

13. Working With Mongoose

* Chapter 205: Module Introduction



What's In This Module?

What is Mongoose?

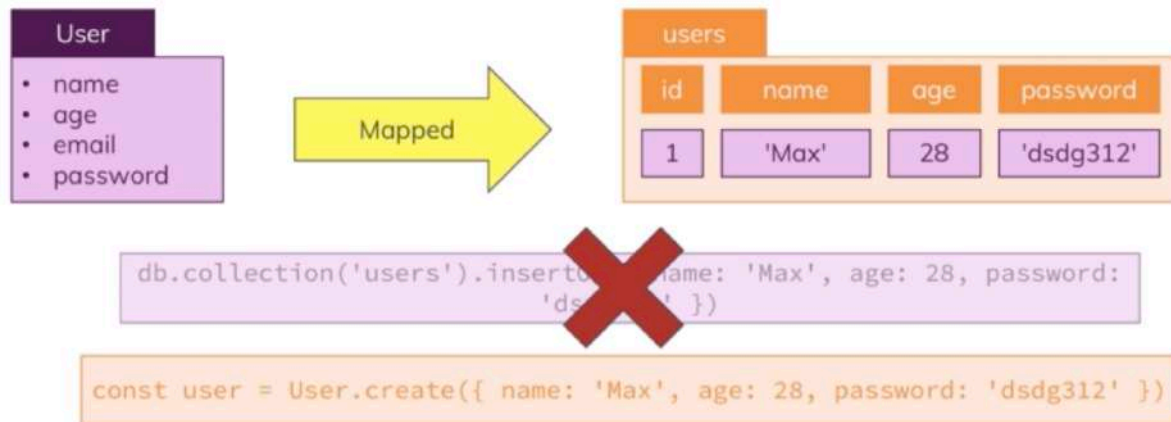
Using Mongoose in Node.js Apps

35 slides

* Chapter 206: What is Mongoose?

What is Mongoose?

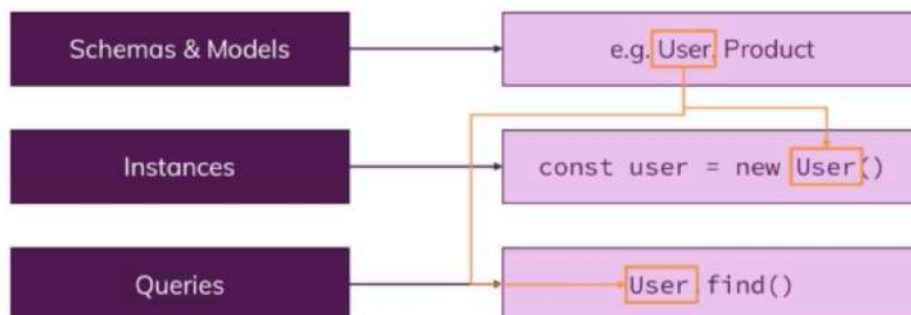
A Object-Document Mapping Library



35 slides

- Mongoose is an ORM standing for 'Object-Document Mapping' Library which is really similar to sequelize which was an ORM standing for 'Object-Relational Mapping' Library.
 - difference between is that MongoDB is not relational database. it's document database.
 - Mongoose allows us to define models with which we then work and where all the queries are done behind the scene which doesn't means that we can't influence and that we can't change somethings.
-

Core Concepts

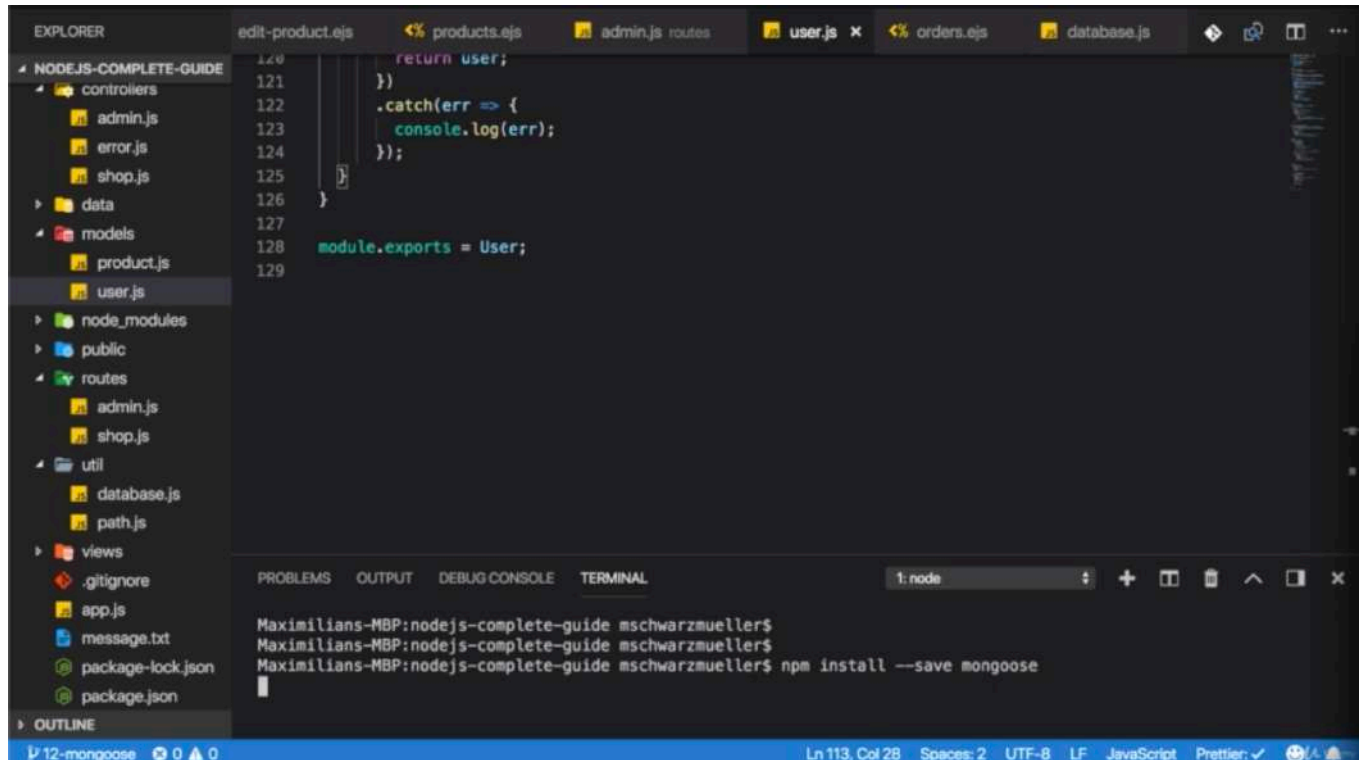


35 slides

* Chapter 207: Connecting To The MongoDB Server With Mongoose

1. update
- delete ./util/database.js

- app.js
- ./routes/shop.js
- ./routes/admin.js
- ./models/product.js
- ./models/user.js



- and we can delete ./util/database.js file and we can go to app.js file and in there, import mongoose.
- in here, it looks like we are connected because we don't get any error here and therefore we are connected to our same MongoDB server by using the Mongoose package.

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose')
8
9 const errorController = require('./controllers/error');
10 const User = require('./models/user')
11
12 const app = express();
13
14 app.set('view engine', 'ejs');
15 app.set('views', 'views');
16
17 const adminRoutes = require('./routes/admin');
18 const shopRoutes = require('./routes/shop');
19
20 app.use(bodyParser.urlencoded({ extended: false }));
21 app.use(express.static(path.join(__dirname, 'public')));
22
23 app.use((req, res, next) => {
24   User.findById('5cb7d12855f5be74b129c0b7c')
25     .then(user => {

```

```

26     req.user = new User(user.name, user.email, user.cart, user._id);
27     next();
28   })
29   .catch(err => console.log(err));
30 });
31
32 app.use('/admin', adminRoutes);
33 app.use(shopRoutes);
34
35 app.use(errorController.get404);
36
37 /**we already have everything in place we need to connect
38  * and mongoose will manage that one connection behind the scenes.
39  * so that in other places where we start using mongoose from the mongoose package,
40  * we use that same connection we set up here.
41  */
42 mongoose
43   .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
  retryWrites=true')
44   .then(result => {
45     app.listen(3000)
46   })
47   .catch(err => {
48     console.log(err)
49   })

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 /*
12 router.get('/', shopController.getIndex);
13
14 router.get('/products', shopController.getProducts);
15
16 router.get('/products/:productId', shopController.getProduct);
17
18 router.get('/cart', shopController.getCart);
19
20 router.post('/cart', shopController.postCart);
21
22 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
23
24 router.post('/create-order', shopController.postOrder);
25
26 router.get('/orders', shopController.getOrders);
27 */
28
29 module.exports = router;
30

```

```

1 // ./routes/admin.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const adminController = require('../controllers/admin');
8
9 const router = express.Router();
10
11 /*
12 // /admin/add-product => GET
13 router.get('/add-product', adminController.getAddProduct);
14
15 // /admin/products => GET
16 router.get('/products', adminController.getProducts);
17
18 // /admin/add-product => POST
19 router.post('/add-product', adminController.postAddProduct);
20
21 router.get('/edit-product/:productId', adminController.getEditProduct);
22
23 router.post('/edit-product', adminController.postEditProduct);
24
25 router.post('/delete-product', adminController.postDeleteProduct);
26 */
27
28 module.exports = router;

```

```

1 //./models/product.js
2
3 /*
4 const mongodb = require('mongodb');
5 const getDb = require('../util/database').getDb;
6
7 class Product {
8   constructor(title, price, description, imageUrl, id, userId) {
9     this.title = title;
10    this.price = price;
11    this.description = description;
12    this.imageUrl = imageUrl;
13    this._id = id ? new mongodb.ObjectId(id) : null
14    this.userId = userId
15  }
16
17  save() {
18    const db = getDb();
19    let dbOp;
20    if (this._id) {
21      // Update the product
22      dbOp = db
23        .collection('products')
24        .updateOne({ _id: this._id }, { $set: this });
25    } else {
26      dbOp = db.collection('products').insertOne(this);
27    }

```

```

28     return dbOp
29     .then(result => {
30         console.log(result);
31     })
32     .catch(err => {
33         console.log(err);
34     });
35 }
36
37 static fetchAll() {
38     const db = getDb();
39     return db
40         .collection('products')
41         .find()
42         .toArray()
43         .then(products => {
44             console.log(products);
45             return products;
46         })
47         .catch(err => {
48             console.log(err);
49         });
50 }
51
52 static findById(prodId) {
53     const db = getDb();
54     return db
55         .collection('products')
56         .find({ _id: new mongodb.ObjectId(prodId) })
57         .next()
58         .then(product => {
59             console.log(product);
60             return product;
61         })
62         .catch(err => {
63             console.log(err);
64         });
65 }
66
67 static deleteById(prodId) {
68     const db = getDb();
69     return db
70         .collection('products')
71         .deleteOne({ _id: new mongodb.ObjectId(prodId) })
72         .then(result => {
73             console.log('Deleted');
74         })
75         .catch(err => {
76             console.log(err);
77         });
78 }
79 }
80
81 module.exports = Product;
82 */

```

```

2
3 /*
4 const mongodb = require('mongodb');
5 const getDb = require('../util/database').getDb;
6
7 const ObjectId = mongodb.ObjectId;
8
9 class User {
10   constructor(username, email, cart, id) {
11     this.name = username;
12     this.email = email;
13     this.cart = cart; // {items: []}
14     this._id = id;
15   }
16
17   save() {
18     const db = getDb();
19     return db.collection('users').insertOne(this);
20   }
21
22   addToCart(product) {
23     const cartProductIndex = this.cart.items.findIndex(cp => {
24       return cp.productId.toString() === product._id.toString();
25     });
26     let newQuantity = 1;
27     const updatedCartItems = [...this.cart.items];
28     if (cartProductIndex >= 0) {
29       newQuantity = this.cart.items[cartProductIndex].quantity + 1;
30       updatedCartItems[cartProductIndex].quantity = newQuantity;
31     } else {
32       updatedCartItems.push({
33         productId: new ObjectId(product._id),
34         quantity: newQuantity
35       });
36     }
37     const updatedCart = {
38       items: updatedCartItems
39     };
40     const db = getDb();
41     return db
42       .collection('users')
43       .updateOne(
44         { _id: new ObjectId(this._id) },
45         { $set: { cart: updatedCart } }
46       );
47   }
48
49   getCart() {
50     const db = getDb();
51     const productIds = this.cart.items.map(i => {
52       return i.productId;
53     });
54     return db
55       .collection('products')
56       .find({ _id: { $in: productIds } })
57       .toArray()

```

```

58 .then(products => {
59   return products.map(p => {
60     return {
61       ...p,
62       quantity: this.cart.items.find(i => {
63         return i.productId.toString() === p._id.toString();
64       }).quantity
65     };
66   });
67 });
68 }
69
70 deleteItemFromCart(productId){
71   const updatedCartItems = this.cart.items.filter(item => {
72     return item.productId.toString() !== productId.toString()
73   })
74   const db = getDb()
75   return db
76     .collection('users')
77     .updateOne(
78       { _id: new ObjectId(this._id) },
79       { $set: { cart: { items: updatedCartItems } } }
80     )
81 }
82
83 addOrder(){
84   const db = getDb()
85   return this.getCart().then(products => {
86     const order = {
87       items: products,
88       user: {
89         _id: new ObjectId(this._id),
90         name: this.name,
91       }
92     }
93     return db
94       .collection('orders')
95       .insertOne(order)
96   })
97   .then(result => {
98     this.cart = { items: [] }
99     return db
100       .collection('users')
101       .updateOne(
102         { _id: new ObjectId(this._id) },
103         { $set: { cart: { items: [] } } }
104       )
105   })
106 }
107
108 getOrders(){
109   const db = getDb()
110   return db
111     .collection('orders')
112     .find({ 'user._id': new ObjectId(this._id) })
113     .toArray()

```



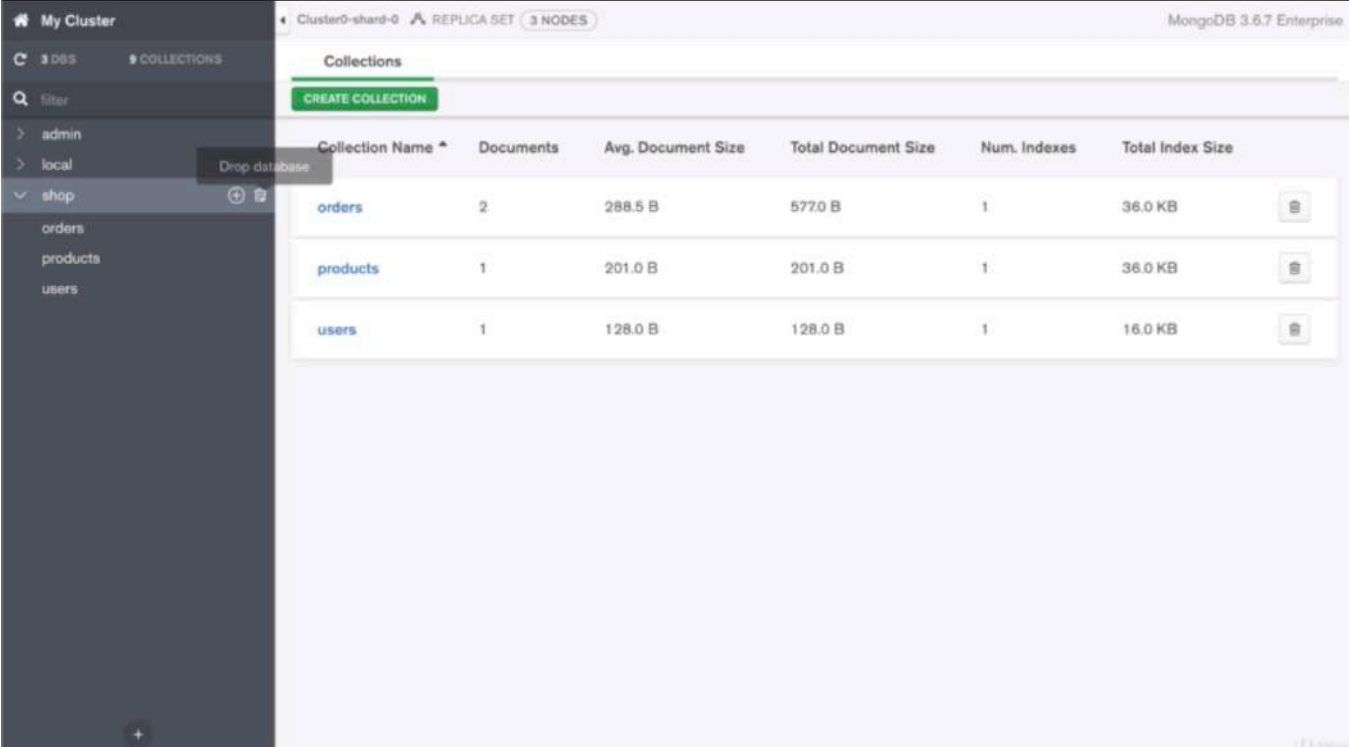
```

114 }
115
116 static findById(userId) {
117     const db = getDb();
118     return db
119         .collection('users')
120         .findOne({ _id: new ObjectId(userId) })
121         .then(user => {
122             console.log(user);
123             return user;
124         })
125         .catch(err => {
126             console.log(err);
127         });
128 }
129 }
130
131 module.exports = User;
132 */

```

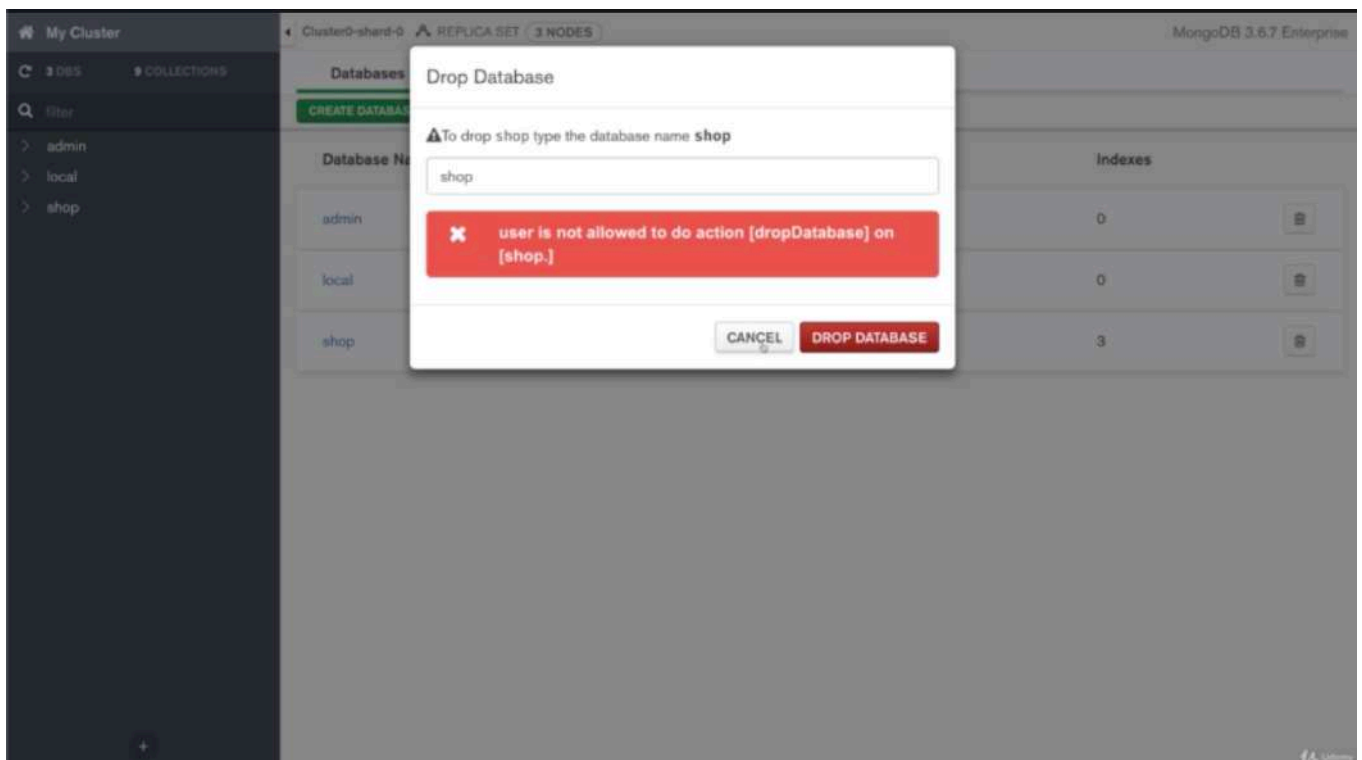
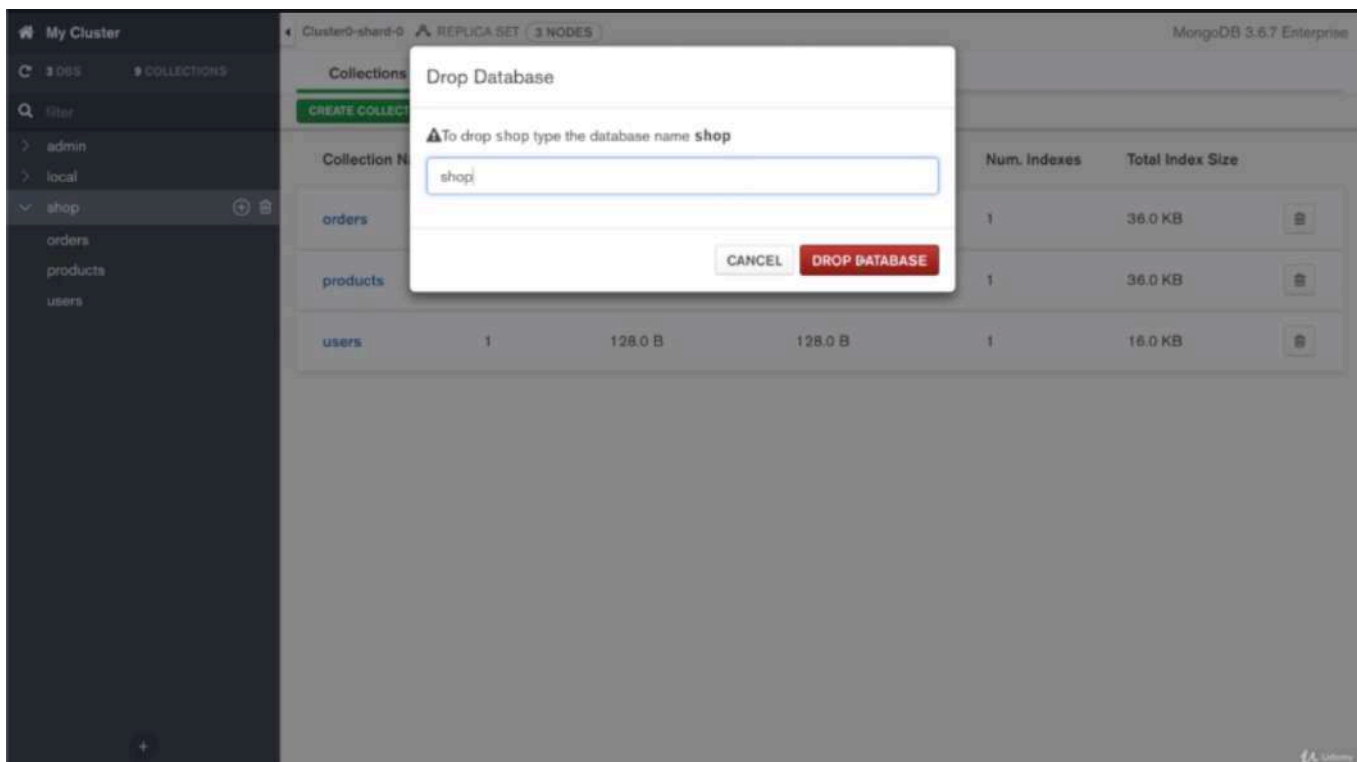
* Chapter 208: Creating The Product Schema

1. update
- ./models/product.js



The screenshot shows the MongoDB Enterprise 3.6.7 interface. On the left, a sidebar lists databases: 'admin', 'local', and 'shop'. The 'shop' database is selected, and a 'Drop database' tooltip is visible. The main panel displays the 'Collections' tab for the 'shop' database. A table lists the collections: 'orders', 'products', and 'users'. Each row shows the number of documents, average document size, total document size, number of indexes, and total index size. There is a 'CREATE COLLECTION' button at the top of the table.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
orders	2	288.5 B	577.0 B	1	36.0 KB
products	1	201.0 B	201.0 B	1	36.0 KB
users	1	128.0 B	128.0 B	1	16.0 KB



- time to fix our code and make it work again. for that first of all, i connected to my MongoDB server with MongoDB Compass again and there i wanna clear everything so that we can start from scratch.
 - therefore, i will go to my 'shop' database and simply delete that entire database.
 - but we got problem that i connected with the wrong user where i'm not allowed to delete a database because i connected with a user who has only read or write access.
-
-
-

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.7 Enterprise

Collections

CREATE COLLECTION

Collection Name

products

users

Drop Collection

⚠ To drop shop.products type the collection name **products**

products

CANCEL DROP COLLECTION

	Num. Indexes	Total Index Size	
1	36.0 KB		
1	16.0 KB		

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.7 Enterprise

Collections

CREATE COLLECTION

Collection Name

users

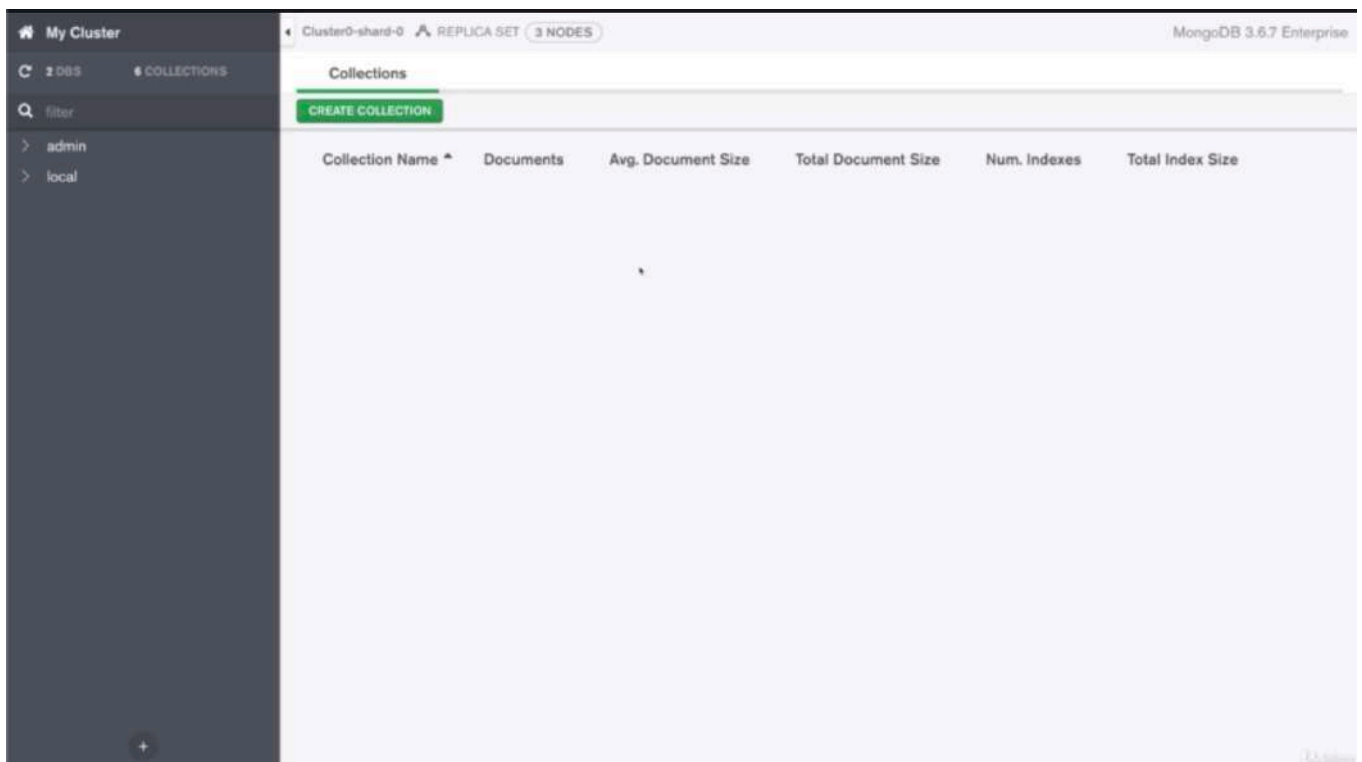
Drop Collection

⚠ To drop shop.users type the collection name **users**

users

CANCEL DROP COLLECTION

	Num. Indexes	Total Index Size	
1	16.0 KB		



- so i will just delete the collections here. the alternative would be to simply connect with a user where i'm allowed to manage the overall database.

- i got rid of the shop database and now we can start working from scratch again.

