WEB TEKNOLOJİLERİ ve PROGRAMLAMA

8. Ders: REST API Yazma

SÜLEYMAN DEMİREL ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

Sunum Planı

01 REST API Kuralları

02 Express ile API Oluşturma

03 GET Metotları: MongoDB'den Veri Okuma

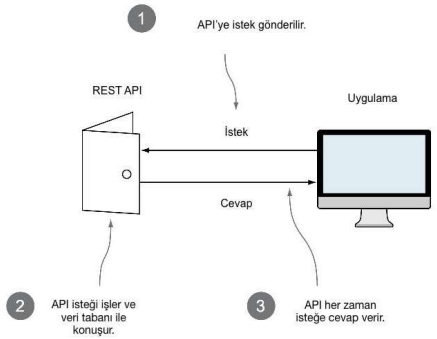
04 Ödev

01 REST API Kuralları

- Şu aşamada veri tabanımız hazır. Ancak veri tabanımızla sadece MongoDB Shell aracılığıyla etkileşim kurduk.
- Bu derste REST API oluşturarak veri tabanımızla HTTP çağrıları aracılığı ile etkileşimde bulunacağız.
- API ile Create (Oluşturma), Read (Okuma), Update (Güncelleme), Delete(Silme) yani **CRUD** fonksiyonlarını gerçekleştireceğiz.
- **REST:** REpresentational State Transfer. Katı kurallara sahip bir protokolden ziyade bir mimari stildir. Durumsuzdur (stateless). Yani o anki kullanıcı durumu ya da geçmişinden habersizdir.
- **API:** Application Programming Interface. Uygulamaların birbirleriyle konuşmasını sağlar.
- **REST API:** Uygulamanız için durumsuz bir arayüz sunar. MERN yığınında ise veri tabanınız için durumsuz bir arayüz oluşturmanızı sağlar. Diğer uygulamaların veriyle çalışmasını sağlar.

01 REST API Kuralları

- REST API'nin bazı standartları vardır.
  - API'ye istek gönderilir.
  - API isteği işler ve veri tabanı ile konuşur.
  - API **her zaman** istekte bulunana cevap gönderir.



01 REST API Kuralları

- REST API isteklere basit standartları olan Request URL (İstek URL) 'leri aracılığı ile cevap verir.
- Sahip olduğunuz URL'ler ile belli başlı işlere destek vermeniz gerekir. Bunlar genelde CRUD işlemleridir.
  - Yeni bir öğe oluşturmak. (Yeni bir mekan oluşturmak)
  - Öğelerin listesini okumak. (Mekanların listesini okumak)
  - Belirli bir öğeyi okumak. (Belirli bir mekan bilgisini okumak)
  - Belirli bir öğeyi güncellemek. (Belirli bir mekanı güncellemek)
  - Belirli bir öğeyi silmek. (Belirli bir mekanı silmek)
- Kendi uygulama örneğimizi düşünecek olursak oluşturacağımız URL'ler ve alacağı parametreler nasıl görünür?

01 REST API Kuralları

İşlem	URL Yolu	Parametreler	Örnek
Yeni mekan oluştur	/api/mekanlar		http://localhost:3000/api/mekanlar
Mekanları oku	/api/mekanlar		http://localhost:3000/api/mekanlar
Belirli bir mekanı oku	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123
Belirli bir mekanı güncelle	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123
Belirli bir mekanı sil	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123

Her bir işlem aynı URL yoluna sahip. 3 tanesi bir mekanı belirlemek için aynı parametreyi alıyor. Farklı işlemler yapmak için aynı URL'yi nasıl kullanacağız? **Cevap istek metotlarında yatıyor.**

01 REST API Kuralları

- HTTP istekleri sunucuya hangi tür işlem yapılacağını söyleyen farklı metotlara sahiptir.
- Bu isteklerin en bilinen türü GET isteğidir.
  - Tarayıcınıza bir URL yazıp Enter tuşuna bastığınızda kullanılan metottur.
- Diğer çok sık kullanılan tür ise POST isteğidir.
  - Bir forma veri girip Gönder tuşuna bastığınızda kullanılan metot.
- 4 temel metot vardır.

