

Modelling Data and Advanced Mongoose

Locations (Geospatial Data)

Sy: 1

```
6  const tourSchema = new mongoose.Schema(  
7    {  
92    startLocation: {  
93      // GeoJSON  
94      type: {  
95        type: String,  
96        default: 'Point',  
97        enum: ['Point'],  
98      },  
99      coordinates: [Number],  
100     address: String,  
101     description: String,  
102   },  
103   locations: [  
104     {  
105       type: {  
106         type: String,  
107         default: 'Point',  
108         enum: ['Point'],  
109       },  
110       coordinates: [Number],  
111       address: String,  
112       description: String,  
113       day: Number,  
114     },  
115   ],  
116   guides: [  
117     {  
118       type: mongoose.Schema.ObjectId,  
119       ref: 'User',  
120     },  
121   ],  
122 },  
127 );
```

Start Location için
kullanılan **Geojson**
veri tipi.

Locations dizisi için
de **Geojson** veri
tipi kullanılmakta.

Modelling Data and Advanced Mongoose

Populating 1

Sy: 2

tourSchema.guides Property

```
6 const tourSchema = new mongoose.Schema(  
7   {  
116     guides: [  
117       {  
118         type: mongoose.Schema.ObjectId,  
119         ref: 'User',  
120       },  
121     ],  
122   },  
127 );
```

tourSchema.find populate

```
1 tourSchema.pre(/^find/, function (next) {  
2   this.populate({  
3     path: 'guides',  
4     select: '-__v -passwordChangedAt',  
5   });  
6  
7   next();  
8 });
```

Get Tour

GET {{URL}} api/v1/tours/5c88fa8cf4afda39709c2951

Params Authorization Headers (9) Body Scripts Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies (1) Headers (21) Test Results

{ } JSON Preview Visualize

```
26 "secretTour": false,  
27 "guides": [  
28   {  
29     "role": "guide",  
30     "_id": "682cb8aaad40925ec430c628",  
31     "name": "User",  
32     "email": "guide@jonas.io"  
33   },  
34 ],  
35 "_id": "5c88fa8cf4afda39709c2951",  
36 "name": "The Forest Hiker",  
37 "duration": 5,  
38 "maxGroupSize": 25,  
39 "difficulty": "easy",  
40 "price": 397,
```

Tour şeması üzerinde oluşturulan **guides property**'si **User** şeması ile ilişki kurmak için kullanılan bir alandır. Tour şeması üzerinde bulunan **guides** özelliği turda görevli kullanıcıların **id** bilgilerini tutar. Guides ve id üzerinden kurulan ilişki ile turda görevli kullanıcı bilgileri alınabilir. Tour bilgileri üzerinde yapılan her **find** ile başlayan sorgularda Tour.guides üzerinde tutulan idler üzerinden **this.populate()** metodu işletilerek kullanıcıların diğer bilgileri getirilir.

Modelling Data and Advanced Mongoose

Review Schema

Sy: 3

reviewSchema

```
1 const reviewSchema = new mongoose.Schema(
2   {
3     review: {
4       type: String,
5       required: [true, 'Review can not be empty!'],
6     },
7     rating: {
8       type: Number,
9       min: 1,
10      max: 5,
11    },
12    createdAt: {
13      type: Date,
14      default: Date.now,
15    },
16    tour: {
17      type: mongoose.Schema.ObjectId,
18      ref: 'Tour',
19      required: [true, 'Review must belong to a tour.'],
20    },
21    user: {
22      type: mongoose.Schema.ObjectId,
23      ref: 'User',
24      required: [true, 'Review must belong to a user'],
25    },
26  },
27  {
28    toJSON: { virtuals: true },
29    toObject: { virtuals: true },
30  },
31 );
32
33 reviewSchema.pre(/^find/, function (next) {
34   this.populate({
35     path: 'tour',
36     select: 'name',
37   }).populate({
38     path: 'user',
39     select: 'name photo',
40   });
41   next();
42 });
43
```

reviewController

```
1 exports.getAllReviews = catchAsync(async (req, res, next) => {
2   const reviews = await Review.find();
3
4   res.status(200).json({
5     status: 'success',
6     results: this.getAllReviews.length,
7     data: {
8       reviews,
9     },
10  });
11 });
12
13 exports.createReview = catchAsync(async (req, res, next) => {
14   const newReview = await Review.create(req.body);
15
16   res.status(201).json({
17     status: 'success',
18     data: {
19       review: newReview,
20     },
21  });
22 });
```

