde global olarak kullanabileceğimiz modüller bulunmaktadır
- Bunlardan biri de **process** modulüdür. Bu modülle o anki nodejs prosesi hakkında bilgi alabilir ve kontrol edebiliriz
- Konsolda program çalıştırırken yollanan parametrelere de process sınıfının argv değişkeni ile ulaşabiliriz.
- Bu değişken bize liste halinde girilen bütün parametreleri gönderir. (Konsolda girilen parametreler birbirlerinden boşluklarla ayrılır.)
- Bu listenin ilk elemanı **nodejs'in çalıştığı yeri**, ikinci parametre **kodlarımızın çalıştığı yeri** sonraki parametreler de konsol ekranından girilen parametreleri verir.

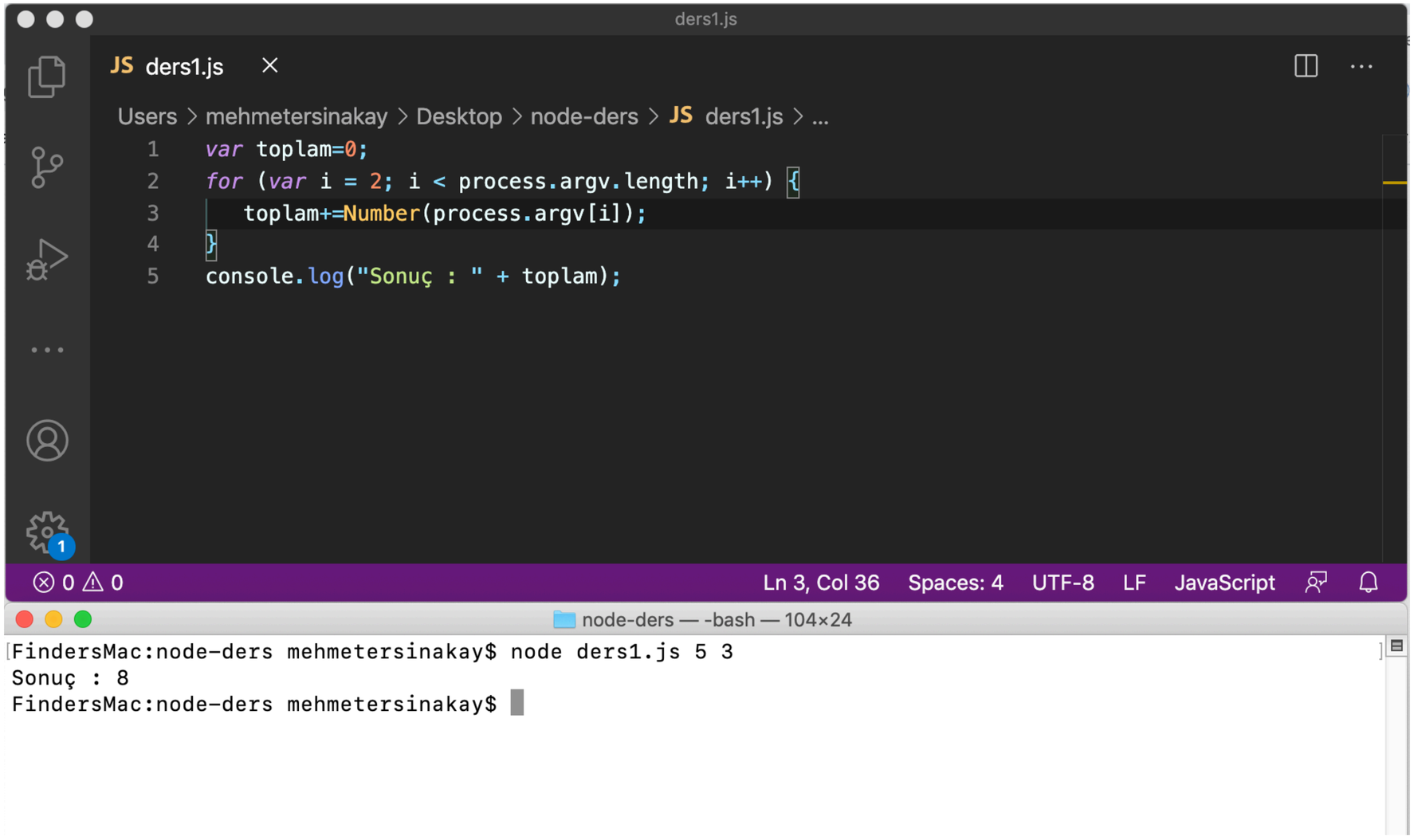
Node.js

Parametre Kullanımı



Node.js

Parametre Kullanımı - Toplama Örneği



Node.js

I/O (Input/Output)

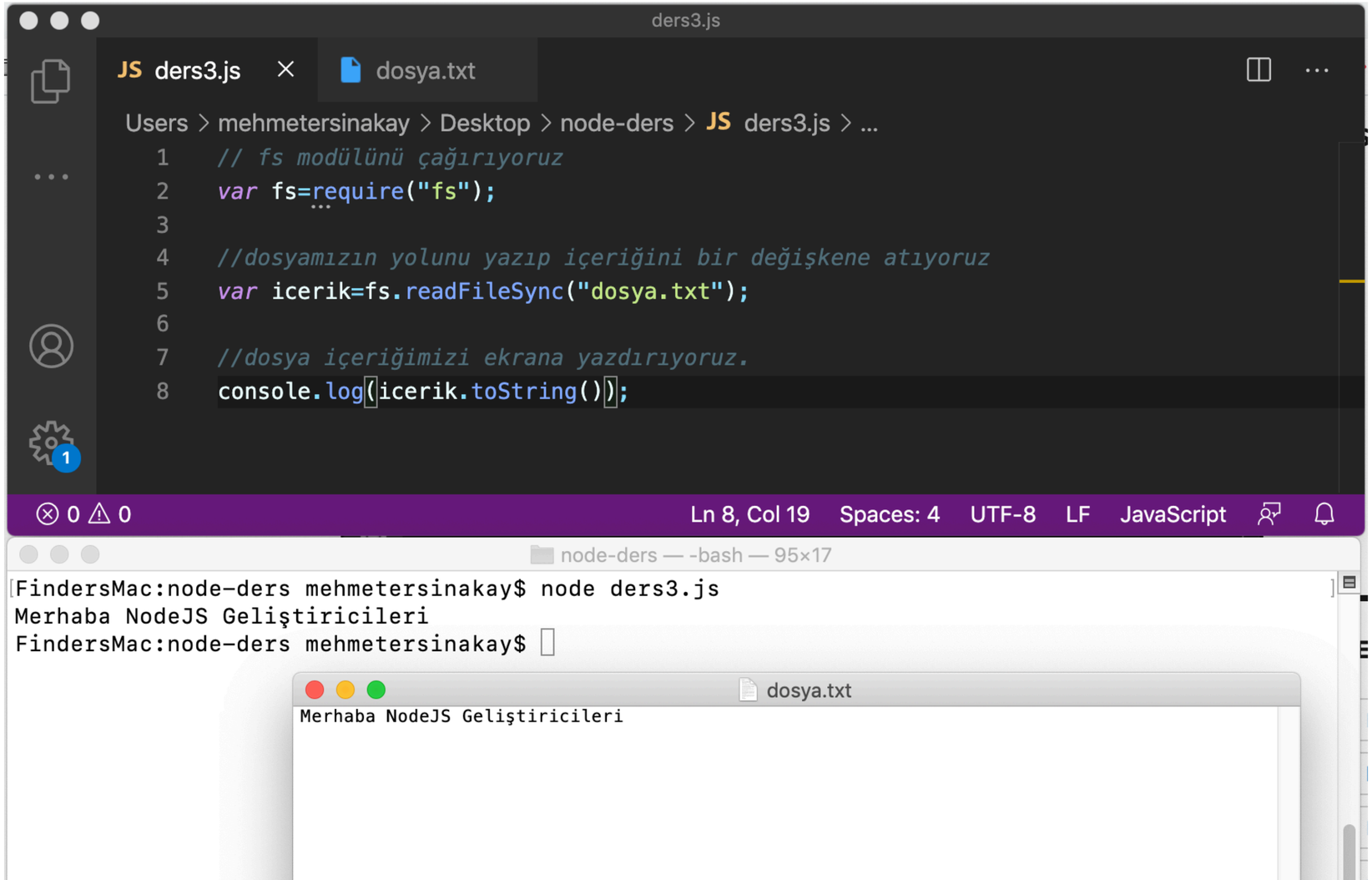
- Nodejs’in güçlü olduğu yanlardan bir taneside I/O işlemleridir
- Global modüllerin yanısıra dışarıdan çağırdığımız modüller de mevcuttur

```
var fs = require(“fs”);
```

- I/O işlemleri için nodejs içerisinde fs(file system) modülü kullanılmaktadır. Dosya okuma, dosya yazma