İstek Metodu (Request)	Kullanımı	Cevap (Response)
POST	Veri tabanında yeni bir veri oluşturur.	Yeni eklenen veri
GET	Veri tabanından veri okur.	Okunan veri
PUT	Veri tabanında veri günceller.	Güncellenmiş veri
DELETE	Veri tabanından veri siler.	Null

01 REST API Kuralları

- Gördüğünüz gibi her bir CRUD işlemi için farklı metotlar kullanılmaktadır.
- İyi tasarlanmış bir REST API farklı istekler için her zaman aynı URL'ye sahiptir.
- Hangi işlemin yapılacağını sunucuya bildiren ise kullanılan metottur. Kendi uygulamamıza uyarlayacak olursak:

İşlem	Metot	URL Yolu	Parametreler	Örnek
Yeni mekan oluştur	POST	/api/mekanlar		http://localhost:3000/api/mekanlar
Mekanları listele	GET	/api/mekanlar		http://localhost:3000/api/mekanlar
Belirli bir mekanı oku	GET	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123
Belirli bir mekanı güncelle	PUT	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123
Belirli bir mekanı sil	DELETE	/api/mekanlar	mekanid	http://localhost:3000/api/mekanlar/123

01 REST API Kuralları

- Peki ya alt dokümanlar için nasıl olacak? Uygulamamızı düşünecek olursak yorumlar var.

İşlem	Metot	URL Yolu	Parametreler	Örnek
Yeni yorum oluştur	POST	/api/mekanlar/mekanid/yorumlar	mekanid	http://localhost:3000/api/mekanlar/123/yorumlar
Belirli bir yorumu oku	GET	/api/mekanlar/mekanid/yorumlar	mekanid yorumid	http://localhost:3000/api/mekanlar/123/yorumlar/456
Belirli bir yorumu güncelle	PUT	/api/mekanlar/mekanid/yorumlar	Mekanid yorumid	http://localhost:3000/api/mekanlar/123/yorumlar/456
Belirli bir yorumu sil	DELETE	/api/mekanlar/mekanid/yorumlar	Mekanid yorumid	http://localhost:3000/api/mekanlar/123/yorumlar/456

- Görüldüğü üzere "yorumların listesini getir" işlemi yok. Çünkü ana dokümanın bir parçası olarak getirilecek.

01 REST API Kuralları

- Başarılı bir REST API için istek formatını standartlaştırmak ne kadar önemliyse cevapları standartlaştırmak da o kadar önemlidir.
- Cevabın iki önemli anahtar bileşeni olmalıdır.
  - Döndürülen veri
  - HTTP durum kodu.
- Bu iki bileşeni birleştirerek istemciye sunmak gerekir.
- API tutarlı bir veri formatında cevabı döndürmelidir. REST API için formatlar XML ya da JSON olabilir.
- MERN yığınına çok güzel uyum sağladığı için biz JSON formatını kullanacağız.
- API'miz her bir istek için şu 3 şeyden birini döndürecek.
  - İsteği cevapladığında dönen veriyi tutacak JSON nesnesi
  - Hata verisini içerecek JSON nesnesi
  - Null cevabı

01 REST API Kuralları

- HTTP Durum Kodları: https://tr.wikipedia.org/wiki/HTTP_durum_kodları
- En popüler 10 HTTP durum kodu:

Durum Kodu	İsim	Kullanımı
200	OK (Tamam)	Başarılı GET, PUT isteği
201	Created (Oluşturuldu)	Başarılı POST isteği
204	No content (İçerik yok)	Başarılı DELETE isteği
400	Bad request (Kötü istek)	Uygun olmayan içerikten dolayı başarısız GET, PUT, POST isteği
401	Unauthorized (Yetkisiz)	Yanlış kimlik bilgileri ile kısıtlı bir URL'ye istekte bulunmak
403	Forbidden (Yasaklı)	İzin verilmeyen bir istekte bulunmak
404	Not found (bulunamadı)	URL'de yanlış parametre kullanımına dayalı başarısız istek
405	Method not allowed (Metoda izin verilmedi)	Verilen URL için istek metodu geçersiz
409	Conflict (Çakışma)	Aynı veri zaten varken başka bir POST isteği yollayarak isteğin başarısız olması
500	Internal Server Error (Sunucu hatası)	Veri tabanı ya da sunucuyla ilgili problem var