```

1 //./models/product.js
2
3 const mongoose = require('mongoose')
4
5 /**'Schema' constructor allows me to create new schemas */
6 const Schema = mongoose.Schema;
7
8 /**you now define the data schema of a product in our case here.
9  * you don't just define which keys you have
10  * but also which type these keys will have.
11  */
12 const productSchema = new Schema({
13   /** this would say
14    * OK so i create a schema for an object
15    * which i will eventually be able to work with
16    * which must have or which will have a title that is of type 'String'
17    *
18    * MongoDB is Schemaless,
19    * so why do we start to create Schemas?
20    * the idea is that whilst we have the flexibility of not being restricted to specific
21    schema,
22    * we will have a certain structure in the data we work with
23    * and therefore Mongoose wanna give you the advantage of focusing on your data
24    * but for that, it needs to know how your data looks like
25    * and therefore we define such a schema for the structure our data will have.
26    *
27    * we could even work with a product and create a new one and save it to the database
28    without setting a title
29    * because we still have the flexibility of not enforcing this,
30    * though what we can do is we can pass an object instead of the type as a value
31    * and then set a type property which could be set to 'String'

```

```

30  * and then set required to true
31  * this is a more complex way of configuring the value for this key.
32  *
33  * we would say, the type of this is a string as before
34  * but it's also required
35  * and now we give up some of the flexibility we had before
36  * and we force all objects to have a title
37  * but in the end, in our application,
38  * every product needs to have a title
39  * because we will run into other errors otherwise.
40  */
41  title: {
42    type: String,
43    required: true
44  },
45  price: {
46    type: Number,
47    required: true
48  },
49  description: {
50    type: String,
51    required: true
52  },
53  imageUrl: {
54    type: String,
55    required: true
56  }
57  /**i don't add '_id'
58   * because this will still be added automatically as an objectId
59   * so we don't need to define here.
60   */
61 })

```

* Chapter 209: Saving Data Through Mongoose

1. update
- ./model/product.js
- ./controllers/admin.js
- app.js
- ./routes/admin.js

Title

Image URL

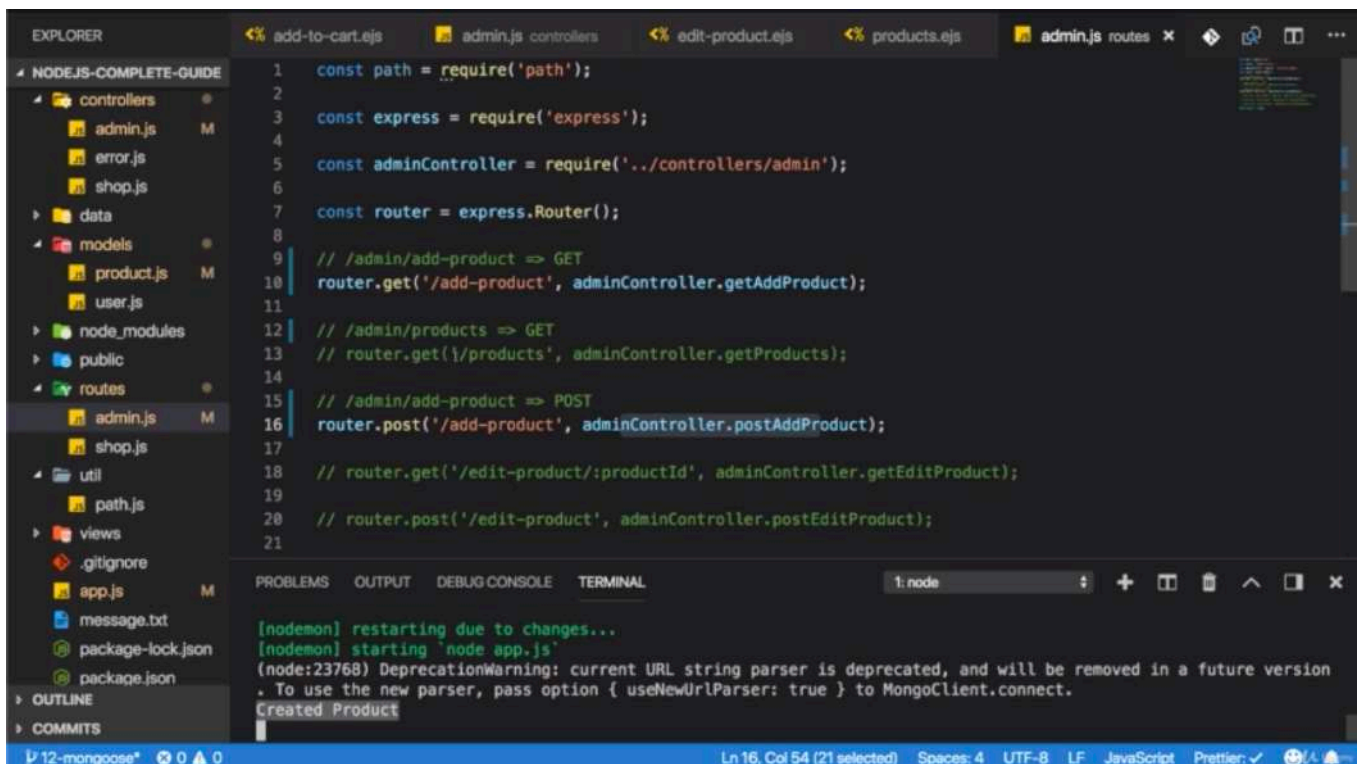
Price

Description

Page Not Found!

8

- let's test this.
 - we get a 'page not found' because we can't load any other pages, that is OK.
-



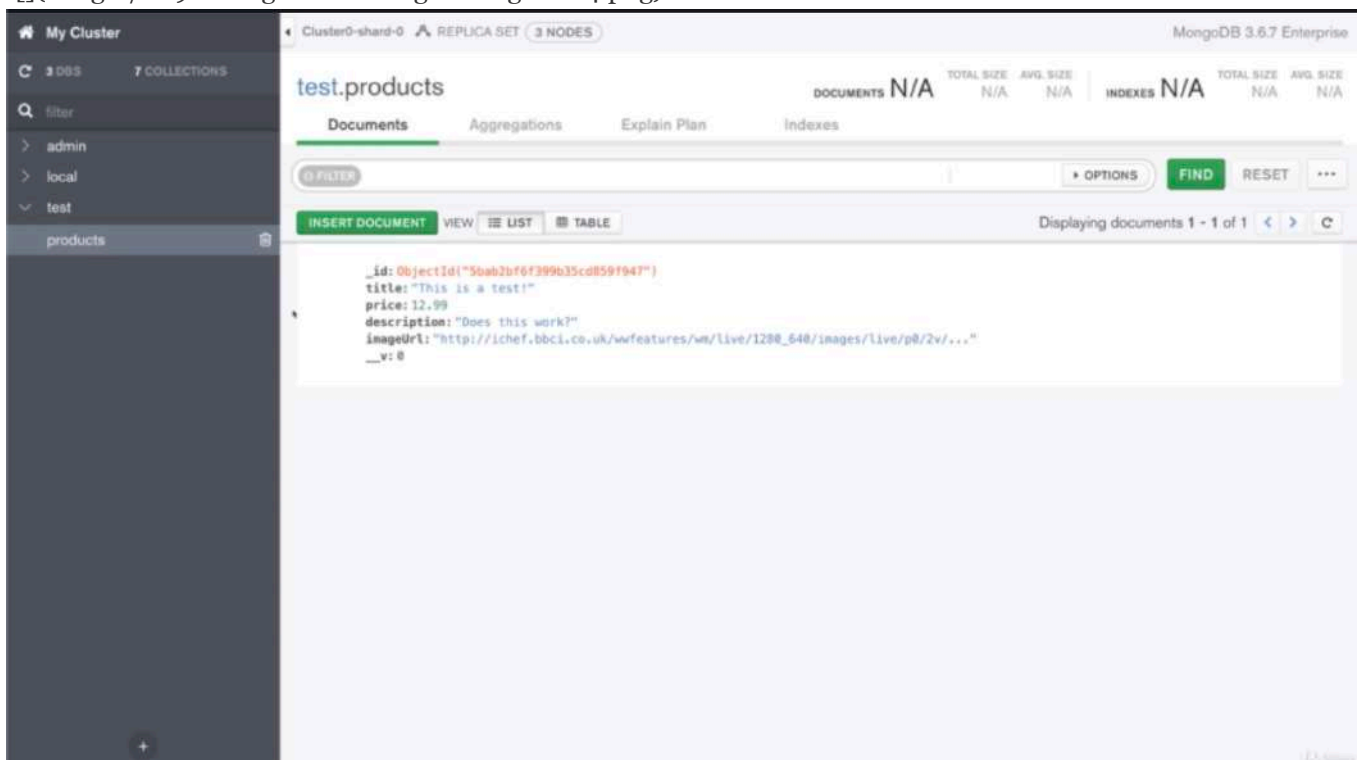
The screenshot shows a VS Code editor with the file explorer on the left displaying a project structure. The main editor window shows the `admin.js` routes file with the following code:

```
1 const path = require('path');
2
3 const express = require('express');
4
5 const adminController = require('../controllers/admin');
6
7 const router = express.Router();
8
9 // /admin/add-product => GET
10 router.get('/add-product', adminController.getAddProduct);
11
12 // /admin/products => GET
13 // router.get('/products', adminController.getProducts);
14
15 // /admin/add-product => POST
16 router.post('/add-product', adminController.postAddProduct);
17
18 // router.get('/edit-product/:productId', adminController.getEditProduct);
19
20 // router.post('/edit-product', adminController.postEditProduct);
21
```

The terminal window at the bottom shows the following output:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:23768) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Created Product
```

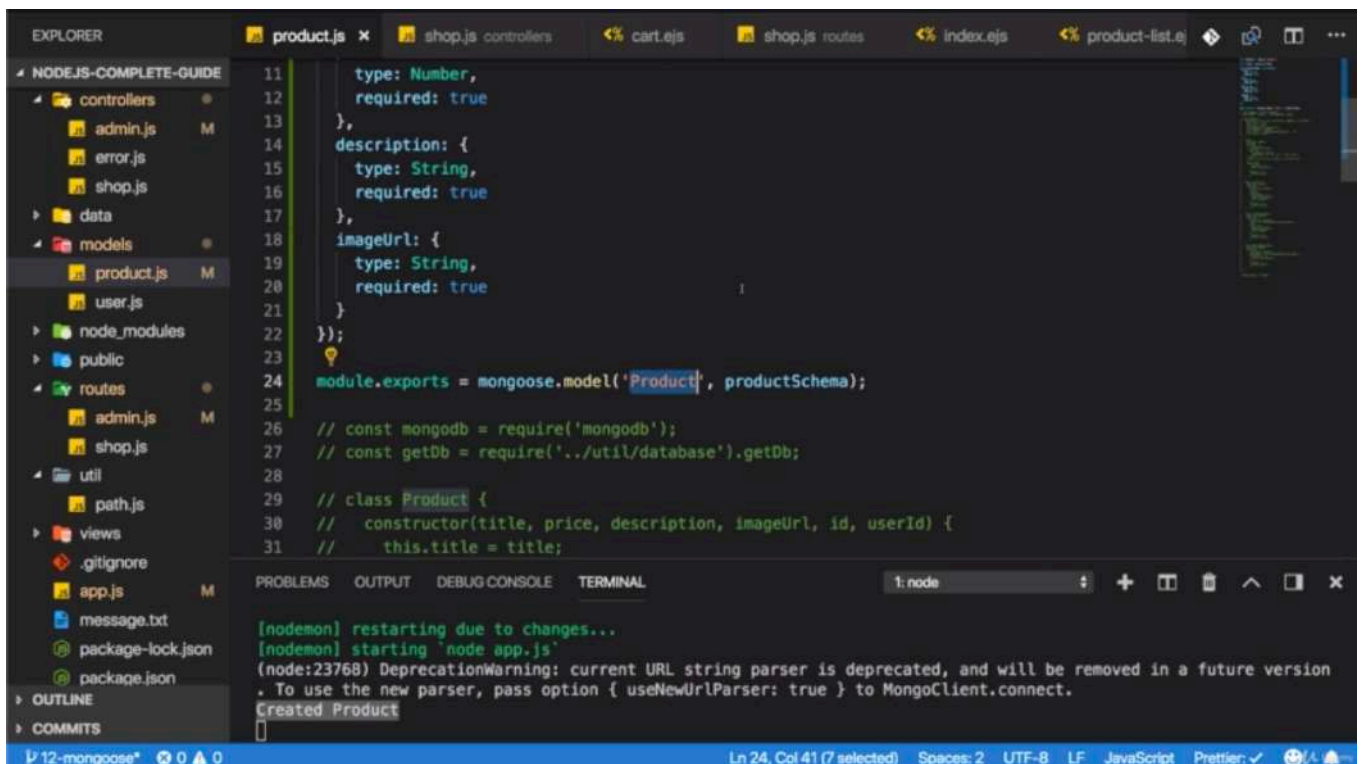
- in the code, i got no error and created product which looks good



- and in MongoDB Compass, let's refresh.

- i connected to the wrong url, i will fix that later.

- i'm connected to the 'test' database instead of the shop database. theoretically it worked. we get a product collection with the product added.



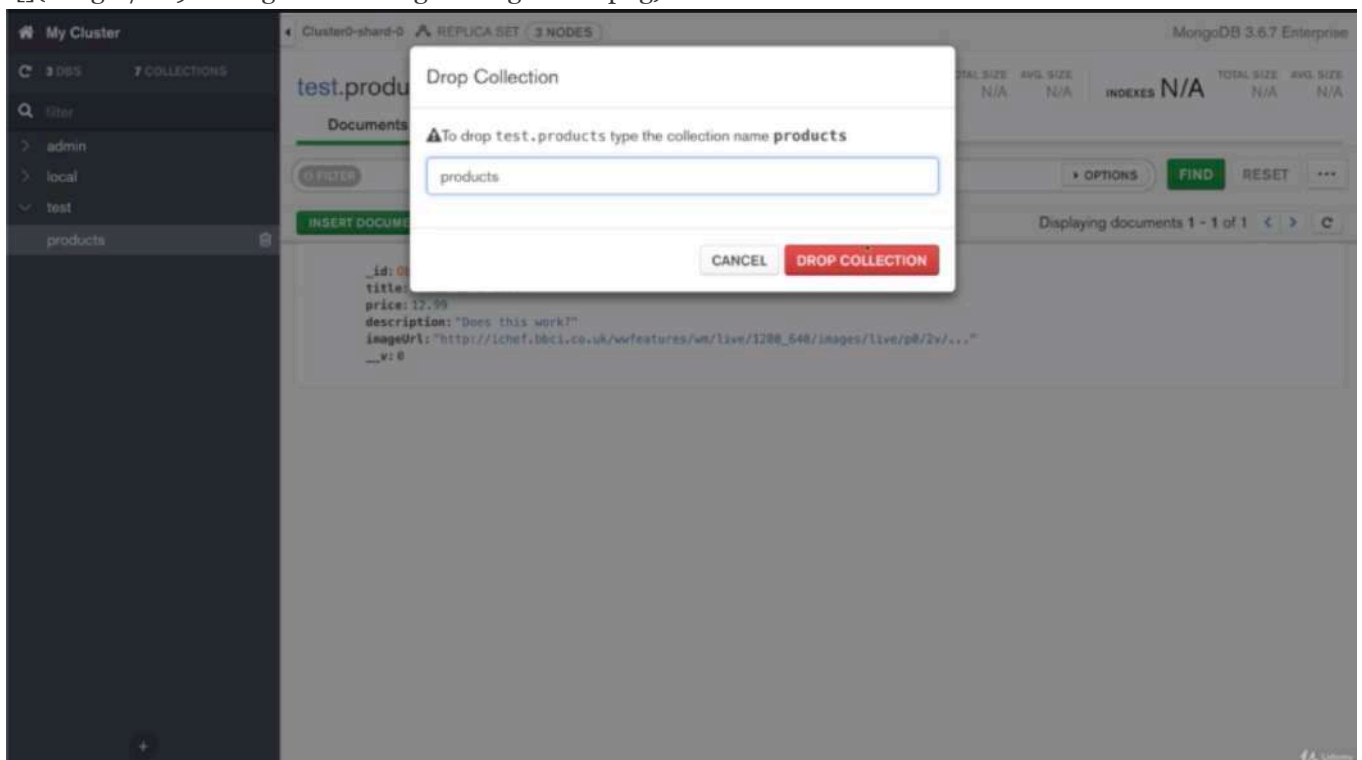
The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like controllers, data, models, node_modules, public, routes, util, views, and files like app.js, message.txt, package-lock.json, and package.json. The code editor shows the product.js file with the following code:

```
11 type: Number,
12 required: true
13 },
14 description: {
15 type: String,
16 required: true
17 },
18 imageUrl: {
19 type: String,
20 required: true
21 }
22 });
23
24 module.exports = mongoose.model('Product', productSchema);
25
26 // const mongoose = require('mongoose');
27 // const getDb = require('../util/database').getDb;
28
29 // class Product {
30 //   constructor(title, price, description, imageUrl, id, userId) {
31 //     this.title = title;
```

The terminal at the bottom shows the following output:

```
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:23768) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version.
To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Created Product
```

- where is the products collection coming from? we never defined that name.
- Mongoose takes your model name, so 'Product', turns it to all lowercase and takes the plural form of that. that will then be used as a collection name.



- i will still drop this collection and quickly fix my connection setting in app.js file

The screenshot shows a VS Code editor with the following components:

- EXPLORER:** A file tree on the left showing a project structure with folders like `controllers`, `data`, `models`, `node_modules`, `public`, `routes`, `util`, and `views`. Files include `admin.js`, `error.js`, `shop.js`, `product.js`, `user.js`, `app.js`, `message.txt`, `package-lock.json`, and `package.json`.
- EDITOR:** The main area shows `app.js` with the following code:

```
33 p.use(errorController.get404);
34
35 mongoose
36 .connect(
37   'mongodb+srv://maximilian:9u4biljMQc4jjqbe@cluster0-ntrwp.mongodb.net/shop?retryWrites=true'
38 )
39 .then(result => {
40   app.listen(3000);
41 })
42 .catch(err => {
43   console.log(err);
44 });
45
```
- TERMINAL:** The bottom panel shows the output of a terminal session:

```
Created Product
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:23827) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
```
- STATUS BAR:** At the bottom, it shows '12-mongoose*' and 'Ln 37, Col 75 (4 selected) Spaces: 2 UTF-8 LF JavaScript Prettier: ✓'.

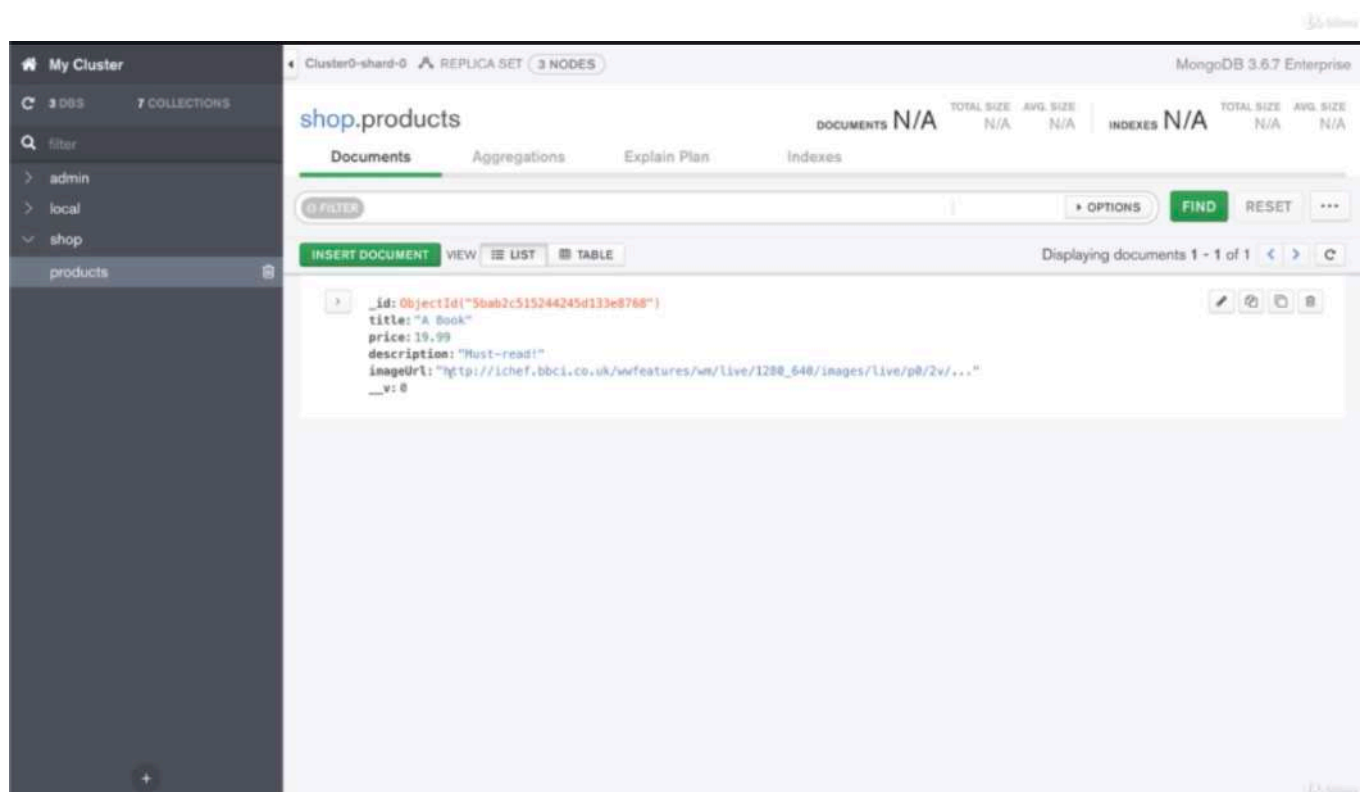
Title

Image URL

Price

Description

Page Not Found!



Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.7 Enterprise

shop.products

DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

Filter

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5bab2c515244245d133e8768"),
  "title": "A Book",
  "price": 19.99,
  "description": "Must-read!",
  "imageUrl": "http://ichef.bbci.co.uk/wwfeatures/wn/Live/1280_640/images/live/p0/2v/...",
  "__v": 0
}
```

```
1 //./models/product.js
2
3 const mongoose = require('mongoose')
4
5 const Schema = mongoose.Schema;
6
7 const productSchema = new Schema({
8   title: {
9     type: String,
10    required: true
11  },
12  price: {
13    type: Number,
14    required: true
15  }
16 })
```

```

15   },
16   description: {
17     type: String,
18     required: true
19   },
20   imageUrl: {
21     type: String,
22     required: true
23   }
24 })
25
26 /**Mongoose also works with so-called models
27  * and the model is also what we will export here.
28  *
29  * 'model()' is a function which is important for mongoose behind the scenes to connect a
   schema with a name.
30  * so here you give that model a name
31  * and that name would be 'Product'
32  *
33  * 2nd argument is the schema
34  * so in my case 'productSchema' we define.
35  */
36 module.exports = mongoose.model('Product', productSchema)

```

```

1 // ./controllers/admin.js
2
3 /**we still import product form our ./models/product.js
4  * because i export a model.
5  * and we can use that in the way i used it here.
6  */
7 const Product = require('../models/product');
8
9 exports.getAddProduct = (req, res, next) => {
10   res.render('admin/edit-product', {
11     pageTitle: 'Add Product',
12     path: '/admin/add-product',
13     editing: false
14   });
15 };
16
17 exports.postAddProduct = (req, res, next) => {
18   const title = req.body.title;
19   const imageUrl = req.body.imageUrl;
20   const price = req.body.price;
21   const description = req.body.description;
22   const product = new Product({
23     /**order-matching with ./models/product.js don't matter */
24     title: title,
25     price: price,
26     description: description,
27     imageUrl: imageUrl
28   })
29   /**now 'product' here is managed by Mongoose
30   * and 'product' happens to have a 'save()' method provided by Mongoose
31   * 'save()' is not defined by us.
32   * we defined 'save()' before but now is not defined by us.
33   *

```

```

34  * we don't get a promise
35  * but mongoose still gives us a 'then()' method
36  *
37  * it also still gives us a 'catch()' method
38  * and therefore this code should continue to work.
39  */
40  product
41    .save()
42    .then(result => {
43      // console.log(result);
44      console.log('Created Product');
45      res.redirect('/admin/products');
46    })
47    .catch(err => {
48      console.log(err);
49    });
50 };
51
52 exports.getEditProduct = (req, res, next) => {
53   const editMode = req.query.edit;
54   if (!editMode) {
55     return res.redirect('/');
56   }
57   const prodId = req.params.productId;
58   Product.findById(prodId)
59     // Product.findById(prodId)
60     .then(product => {
61       if (!product) {
62         return res.redirect('/');
63       }
64       res.render('admin/edit-product', {
65         pageTitle: 'Edit Product',
66         path: '/admin/edit-product',
67         editing: editMode,
68         product: product
69       });
70     })
71     .catch(err => console.log(err));
72 };
73
74 exports.postEditProduct = (req, res, next) => {
75   const prodId = req.body.productId;
76   const updatedTitle = req.body.title;
77   const updatedPrice = req.body.price;
78   const updatedImageUrl = req.body.imageUrl;
79   const updatedDesc = req.body.description;
80
81   const product = new Product(
82     updatedTitle,
83     updatedPrice,
84     updatedDesc,
85     updatedImageUrl,
86     prodId
87   );
88   product
89     .save()

```

```

90     .then(result => {
91         console.log('UPDATED PRODUCT!');
92         res.redirect('/admin/products');
93     })
94     .catch(err => console.log(err));
95 };
96
97 exports.getProducts = (req, res, next) => {
98     Product.fetchAll()
99     .then(products => {
100         res.render('admin/products', {
101             prods: products,
102             pageTitle: 'Admin Products',
103             path: '/admin/products'
104         });
105     })
106     .catch(err => console.log(err));
107 };
108
109 exports.postDeleteProduct = (req, res, next) => {
110     const prodId = req.body.productId;
111     Product.deleteById(prodId)
112     .then(() => {
113         console.log('DESTROYED PRODUCT');
114         res.redirect('/admin/products');
115     })
116     .catch(err => console.log(err));
117 };

```

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7  const mongoose = require('mongoose')
8
9  const errorController = require('./controllers/error');
10 //const User = require('./models/user')
11
12 const app = express();
13
14 app.set('view engine', 'ejs');
15 app.set('views', 'views');
16
17 const adminRoutes = require('./routes/admin');
18 const shopRoutes = require('./routes/shop');
19
20 app.use(bodyParser.urlencoded({ extended: false }));
21 app.use(express.static(path.join(__dirname, 'public')));
22
23 /*
24 app.use((req, res, next) => {
25     User.findById('5cb7d12855f74b129c0b7c')
26     .then(user => {
27         req.user = new User(user.name, user.email, user.cart, user._id);
28         next();

```

```

29     })
30     .catch(err => console.log(err));
31 });
32 */
33
34 app.use('/admin', adminRoutes);
35 app.use(shopRoutes);
36
37 app.use(errorController.get404);
38
39 /**we already have everything in place we need to connect
40 * and mongoose will manage that one connection behind the scenes.
41 * so taht in other places where we start using mongoose from the mongoose package,
42 * we use that same connection we set up here.
43 */
44 mongoose
45   .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
retryWrites=true')
46   .then(result => {
47     app.listen(3000)
48   })
49   .catch(err => {
50     console.log(err)
51   })

```

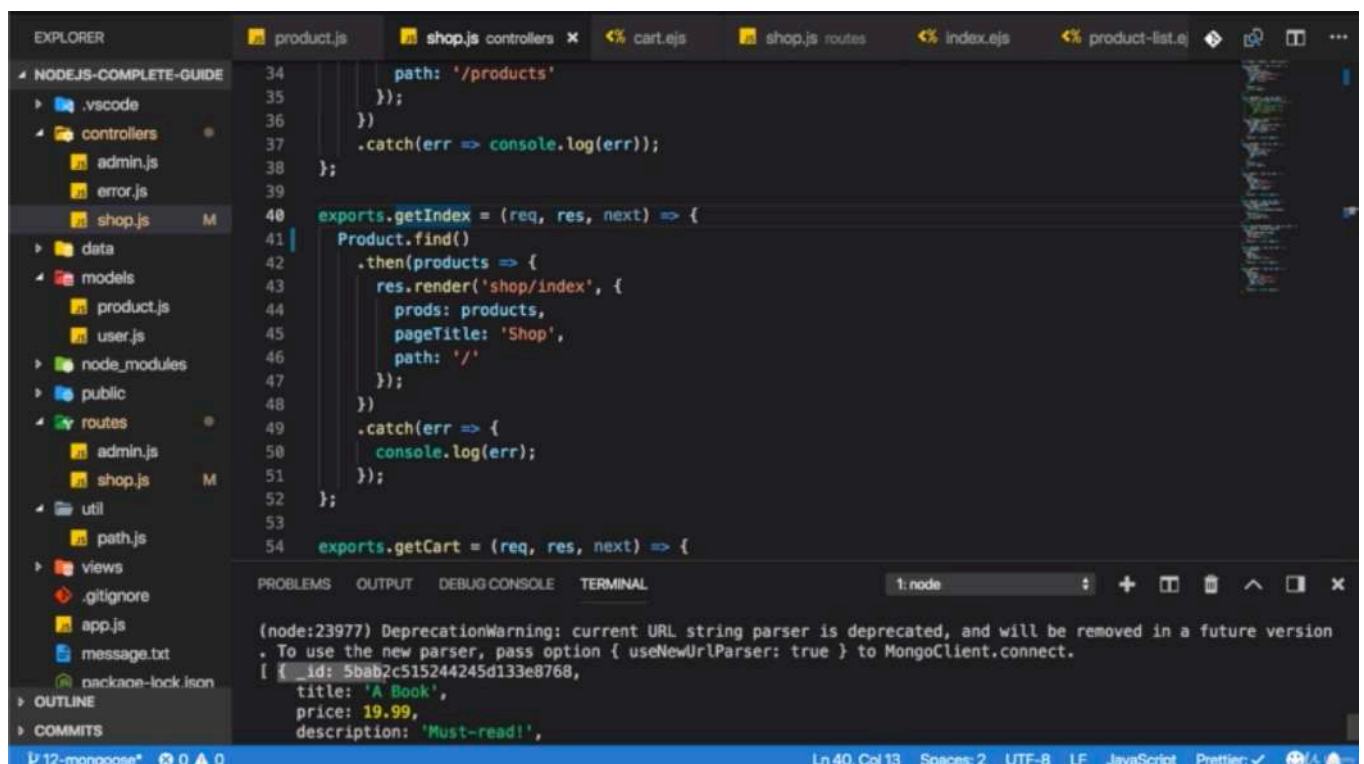
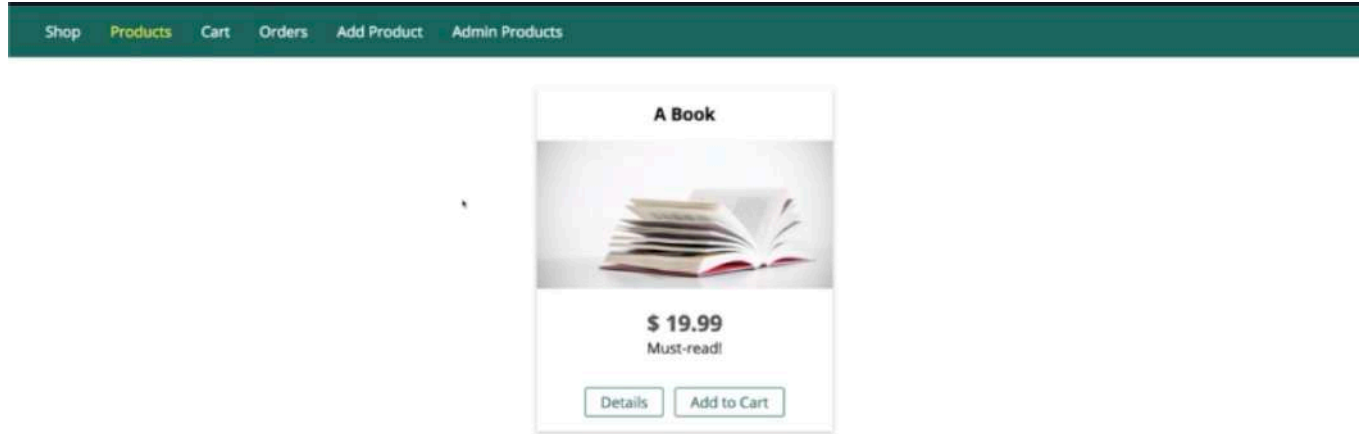
```

1 // ./routes/admin.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const adminController = require('../controllers/admin');
8
9 const router = express.Router();
10
11 // /admin/add-product => GET
12 router.get('/add-product', adminController.getAddProduct);
13
14 /*
15 // /admin/products => GET
16 router.get('/products', adminController.getProducts);
17 */
18
19 // /admin/add-product => POST
20 router.post('/add-product', adminController.postAddProduct);
21
22 /*
23 router.get('/edit-product/:productId', adminController.getEditProduct);
24
25 router.post('/edit-product', adminController.postEditProduct);
26
27 router.post('/delete-product', adminController.postDeleteProduct);
28 */
29
30 module.exports = router;

```

* Chapter 210: Fetching All Products

1. update
- ./controllers/shop.js
- ./routes/shop.js



- here what i can see is the output of the data that was fetched. and i get an array because 'find()' when used with mongoose automatically gives me that array here.

- 1 //./controllers/shop.js
- 2

```

3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   /**'find()' method works a bit differently when used with mongoose
7    * it doesn't give us a cursor.
8    * it does give us the products,
9    * we could add '.cursor'
10   * and call this to get access to the cursor
11   * and then use each async which would allow us to loop through them
12   * or 'next()' to get the next element.
13   * but i will just use 'find()'
14   * and this will essentially give me all my products automatically
15   */
16   Product.find()
17     .then(products => {
18       console.log(products)
19       res.render('shop/product-list', {
20         prods: products,
21         pageTitle: 'All Products',
22         path: '/products'
23       });
24     })
25     .catch(err => {
26       console.log(err);
27     });
28 };
29
30 exports.getProduct = (req, res, next) => {
31   const prodId = req.params.productId;
32   // Product.findAll({ where: { id: prodId } })
33   // .then(products => {
34   //   res.render('shop/product-detail', {
35   //     product: products[0],
36   //     pageTitle: products[0].title,
37   //     path: '/products'
38   //   });
39   // })
40   // .catch(err => console.log(err));
41   Product.findById(prodId)
42     .then(product => {
43       res.render('shop/product-detail', {
44         product: product,
45         pageTitle: product.title,
46         path: '/products'
47       });
48     })
49     .catch(err => console.log(err));
50 };
51
52 exports.getIndex = (req, res, next) => {
53   Product.find()
54     .then(products => {
55       res.render('shop/index', {
56         prods: products,
57         pageTitle: 'Shop',
58         path: '/'

```



```

59     });
60   })
61   .catch(err => {
62     console.log(err);
63   });
64 };
65
66 exports.getCart = (req, res, next) => {
67   req.user
68     .getCart()
69     .then(products => {
70       res.render('shop/cart', {
71         path: '/cart',
72         pageTitle: 'Your Cart',
73         products: products
74       });
75     })
76     .catch(err => console.log(err));
77 };
78
79 exports.postCart = (req, res, next) => {
80   const prodId = req.body.productId;
81   Product.findById(prodId)
82     .then(product => {
83       return req.user.addToCart(product);
84     })
85     .then(result => {
86       console.log(result);
87       res.redirect('/cart');
88     });
89 };
90
91 exports.postCartDeleteProduct = (req, res, next) => {
92   const prodId = req.body.productId;
93   req.user
94     .deleteItemFromCart(prodId)
95     .then(result => {
96       res.redirect('/cart');
97     })
98     .catch(err => console.log(err));
99 };
100
101 exports.postOrder = (req, res, next) => {
102   let fetchedCart;
103   req.user
104     .addOrder()
105     .then(result => {
106       res.redirect('/orders');
107     })
108     .catch(err => console.log(err));
109 };
110
111 exports.getOrders = (req, res, next) => {
112   req.user
113     .getOrders()
114     .then(orders => {

```

```

115     res.render('shop/orders', {
116         path: '/orders',
117         pageTitle: 'Your Orders',
118         orders: orders
119     });
120 }
121 .catch(err => console.log(err));
122 };
123

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 /*
16 router.get('/products/:productId', shopController.getProduct);
17
18 router.get('/cart', shopController.getCart);
19
20 router.post('/cart', shopController.postCart);
21
22 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
23
24 router.post('/create-order', shopController.postOrder);
25
26 router.get('/orders', shopController.getOrders);
27 */
28
29 module.exports = router;
30

```

* Chapter 211: Fetching A Single Product

1. update
- ./controllers/shop.js
- ./routes/shop.js



A Book



19.99

Must-read!

Add to Cart

```
1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
```

```

15 router.get('/products/:productId', shopController.getProduct);
16
17 /*
18 router.get('/cart', shopController.getCart);
19
20 router.post('/cart', shopController.postCart);
21
22 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
23
24 router.post('/create-order', shopController.postOrder);
25
26 router.get('/orders', shopController.getOrders);
27 */
28
29 module.exports = router;
30

```

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.find()
7     .then(products => {
8       console.log(products)
9       res.render('shop/product-list', {
10         prods: products,
11         pageTitle: 'All Products',
12         path: '/products'
13       });
14     })
15     .catch(err => {
16       console.log(err);
17     });
18 };
19
20 exports.getProduct = (req, res, next) => {
21   const prodId = req.params.productId;
22   /**mongoose has a 'findById()' method
23    * so little convenience method that defines for us.
24    * so again 'findById()' method is not our own method,
25    * it's defined by Mongoose
26    *
27    * and even pass a string to 'findById()'
28    * and mongoose will automatically convert this to an objectId
29    * so it will handle that for us as well.
30    */
31   Product.findById(prodId)
32     .then(product => {
33       res.render('shop/product-detail', {
34         product: product,
35         pageTitle: product.title,
36         path: '/products'
37       });
38     })
39     .catch(err => console.log(err));
40 };

```

```
41
42 exports.getIndex = (req, res, next) => {
43   Product.find()
44     .then(products => {
45       res.render('shop/index', {
46         prods: products,
47         pageTitle: 'Shop',
48         path: '/'
49       });
50     })
51     .catch(err => {
52       console.log(err);
53     });
54 };
55
56 exports.getCart = (req, res, next) => {
57   req.user
58     .getCart()
59     .then(products => {
60       res.render('shop/cart', {
61         path: '/cart',
62         pageTitle: 'Your Cart',
63         products: products
64       });
65     })
66     .catch(err => console.log(err));
67 };
68
69 exports.postCart = (req, res, next) => {
70   const prodId = req.body.productId;
71   Product.findById(prodId)
72     .then(product => {
73       return req.user.addToCart(product);
74     })
75     .then(result => {
76       console.log(result);
77       res.redirect('/cart');
78     });
79 };
80
81 exports.postCartDeleteProduct = (req, res, next) => {
82   const prodId = req.body.productId;
83   req.user
84     .deleteItemFromCart(prodId)
85     .then(result => {
86       res.redirect('/cart');
87     })
88     .catch(err => console.log(err));
89 };
90
91 exports.postOrder = (req, res, next) => {
92   let fetchedCart;
93   req.user
94     .addOrder()
95     .then(result => {
96       res.redirect('/orders');
```

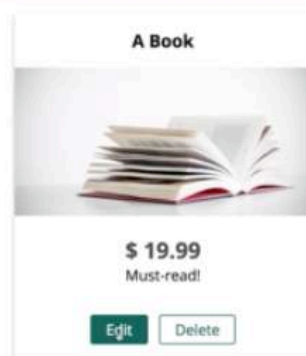
```

97     })
98     .catch(err => console.log(err));
99   });
100
101   exports.getOrders = (req, res, next) => {
102     req.user
103       .getOrders()
104       .then(orders => {
105         res.render('shop/orders', {
106           path: '/orders',
107           pageTitle: 'Your Orders',
108           orders: orders
109         });
110       })
111       .catch(err => console.log(err));
112   });
113

```

* Chapter 212: Updating Products

1. update
 - ./controllers/admin.js
 - ./routes/admin.js



Title

A Book

Image URL

<http://ichef.bbci.co.uk/ww/features/wm/live>

Price

19,99

Description

Must-read!

I

Update Product

Title

A Book!

Image URL

<http://ichef.bbci.co.uk/ww/features/wm/live>

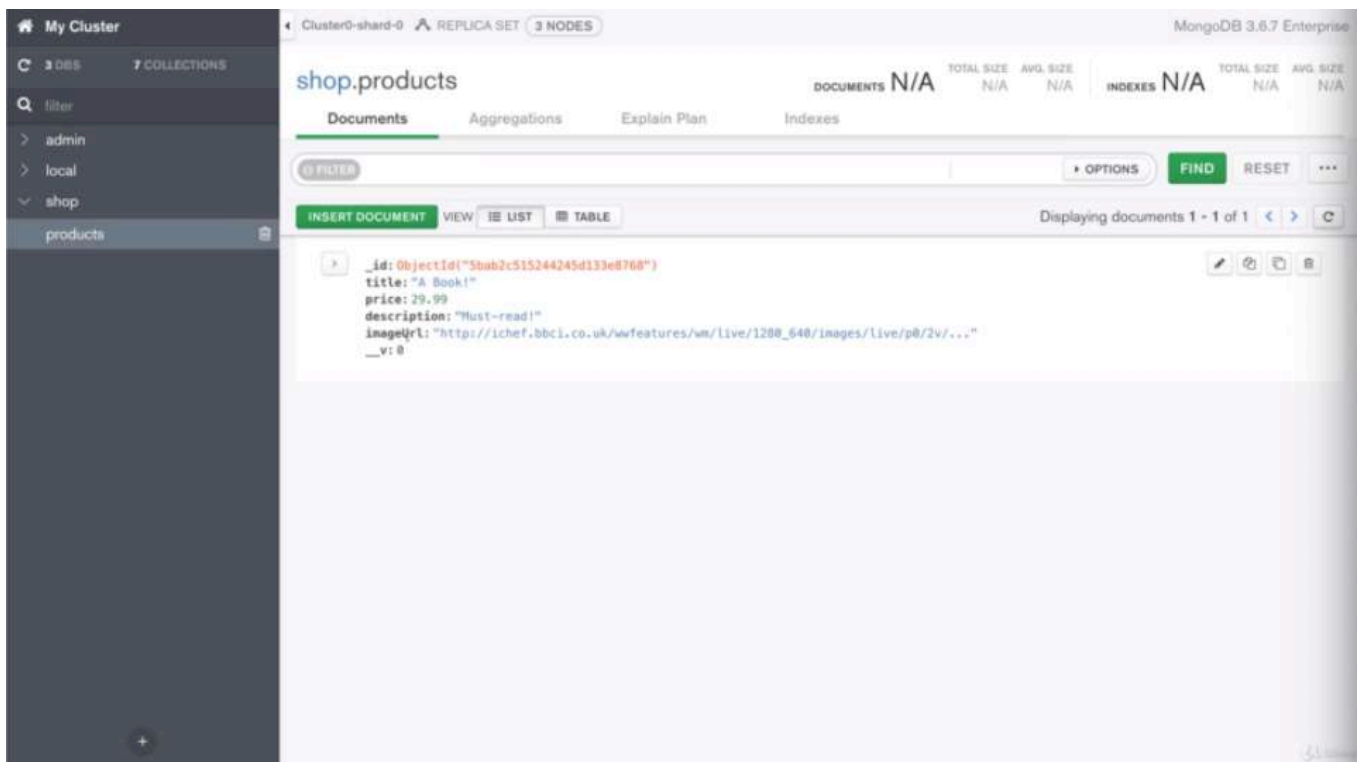
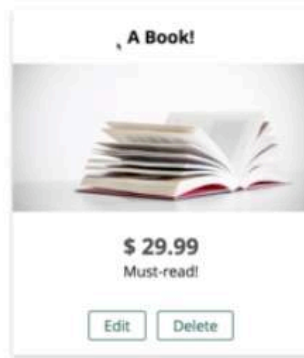
Price

29,99

Description

Must-read!

Update Product



```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
```



```

15  const imageUrl = req.body.imageUrl;
16  const price = req.body.price;
17  const description = req.body.description;
18  const product = new Product({
19    title: title,
20    price: price,
21    description: description,
22    imageUrl: imageUrl
23  })
24  product
25    .save()
26    .then(result => {
27      // console.log(result);
28      console.log('Created Product');
29      res.redirect('/admin/products');
30    })
31    .catch(err => {
32      console.log(err);
33    });
34  };
35
36  exports.getEditProduct = (req, res, next) => {
37    const editMode = req.query.edit;
38    if (!editMode) {
39      return res.redirect('/');
40    }
41    const prodId = req.params.productId;
42    Product.findById(prodId)
43      .then(product => {
44        if (!product) {
45          return res.redirect('/');
46        }
47        res.render('admin/edit-product', {
48          pageTitle: 'Edit Product',
49          path: '/admin/edit-product',
50          editing: editMode,
51          product: product
52        });
53      })
54      .catch(err => console.log(err));
55  };
56
57  exports.postEditProduct = (req, res, next) => {
58    const prodId = req.body.productId;
59    const updatedTitle = req.body.title;
60    const updatedPrice = req.body.price;
61    const updatedImageUrl = req.body.imageUrl;
62    const updatedDesc = req.body.description;
63    /**i first of all find the product
64     * and i get back a full mongoose object
65     * hence i can manipulate it and call save again
66     * i return the result of that
67     * and then call 'then()' on that to redirect once the saving was done.
68     */
69    Product
70      .findById(prodId)

```

