Furkan Kazım Çam – 21360859036

Node.js ile Web Programlama 13.Hafta Ders Raporu

Bu hafta mongoose modülü ile veri doğrulama işlemleri yapacağız.

Nesne-Belge Eşleyici (Object-Document Mapper, ODM), yazılım geliştirmede, nesne yönelimli programlama dilleri ile NoSQL veritabanları arasında bir köprü görevi gören bir araçtır. Bu araçlar, nesne yönelimli paradigmalarla veri manipülasyonunu kolaylaştırmak amacıyla kullanılır. ODM, nesnelerin NoSQL belgeleriyle eşleşmesini sağlar ve böylece veri manipülasyonu ve saklama işlemleri daha doğal ve daha az karmaşık hale gelir.

ODM'nin Temel Özellikleri

1. **Nesne Eşleştirme**: ODM, programlama dilindeki sınıflar (nesneler) ile veritabanındaki belgeler arasında bir eşleştirme yapar. Bu, geliştiricilerin doğrudan nesnelerle çalışarak veritabanına erişmesini sağlar.
2. **CRUD İşlemleri**: ODM, oluşturma (Create), okuma (Read), güncelleme (Update) ve silme (Delete) işlemlerini nesne yönelimli bir şekilde gerçekleştirir. Bu işlemler, veritabanı sorgularını ve manipülasyonlarını kolaylaştırır.
3. **Veri Doğrulama**: ODM, nesne modellerine dayalı olarak veri doğrulama işlemlerini destekler. Bu, veritabanına kaydedilen verilerin doğruluğunu ve bütünlüğünü sağlar.
4. **İlişkisel Veriler**: ODM, nesneler arasındaki ilişkileri (birçok-bire, bire-bir, vb.) yönetir ve bu ilişkilerin veritabanında nasıl temsil edileceğini belirler.

ODM Kullanımının Avantajları

- **Geliştirici Üretkenliği**: Geliştiriciler, karmaşık veritabanı sorguları yazmak yerine nesnelerle çalışarak daha hızlı ve verimli bir şekilde kod yazabilir.
- **Veritabanı Bağımsızlığı**: ODM, uygulamalarıntabanı değişikliklerinin veya göçlerinin daha kolay yönetilmesini sağlar.
- Kodun Okunabilirliği ve Bakımı: ODM kullanımı, kodun daha okunabilir ve bakımı daha kolay olmasını sağlar. Nesneler ve onların etkileşimleri, doğal programlama dilinde ifade edilebilir.
- Veri Modelleme Kolaylığı: Nesne modelleri, yazılımın veri yapılarını doğrudan yansıtabilir, bu da veri modellemesini ve tasarımını daha sezgisel hale getirir.

Avantajları

- **Kolay Kullanım:** Nesne yönelimli programlama paradigmasına uygun olduğu için, geliştiriciler veritabanı işlemlerini daha doğal bir şekilde gerçekleştirebilir.
- **Veri Doğruluğu:** Veri modeline uygunluk kontrolü sayesinde, veritabanındaki verilerin doğruluğu ve tutarlılığı sağlanır.
- **Yüksek Üretkenlik:** Sorgulama ve güncelleme işlemlerinin daha basit ve okunabilir hale gelmesi, geliştirme sürecini hızlandırır.

Dezavantajları

- **Performans:** Bazı durumlarda, doğrudan veritabanı sorguları yerine ODM kullanmak performans kaybına neden olabilir.
- **Ek Bağımlılık:** ODM kullanımı, uygulamanın ek bir kütüphaneye bağımlı olmasına neden olur, bu da bakım ve güncelleme süreçlerini etkileyebilir.

Sonuç olarak, ODM'ler MongoDB gibi belge tabanlı veritabanlarıyla çalışmayı kolaylaştıran ve geliştiricilere esneklik sunan araçlardır. Ancak, kullanım kararı verilirken performans ve bağımlılık gibi faktörler de göz önünde bulundurulmalıdır.

