



Node.js ile Web Programlama Rapor - 14

Mahire Zühal Özdemir

İÇİNDEKİLER

ŞEKİLLER DİZİN	3
HTTP ENDPOINT NEDİR?	4
POSTMAN	4
Veritabanı Bağlantısı ve Gerçek Proje Uygulaması	7
1. index.js Dosyası İçerisinde express Başlatarak Belirtilen url'e Postman ile Ulaşmak.....	7
2. Post İşlemi İle Veri Göndermek	8
3. Geçen Hafta Oluşturduğumuz Dosyaları Projeye Entegre Etme ve Klasör Yapısını Güncellemek.....	9
4. Veritabanına Belge Eklemek.....	11
5. Koleksiyondaki tüm belgeleri getirelim.	12
6. Spesifik Bir Özelliğe Göre Koleksiyonda Arama Yapma.....	13
7. ID değerine göre koleksiyon üzerinde arama yapma	14

ŞEKİLLER DİZİNİ

Şekil 1: Postman Kurulum	4
Şekil 2: Development Dependency Ekleme	5
Şekil 3: Express Modülü Yükleme	5
Şekil 4: Script Default Test	5
Şekil 5: Scripti Çalıştırma	6
Şekil 6: Script'i Çalıştırma Çıktı	6
Şekil 7: index.js dosyası	7
Şekil 8: index.js çıktısı	7
Şekil 9: Post Test	8
Şekil 10: Postman Çıktısı	8
Şekil 11: App.use Komutu	8
Şekil 12: Postman İstek Body	9
Şekil 13: Konsol İstek Body	9
Şekil 14: Proje Klasör Yapısı	9
Şekil 15: Task.js dosyası	10
Şekil 16: User.js	10
Şekil 17: Mongoose.js	10
Şekil 18: Import Task ve User	10
Şekil 19: users koleksiyonuna belge ekleme	11
Şekil 20: Postman Veri Ekleme	11
Şekil 21: MongoDB Belge Kontrolü	11
Şekil 22: Post - Task.js	12
Şekil 23: Task Ekleme	12
Şekil 24: find Metodu- Task	13
Şekil 25: Find Metodu Çıktı	13
Şekil 26: Find Metodu - Spesifik	14
Şekil 27: Find Çıktısı Completed true	14
Şekil 28: findById Metodu	14
Şekil 29: FindById Çıktısı	15

HTTP ENDPOINT NEDİR?

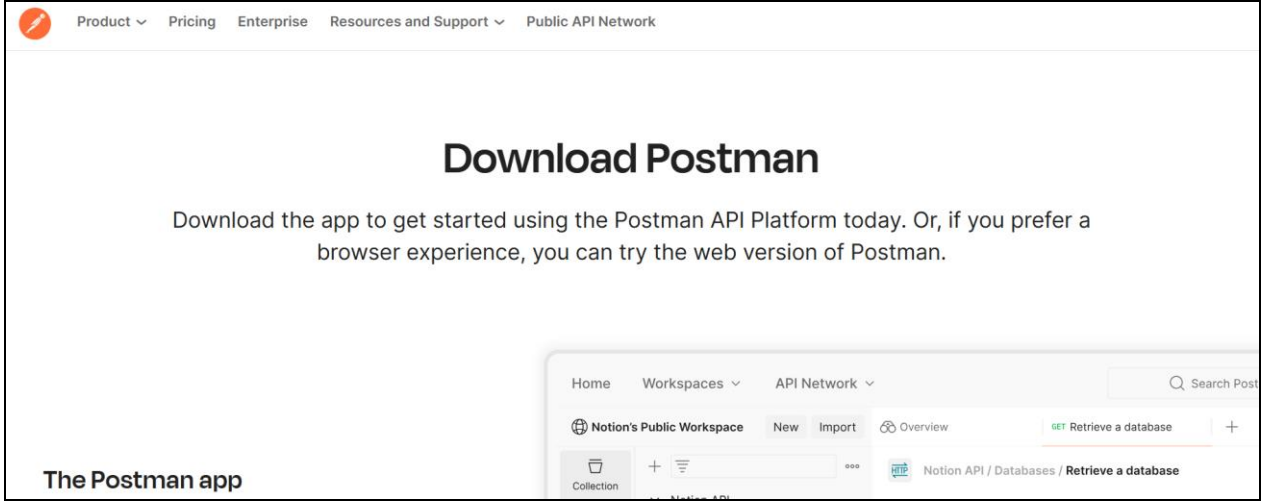
HTTP endpoint, genellikle bir web hizmeti veya uygulamasının dış dünyaya erişim sağlamak için sunduğu bir adres veya noktadır.