```

71     .then(product => {
72         product.title = updatedTitle
73         product.price = updatedPrice
74         product.description = updatedDesc
75         product.imageUrl = updatedImageUrl
76         /**i can move 'product.save()' into my function
77          * and call 'save()' on the product that was fetched from the database
78          * because thanks to mongoose,
79          * this will now not be a javascript object with the data
80          * but we will have a full mongoose object here with all the mongoose methods like
81          'save()'
82          * if we call 'save()' on an existing object,
83          * it will not be saved as a new one,
84          * but the changes will be saved.
85          * so it will automatically do an update behind the scenes.
86          */
87         return product
88         .save()
89     })
90     .then(result => {
91         console.log('UPDATED PRODUCT!');
92         res.redirect('/admin/products');
93     })
94     .catch(err => console.log(err));
95 };
96 exports.getProducts = (req, res, next) => {
97     Product.find()
98     .then(products => {
99         res.render('admin/products', {
100             prods: products,
101             pageTitle: 'Admin Products',
102             path: '/admin/products'
103         });
104     })
105     .catch(err => console.log(err));
106 };
107
108 exports.postDeleteProduct = (req, res, next) => {
109     const prodId = req.body.productId;
110     Product.deleteById(prodId)
111     .then(() => {
112         console.log('DESTROYED PRODUCT');
113         res.redirect('/admin/products');
114     })
115     .catch(err => console.log(err));
116 };

```

```

1 // ./routes/admin.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const adminController = require('../controllers/admin');
8
9 const router = express.Router();

```

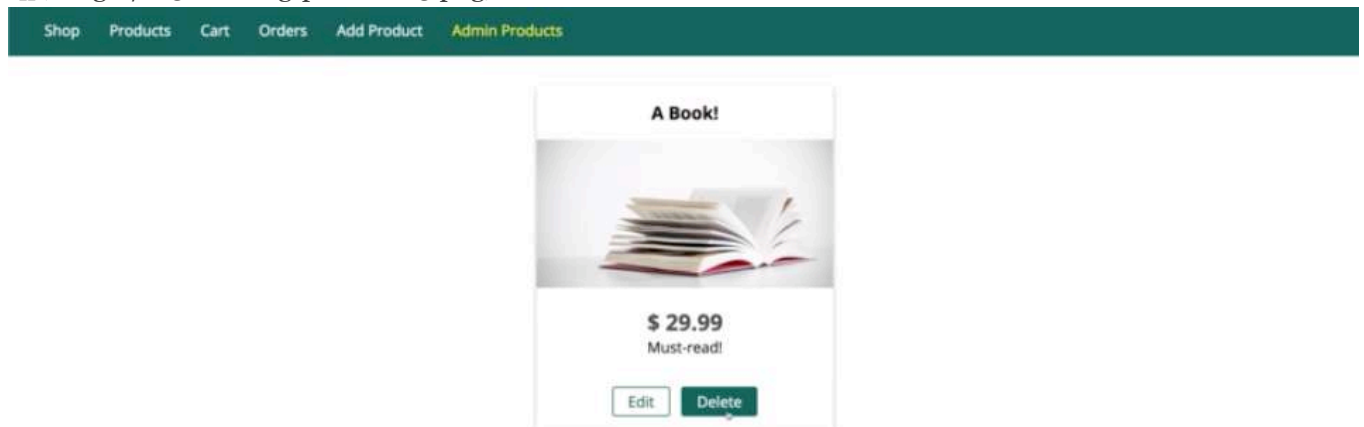
```

10
11 // /admin/add-product => GET
12 router.get('/add-product', adminController.getAddProduct);
13
14
15 // /admin/products => GET
16 router.get('/products', adminController.getProducts);
17
18
19 // /admin/add-product => POST
20 router.post('/add-product', adminController.postAddProduct);
21
22
23 router.get('/edit-product/:productId', adminController.getEditProduct);
24
25
26 router.post('/edit-product', adminController.postEditProduct);
27
28 /*
29 router.post('/delete-product', adminController.postDeleteProduct);
30 */
31
32 module.exports = router;

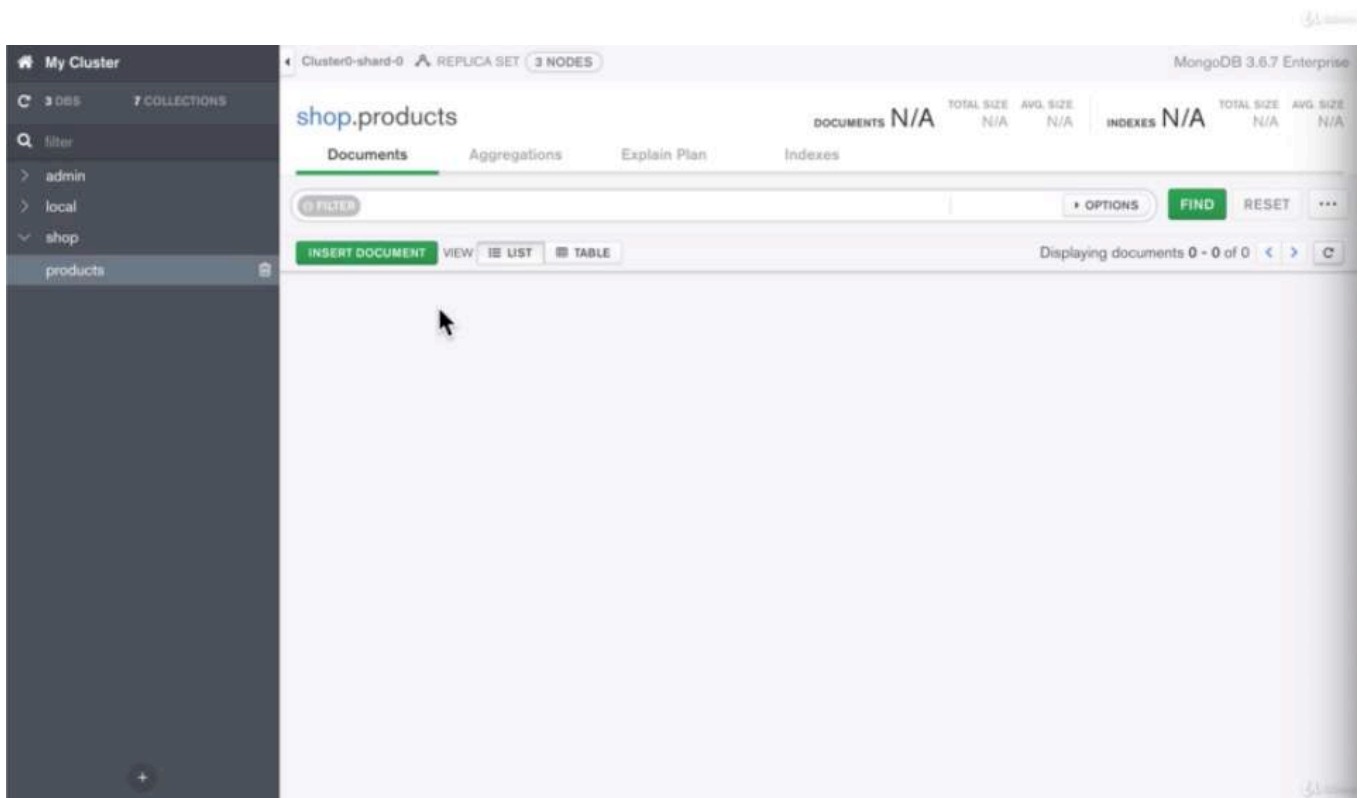
```

* Chapter 213: Deleting Products

1. update
 - ./controllers/admin.js
 - ./routes/admin.js



No Products Found!



```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
```

```

15  const imageUrl = req.body.imageUrl;
16  const price = req.body.price;
17  const description = req.body.description;
18  const product = new Product({
19    title: title,
20    price: price,
21    description: description,
22    imageUrl: imageUrl
23  })
24  product
25    .save()
26    .then(result => {
27      // console.log(result);
28      console.log('Created Product');
29      res.redirect('/admin/products');
30    })
31    .catch(err => {
32      console.log(err);
33    });
34  };
35
36  exports.getEditProduct = (req, res, next) => {
37    const editMode = req.query.edit;
38    if (!editMode) {
39      return res.redirect('/');
40    }
41    const prodId = req.params.productId;
42    Product.findById(prodId)
43      .then(product => {
44        if (!product) {
45          return res.redirect('/');
46        }
47        res.render('admin/edit-product', {
48          pageTitle: 'Edit Product',
49          path: '/admin/edit-product',
50          editing: editMode,
51          product: product
52        });
53      })
54      .catch(err => console.log(err));
55  };
56
57  exports.postEditProduct = (req, res, next) => {
58    const prodId = req.body.productId;
59    const updatedTitle = req.body.title;
60    const updatedPrice = req.body.price;
61    const updatedImageUrl = req.body.imageUrl;
62    const updatedDesc = req.body.description;
63
64    Product
65      .findById(prodId)
66      .then(product => {
67        product.title = updatedTitle
68        product.price = updatedPrice
69        product.description = updatedDesc
70        product.imageUrl = updatedImageUrl

```

```

71     return product
72     .save()
73   })
74   .then(result => {
75     console.log('UPDATED PRODUCT!');
76     res.redirect('/admin/products');
77   })
78   .catch(err => console.log(err));
79 };
80
81 exports.getProducts = (req, res, next) => {
82   Product.find()
83     .then(products => {
84       res.render('admin/products', {
85         prods: products,
86         pageTitle: 'Admin Products',
87         path: '/admin/products'
88       });
89     })
90     .catch(err => console.log(err));
91 };
92
93 exports.postDeleteProduct = (req, res, next) => {
94   const prodId = req.body.productId;
95   /**'findByIdAndRemove()' is a built-in method provided by mongoose that should remove a
document
96   */
97   Product.findByIdAndRemove(prodId)
98     .then(() => {
99       console.log('DESTROYED PRODUCT');
100       res.redirect('/admin/products');
101     })
102     .catch(err => console.log(err));
103 };

```

```

1 // ./routes/admin.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const adminController = require('../controllers/admin');
8
9 const router = express.Router();
10
11 // /admin/add-product => GET
12 router.get('/add-product', adminController.getAddProduct);
13
14 // /admin/products => GET
15 router.get('/products', adminController.getProducts);
16
17 // /admin/add-product => POST
18 router.post('/add-product', adminController.postAddProduct);
19
20 router.get('/edit-product/:productId', adminController.getEditProduct);
21
22 router.post('/edit-product', adminController.postEditProduct);

```

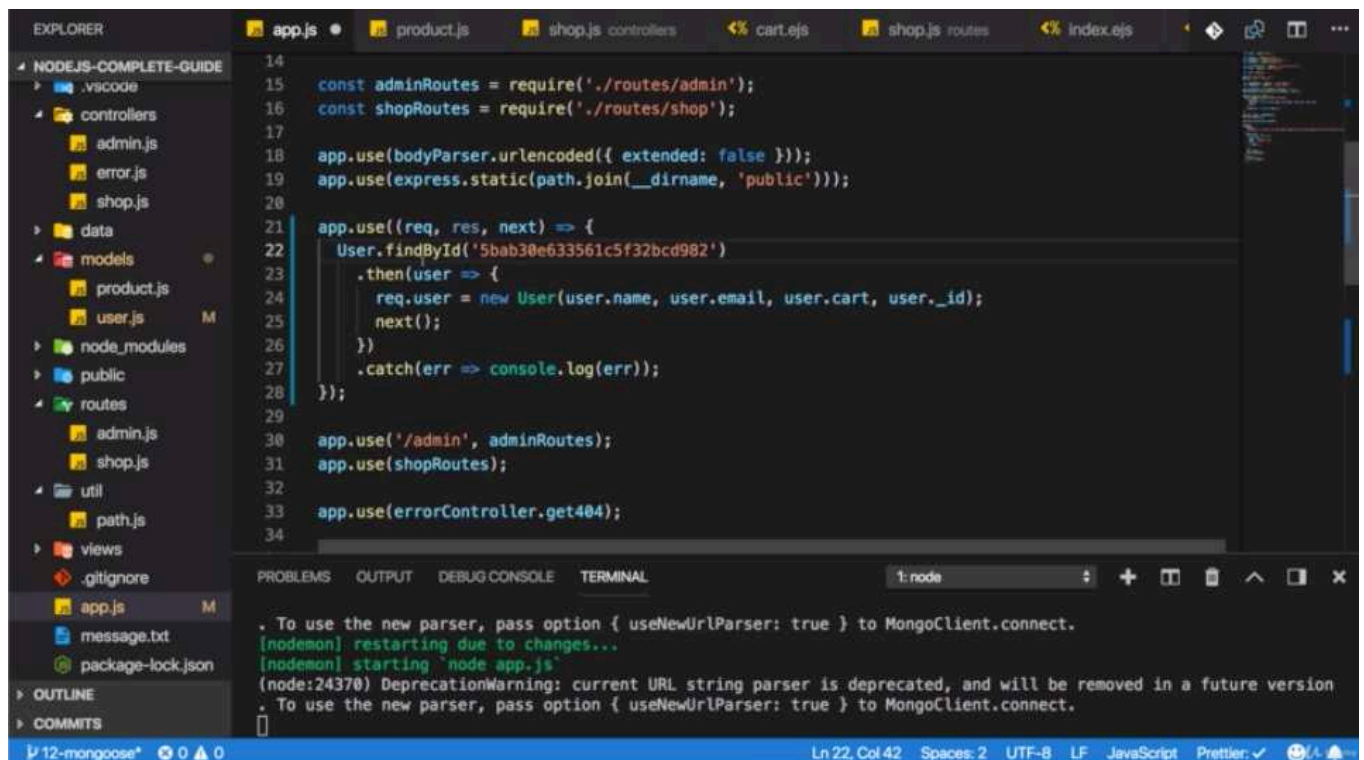
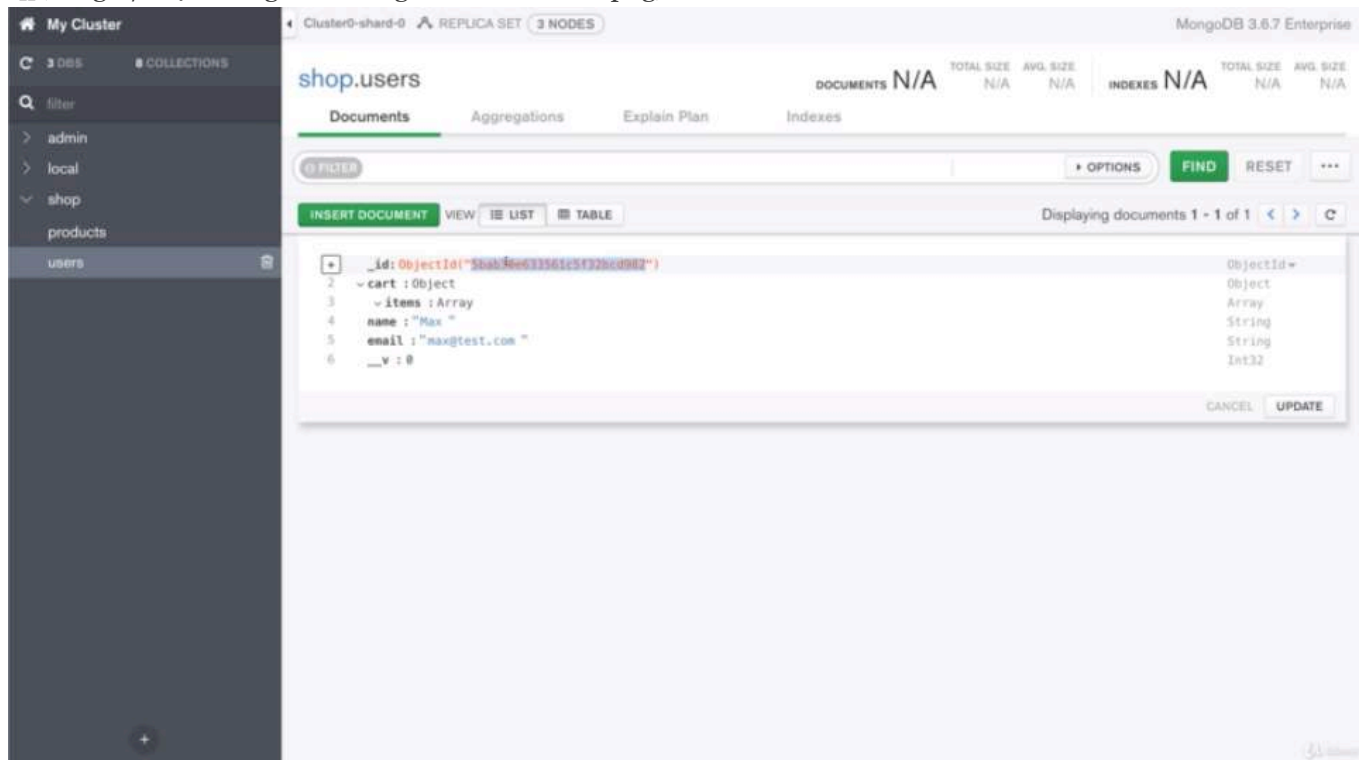
```

23
24 router.post('/delete-product', adminController.postDeleteProduct);
25
26 module.exports = router;

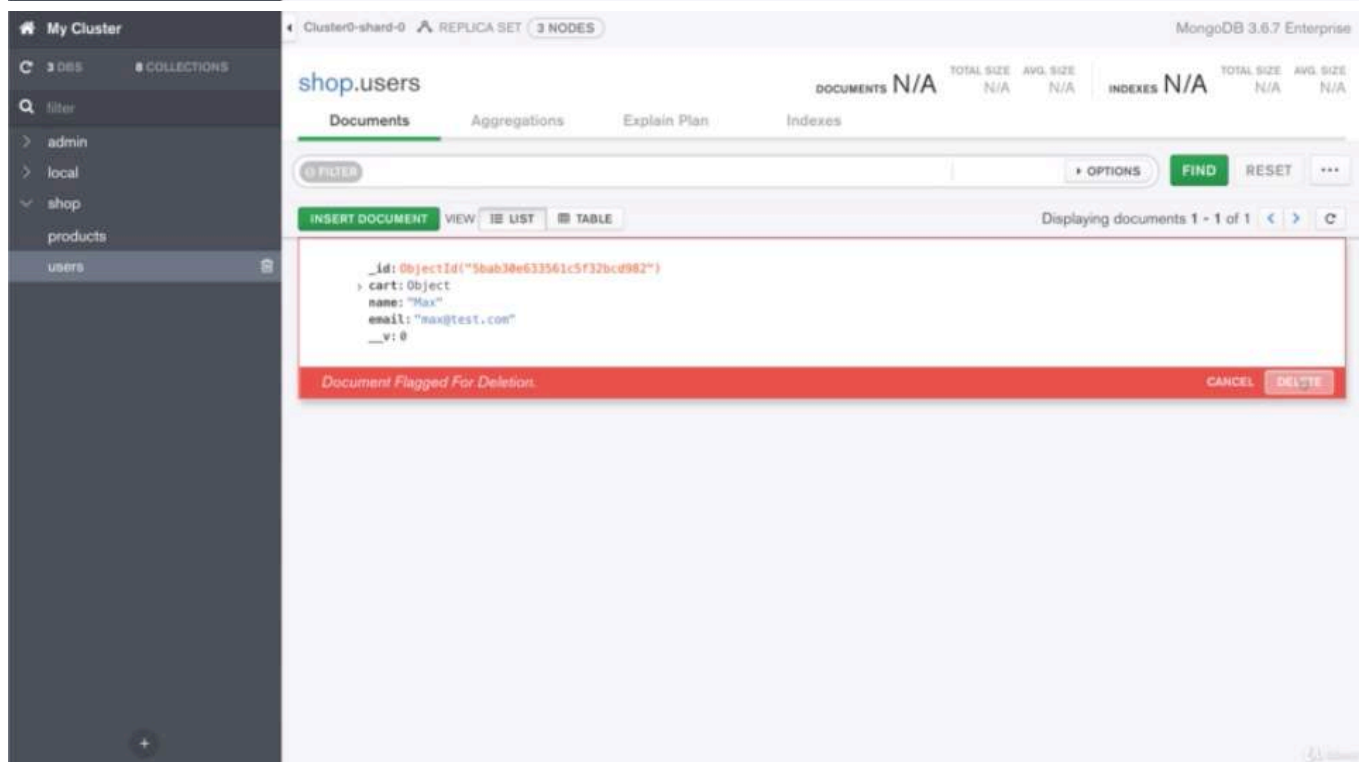
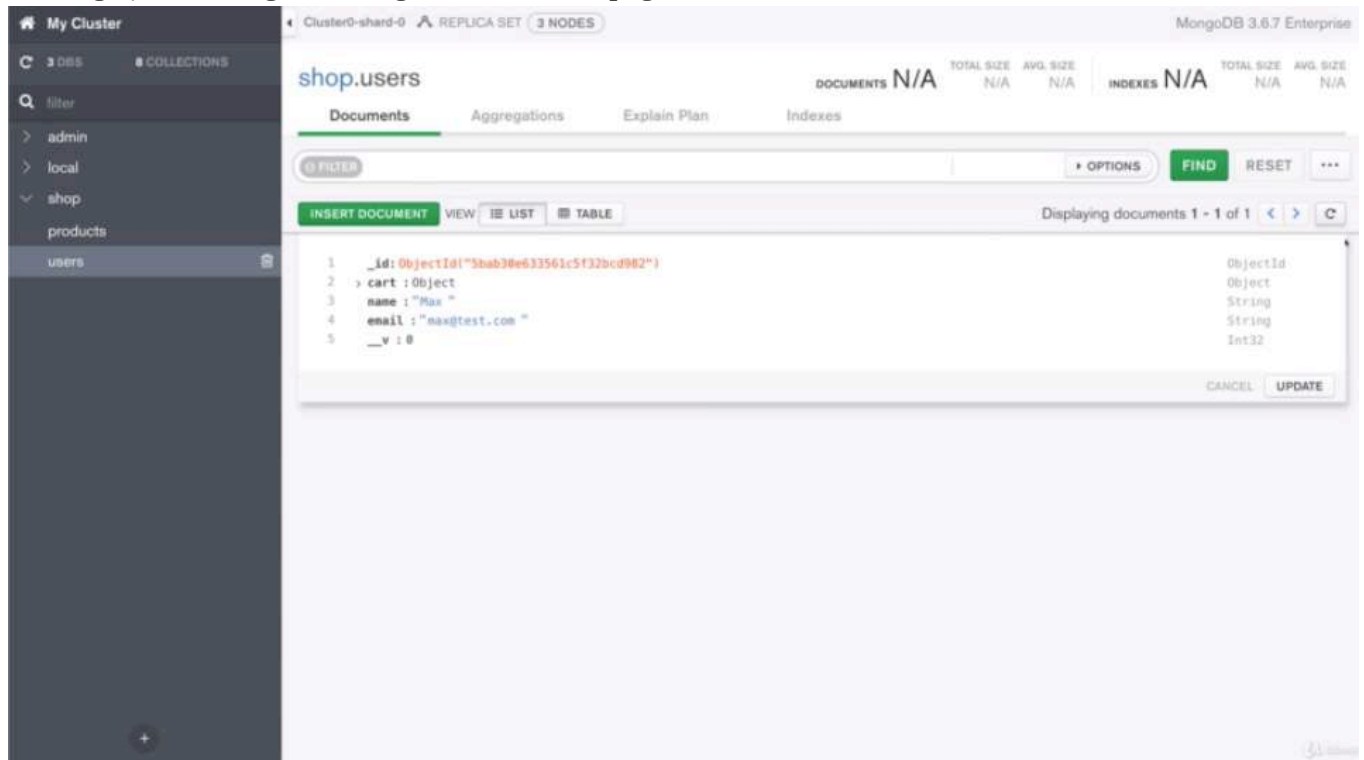
```

* Chapter 214: Adding And Using A User Model

1. update
- ./models/user.js
- app.js



- with that `_id`, let me copy that ID and go back to the `app.js` file and comment this middleware back in here.
- i just need to paste in that ID of that user we just created. '`findById()`' is a method provided by mongoose so this will work.



EXPLORER

- NODEJS-COMPLETE-GUIDE
 - .vscode
 - controllers
 - admin.js
 - error.js
 - shop.js
 - data
 - models
 - product.js
 - user.js
 - node_modules
 - public
 - routes
 - admin.js
 - shop.js
 - util
 - path.js
 - views
 - .gitignore
 - app.js
 - message.txt
 - package-lock.json
- OUTLINE
- COMMITTS

