Node.js

index.js — blogapp

package.json | index.js

index.js > app.use() callback

```
1  const express = require("express");
2
3  const app = express();
4
5  app.use(function(req, res) {
6      res.end("hello world");
7  });
8
9  app.listen(3000, function() {
10     console.log("listening on port 3000");
11 });
```

2. Tanımları yaptıktan sonra projenin hangi portta çalışacağını yazmamız gerek.

1. Terminale;
npm init ve npm i express yazarak package.json dosyasında proje bilgilerinin oluşmasını sağlıyoruz.

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
sadikturan-Mac:blogapp sadikturan$ node index.js
listening on port 3000
^C
```

3. Çalıştır komutu

Yeni klasör

index.js | package.json | package-lock.json

package.json > scripts

```
1  {
2    "dependencies": {
3      "express": "^4.19.2"
4    },
5    "name": "deneme",
6    "version": "1.0.0",
7    "main": "index.js",
       Debug
8    "scripts": {
9      "test": "echo \"Error: no test specified\" &
10     "start": "npx nodemon index.js"
11   },
12   "author": "",
13   "
14   "
15   "
16
```

Bu komut, **nodemon** aracını kullanarak **server.js** dosyanızı başlatacaktır. Daha sonra terminalde **npm start** komutunu çalıştırarak sunucunuzu başlatabilirsiniz. Bu, projenizi başlatmayı kolaylaştıracaktır.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Berat\Desktop\Yeni klasör> npm i nodemon --save-dev

added 33

16 packag
  run `np

found 0 v
PS C:\Users\Berat\Desktop\Yeni klasör> npx nodemon
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Listenin on port 3000
Terminate batch job (Y/N)?
PS C:\Users\Berat\Desktop\Yeni klasör> npm start
```

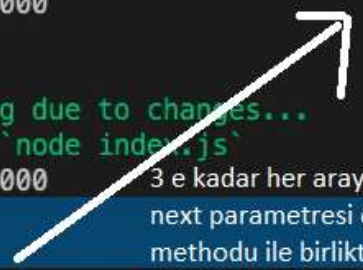
Evet, sunucuyu canlı hale getirirken ve kaydederken otomatik olarak yenilenmesini sağlamak için `nodemon` paketini yüklemek iyi bir fikirdir.

```
4
5   app.use(function(req, res, next) {
6       console.log("middleware 1");
7       next();
8   });
9
10  app.use(function(req, res, next) {
11      console.log("middleware 2");
12      next();
13  });
14
15  app.use(function(req, res) {
16      console.log("middleware 3");
17      res.send("<h1>homepage</h1>");
18  });
19
20  app.listen(3000, function() {
21      console.log("listening on port 3000");
22  });
```

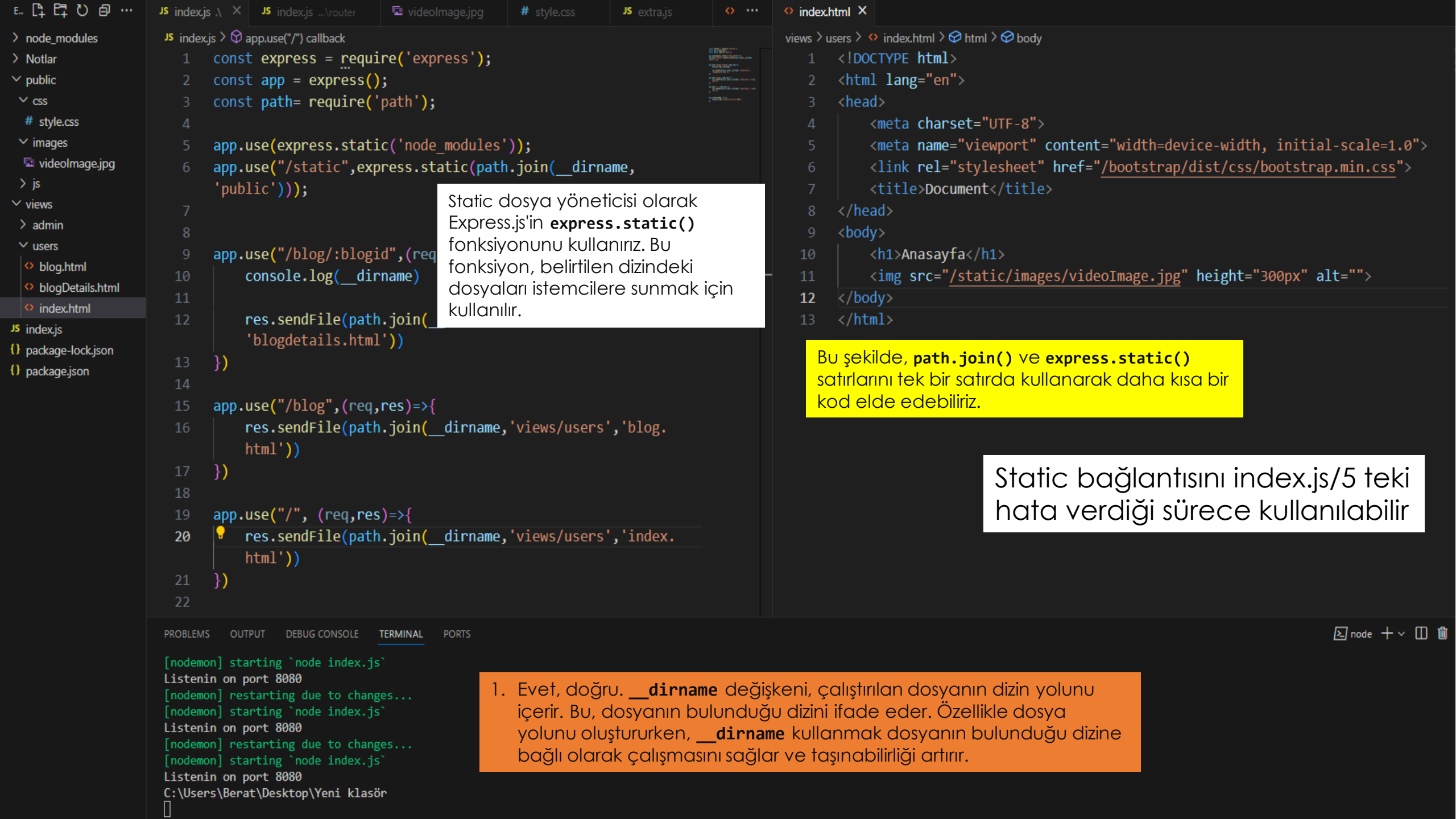
middleware eklediğimiz zaman mutlaka next parametresi eklenmeli yoksa site crash olur