HTTP (Hypertext Transfer Protocol), web tarayıcıları ve sunucular arasındaki iletişim için kullanılan bir protokoldür. Bir HTTP endpoint, bir sunucu üzerindeki belirli bir kaynağa erişmek veya bir işlemi gerçekleştirmek için belirli bir URL'ye (Uniform Resource Locator) sahip olan bir noktadır.

POSTMAN

Postman, API'leri test etmek ve geliştirmek için kullanılan popüler bir uygulamadır. Ayrıca, Postman'ın bir npm modülü de vardır, bu da Postman'ı Node.js projelerinde kullanmanızı sağlar.

İşletim sisteminizi seçerek [postman web sitesinden](#) size uygun olan paketi yükleyebilirsiniz.



Şekil 1: Postman Kurulum

Postman'ın npm modülünü projemize development dependency (geliştirme bağımlılığı) olarak eklemek için aşağıdaki komut çalıştırılır.

```
\Hafta-14> npm i nodemon --save-dev
```

Şekil 2: Development Dependency Ekleme

(Development dependency olarak eklemek, bu modülün yalnızca geliştirme ortamında kullanılacağını belirtir ve üretim ortamında gerekli olmadığını gösterir.)

Express modülünü projeye yüklemek için aşağıdaki komut çalıştırılır.

```
\Hafta-14> npm i express
```

Şekil 3: Express Modülü Yüklemek

'package.json', Node.js projelerinde proje bilgilerini ve bağımlılıkları yönetmek için kullanılan bir dosyadır. Bu dosya aynı zamanda npm script'lerini tanımlar.

Package.json dosyasında bulunan script kısmındaki test işlemini siliyoruz.

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

Şekil 4: Script Default Test

Yerine start ve dev alanlarını ekleyerek projeyi development olarak geliştiriyoruz.

```
"scripts": {  
  "start": "node src/index.js",  
  "dev": "nodemon src/index.js"  
},
```

Neden bu işlemi yaptık:

- Varsayılan olarak bulunan test script'i, genellikle "Test tanımlanmadı" mesajı vererek çıkış yapar.

- Geliştirme sürecinde genellikle test script'i yerine uygulamayı başlatma ve geliştirme modunda çalıştırma script'leri kullanılır.
- **start script'i**, projenizi başlatmak için kullanılır. Bu komut genellikle Node.js projelerinde uygulamanın ana dosyasını çalıştırır.
- **node src/index.js**; src/index.js dosyasını çalıştırarak uygulamanızın başlangıç noktasını tanımlar.
- **dev script'i**, geliştirme sırasında kullanılan bir komuttur. Nodemon ile birlikte kullanılarak, kod değişikliklerini izler ve kod değiştiği anda uygulamayı otomatik olarak yeniden başlatır.

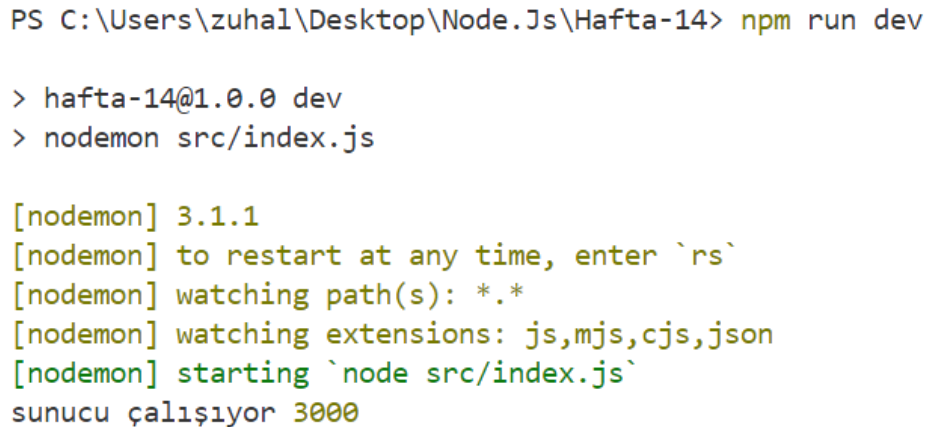
Scripti başlatmak için aşağıdaki komut çalıştırılır.



```
Hafta-14> npm run dev
```