Create New Review

HTTP Jonas Natours 2025 / Review / Create New Review

POST {{URL}} api/v1/reviews

Params Authorization Headers (11) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   "review": "Loved it!",
3   "rating": 4,
4   "tour": "5c88fa8cf4afda39709c2951",
5   "user": "682da32af55f7ce393e30f41"
6 }
```

Body Cookies (1) Headers (21) Test Results

JSON Preview Visualize

```
1 {
2   "status": "success",
3   "data": {
4     "review": {
5       "_id": "682da3baf55f7ce393e30f47",
6       "review": "Loved it!",
7       "rating": 4,
8       "tour": "5c88fa8cf4afda39709c2951",
9       "user": "682da32af55f7ce393e30f41",
10      "createdAt": "2025-05-21T09:58:18.510Z",
11      "__v": 0,
12      "id": "682da3baf55f7ce393e30f47"
13    }
14  }
```

Get Review

GET {{URL}} api/v1/reviews

Params Authorization Headers (9) Body Scripts Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies (1) Headers (21) Test Results

JSON Preview Visualize

```
31 {
32   "_id": "682da3baf55f7ce393e30f47",
33   "review": "Loved it!",
34   "tour": {
35     "guides": [
36       {
37         "role": "guide",
38         "_id": "682cb8aaad40925ec430c628",
39         "name": "User",
40         "email": "guide@jonas.io"
41       }
42     ],
43     "_id": "5c88fa8cf4afda39709c2951",
44     "name": "The Forest Hiker",
45     "durationWeeks": null,
46     "id": "5c88fa8cf4afda39709c2951"
47   },
48   "user": {
49     "_id": "682da32af55f7ce393e30f41",
50     "name": "User"
51   },
52   "createdAt": "2025-05-21T09:58:18.510Z",
53   "__v": 0,
54   "id": "682da3baf55f7ce393e30f47"
55 }
```

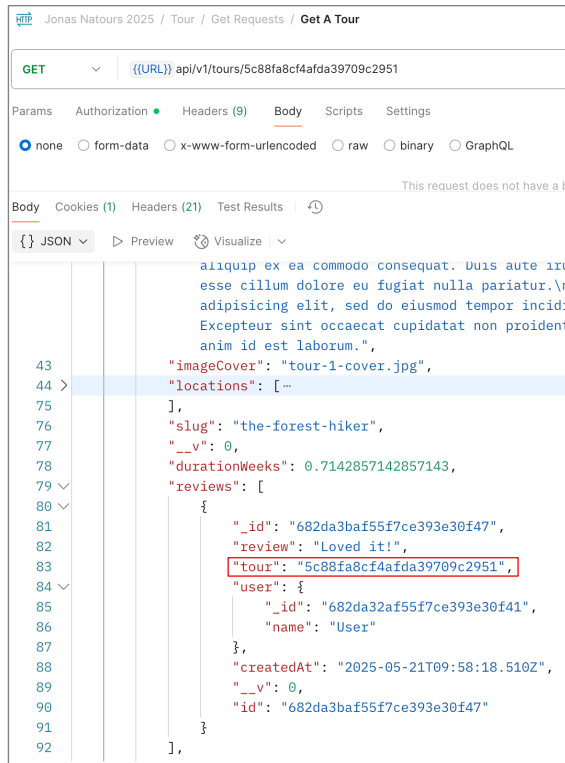
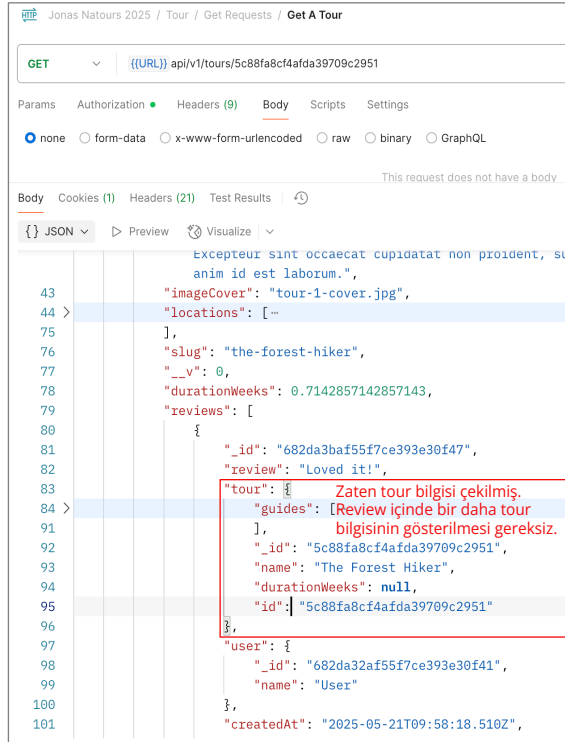
Turlar hakkında yapılan görüşleri **Review Schema** üzerinden veritabanına yazılır. Girilen bir **review** üzerinde **tourId** ve **userId** bulundurmak tour ve user dökümanlarının gereksiz büyümesini engeller. Gerektiğinde review üzerinde bulunan tourId ve userId ile ilgili dökümanların bilgileri **populate** metoduyla çekilebilir.

Modelling Data and Advanced Mongoose

Populate Fault!

Sy: 4

Get A Tour



reviewSchema Find Middleware