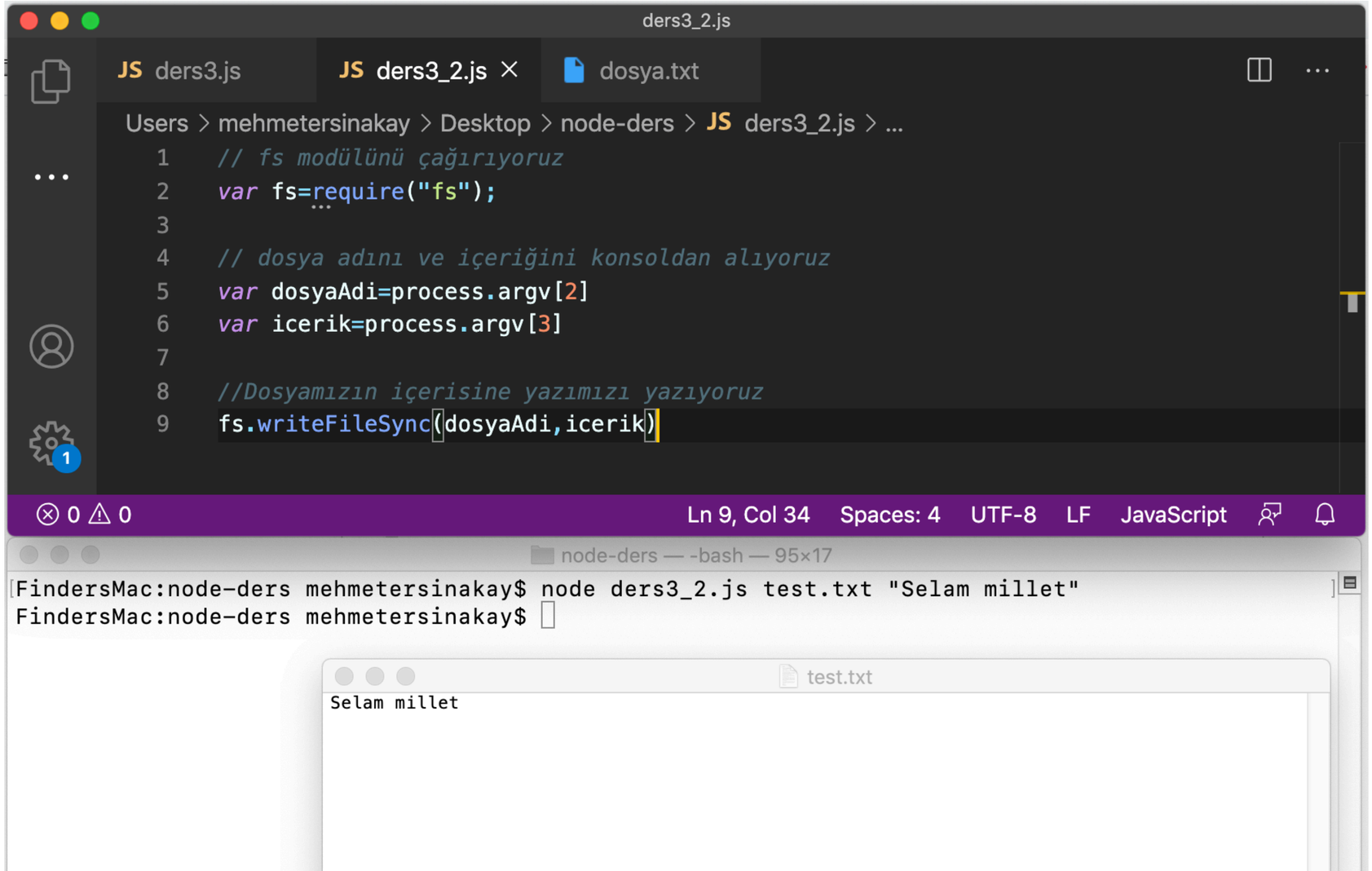
Node.js

I/O (Input/Output) - readFileSync



Node.js

I/O (Input/Output) - writeFileSync



Node.js

Senkron Programlama

- Yazdığımız programların çoğu kodların yazılış sırasına göre yukarıdan aşağıya doğru işleyerek ilerler
- Her şey sıra ile yapılır
- Global modüllerin yanısıra dışarıdan çağırdığımız modüller de mevcuttur

```
var fs=require("fs");
var dosyaIcerigi=fs.readFileSync(process.argv[2]);
console.log(“Program devam ediyor...”);
var icerik=dosyaIcerigi.toString();
console.log(icerik);
```

- Hiçbir işlem birbirinin önüne geçemez. Ve birbirini beklemek zorundadır. Sırayla, senkron bir şekilde işlemek zorundadır. Bu tip programlama metoduna **Senkron Programlama** denir

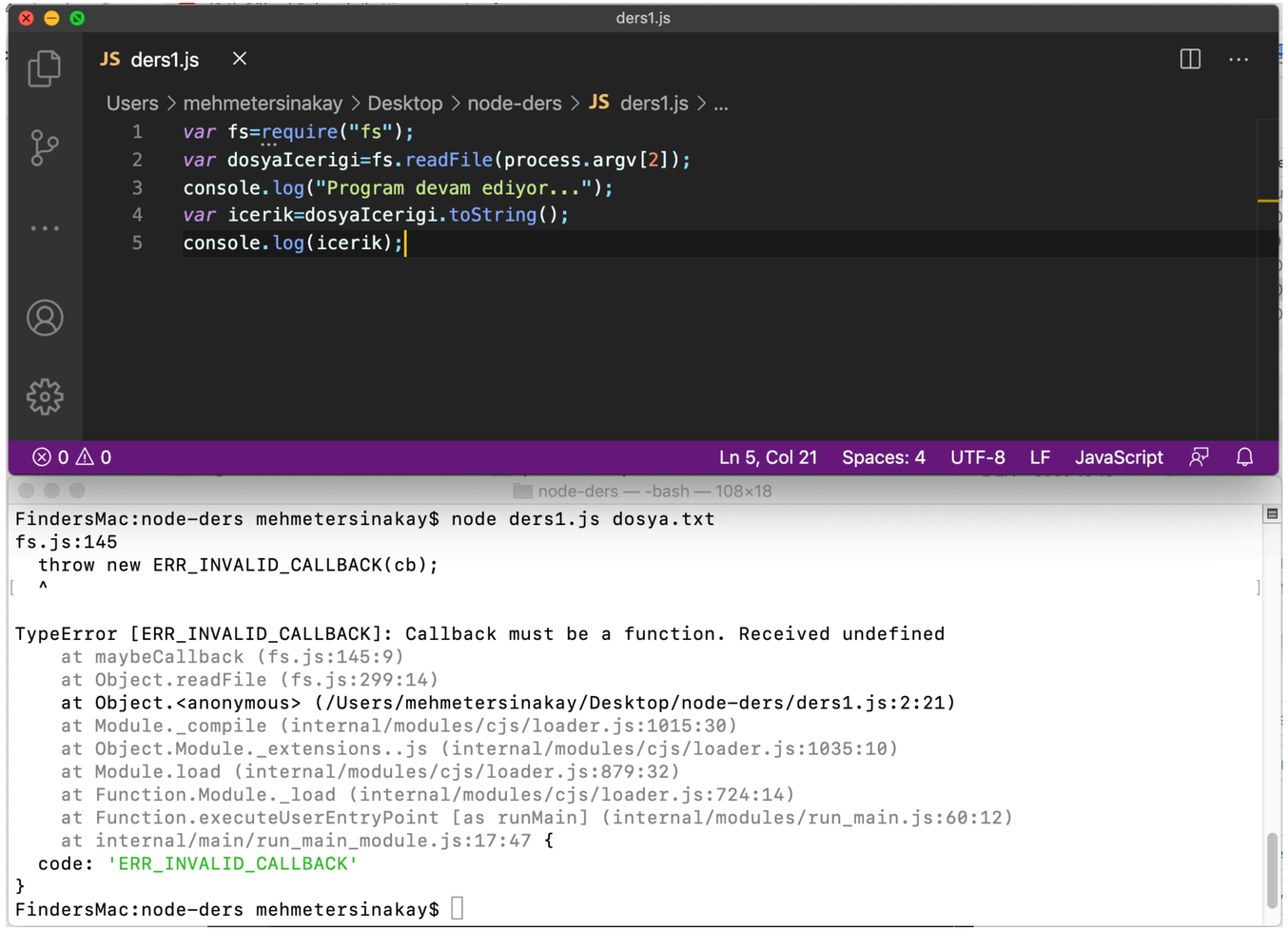
Node.js

Asenkron Programlama

- Senkron programlamadaki her satırın sırayla çalışması ve her bir işlemin birbirini beklemesi yeri geldiğinde programımızı çok yavaşlatabilir, hatta işlem bitene kadar durdurabilir.
- Örneğin ekrana “Program devam ediyor…” yazdırmak için bir önceki işlemin bitmesini beklemek gerekir, dosya içeriği çok büyükse bu işlemler dakikalar bile alabilir.
- İşte bu tip durumlar için asenkron fonksiyonlar kullanırız. Kod akışının sırayla çalışmadığı, işlemlerin birbirini beklemediği, kod akışının işlem durumlarına göre devam ettiği programlamaya **Asenkron Programlama** denir.

Node.js

Asenkron Programlama



- Asenkron işlemler için modüller içerisinde asenkron fonksiyonlar vardır.
- Örneğin fs modülü içerisinde kullanmış olduğumuz readFileSync metodunun asenkron versiyonu “readFile” metodudur. Örneğimizi bu metot ile yeniden yazalım.
- Bu şekilde kodlarımızı asenkron yapmış olduk. Artık 2. satırdaki işlem çalışırken 3,4 ve 5 satırlar onun sonucunu beklemeden devam eder. Ancak kodlarımız hata verecek. Neden?
- Çünkü biz 4. satırda 2. satırdaki kodun sonucunu kullanıyoruz. Ama daha 2. satırdaki kodun işi bitmedi. (Eğer okunan dosya çok çok küçükse küçük bir ihtimal bitmiş olabilir.)
- Peki biz ikinci satırdaki kodun işinin bittiğini nereden anlayacağız? Bittikten sonra yapmamız gerekenleri nerede belirteceğiz? **Callback** fonksiyonunda.