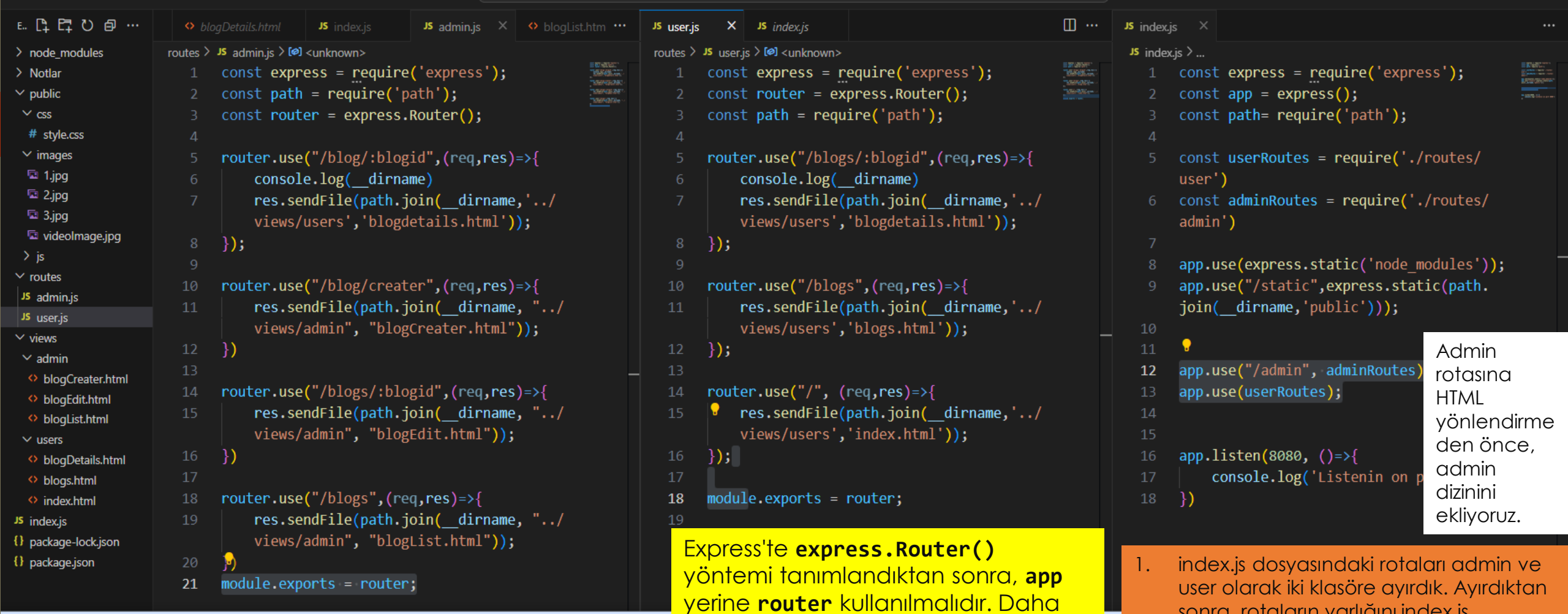
TERMINAL PROBLEMS OUTPUT DEBUG CONS

```
listening on port 3000
middleware 1
middleware 2
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
listening on port 3000
middleware 1
middleware 2
middleware 3
```



3 e kadar her arayazılım yazılır fakat 3 te next parametresi olmadığı için yazılım send methodu ile birlikte biter! (send methodu sitenin yüklenmesi için son yanıttır.


node_modules
Notlar
public
css
style.css
images
videoImage.jpg
js
views
admin
users
blog.html
blogDetails.html
index.html
index.js
package-lock.json
package.json
index.js
app.use("/") callback
const express = require('express');
const app = express();
const path= require('path');
app.use(express.static('node_modules'));
app.use("/static",express.static(path.join(__dirname,
'public')));
app.use("/blog/:blogid",(req
console.log(__dirname)
res.sendFile(path.join(__
'blogdetails.html'))
})
app.use("/blog",(req,res)=>{
res.sendFile(path.join(__dirname,'views/users','blog.
html'))
})
app.use("/", (req,res)=>{
res.sendFile(path.join(__dirname,'views/users','index.
html'))
})
Static dosya yöneticisi olarak Express.js'in express.static() fonksiyonunu kullanırız. Bu fonksiyon, belirtilen dizindeki dosyaları istemcilere sunmak için kullanılır.
index.html
views > users > index.html > html > body
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="/bootstrap/dist/css/bootstrap.min.css">
<title>Document</title>
</head>
<body>
<h1>Anasayfa</h1>
<img src="/static/images/videoImage.jpg" height="300px" alt="">
</body>
</html>
Bu şekilde, path.join() ve express.static() satırlarını tek bir satırda kullanarak daha kısa bir kod elde edebiliriz.
Static bağlantısını index.js/5 teki hata verdiği sürece kullanılabilir
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
node
[nodemon] starting `node index.js`
Listenin on port 8080
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listenin on port 8080
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listenin on port 8080
C:\Users\Berat\Desktop\Yeni klasör
1. Evet, doğru. __dirname değişkeni, çalıştırılan dosyanın dizin yolunu içerir. Bu, dosyanın bulunduğu dizini ifade eder. Özellikle dosya yolunu oluştururken, __dirname kullanmak dosyanın bulunduğu dizine bağlı olarak çalışmasını sağlar ve taşınabilirliği artırır.