```
33 reviewSchema.pre(/^find/, function (next) {
34   this.populate({
35     path: 'tour',
36     select: 'name',
37   }).populate({
38     path: 'user',
39     select: 'name photo',
40   });
41   next();
42 });
```

reviewSchema Middleware Fix

```
1 reviewSchema.pre(/^find/, function (next) {
2   this.populate({
3     path: 'user',
4     select: 'name photo',
5   });
6   next();
7 });
```

tourSchema Middleware Fix

```
1 // Virtual populate
2 tourSchema.virtual('reviews', {
3   ref: 'Review',
4   foreignField: 'tour',
5   localField: '_id',
6 });
```

Virtual Property

tourControl.getTour()

```
1 exports.getTour = catchAsync(async (req, res, next) => {
2   const tour = await Tour.findById(req.params.id).populate('reviews');
3
4   if (!tour) {
5     return next(new AppError('No tour found with that ID', 404));
6   }
7
8   res.status(200).json({
9     status: 'success',
10    data: {
11      tour,
12    },
13  });
14 });
```

Review Schema üzerinde **tour** ve **user** için **populate** işlemi uygulanarak bir **tour** dokümanı çekildiğinde, **tour** içinde bulunan **review** bilgisi üzerinden yeniden **tour** bilgisi gelir. Bilginin bu şekilde sürekli tekrar etmesi istenen bir durum değildir. Bu durumu düzeltmek için **reviewSchema** üzerinde bulunan **find middleware** üzerinde ki **tour populate** silinir. **tourSchema** üzerinde gereksiz alan barındırmamak için **review** dökümanına ait bir bilgi saklanmaz. Fakat **tour** üzerinde **review** bilgisini görüntülemek gerektiğinde bu durum çözmek için **virtual property** tanımlanabilir. **Virtual Property** hafızada çalışan sanal bir alan olduğu için bilgisayarda yer kaplamaz. **tourSchema** üzerinde **reviews** için **virtual property** hazırlandıktan sonra, **tourControl.getTour()** metodunda **populate('reviews')** uygulanır.

Modelling Data and Advanced Mongoose

Simple Nested Routers

Sy: 5

Sign Up New User

