Node.JS ile Web Programlama
Rapor - 13

Mahire Zühal Özdemir

İÇİNDEKİLER

ŞEKİLLER DİZİN

Mongoose Nedir?

Mongoose, Node.js ortamında MongoDB ile çalışmayı kolaylaştıran popüler bir kütüphanedir.

Mongoose, MongoDB veritabanına bağlanmayı, veri modelleri oluşturmayı ve veritabanı işlemlerini daha kolay ve güvenli hale getiren özellikler sunar.

ODM (Object-Document Mapper) Nedir?

Mongoose, bir Object-Document Mapper (ODM) olarak çalışır. Bu, nesneler (JavaScript nesneleri) ile belgeler (MongoDB belgeleri) arasındaki ilişkiyi yönetir.

Mongoose, doğrudan MongoDB ile çalışmak üzere tasarlanmıştır.

Mongoose ve MongoDb beraber kullanmak için proje klasörümüzü açıyoruz.

1. Mongoose'u Yükleme:

```
\Hafta-13> npm i mongoose
```

Şekil 1: Mongoose İndirme

Geliştirmeye devam ettiğimiz uygulamamız task-manager klasöründe sırayla src -> db içerisine mongoose.js dosyasını oluşturuyoruz.

2. Mongoose İmport Etme:

```
JS mongoose.js > ...
1   const mongoose = require('mongoose')
```

Şekil 2: Mongoose İmport İşlemi

3. Mongoose Kullanarak MongoDb'ye Bağlanma:

```
mongoose.connect('mongodb://127.0.0.1:27017/task-manager-api', {
    useNewUrlParser: true,
    useCreateIndex: true
    })
```

Şekil 3: MongoDB Bağlantı

- *mongodb://127.0.0.1:27017/task-manager-api* : mongodb – mongodb'nin çalıştığı sunucu adresi – bağlantı kurulacak veri tabanı adı

- *useNewUrlParser: true :* Url çözücüsünü çalıştırır, url hatası önlenir.
- *useCreateIndex: true :* Mongoose'un *ensureIndex* yerine *createIndex* fonksiyonunu kullanmasını sağlar.

4. Model Oluşturma:

```
const User = mongoose.model('User', {
name: {
type: String
},
age: {
type: Number
}
})
```

Şekil 4: Model Oluşturma

- *mongoose.model:* Mongoose ile yeni bir model (koleksiyon) tanımlanır.
- Model adı: User Koleksiyon adı: users Şema name ve age değişkenlerini alır. Name veri türü: string age veri türü number olarak belirtilir.

5. Yeni Kullanıcı Belgesi Oluşturma:

```
const me = new User({
name: 'Ali',
age: 24
})
```

Şekil 5: Yeni Kullanıcı Belgesi Oluşturma

- *new User:* User modelini kullanarak yeni bir kullanıcı belgesi oluşturur.
- Yeni belge verileri ; İsim= Ali ve age=24 bilgilerini içerir.

6. Oluşturulan Belgeyi Veri Tabanına Kaydetme:

```
me.save().then(() => {
console.log(me)
}).catch((error) => {
console.log('Error!', error)
})
```

Şekil 6: Veritabanına Belge Kaydetme

- *me.save():* Oluşturulan kullanıcı belgesini MongoDB'ye kaydetmeye çalışır.
- *.then(() => { console.log(me) }):* Kaydetme işlemi başarılı olursa, konsola kaydedilen belgeyi (me) yazdırır.
- *.catch((error) => { console.log('Error!', error) }):* Kaydetme işlemi başarısız olursa, hatayı yakalar ve konsola 'Error!' mesajıyla birlikte hatayı yazdırır.

