

MAKÜ|BTF

BUCAK TEKNOLOJİ FAKÜLTESİ

Web Tasarımı ve Programlama

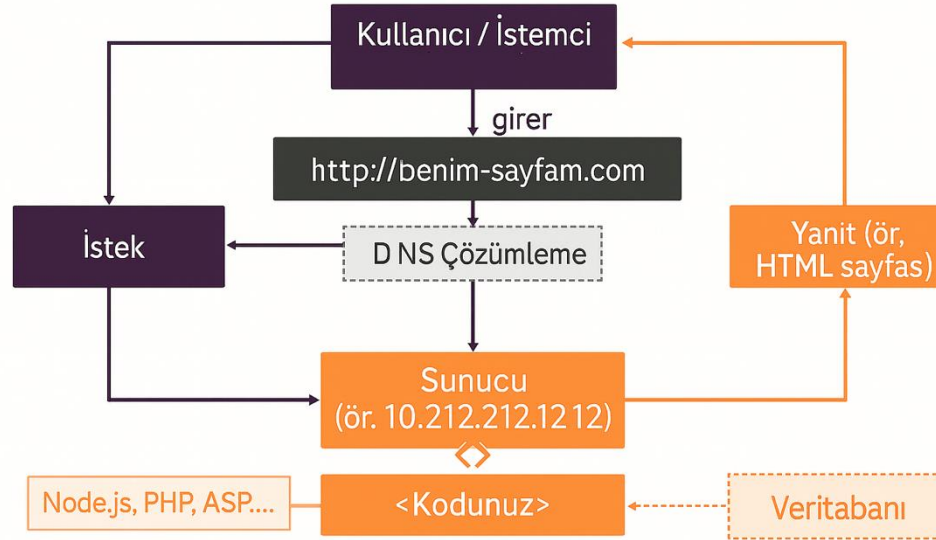
Ders 5: Nodejs Temelleri

Dr. Öğr. Üyesi Elvan DUMAN

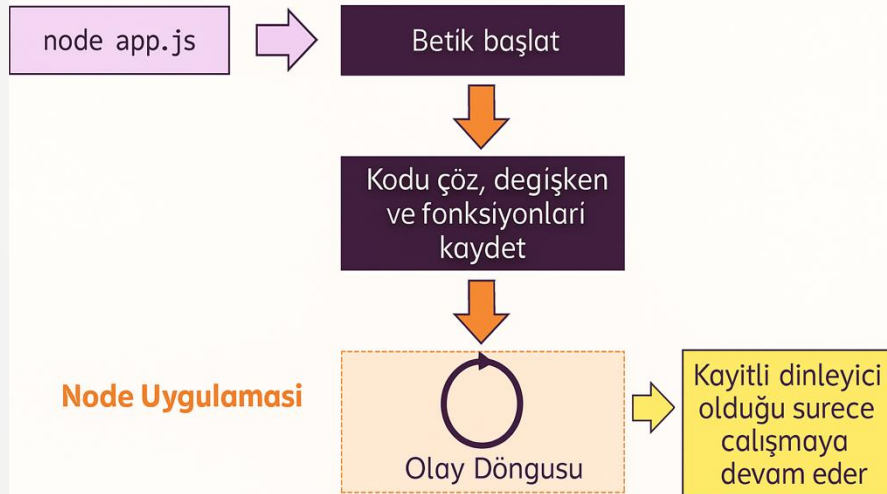
eduman@mehmetakif.edu.tr
Yazılım Mühendisliği Bölümü

1. Web'in Çalışma Mantığı
2. Node.js ile İlk Sunucu
3. İstek (Request) ve Yanıt (Response)
4. Dosya İşlemleri (fs Modülü)
5. NPM ve package.json
6. Express.js'e Giriş
7. Middleware Mantığı
8. Express.js ile Routing