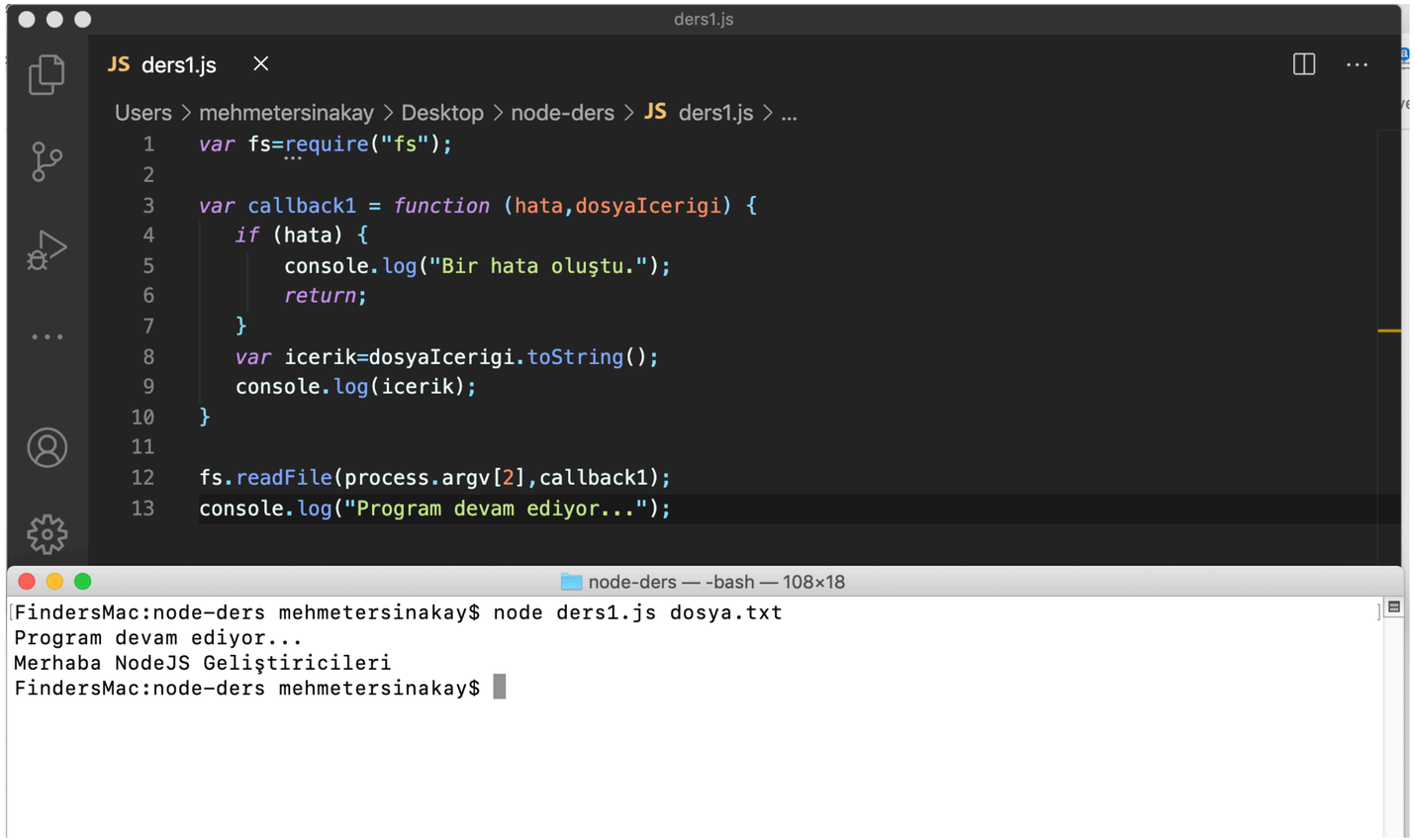
Node.js

Callback fonksiyonu

- Asenkron fonksiyonlar içerisindeki işlemler bittikten hemen sonra yapılacakların belirlendiği metotlara callback metotları denir.
- Bu metotlar asenkron metotlara parametre olarak verilir. Sadece bitiminde değil herhangi bir hata sonucunda da callback fonksiyonları kullanılabilir. Bu tamamen asenkron metodun yapısıyla alakalıdır.
- Nodejs içerisindeki callback fonksiyonlarında genellikle en az iki tane parametre belirlenir. Biri hata durumunu kontrol etmek için, diğeri fonksiyon sonucunu almak için kullanılır.
- Callback fonksiyonunun kaç tane parametre aldığını, yapısının nasıl olduğunu, hangisi sıra ile yazılacağını … vs bilgileri kullanacağımız modül ve fonksiyonun dokümantasyonundan öğrenebiliriz

Node.js

Callback fonksiyonu Örnek-1



- İlk satırda fs modülünü çağırdık.
- 12. satırda parametre olarak yolladığımız dosyanın içeriğini **okumaya başladık**.
- 12. satırdaki kodumuz devam ederken 13. satırda ekrana “Program devam ediyor…” yazdık.
- Diğer taraftan 12. satırdaki kod ne zaman biterse callback1 fonksiyonunu çağıracak.
- callback1 fonksiyonu içerisinde de ilk olarak hata olup olmadığını kontrol ettik. Varsa konsol ekranına “Bir hata oluştu.” yazdırdık
- Hata yoksa dosyaIcerigi degiskenini string ifadeye çevirdik. Daha sonrada içeriği ekrana yazdırdık.

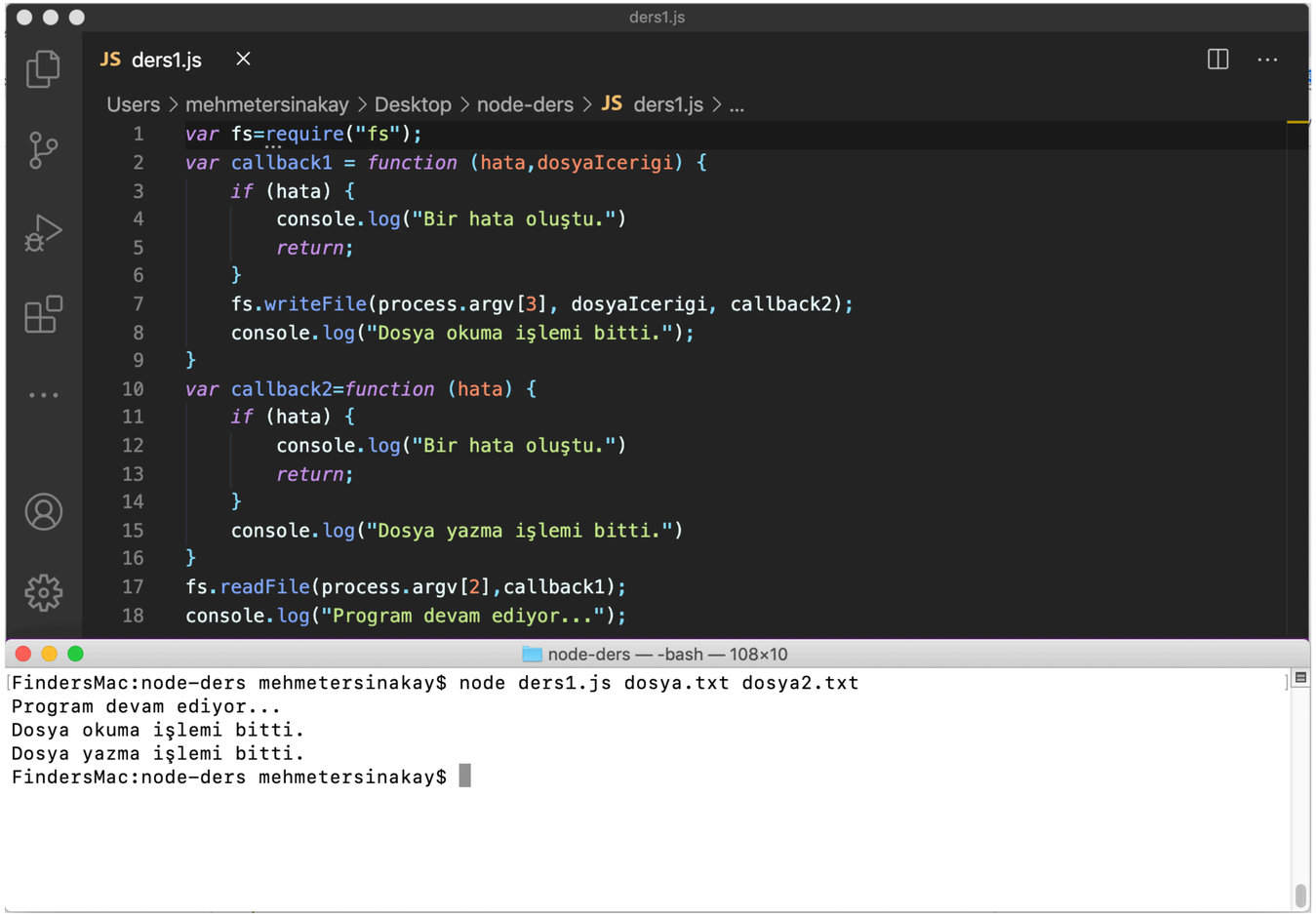
Node.js

Callback fonksiyonu Örnek-2

- Dosyanın birinden okuduğumuz içeriği ikinci bir dosyaya yazacağız.
- Dosya isminide parametre olarak alacağız.
- Okuma işlemi için daha önce kullandığımız fs modülününün readFile fonksiyonunu, yazma işlemi içinde fs modülünün writeFile fonksiyonunu kullanacağız.
- writeFile fonksiyonu da readFile gibi asenkron bir fonksiyon. (Daha önce kullandığımız writeFileSync fonksiyonunun asenkron versiyonu.)
- Hem okuma hem yazma işlemlerimiz asenkron olacak

Node.js

Callback fonksiyonu Örnek-2



- İlk olarak fs modülünü çağırıyoruz.
- Sonra callback1 ve callback2 adında iki tane fonksiyon tanımlıyoruz.
- Daha sonra (17. satırda) dosya okuma işlemine **başlanır**. Sonra ekrana “Program devam ediyor…” yazdırdık. Dosya okuma işlemi bittikten sonra callback1 fonksiyonu çalışır. callback1 içerisinde öncelikle hata kontrolü yapılır.
- Hata yoksa writeFile fonksiyonu ile yazma işlemine **başlanır**. Sonrasında konsol ekranına “Dosya okuma işlemi bitti.” yazdırılır.
- Yazma işlemi bittikten sonra ise callback2 fonksiyonu çalışır. callback2 içerisinde de ilk olarak hata kontrolü yapılır. Daha sonra konsol ekranına “””Dosya yazma işlemi bitti.” yazdırılır.

Node.js

Modüler Programlama

- Yazdığımız kodları yönetilebilir hale getirmek, kod karmaşasını ortadan kaldırmak ve tekrar kullanılabilir kodlar oluşturmak için mantıksal olarak aynı işleri yapan kod veya dosyaların bir araya getirilerek oluşturulduğu, tek başına çalışamayan ana programlar tarafından kullanılarak çalıştırılan program parçalarına modül(yada kütüphane yada paket) denir

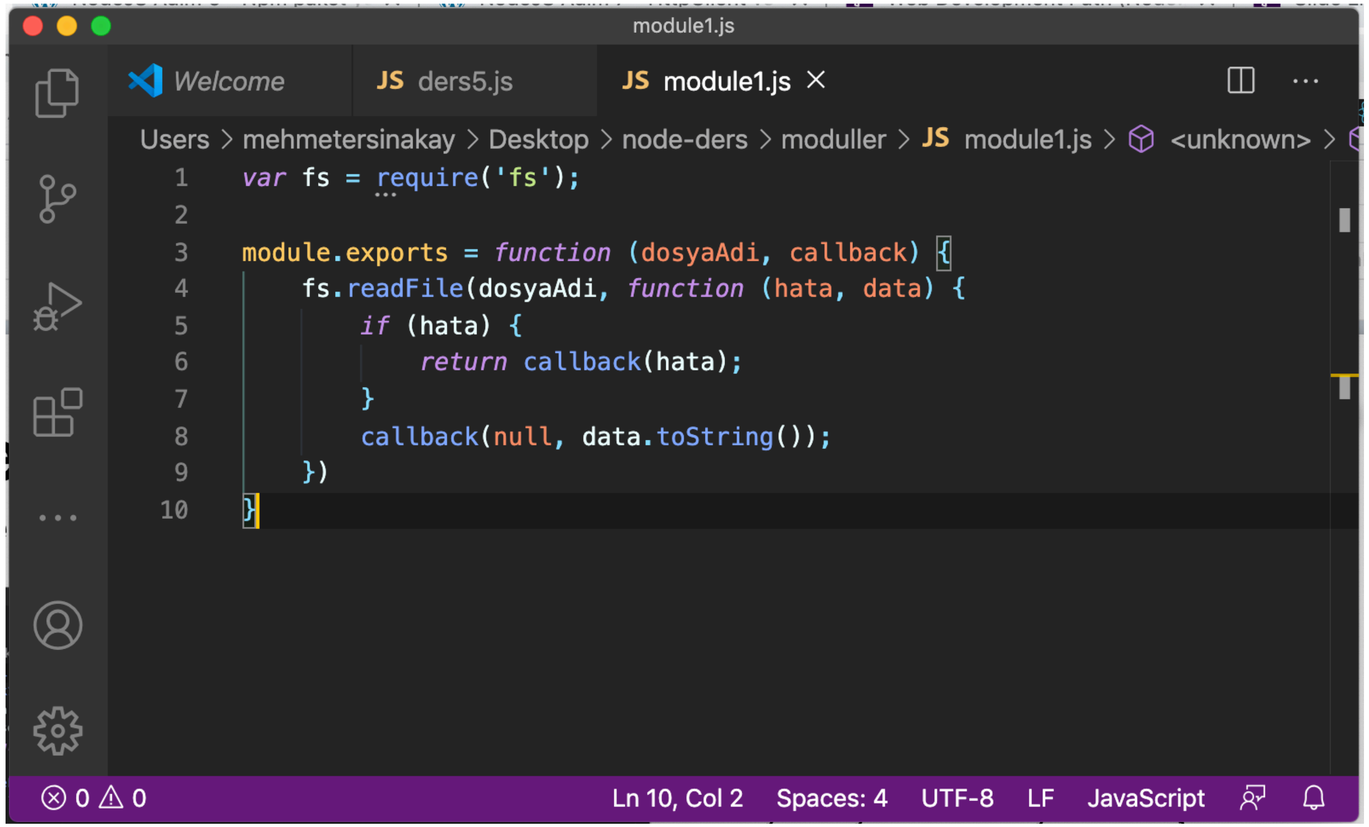
- nodejs üzerinde kullanılan bir çok modül bulunmaktadır.