```
11
12 app.set('view engine', 'ejs');
13 app.set('views', 'views');
14
15 const adminRoutes = require('./routes/admin');
16 const shopRoutes = require('./routes/shop');
17
18 app.use(bodyParser.urlencoded({ extended: false }));
19 app.use(express.static(path.join(__dirname, 'public')));
20
21 app.use((req, res, next) => {
22   User.findById(req.user.id)
23     .then(user => {
24       req.user = user;
25       next();
26     })
27     .catch(err => console.log(err));
28 });
29
30 app.use('/admin', adminRoutes);
31 app.use(shopRoutes);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:24425) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

Ln 22, Col 18 Spaces: 2 UTF-8 LF JavaScript Prettier: ✓

My Cluster Cluster0-shard-0 REPLICASET 3 NODES MongoDB 3.6.7 Enterprise

shop.users DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

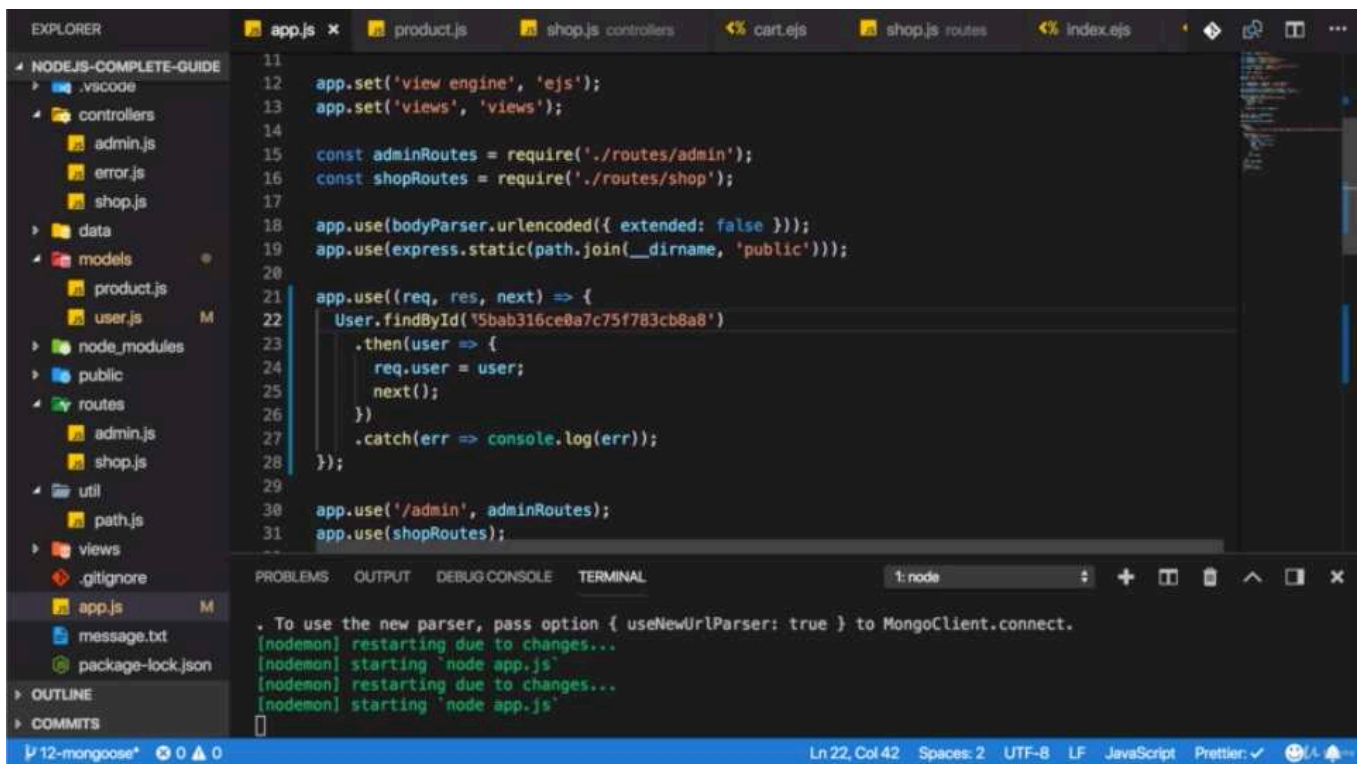
0 FILTER OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

1	_id: ObjectId("5bab3167e07c75f783c08a")	ObjectId
2	cart: Object	Object
3	name: "Max "	String
4	email: "max@test.com "	String
5	_v: 0	Int32

CANCEL UPDATE



```
1 //./models/user.js
2
3 const mongoose = require('mongoose')
4
5 const Schema = mongoose.Schema
6
7 const userSchema = new Schema({
8   name: {
9     type: String,
10    required: true
11  },
12  email: {
13    type: String,
14    required: true
15  },
16  cart: {
17    /**i wanna have an array of documents where i have a productId
18     * which i will configure with this document.
19     *
20     * we have a 'Types' field
21     * and there we got all these special types like ObjectId
22     * so i'm telling mongoose that
23     * this will actually store an objectId
24     * because it will store a reference to a product
25     */
26    items: [{
27      productId: {
28        type: Schema.Types.ObjectId,
29        required: true
30      },
31      quantity: {
32        type: Number,
33        required: true
34      }
35    }
36  ]
37 }
```

```

36   }
37 })
38
39 /**i will export this by calling mongoose model,
40 * give this a name and the name will be 'User'
41 * hence this will be stored in a 'Users'
42 * because mongoose will automatically take the plural lowercase version of that as a
  collection name
43 */
44 module.exports = mongoose.model('User', userSchema)

```

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose')
8
9 const errorController = require('./controllers/error');
10 const User = require('./models/user')
11
12 const app = express();
13
14 app.set('view engine', 'ejs');
15 app.set('views', 'views');
16
17 const adminRoutes = require('./routes/admin');
18 const shopRoutes = require('./routes/shop');
19
20 app.use(bodyParser.urlencoded({ extended: false }));
21 app.use(express.static(path.join(__dirname, 'public')));
22
23
24 app.use((req, res, next) => {
25   User.findById('5cbb2b2c80bd7193adb9eeeb')
26     .then(user => {
27       /**i can store that user in my request
28       * and keep in mind, 'user' on right side is a full mongoose model
29       * so we can call all these mongoose model functions or methods on that user object
30       * and therefore also on the user object which i store here.
31       */
32       req.user = user
33       next();
34     })
35     .catch(err => console.log(err));
36 });
37
38 app.use('/admin', adminRoutes);
39 app.use(shopRoutes);
40
41 app.use(errorController.get404);
42
43 mongoose
44   .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
  retryWrites=true')
45   .then(result => {

```

```

46  /**if i give 'findOne()' no arguments,
47  * it will always give me back the first user it finds
48  */
49  User
50  .findOne()
51  .then(user => {
52    if(!user){
53      const user = new User({
54        name: 'Max',
55        email: 'max@test.com',
56        cart: {
57          items: []
58        }
59      })
60      user.save()
61    }
62  })
63  app.listen(3000)
64  })
65  .catch(err => {
66    console.log(err)
67  })

```

* Chapter 215: Using Relations In Mongoose

1. update

- ./models/product.js
- ./models/user.js
- ./controllers/admin.js

Shop
Products
Cart
Orders
Add Product
Admin Products

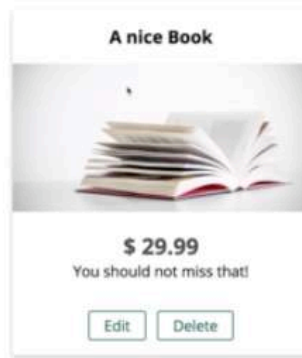
Title

Image URL

Price

Description

Add Product



My Cluster Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.7 Enterprise

shop.products DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

0 FILTER OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5bab325948c1835fd8d563c6"),
  "title": "A nice Book",
  "price": 29.99,
  "description": "You should not miss that!",
  "imageUrl": "http://ichef.bbci.co.uk/ww/features/wm/live/1288_648/images/Live/p0/2v/...",
  "userId": ObjectId("5bab316ce8a7c75f783cb0a8"),
  "__v": 0
}
```

```
1 //./models/product.js
2
3 const mongoose = require('mongoose')
4
5 const Schema = mongoose.Schema;
6
7 const productSchema = new Schema({
8   title: {
9     type: String,
10    required: true
11  },
12  price: {
13    type: Number,
14    required: true
15  }
16 })
```

```

15 },
16 description: {
17   type: String,
18   required: true
19 },
20 imageUrl: {
21   type: String,
22   required: true
23 },
24 userId: {
25   /**this will be a reference to a user,
26    * so this will be of type 'Schema.Types.ObjectId'
27    *
28    * and we can add a special 'ref' configuration
29    * and 'ref' takes a string where we tell mongoose
30    * hey which other mongoose model is related to the data in that field.
31    * we know that we will store a userId
32    * but because the type is objectId, this is not obvious
33    * this could be any objectId of any object
34    * so i will add 'User'
35    * and you use the name of your model which you wanna relate this,
36    * so since our model here is named user,
37    * i will name it user here,
38    * so i refer to my User model here
39    *
40    * with that, i got a relation set up.
41    */
42   type: Schema.Types.ObjectId,
43   ref: 'User',
44   required: true
45 }
46 })
47
48 module.exports = mongoose.model('Product', productSchema)
49

```

```

1  //./models/user.js
2
3  const mongoose = require('mongoose')
4
5  const Schema = mongoose.Schema
6
7  const userSchema = new Schema({
8    name: {
9      type: String,
10     required: true
11   },
12   email: {
13     type: String,
14     required: true
15   },
16   /**this also means that in my User model where i store productId,
17    * i can also add a reference here and refer to product
18    * because i know that for every user in the cart items,
19    * i will store products where i refer to some ID
20    * and that ID happens to refer to a product stored or defined through the Product model
21    * now we got relation set up with ref.

```

```

22  *
23  * you only need this when using references,
24  * when using embedded documents as we do with the cart,
25  * you don't need to do anything
26  * because you use an embedded document,
27  * this already has kind of an implicit relation that is managed inside of one document
28  *
29  */
30  cart: {
31    items: [{
32      productId: {
33        type: Schema.Types.ObjectId,
34        ref: 'Product',
35        required: true
36      },
37      quantity: {
38        type: Number,
39        required: true
40      }
41    }]
42  }
43 })
44
45 module.exports = mongoose.model('User', userSchema)
46

```

```

1  // ./controllers/admin.js
2
3  const Product = require('../models/product');
4
5  exports.getAddProduct = (req, res, next) => {
6    res.render('admin/edit-product', {
7      pageTitle: 'Add Product',
8      path: '/admin/add-product',
9      editing: false
10   });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
15   const imageUrl = req.body.imageUrl;
16   const price = req.body.price;
17   const description = req.body.description;
18   const product = new Product({
19     title: title,
20     price: price,
21     description: description,
22     imageUrl: imageUrl,
23     /**principle is 'req.user._id'
24     * conveniently in mongoose,
25     * you can store the entire user object like 'req.user' not 'req.user._id'
26     * 'req.user' is the entire user object not just the ID
27     * and mongoose will just pick the ID from that object
28     */
29     userId: req.user
30   })
31   product

```

```

32     .save()
33     .then(result => {
34         // console.log(result);
35         console.log('Created Product');
36         res.redirect('/admin/products');
37     })
38     .catch(err => {
39         console.log(err);
40     });
41 };
42
43 exports.getEditProduct = (req, res, next) => {
44     const editMode = req.query.edit;
45     if (!editMode) {
46         return res.redirect('/');
47     }
48     const prodId = req.params.productId;
49     Product.findById(prodId)
50     .then(product => {
51         if (!product) {
52             return res.redirect('/');
53         }
54         res.render('admin/edit-product', {
55             pageTitle: 'Edit Product',
56             path: '/admin/edit-product',
57             editing: editMode,
58             product: product
59         });
60     })
61     .catch(err => console.log(err));
62 };
63
64 exports.postEditProduct = (req, res, next) => {
65     const prodId = req.body.productId;
66     const updatedTitle = req.body.title;
67     const updatedPrice = req.body.price;
68     const updatedImageUrl = req.body.imageUrl;
69     const updatedDesc = req.body.description;
70
71     Product
72     .findById(prodId)
73     .then(product => {
74         product.title = updatedTitle
75         product.price = updatedPrice
76         product.description = updatedDesc
77         product.imageUrl = updatedImageUrl
78         return product
79     })
80     .save()
81     .then(result => {
82         console.log('UPDATED PRODUCT!');
83         res.redirect('/admin/products');
84     })
85     .catch(err => console.log(err));
86 };
87

```



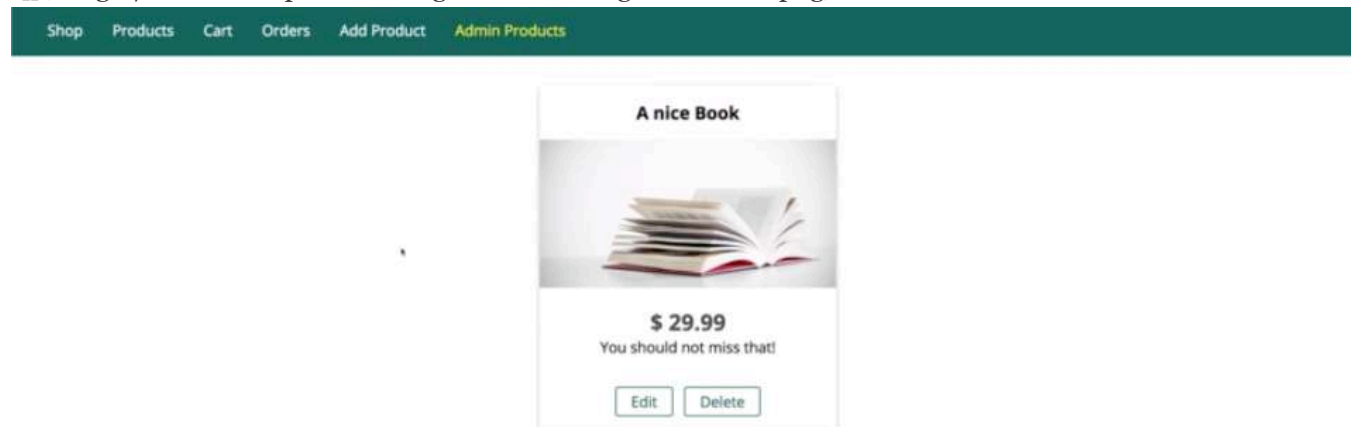
```

88 exports.getProducts = (req, res, next) => {
89   Product.find()
90     .then(products => {
91       res.render('admin/products', {
92         prods: products,
93         pageTitle: 'Admin Products',
94         path: '/admin/products'
95       });
96     })
97     .catch(err => console.log(err));
98 };
99
100 exports.postDeleteProduct = (req, res, next) => {
101   const prodId = req.body.productId;
102   Product.findByIdAndRemove(prodId)
103     .then(() => {
104       console.log('DESTROYED PRODUCT');
105       res.redirect('/admin/products');
106     })
107     .catch(err => console.log(err));
108 };

```

* Chapter 216: One Important Thing About Fetching Relations

1. update
- ./controllers/admin.js



```
72 console.log('rendered products: ');
73 res.redirect('/admin/products');
74 })
75 .catch(err => console.log(err));
76 };
77
78 exports.getProducts = (req, res, next) => {
79   Product.find()
80     .populate('userId')
81     .then(products => {
82       console.log(products);
83       res.render('admin/products', {
84         prods: products,
85         pageTitle: 'Admin Products',
86         path: '/admin/products'
87       });
88     })
89     .catch(err => console.log(err));
90 };
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

Cmd + click to follow link: '1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',

```
userId:
{ cart: {Object},
  _id: 5bab316ce0a7c75f783cb8a8,
  name: 'Max',
  email: 'max@test.com',
  __v: 0 },
  __v: 0 } }
```

- if i reload this page, you will see the userId, but the full user object and that can be really helpful for fetching data because this gives you all the data in one step, instead of writing nested queries on your own.



```
72 console.log('Created Product: ');
73 res.redirect('/admin/products');
74 })
75 .catch(err => console.log(err));
76 };
77
78 exports.getProducts = (req, res, next) => {
79   Product.find()
80     .select('title price -_id')
81     .populate('userId', 'name')
82     .then(products => {
83       console.log(products);
84       res.render('admin/products', {
85         prods: products,
86         pageTitle: 'Admin Products',
87         path: '/admin/products'
88       });
89     })
90     .catch(err => console.log(err));
91 }
```

```
node
[monodemon] restarting due to changes...
[monodemon] starting 'node app.js'
(node:24690) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[ { title: 'A nice Book',
  price: 29.99,
  _id: '5bab316ce0a7c75f783cb8a8',
  name: 'Max' } ]
```

- if you reload this page, you already see some data as missing because we didn't retrieve it.
- and you see it in the data that gets logged here too. we only retrieve the title and the price, we explicitly excluded the Id. for the userId, we didn't explicitly exclude the userId. so we got that.

```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   res.render('admin/edit-product', {
7     pageTitle: 'Add Product',
8     path: '/admin/add-product',
9     editing: false
10  });
11 };
12
13 exports.postAddProduct = (req, res, next) => {
14   const title = req.body.title;
15   const imageUrl = req.body.imageUrl;
16   const price = req.body.price;
17   const description = req.body.description;
18   const product = new Product({
19     title: title,
20     price: price,
21     description: description,
22     imageUrl: imageUrl,
23     userId: req.user
24   })
25   product
26     .save()
27     .then(result => {
28       // console.log(result);
29       console.log('Created Product');
30       res.redirect('/admin/products');
31     })
32 }
```

```

32     .catch(err => {
33         console.log(err);
34     });
35 };
36
37 exports.getEditProduct = (req, res, next) => {
38     const editMode = req.query.edit;
39     if (!editMode) {
40         return res.redirect('/');
41     }
42     const prodId = req.params.productId;
43     Product.findById(prodId)
44         .then(product => {
45             if (!product) {
46                 return res.redirect('/');
47             }
48             res.render('admin/edit-product', {
49                 pageTitle: 'Edit Product',
50                 path: '/admin/edit-product',
51                 editing: editMode,
52                 product: product
53             });
54         })
55         .catch(err => console.log(err));
56 };
57
58 exports.postEditProduct = (req, res, next) => {
59     const prodId = req.body.productId;
60     const updatedTitle = req.body.title;
61     const updatedPrice = req.body.price;
62     const updatedImageUrl = req.body.imageUrl;
63     const updatedDesc = req.body.description;
64
65     Product
66         .findById(prodId)
67         .then(product => {
68             product.title = updatedTitle
69             product.price = updatedPrice
70             product.description = updatedDesc
71             product.imageUrl = updatedImageUrl
72             return product
73         })
74         .save()
75         .then(result => {
76             console.log('UPDATED PRODUCT!');
77             res.redirect('/admin/products');
78         })
79         .catch(err => console.log(err));
80 };
81
82 /**we wanna get all the user data for the related user and not just the Id
83 */
84 exports.getProducts = (req, res, next) => {
85     Product.find()
86         /**this allows you to define which fields you wanna select or unselect,
87         * so which fields should be retrieved from the database

```

```

88     * and there you pass a string where you could say for a product,
89     * maybe you wanna get the title and the price
90     * but you don't need description and anything else.
91     *
92     * so you could say 'title price'
93     * and you could even exclude something like '_id' by '-' in front of '_id' like '-_id'
94     * the same can be done on 'populate()' by passing a 2nd argument.
95     *
96     */
97     .select('title price -_id')
98     /**'populate()' allows you to tell mongoose to populate a certain field with all the
    detail information and not just the Id
99     * i could add 'populate()'
100    * and then you first of all describe the path which you wanna populate
101    * in my case, that's just the userId field
102    * but you could also point at nested paths
103    */
104    .populate('userId', 'name')
105    .then(products => {
106        console.log(products)
107        res.render('admin/products', {
108            prods: products,
109            pageTitle: 'Admin Products',
110            path: '/admin/products'
111        });
112    })
113    .catch(err => console.log(err));
114 };
115
116 exports.postDeleteProduct = (req, res, next) => {
117     const prodId = req.body.productId;
118     Product.findByIdAndRemove(prodId)
119     .then(() => {
120         console.log('DESTROYED PRODUCT');
121         res.redirect('/admin/products');
122     })
123     .catch(err => console.log(err));
124 };

```

* Chapter 217: Working On The Shopping Cart

1. update
 - ./models/user.js
 - ./models/user.js
 - ./routes/shop.js

A nice Book



\$ 29.99

You should not miss that!

[Details](#)

[Add to Cart](#)

Page Not Found!

EXPLORER

- NODEJS-COMLETE-GUIDE
 - .vscode
 - controllers
 - admin.js
 - error.js
 - shop.js
 - data
 - models
 - product.js
 - user.js
 - node_modules
 - public
 - routes
 - admin.js
 - shop.js
 - util
 - path.js
 - views
 - .gitignore
 - app.js
 - message.txt
 - package-lock.json
 - OUTLINE
 - COMMITTS

```
6
7 const router = express.Router();
8
9 router.get('/', shopController.getIndex);
10
11 router.get('/products', shopController.getProducts);
12
13 router.get('/products/:productId', shopController.getProduct);
14
15 // router.get('/cart', shopController.getCart);
16
17 router.post('/cart', shopController.postCart);
18
19 // router.post('/cart-delete-item', shopController.postCartDeleteProduct);
20
21 // router.post('/create-order', shopController.postOrder);
22
23 // router.get('/orders', shopController.getOrders);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
'http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',
userId: 5bab316ce0a7c75f783cb8a8,
__v: 0 } ]
{ cart: { items: [ [Object] ] },
  _id: 5bab316ce0a7c75f783cb8a8,
  name: 'Max',
  email: 'max@test.com',
  __v: 0 }
```

Ln 17, Col 47 Spaces: 4 UTF-8 LF JavaScript Prettier: ✓

My Cluster Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.7 Enterprise

shop.users DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

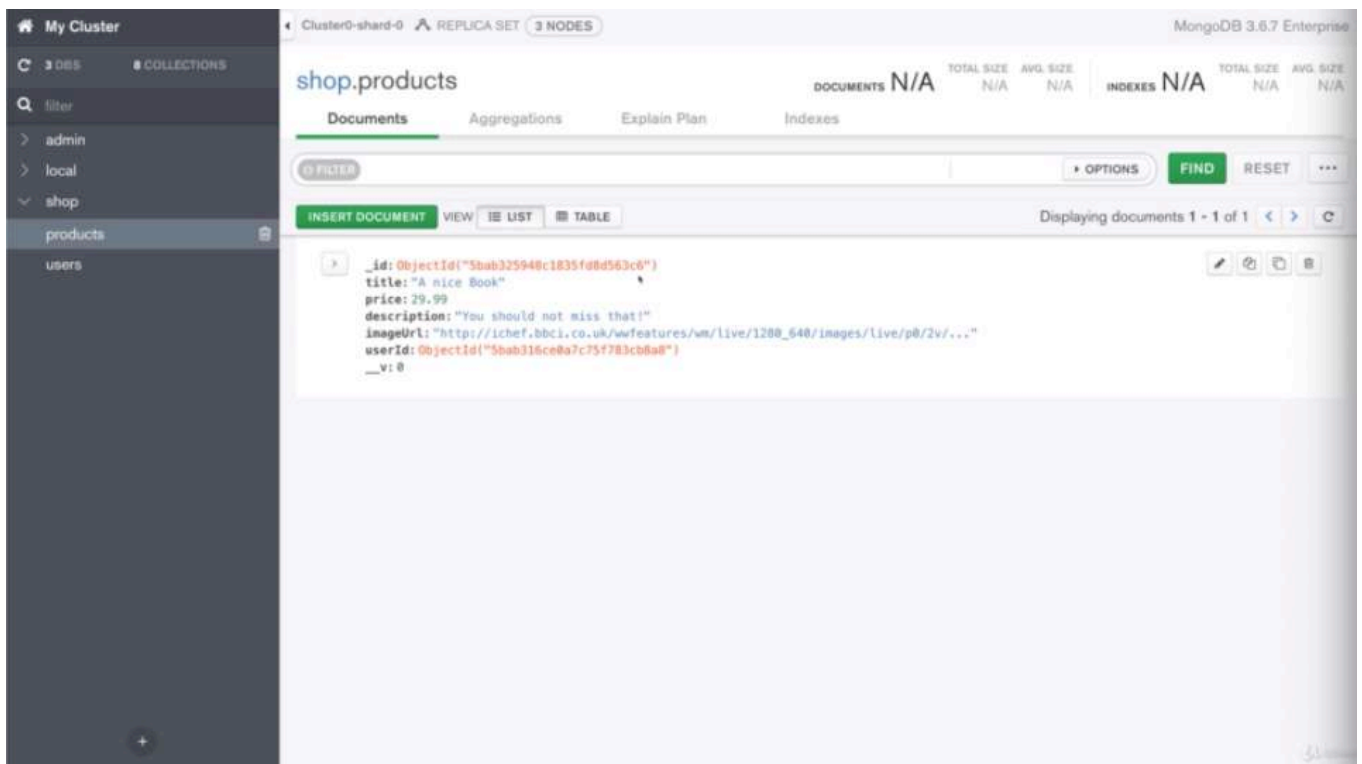
Documents Aggregations Explain Plan Indexes

0 FILTER OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5bab316ce0a7c75f783cb8a8"),
  "cart": {
    "items": Array
      0: {
        "_id": ObjectId("5bab35174da0b56107814f4a"),
        "productId": ObjectId("5bab325948c1835fe8d563c6"),
        "quantity": 1
      }
  },
  "name": "Max",
  "email": "max@test.com",
  "__v": 0
}
```