WEB NASIL ÇALIŞIR



- Kullanıcı (istemci) bir web sayfasına tarayıcıyla erişir.
- Tarayıcı adresi DNS çözümleme ile IP adresine dönüştürür.
- Sunucuya istek (request) gönderilir.
- Sunucu isteği işler, yanıt (response) gönderir.
- Yanıt: HTML sayfası, dosya veya veri (ör. JSON).
- Sunucu gelen isteği işler, yanıt üretir.
- Kod örnekleri: Node.js, PHP, ASP.NET...
- İstek ve yanıt başlık (header) bilgileri taşır.
- HTTP iletişim standardıdır, HTTPS ise şifreli versiyonudur.
- Gerçek sistemlerde (production) HTTPS zorunludur.



```
let http = require('http');
```

```
const server=http.createServer((req,res)  
=> {  
  console.log(req);  
  //process.exit()  
});
```

```
server.listen(3000);
```

- node app.js komutu betiği başlatır.
- Node.js kodu çözümler, değişkenleri ve fonksiyonları kaydeder.
- Program hemen bitmez. event loop (olay döngüsü) devreye girer.
- Olay döngüsü, kayıtlı dinleyiciler (event listeners) olduğu sürece çalışmaya devam eder.
- createServer() . kalıcı bir “istek dinleyicisi” oluşturur.

```
let http = require('http');

const server=http.createServer((req,res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>İlk Nodejs
başlığı</title></head>');
  res.write('<body><h1>Merhaba Nodejs</h1></body>');
  res.write('</html>');
  res.end();

});
server.listen(3000);
```

- İsteğe ait bilgiler** : req.url, req.method, req.headers
- res.setHeader('Content-Type', 'text/html')** : tarayıcıya yanıt türünü bildirir.
- res.write()** : HTML içeriğini satır satır yazar.
- res.end()** : yanıt tamamlanır ve istemciye gönderilir.
- Tarayıcıda http://localhost:3000** : “Merhaba Node.js” görüntülenir.

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

const server=http.createServer((req,res) => {

const url = req.url;
const method=req.method;
if(url == '/') {
res.write('<html>');
res.write('<head><title>Kayıt Formu</title></head>');
res.write('<body><form action="/kaydet" method="POST"><input type="text"
name="bilgi"><button type="submit">Kaydet</button></form></body>');
res.write('</html>');
return res.end();
}

if(url=='/kaydet' && method==='POST') {
fs.writeFileSync('bilgi.txt', 'Elvan');
res.statusCode=302;
res.setHeader('Location', '/');
return res.end();
}
res.setHeader('Content-Type', 'text/html');
res.write('<html>');
res.write('<head><title>İlk Nodejs başlığı</title></head>');
res.write('<body><h1>Merhaba Nodejs</h1></body>');
res.write('</html>');
res.end();
//process.exit()
});
server.listen(3000);
```

- http ve fs modülleri içe aktarılır.
- / isteğinde kullanıcıya HTML formu gönderilir.
- Formun action değeri /kaydet, method değeri POST olur.
- /kaydet isteğinde form verisi işlenir.
- fs.writeFileSync() fonksiyonu: Dosyaya veri kaydeder.
- res.statusCode = 302 ve res.setHeader('Location', '/'): Kullanıcıyı yönlendirir.

NOT: Ancak nodejs ile body'den gelen mesajları parçalamak (parsing) zor ve zahmetlidir. Bu sebeple express.js kurup ilerlemek daha mantıklıdır.

- **npm** : Node Package Manager, Node.js ile birlikte kurulur.
- Görevleri:
 - Üçüncü taraf paketleri yüklemek.
 - Projeyi tanımlamak ve yönetmek.

npm init : Proje için **package.json** dosyası oluşturur.

Bu dosya:

- Proje adı, sürüm, açıklama, yazar bilgilerini içerir.
- "scripts" alanında özel komutlar tanımlanır.

JSON biçimi: anahtarlar ve değerler çift tırnakla yazılır.

- **Npm init çalışmazsa**

- Windows arama çubuğuna **PowerShell** yaz.
- “**Windows PowerShell**” üzerine sağ tıklayıp “**Yönetici olarak çalıştır (Run as Administrator)**” seç.
- Aşağıdaki komutu yaz ve **Enter**’a bas:
- **Set-ExecutionPolicy RemoteSigned**

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned
```

```
Execution Policy Change
```