node_modules
Notlar
public
css
style.css
images
1.jpg
2.jpg
3.jpg
videoImage.jpg
js
routes
admin.js
user.js
views
admin
blogCreater.html
blogEdit.html
blogList.html
users
blogDetails.html
blogs.html
index.html
index.js
package-lock.json
package.json
routes > admin.js > <unknown>
const express = require('express');
const path = require('path');
const router = express.Router();
router.use("/blog/:blogid",(req,res)=>{
console.log(__dirname)
res.sendFile(path.join(__dirname,'../views/users','blogdetails.html'));
});
router.use("/blog/creater",(req,res)=>{
res.sendFile(path.join(__dirname, "../views/admin", "blogCreater.html"));
})
router.use("/blogs/:blogid",(req,res)=>{
res.sendFile(path.join(__dirname, "../views/admin", "blogEdit.html"));
})
router.use("/blogs",(req,res)=>{
res.sendFile(path.join(__dirname, "../views/admin", "blogList.html"));
})
module.exports = router;
routes > user.js > <unknown>
const express = require('express');
const router = express.Router();
const path = require('path');
router.use("/blogs/:blogid",(req,res)=>{
console.log(__dirname)
res.sendFile(path.join(__dirname,'../views/users','blogdetails.html'));
});
router.use("/blogs",(req,res)=>{
res.sendFile(path.join(__dirname,'../views/users','blogs.html'));
});
router.use("/", (req,res)=>{
res.sendFile(path.join(__dirname,'../views/users','index.html'));
});
module.exports = router;
index.js > ...
const express = require('express');
const app = express();
const path= require('path');
const userRoutes = require('./routes/user')
const adminRoutes = require('./routes/admin')
app.use(express.static('node_modules'));
app.use("/static",express.static(path.join(__dirname,'public')));
app.use("/admin", adminRoutes)
app.use(userRoutes);
app.listen(8080, ()=>{
console.log('Listenin on p
})
Admin rotasına HTML yönlendirmeden önce, admin dizinini ekliyoruz.
Express'te express.Router() yöntemi tanımlandıktan sonra, app yerine router kullanılmalıdır. Daha sonra bu router modülü module.exports kullanılarak index.js gibi ana uygulama dosyasına bildirilmelidir.
1. index.js dosyasındaki rotaları admin ve user olarak iki klasöre ayırdık. Ayırdıktan sonra, rotaların varlığını index.js dosyasında app.use() kullanarak belirttik.
Sunu 1.pptx - Microsoft PowerP
Document
127.0.0.1:8080
BlogApp
Blog Listesi
Web Geliştirme
Mobil Uygulamalar
İçerikler
Komple Uygulamalı web geliştirme

Template Engine'ler

- Ejs
- Pub
- Handlebars

- HTML, XML veya diğer metin biçimlerini düz JavaScript ile üretebilen basit bir şablonlama dilidir. Node.js ile sıklıkla kullanılır ve sunucuda oluşturulan dinamik web sayfalarını oluşturmak için kullanılır.

Buradan itibaren Tüm blog ları movie olarak değiştirmeye karar verdim çünkü film arayüzlü bir site yapmak benim için daha eğlenceli olacağını düşündüm sizin de değiştirmenizi önerim!!!

EJS dosyalarını render ettikten sonra, path'e ihtiyacımız kalmıyor.

Burada ejs i view dizinine ekliyoruz

Kolaylık için herhangi bir ejs uygulaması indirebiliriz.

HTML dosyalarını önce index.js içine ekledikten sonra, bunları EJS dosyalarıyla değiştiriyoruz.

EJS içindeki **ejsfor** ile for döngüsü içerisine aldığımız veriyi **user.jsten** alıyoruz.

Data bilgisini aynı zamanda render dosyasına tanımlamalıyız.

```
const data = {
    title: "Popüler Kültür",
    catagories : ["Kült Filmler", "Dram Filmler", "Absürt Filmler", "Aksiyon Filmleri"],
    blogs: [
        {
            blogid: 1,
            blogTitle: "The Gotfather",
            description: "Baba, Mario Puzo'nun yazd[illegible] ayn[illegible] adl[illegible] romandan uyarlanan, Francis Ford Coppol
            Pacino'nun başrollerini paylaşt[illegible] filmdir.",
            image: "a.jpg"
        },
        {
            blogid: 2,
            blogTitle: "12 Angry Man",
            description: "12 Öfkeli Adam, Sidney Lumet'in yönettiği 1957 ABD yap[illegible] drama filmidir. Regi
            film bir duruşmada bir jüri üyesinin diğer on bir jüri üyesini şüphelinin suçsuz olduğu konu
            çabalar[illegible] hakk[illegible]ndad[illegible]r.",
            image: "b.jpg"
        }
```

Veri üzerindeki bir diziye içerik atanabiliyor.


```
            <div class="list-group">
                <% for( let index = 0; index < catagories.length; index++ ) { %>
                    <a href="" class="list-group-item list-group-item-action"><%= catagories[index] %></a>
                <% } %>
            </div>
        </div>
        <div class="col-md-9">
            <h3><%=title%></h3>

            <% blogs.forEach(b => { %>
                <div class="card mb-3">
                    <div class="row">
                        <div class="col-md-3">
                            <img src="/static/images/<%=b.image%>" class="im
                        </div>
                        <div class="col-md-9">
                            <div class="card-body">
                                <h5 class="card-title"><%=b.blogTitle %></h5
                                <p class="card-text"><%=b.description %></p>
                            </div>
                        </div>
                    </div>
                </div>
            <% }) %>
```

For döngüsü yerine **ejs.each** yöntemini kullanarak aynı işlemi gerçekleştirebiliriz. 'b' olarak adlandırdığımız fonksiyonu tanımlayarak EJS dosyalarını **b.title** şeklinde çağırmamız gerekir.