Şekil 5: Scripti Çalıştırma

npm run dev, Node.js projelerinde package.json dosyasında tanımlı olan dev script'ini çalıştırmak için kullanılan bir komuttur. Bu komut, geliştirme sırasında sıkça kullanılır. Kod değişikliklerinde güncellemeyi sağlar.



```
PS C:\Users\zuhal\Desktop\Node.Js\Hafta-14> npm run dev

> hafta-14@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
sunucu çalışıyor 3000
```

Şekil 6: Script'i Çalıştırma Çıktı

Veritabanı Bağlantısı ve Gerçek Proje Uygulaması

Projemizi geliştirmek için geçen hafta yazmış olduğumuz kodları kullanacağız. Projeyi adım adım uygulayabiliriz.

1. index.js Dosyası İçerisinde express Başlatarak Belirtilen url'e Postman ile Ulaşmak

index.js dosyasını aşağıdaki gibi dolduruyoruz. Express uygulaması oluşturarak portu dinliyoruz.

```
const express = require('express')
const port = process.env.PORT || 3000
//bulutta çalışan bir proje için port 3000 tanımlanamaz
//port bulut tarafından atanan port olarak belirlenir.
// bu yüzden development aşamasında process tarafından atanan port kullanılır

const app = express()

app.listen(port,()=>{
  console.log('sunucu ${port} portunda çalışıyor')
})
```

Şekil 7: index.js dosyası

npm run dev ile kodu çalıştırdığımızda çıktı aşağıdaki gibi olacaktır.

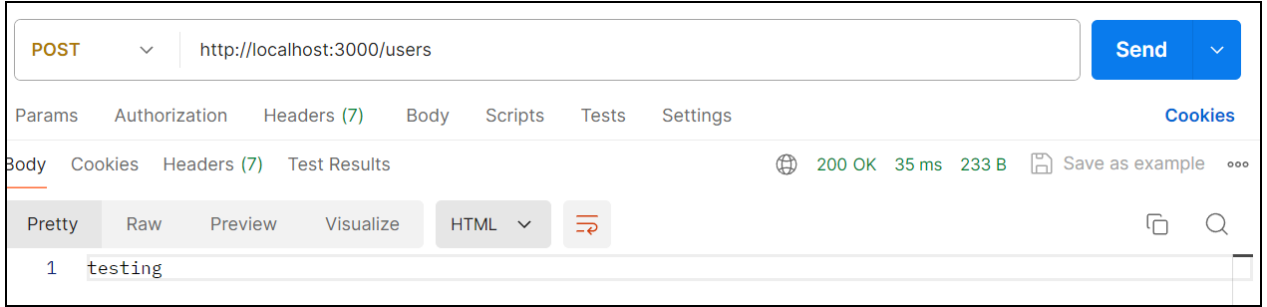
```
[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
sunucu 3000 portunda çalışıyor
```

Şekil 8: index.js çıktısı

Port bağlantısı sağlandıktan sonra index.js dosyasında aşağıdaki gibi post isteği oluşturarak postman uygulamasında açıyoruz.

```
app.post('/users',(req,res)=>{  
  res.send('testing')  
})
```

Şekil 9: Post Test



Şekil 10: Postman Çıktısı

Postman'e ilgili url'i girip, post metodunu seçtikten sonra send işlemi gerçekleştiriyoruz. Send ile gelen çıktıyı aşağıdaki alanda görüntülüyoruz.

2. Post İşlemi İle Veri Göndermek

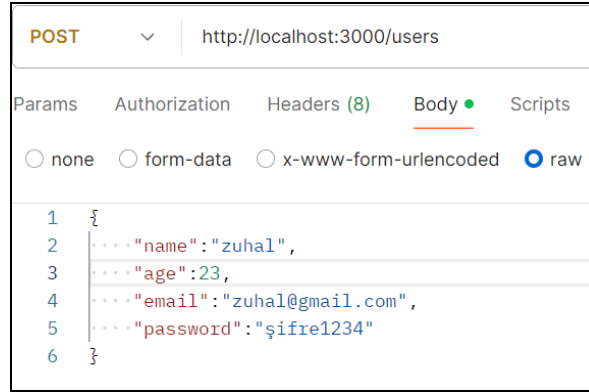
İlk olarak, json veriyi hatasız bir şekilde işlemek için `app.use` komutunu ekliyoruz.