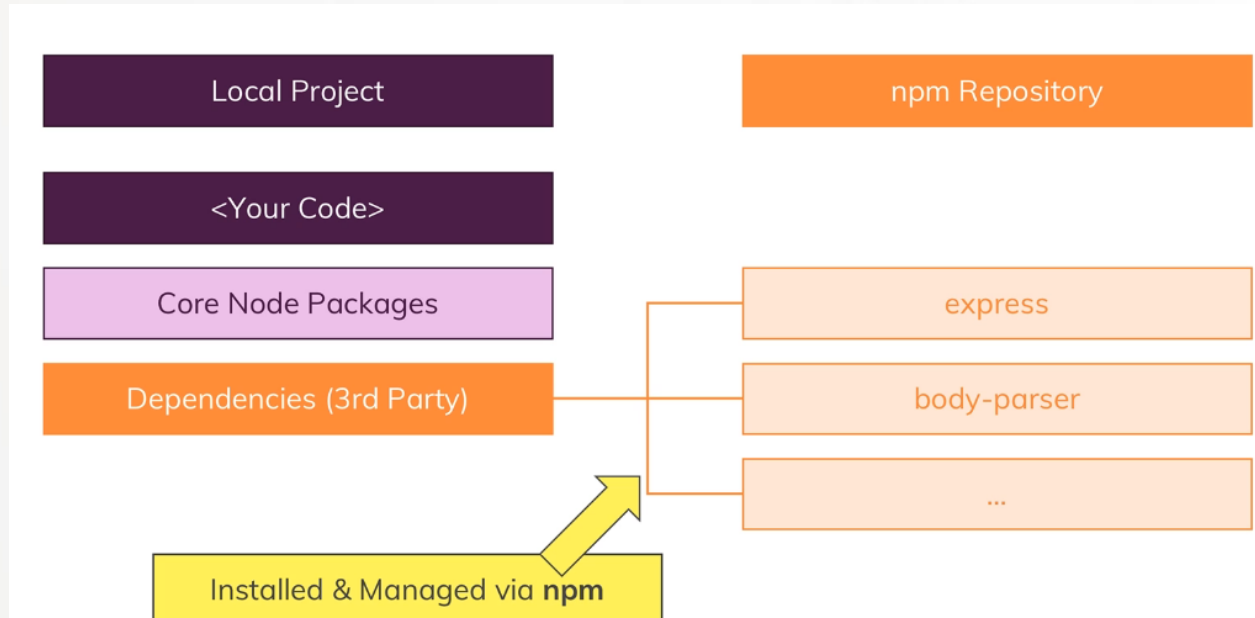
```
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose you to the security risks described in the about_Execution_Policies help topic at
```

```
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
```

```
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y
```

- {
- "name": "node_app",
- "version": "1.0.0",
- "description": "",
- "main": "app.js",
- "scripts": {
- "test": "echo \"Error: no test specified\" && exit 1",
- "start": "node app.js"
- },
- "author": "elvan",
- "license": "ISC"
- }

- Package.json dosyası oluşacaktır.
- "start": "node app.js" eklemesiyle birlikte node app.js yerine npm start ile sunucu çalıştırılabilir.



- package.json varsa proje “yönetilen” bir Node projesidir; **üçüncü taraf paketler** (express, body-parser, nodemon vb.) **npm** ile kurulabilir.
- Paket tipleri:
 - dependencies**: üretimde (production) da gerekenler.
 - devDependencies**: sadece geliştirme sırasında yardımcı olanlar (örn. **nodemon** otomatik yeniden başlatma).
 - g (global)**: makinede her yerde kullanılacak kurulum.
- node_modules klasörü çok büyüyebilir; gerekirse silebilirsin. Projeyi yeniden açınca npm install ile geri kurulur.
- package-lock.json tam sürümleri kilitler; projeyi paylaşan/alan herkes aynı sürümleri kurar.