02 Express ile API Oluşturma

- Daha önceden controller'lar ve rotalar oluşturmuştuk. İstersek bunları kullanarak API'mizi oluşturabiliriz.
- Ancak en uygun ve önerilen yöntem API kodunu uygulamadan ayrıştırmak, uygulamamızı daha anlaşılır ve basit hale getirmektir.
- Bunun için API ile ilgili tüm kodları ayrı bir klasör içine yerleştireceğiz.
- Uygulama dizininin (kök dizin) içinde "app_server" klasörünün hemen yanında "app_api" adında bir klasör oluşturun.
- "app_api" klasörü API'mize ait rotalar, controller'lar ve modellere ait herşeyi barındıracak.
- Ana Express uygulamamızda yaptığımız gibi "app_api/routes" klasörü içinde tüm rotalarımızı tutatacak boş bir "index.js" dosyası oluşturun.
- Sonraki adım ana uygulama "app.js" içinde bunu tanıtmak.

02 Express ile API Oluşturma

- "app.js" içinde yapmamız gereken:
  - **var apiRouter= require('./app_api/routes/index');**
- Bir sonraki adım rotaları ne zaman kullanacağımızı belirtmek. Tüm API rotalarımız "/api/" ile başlayacak. Bunu app.js içinde belirtmeliyiz (**var app = express();** komutundan sonra yazılacak).
  - **app.use('/api',apiRouter);**

- Şimdi tüm rotalarımızı "index.js" içinde tanımlayalım: **Mekanlar için**

```
router
.route('/mekanlar')
.get(ctrlMekanlar.mekanlariListele)
.post(ctrlMekanlar.mekanEkle);
```

```
router
.route('/mekanlar/:mekanid')
.get(ctrlMekanlar.mekanGetir)
.put(ctrlMekanlar.mekanGuncelle)
.delete(ctrlMekanlar.mekanSil);
```


02 Express ile API Oluşturma

- Şimdi tüm rotalarımızı "index.js" içinde tanımlayalım:
  - **Yorumlar için**

```
router
.route('/mekanlar/:mekanid/yorumlar')
.post(ctrlYorumlar.yorumEkle);
```

```
router
.route('/mekanlar/:mekanid/yorumlar/:yorumid')
.get(ctrlYorumlar.yorumGetir)
.put(ctrlYorumlar.yorumGuncelle)
.delete(ctrlYorumlar.yorumSil);
```

- Rotaları tanımladık. İçlerinde hangi controller'larla bağlantılı olacağını yazdık.
- Ancak henüz controller'larımızı tanımlamadık. **Dolayısıyla şu aşamada uygulama çalışmayacaktır.**

02 Express ile API Oluşturma

```
app_api > routes > JS index.js > ...
1  var express = require('express');
2  var router = express.Router();
3  var ctrlMekanlar=require('../controllers/mekanlar');
4  var ctrlYorumlar=require('../controllers/yorumlar');
5  router
6  .route('/mekanlar')
7  .get(ctrlMekanlar.mekanlariListele)
8  .post(ctrlMekanlar.mekanEkle);
9
10 router
11 .route('/mekanlar/:mekanid')
12 .get(ctrlMekanlar.mekanGetir)
13 .put(ctrlMekanlar.mekanGuncelle)
14 .delete(ctrlMekanlar.mekanSil);
15
16 router
17 .route('/mekanlar/:mekanid/yorumlar')
18 .post(ctrlYorumlar.yorumEkle);
19
20 router
21 .route('/mekanlar/:mekanid/yorumlar/:yorumid')
22 .get(ctrlYorumlar.yorumGetir)
23 .put(ctrlYorumlar.yorumGuncelle)
24 .delete(ctrlYorumlar.yorumSil);
25
26 module.exports=router;
```

Rota Tanımları (app_api/routes/index.js)

02 Express ile API Oluşturma

- API'mizin veri tabanıyla konuşması için Mongoose'u ve kullanacağımız modelimizi (mekan) controller'umuza ("app_api/controllers/mekanlar.js" ve "app_api/controllers/yorumlar.js" içinde) tanıtmamız gerekir. İlk satırlara yazılacak kod:
  - **var mongoose = require("mongoose");**
  - **var Mekan = mongoose.model("mekan");**
- Veri tabanı işlemleriyle ve Mongoose ile ilgili işlemler hep API'de olacak.
- Dolayısıyla "app_server/models" klasörümüzü "app_api" klasörüne taşıyacağız.
- Aynı zamanda uygulamamıza taşıma yaptığımızı bildirmemiz ve yeni yolu tanıtmamız gerekiyor(app.js dosyasında).
  - require('./**app_server**/models/db'); komutu require('./**app_api**/models/db') ile değişecek;
- API'mizdeki GET, POST, PUT, DELETE metotlarını test etmek için VS Code Thunder Client eklentisini kullanacağız.