  - Bunlardan bazıları nodejs çalışırken otomatik yüklenir(daha önce kullandığımız process gibi)

  - bazılarını da kodlama sırasında kodlar yardımı ile biz yükleriz(daha önce kullandığımız fs modülü gibi).

Node.js

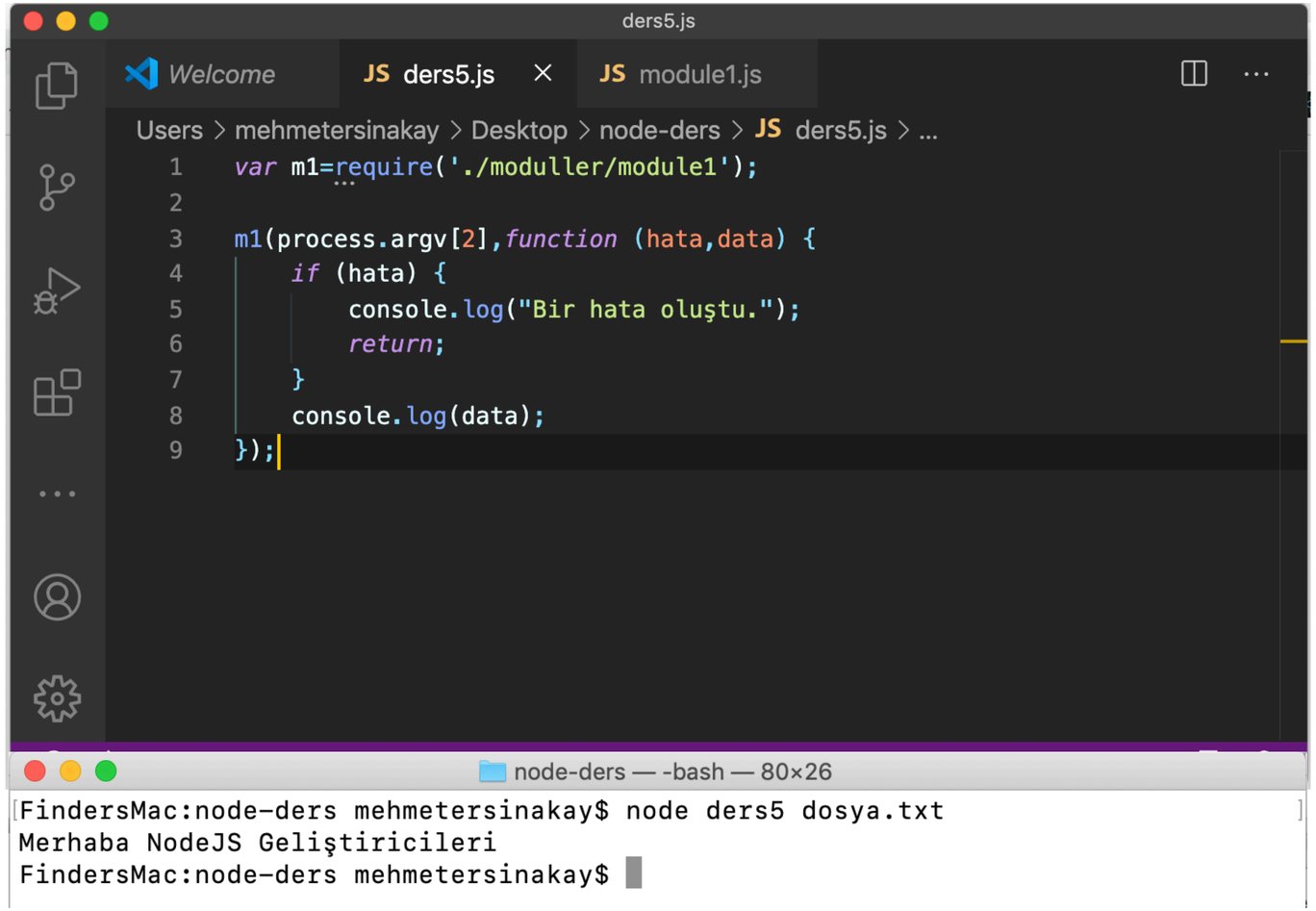
Modüler Programlama Örnek-1



- Bu modülde modüle yollanan dosya ismine göre dosya okuyup içeriğini geri gönderiyoruz.
- module.exports = function(…) … kodları ile dosyamızı bir modüle çevirmiş olduk. Bu kodlar sayesinde bu dosyayı çağıran her program yazmış olduğumuz modülü bir fonksiyon olarak kullanabilecek.
- Fonksiyonumuz içerisinde fs modülü içerisindeki readFile fonksiyonu ile bize yollanan dosyaAdi değişkeni içerisindeki dosyayı okuyoruz. readFile'ın callback fonksiyonu içerisinde ilk olarak hata kontrolü yapıyoruz. eğer hata varsa bize yollanan callback değişkeninin ilk parametresine veriyoruz ve geri döndürüyoruz.
- Eğer hata yoksa kodlarımız devam edecek. Bir sonraki satırda callback değişkeninin ilk parametresine null, ikinci parametresine okumuş olduğumuz dosyanın içeriğini stringe çevirip veriyoruz

Node.js

Modüler Programlama Örnek-1



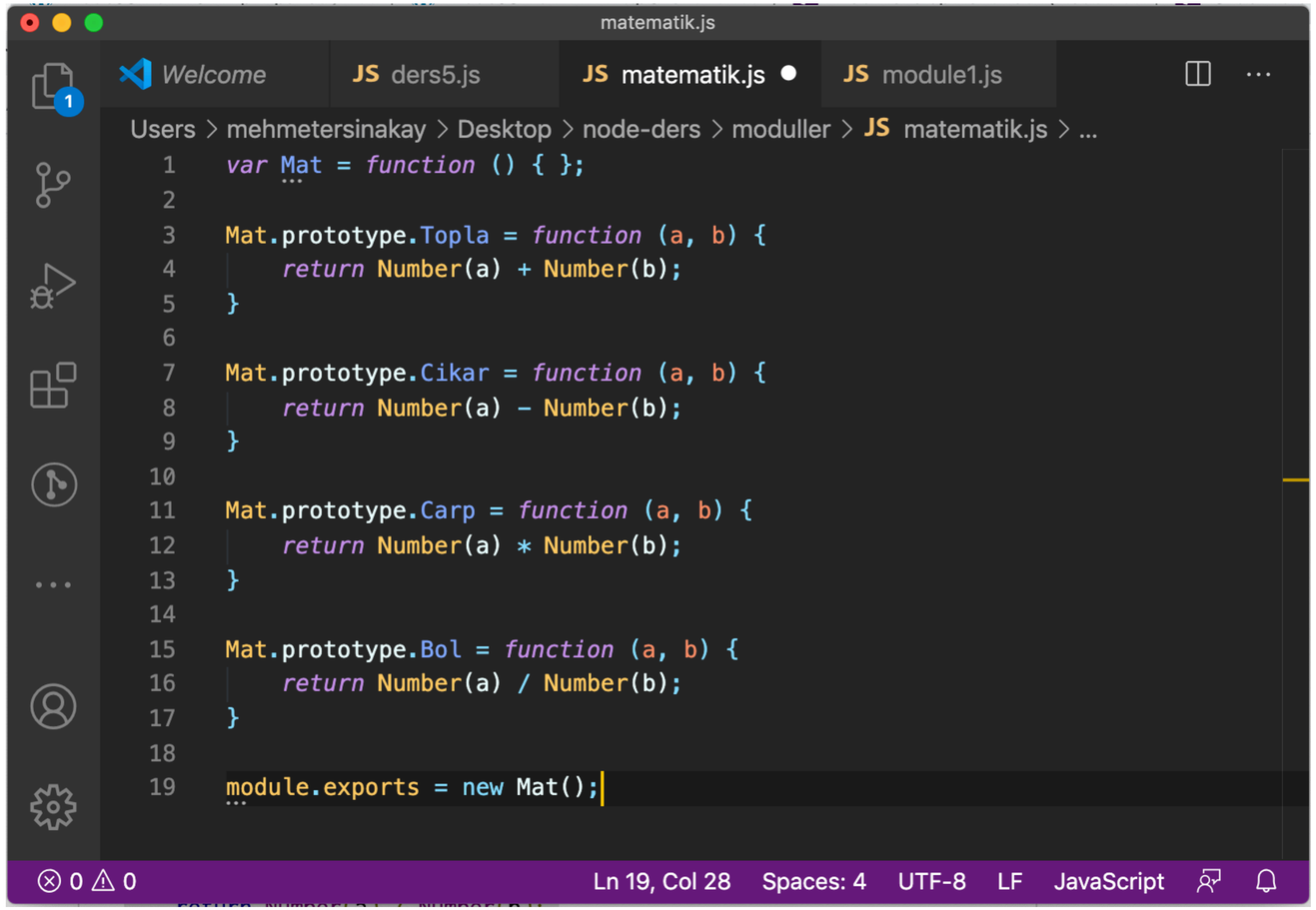
- İlk satırda yazmış olduğumuz modülü çağırıp m1 değişkenine atadık. Bizim modülümüz tek bir fonksiyondan oluşuyor. Dolayısıyla m1 değişkeninin türü aslında bir fonksiyon. Yani biz m1 değişkenini bir fonksiyon gibi kullanacağız.

- Modülümüzün ilk değişkeni okuyacağımız dosyanın ismiydi. Biz dosya ismini konsoldan parametre olarak alacağız. Almış olduğumuz bu parametreyi de m1 değişkenine yani modülümüzün ilk parametresine vereceğiz.