`app.use(express.json())` ifadesi, bir Express uygulamasında gelen isteklerin gövde verisinin JSON formatında olduğunu belirtir. Bu ifade, Express uygulamasının JSON verileri işleyebilmesini sağlayan bir ara yazılım (middleware) olarak kullanılır.

```
app.use(express.json())  
app.post('/users',(req,res)=>{  
  console.log(req.body)  
})
```

Şekil 11: App.use Komutu

Gönderilen isteği görüntüleyebilmek için konsola yazdırıyoruz.



Şekil 12: Postman İstek Body

İsteği postman ile çalıştırdığımızda konsolda aşağıdaki metni elde ediyoruz.

```
sunucu 3000 portunda çalışıyor  
{  
  name: 'zuhal',  
  age: 23,  
  email: 'zuhal@gmail.com',  
  password: 'şifre1234'  
}
```

Şekil 13: Konsol İstek Body

3. Geçen Hafta Oluşturduğumuz Dosyaları Projeye Entegre Etme ve Klasör Yapısını Güncellemek

Önceden oluşturduğumuz dosyaları projeye entegre edip klasör yapısını aşağıdaki gibi güncelliyoruz.

```
proje_klasörü  
├─ src  
│   └─ index.js  
├─ db  
│   └─ mongoose.js  
└─ models  
    ├── user.js  
    └─ task.js
```

Şekil 14: Proje Klasör Yapısı

Task.js dosyasını geçen hafta oluşturduğumuz dosyadan aşağıdaki gibi alıyoruz.

```
const mongoose = require('mongoose')

const schema = new mongoose.Schema({description:{type:String,required:true},
  completed:{type:Boolean,default:false}})

const Task = mongoose.model('Task', schema)

module.exports = Task
```

Şekil 15: Task.js dosyası

User.js dosyasını geçen haftadan alarak her iki dosyada exports işlemlerini gerçekleştiriyoruz.

```
const User = mongoose.model('User', schema)

module.exports = User
```

Şekil 16: User.js

Mongoose.js dosyasında aşağıdaki bilgilerin kalması bizim için yeterli.

```
const mongoose = require('mongoose')

const uri = "mongodb+srv://[ ]:mongodb.net/mydb";
//dbName uri sonuna eklenebilir

mongoose.connect(uri)
```

Şekil 17:Mongoose.js

Index.js dosyası içerisine ilgili dosyaları import ediyoruz.

```
const mongoose = require('../db/mongoose')
const User = require('../models/user')
const Task = require('../models/task')
```

Şekil 18: Import Task ve User

Artık index.js dosyamızda user ve task ile veritabanına CRUD işlemleri yapabiliriz.

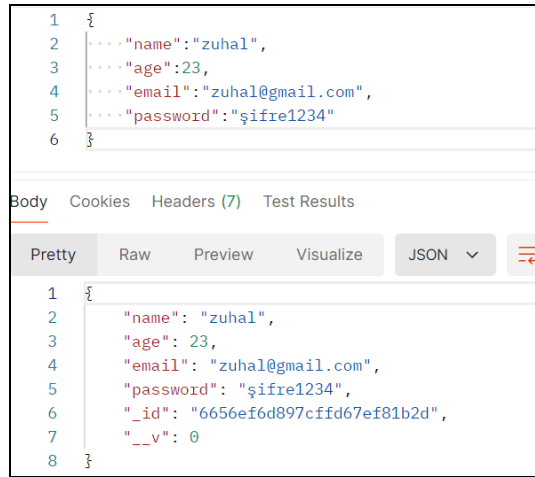
4. Veritabanına Belge Ekleme

Post işlemini aşağıdaki gibi güncelledikten sonra postman ile isteğimizi gönderiyoruz.

```
app.post('/users', (req, res) => {  
  const user = new User(req.body)  
  user.save().then(() => { res.status(201).send(user) })  
    .catch((e) => { res.status(400).send(e) })  
})
```

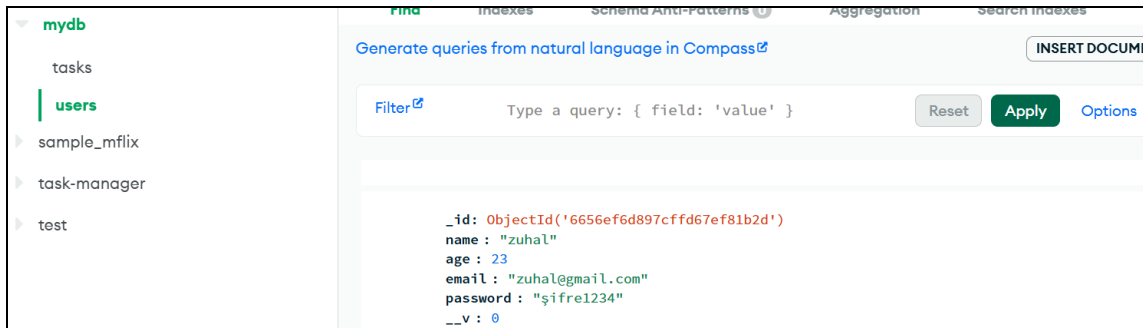
Şekil 19: users koleksiyonuna belge ekleme