02 Express ile API Oluşturma

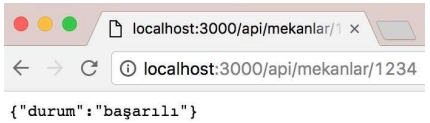
```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
require('./app_api/models/db');
var indexRouter = require('./app_server/routes/index');
var usersRouter = require('./app_server/routes/users');
var apiRouter = require('./app_api/routes/index');
var app = express();
app.use("/api",apiRouter);
```

app.js son hali (**Not**: require('./**app_api**/models/db'); kodu rota tanımlarından önce yazılmalıdır.

02 Express ile API Oluşturma

- Mekanlar ve yorumlar için ayrı ayrı 2 controller oluşturacağız. Bunlar "mekanlar.js" ve "yorumlar.js" ve "app_api/controllers" klasöründe yer alacak.
- Rotaları ve fonksiyonları test etmek için bir cevap döndüreceğiz. Cevap JSON formatında olacak.
- Bunun için aşağıda soldaki kod parçacığını tüm controller metotlarımıza (mekanlar.js ve yorumlar.js için ayrı ayrı) yazacağız. Sağdaki gibi test ettiğinizde aynı ekranı almanız gerekir.
- Testler başarılı olduktan sonra da bunların yerine gerçek işi yapan kodları yazacağız.

```
const cevapOlustur = function (res,status,content) {
  res
    .status(status)
    .json(content);
};
```



02 Express ile API Oluşturma

```
var mongoose = require('mongoose');
var Mekan = mongoose.model("mekan");
const cevapOlustur=function(res,status,content){
    res.status(status).json(content);
}
const mekanlariListele=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const mekanEkle=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const mekanGetir=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const mekanGuncelle=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const mekanSil=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
module.exports={
    mekanlariListele,
    mekanEkle,
    mekanGetir,
    mekanGuncelle,
    mekanSil
}
```

app_api/controllers/mekanlar.js dosyası

SON HALİ

02 Express ile API Oluşturma

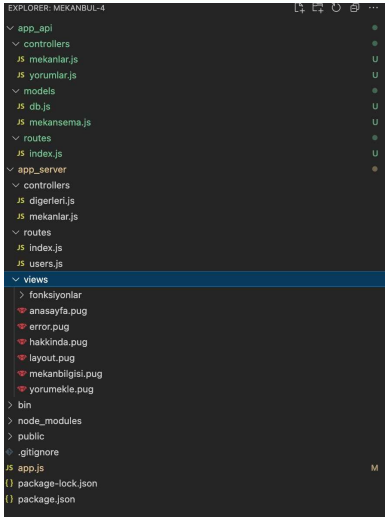
```
var mongoose = require('mongoose');
var Mekan = mongoose.model("mekan");
const cevapOlustur=function(res,status,content){
    res.status(status).json(content);
}

const yorumEkle=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const yorumGetir=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const yorumGuncelle=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
const yorumSil=function(req,res){
    cevapOlustur(res,200,{"durum":"başarılı"});
}
module.exports={
    yorumEkle,
    yorumGetir,
    yorumGuncelle,
    yorumSil
}
```

app_api/controllers/yorumlar.js dosyası

SON HALİ

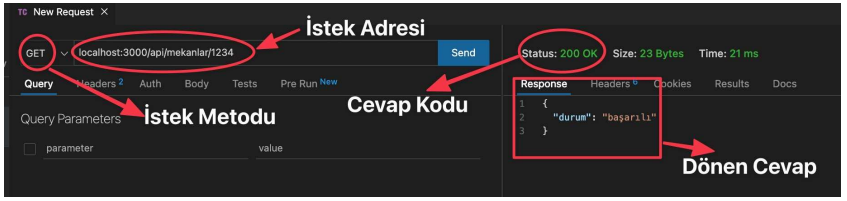
02 Express ile API Oluşturma



Son Klasör Yapısı

03 GET Metotları: MongoDB'den Veri Okuma

- GET metotları veri tabanını sorgulama ve sorgu sonucunda veri döndürme amacıyla kullanılır.
- Aşağıda Thunder Client aracılığı ile yapılan bir GET isteği örneği mevcuttur.