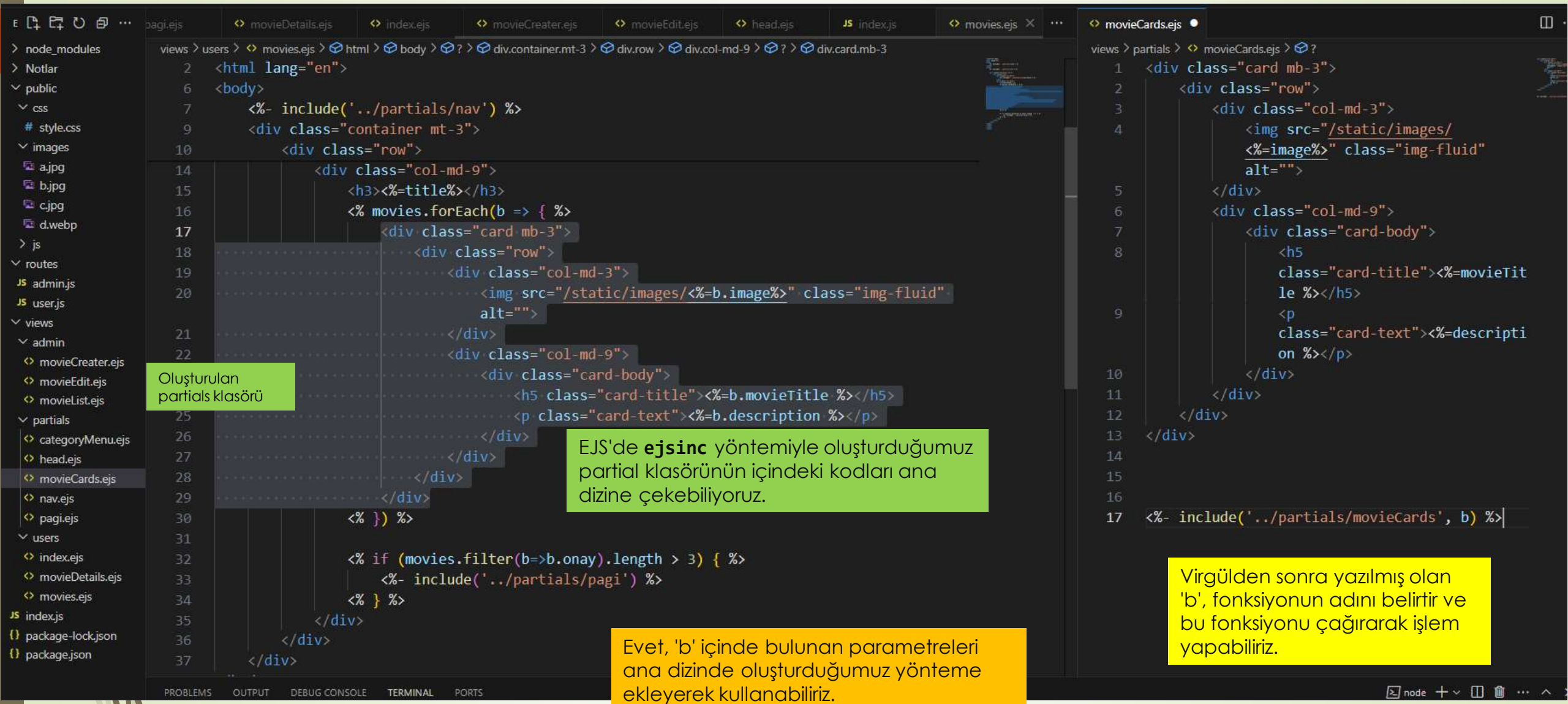
```
            yönettiği, Marlon Brando ve Al Pacino'nun
            başrollerini paylaşt[illegible] filmdir.",
            image: "a.jpg",
            main: true,
            onay: true
        },
        {
            blogid: 2,
            blogTitle: "12 Angry Man",
            description: "12 Öfkeli Adam, Sidney Lumet'in
            yönettiği 1957 ABD yap[illegible] drama filmidir. Reginald
            Rose'un ayn[illegible] adl[illegible] oyunundan uyarlanan film bir
            duruşmada bir jüri üyesinin diğer on bir jüri
            üyesini şüphelinin suçsuz olduğu konusunda, makul
            şüphe temelinde, ikna etme çabalar[illegible] hakk[illegible]ndad[illegible]r.",
            image: "b.jpg",
            main: true,
            onay: true
```

EJS'te **if** ve **ejsif** kullanarak, belirli içeriğin sayfada görünüp görünmemesini, örneğin **main** veya **onay** gibi boolean değerlere bağlı olarak kontrol edebiliriz.

```
                <% if (blogs.filter(b=>b.onay).length > 3) { %>
                    <nav aria-label="Page navigation example">
                        <ul class="pagination">
                            <li class="page-item"><a class="page-link" href="#">Previous</
                            a></li>
                            <li class="page-item"><a class="page-link" href="#">1</a></li>
                            <li class="page-item"><a class="page-link" href="#">2</a></li>
                            <li class="page-item"><a class="page-link" href="#">3</a></li>
                            <li class="page-item"><a class="page-link" href="#">Next</a></li>
                        </ul>
                    </nav>
                <% } %>
            </div>
        </div>
```

Partial View's

Kodun daha düzenli ve kolay değiştirilebilmesi için partial view kullanırız!




```
<%- include('../partials/movieCards', b) %>
```