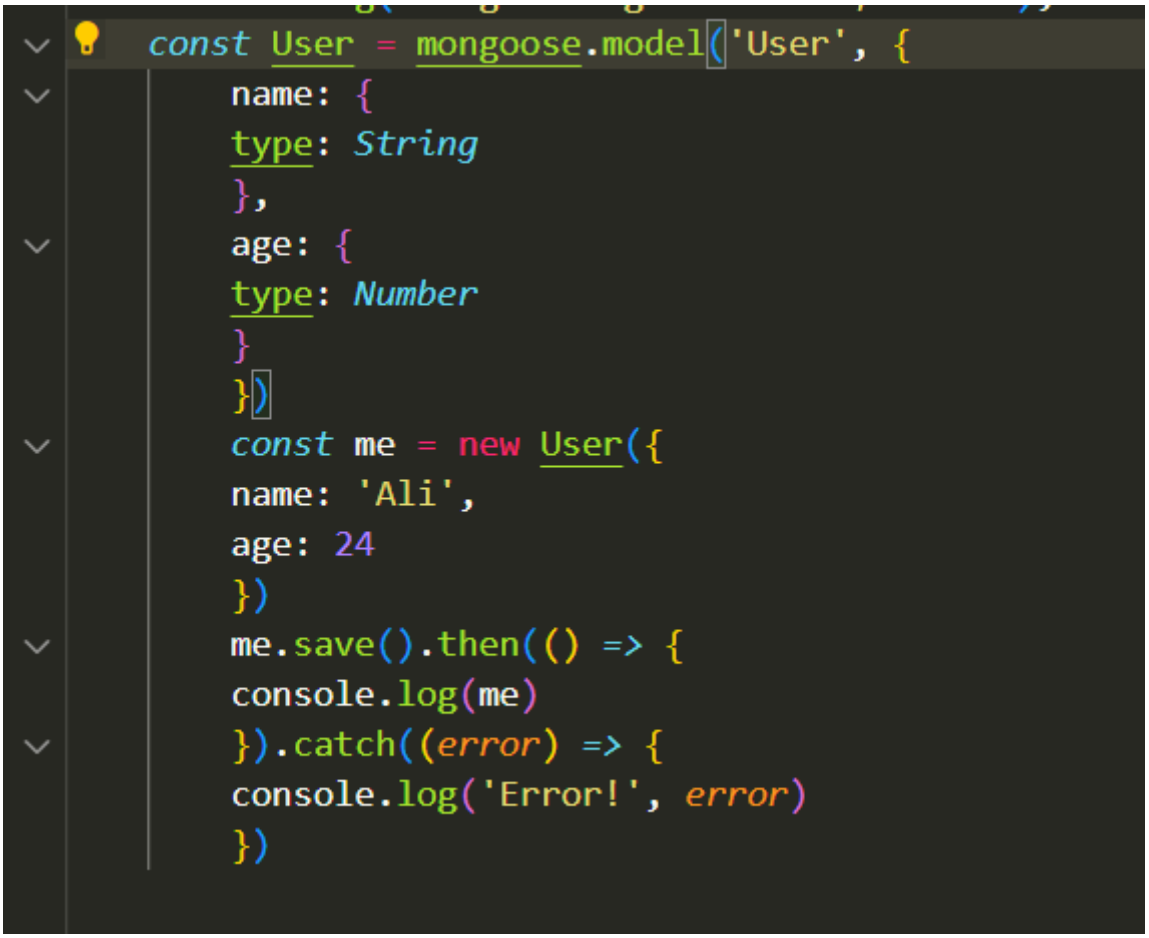
Bir ODM örneği olan mongoose modülünü öğreneceğiz.

Npm i mongoose ile modülü indiriniz. Mongoose.js dosyası oluşturunuz ve modülü sayfaya require ediniz. Ardından ise bir değişkene mongodb connection stringinizi veriniz. Sonrasında ise bağlantı kodunu yazınız. Bağlantı kodu aşağıda verilmiştir.

```
const mongoose = require('mongoose');  846.7k (gzipped: 228.1k)

const database = "mongodb+srv://Furkann:1708@furkan.1cbjktx.mongodb.net/?retryWrites=true&w=majority&appName=Furkan";

Complexity is 5 Everything is cool!
async function connectToDatabase() {
  try {
    await mongoose.connect(database, {
      useUnifiedTopology: true, // Güncel bağlantı seçeneği
    });
    console.log('MongoDB bağlantısı başarılı!');
  } catch (error) {
    if (error instanceof mongoose.Error.MongooseServerSelectionError) {
      console.error('MongoDB bağlantısı başarısız: Beyaz listeye alınmamış bir IP adresinden bağlanmaya çalışıyor olabilirsiniz. Lütfen IP adresinizi kontrol
    } else if (error instanceof mongoose.Error.MongoNetworkError) {
      console.error('MongoDB bağlantısı başarısız: Ağ bağlantısı hatası. İnternet bağlantınızı ve MongoDB Atlas ayarlarınızı kontrol edin.');
    } else {
      console.error('MongoDB bağlantısı başarısız:', error.message);
    }
  }
}

connectToDatabase();
```