03 GET Metotları: MongoDB'den Veri Okuma

- MekanBul için oluşturacağımız API'de 3 adet GET isteği var.
  - Mekanları listeleme
  - Belirli bir mekanı getirme
  - Belirli bir yorumu getirme
- Mongoose'u kullanarak veri tabanımızla modeller aracılığı ile iletişim kuracağız.
- Mongoose modellerinin veri tabanı ile etkileşim sağlayan bazı metotları vardır.
  - Tek bir dokümanı "id" si ile bulmak için "findById" metodu.
  - Verilen sorgu parametresine göre arama yapmak için "find" metodu
  - Sorguyla eşleşen ilk dokümanı getiren "findOne" metodu
  - Verilen enlem, boylam bilgilerine en yakın yeri bulmak için "geoNear" metodu
  - geoNear metoduna sorgulama yeteneği ekleyen "geoSearch" metodu

03 GET Metotları: MongoDB'den Veri Okuma

- **Mekan.findById(mekanid):** mekanID parametresi alır ve Mekan modeli üzerinden çağrılır. Herhangi bir veri tabanı bağlantısı kurmaz. Sadece hangi sorgunun çalıştırılacağını söyler.
- Sorguyu çalıştırmak ***exec*** metodunun görevidir.
- ***exec*** metodu işlem bittiğinde çağrılacak geri çağrım metodunu (callback function) parametre olarak alır. Geri çağrım metodu 2 parametre alır.
  - Hata nesnesi
  - Bulunan dokümanın örneği
- Parametrelere herhangi bir isim atayabilirsiniz.
- Bu yöntem veri tabanı etkileşiminin asenkron olmasını sağlar. Dolayısıyla ana Node işlemini bloke etmez.

03 GET Metotları: MongoDB'den Veri Okuma

- Metodun başarılı bir şekilde çalışması için *mekanid* parametresinin URL'den alınarak findById metoduna geçirilmesi lazım.
- Ayrıca exec metodu çağrıldığında çalışacak bir çıkış fonksiyonu sağlanması gerekecek.
- Express URL parametlerini almamız için imkan sağlar. Parametreler "param" nesnesi içinde yer alır.
- Eğer rotamızı aşağıdaki gibi tanımladıysak:

```
router
.route('/mekanlar/:mekanid')
.get(ctrlMekanlar.mekanGetir)
```

- mekanid parametresine controller içinden ***req.params.mekanid*** diyerek erişebiliriz.
- Daha sonra ***cevapOlustur*** metodunu çıkış fonksiyonu olarak "exec" metoduna ekleyebiliriz.

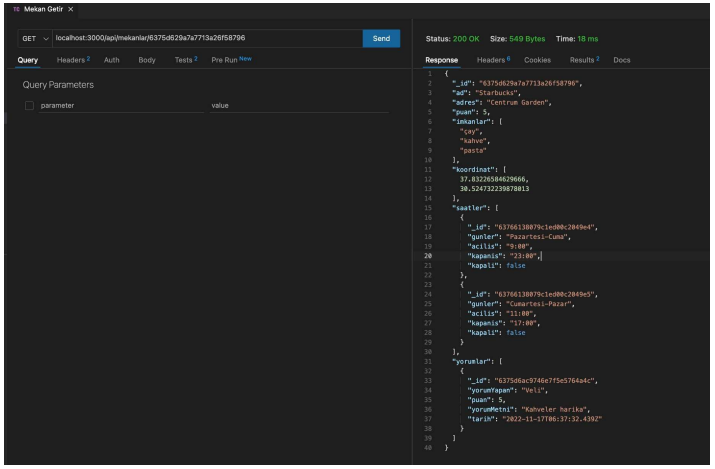
03 GET Metotları: MongoDB'den Veri Okuma