- İkinci parametre olarak da bir fonksiyon yazıyoruz. Bu fonksiyon bizim modülümüzdeki callback değişkenine denk geliyor. Yani modülümüzün işi bittiğinde yada hata olduğunda bu fonksiyon çalışacak. Bu fonksiyonda ilk olarak hata kontrolü yaptık. Daha sonra gelen data değişkenini console ekranına yazdırdık.

Node.js

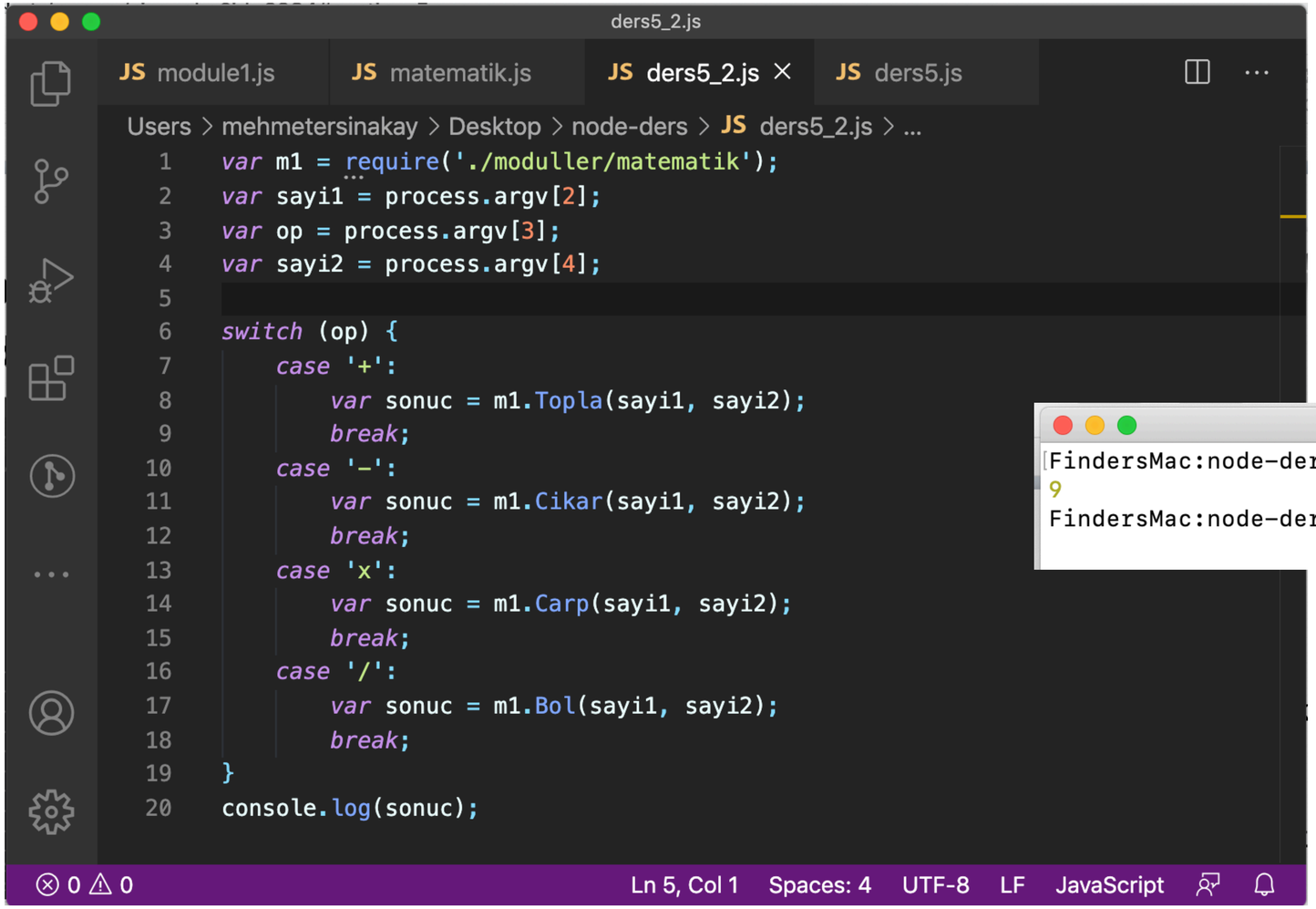
Modüler Programlama Örnek-2

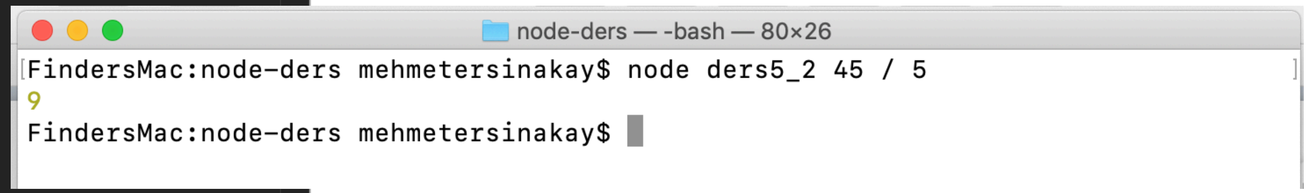


- Peki fs modülünde olduğu gibi içerisinde birden fazla fonksiyon olan bir modülü nasıl hazırlarız?
- İkinci örneğimizde Matematik adında bir modül yapacağız. Modülümüzün 4 tane fonksiyonu olacak : Topla, Çıkar, Çarp, Böl.
- Mat adında bir javascript sınıfı oluşturduk. Bu nesneye Topla, Cikar, Carp, Bol adında 4 tane fonksiyon oluşturduk.
- En son satırda da oluşturmuş olduğumuz Mat sınıfının bir nesnesini modül olarak atadık
- Modülümüz içerisindeki fonksiyonlar asenkron değiller. Kodlarımız içerisinde herhangi bir asenkron işlem yapmadığımız için kodlarımız da asenkron olmadı.
- Önceki örnekte modülümüz içerisinde asenkron bir fonksiyon kullanmıştık. Bu nedenle modülümüz asenkron olmuştu.

Node.js

Modüler Programlama Örnek-2





Node.js

Npm Paket Yöneticisi

- Yazdığımız bu modülleri başkalarının kullanmasını sağlayabilir miyiz?
- Başkalarının yazdığı modülleri biz kullanabilir miyiz?
    - Daha sonra bu modülün sahibi paketi güncellediğinde kolayca güncelleyebilir miyiz?
- Bunu npm(node package manager) ile yapabiliriz.
- npm nodejs'i yüklediğimizde yüklenen bir araçtır.
- npm ile oluşturmuş olduğumuz modülleri başkalarının kullanması için yayınlayabiliriz.
- npm resmi olarak https://www.npmjs.com adresini kullanır. Yayınlamış olduğumuz paketlere herkes bu adresten erişebilir. Yine npm ile herkesin açık olarak yayınladığı paketlere biz de buradan ulaşabiliriz

Node.js

Npm Paket Yöneticisi

- npm tamamen konsol üzerinden kullanılan bir araçtır. npm ile tüm işlerimizi konsol komutları ile yapacağız.
- Paket yüklemek için kullanacağımız komut ise şu şekilde:
  npm install *paketadi*
- Bu komut ile npm üzerindeki bütün paketlere ulaşabiliriz. Tabi bu paketlerin ne işe yaradıkları ve nasıl kullanıldıklarını paketlerin dökümantasyonundan öğrenmeliyiz.
- Örnek olarak **image-size** paketini yükleyelim. Bunun için konsol ekranına npm install image-size yazmamız yeterli. Bu komut npm'nin deposundan image-size paketini bizim için indirecektir.
- Peki ama nereye yükleyecek? npm ile yüklediğimiz paketler bulunduğumuz klasör içerisindeki **node_modules** adındaki bir klasöre yüklenir.(Klasör yoksa oluşturulur.)
- Bu paketleri kod içerisinde nodejs'in var olan paketleri gibi kullanabiliriz. Örneğin image-size paketini kullanmak için var img_size=require("image-size"); yazmamız yeterli.
- Nodejs bu paketi node_modules klasörü içerinde bulup img_size değişkenine atacaktır. image-size içerisinde bütün nesne ve fonksiyonları da img_size değişkeni ile erişebilir.

Node.js

Npm Paket Yöneticisi - Global Kullanım

- npm install -global *paketadı*
  npm install -g *paketadı*

- **image-size** paketini global olarak yüklemek için konsola şu komutu vermemiz yeterli:
  npm install –global image-size

- Bu komut ile image-size paketini bilgisayarımızda bulunan global npm deposuna yüklüyor ve biz image-size paketini konsol üzerinden istediğimiz gibi kullanabiliyoruz.

- Mesela image-size ile bir resmin ölçülerini öğrenmek için

  image-size images/resim.jpg

  komutunu çalıştırmamız yeterlidir.

Node.js

Npm Paket Yöneticisi

- npm ile sadece nodejs paketleri yüklemiyoruz. Nodejs paketleri dışında css paketleri, javascript paketleri gibi paketler de yükleyebiliriz.
- Mesela bootstrap css framework'ünü npm ile yükleyebiliriz. Bunun için şu komutu vermemiz yeterli
  npm install bootstrap
- Aynı şekilde jquery javascript kütüphanesini yüklemek istiyorsak şu komutu vermemiz yeterlidir:
  npm install jquery
- npm'nin bir diğer yeteneği de yüklemek istediğimiz paketlerin istediğimiz versiyonunu yükleyebiliyor olmamız. Mesela jquery'nin 1.11.2 versiyonunu yüklemek istiyorsak şu komut bizim için yeterli olacaktır:
  npm install jquery@1.11.2
- Yükleyeceğimiz paketin çalışması için başka bir paket yüklenmesi gerekiyorsa npm bizim için onları da yüklüyor. Mesela **jquery-validation** paketinin çalışması için jquery paketinin de yüklenmesi gerekiyor. Siz npm install jquery-validation komutunu çalıştırdığınızda npm jquery-validation paketiyle beraber jquery paketini zaten yüklemiş oluyor.

Node.js

package.json

- Peki bizim projemiz için toplam paket sayısı 50yi geçti diyelim. O zaman bizim projemizin boyutu çok yükseleceği için taşınabilirliği düşecektir.

- Projeyi iş arkadaşımıza vereceğiz diyelim. 50 paketin hepsiyle beraber kopyalayıp vereceğiz yada diyelimki kodlarımızı versiyon kontrol sistemine yüklüyoruz ve sürekli versiyonlama yapıyoruz. Yükleme ve versiyonlamanın tamamı 50 paket ile birlikte olacak. Bu da çok zaman alacaktır. Bu işin kolay bir yolu yok mu? Tabiki var. **package.json** dosyası.