- `npm install nodemon --save-dev`
- Geliştirme (kaydedince otomatik yeniden başlatır):
- **Aktif etmek için Package.json dosyasında start kısmına artık nodemon.js yazılır.**

```
Debug  
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "nodemon app.js"  
},
```

- **Tekrar npm start dediğimizde artık güncelleme sonrası sunucu otomatik yüklenir.**

```
PS C:\Users\EDuman\Desktop\node_app> npm start  
  
> node_app@1.0.0 start  
> nodemon app.js  
  
[nodemon] 3.1.10  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node app.js`  
|
```

- `npm install nodemon --save-dev`
- Geliştirme (kaydedince otomatik yeniden başlatır):
- **Aktif etmek için Package.json dosyasında start kısmına artık nodemon.js yazılır.**

```
Debug  
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "nodemon app.js"  
},
```

- **Tekrar npm start dediğimizde artık güncelleme sonrası sunucu otomatik yüklenir.**

```
PS C:\Users\EDuman\Desktop\node_app> npm start  
  
> node_app@1.0.0 start  
> nodemon app.js  
  
[nodemon] 3.1.10  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node app.js`  
|
```

- Vanilla (saf) Node.js ile sunucu yazmak zahmetlidir.
- Örnek: İstek gövdesini (body) ayrıştırmak için data ve end olaylarını manuel dinlememiz gerekir.
- Farklı veri türleri (ör. dosya, JSON, form verisi) için yeni mantıklar yazmak gerekir.
- Express.js bu karmaşayı ortadan kaldırır.
- İstekleri, yönlendirmeleri (routes) ve yanıtları yönetmeyi kolaylaştırır.
- Gerekli olduğunda ek paketlerle (ör. body-parser) kolayca genişletilebilir.

- Express bir **framework**:
 - Yardımcı fonksiyonlar + belirli kurallar ve yapı önerileri sunar.
 - Kodun düzenli, okunabilir ve genişletilebilir olmasını sağlar.
- Node.js için diğer framework örnekleri:
 - Adonis.js (Laravel benzeri)
 - Koa, Sails.js, vb.
- **Express.js** en popüler ve esnek framework'tür.
- Az ama güçlü özelliklerle gelir.
- Binlerce eklenti paketiyle genişletilebilir yapıdadır.
- **Yüklemek için**
 - `npm install --save express`
- Save-dev yapmadık çünkü sunucuda da kurulmasını istiyoruz. Geliştirme için değil üretim için gerekli.

Temel Kullanım:

```
const http = require('http');  
  
const express = require('express');  
  
const app = express();  
  
const server = http.createServer(app);  
  
server.listen(3000);
```

Middleware

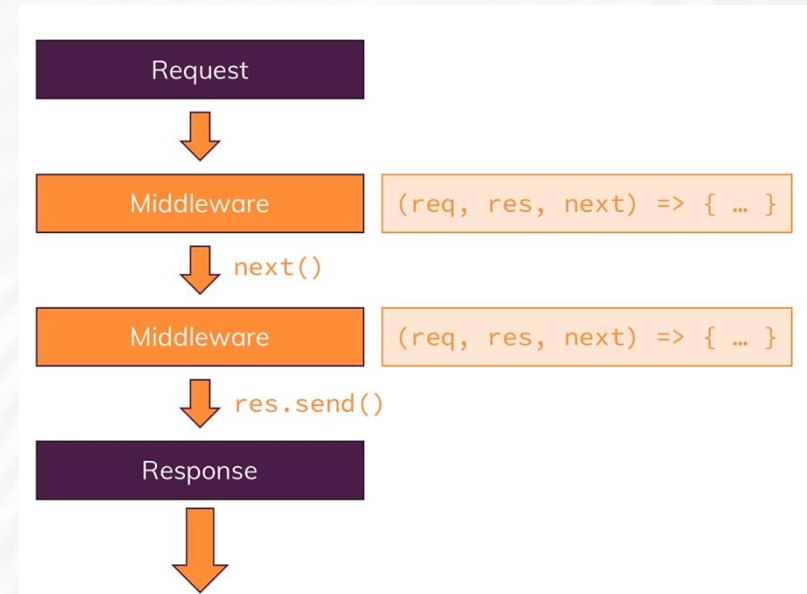
Express.js yapısının temeli: **Middleware (Ara Katmanlar)**

