

EdX and its Members use cookies and other tracking technologies for performance, analytics, and marketing purposes. By using this website, you accept this use. Learn more about these technologies in the [Privacy Policy](#).



[Course](#) > [Workin...](#) > [Creatin...](#) > [Custo...](#)

Custom Mongoose Methods

Video: Custom Mongoose Methods

```
mongoose.js
1  const mongoose = require('mongoose')
2  mongoose.Promise = global.Promise
3  mongoose.connect('mongodb://localhost:27017/edx-course-db', {useMongoClient: true})
4
5  const bookSchema = mongoose.Schema({ name: String })
6
7  bookSchema.method({
8    buy(quantity, customer, callback) {
9      var bookToPurchase = this
10     console.log('buy')
11     return callback()
12   },
13   refund(customer, callback) {
14     //process the refund
15     console.log('refund')
16     return callback()
17   }
18 })
19
20 bookSchema.static({
21   getZeroInventoryReport(callback) {
22     //run a query on all books and get the ones with zero inventory
23     console.log('getZeroInventoryReport')
24     let books = []
25     return callback(books)
26   },
27   getCountOfBooksById(bookId, callback) {
28     //run a query and get the number of books left for a given book
29     console.log('getCountOfBooksById')
30     let count = 0
```

0:15 / 3:49

1.50x



Video

[Download video file](#)

Transcripts

[Download SubRip \(.srt\) file](#)

[Download Text \(.txt\) file](#)

Custom Static and Instance Methods

In addition to dozens of built-in Mongoose model methods, we can add custom ones. For example, to initiate a purchase, we can call the `buy` method on the document `practicalNodeBook` after we implement the custom instance method `buy()`:

```
bookSchema.method({
  buy: function(quantity, customer, callback) {
    var bookToPurchase = this
    //create a purchase order and invoice customer
    return callback(results)
  },
  refund: function(customer, callback) {
    //process the refund
    return callback(results)
  }
})
```

Static methods are useful when we either don't have a particular document object or we don't need it:

```
bookSchema.static({
  getZeroInventoryReport: function(callback) {
    //run a query on all books and get the ones with zero inventory
    return callback(books)
  },
  getCountOfBooksById: function(bookId, callback){
    //run a query and get the number of books left for a given book
    return callback(count)
  }
})
```

Hooks for Keeping Code Organized

In a complex application with a lot of interrelated objects, we might want to execute certain logic before saving an object. Hooks are a good place to store such logic. For example, we might want to upload a PDF to the web site before saving a book document:

```
bookSchema.pre('save', function(next) {  
  //prepare for saving  
  //upload PDF  
  return next()  
})
```

On the other hand, before removing, we need to make sure there are no pending purchase orders for this book:

```
bookSchema.pre('remove', function(next) {  
  //prepare for removing  
  return next(e)  
})
```

Hooks are triggered before or after an event is executed. Hooks allow you to place business logic where it's best suited - in the model schemas.

Note: Hooks and methods must be added to the schemas before compiling them to models—in other words, before calling the `mongoose.model()` method.