Kodu node mongoose.js ile çalıştırdıktan sonra çıktımız aşağıdaki gibi olur:

```
PS C:\Users\zuhal\Desktop\Node.Js\Hafta-13> node mongoose.js
{ name: 'Ali', age: 24, _id: new ObjectId('6652537a40136838bf55711f') }
```

Şekil 7: Kod Çıktısı

7. Yeni bir Model Oluşturma

```
const Task = mongoose.model('Task', {
    description: {
    type: String
    },
    completed: {
    type: Boolean
    }
})
```

Şekil 8: tasks Koleksiyonu Oluşturma

- *mongoose.model:* Mongoose kullanarak yeni bir model (koleksiyon) tanımlar.

- *'Task':* Modelin adı, MongoDB'de "tasks" adlı bir koleksiyon oluşturur (Mongoose otomatik olarak model ismini çoğul hale getirir).
- *description:* Belge bilgileri: description ; görevin açıklamasını içeren bir alan. Türü String.
- *completed:* Belge bilgileri: completed ;görevin tamamlanıp tamamlanmadığını belirten bir alan. Türü Boolean.

8. Yeni Belge Oluşturma:

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

Şekil 9: Yeni belge ekleme

- new Task: Task modelini kullanarak yeni bir görev belgesi oluştur.
- Görev açıklamasını ve tamamlanıp tamamlanmadığını belirt.

9. Belgeyi Veritabanına Kaydet:

```
task.save().then(() => {
console.log(task)
}).catch((error) => {
console.log(error)
})
```

Şekil 10: Belgeyi Kaydetme

- Oluşturulan belge koleksiyona keydedilir. Hata durumunda ekrana hata mesajı yazdırılır.

10.Veri Doğrulama ve Temizleme:

Required ile zorunlu belirlenen bir alanın belge kaydetme işlemi sırasında zorunlu alan boş bırakıldığında hata mesajı alınır. Belge kaydetme işlemi gerçekleşmez.

```
const User = mongoose.model('User', {
    name: {
    type: String,
    required: true
    },
    age: { type: Number}
})
```

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

me.save().then(() => {
console.log(me)
}).catch((error) => {
console.log('Error!', error)
})
```

Şekil 11: Hatalı belge kaydı - Zorunlu alanın boş bırakılması

Hatayı düzeltmek için required belirtilen alanın doldurulması gerekir.

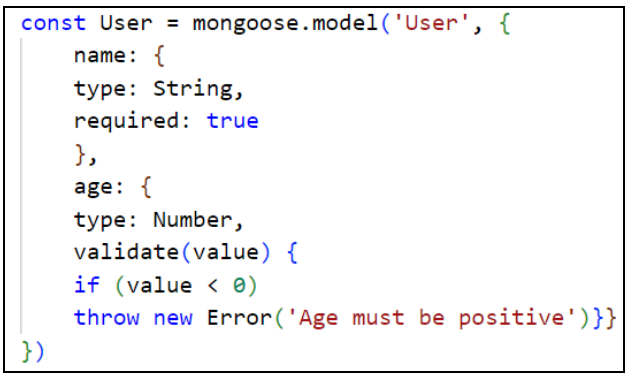
```
const me = new User({name: 'Ali'})