- **package.json dosyası;** projemizin ana dizininde bulunan ve projemiz hakkında bilgiler içeren, projemiz içerisinde kullanılan paketlerin listesini tutan, projemiz için konsolda kullanılabilecek komutlar için kısa yollar eklenebilen ve daha fazlasını içeren özel bir json dosyasıdır.

- Çalışma klasörümüz kök seviyedeyken **npm init** yazarak konsol üzerinden adım adım dosyamızı oluşturabilir yada kendimiz de json formatında içeriğini yazıp kaydedebiliriz

Node.js

package.json

```
{
  "name": "proje_adi",
  "version": "1.0.0",
  "keywords": [
    "nodejs",
    "npm",
    "ders",
    "ornek"
  ],
  "description": "proje açıklaması",
  "main": "program.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": “Ersin Akay",
  "license": "MIT",
  "dependencies": {
    "bootstrap": "^3.3.7",
    "jquery-ui": "^1.12.0",
    "lodash": "^4.15.0"
  }
}
```

- name alanı projemizin adı,
- version alanı projemizin şu an hangi versiyonda olduğu,
- keywords alanı projemiz hakkındaki anahtar kelimeleri,
- description alanı projemiz hakkında açıklama,
- main alanı projemizi çalıştırmaya hangi dosyadan başlanacağını,
- scripts projemiz için konsolda çalıştırabileceğimiz komutların kısaltmaları,
- author alanı proje yazarı/sahibi,
- license proje lisansı
- dependencies projemizin bağımlılıkları yani projemizde kullandığımız paketlerin listesi

Node.js

package.json

- Dosyamızı oluşturuk. Bu dosya projemiz hakkında bir çok bilgi barındırıyor. En önemlisi de paket listesi.
- Artık projemizi taşırken **node_modules** klasörünü silebiliriz yada projemizi versiyon kontrol sistemlerine aktarırken node_modules klasörünü hariç tutabiliriz(exclude).
- Peki ama projeyi alan başka bir kişi paketleri tekrar nasıl yükleyecek?
  npm install
- Biz bu komutu konsola yazdığımızda npm package.json dosyasına bakacak ve içerisindeki paketleri yükleyecek. Özellikle github üzerinden indirdiğiniz veya checkout yaptığınız projelerin çoğunda projeyi çalıştırmadan önce bu komutu vermeniz gerekir.
- package.json içerisindeki scripts kısmı için projemiz için konsolda kullanacağımız komutların listesini tutabiliriz demiştik.
- Mesela programımızı konsolda çalıştırmak için node program.js komutunu veriyorduk. Bunu package.json içerindeki script kısmına ekleyelim. Artık projemizi npm ile başlatabiliriz. Konsola şu komutu vermemiz yeterli : **npm start**

```
{
  ...
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node program.js"
  },
  ...
}
```

Node.js

HttpClient ve Events(Olaylar)

- Önceki derslerimizde callback fonksiyonundan bahsetmiştik. Callback fonksiyonu sayesinde asenkron işlemlerin bitiminde yapmamız gerekenleri belirtebiliyorduk. Peki asenkron işlemler sırasında “bitmek” olayı dışında başka olaylar meydana geldiğinde callback fonksiyonu gibi bir fonksiyonla o olay sırasında yapılması gereken işlemleri de belirtebilir miyiz?

- nodejs’in olaylar özelliği sayesinde meydana gelen her olayda bir fonksiyon çalışmasını sağlayabiliriz.

- Örnek olarak bir web sayfasını çağıracağız ve onun üzerinde meydana gelen olaylara fonksiyon atayacağız. Bunun için kullanacağımız modül nodejs’in kendi modülü olan http modülü.

- Bu modül sayesinde http protokolü ile ilgili her işlemi yapabiliriz. Nodejs’in en güçlü olduğu yanlardan bir tanesi web. Bunu sağlayan en önemli parçalardan bir tanesi de http modülüdür.

- Nodejs’in hem asenkron mimarisi, hem olay bazlı yapısı, hem de http modülü birleşince nodejs’in web üzerinde ölçeklenebilir web siteleri, web uygulamaları ve api yazmak çok cazip hale gelmiştir.

Node.js

HttpClient ve Events(Olaylar)

```
var http = require("http")

http.get(“http://www.google.com”,function (response) {
    response.setEncoding('utf8')
    response.on('data',function (data) {
        console.log(data)
    })
})
```

- İlk satırda nodejs’in global http modülünü çağırıp http değişkenine atıyoruz. Daha sonra http modülünün get fonksiyonu ile **http://www.google.com** adresine istekte bulunuyoruz.
- İstek işleminin bitiminde de callback fonksiyonunu çalıştırıyoruz. Callback fonksiyonumuza tek parametre dönderiliyor. O da response yani cevap değişkeni. response değişkeni ile yapmış olduğumuz isteğe verilecek cevap hakkında bilgi alabiliriz.
- Callback fonksiyonunun ilk satırında setEncoding ile yazı formatımızı utf8 olarak belirliyoruz. Yazılarımız bozuk çıkmasın diye.
- response değişkeninin on metodu ile bize gönderilecek cevap ile ilgili olaylar meydana geldiğinde işlemler yapabiliriz. Örneğimizde response.on(‘data’,…) şeklinde yazmış olduğumuz ifade şu anlama geliyor: response değişkeninde her data olayı gerçekleştiğinde şu fonksiyonu çalıştır.
- data olayı nedir peki? Biz bir web adresine istekte bulunduğumuzda web adresi bize tek seferde teslim edilmez. Parça parça teslim edilir. İşte her bir parça bize teslim edildiğinde data olayı çalışır. Yani data olayı data teslim edildi olayıdır. Buradaki callback fonksiyonu da her data teslim edildiğinde çalışır ve bu datayı konsola yazar.

Node.js

HttpClient ve Events(Olaylar)

```
var http = require("http")

http.get(process.argv[2], function (response) {
    response.setEncoding('utf8')
    response.on('data', function (data) {
        console.log(data)
    })
    response.on('end', function () {
        console.log('Yüklenme işlemi bitti.')
    })
    response.on('error', function (hata) {
        console.log('Yüklenme sırasında bir hata oluştu:' +
hata.message)
    })
})
.on('error', function (hata) {
    console.log("Web adresinin açılması sırasında bir hata oluştu."
+ hata.message)
})
```

- Bu örnekte web adresini konsoldan parametre olarak alıyoruz.
- Ek olarak da iki tane daha olay var.
- Birisi end olayı. Adından anlaşılacağı üzere yükleme işlemi bittiğinde tetiklenen olaydır.
- Diğeri ise error olayı. Bu da herhangi bir hata olduğunda tetiklenen bir olaydır.

Node.js

API

- Olaylar yardımı ile bir web adresine istekte bulunup cevabı almayı gördük. Peki bu bizim ne işimize yarayacak?

- Gelen html içeriği xml parser araçları yardımı ile parse edebilirsiniz yani ayıklayabilirsiniz ve içerisinde işinize yarayabilecek verileri alabilirsiniz veya API servislerini kullanabilirsiniz.

- API'lar yine http protokolü ile çalışan URL'ler üzerinden ulaşabildiğimiz bize daha net bilgi veren json veya xml formatında kullanılabilen web servisleridir.

Node.js

API (https://openweathermap.org/)

```
var http = require("http")

http.get('http://api.openweathermap.org/data/2.5/weather?q=' + process.argv[2] +
'&units=metric&appid=fa31b0a8e81dcc4a5cd5dcb76b203652', function (response) {
    response.setEncoding('utf8')

    var sonuc = '';

    response.on('data', function (data) {
        sonuc = sonuc + data;
    })
    response.on('end', function () {
        var jsonSonuc = JSON.parse(sonuc);
        console.log(jsonSonuc.main.temp)
    })
    response.on('error', function (hata) {
        console.log('Yüklenme sırasında bir hata oluştu:' + hata.message)
    })
})
.on('error', function (hata) {
    console.log("Açılma sırasında bir hata oluştu." + hata.message)
})
```

- Bu örneği çalıştırabilmeniz için öncelikle https://openweathermap.org/ adresinden ücretsiz üyelik oluşturup bir API KEY edinmeniz gerekmektedir.
- http.get(…) fonksiyonuna API'ın URL'ini yazıyoruz.
- URL içerisindeki process.argv[2] bölümünde ise konsoldan aldıdığımız il adını URL içerisine ekliyoruz.
- URL'in sonundaki appid bölümündeki Key alanını siz kendi keyiniz ile değiştiriniz.
- Daha sonra callback fonksiyonumuz çalışıyor. Callback fonksiyonu içerisinde sonuc isminde bir değişken ile datayı alıyoruz
- end olayında ise elde ettiğimiz sonuc değişkenini JSON'a dönüştürüyor ve jsonSonuc değişkenine eşitliyoruz.
- Son olarak da ekrana jsonSonuc nesnesinin main.temp değerini yazdırıyoruz.
- Derece bilgisi(main.temp) direk yazdırmadan önce jsonSonuc değerini yazdırarak dönen son veriyi ve bu API'nin kendi dokümantasyonunu incelemek faydalı olacaktır.

Node.js

Web Sunucu (Web Server) - Örnek 1

```
var http = require("http")

var server = http.createServer(function (request, response)
{
    var tarih = new Date();
    var formatliTarih = tarih.getDate() + "." +
(tarih.getMonth() + 1)
    + "." + tarih.getFullYear();
    formatliTarih += " " + tarih.getHours() + ":" +
tarih.getMinutes();
    console.log(formatliTarih)
    response.end(formatliTarih)
})
server.listen(8080)
console.log("Server Başlatıldı. Tarayıcı üzerinden http://
localhost:8080"
           +" adresinden ulaşabilirsiniz.")
```

- http.createServer(…) fonksiyonu ile yeni bir server oluşturuyoruz ve bunu server değişkenine atıyoruz.
- createServer fonksiyonuna callback fonksiyonu olarak verdiğimiz fonksiyon server'a her çağrı yapıldığında çalışır. request ve response olmak üzere iki parametre alır.
- request server'a yapılan istek hakkında bilgi almak için kullanılıyor, response ise yapılan isteğe cevap döndürmek için kullanılıyor.
- Fonksiyonun ilk satırında bir tarih nesnesi oluşturuluyor. Daha sonraki iki satırda ekrana formatlı olarak tarih yazdırmak için formatliTarih değişkeni oluşturup değerini atıyoruz. Daha sonraki satırda konsol ekranına formatliTarih nesnesini yazdırıyoruz.
- Bir sonraki satırda response nesnesini kullanarak yapılan isteğe cevap döndürüyoruz.
- Sonraki satırda yani server.listen(8080) kodu ile oluşturmuş olduğumuz serverı 8080 portu üzerinden başlatıyoruz.
- Sonraki satırda da konsola serverımızın başladığını haber veren bir yazı yazıyoruz.

NOT: Konsolda programı çalıştırdığınızda server sürekli çalışacaktır. Kapatmak için CTRL + C tuşlarını kullanabilirsiniz.