- Aşağıda bu metodun kullanımı gösterilmektedir.

```
const mekanGetir=function(req, res) {
    Mekan.findById(req.params.mekanid)
    .exec(function(hata,mekan){
        cevapOlustur(res,200,mekan);
    }
    )
}
```

- Artık temel anlamda bir API metodumuz oldu. Artık MongoDB'ye eklemiş olduğunuz dokümanın ID'sini (_id) alarak Thunder'da istekte bulunabilirsiniz.
  - localhost:3000/api/mekanlar/636cdd734a41c609c82b3444
  - **Yukarıda sonda yer alan mekanid benim veri tabanımda olan ID. Sizin ID'niz farklı olacaktır.**
  - Eğer ID yoksa ya da hatalı ID girerseniz null değer dönecektir. Herhangi bir uyarı ya da mesaj görülmeyecektir. Sadece 200 durum kodu ve null değer dönecektir.

03 GET Metotları: MongoDB'den Veri Okuma



Örnek Thunder Client Çıktısı

03 GET Metotları: MongoDB'den Veri Okuma

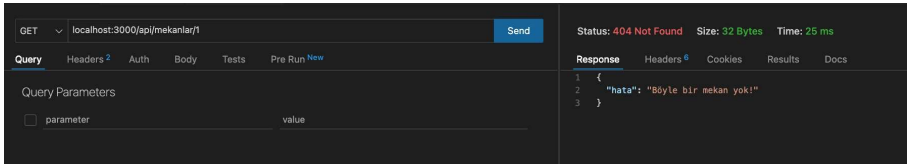
- Oluşturduğunuz controller'ın temel problemi sadece başarılı durumda bir değer döndürmesi ve hata oluştuğunda bilgilendirmemesidir.
- İyi bir API hata oluştuğunda da ne tür hata olduğunu istekte bulunana bildirmelidir.
- Basit if komutlarıyla hataları yakalayabiliriz:
  - İstekte mekanid parametresi yer almıyorsa
  - findById metodu herhangi bir mekan döndürmüyorsa
  - findById metodu bir hata döndürüyorsa
- Başarısız bir GET isteğinin kodu 404'tür.

03 GET Metotları: MongoDB'den Veri Okuma

```
const mekanGetir = function (req, res) {
  if (req.params && req.params.mekanid) {
    Mekan.findById(req.params.mekanid).exec(function (hata, mekan) {
      if (!mekan) {
        cevapOlustur(res, 404, { "hata": "Böyle bir mekan yok!" });
      } else if (hata) {
        cevapOlustur(res, 404, { "hata": hata });
      } else {
        cevapOlustur(res, 200, mekan);
      }
    });
  } else {
    cevapOlustur(res, 404, { "hata": "İstekte mekanid yok!" });
  }
};
```

mekanlar.js mekanGetir metodunun son hali

03 GET Metotları: MongoDB'den Veri Okuma



Olmayan bir ID girildiğinde Thunder Client çıktısı

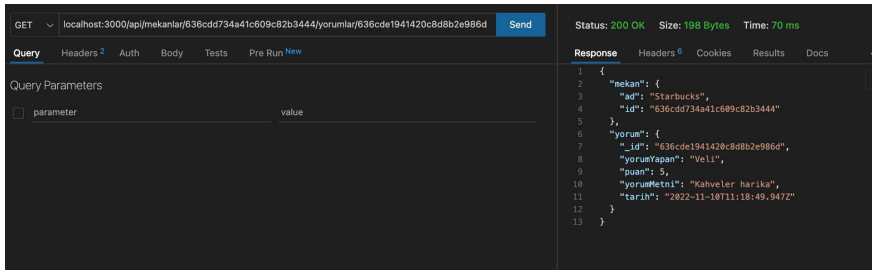
03 GET Metotları: MongoDB'den Veri Okuma

- Bir mekanı bulmak için yaptığımız işlemleri alt dokümanları (yorumlar) bulmak için de yapabiliriz.
- Bunun için:
  - Önce ana dokümanın id'sini bulmamız gerekir.
  - URL'mizde ek olarak yorumid parametresi kullanmalıyız.
  - Tüm dokümanı getirmek yerine mekan adı ve yorumları getirebiliriz. (Hızı artırmış ve bant genişliğini idareli kullanmış oluruz.)
  - Belli bilgileri getirmek için aşağıdaki komutu kullanarak limitleme yapabiliriz.
    - **.select('ad yorumlar') //sadece ad ve yorumları getirir**
  - yorumid ile eşleşen doküman varsa bulunur.
  - Uygun JSON formatıyla yorum döndürülür.
  - Mongoose alt dokümanlarda id kullanarak veri bulmak için ***id*** isimli bir metoda sahiptir.
    - **Mekan.yorumlar.id(req.params.yorumid);**
  - Son olarak mekanlar için yaptığımız hata kontrolünü yorumlar içinde yapmalıyız.