Products

id	name	price
1	Samsung S6	2000
2	Samsung S7	3000
3	Samsung S8	4000

Users

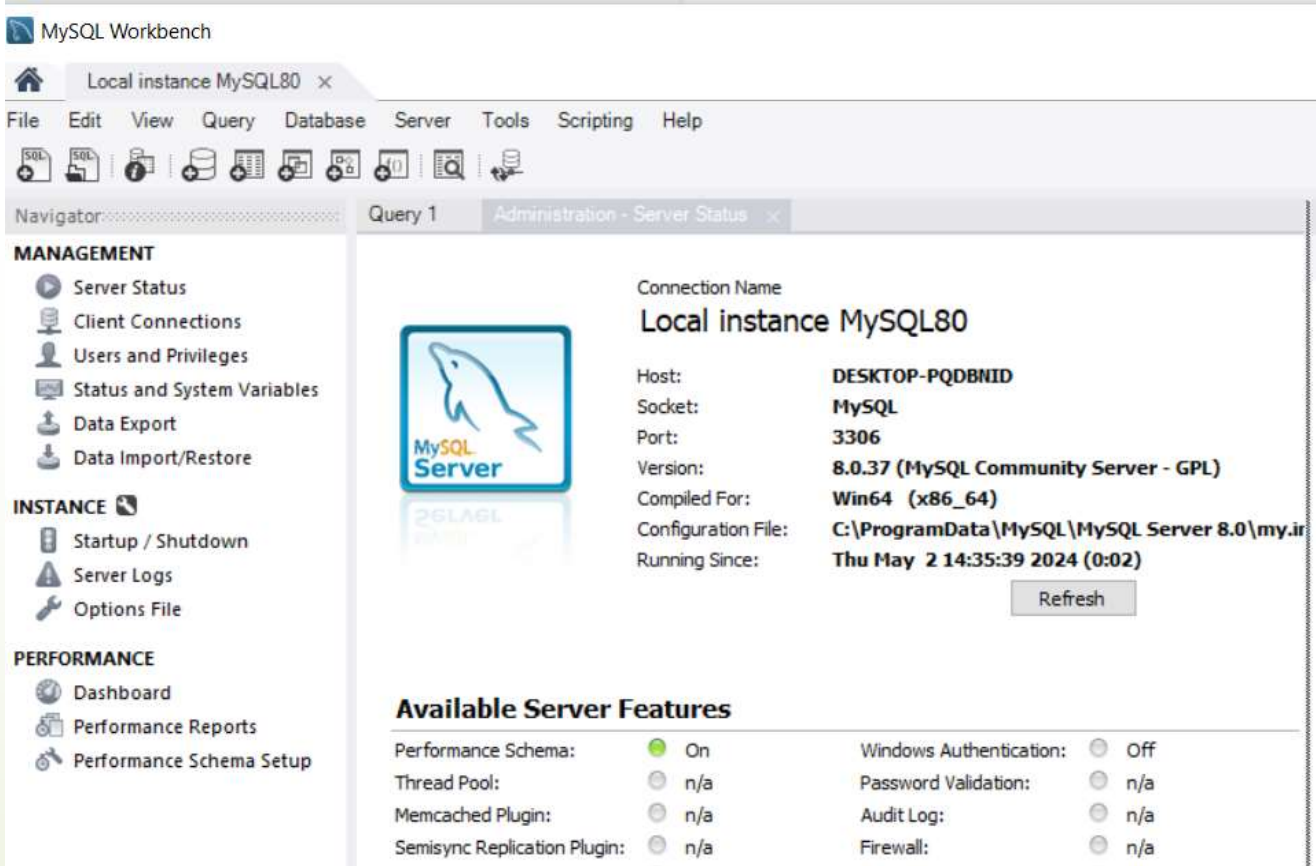
id	name	email
1	Sadık	sadik@gmail.com
2	Çınar	cinar@gmail.com
3	Emel	emel@gmail.com

Orders

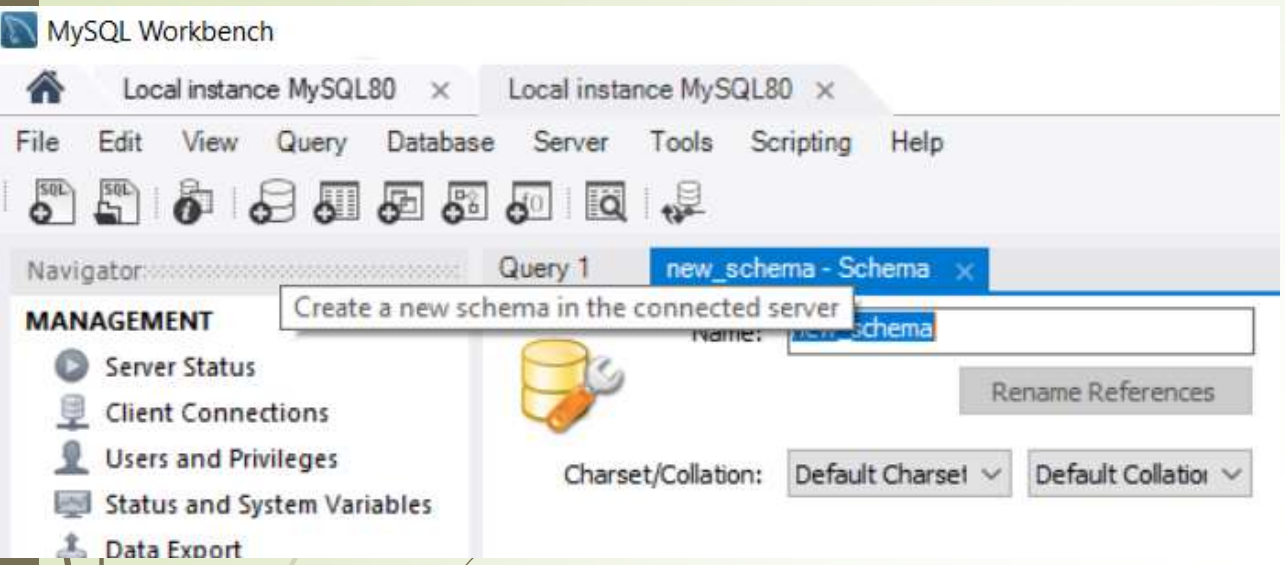
id	userid	productid
1	1	1
2	1	2
3	2	2

Her bir veri için veritabanında bir kimlik (ID), isim ve bir dizi tanım atamalıyız. Bu, sistemi daha karmaşık halden daha anlaşılabilir bir hale getirecektir.

MySQL'in son sürümünü ve Workbench'ini indirdikten sonra, uygulamaya kaydettiğimiz parolayı girerek sunucumuzu aktif hale getiriyoruz.



MySQL veri girme



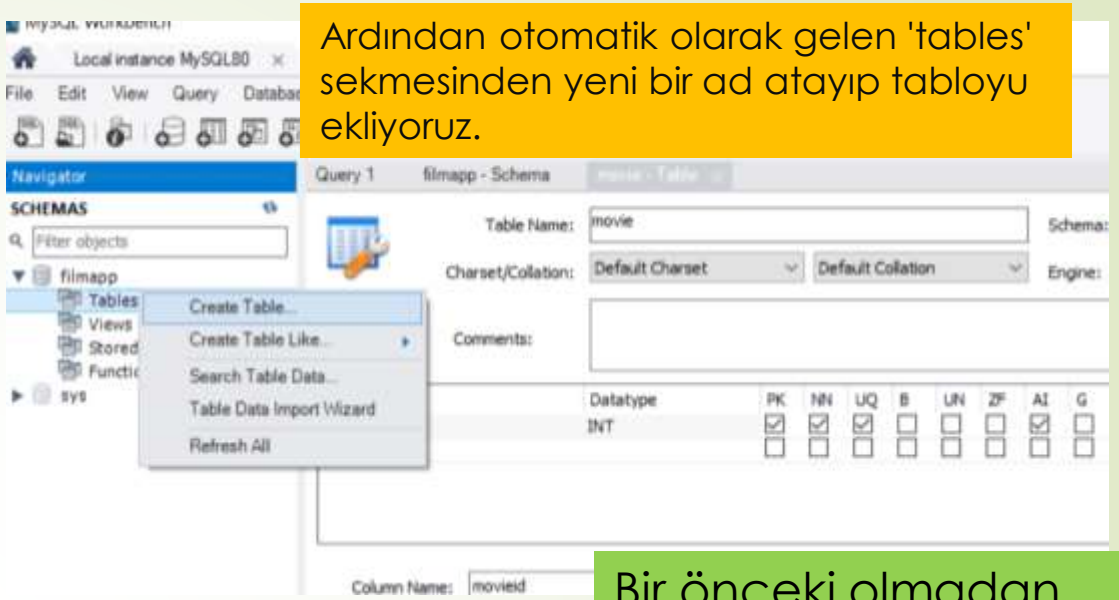
Bu simgeye tıklayarak kendimize yeni bir şema ekliyoruz.

