

Web Tasarımı ve Programlama

Ders 6:Routing ve Form İşlemleri)

Dr. Öğr. Üyesi Elvan DUMAN

eduman@mehmetakif.edu.tr
Yazılım Mühendisliği Bölümü

1. HTML Dosyalarından Şablonlara Geçiş
2. Statik ve Dinamik İçerik Farkı
3. Templating Engine Nedir?
4. EJS (Embedded JavaScript) Yapısı ve Kurulumu
5. `res.render()` Metodu ile Veri Aktarımı
6. EJS Sözdizimi: Değişkenler, Döngüler ve Koşullar

Form Gönderimi ve POST İstekleri

- Amaç: Kullanıcıdan veri alıp sunucuya **POST** ile göndermek
- Örnek senaryo: “Ürün Ekle” formu
- Dönüş: Başarılı gönderimden sonra anasayfaya yönlendirme (**redirect**)
- **Not:** Geliştirmenin bu aşamasında sade HTML döndürüyoruz; sonra şablon motoru kullanacağız.

```
<form action="/urun-ekle" method="POST">  
  <input type="text" name="title" />  
  <button type="submit">Ürün Ekle</button>  
</form>
```

Redirect ile Kolay Yönlendirme

```
res.redirect('/');
```

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

app.use('/urun-ekle', (req, res, next) => {
  res.send('<form action="/urun" method="POST"><input
    type="text" name="urun_adi"><button type="submit">Ürün
    Ekle</button></form>');
});

app.use('/urun', (req, res, next) => {
  console.log(req.body);
  res.redirect('/');
})
app.use('/', (req, res, next) => {
  res.send('<h1>Express Anasayfa katmanından merhaba</h1>')
  next(); // yeni katmana gitmesine müsaade eder
});

app.listen(3000);
```

Sarı işaretli kod çalıştığında undefined olarak cevap döndürür.

Çünkü HTTP gövdesini Express varsayılan olarak parse etmez!!

Body Parser Kurulumu

```
npm install --save body-parser
```

Kullanımı

```
const bodyParser = require('body-parser');  
app.use(bodyParser.urlencoded({ extended: false }));
```

- **urlencoded:** Form verileri için uygun
- **extended: false:** Basit key-value yapıları (varsayılan önerilen kullanım)
- Middleware'i **rota tanımlarından önce** ekle

Not: Yeni projelerde `express.urlencoded({ extended:false })` da kullanılabilir.

```
const http = require('http');
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
```

```
app.use(bodyParser.urlencoded({ extended: false }));
```

```
app.use('/urun-ekle', (req,res, next) => {
  res.send('<form action="/urun" method="POST"><input type="text"
    name="urun_adi"><button type="submit">Ürün Ekle</button></form>');
});
app.use('/urun', (req,res,next) => {
  console.log(req.body);
  res.redirect('/');
})
app.use('/', (req,res, next) => {
  res.send('<h1>Express Anasayfa katmanından merhaba</h1>')
  next(); // yeni katmana gitmesine müsaade eder
});
app.listen(3000);
```

```
{ urun_adi: 'aaaa' }
```

- `app.use(...)` **tüm HTTP metodları** için tetiklenir (GET, POST, PUT, PATCH, DELETE...)
- Yol filtresi yapabiliriz (örn. `/urun`), fakat **metot bazı filtre** yapmaz
- Çözüm: **metoda özel** yardımcılar: `app.get(...)`, `app.post(...)`, `app.put(...)`, `app.patch(...)`, `app.delete(...)`

```
app.get('/urun-ekle', (req, res, next) => {  
  res.send('<form action="/urun" method="POST"><input type="text"  
    name="urun_adi"><button type="submit">Ürün Ekle</button></form>');  
});  
app.post('/urun', (req, res, next) => {  
  console.log(req.body);  
  res.redirect('/');  
})
```

Neden Kodumuzu Bölmeliyiz?

- Şu anda tüm kod **app.js** içinde
- Küçük uygulamalarda sorun değil, ama büyüdükçe **okunabilirlik azalır**
- Çözüm: Rotaları (routes) **ayrı dosyalara** bölmek

Express Router Nedir?

- Express, çoklu dosya yapısı için özel bir çözüm sunar: **Router**
- Router, “mini Express uygulaması” gibi davranır
- Her router, kendi get, post, use metodlarını kullanabilir
- Sonra bu router ana uygulamaya (app.js) **bağlanır**