- the most importantly, i got my productId and that productId which ends with '3c6' should be the Id of that productId

```

1  //./models/user.js
2
3  const mongoose = require('mongoose')
4
5  const Schema = mongoose.Schema
6
7  const userSchema = new Schema({
8    name: {
9      type: String,
10     required: true
11   },
12   email: {
13     type: String,
14     required: true
15   },
16   cart: {
17     items: [{
18       productId: {
19         type: Schema.Types.ObjectId,
20         ref: 'Product',
21         required: true
22       },
23       quantity: {
24         type: Number,
25         required: true
26       }
27     }]
28   }
29 })
30
31 /**'methods' key is an object which allows you to add your own methods
32  * by adding them to Cart

```



```

33 * and it has to be a function written like this
34 * so that the 'this' keyword still refers to the schema and not to something else
35 * and in this function you can add your own logic
36 * and that is exactly what i wanna do
37 * in this method, i wanna add the logic i had in 'addToCart()' before
38 *
39 * the function should also receive the 'product' argument which i wanna add
40 * that is something we required in the past as well.
41 *
42 * 'addToCart' here will be called on a real instance based on that schema.
43 *
44 */
45 userSchema.methods.addToCart = function(product){
46   const cartProductIndex = this.cart.items.findIndex(cp => {
47     return cp.productId.toString() === product._id.toString();
48   });
49   let newQuantity = 1;
50   const updatedCartItems = [...this.cart.items];
51   if (cartProductIndex >= 0) {
52     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
53     updatedCartItems[cartProductIndex].quantity = newQuantity;
54   } else {
55     updatedCartItems.push({
56       /*'new ObjectId' will not work
57        * so i can store product Id like this 'product._id'
58        *
59        * make sure that the names you used up there in your schema are the naes you use down
60        there for creating new data
61        */
62       productId: product._id,
63       quantity: newQuantity
64     });
65   }
66   const updatedCart = {
67     items: updatedCartItems
68   }
69   this.cart = updatedCart
70   /*this should be a utility method that saves itself
71    * so where the object saves itself by using the built-in 'save()' method
72    * where we update the cart
73    */
74   return this.save()
75 }
76
77 module.exports = mongoose.model('User', userSchema)

```

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.find()
7     .then(products => {
8       console.log(products)
9       res.render('shop/product-list', {
10         prods: products,

```

```

11     pageTitle: 'All Products',
12     path: '/products'
13   });
14 })
15 .catch(err => {
16   console.log(err);
17 });
18 };
19
20 exports.getProduct = (req, res, next) => {
21   const prodId = req.params.productId;
22   Product.findById(prodId)
23     .then(product => {
24       res.render('shop/product-detail', {
25         product: product,
26         pageTitle: product.title,
27         path: '/products'
28       });
29     })
30     .catch(err => console.log(err));
31 };
32
33 exports.getIndex = (req, res, next) => {
34   Product.find()
35     .then(products => {
36       res.render('shop/index', {
37         prods: products,
38         pageTitle: 'Shop',
39         path: '/'
40       });
41     })
42     .catch(err => {
43       console.log(err);
44     });
45 };
46
47 exports.getCart = (req, res, next) => {
48   req.user
49     .getCart()
50     .then(products => {
51       res.render('shop/cart', {
52         path: '/cart',
53         pageTitle: 'Your Cart',
54         products: products
55       });
56     })
57     .catch(err => console.log(err));
58 };
59
60 exports.postCart = (req, res, next) => {
61   const prodId = req.body.productId;
62   Product.findById(prodId)
63     .then(product => {
64       return req.user.addToCart(product);
65     })
66     .then(result => {

```

```

67     console.log(result);
68     res.redirect('/cart');
69   });
70 };
71
72 exports.postCartDeleteProduct = (req, res, next) => {
73   const prodId = req.body.productId;
74   req.user
75     .deleteItemFromCart(prodId)
76     .then(result => {
77       res.redirect('/cart');
78     })
79     .catch(err => console.log(err));
80 };
81
82 exports.postOrder = (req, res, next) => {
83   let fetchedCart;
84   req.user
85     .addOrder()
86     .then(result => {
87       res.redirect('/orders');
88     })
89     .catch(err => console.log(err));
90 };
91
92 exports.getOrders = (req, res, next) => {
93   req.user
94     .getOrders()
95     .then(orders => {
96       res.render('shop/orders', {
97         path: '/orders',
98         pageTitle: 'Your Orders',
99         orders: orders
100       });
101     })
102     .catch(err => console.log(err));
103 };
104

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 /*
18 router.get('/cart', shopController.getCart);

```

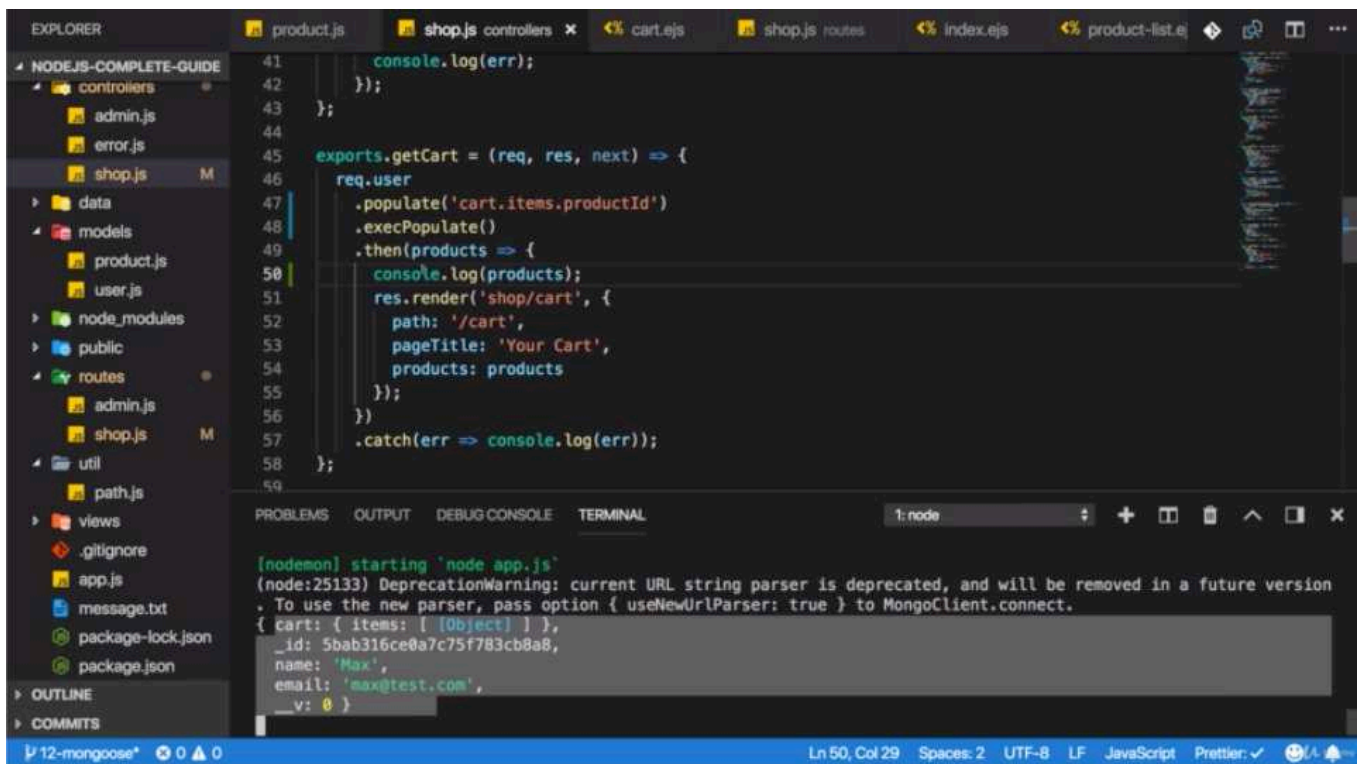
```
19 */
20
21 router.post('/cart', shopController.postCart);
22
23 /*
24 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
25
26 router.post('/create-order', shopController.postOrder);
27
28 router.get('/orders', shopController.getOrders);
29 */
30
31 module.exports = router;
32
```

* Chapter 218: Loading The Cart

1. update
 - ./controllers/shop.js
 - ./routes/shop.js
 - ./views/shop/cart.ejs



No Products in Cart!



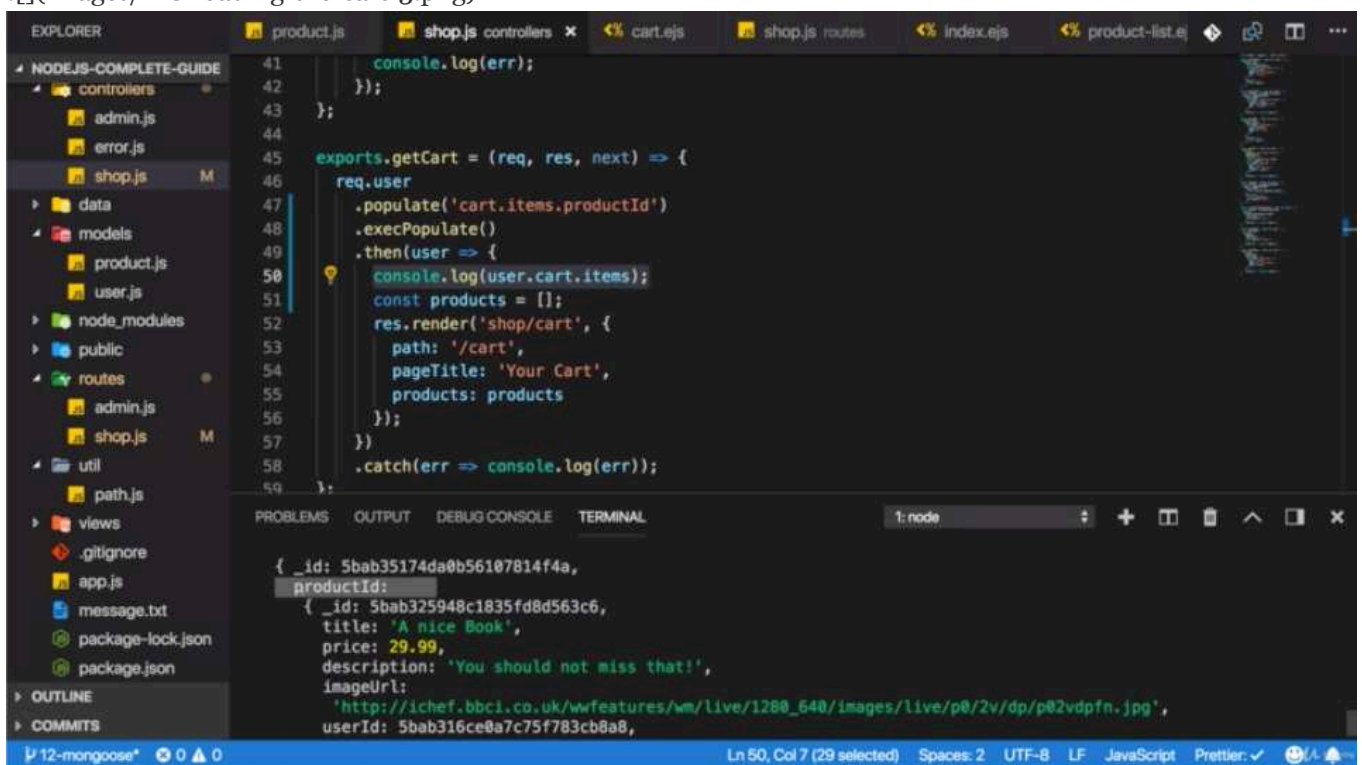
```
41 console.log(err);
42 });
43 };
44
45 exports.getCart = (req, res, next) => {
46   req.user
47     .populate('cart.items.productId')
48     .execPopulate()
49     .then(products => {
50       console.log(products);
51       res.render('shop/cart', {
52         path: '/cart',
53         pageTitle: 'Your Cart',
54         products: products
55       });
56     })
57     .catch(err => console.log(err));
58 };
59
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
[nodemon] starting 'node app.js'
(node:25133) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version.
To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
{ cart: { items: [ [Object] ] },
  _id: 5bab316ce0a7c75f783cb8a8,
  name: 'Max',
  email: 'max@test.com',
  __v: 0 }
```

- we see what we have is the full user object which makes sense because we are not fetching products, we still work with the full user.



```
41 console.log(err);
42 });
43 };
44
45 exports.getCart = (req, res, next) => {
46   req.user
47     .populate('cart.items.productId')
48     .execPopulate()
49     .then(user => {
50       console.log(user.cart.items);
51       const products = [];
52       res.render('shop/cart', {
53         path: '/cart',
54         pageTitle: 'Your Cart',
55         products: products
56       });
57     })
58     .catch(err => console.log(err));
59 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
{ _id: 5bab35174da0b56107814f4a,
  productId:
    { _id: 5bab325948c1835fd8d563c6,
      title: 'A nice Book',
      price: 29.99,
      description: 'You should not miss that!',
      imageUrl:
        'http://ichef.bbci.co.uk/wwfeatures/wm/Live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',
      userId: 5bab316ce0a7c75f783cb8a8,
```

- you see what i log, 'user.cart.items' now is an array of items where the productId is populated with the product data. so now it works a bit different than before but it still gives us the data we need.


```

1 <%= include('../includes/head.ejs') %>
2 <link rel="stylesheet" href="/css/cart.css">
3 </head>
4
5 <body>
6   <%= include('../includes/navigation.ejs') %>
7   <main>
8     <%= if (products.length > 0) { %>
9       <ul class="cart_item-list">
10        <%= products.forEach(p => { %>
11          <li class="cart_item">
12            <h1><%= p.title %></h1>
13            <h2>Quantity: <%= p.quantity %></h2>
14            <form action="/cart-delete-item" method="POST">
15              <input type="hidden" value="<%= p._id %>" name="productId">
16              <button class="btn danger" type="submit">Delete</button>
17            </form>
18          </li>
19        </>
20      </ul>
21    <%= } %>
22  </main>
23 </body>

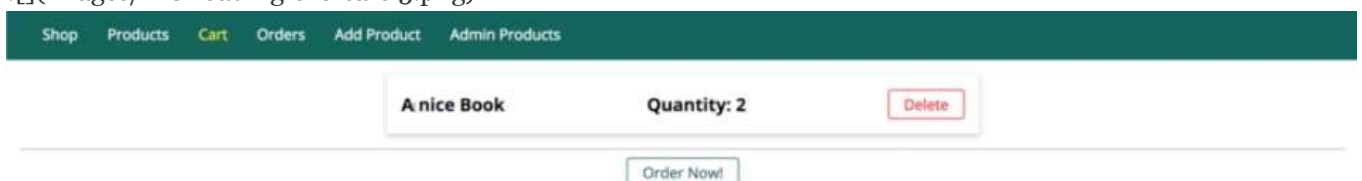
```

```

CoreMongooseArray [
  { _id: 5bab35174da0b56107814f4a,
    productId:
      { _id: 5bab325948c1835fd8d563c6,
        title: 'A nice Book',
        price: 29.99,
        description: 'You should not miss that!',
        imageUrl: 'http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',

```

- we loop through all products which is fine, but our product data will then be nested in a product field and you could also rename this to just product in your schema therefore
 - but i still have 'productId' here. the title is not available on the top-level object which would be this object on the log
 - but on the nested productId object. so we have to say 'p.productId.title' instead 'p.title'.
-



```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.find()
7     .then(products => {

```

```

8     console.log(products)
9     res.render('shop/product-list', {
10         prods: products,
11         pageTitle: 'All Products',
12         path: '/products'
13     });
14 })
15 .catch(err => {
16     console.log(err);
17 });
18 };
19
20 exports.getProduct = (req, res, next) => {
21     const prodId = req.params.productId;
22     Product.findById(prodId)
23     .then(product => {
24         res.render('shop/product-detail', {
25             product: product,
26             pageTitle: product.title,
27             path: '/products'
28         });
29     })
30     .catch(err => console.log(err));
31 };
32
33 exports.getIndex = (req, res, next) => {
34     Product.find()
35     .then(products => {
36         res.render('shop/index', {
37             prods: products,
38             pageTitle: 'Shop',
39             path: '/'
40         });
41     })
42     .catch(err => {
43         console.log(err);
44     });
45 };
46
47 exports.getCart = (req, res, next) => {
48     req.user
49     /** 'populate()' doesn't return a promise 'then()'
50     * so calling 'then()' on it would not work
51     * we have to chain 'execPopulate()' after that
52     * and then we will get a promise.
53     */
54     .populate('cart.items.productId')
55     .execPopulate()
56     .then(user => {
57         const products = user.cart.items
58         res.render('shop/cart', {
59             path: '/cart',
60             pageTitle: 'Your Cart',
61             products: products
62         });
63     })

```

```

64     .catch(err => console.log(err));
65 };
66
67 exports.postCart = (req, res, next) => {
68     const prodId = req.body.productId;
69     Product.findById(prodId)
70     .then(product => {
71         return req.user.addToCart(product);
72     })
73     .then(result => {
74         console.log(result);
75         res.redirect('/cart');
76     });
77 };
78
79 exports.postCartDeleteProduct = (req, res, next) => {
80     const prodId = req.body.productId;
81     req.user
82     .deleteItemFromCart(prodId)
83     .then(result => {
84         res.redirect('/cart');
85     })
86     .catch(err => console.log(err));
87 };
88
89 exports.postOrder = (req, res, next) => {
90     let fetchedCart;
91     req.user
92     .addOrder()
93     .then(result => {
94         res.redirect('/orders');
95     })
96     .catch(err => console.log(err));
97 };
98
99 exports.getOrders = (req, res, next) => {
100     req.user
101     .getOrders()
102     .then(orders => {
103         res.render('shop/orders', {
104             path: '/orders',
105             pageTitle: 'Your Orders',
106             orders: orders
107         });
108     })
109     .catch(err => console.log(err));
110 };
111
112
113 // ./routes/shop.js
114
115 const path = require('path');
116
117 const express = require('express');
118
119 const shopController = require('../controllers/shop');
120

```



```

9  const router = express.Router();
10
11  router.get('/', shopController.getIndex);
12
13  router.get('/products', shopController.getProducts);
14
15  router.get('/products/:productId', shopController.getProduct);
16
17  router.get('/cart', shopController.getCart);
18
19  router.post('/cart', shopController.postCart);
20
21  /*
22  router.post('/cart-delete-item', shopController.postCartDeleteProduct);
23
24  router.post('/create-order', shopController.postOrder);
25
26  router.get('/orders', shopController.getOrders);
27  */
28
29  module.exports = router;
30

```

```

1  <!--./views/shop/cart.ejs-->
2
3  <%- include('../includes/head.ejs') %>
4      <link rel="stylesheet" href="/css/cart.css">
5      </head>
6
7      <body>
8          <%- include('../includes/navigation.ejs') %>
9          <main>
10              <% if (products.length > 0) { %>
11                  <ul class="cart__item-list">
12                      <!--we loop through all products which is fine
13                      but our product data will then be nested in a product field
14                      and you could also rename this to just 'product' in your schema
15
16                      therefore
17
18                      i still have 'productId' here the title is not available on the top-
19                      level object which would be this object on the log
20                      but on the nested productId object, so we have to say
21                      'p.productId.title' instead 'p.title'
22
23                      and the quantity is on the top-level object. so this is fine.
24                      the productId again can be found on the productId nested or embedded
25
26                      document
27
28                      -->
29                      <% products.forEach(p => { %>
30                          <li class="cart__item">
31                              <h1><%= p.productId.title %></h1>
32                              <h2>Quantity: <%= p.quantity %></h2>
33                              <form action="/cart-delete-item" method="POST">
34                                  <input type="hidden" value="<%= p.productId._id %>"
35                                  name="productId">
36                                  <button class="btn danger" type="submit">Delete</button>
37                              </form>

```

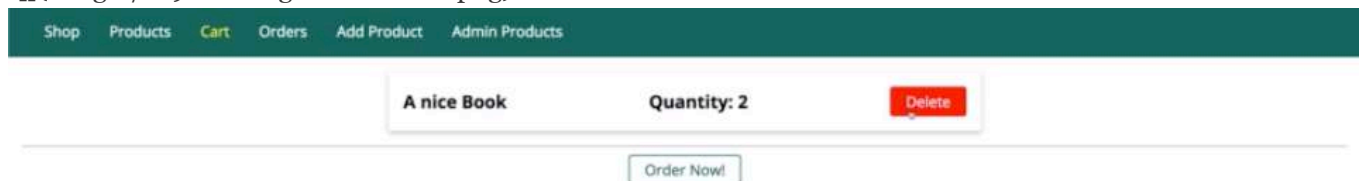
```

30         </li>
31     <% }) %>
32 </ul>
33 <hr>
34 <div class="centered">
35     <form action="/create-order" method="POST">
36         <button type="submit" class="btn">Order Now!</button>
37     </form>
38 </div>
39
40 <% } else { %>
41     <h1>No Products in Cart!</h1>
42 <% } %>
43 </main>
44 <%- include('../includes/end.ejs') %>

```

* Chapter 219: Deleting Cart Items

1. update
 - ./models/user.js
 - ./controllers/shop.js
 - ./routes/shop.js



No Products in Cart!