Node.js

Web Sunucu (Web Server) - Örnek 2

```
var http = require("http")
var fs=require("fs")

var server = http.createServer(function (request,
response) {
    fs.createReadStream("ornek.html").pipe(response)
})
server.listen(8080)
console.log("Server Başlatıldı. Tarayıcı üzerinden
http://localhost:8080"
           +" adresinden ulaşabilirsiniz.")
```

- Bu örneğimizde, bir önceki örnekten farklı olarak ikinci satırda fs modülümüzü çağırıyoruz
- Callback fonksiyonunda fs modülünün createReadStream fonksiyonunu kullanıyoruz.
- Bu fonksiyon parametre olarak okuyacağı dosyanın adını alıyor ve bir okuma işlemi başlatıyor.
- Fonksiyonun devamındaki pipe fonksiyonu ise; ilk işlemin çıktısını alıp ikinci işleme veriyor.
- İlk işlem yani ornek.html dosyasının okunma işleminin çıktısı yani ornek.html dosyasını al sonraki işlem yani pipe fonksiyonuna parametre olarak verilen response nesnesine ver.
- response nesnesi de dosyayı alıp cevap olarak html dosyasını alan tarayıcıda html içeriğini işleyip ekranda gösterecek.

Node.js

Web Sunucu (Web Server) - Örnek 3

```
var http = require("http")
var url = require("url")
var fs = require("fs")

var server = http.createServer(function (request, response)
{
    var urlObject = url.parse(request.url)
    if (urlObject.pathname == '/ornek1') {
        fs.createReadStream('ornek1.html').pipe(response)
    } else if (urlObject.pathname == '/ornek2') {
        fs.createReadStream('ornek2.html').pipe(response)
    }
    else {
        fs.createReadStream('ornek.html').pipe(response)
    }
})
server.listen(8080)
console.log("Server Başlatıldı. Tarayıcı üzerinden http://
localhost:8080"
           +" adresinden ulaşabilirsiniz.")
```

- Bu örneğimizde kodlarımıza bakacak olursak kullandığımız 3 tane modülümüz var: http, url ve fs.
- Sonrasında daha önce öğrendiğimiz server oluşturma işlemini yapıyoruz.
- Callback fonksiyonunun ilk satırında bize gönderilen yani kullanıcının tarayıcıya yazmış olduğu url'yi parse(ayrıştırma) işlemine alıyoruz ve urlObject değişkenine atıyoruz.
- Böylece url hakkında her şeye daha sade olarak erişebiliriz. Daha sonrasında basit if deyimleriyle url'ye yazılan değerleri kontrol ediyoruz.
- Bu if ifadelerinin içerisinde de url değerine göre daha önce gördüğümüz gibi kullanıcıya dosya gönderiyoruz. En son else kısmında da geriye kalan her ihtimal için ornek.html dosyasını gönderiyoruz.
- Html ve css biliyorsanız artık nodejs ile basit bir site yapabilirsiniz

Node.js

Web Sunucu (Web Server) - Örnek 4 (API)

```
var http = require("http")
var url = require("url")

var server = http.createServer(function (request, response) {
    var urlObject = url.parse(request.url)
    var zaman=new Date()

    if (urlObject.pathname == '/tarih') {
        var tarih={
            yil:zaman.getFullYear(),
            ay:zaman.getMonth()+1,
            gun:zaman.getDate()
        };
        response.end(JSON.stringify(tarih))
    } else if (urlObject.pathname == '/saat') {
        var saat={
            saat:zaman.getHours(),
            dakika:zaman.getMinutes(),
            saniye:zaman.getSeconds()
        };
        response.end(JSON.stringify(saat))
    }
    else {
        response.writeHead(200, {"Content-Type": "text/html; charset=utf-8"});
        response.write("Lütfen saati öğrenmek için /saat ,")
        response.write("tarihi öğrenmek için /tarih adreslerini kullanınız.")
        response.end()
    }
})
server.listen(8080)
console.log("Server Başlatıldı. Tarayıcı üzerinden http://localhost:8080"
          +" adresinden ulaşabilirsiniz.")
```

- url /tarih ise tarih adında bir nesne oluşturuyoruz. İçerisine yil, ay, gun değişkenlerini ve değerlerini giriyoruz. Daha sonra JSON.stringify(…) ile nesnemizi json'a çeviriyoruz ve response.end(…) ile dönüştürmüş olduğumuz değişkeni geri döndürüyoruz.
- Sonraki else if kod bloğunda url /saat ise yeni bir saat değişkeni oluşturup içerisine saat, dakika, saniye değişkenleri ve değerlerini yerleştiriyoruz. Ve json nesnesine çevirip kullanıcıya döndürüyoruz.
- else bloğunda ise kullanıcı bu iki url dışında herhangi bir şey yazmışsa veya boş bırakmışsa kullanıcıya bilgilendirme metni yolluyoruz.
- İlk olarak response.writeHead(…) fonksiyonu ile yollayacağımız metnin türünü ve kodlama standardını belirtiyoruz.
- Daha sonraki iki satırda response.write(…) fonksiyonu ile kullanıcıya bilgilendirme yazısı yolluyoruz.
- Son olarakta response.end() ile işlemlerimizi bitirip kullanıcıya yaptığımız işlemleri döndürüyoruz.

Node.js

Express Framework

- Web server oluşturmayı ve kendi web sitelerimizi nasıl yapabileceğimizi gördük.
- Peki bu işin daha profesyonel olan bir yolu yok mu? Daha kolay bir şekilde web sitesi yapabileceğimiz, bir çok angarya işten bizi kurtaracak bir yol yok mu?
- Tabiki var. Nodejs için yazılmış Express Web Framework'u tam da bu işe yarıyor.
- npm üzerinden paket olarak yükleyebileceğimiz Express sayesinde Url parse işlemlerini(routing) daha kolay bir şekilde yapabilir, statik dosya yönetimi işlerimizi daha kolay bir şekilde yapabiliriz.
- Kısaca nodejs ile web sitesi/uygulaması yapmak için gerekli tüm alt yapıyı Expressjs bize sağlamaktadır.
- Express'i yüklemek için konsola 'npm install express' yazmamız yeterli.

Node.js

Express Framework - Yönlendirme (Routing)

- Routing yani Yönlendirme gelen Url'e göre nasıl bir işlem yapılması gerektiğini, nereye yönlendirilmesi gerektiğini belirttiğimiz yapıdır.
- Express içerisindeki routing yapısı Http methodlarıyla çalışır. Genel yapı app.METHOT(URL_YOLU, CALLBACK) şeklindedir.
- Metot kısmında Http protokolünün GET, POST, PUT, DELETE … gibi metotlarından biri kullanılır.
- Web siteleri/uygulamaları HTTP protokolü üzerinde çalışır. Siz bir tarayıcıya site adresini yazdığınızda Sunucuya HTTP protokolü üzerinden HTTP'nin herhangi bir metodu ile beraber gider. Bu sayede sunucu üzerinde url ve http metoduna göre işlem yapılabilir. Varsayılan HTTP Metodu GET metodudur. Siz tarayıcı üzerinden bir adrese girmeye çalıştığınızda tarayıcı varsayılan olarak GET metodunu kullanır

Node.js

Express Framework - Yönlendirme (Routing)

```
var express=require('express');
var app=express();

app.get('/',function (req,res) {
    res.send('Ana Sayfa!');
});

app.post('/',function (req,res) {
    res.send('Ana Sayfa Post!');
});

app.get('/Sayfa1',function (req,res) {
    res.send('Sayfa 1!');
});

app.post('/SayfaPost',function (req,res) {
    res.send('Sayfa Post!');
});

app.put('/SayfaPut',function (req,res) {
    res.send('Sayfa Put!');
});

app.delete('/SayfaDelete',function (req,res) {
    res.send('Sayfa Delete!');
});

app.listen(3000,function () {
    console.log('Uygulama 3000 portunda çalışmakta.')
});
```

- İlk olarak yüklemiş olduğumuz express paketini express değişkenine attık.
- Daha sonra app isminde bir express uygulaması oluşturduk.
- Bir sonraki işlemde bir route oluşturuyoruz. Bu route üzerinde url olarak ‘/’ girildiğinde yani boş bırakıldığında ne olması gerektiğini belirttik.
- Bunun için oluşturmuş olduğumuz app uygulamasının get fonksiyonunu kullandık. get fonksiyonunun callback fonksiyonu iki değişken alıyor: req,res. Birisi request yani isteği yönettiğimiz req diğeri response yani cevabı yönettiğimiz res değişkeni.
- Callback fonksiyonu içerisinde res değişkeninin send fonksiyonu ile kullanıcıya bilgi gönderebiliyoruz. Burada kullanıcıya Ana Sayfa! yazısını gönderdik.
- Diğer route işleminde yine url olarak ‘/’ girildiğinde ama Http Post metodu kullanıldığında ne olması gerektiğine bakıyoruz.
- Diğer route işlemlerinde farklı Http metodları ve farklı Urller üzerinde neler yapılması gerektiğini belirtiyoruz.
- Son bölümünde de app express uygulamasını 3000 portundan çalışmasını sağlıyoruz.

Node.js

Express Framework - Yönlendirme (Routing)

```
var express=require('express');
var app=express();

app.get('/',function (req,res) {
    res.send('Ana Sayfa!');
});

app.all('/sayfahepsi',function (req,res) {
    res.send('Sayfa Hepsi!');
});

app.get('/ab?cd',function (req,res) {
    res.send('Sayfa ab?cd!');
});

app.get('/de+fg',function (req,res) {
    res.send('Sayfa de+fg!');
});

app.get('/hi*jk',function (req,res) {
    res.send('Sayfa hi*jk!');
});

app.get('/sayfa/:degisken1/parametre/:degisken2', function(req,
res) {
    res.send(req.params);
});

app.listen(3000,function () {
    console.log('Uygulama 3000 portunda çalışmakta.')
});
```

- İkinci route tanımına bakacak olursak, farklı olarak app.all(…) fonksiyonu kullanıldı. Bu fonksiyon URL sağlandığı sürece bu route tanımının bütün HTTP metotlarıyla çalışacağını gösteriyor.
- Üçüncü route tanımına bakalım. Burada da farklı olarak URL kısmında “/ab?cd” ifadesi kullanıldı. Buradaki ? işareti kendinden önceki gelen harfin seçimlik olduğunu gösteriyor. Yani biz tarayıcı üzerinden “/abcd” de yazsak “/acd” de yazsak yine bu route çalışacak demektir.
- Dördünce route tanımında ise URL kısmında “/de+fg” ifadesi kullanıldı. Buradaki + işaretinin anlamı ise kendinden önceki gelen harfin istediği kadar çoğaltılabileceği anlamına geliyor. Yani “/defg” de yazsak “/deeeeeefg” de yazsak bu route çalışacak.
- Beşinci routing tanımında ise URL kısmında “/hi*jk” ifadesi kullanıldı. Buradaki * işaretinin anlamı araya istediğiniz yazabilirsiniz demek. Yani “/hijk” yazsakta “/hi123456jk” yazsakta bu route çalışacak demektir.
- Altıncı ve son route tanımına bakacak olursak burada da URL içerisinde degisken1 ve degisken2 diye iki tane degişken tanımı bulunmaktadır. Değişken tanımları önlerine konan : işareti ile belirtilmektedir. URL içerisindeki bu değişkenlere ne yazılırsa Program içerisinden onlara erişebiliriz demektir.
- Bu değişkenlere route tanımının callback fonksiyonundaki req değişkeninin params değişkeni ile erişebiliriz. Örneğin URL “/sayfa/111/parametre/222” yazdığımızda bu değişkenlere “req.params.degisken1” yada “req.params.degisken2” şeklinde erişebiliriz.