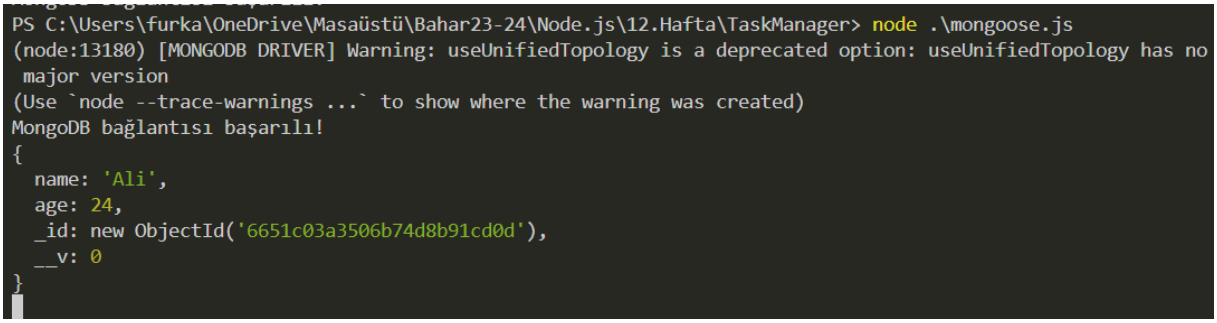
Çıkan hata türüne göre hata mesajlarını terminale yazdıracaktır.

Veri tabanına göndereceğiniz verinin modelini tanımlayınız.



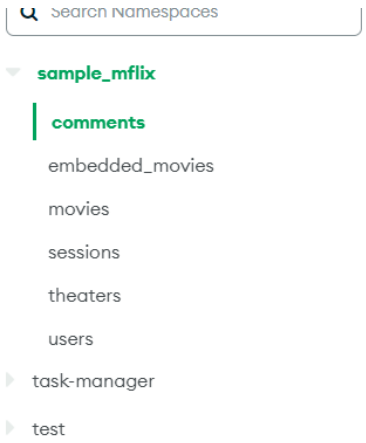
```
const User = mongoose.model('User', {
    name: {
    type: String
    },
    age: {
    type: Number
    }
    })
    const me = new User({
    name: 'Ali',
    age: 24
    })
    me.save().then(() => {
    console.log(me)
    }).catch((error) => {
    console.log('Error!', error)
    })
```

Bu kod name ve age inputu alan bir model tanımlar ve name'i ali age2i 24 olan bir me tanımlar en sonda ise bu veriyi veri tabanına gönderir.



```
PS C:\Users\furka\OneDrive\Masaüstü\Bahar23-24\Node.js\12.Hafta\TaskManager> node .\mongoose.js
(node:13180) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no
 major version
(Use `node --trace-warnings ...` to show where the warning was created)
MongoDB bağlantısı başarılı!
{
  name: 'Ali',
  age: 24,
  _id: new ObjectId('6651c03a3506b74d8b91cd0d'),
  __v: 0
}
```

Terminale yazdırdı şimdi de veri tabanını kontrol ediniz.




```
_id: ObjectId('6651c03a3506b74d8b91cd0d')
name : "Ali"
age : 24
__v : 0
```

Test adında bir veri tabanı oluştu içindeki veri ise gönderdiğiniz veri.

Task modeli oluşturarak devam ediniz.

```
const Task = mongoose.model('Task', {
    description: {
    type: String
    },
    completed: {
    type: Boolean
    }
    })
    const task = new Task({
    description: 'Lear the mongoose library',
    completed: false
    })
    task.save().then(() => {
        console.log(task)
        }).catch((error) => {
        console.log(error)
        })
```

Bir description ve completed girdisi alıyor. Ve buna veriyi gönderiyoruz.

```
(node:16140) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopolo
 major version
(Use `node --trace-warnings ...` to show where the warning was created)
MongoDB bağlantısı başarılı!
{
  description: 'Lear the mongoose library',
  completed: false,
  _id: new ObjectId('6651c159cc67c3e0ee775d7b'),
  __v: 0
}
```

Bir sıkıntı çıkmadı bir de veritabanını kontrol edelim.

```
_id: ObjectId('6651c159cc67c3e0ee775d7b')
description : "Lear the mongoose library"
completed : false
__v : 0
```

Bir sıkıntı yok her şey istediğimiz gibi gidiyor.

Önceki user modelimizin name alanına required özelliğini ekleyelim ve bu özelliği zorunlu hale getirelim.

```
name: {
type: String,
required:true
},
```

```
})
const me = new User({

})
```

Böyle bir veri gönderirsek hata verir. Şimdi veriyi doğru şekilde gönderiniz.

```
const me = new User({
name: "furkan"
})
```

```
PS C:\Users\furka\OneDrive\Masaüstü\Bahar23-24\Node.js\12.Hafta\TaskManager> node .\mongoose.js
(node:8496) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopol
major version
(Use `node --trace-warnings ...` to show where the warning was created)
MongoDB bağlantısı başarılı!
{
  name: 'furkan',
  _id: new ObjectId('6651c2853ef117dc3557405a'),
  __v: 0
```

Terminalde bir sıkıntı yok veri tabanını kontrol ediniz.

```
_id: ObjectId('6651c2853ef117dc3557405a')
name : "furkan"
__v : 0
```

Doğru yolda ilerliyoruz.