```
// routes/admin.js
```

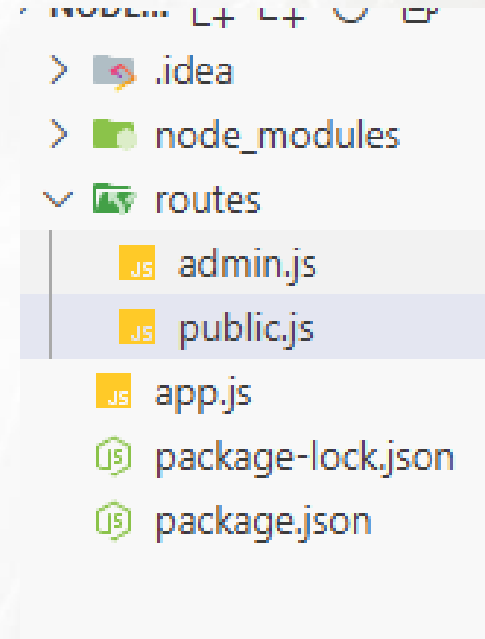
```
const express = require('express');  
const router = express.Router(); // Büyük 'R' harfli!
```

```
module.exports = router;
```

Router'ı Dışa Aktarma

```
module.exports = router;
```

Artık admin.js dışındaki dosyalar bu rotaları import edebilir. Router, **geçerli bir middleware** işlevi gibi çalışır. Bu sayede app.use(adminRoutes) şeklinde bağlanabilir



```
const http = require('http');
const express = require('express');
const app = express();
const bodyParser = require('body-parser');

const adminRoutes = require('./routes/admin'); // .js otomatik
const publicRoutes = require('./routes/public'); // .js otomatik
app.use(bodyParser.urlencoded({ extended: false }));

app.use(adminRoutes);
app.use(publicRoutes);
app.listen(3000);
```

```
const express = require('express');
const router = express.Router();

router.get('/urun-ekle', (req, res, next) => {
  res.send('<form action="/urun" method="POST"><input
type="text" name="urun_adi"><button
type="submit">Ürün Ekle</button></form>');
});

router.post('/urun', (req, res, next) => {
  console.log(req.body);
  res.redirect('/');
});

module.exports = router;
//routes/admin.js
```

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

const router = express.Router();

router.use('/', (req, res, next) => {
  res.send('<h1>Express Anasayfa
katmanından merhaba</h1>');
});

module.exports = router;

//routes/public.js
```

Route Prefixing (Yol Ön Eki) Nedir?

- Router dosyalarındaki rotalar genellikle ortak bir başlangıç yoluna sahiptir: /admin/urun-ekle, /admin/urun-duzenle, /admin/urun-sil vb.
- Her rota için /admin yazmak yerine bu ortak kısmı app.js içinde tanımlayabiliriz. Buna “prefix” (ön ek) kullanımı denir.

```
// routes/admin.js
router.get('/admin/urun-ekle', ...);
router.post('/admin/urun-ekle', ...);
```

Her rotada /admin kısmı tekrarlanır.

Prefix’li Yapı (Tavsiye Edilen)

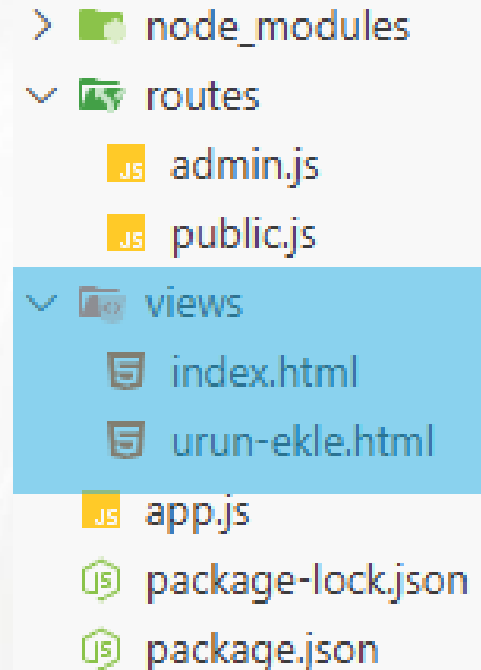
```
// app.js
const adminRoutes = require('./routes/admin');
app.use('/admin', adminRoutes);
```

```
// routes/admin.js
router.get('/urun-ekle', ...);
router.post('/urun-ekle', ...);
```