Node.js

Express Framework - Sabit Dosyalar (Static Files)

```
var express=require('express');
var app=express();

app.get('/',function (req,res) {
    res.send('Ana Sayfa!');
});

app.use(express.static('public'));

app.use('/js',express.static('libs'));

app.listen(3000,function () {
    console.log('Uygulama 3000 portunda
çalışmakta.')
});
```

- Büyük projelerin çoğunda onlarca css, js ve resim dosyaları vardır. Hepsi için tek tek route tanımlaması mı yapmamız gerekiyor?

- app.use(express.static('public'));

  uygulamamızda public klasörünün dışarıdan direkt olarak erişilebileceğini ifade ediyoruz. Yani public klasörü içerisinde bulunan bütün dosyaları URL'in sonuna ekleyerek çağırabiliriz. (klasörün adını yani public yazmadan).

- app.use('/js',express.static('libs'));

  sanal klasör kullanıyoruz. Yani static klasörünün içerisindeki herşeyi dışarıdan erişime açıyoruz. (klasörün adını yani libs yazmadan). Ama bu dosyalara erişirken başına js yazarak erişiyoruz.

Node.js

Express Framework - Middleware

- Web projeleri doğası gereği request-response yapısına sahiptir. Kullanıcı yazmış olduğumuz uygulamaya bir istekte bulunur(request). Uygulama bu isteği alır belirli aşamalardan geçirerek istek yapan kullanıcıya bir cevap oluşturur ve yollar(response). Burada bahsettiğimiz aşamalar kullanmış olduğumuz framework'e göre değişir.

- Bizim nodejs ile kullandığımız expressjs de belirli aşamalara sahiptir. Gelen istekler expressjs'nin yaşam döngüsü içerisindeki her bir aşamadan geçerek cevap oluşturulur. İşte bu her bir aşamaya expressjs'de middleware denir.

- Middleware aslında bir fonksiyondur. Fonksiyon imzası şu şekildedir: function(req, res, next) { … } .Buradaki req (request) ile bize gönderilen istek hakkındaki bilgilere ulaşabiliriz, res (response) ile verilecek olan cevaba müdahale edebiliriz, next ile de bir sonraki middleware fonksiyonunu çalıştırabiliriz.

Node.js

Express Framework - Middleware

```
var express=require('express');
var app=express();

var logger= function (req, res, next) {
  var tarih = new Date();
  var formatliTarih = tarih.getDate() + "." + (tarih.getMonth() +
1) + "." + tarih.getFullYear();
  formatliTarih += " " + tarih.getHours() + ":" +
tarih.getMinutes();
  console.log(formatliTarih + ':Yeni istek')
  next()
};

app.use(logger);

app.get('/',function (req,res) {
    res.send('Merhaba Dünya!');
});

app.listen(3000,function () {
    console.log('Uygulama 3000 portunda çalışmakta.')
});
```

- logger adında bir fonksiyon oluşturuyoruz. Bu fonksiyon 3 tane parametre alıyor: req, res ve next. Yani middleware fonksiyon formatına uygun.
- Bu fonksiyon içerisinde tarih formatını oluşturuyoruz.
- Diğer adımda da konsol üzerine bu tarihi ve “Yeni istek” yazısını yazdırıyoruz.
- Fonksiyonun son satırında da parametre olarak gelen **next()** fonksiyonunu çalıştırıyoruz.
- **app.use(logger);** ile bu fonksiyonu uygulamamıza bir middleware olarak kullanmasını söylüyoruz.
- Uygulamamızı çalıştırdığımızda logger middleware gelen her istek için konsol ekranına tarihi ve yeni bir istek geldiğini belirten yazısını yazacak.
- En basit hali ile bir middleware fonksiyonu yazmış olduk.

Node.js

Express Framework - Middleware

- Expressjs middleware türleri;
    - Application-level middleware
    - Router-level middleware
    - Error-handling middleware
    - Built-in middleware
    - Third-party middleware

Node.js

Express Framework - Middleware

```
var express = require('express');
var app = express();
var router = express.Router();
var cookieParser = require('cookie-parser');

// Application-level middleware
app.use(function (req, res, next) {
    console.log(Date.now() + ': Yeni istek, Method:' + req.method + ', URL:' +
req.originalUrl)
    next();
});

app.get('/', function (req, res, next) {
    res.send('Merhaba Dünya!');
    next();
});

// Router-level middleware
router.use(function (req, res, next) {
    console.log('Yeni istek(Router-level middleware):', Date.now());
    next();
})

app.use('/', router)

// Error-handling middleware
app.use(function (err, req, res, next) {
    console.error("Hata:" + err.stack);
    res.status(500).send('Bir hata oluştu!');
});

// Built-in middleware
app.use(express.static('public'));

// Third-party middleware
app.use(cookieParser());

app.listen(3000, function () {
    console.log('Uygulama 3000 portunda çalışmakta.');
});
```

- Daha önce kullandığımız app.use(…), app.METHOD(…) (METHOD dan kastımız GET,PUT,POST,DELETE… gibi http metotları) kullanımları aslında bir **Application-level middleware**.

- Daha sonra expressjs içerisindeki **Router-level middleware** kullanımı görmekteyiz. Bunu kullanmak için öncelikle en üstte **var router = express.Router()** tanımı yaptığımıza dikkat edin. app.use yerine router.use diyoruz. Asıl dikkat edilmesi gereken yer **app.use('/', router)** tanımı. Bu tanım ile expressjs route tanımı içerisinde yapmış olduğumuz değişiklikleri expressjs sistemine tanıtmış oluyoruz

- Bir diğer middleware çeşidi de **Error-handling middleware**. Error-handling middleware sistem üzerinde herhangi bir hata gerçekleştiğinde çalışır. Kullanımında dikkat etmemiz gereken yer fonksiyon imzası. Burada kullanılan fonksiyon 4 parametre almaktadır: app.use(function(**err, req, res, next**){ … }); . Buradaki err parametresi ile meydana gelen hata hakkında ekstra bilgiler elde edip hata türüne göre işlem yapabiliriz.

- Daha sonraki satırda **Built-in middleware** örneği bulunmakta. Aslında bu tanım daha kullandığımız statik dosya tanımlama adımı. public klasöründeki varolan dosyaları dış kullanıma açmaya yarıyor.

- Son middleware türümüz de **Third-party middleware**'lar. Tabi bunları kullanmadan npm install ile modülü kurmamız ve en üstte **var cookieParser = require('cookie-parser')** şeklinde modüllerini yüklememiz gerekiyor. Bunlar expressjs topluluğunun genel kullanım için yazdığı middleware'lar.

Node.js

Express Framework - Şablon Motorları(Template Engines)

- Web Projelerinde proje büyüdükçe görünümlerin(view) dinamik olarak üretilmesi ve bunun iyi yönetilmesi ihtiyacı oluşur. Bu ihtiyaç şablon motorları ile karşılanmaktadır.
- Express framework ile kullanabileceğimiz bazı şablon motorları;
  - ejs
  - handlebars
  - pug
  - mustache
  - jade

Node.js

Express Framework - Şablon Motorları(Template Engines)

```
var express = require('express');
var app = express();

app.set('view engine', 'ejs');

var users = [
    { name: 'Ahmet Beyaz', email: 'aaa@gmail.com' },
    { name: 'Mehmet Sarı', email: 'bbb@gmail.com' },
    { name: 'Ayşe Pembe', email: 'ccc@gmail.com' }
  ];

app.get("/test", function (req, res) {

  res.render('index', {
    users: users,
    title: "Kullanıcı Listesi",
  });

});


app.get('/',
    function(req, res){
        res.send("Merhaba anasayfa..")
    }
);

app.use(express.static('public'));
app.use('/js', express.static('libs'));

app.listen(4000, function(){ console.log('Express çalışıyorr...')})
```

- ejs şablon motorunu **npm install ejs** ile kuruyoruz
- app.set(‘view engine’) ile şablon motorunu seçiyoruz.
- Varsayılan olarak expressjs’de web görünümleri (‘views’) klasöründe yer alacak şekildedir.
- res.render ile ilk parametrede views klasöründeki index.ejs dosyasının render edileceğini, ikinci parametrede gönderilen datayı temsil etmektedir

Node.js

Express Framework - Şablon Motorları(Template Engines)

```
<!doctype html>
<html lang="en">
  <head>
   <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/
bootstrap.min.css" rel="stylesheet" integrity="sha384-
giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"
crossorigin="anonymous">

    <title><%= title %></title>
  </head>
  <body>
    <div class="container mt-5">
        <h1><%= title %></h1>
        <table class="table table-striped table-hover">
            <tr><th>Ad Soyad</th><th>E-Posta</th></tr>
            <% users.forEach(function(user){ %>
                <tr><td><%= user.name %></td><td><%= user.email %></td></tr>
            <% }) %>
        </table>
    </div>
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/
bootstrap.bundle.min.js" integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/
t7LECLbyPA2x65Kgf80OJFdroafW" crossorigin="anonymous"></script>

 </body>
</html>
```

- index.ejs sayfası
- <% %> işaretleri arasında gönderilen data ile ilgili çıktı işlemleri yapılabilmektedir
- Ayrıca javascript metodları da kullanılabilmektedir

Node.js

Express Framework - Şablon Motorları(Template Engines)

```
<%- include('header'); %>
        <h1><%= title %></h1>
        <table class="table table-striped table-hover">
            <tr><th>Ad Soyad</th><th>E-Posta</th></tr>
            <% users.forEach(function(user){ %>
                <tr><td><%= user.name %></td><td><%= user.email
%></td></tr>
            <% }) %>
        </table>
<%- include('footer'); %>
```

- tekrarlamayan kod parçalarını başka .ejs dosyalarına alabiliriz.
- Örneğin sayfaların baş ve son kısımlarını header.ejs ve footer.ejs dosyalarına aktarıp kaydedebiliriz.
- Daha sonra da index.ejs'de <%- include('header'); %> komutuyla olması gereken yerde çağırabiliriz
- Böyle statik ve dinamik yapıları birbirinden ayırarak daha etkin yönetilebilir bir şablon yapısı oluşturabiliriz