Mysql'i **`npm i mysql2`** komutuyla indirebilirsiniz.

```
RR! 404 tarball, folder, http url, or git url.

RR! A complete log of this run can be found in: C:\
Users\Berat\Desktop\Yeni klasör> npm i mysql2

11 packages, and audited 123 packages in 2s
```

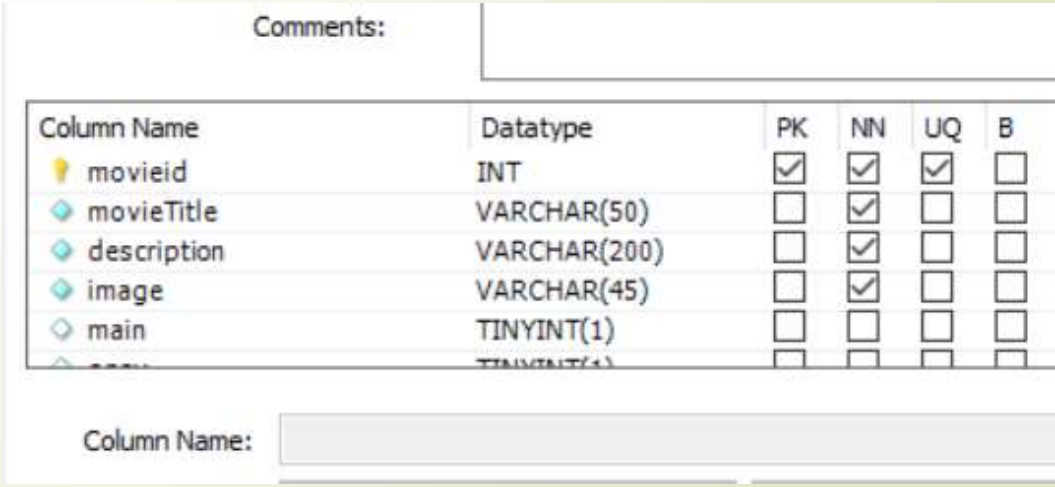


Ardından otomatik olarak gelen 'tables' sekmesinden yeni bir ad atayıp tabloyu ekliyoruz.

Gelen verileri ID'lerinin tamamen eş olacak şekilde giriyoruz.

Bir önceki olmadan diğerini kabul etmeyeceğimiz veri

Ilk data



Boolen yapı

```
const mysql= require('mysql2');
const config = require('./config')
let connection = mysql.createConnection(config.db)

connection.connect( function(err){
    if(err){
        return console.log(err);
    }

    connection.query("select * from movie", function(err, result){
        console.log(result[0].baslik)
    })

    console.log("MySQL server bağlantısı yapıldı");
})

app.use(express.static('node_modules'));
app.use("/static",express.static(path.join(__

app.use("/admin", adminRoutes);
app.use(userRoutes);

app.listen(8080, ()=>{
    console.log('Listenin on port 8080')
})
```

```
const config = {
    db: {
        host: "localhost",
        user: "root",
        password: [redacted]
        database: "filmapp"

    }
}

module.exports = config;
```

Mysql2'yi indirdikten sonra, projenizin içinde **config** adında bir dosya oluşturarak MySQL bağlantı bilgilerini tanımlıyoruz. Ardından, **connection.query** metodu kullanılarak MySQL Workbench'ten gelen bilgiler, bu **config** dosyasına girdiğimiz bilgilerle ulaştırılıyor.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

    movieTitle: 'Good Fellas',
    description: "Sıkı Dostlar, 1990 yılında çekilen, Martin Scorsese'nin yönettiği, Nicholas Pileggi'nin Wiseguy adlı kitabından uyarlanmış, Henry Hill adında bir gangsteri
ir.",
    image: 'c.jpg',
    main: 1,
    onay: 1
  },
```

```
});

router.use("/movies",(req,res)=>{
    db.execute('select * from movie')
    .then(result =>{
        console.log(result[0]);
        res.render('users/movies', {
            title: "Tüm Filmler",
            movies: result[0],
            categories: data.categories
        });
    }).catch(err => console.log(err))
});

router.use("/", function(req,res){
    db.execute('select * from movie')
    .then(result =>{
        console.log(result[0]);
        res.render('users/index', {
            title: "Popüler Kültür",
            movies: result[0],
            categories: data.categories
        });
    }).catch(err => console.log(err))
```

Gelen verilere, istediğimiz dosyaların görünmesini istediğimiz yerlere, genellikle EJS (Embedded JavaScript) kullanarak yerleştiriyoruz. İstenilen bilginin alındığı yerlerde, genellikle **req** ve **res** parametreleri ile çalışarak, bu bilgileri execute ediyoruz.

```
            <%- include('../partials/categoryMenu') %>
        </div>
        <div class="col-md-9">
            <h3><%=title%></h3>
            <% movies.forEach(b => { %>
                <% if (b.main) { %>
                    <%- include('../partials/movieCards', b) %>
                <% } %>
            <% }) %>
        </div>
    </div>
```

Istenilen bilgi

```
.use("/", function(req,res){
.execute('select * from movie where onay=1 and main=1')
hen(result =>{
  console.log(result[0]);
```

Bu koşulları yazdıktan sonra if bloğuna ihtiyacımız kalmaz.