Validatör modülüne geçmeden önce kendiniz validatör yazabilir misiniz kontrol ediniz.

Bunun için user modelinin yaş tanımına bir ekleme yapalım ve yaş değişkeni eksi değer alamasın.

```
age: {
    type: Number,
    Complexity is 3 Everything is cool!
    validate(value) {
    if (value < 0) {
    throw new Error('Age must be positive')
    }
    }
    }
```

Age değişkeniniz artık böyle olmalıdır. Bu işinizi görmektedir.

```
const me = new User({
name: "furkan",
age:-5
})
```

Böyle bir veri gönderiminde hata alıcaksınız ve veri veri tabanına kaydedilmeyecektir. Nedeni ise bizim yazdığımız validation metodu.

Bu validation metodlarını tek tek elimizle yazmaktansa bunu sağlayan modül olan validation modülünü indirin.

Npm i validatör ile indirip const validator = require(‘validator’) ile modülü kullanabilirsiniz.

```
email: {
    type: String,
    required: true,
    Complexity is 3 Everything is cool!
    validate(value) {
    if (!validator.isEmail(value)) {
    throw new Error('Email is invalid')
    }
    }
    },
```

User’a email değişkeni ekleyerek devam ediniz. Bu değişken validatör modülü ile kontrol ediliyor. Email formatında değil ise hata döndürür.

```
const me = new User({
name: "furkan",
email: 'furkan@'

})
```

Veriyi gönderiniz ve sonuca bakınız. Hata alıcaksınız. Nedeni ise email formatında olmamasıdır.

Veriye daha fazla kontrol ekleyerek devam ediniz.

```
email: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
    Complexity is 3 Everything is cool!
    validate(value) {
    if (!validator.isEmail(value)) {
    throw new Error('Email is invalid')
    }
    }
    },
```

Trim boşluk var mı diye kontrol ederken lowercase bütün harfler küçük mü onu kontrol eder.

```
   const me = new User({
  name: "furkan",
  email: 'furkan@gmail.com'

   })
```

```
MongoDB bağlantısı başarılı!
{
  name: 'furkan',
  email: 'furkan@gmail.com',
  _id: new ObjectId('6651c5200343394d0598213a'),
  __v: 0
}
```

Göründüğü gibi bütün kurallara uyulduğunda veri gönderiliyor.

Veri tabanından da kontrol edebilirsiniz.

```
_id: ObjectId('6651c5200343394d0598213a')
name : "furkan"
email : "furkan@gmail.com"
__v : 0
```

```
password: {
    type: String,
    required: true,
    minlength: 7,
    trim: true,
    Complexity is 3 Everything is cool!
    validate(value) {
    if (value.toLowerCase().includes('password')) {
    throw new Error('Password cannot contain “password”')
    }
    }
    },
```

Şifre alanı ekleyiniz. Verinin gerekliliği 7 karakterden uzun olması ve boşluk olmamasıdır. Ona göre bir veri gönderiniz.

```
    const me = new User({
    name: "furkan",
    email: 'furkan@gmail.com',
    password: "furkankazımcam"

    })
```

```
MongoDB bağlantısı başarılı!
{
  name: 'furkan',
  password: 'furkankazımcam',
  email: 'furkan@gmail.com',
  _id: new ObjectId('6651c5e45c41c28cb2bc7de4'),
  __v: 0
}
```

Veri tabanından da kontrol edebilirsiniz.

```
_id: ObjectId('6651c5e45c41c28cb2bc7de4')
name : "furkan"
password : "furkankazımcam"
email : "furkan@gmail.com"
__v : 0
```

En son olarak ise task modelini güncelleyin ve ona veri gönderiniz.

```
const Task = mongoose.model('Task', {
    description: {
    type: String,
    required: true,
    trim: true
    },
    completed: {
    type: Boolean,
    default: false
    }
    })
```

Aynı şekilde description bölümü zorunludur.

```
    const task = new Task({
    description: 'Lear the mongoose library',
    completed: true
    })
```

```
MongoDB bağlantısı başarılı!
{
  description: 'Lear the mongoose library',
  completed: true,
  _id: new ObjectId('6651c6bbeeafe74915a177e1'),
  __v: 0
}
```

Veri tabanından da kontrol edebilirsiniz.

```
_id: ObjectId('6651c6bbeeafe74915a177e1')
description : "Lear the mongoose library"
completed : true
__v : 0
```