me.save().then(() => {
console.log(me)
}).catch((error) => {
console.log('Error!', error)
})
```

Şekil 12: Başarılı belge kaydı - Zorunlu alanın doldurulması

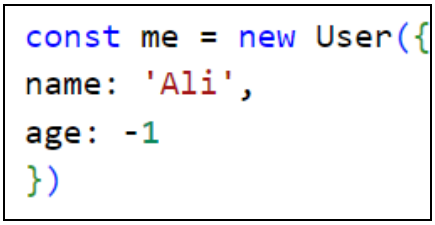
11. Required ve Validate Parametreleri

- name: Türü String olan ve required: true ile zorunlu kılınan bir alan. Bu, name alanının boş bırakılamayacağı anlamına gelir.
- age: Türü Number olan bir alan. Ayrıca, validate fonksiyonu ile ek bir doğrulama (validation) uygulanır:
- validate(value) { ... }: age alanının değerini doğrulamak için özel bir fonksiyon tanımlar.
- if (value < 0) { throw new Error('Age must be positive'); }: Eğer age değeri 0'dan küçükse, bir hata fırlatır ve hata mesajı olarak "Age must be positive" (Yaş pozitif olmalıdır) verir. Bu, negatif yaş değerlerinin kabul edilmemesini sağlar.



Şekil 13: Required ve Validate

- new User({ name: 'Ali', age: -1 }): name alanı "Ali" ve age alanı -1 olan yeni bir kullanıcı belgesi oluşturur.



Şekil 14: Belge Kaydı

- Belge kaydedilmeye çalışılır. age değeri negatif olduğundan doğrulama başarısız olur ve bir hata fırlatılır.

Validator. Js Nedir?

validator.js, Node.js ortamında kullanılan popüler bir doğrulama ve sanitize (temizleme) kütüphanesidir. Bu kütüphane, yaygın doğrulama ihtiyaçlarını karşılamak için çeşitli fonksiyonlar sunar ve genellikle kullanıcı girdilerini doğrulamak ve temizlemek için kullanılır.

Özellikleri

- Doğrulama Fonksiyonları: E-posta adresleri, URL'ler, IP adresleri, kredi kartı numaraları, gibi yaygın girdileri doğrulamak için birçok hazır fonksiyon sağlar.

- Temizleme Fonksiyonları: Girdileri belirli bir formatta temizlemek veya dönüştürmek için fonksiyonlar sunar. Örneğin, karakterleri silmek veya belirli karakterleri kaldırmak gibi işlemler.

Validator.js Yüklemek İçin

```
npm i validator
```

Şekil 15: Validator İndirme

Validator.JS İmport Etmek İçin

```
const validator = require('validator')
```

Şekil 16: Validator İmport Etme

Validator Js ile Validasyon İşlemleri

```
const User = mongoose.model('User', {
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        validate(value) {
            if (!validator.isEmail(value)) {
                throw new Error('Email is invalid')}}
    },
    age: {
        type: Number,
        validate(value) {
        if (value < 0) {
        throw new Error('Age must be positive')}
    }}
})
```

Şekil 17: Validator - isEmail Fonksiyonu

```
const me = new User({
name: 'Ali',
email: 'ali@'
})
```

Şekil 18: Kod Denemesi

Kodu ali@ email değeri ile denersek hata alırız. Çünkü böyle bir email hesabı olmadığı için doğrulama hatası verir.

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

Şekil 19: Email validator değerleri

Ek özellikler incelendiğinde:

- trim :true = girdide bulunan boşlukları siler.
- lowercase:true = küçük harflere dönüştürür.

Şifre Alanı

- type: String: password alanının türünü belirtir. Bu durumda, şifre bir string (metin) olmalıdır.
- required: true: Bu alanın zorunlu olduğunu belirtir. Şifre alanı boş bırakılamaz.
- minlength: 7: Şifrenin minimum uzunluğunu belirler. Şifre en az 7 karakter uzunluğunda olmalıdır.
- trim: true: Şifre değerinin başındaki ve sonundaki beyaz boşlukları otomatik olarak kırpar. Örneğin, kullanıcı " password " şeklinde bir değer girerse, bu değer "password" olarak kaydedilir.
- validate(value): Özel bir doğrulama fonksiyonu tanımlar. Bu fonksiyon, şifrenin belirli bir koşulu sağlamasını kontrol eder:
- value.toLowerCase().includes('password'): Şifre değeri küçük harfe dönüştürülür ve içinde "password" kelimesi geçip geçmediği kontrol edilir. Eğer şifre "password" kelimesini içeriyorsa, doğrulama başarısız olur.

```
password: {
    type: String,
    required: true,
    minlength: 7,
    trim: true,
    validate(value) {
        if (value.toLowerCase().includes('password')) {
            throw new Error('Password cannot contain "password"')
        }
    }
    },
})
```

Şekil 20: Şifre Alanı