03 GET Metotları: MongoDB'den Veri Okuma

```
const yorumGetir = function (req, res) {
  if (req.params && req.params.mekanid && req.params.yorumid) {
    Mekan.findById(req.params.mekanid)
      .select("ad yorumlar")
      .exec(function (hata, mekan) {
        var cevap, yorum;
        if (!mekan) {
          cevapOlustur(res, 404, {
            "hata": "Böyle bir mekan yok!",
          });
          return;
        } else if (hata) {
          cevapOlustur(res, 400, hata);
          return;
        }
        if (mekan.yorumlar && mekan.yorumlar.length > 0) {
          yorum = mekan.yorumlar.id(req.params.yorumid);
          if (!yorum) {
            cevapOlustur(res, 404, {
              "hata": "Böyle bir yorum yok!",
            });
          }
          else {
            cevap = {
              mekan: {
                ad: mekan.ad,
                id: req.params.mekanid,
              },
              yorum: yorum,
            };
            cevapOlustur(res, 200, cevap);
          }
        } else {
          cevapOlustur(res, 404, {
            "hata": "Hiç yorum yok",
          });
        }
      });
  } else {
    cevapOlustur(res, 404, {
      "hata": "Bulunamadı, mekanid ve yorumid mutlaka girilmeli.",
    });
  }
};
```

03 GET Metotları: MongoDB'den Veri Okuma



Thunder Client ile Yorum Getirme GET İşlemi
(mekanid ve yorumid sizde farklı olacaktır)

03 GET Metotları: MongoDB'den Veri Okuma

- Uygulamanın ana sayfasında mekanlar kullanıcının konumuna göre uzaklıklarıyla beraber listeleniyor.
- MongoDB ve Mongoose yakındaki yerleri bulmamızı sağlayacak özel metotlara sahiptir.
- Belli bir noktaya olan uzaklığa göre mekanları listelememizi "geoNear" fonksiyonu sağlayacak.
- Bu metot 3 parametre alır:
  - geoJSON formatında enlem ve boylam bilgisi içeren coğrafi nokta (konum)
  - options (seçenekler) nesnesi
  - Geri çağırma metodu (callback function)
  - Örnek: mekan.geoNear(nokta, secenekler, callback);

03 GET Metotları: MongoDB'den Veri Okuma

- Rota tanımında URL parametresi olarak enlem ve boylam yer almamaktadır. Onun yerine bunu sorgu şeklinde yazmak daha uygundur.
  - **api/mekanlar?enlem=33.22&boylam=32.12** gibi
  - **req.query.enlem** ya da **req.query.boylam** diyerek parametreyi alabiliriz.
- geoNear fonksiyonun gerekli tek bir option (seçenek) parametresi vardır. Bu parametre "spherical" dir. Bu aramanın küresel mi yoksa düz bir düzlem üzerinde mi yapılacağını belirtir.
- İstenirse sonuçların sayısı ***limit*** seçeneği ile limitlenebilir.
- Ek olarak maksimum mesafe ***maxDistance*** parametresi ile sınırlandırılabilir.
- Mekan.aggregate diyerek belirlenen kriterlere uyan mekanları listeleyebiliriz. Aggregate metodu dokümandaki herşeyi getirir. Bu yüzden sadece ihtiyaç olan bilgi **map** fonksiyonuyla azaltılmalı.