Artık dummy metin yerine dosyadan HTML döndüreceğiz.
MVC'ye hazırlık olsun diye views klasörü oluşturup görünümümüzü burada toplayacağız.

views/ Klasörü

- Tüm kullanıcıya sunulan sayfalar burada
- Şimdilik **düz HTML**; ileride **template engine** (EJS/Pug) ekleyeceğiz.
- Dosyalar HTML formatında kaydedilecek.

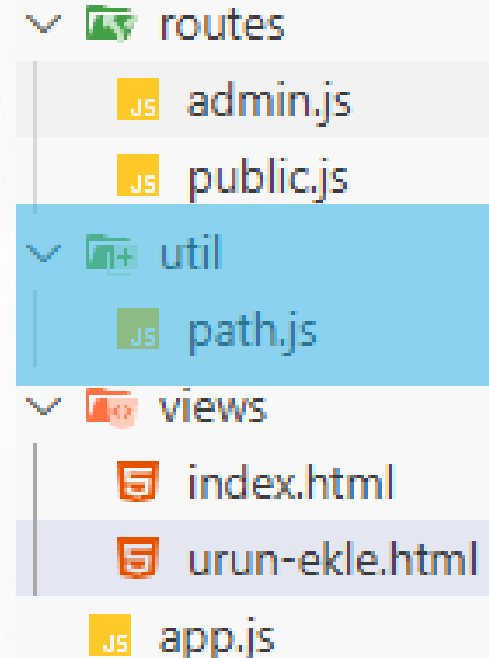


rootDir Yardımcısı

```
// util/path.js (CommonJS)  
const path = require('path');
```

```
// app.js çalıştırıldığı dizinin yolu:  
module.exports = path.dirname(require.main.filename);
```

- **require.main** : şu anda çalışan **ana dosya** (örneğin **app.js**) hakkında bilgi döndürür.
- **.filename** : o ana dosyanın **tam yolunu** verir.
- **path.dirname(...)** : bu tam yolun dizinini çıkarır (örneğin /Users/elvan/project).



Statik Dosyalar

- Stil etiketlerini (inline <style>), resim ve diğer dosyaları HTML'den çıkarıp **harici** dosyalara taşımak için kullanılır.
- Konvansiyon: herkese açık dosyalar için **public/** klasörü
- Amaç: HTML temiz kalsın, stiller tek yerden yönetilsin

```
/ app.js
const path = require('path');
const express = require('express');
const app = express();
app.use(express.urlencoded({ extended: false }));

// Statik dosyalar:
app.use(express.static(path.join(rootDir, 'public')));
```

- HTML'in içine **değişken**, **döngü**, **koşul** gibi ifadeleri yerleştirmemizi sağlar.
- Server tarafında çalışır, çıktı olarak **hazır HTML** üretir.
- Örnek motorlara şunlar verilebilir.
 - **Pug (Jade)**
 - **EJS (Embedded JavaScript)**
 - **Handlebars (hbs)**

Şu Sorulara Cevap Arıyoruz

- Veri (örneğin ürünler) nasıl yönetilir ve saklanır?
- Sunucu bu veriyi **şablonlara** nasıl aktarır?
- Şablon motorları (Pug, EJS, Handlebars) nasıl yapılandırılır?
- HTML sayfaları nasıl **dinamik** hale getirilir?

- **Geçici Veri Yönetimi (DB'siz)**
- Henüz veritabanımız yok, veriyi bellekte tutmayı göreceğiz.
- Amaç: /admin/add-product'tan gelen başlığı bir diziye eklemek
- Sonra / (public) tarafında bu diziyi kullanmak.