Postman ile isteğimizi json formatında gönderiyoruz. Ekleme işlemi başarılı olduğu durumda json verisini yazdırıyoruz. Hata durumunda hata kodlarıyla hatayı yazdırıyoruz.



Şekil 20: Postman Veri Ekleme

Postman ile istek çıktısı yazdırıldı. MongoDB üzerinden veritabanını kontrol ederek verimizin eklendiğini teyit edebiliriz.

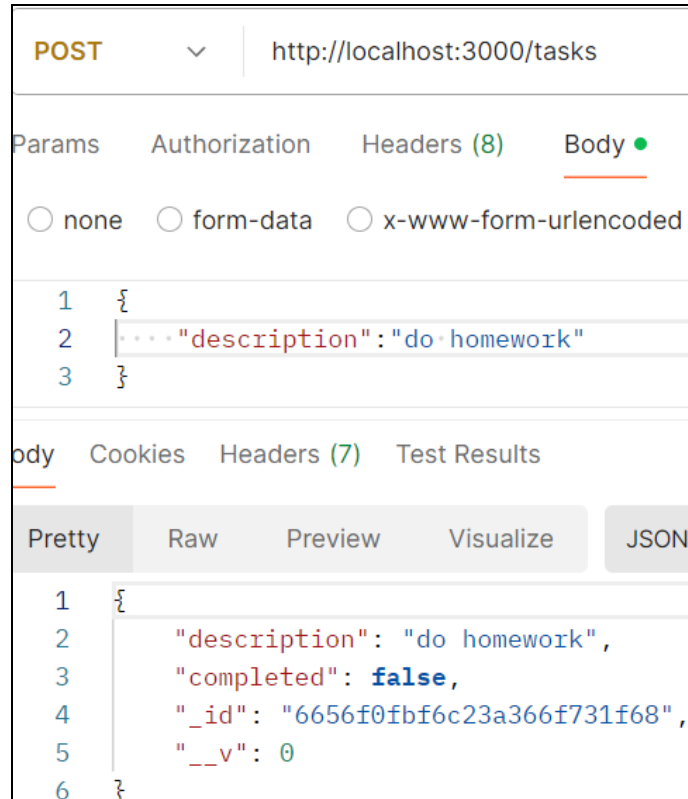


Şekil 21: MongoDB Belge Kontrolü

Aynı işlemi task.js için de yapabiliriz.

```
app.post('/tasks', (req, res) => {
  const task = new Task(req.body)
  task.save().then(() => {res.status(201).send(task)})
    .catch((e) => {res.status(400).send(e)})
})
```

Şekil 22: Post - Task.js



Şekil 23: Task Ekleme

Task.js dosyasında completed parametresinin default olarak false atandığı unutulmamalı.

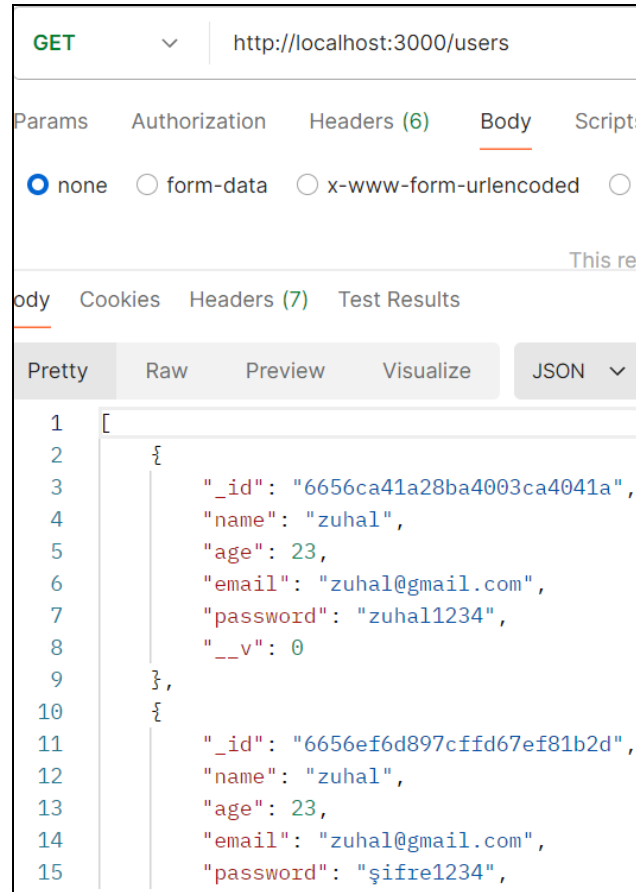
5. Koleksiyondaki tüm belgeleri getirelim.