```
POST /api/v1/users/signup

{
  "name": "jonas",
  "email": "hello@jonas.io",
  "password": "pass1234",
  "passwordConfirm": "pass1234"
}
```

```
{
  "status": "success",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZiI6IjY4MmUxNjhiNTFhZGQxNTg3NDExZWZlZSIsIm1ldCI6MTc0ZGZlZSIsIm5kaWZlIjoxNjI0ODkxZDQ6NzBqA21uZGJlbnR0e0P0V36D6G2bexk",
  "data": {
    "user": {
      "role": "user",
      "active": true,
      "_id": "682e168b51add1587411ee3e",
      "name": "jonas",
      "email": "hello@jonas.io",
      "__v": 0
    }
  }
}
```

Get A Tour

```
GET /tours/5c88fa8cf4afda39709c2951

{
  "secretTour": false,
  "guides": [],
  "_id": "5c88fa8cf4afda39709c2951",
  "name": "The Forest Hiker",
  "duration": 5,
  "maxGroupSize": 25,
  "difficulty": "easy",
  "price": 397,
  "summary": "Breathtaking hike through the Canadian Banff National Park",
  "rating": 5,
  "review": "Best tour in the world!"
}
```

Create New Review on Tour

```
POST /tours/5c88fa8cf4afda39709c2951/reviews

{
  "rating": 5,
  "review": "Best tour in the world!"
}
```

```
{
  "status": "success",
  "data": {
    "review": {
      "_id": "682e17ecef77e55c84643676",
      "rating": 5,
      "review": "Best tour in the world!",
      "tour": "5c88fa8cf4afda39709c2951",
      "user": "682e168b51add1587411ee3e",
      "createdAt": "2025-05-21T18:14:04.012Z",
      "__v": 0,
      "id": "682e17ecef77e55c84643676"
    }
  }
}
```

tourRoute

:tourId
/tours/5c88fa8cf4afda39709c2951/reviews

```
1 const express = require('express');
2 const authController = require('../controllers/authController');
3 const reviewController = require('../controllers/reviewController');
4
5 const router = express.Router();
6
7 // POST /tour/234fad4/reviews
8 // GET /tour/234fad4/reviews
9 // GET /tour/234fad4/reviews/94887fda
10
11 router
12   .route('/:tourId/reviews')
13   .post(
14     authController.protect,
15     authController.restrictTo('user'),
16     reviewController.createReview
17   );
18
19 module.exports = router;
```

createReview

```
1 const Review = require('../models/reviewModel');
2 const catchAsync = require('../utils/catchAsync');
3
4 exports.createReview = catchAsync(async (req, res, next) => {
5   // Allow nested routers
6   if (!req.body.tour) req.body.tour = req.params.tourId;
7   if (!req.body.user) req.body.user = req.user.id;
8
9   const newReview = await Review.create(req.body);
10
11   res.status(201).json({
12     status: 'success',
13     data: {
14       review: newReview,
15     },
16   });
17 });
```

```
_id: ObjectId('682e17ecef77e55c84643676')
rating: 5
review: "Best tour in the world!"
tour: ObjectId('5c88fa8cf4afda39709c2951')
user: ObjectId('682e168b51add1587411ee3e')
createdAt: 2025-05-21T18:14:04.012+00:00
__v: 0
```

Bir review kaydını **/tours:tourdId/reviews** route'u üzerinden yapmak gerekir. review kaydı bu şekilde yapılırsa review için gerekli olan tourId verisi route üzerinden alınır ve tourId manuel olarak girilmez.

createReview metodu incelendiğinde review için gerekli olan tourId ve userId bilgilerinin nasıl alındığı görülmektedir.

tourId **/tours:tourdId/reviews** üzerinden params ile userId protect fonksiyonu ile login olan kullanıcı bilgileri üzerinden alınmaktadır.