```
1  ../models/user.js
2
3  const mongoose = require('mongoose')
4
5  const Schema = mongoose.Schema
6
7  const userSchema = new Schema({
8    name: {
9      type: String,
10     required: true
11   },
12   email: {
13     type: String,
14     required: true
15   },
16   cart: {
17     items: [{
18       productId: {
19         type: Schema.Types.ObjectId,
20         ref: 'Product',
21         required: true
22       },
23       quantity: {
24         type: Number,
25         required: true
26       }
27     }]
28   }
29 })
30
31 userSchema.methods.addToCart = function(product){
32   const cartProductIndex = this.cart.items.findIndex(cp => {
33     return cp.productId.toString() === product._id.toString();
34   });
35   let newQuantity = 1;
```

```

36   const updatedCartItems = [...this.cart.items];
37   if (cartProductIndex >= 0) {
38     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
39     updatedCartItems[cartProductIndex].quantity = newQuantity;
40   } else {
41     updatedCartItems.push({
42       productId: product._id,
43       quantity: newQuantity
44     });
45   }
46   const updatedCart = {
47     items: updatedCartItems
48   }
49   this.cart = updatedCart
50   return this.save()
51 }
52
53 userSchema.methods.removeFromCart = function(productId){
54   const updatedCartItems = this.cart.items.filter(item => {
55     return item.productId.toString() !== productId.toString()
56   })
57   this.cart.items = updatedCartItems
58   return this.save()
59 }
60
61 module.exports = mongoose.model('User', userSchema)

```



```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 router.get('/cart', shopController.getCart);
18
19 router.post('/cart', shopController.postCart);
20
21 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
22
23 /*
24 router.post('/create-order', shopController.postOrder);
25
26 router.get('/orders', shopController.getOrders);
27 */
28
29 module.exports = router;
30

```

```
1  ../controllers/shop.js
2
3  const Product = require('../models/product');
4
5  exports.getProducts = (req, res, next) => {
6    Product.find()
7      .then(products => {
8        console.log(products)
9        res.render('shop/product-list', {
10          prods: products,
11          pageTitle: 'All Products',
12          path: '/products'
13        });
14      })
15      .catch(err => {
16        console.log(err);
17      });
18  };
19
20  exports.getProduct = (req, res, next) => {
21    const prodId = req.params.productId;
22    Product.findById(prodId)
23      .then(product => {
24        res.render('shop/product-detail', {
25          product: product,
26          pageTitle: product.title,
27          path: '/products'
28        });
29      })
30      .catch(err => console.log(err));
31  };
32
33  exports.getIndex = (req, res, next) => {
34    Product.find()
35      .then(products => {
36        res.render('shop/index', {
37          prods: products,
38          pageTitle: 'Shop',
39          path: '/'
40        });
41      })
42      .catch(err => {
43        console.log(err);
44      });
45  };
46
47  exports.getCart = (req, res, next) => {
48    req.user
49      .populate('cart.items.productId')
50      .execPopulate()
51      .then(user => {
52        const products = user.cart.items
53        res.render('shop/cart', {
54          path: '/cart',
55          pageTitle: 'Your Cart',
56          products: products
```

```

57     });
58   })
59   .catch(err => console.log(err));
60 };
61
62 exports.postCart = (req, res, next) => {
63   const prodId = req.body.productId;
64   Product.findById(prodId)
65     .then(product => {
66       return req.user.addToCart(product);
67     })
68     .then(result => {
69       console.log(result);
70       res.redirect('/cart');
71     });
72 };
73
74 exports.postCartDeleteProduct = (req, res, next) => {
75   const prodId = req.body.productId;
76   req.user
77     .removeFromCart(prodId)
78     .then(result => {
79       res.redirect('/cart');
80     })
81     .catch(err => console.log(err));
82 };
83
84 exports.postOrder = (req, res, next) => {
85   let fetchedCart;
86   req.user
87     .addOrder()
88     .then(result => {
89       res.redirect('/orders');
90     })
91     .catch(err => console.log(err));
92 };
93
94 exports.getOrders = (req, res, next) => {
95   req.user
96     .getOrders()
97     .then(orders => {
98       res.render('shop/orders', {
99         path: '/orders',
100         pageTitle: 'Your Orders',
101         orders: orders
102       });
103     })
104     .catch(err => console.log(err));
105 };
106

```

* Chapter 220: Creating & Getting Orders

1. update
- ./controllers/shop.js
- ./models/order.js

- ./routes/shop.js



Title
Second Product


Image URL
dasfas

Price
1222

Description
fdasfsa

Add Product

A nice Book




\$ 29.99

You should not miss that!

EditDelete

Second Product

Second Product


\$ 1222

fdasfsa

EditDelete

No Products in Cart!

A nice Book




\$ 29.99

You should not miss that!

Details

Add to Cart

Second Product

Second Product

\$ 1222

fdasfsa

Details

Add to Cart

A nice Book	Quantity: 1	Delete
-------------	-------------	--------

Order Now!

A nice Book



\$ 29.99

You should not miss that!

Details

Add to Cart

Second Product

\$ 1222

fdasfsa

Details

Add to Cart

A nice Book



29.99

You should not miss that!

Add to Cart

A nice Book	Quantity: 2	Delete
-------------	-------------	--------

Order Now!

A nice Book	Quantity: 2	Delete
-------------	-------------	--------

Order Now!

Page Not Found!

x

My Cluster

DBS

COLLECTIONS

filter

> admin

> local

> shop

orders

products

users

Cluster0-shard-0

REPLICA SET

3 NODES

MongoDB 3.6.7 Enterprise

shop.orders

DOCUMENTS N/A

TOTAL SIZE N/A

AVG. SIZE N/A

INDEXES N/A

TOTAL SIZE N/A

AVG. SIZE N/A

Documents

Aggregations

Explain Plan

Indexes

FILTER

OPTIONS

FIND

RESET

...

INSERT DOCUMENT

VIEW

LIST

TABLE

Displaying documents 1 - 1 of 1

>

_id: ObjectId("5bab3bb6673aa76406000591")

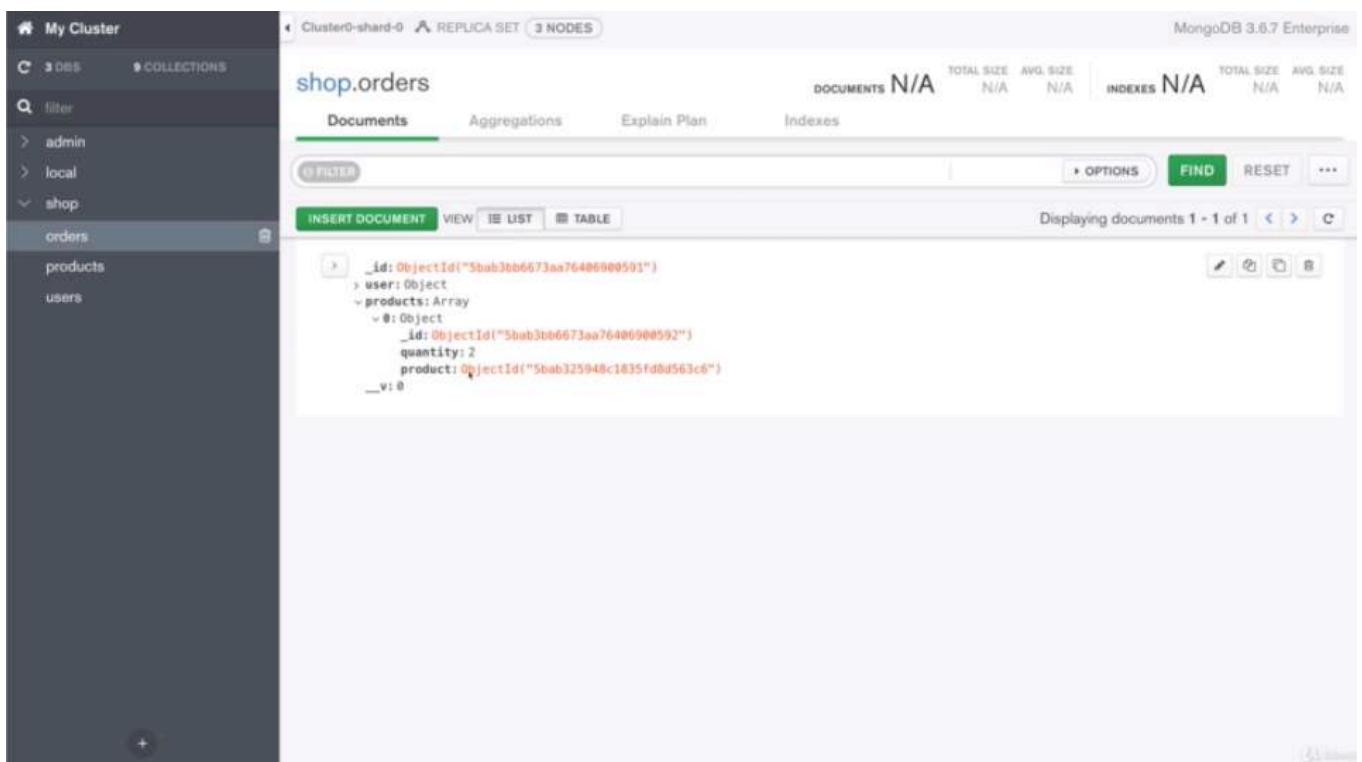
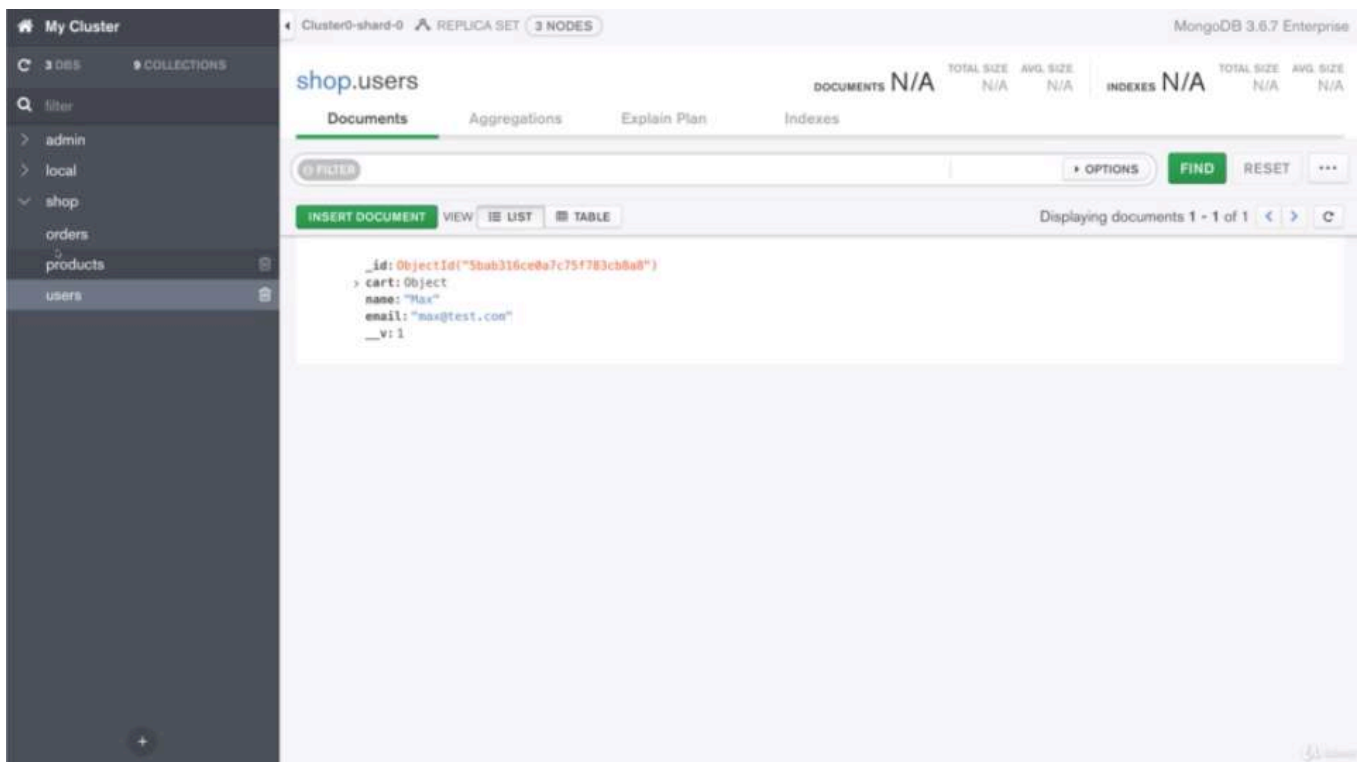
- user: Object

name: "Max"

userId: ObjectId("5bab316ce8a7c75f783cb9a8")

> products: Array

__v: 0



- let's have a look into compass and if we refresh the entire setup, we got the orders collection and in there i got an order with some user data 'userId'.
- let's confirm the userId with users collection. it ends with '8a8'.
- and let's have a look at the products there too. i got a quantity of 2 and i got my productId.

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Order = require('../models/order')
5
6 exports.getProducts = (req, res, next) => {
7   Product.find()
8     .then(products => {
9       console.log(products)

```

```

10     res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products'
14     });
15 }
16 .catch(err => {
17     console.log(err);
18 });
19 };
20
21 exports.getProduct = (req, res, next) => {
22     const prodId = req.params.productId;
23     Product.findById(prodId)
24     .then(product => {
25         res.render('shop/product-detail', {
26             product: product,
27             pageTitle: product.title,
28             path: '/products'
29         });
30     })
31     .catch(err => console.log(err));
32 };
33
34 exports.getIndex = (req, res, next) => {
35     Product.find()
36     .then(products => {
37         res.render('shop/index', {
38             prods: products,
39             pageTitle: 'Shop',
40             path: '/'
41         });
42     })
43     .catch(err => {
44         console.log(err);
45     });
46 };
47
48 exports.getCart = (req, res, next) => {
49     req.user
50     .populate('cart.items.productId')
51     .execPopulate()
52     .then(user => {
53         const products = user.cart.items
54         res.render('shop/cart', {
55             path: '/cart',
56             pageTitle: 'Your Cart',
57             products: products
58         });
59     })
60     .catch(err => console.log(err));
61 };
62
63 exports.postCart = (req, res, next) => {
64     const prodId = req.body.productId;
65     Product.findById(prodId)

```

```

66     .then(product => {
67         return req.user.addToCart(product);
68     })
69     .then(result => {
70         console.log(result);
71         res.redirect('/cart');
72     });
73 };
74
75 exports.postCartDeleteProduct = (req, res, next) => {
76     const prodId = req.body.productId;
77     req.user
78         .removeFromCart(prodId)
79         .then(result => {
80             res.redirect('/cart');
81         })
82         .catch(err => console.log(err));
83 };
84
85 exports.postOrder = (req, res, next) => {
86     /**this is the approach for fetching all the products that are in the users cart*/
87     req.user
88         .populate('cart.items.productId')
89         .execPopulate()
90         .then(user => {
91             const products = user.cart.items.map(i => {
92                 /**we will have a product field
93                  * and the product field should have all the product data
94                  * so that we will store everything that i had in i.productId before
95                  * because that was the old structure we had in there.
96                  * now we have this structure.
97                  * we have an array of products which just have a quantity
98                  * and then the product detail data which is exactly the structure we expect to get in
99                  *
100                 */
101                 return { quantity: i.quantity, product: i.productId }
102             })
103             /**initialize */
104             const order = new Order({
105                 user: {
106                     name: req.user.name,
107                     userId: req.user
108                 },
109                 products: products
110             })
111             order.save()
112         })
113         .then(result => {
114             res.redirect('/orders');
115         })
116         .catch(err => console.log(err));
117 };
118
119 exports.getOrders = (req, res, next) => {
120     req.user

```

```

121     .getOrders()
122     .then(orders => {
123         res.render('shop/orders', {
124             path: '/orders',
125             pageTitle: 'Your Orders',
126             orders: orders
127         });
128     })
129     .catch(err => console.log(err));
130 };
131

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 router.get('/cart', shopController.getCart);
18
19 router.post('/cart', shopController.postCart);
20
21 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
22
23 router.post('/create-order', shopController.postOrder);
24
25 /*
26 router.get('/orders', shopController.getOrders);
27 */
28
29 module.exports = router;
30

```

```

1 // ./models/order.js
2
3 const mongoose = require('mongoose')
4
5 const Schema = mongoose.Schema
6
7 const orderSchema = new Schema({
8     products: [{
9         product: { type: Object, required: true },
10        quantity: { type: Number, required: true }
11    }],
12    user: {
13        name: {
14            type: String,

```



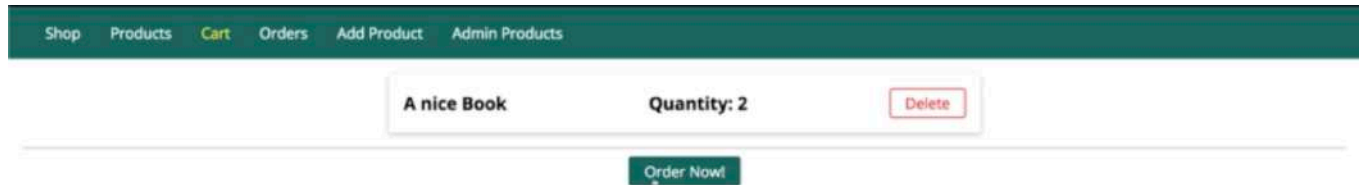
```

15         required: true
16     },
17     userId: {
18         type: Schema.Types.ObjectId,
19         required: true,
20         ref: 'User'
21     }
22 }
23 });
24
25 module.exports = mongoose.model('Order', orderSchema)

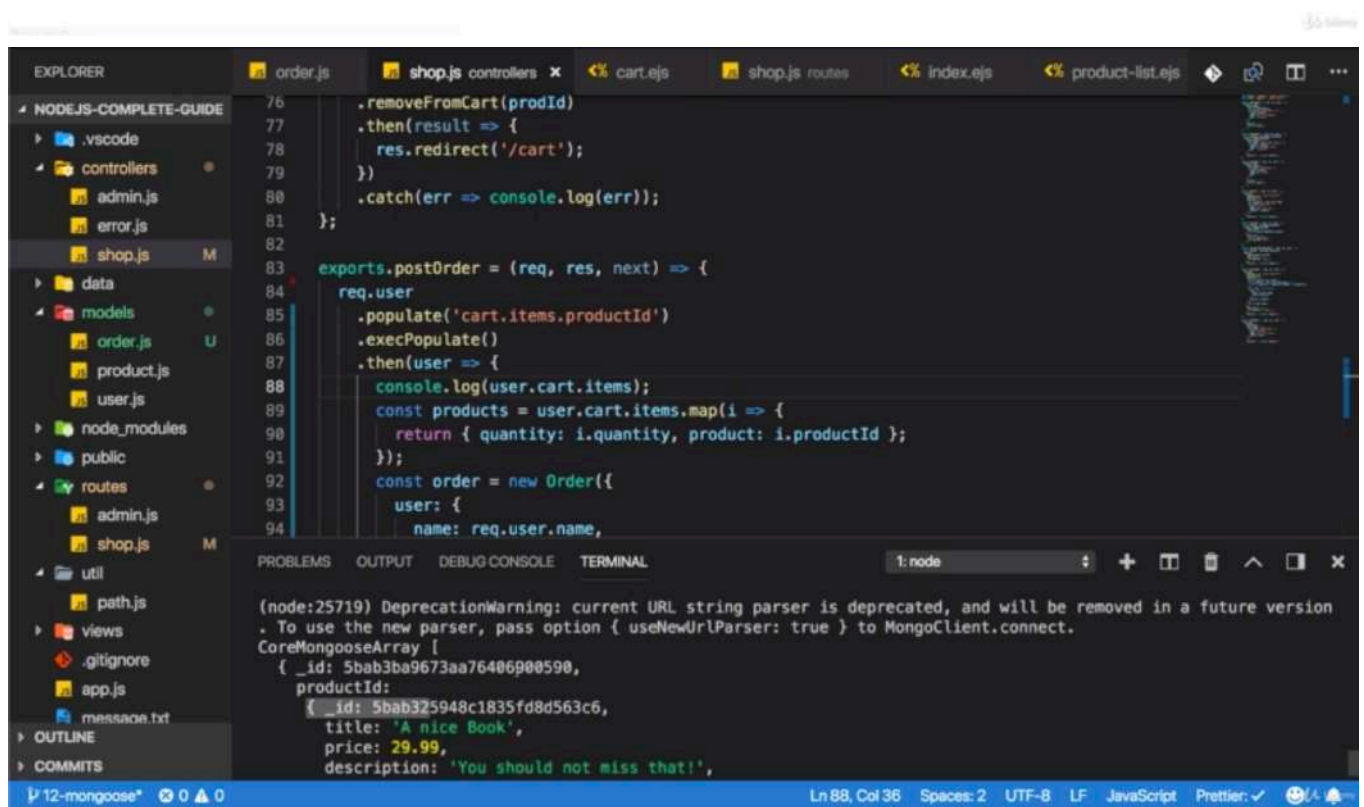
```

* Chapter 221: Storing All Order Related Data

1. update
- ./controllers/shop.js



Page Not Found!



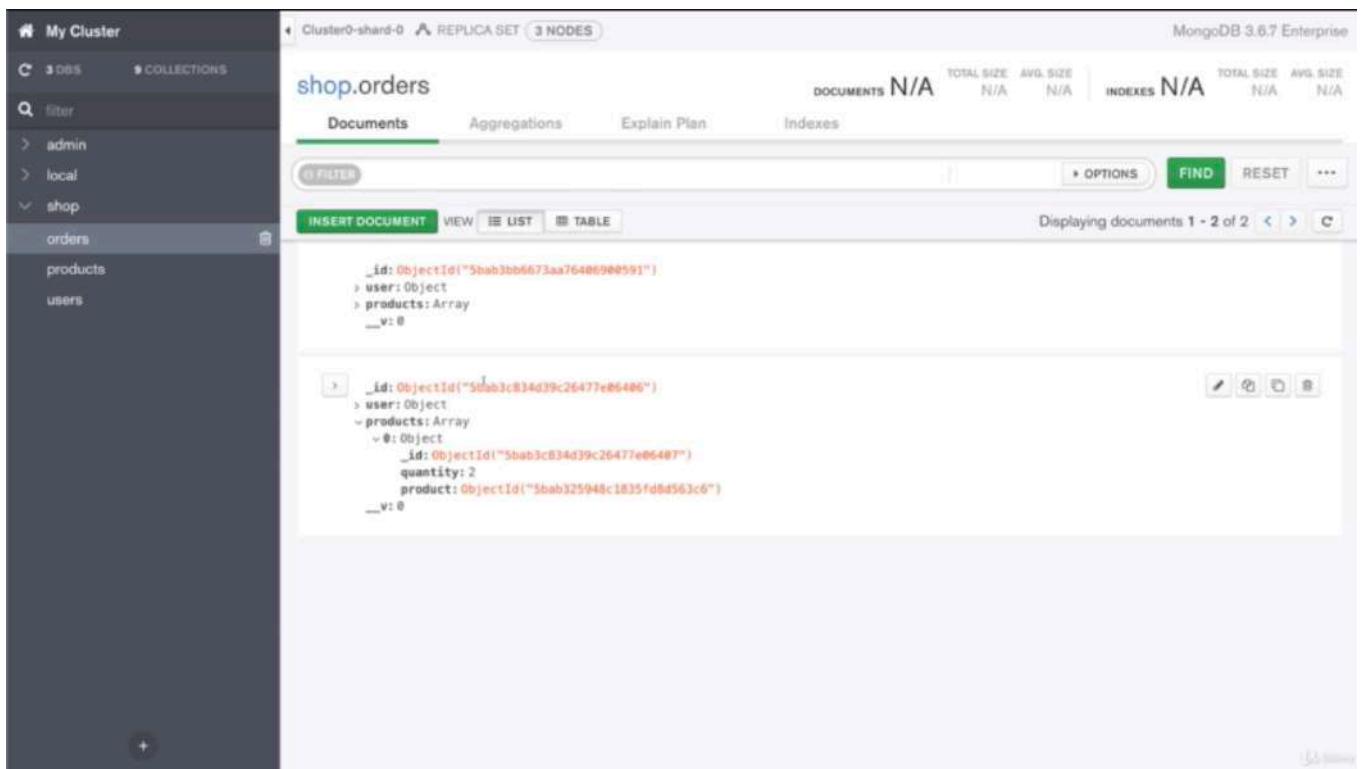
The screenshot shows a VS Code editor with a 'Page Not Found!' message at the top. The Explorer panel on the left shows a file tree for a project named '12-mongoose'. The main editor area displays the 'order.js' file, which contains a function 'postOrder' that logs user cart items and creates a new order. The terminal window at the bottom shows a deprecation warning from Node.js and a log output from MongoDB showing a successful order creation with a full product object.

```
76 .removeFromCart(prodId)
77 .then(result => {
78   res.redirect('/cart');
79 })
80 .catch(err => console.log(err));
81 };
82
83 exports.postOrder = (req, res, next) => {
84   req.user
85     .populate('cart.items.productId')
86     .execPopulate()
87     .then(user => {
88       console.log(user.cart.items);
89       const products = user.cart.items.map(i => {
90         return { quantity: i.quantity, product: i.productId };
91       });
92       const order = new Order({
93         user: {
94           name: req.user.name,
```

(node:25719) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

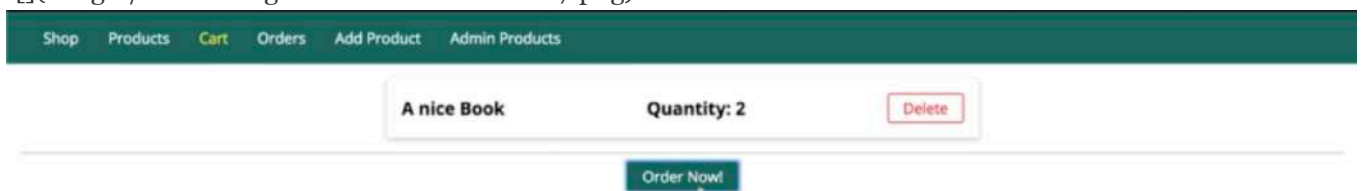
```
CoreMongooseArray [
  { _id: 5bab3ba9673aa76406p00590,
    productId:
      { _id: 5bab325948c1835fd8d563c6,
        title: 'A nice Book',
        price: 29.99,
        description: 'You should not miss that!',
```

- if i click order now and i increase the console, we see productId does hold a full object and not just the Id which is what gets stored.