```
<% movies.forEach(b => { %>
    <% if (b.main) { %>
        <%- include('../partials/movieCards', b) %>
    <% } %>
<% }) %>
/div>
```

Categoryden aldığımız name bilgisini kesinlikle yazmamız gerek

MySQL içinde yeni oluşturduğumuz ***category*** *tablosuna veri girmek için* ***db.execute*** *yöntemini kullanıyoruz. Bu yöntemle, veritabanında belirli bir sorguyu veya komutu çalıştırabiliriz. Örneğin,* ***INSERT INTO category (column1, column2, ...) VALUES (value1, value2, ...)*** *şeklinde bir SQL sorgusu kullanarak yeni verileri ekleyebiliriz.*

SQL'de filtreleme için **WHERE** koşullarını kullanırız. Birden fazla koşul varsa, bunları **AND** veya **OR** operatörleriyle birleştirebiliriz. Örneğin, **WHERE** koşullarını kullanarak bir sorgu oluştururken, bir koşulun yanı sıra başka bir koşulu da belirleyebiliriz.

Örnekler:

- **`WHERE koşul1 = 1 AND koşul2 = 'değer'`**: Koşul1'in 1'e eşit ve Koşul2'nin belirli bir değere eşit olduğu durumları getirir.
- **`WHERE koşul1 = 1 OR koşul2 < 50`**: Koşul1'in 1'e eşit olduğu veya Koşul2'nin 50'den küçük olduğu durumları getirir.

Bu şekilde, birden fazla koşulu **AND** veya **OR** operatörleriyle birleştirerek sorgumuzu oluşturabiliriz.

```
db.execute("select * from blog
    .then(result => {
        db.execute("select * fr
            .then(sonuc => {
                res.render("use
                    title: "Tüm
                    blogs: resu
                    categories:
                })
            })
            .catch(err => conso
        })
        .catch(err => console.l
    });
```

```
56          const category = await db.execute('select * from category')
57          res.render('users/movies', {
58              title: "Tüm Filmler",
59              movies: movieCard[0],
60              categories: category[0],
61          });
62      } catch (error) {
63          console.log(error)
64      }
65  });
66
67  router.use("/", async(req,res)=>{
68      try {
69          const movieCard = await db.execute('select * from movie where
            onay=1 and main=1')
70          const category = await db.execute('select * from category')
71          res.render('users/movies', {
72              title: "Tüm Filmler",
```

kısa bi şekilde yazımını yapabiliriz

Ve artık veriye ihtiyacımız kalmadığında, onu silmek için veritabanından ilgili kaydı kaldırabiliriz.

```
6   const db = require('../data/db');
7
8
9   const data = {
10      title: "Popüler Kültür",
11      categories : ["Kült Filmler", "Dram Filmler", "Absürt Fi
12      movies: [
13          {
14              movieid: 1,
15              movieTitle: "The Gotfather",
16              description: "Baba, Mario Puzo'nun yazdığı aynı
                Pacino'nun başrollerini paylaştığı filmdir.",
17              image: "a.jpg",
18              main: true,
19              onay: true
20          },
21          {
```

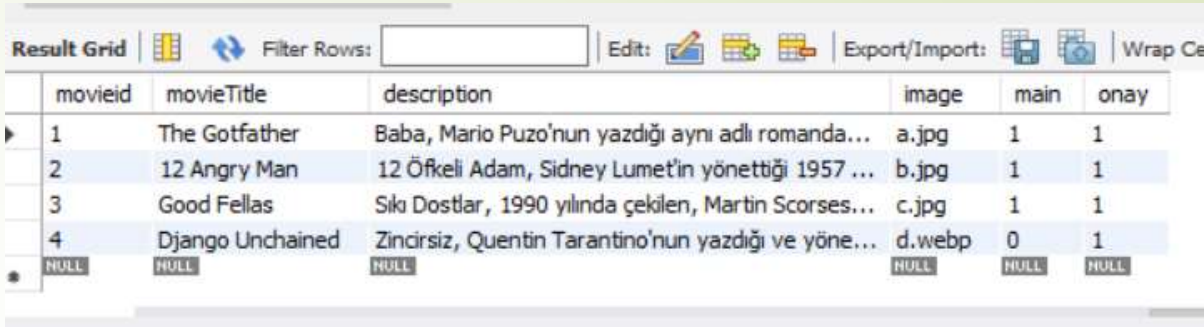
Detay Sayfaları

```
router.use("/movies/:movieid",async(req,res)=>{
    const id = req.params.movieid;
    try {
        const [movie, ] = await db.execute('select * from movie where movieid=?',[id])

        if(movie[0]){
            res.render('users/movieDetails',{
                title: movie[0].movieTitle,
                movie: movie[0]
            })
        }
        res.redirect('/')