03 GET Metotları: MongoDB'den Veri Okuma

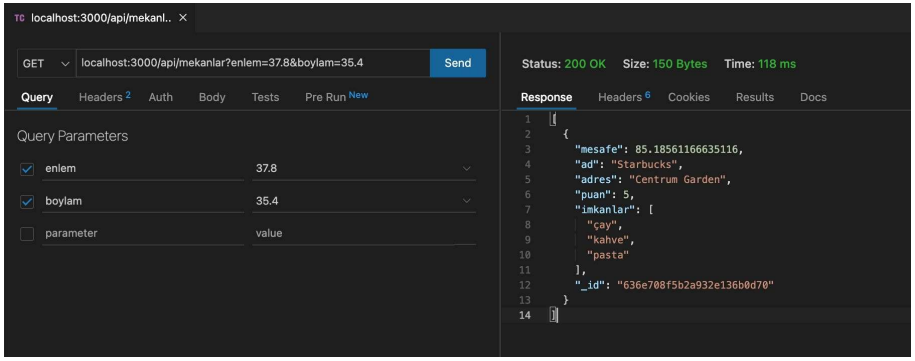
```
const mekanlariListele = async (req, res) => {
  var boylam = parseFloat(req.query.boylam);
  var enlem = parseFloat(req.query.enlem);
  var koordinat = {
    type: "Point",
    coordinates: [enlem, boylam],
  };
  var geoOptions = {
    distanceField: "mesafe",
    spherical: true,
  };
  if ((!enlem && boylam !== 0) || (!enlem && boylam !== 0)) {
    cevapOlustur(res, 404, {
      "hata": "enlem ve boylam zorunlu parametreler",
    });
    return;
  }
  try {
    const sonuc = await Mekan.aggregate([
      {
        $geoNear: {
          near: koordinat,
          ...geoOptions,
        },
      },
    ]);
    const mekanlar = sonuc.map((mekan) => {
      return {
        mesafe: cevrimler.kilometre2Radyan(mekan.mesafe),
        ad: mekan.ad,
        adres: mekan.adres,
        puan: mekan.puan,
        imkanlar: mekan.imkanlar,
        _id: mekan._id,
      };
    });
    cevapOlustur(res, 200, mekanlar);
  } catch (e) {
    cevapOlustur(res, 404, e);
  }
};
```

mekanlariListele fonksiyonunun üstünde tanımlanacak çevrim metodu

```
var cevrimler = (function () {
  var dunyaYariCap = 6371; // km
  var radyan2Kilometre = function (radyan) {
    return parseFloat(radyan * dunyaYariCap);
  };
  var kilometre2Radyan = function (mesafe) {
    return parseFloat(mesafe / dunyaYariCap);
  };
  return {
    radyan2Kilometre: radyan2Kilometre,
    kilometre2Radyan: kilometre2Radyan,
  };
})();
```

Mekanları listeleme metodu(app_api/controllers/mekanlar.js)

03 GET Metotları: MongoDB'den Veri Okuma



Mekan Listeleme Thunder Client GET İsteği Çıktısı

04 Ödev

- Veritabanı bağlantı adresini MongoDB Cloud üzerindeki bağlantı adresi olarak ayarlayın.
- Veritabanında mutlaka en az 1 mekan ve mekanın içinde en az 1 yorum olmalı.
- Kalan API isteklerinin kodlarını yazın ve projenizi odev5 dalı altında Github reponuza gönderin ve odev5'i varsayılan repo olarak ayarlayın.
- Projenizi replite bağlayın. İsteyen odev5…. ile başlayan bir replit oluşturabilir.
- Proje klasöründe resimler isimli bir klasör oluşturun. Bu klasöre 3 GET isteğinin çıktılarının ekran görüntülerini ayrı ayrı ekleyin.
- Proje klasöründe README.md isimli bir dosya oluşturun. Markdown ile GET API isteklerinin bağlantılarını madde madde ekleyin.
- Her bir bağlantıya tıkladığımda ekranda sonuç görebilmeliyim.
- Her istek bağlantısının altında isteklerin başarılı olduğunu gösteren resimler klasörü içindeki ekran görüntülerini ekleyin.
- Anlatılanlar eksiksiz yapıldığında ve **22.11.2022 Salı** saat **23.59.00'dan önce** gönderildiğinde puan alacaksınız.
- Örnek README.md çıktısı bir sonraki sayfada yer almaktadır.
- Göndermeden önce mutlaka test edin. Discord üzerinden ilgili öğretimin konu başlığı altından Ad Soyad, Numara, Github adresinizi gönderin.

04 Ödev