//admin.js

```
const path = require('path');  
const express = require('express');
```

```
const rootDir = require('../util/path');  
const router = express.Router();
```

```
const urunler=[];
```

```
router.get('/urun-ekle', (req,res, next) => {  
  res.sendFile(path.join(rootDir, 'views', 'urun-ekle.html'));  
});  
router.post('/urun-ekle', (req,res,next) => {  
  urunler.push({urun_adi:req.body.urun_adi});  
  res.redirect('/');  
});
```

```
exports.routes = router;  
exports.urunler= urunler;
```

//app.js

```
const path = require('path');  
const http = require('http');  
const express = require('express');  
const app = express();  
const bodyParser = require('body-parser');
```

```
const rootDir = require('../util/path');
```

```
const adminRoutes = require('./routes/admin').routes; //js otomatik  
const publicRoutes = require('./routes/public'); //js otomatik
```

```
app.use(bodyParser.urlencoded({ extended: false }));  
app.use(express.static(path.join(rootDir, 'public')));
```

```
app.use('/admin', adminRoutes);  
app.use(publicRoutes);
```

```
app.use((req, res, next) => {  
  res.status(404).send('<h1>Page Not Found</h1>');  
});  
app.listen(3000);
```

Neden Templating Engine?

- Statik HTML → her kullanıcıya aynı içerik
- Gerçek uygulama → veriye göre değişen sayfalar
- Çözüm: **Templating engine (şablon motoru)**
→ HTML içinde **dinamik alanlar (placeholder)** tanımlayıp sunucudan gelen verileri HTML'e enjekte eder.

Templating Engine Nasıl Çalışır?

1. Bir HTML benzeri dosya oluşturulur (örneğin public.ejs)
2. İçinde bazı **değişkenler / boşluklar** tanımlanır
`<p><%= name %></p>`
3. Express sunucusu bu şablonu işler
4. Şablondaki değişkenleri **veriyle doldurur**
5. Ortaya çıkan “gerçek” HTML kullanıcıya gönderilir

- Kullanıcı **hiçbir zaman şablonu** görmez
- Yalnızca **motor tarafından oluşturulmuş HTML çıktısını** görür
- Bu işlem **sunucu tarafında** gerçekleşir.

Neden Şablon Motoru Kullanılır?

- Kod tekrarını azaltır
- HTML + JS mantığını ayırır (MVC yapısına uygun)
- Aynı şablondan farklı verilerle farklı sayfalar üretilebilir
- Sunucu tarafında güvenli veri yerleştirme yapılır (XSS riskini azaltır)

Engine	HTML Söz Dizimi	Dinamik İçerik Simgesi	Not
EJS	Gerçek HTML	<code><%= %></code> veya <code><% %></code>	JS ifadeleri direkt yazılabilir
Pug	Kısaltılmış özel sözdizimi	<code>#{ }</code>	HTML etiketleri yerine girinti yapısı
Handlebars (HBS)	Gerçek HTML	<code>{{ }}</code>	Basit, güvenli, az özellikli

EJS

Pug (Jade)

Handlebars

`<p><%= name %></p>`

`p #{name}`

`<p>{{ name }}</p>`

Bu ders kapsamında ejs göreceğiz.

- **EJS (Embedded JavaScript)**, HTML dosyalarının içine **JavaScript kodu gömmemizi** sağlayan bir şablon motorudur.
- Express.js ile birlikte kullanıldığında, sunucu tarafında **dinamik HTML üretimi** yapar.
- Yani HTML içinde **değişken, döngü, koşul** gibi yapılar yazabiliriz.
- res.render() metodu, .ejs uzantılı bir dosyayı işler.
- Dosya içindeki `<%= %>` alanlarını verilen veriyle doldurur.
- Ortaya çıkan **HTML çıktısı** tarayıcıya gönderilir.
- Kullanıcı sadece sonuç HTML'i görür, EJS kodlarını görmez.

Kurulum:

npm install express ejs --save

```
// app.js
app.set('view engine', 'ejs');
app.set('views', 'views');
```

- view engine:** Express'e hangi motorun kullanılacağını söyler.
- views:** Şablon dosyalarının bulunduğu klasörü belirtir.

- res.render('public', { urunler });
- Denildiğinde dediğimizde views/shop.ejs dosyası çalıştırılır.

```
<ul>
  <% for (let urun of urunler) { %>
    <li><%= urun.urun_adi %></li>
  <% } %>
</ul>
```

for döngüsüyle ürünleri listeler.
<%= %> değişkenin değerini HTML içinde gösterir.

Ders 6 Sunum Sonu

İletişim : eduman@mehmetakif.edu.tr