- Middleware = Gelen isteğin (request) bir dizi fonksiyondan geçmesi
- Tek bir büyük fonksiyon yerine, görevleri küçük parçalara ayırmamızı sağlar.

Her middleware:

- request → İstek verilerini taşır
- response → Yanıt oluşturur
- next() → Sıradaki middleware'e geçişi sağlar

Middleware zinciri: Üstten alta doğru işler.



- Eğer next() çağrılmazsa istek zincirde durur ve yanıt gönderilmez.

```
const http = require('http');
const express = require('express');
const app = express();

app.use((req,res, next) => {
  console.log('Ara katmandayım.');
```

next(); // yeni katmana gitmesine müsaade eder

```
});
app.use((req,res, next) => {
  console.log('2. Ara katmandayım.');
```

});

```
const server = http.createServer(app);
server.listen(3000);
```

Middleware eklemek için `app.use()` yöntemi kullanılır.

İstek yukarıdan aşağıya tüm middleware'lerden geçer.

`next()` çağrılmazsa zincir durur → mutlaka `res.send()` veya `res.end()` ile yanıt ver.

Üçüncü taraf paketler (ör. `body-parser`, `morgan`) kolayca middleware olarak eklenebilir.

Eğer next() çağrılmazsa istek zincirde durur.

Express varsayılan olarak **otomatik yanıt göndermez**.

Bu durumda yanıtı **bizim göndermemiz gerekir**.

res.send() fonksiyonu ile yanıt göndermek çok daha kolaydır.

res.send()

- Her tür veriyi (HTML, JSON, string, vb.) gönderebilir.
- Gönderilen içeriğe göre Content-Type başlığını (header) otomatik ayarlar.

```
const http = require('http');
const express = require('express');
const app = express();

app.use((req, res, next) => {
  console.log('Ara katmandayım. ');
  res.send('<h1>Express Ara katmandan merhaba</h1>');
  next(); // yeni katmana gitmesine müsaade eder
});

app.use((req, res, next) => {
  console.log('2. Ara katmandayım. ');
});

/*
const server = http.createServer(app);
server.listen(3000);
*/
app.listen(3000); // yukardaki kodun yerine kullanılabilir.
```

Express.js, farklı URL'ler (routelar) için farklı işlemler tanımlamamıza izin verir.

app.use() metodunun ilk parametresi olarak path (yol) belirtilebilir:

Eğer path verilmezse tüm isteklerde çalışır.

Express, middleware'leri üstten alta doğru çalıştırır.

next() çağrılmazsa zincir durur, alttaki middleware'ler çalışmaz. Bu yüzden rota tanımları sırası önemlidir.

```
const http = require('http');
const express = require('express');
const app = express();

app.use('/kaydet', (req, res, next) => {
  console.log('Kaydet katmanındayım.');
```

```
  res.send('<h1>Express Kaydet katmanından merhaba</h1>');
  next(); // yeni katmana gitmesine müsaade eder
});
app.use('/', (req, res, next) => {
  console.log('Ara katmandayım.');
```

```
  res.send('<h1>Express Ara katmandan merhaba</h1>');
  next(); // yeni katmana gitmesine müsaade eder
});

app.listen(3000);
```

Ders 5 Sunum Sonu

İletişim : eduman@mehmetakif.edu.tr