    } catch (error) {
        console.log(error)
    }
```

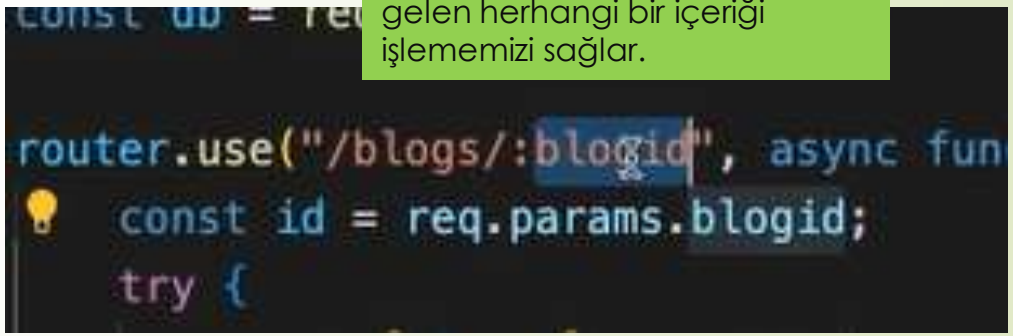
Detay sayfasını **try-catch** bloğu içinde, **id** parametresiyle yazmalıyız. Bu, genellikle bir URL'den veya bir istekten gelen bir parametredir ve belirli bir kaydın detaylarını almak için kullanılır. **try-catch** bloğu, olası hataları yönetmek için kullanılır.



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Ce

movieid	movieTitle	description	image	main	onay
1	The Gotfather	Baba, Mario Puzo'nun yazdığı aynı adlı romanda...	a.jpg	1	1
2	12 Angry Man	12 Öfkeli Adam, Sidney Lumet'in yönettiği 1957 ...	b.jpg	1	1
3	Good Fellas	Sıkı Dostlar, 1990 yılında çekilen, Martin Scorses...	c.jpg	1	1
4	Django Unchained	Zincirsiz, Quentin Tarantino'nun yazdığı ve yöne...	d.webp	0	1
NULL	NULL	NULL	NULL	NULL	NULL

Slash'ten sonra **params** yapısına girdiğimiz herhangi bir içerik, detay sayfasında kullanılabilir. Bu, isteğin URL'sindeki parametrelerden gelen herhangi bir içeriği işlememizi sağlar.

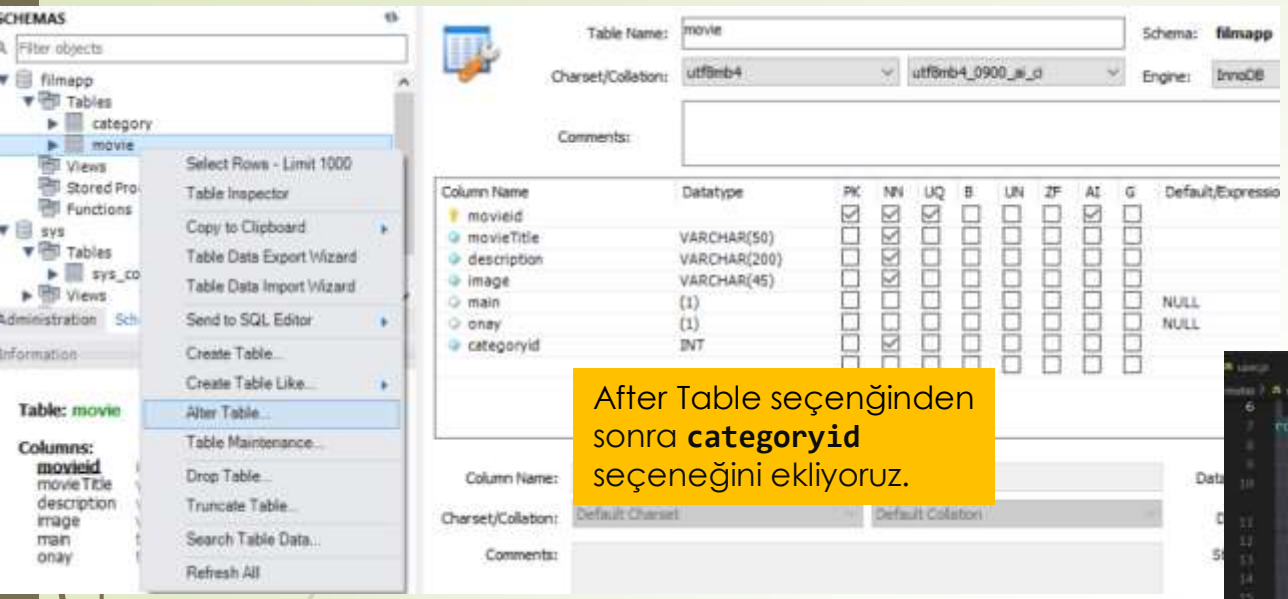


```
router.use("/blogs/:blogid", async fun
    const id = req.params.blogid;
    try {
```

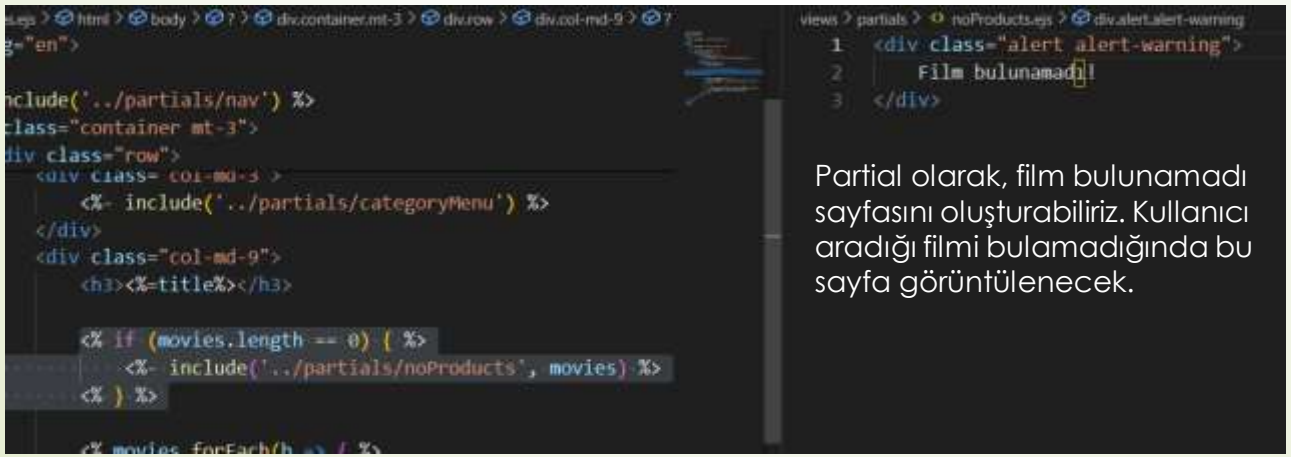
```
<!DOCTYPE html>
<html lang="en">
<head>
    <%- include('../partials/head') %>
</head>
<body>
    <%- include('../partials/nav') %>

    <div class="container mt-3">
       <div class="card">
        <div class="card-body">
            <div class="row">
                <div class="col-md-4">
                    <img src="/static/images/<%=movie.image%>" class="img-fluid" alt="">
                </div>
                <div class="col-md-8">
                    <h3><%=movie.movieTitle%></h3>
                    <p><%= movie.description %></p>
                </div>
            </div>
```

detay içeriğini düzenliyoruz



After Table seçeğinden sonra **categoryid** seçeneğini ekliyoruz.



Partial olarak, film bulunamadı sayfasını oluşturabiliriz. Kullanıcı aradığı filmi bulamadığında bu sayfa görüntülenecek.

Kategori için ayrı oluşturduğumuz router'a **use** yöntemiyle gerekli alanları dolduruyoruz. Ayrıca, kategori menüsünü değiştirilebilir hale getirmek için **selectedCategory**'yi **categoryId**'ye eşitliyoruz. Bu sayede, kullanıcı farklı bir kategori seçtiğinde, **selectedCategory** değeri güncellenir ve ilgili kategori içeriği yüklenir.