!!! get metodu kullanıldığına dikkat edin.

Koleksiyondaki tüm belgelere erişmek için find metodu kullanılır. Çıktı olarak bir dizi alınır. Diziyi aşağıdaki gibi then ve catch blokları ile yazdırabiliriz.

```
app.get('/users', (req, res) => {
  User.find({}).then((users) => {
    res.send(users)
  }).catch((e) => {
    res.status(400).send(e)
  })
})
```

Şekil 24: find Metodu- Task



Şekil 25: Find Metodu Çıktı

Aynı işlemi task için de uygulayabiliriz.

6. Spesifik Bir Özelliğe Göre Koleksiyonda Arama Yapma

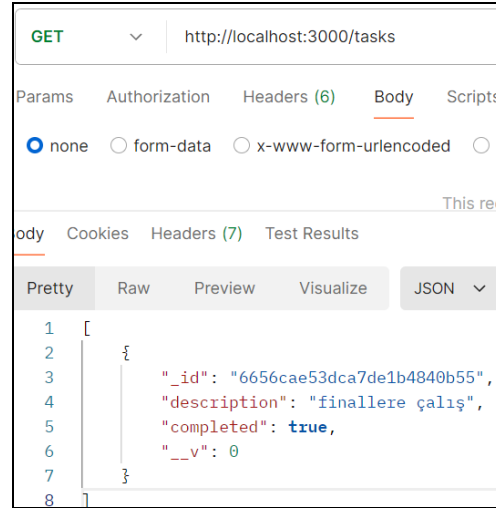
Task koleksiyonu içerisinde `completed:true` olan task'leri görüntülemek için `get` ve `find` metodunu aşağıdaki gibi düzenliyoruz.

```

app.get('/tasks',(req,res)=>{
  Task.find({completed:true}).then((tasks)=>{
    res.send(tasks)
  }).catch((e)=>{
    res.status(400).send(e)
  })
})

```

Şekil 26: Find Metodu - Spesifik



Şekil 27: Find Çıktısı Completed true

7. ID değerine göre koleksiyon üzerinde arama yapma

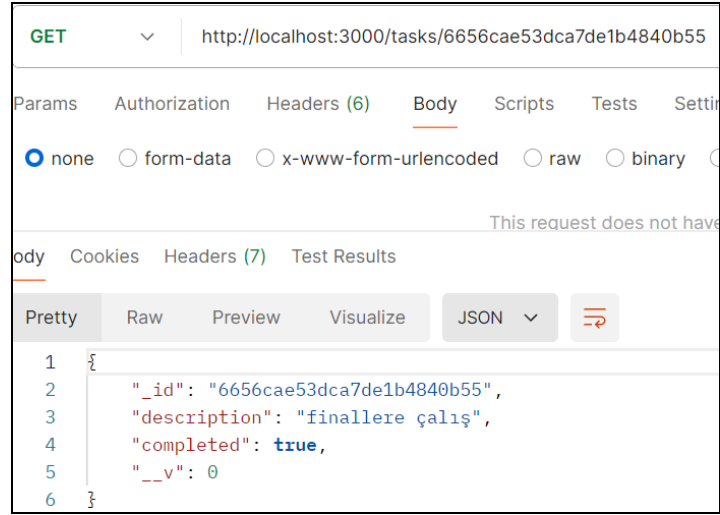
```

app.get('/tasks/:id',(req,res)=>{
  //id _ ile tanımlanmalı
  const _id= req.params.id
  Task.findById({_id}).then((task)=>{
    if(!task){
      return res.status(404).send()
    }
    res.send(task)
  }).catch((e)=>{
    res.status(500).send(e)
  })
})

```

Şekil 28: findById Metodu

!!! id değişkeninin `_id` olarak tanımlanması gerektiğini unutmayın.



Şekil 29: FindById Çıktısı