

M O N G O O S E -  
M O D E L I N G  
R E L A T I O N S H I P S  
B E T W E E N  
C O N N E C T E D  
D A T A

P R O J E C T





# PROJECT- BUILD THE RENTALS API

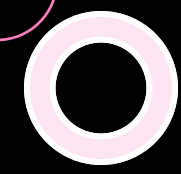
NOW LET TAKES AN APPLICATION  
TO THE NEXT LEVEL



# BUILD THE RENTALS API

Create a new  
rental  
POST /api/rentals

Get the list of  
rentals  
Get /api/rentals





# BUILD THE RENTALS API - SOLUTION

Create a new file rental.  
into folder 'models'

Create a new file rentals.js  
into folder 'routes'



# rental.js



```
const Joi = require('joi');
const mongoose = require('mongoose');

const Rental = mongoose.model('Rental', new mongoose.Schema({
  customer: {
    ...
  },
  movie: {
    ...
  },
  dateOut: {
    type: Date,
    required: true,
    default: Date.now
  },
  dateReturned: {
    type: Date
  },
  rentalFee: {
    type: Number,
    min: 0
  }
}));

function validateRental(rental) {
  const schema = Joi.object({
    customerId: Joi.string().required(),
    movieId: Joi.string().required()
  });

  return schema.validate(rental);
}

exports.Rental = Rental;
exports.validate = validateRental;
```

```
customer: {
  type: new mongoose.Schema({
    name: {
      type: String,
      required: true,
      minlength: 5,
      maxlength: 50
    },
    isGold: {
      type: Boolean,
      default: false
    },
    phone: {
      type: String,
      required: true,
      minlength: 5,
      maxlength: 50
    }
  }),
  required: true
},
```

```
movie: {
  type: new mongoose.Schema({
    title: {
      type: String,
      required: true,
      trim: true,
      minlength: 5,
      maxlength: 255
    },
    dailyRentalRate: {
      type: Number,
      required: true,
      min: 0,
      max: 255
    }
  }),
  required: true
}
```



```
const {Rental, validate} = require('../models/rental');
const {Movie} = require('../models/movie');
const {Customer} = require('../models/customer');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.get('/', async (req, res) => {
  const rentals = await Rental.find().sort('-dateOut');
  res.send(rentals);
});

router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);
  const customer = await Customer.findById(req.body.customerId);
  if (!customer) return res.status(400).send('Invalid customer. ');
  const movie = await Movie.findById(req.body.movieId);
  if (!movie) return res.status(400).send('Invalid movie. ');
  if (movie.numberInStock === 0) return res.status(400).send('Movie not in stock. ');

  let rental = new Rental({
    customer: {
      _id: customer._id,
      name: customer.name,
      phone: customer.phone
    },
    movie: {
      _id: movie._id,
      title: movie.title,
      dailyRentalRate: movie.dailyRentalRate
    }
  });

  rental = await rental.save();
  movie.numberInStock--;
  movie.save();

  res.send(rental);
});

router.get('/:id', async (req, res) => {
  const rental = await Rental.findById(req.params.id);

  if (!rental) return res.status(404).send('The rental with the given ID was not found. ');
  res.send(rental);
});

module.exports = router;
```





# POSTMAN

In postman create new rental

http://localhost:3000/api/rentals

POST http://localhost:3000/api/rentals

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON


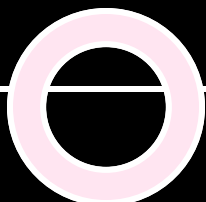

```
1
2  .... "customerId": "613c8e3551b72f0052007466",
3  .... "movieId": "613c8ecc0bae5ded8684dbf6"
4
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 56 ms Size: 499 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2  "customer": {
3    "name": "Marry",
4    "isGold": false,
5    "phone": "+37505888555",
6    "_id": "613c8e3551b72f0052007466"
7  },
8  "movie": {
9    "title": "Terminator",
10   "dailyRentalRate": 2,
11   "_id": "613c8ecc0bae5ded8684dbf6"
12 },
13 "_id": "613cbe509d760dd5a63ac833",
14 "dateOut": "2021-09-11T14:33:52.147Z",
15 "__v": 0
16
```





ObjectIDs are generated by MongoDB driver and are used to uniquely identify a document.

They consist of 12 bytes:

- 4 bytes: timestamp
- 3 bytes: machine identifier
- 2 bytes: process identifier
- 3 bytes: counter

ObjectIDs are almost unique. In theory, there is a chance for two ObjectIDs to be equal but the odds are very low (1/16,000,000) for most real-world applications







`npm i joi-objectid`

# Validating ObjectIDs

```
const Joi = require('joi');
Joi.objectId = require('joi-objectid')(Joi);
const mongoose = require('mongoose');

const Rental = mongoose.model('Rental', new mongoose.Schema({
  ...
}));

function validateRental(rental) {
  const schema = Joi.object({
    customerId: Joi.objectId().required(),
    movieId: Joi.objectId().required()
  });

  return schema.validate(rental);
}

exports.Rental = Rental;
exports.validate = validateRental;
```

Insert code

Replace code





# POSTMAN

In postman create new rental

http://localhost:3000/api/rentals Save

**POST** ▼ http://localhost:3000/api/rentals Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies Beautify

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

```
1 {
2   ... "customerId": "1234",
3   ... "movieId": "613c8ecc0bae5ded8684dbf6"
4 }
```

Body Cookies Headers (7) Test Results 🌐 Status: 400 Bad Request Time: 29 ms Size: 309 B Save Response ▼

Pretty Raw Preview Visualize **HTML** ▼ ≡

```
1 "customerId" with value "1234" fails to match the valid mongo id pattern
```