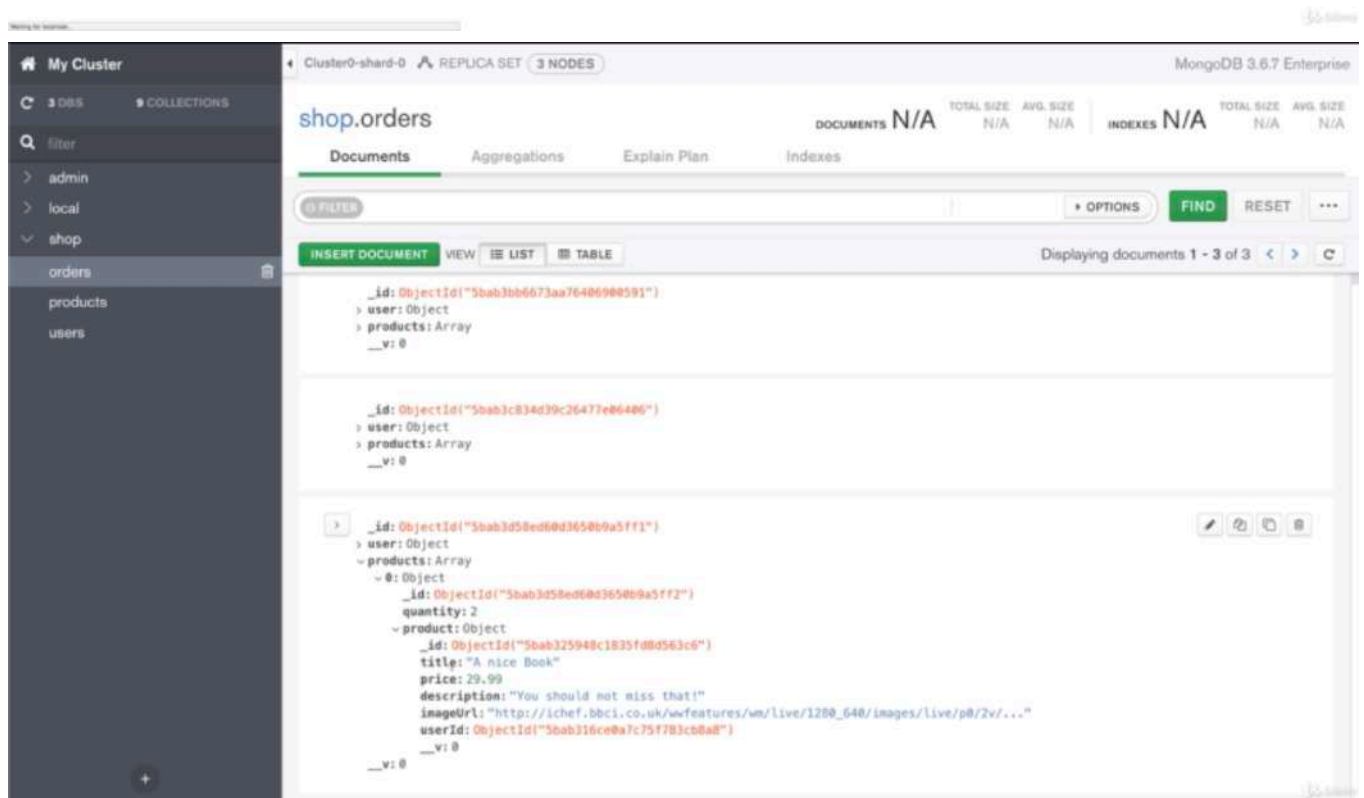


- if i refresh my orders in MongoDB Compass, i got 2 but both orders just have the productId in there, not the full product. i wanna have the full product data though.

- so when i store the productId, i can wrap that in curly braces to create a new javascript object.



Page Not Found!



- if i go back to my compass interface and i have a look at this new order, i see i got all the product detail data in there too.

- and this allows me to store all the data i wanna store with every order.

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Order = require('../models/order')
5
6 exports.getProducts = (req, res, next) => {
7   Product.find()
8     .then(products => {
9       console.log(products)
10      res.render('shop/product-list', {

```

```

11     prods: products,
12     pageTitle: 'All Products',
13     path: '/products'
14   });
15 }
16 .catch(err => {
17   console.log(err);
18 });
19 };
20
21 exports.getProduct = (req, res, next) => {
22   const prodId = req.params.productId;
23   Product.findById(prodId)
24     .then(product => {
25       res.render('shop/product-detail', {
26         product: product,
27         pageTitle: product.title,
28         path: '/products'
29       });
30     })
31     .catch(err => console.log(err));
32 };
33
34 exports.getIndex = (req, res, next) => {
35   Product.find()
36     .then(products => {
37       res.render('shop/index', {
38         prods: products,
39         pageTitle: 'Shop',
40         path: '/'
41       });
42     })
43     .catch(err => {
44       console.log(err);
45     });
46 };
47
48 exports.getCart = (req, res, next) => {
49   req.user
50     .populate('cart.items.productId')
51     .execPopulate()
52     .then(user => {
53       const products = user.cart.items
54       res.render('shop/cart', {
55         path: '/cart',
56         pageTitle: 'Your Cart',
57         products: products
58       });
59     })
60     .catch(err => console.log(err));
61 };
62
63 exports.postCart = (req, res, next) => {
64   const prodId = req.body.productId;
65   Product.findById(prodId)
66     .then(product => {

```

```

67     return req.user.addToCart(product);
68   })
69   .then(result => {
70     console.log(result);
71     res.redirect('/cart');
72   });
73 };
74
75 exports.postCartDeleteProduct = (req, res, next) => {
76   const prodId = req.body.productId;
77   req.user
78     .removeFromCart(prodId)
79     .then(result => {
80       res.redirect('/cart');
81     })
82     .catch(err => console.log(err));
83 };
84
85 exports.postOrder = (req, res, next) => {
86   req.user
87     .populate('cart.items.productId')
88     .execPopulate()
89     .then(user => {
90       /**if i click 'order now'
91        * and i increase the console,
92        * we see productId does hold a full object
93        * and not just the Id which is what gets stored.
94        *
95        * if i refresh my orders in MongoDB Compass,
96        * i got 2 but both orders have the productId in there,
97        * not the full product,
98        * but i wanna have the full product data though.
99        *
100       * so when i store the productId,
101       * i can wrap that in curly braces to create a new javascript object.
102       * and use the spread operator
103       * and use that not directly on the productId
104       * but on a special field,
105       * Mongoose gives me '_doc'
106       * i can access 'productId'
107       * because productId will be an object with a lot of metadata attached to it
108       * even though we can't directly see that when console logging it
109       * but with '._doc',
110       * we get access to the data that is in there
111       * and then with the spread operator inside of a new object,
112       * we pull out all the data in that document we retrieved
113       * and store it in a new object which we save here as a product
114       */
115       console.log(user.cart.items)
116       const products = user.cart.items.map(i => {
117
118         return { quantity: i.quantity, product: { ...i.productId._doc } }
119       })
120       const order = new Order({
121         user: {
122           name: req.user.name,

```

```


123         userId: req.user
124     },
125     products: products
126 })
127 order.save()
128 })
129 .then(result => {
130     res.redirect('/orders');
131 })
132 .catch(err => console.log(err));
133 };
134
135 exports.getOrders = (req, res, next) => {
136     req.user
137     .getOrders()
138     .then(orders => {
139         res.render('shop/orders', {
140             path: '/orders',
141             pageTitle: 'Your Orders',
142             orders: orders
143         });
144     })
145     .catch(err => console.log(err));
146 };
147

```

* Chapter 222: Clearing The Cart After Storing An Order

1. update
 - ./controllers/shop.js
 - ./models/user.js

A nice Book




\$ 29.99

You should not miss that!

Details

Add to Cart

Second Product

Second Product

\$ 1222

fdasfsa

Details

Add to Cart

135 items

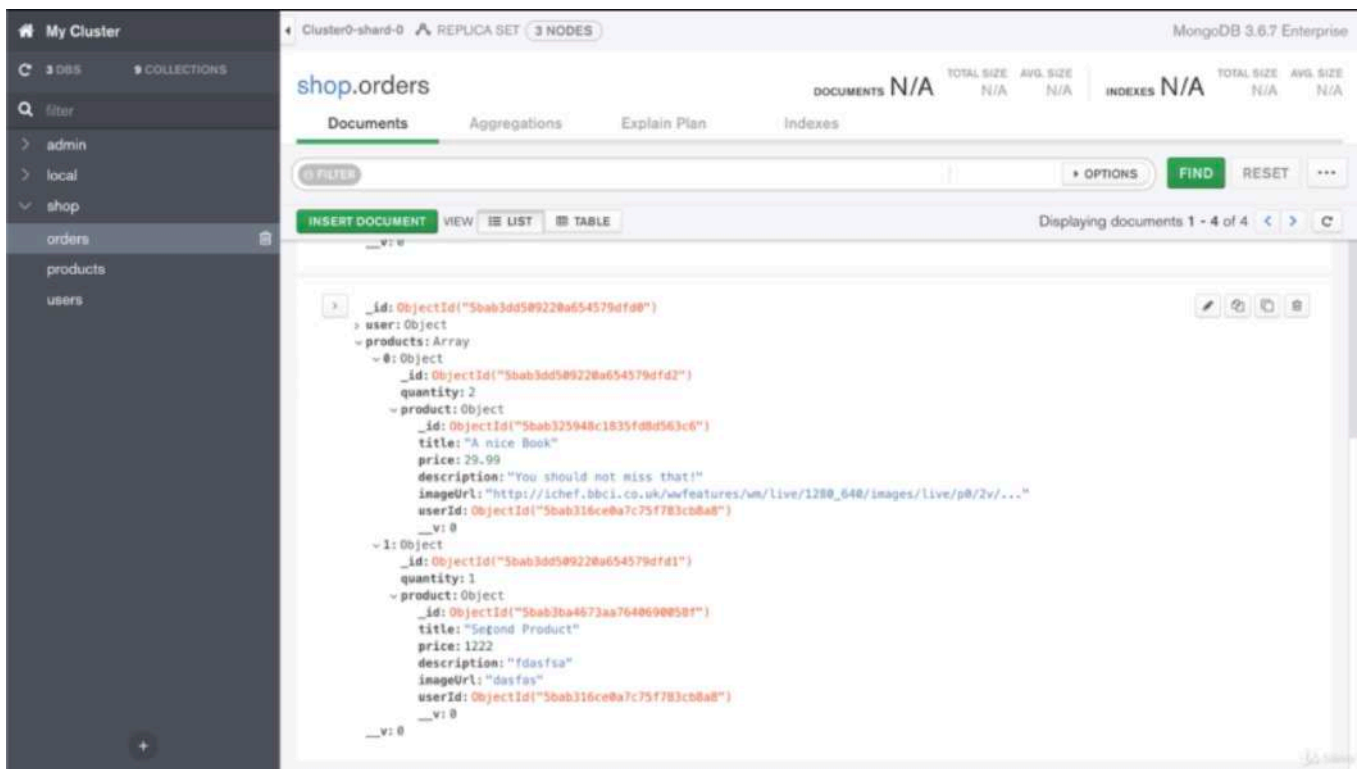
A nice Book	Quantity: 2	Delete
Second Product	Quantity: 1	Delete

Order Now!

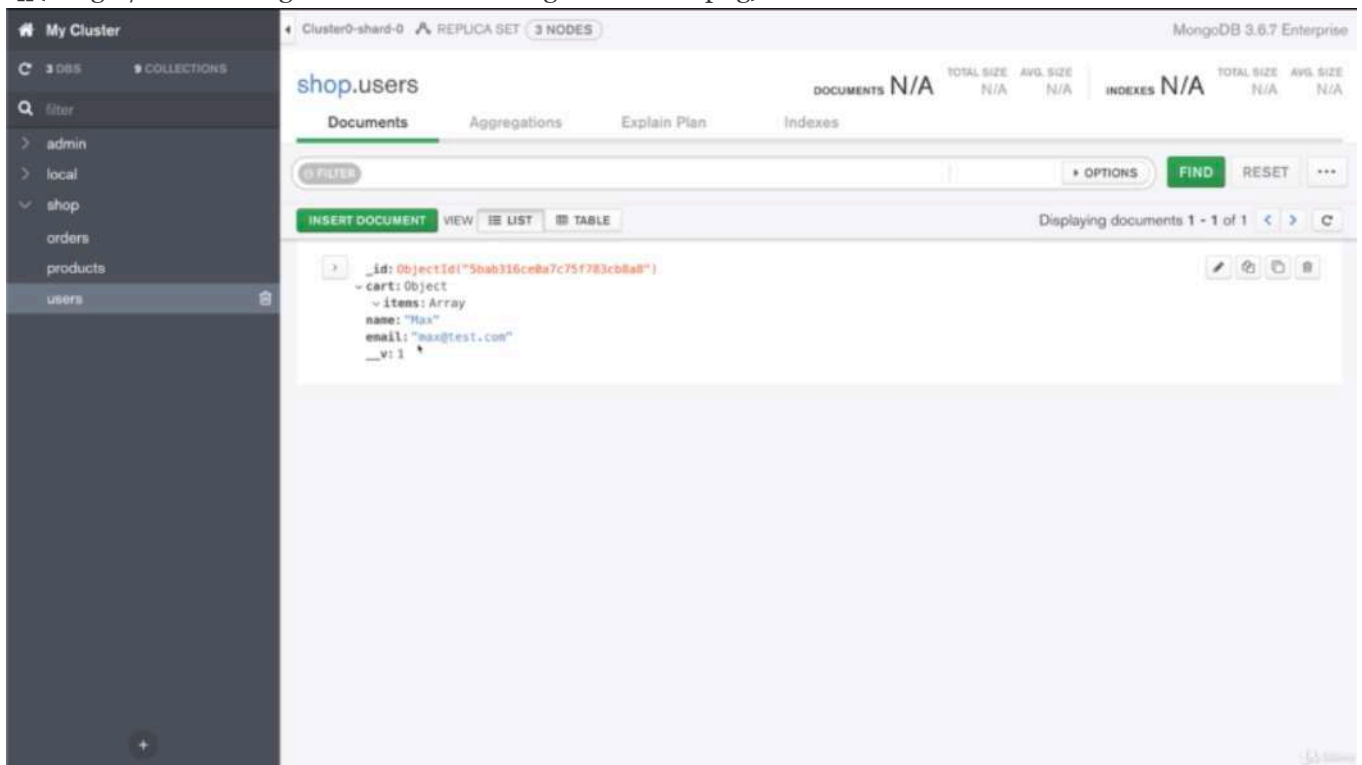
135 items

Page Not Found!

No Products in Cart!



- this new order should have 2 products. one with quantity 2, that was our 'nice book' and one with quantity 1, that was the 'second product'



- and in the users, the cart is empty.

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Order = require('../models/order')
5
6 exports.getProducts = (req, res, next) => {
7   Product.find()
8     .then(products => {
9       console.log(products)

```

```

10     res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products'
14     });
15 }
16 .catch(err => {
17     console.log(err);
18 });
19 };
20
21 exports.getProduct = (req, res, next) => {
22     const prodId = req.params.productId;
23     Product.findById(prodId)
24         .then(product => {
25             res.render('shop/product-detail', {
26                 product: product,
27                 pageTitle: product.title,
28                 path: '/products'
29             });
30         })
31         .catch(err => console.log(err));
32 };
33
34 exports.getIndex = (req, res, next) => {
35     Product.find()
36         .then(products => {
37             res.render('shop/index', {
38                 prods: products,
39                 pageTitle: 'Shop',
40                 path: '/'
41             });
42         })
43         .catch(err => {
44             console.log(err);
45         });
46 };
47
48 exports.getCart = (req, res, next) => {
49     req.user
50         .populate('cart.items.productId')
51         .execPopulate()
52         .then(user => {
53             const products = user.cart.items
54             res.render('shop/cart', {
55                 path: '/cart',
56                 pageTitle: 'Your Cart',
57                 products: products
58             });
59         })
60         .catch(err => console.log(err));
61 };
62
63 exports.postCart = (req, res, next) => {
64     const prodId = req.body.productId;
65     Product.findById(prodId)

```

```

66     .then(product => {
67         return req.user.addToCart(product);
68     })
69     .then(result => {
70         console.log(result);
71         res.redirect('/cart');
72     });
73 };
74
75 exports.postCartDeleteProduct = (req, res, next) => {
76     const prodId = req.body.productId;
77     req.user
78         .removeFromCart(prodId)
79         .then(result => {
80             res.redirect('/cart');
81         })
82         .catch(err => console.log(err));
83 };
84
85 exports.postOrder = (req, res, next) => {
86     req.user
87         .populate('cart.items.productId')
88         .execPopulate()
89         .then(user => {
90             console.log(user.cart.items)
91             const products = user.cart.items.map(i => {
92
93                 return { quantity: i.quantity, product: { ...i.productId._doc } }
94             })
95             const order = new Order({
96                 user: {
97                     name: req.user.name,
98                     userId: req.user
99                 },
100                 products: products
101             })
102             order.save()
103         })
104         .then(result => {
105             return req.user.clearCart()
106         })
107         .then(() => {
108             res.redirect('/orders');
109         })
110         .catch(err => console.log(err));
111 };
112
113 exports.getOrders = (req, res, next) => {
114     req.user
115         .getOrders()
116         .then(orders => {
117             res.render('shop/orders', {
118                 path: '/orders',
119                 pageTitle: 'Your Orders',
120                 orders: orders
121             });

```

```
122     })
123     .catch(err => console.log(err));
124 };
125
```

```
1  //./models/user.js
2
3  const mongoose = require('mongoose')
4
5  const Schema = mongoose.Schema
6
7  const userSchema = new Schema({
8    name: {
9      type: String,
10     required: true
11   },
12   email: {
13     type: String,
14     required: true
15   },
16   cart: {
17     items: [{
18       productId: {
19         type: Schema.Types.ObjectId,
20         ref: 'Product',
21         required: true
22       },
23       quantity: {
24         type: Number,
25         required: true
26       }
27     }]
28   }
29 })
30
31 userSchema.methods.addToCart = function(product){
32   const cartProductIndex = this.cart.items.findIndex(cp => {
33     return cp.productId.toString() === product._id.toString();
34   });
35   let newQuantity = 1;
36   const updatedCartItems = [...this.cart.items];
37   if (cartProductIndex >= 0) {
38     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
39     updatedCartItems[cartProductIndex].quantity = newQuantity;
40   } else {
41     updatedCartItems.push({
42       productId: product._id,
43       quantity: newQuantity
44     });
45   }
46   const updatedCart = {
47     items: updatedCartItems
48   }
49   this.cart = updatedCart
50   return this.save()
51 }
52
```

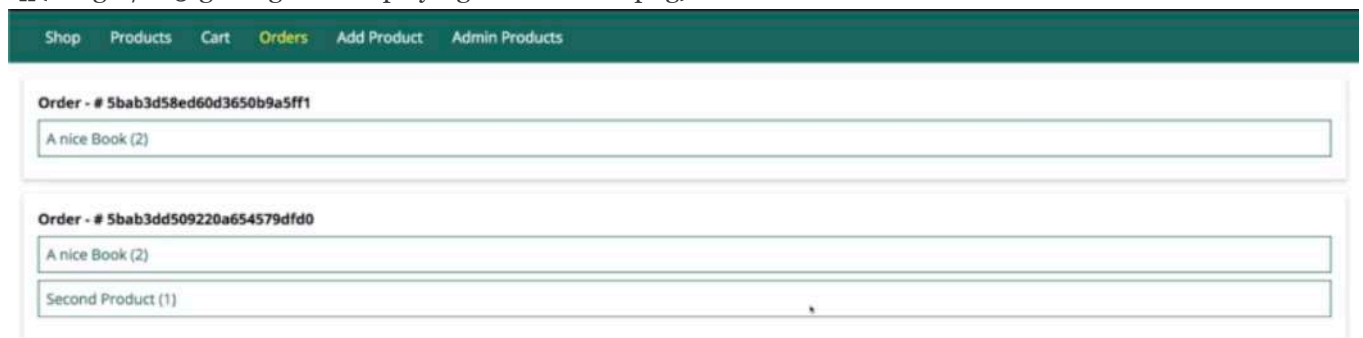
```

53 userSchema.methods.removeFromCart = function(productId){
54   const updatedCartItems = this.cart.items.filter(item => {
55     return item.productId.toString() !== productId.toString()
56   })
57   this.cart.items = updatedCartItems
58   return this.save()
59 }
60
61 userSchema.methods.clearCart = function(){
62   this.cart = { items: [] }
63   return this.save()
64 }
65
66 module.exports = mongoose.model('User', userSchema)
67

```

* Chapter 223: Getting & Displaying The Orders

1. update
- ./controllers/shop.js
- ./models/user.js
- ./routes/shop.js
- ./views/shop/orders.ejs



35 Orders

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4 const Order = require('../models/order')
5
6 exports.getProducts = (req, res, next) => {
7   Product.find()
8     .then(products => {
9       console.log(products)

```

```

10     res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products'
14     });
15 }
16 .catch(err => {
17     console.log(err);
18 });
19 };
20
21 exports.getProduct = (req, res, next) => {
22     const prodId = req.params.productId;
23     Product.findById(prodId)
24         .then(product => {
25             res.render('shop/product-detail', {
26                 product: product,
27                 pageTitle: product.title,
28                 path: '/products'
29             });
30         })
31         .catch(err => console.log(err));
32 };
33
34 exports.getIndex = (req, res, next) => {
35     Product.find()
36         .then(products => {
37             res.render('shop/index', {
38                 prods: products,
39                 pageTitle: 'Shop',
40                 path: '/'
41             });
42         })
43         .catch(err => {
44             console.log(err);
45         });
46 };
47
48 exports.getCart = (req, res, next) => {
49     req.user
50         .populate('cart.items.productId')
51         .execPopulate()
52         .then(user => {
53             const products = user.cart.items
54             res.render('shop/cart', {
55                 path: '/cart',
56                 pageTitle: 'Your Cart',
57                 products: products
58             });
59         })
60         .catch(err => console.log(err));
61 };
62
63 exports.postCart = (req, res, next) => {
64     const prodId = req.body.productId;
65     Product.findById(prodId)

```

```

66     .then(product => {
67         return req.user.addToCart(product);
68     })
69     .then(result => {
70         console.log(result);
71         res.redirect('/cart');
72     });
73 };
74
75 exports.postCartDeleteProduct = (req, res, next) => {
76     const prodId = req.body.productId;
77     req.user
78         .removeFromCart(prodId)
79         .then(result => {
80             res.redirect('/cart');
81         })
82         .catch(err => console.log(err));
83 };
84
85 exports.postOrder = (req, res, next) => {
86     req.user
87         .populate('cart.items.productId')
88         .execPopulate()
89         .then(user => {
90             console.log(user.cart.items)
91             const products = user.cart.items.map(i => {
92
93                 return { quantity: i.quantity, product: { ...i.productId._doc } }
94             })
95             const order = new Order({
96                 user: {
97                     name: req.user.name,
98                     userId: req.user
99                 },
100                 products: products
101             })
102             order.save()
103         })
104         .then(result => {
105             return req.user.clearCart()
106         })
107         .then(() => {
108             res.redirect('/orders');
109         })
110         .catch(err => console.log(err));
111 };
112
113 exports.getOrders = (req, res, next) => {
114     /**i can find all orders
115     * and let's have a look at the ./models/order.js
116     * 'userId' is nested object in 'user' object,
117     * this nested key is equal to the 'userId' of the logged-in user.
118     * so '"user.userId": req.user._id'
119     * this is the check i wanna make
120     * and this will give me all orders that belong to that user.
121     */

```



```

122  *
123  */
124  Order
125    .find({ 'user.userId': req.user._id })
126    .then(orders => {
127      res.render('shop/orders', {
128        path: '/orders',
129        pageTitle: 'Your Orders',
130        orders: orders
131      });
132    })
133    .catch(err => console.log(err));
134 };
135

```

```

1  //./models/user.js
2
3  const mongoose = require('mongoose')
4
5  const Schema = mongoose.Schema
6
7  const userSchema = new Schema({
8    name: {
9      type: String,
10     required: true
11   },
12   email: {
13     type: String,
14     required: true
15   },
16   cart: {
17     items: [{
18       productId: {
19         type: Schema.Types.ObjectId,
20         ref: 'Product',
21         required: true
22       },
23       quantity: {
24         type: Number,
25         required: true
26       }
27     }]
28   }
29 })
30
31 userSchema.methods.addToCart = function(product){
32   const cartProductIndex = this.cart.items.findIndex(cp => {
33     return cp.productId.toString() === product._id.toString();
34   });
35   let newQuantity = 1;
36   const updatedCartItems = [...this.cart.items];
37   if (cartProductIndex >= 0) {
38     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
39     updatedCartItems[cartProductIndex].quantity = newQuantity;
40   } else {
41     updatedCartItems.push({
42       productId: product._id,

```

```

43     quantity: newQuantity
44   });
45 }
46 const updatedCart = {
47   items: updatedCartItems
48 }
49 this.cart = updatedCart
50 return this.save()
51 }
52
53 userSchema.methods.removeFromCart = function(productId){
54   const updatedCartItems = this.cart.items.filter(item => {
55     return item.productId.toString() !== productId.toString()
56   })
57   this.cart.items = updatedCartItems
58   return this.save()
59 }
60
61 userSchema.methods.clearCart = function(){
62   this.cart = { items: [] }
63   return this.save()
64 }
65
66 module.exports = mongoose.model('User', userSchema)
67

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8
9 const router = express.Router();
10
11 router.get('/', shopController.getIndex);
12
13 router.get('/products', shopController.getProducts);
14
15 router.get('/products/:productId', shopController.getProduct);
16
17 router.get('/cart', shopController.getCart);
18
19 router.post('/cart', shopController.postCart);
20
21 router.post('/cart-delete-item', shopController.postCartDeleteProduct);
22
23 router.post('/create-order', shopController.postOrder);
24
25 router.get('/orders', shopController.getOrders);
26
27 module.exports = router;
28

```

```

1 <!--./views/shop/orders.ejs-->
2

```

```

3 <%- include('../includes/head.ejs') %>
4 </head>
5
6 <body>
7   <%- include('../includes/navigation.ejs') %>
8   <main>
9     <% if (orders.length <= 0) { %>
10      <h1>Nothing there!</h1>
11    <% } else { %>
12      <ul>
13        <% orders.forEach(order => { %>
14          <li>
15            <h1># <%= order._id %></h1>
16            <ul>
17              <% order.products.forEach(p => { %>
18                <!--inside here, we have the product
19                and there we have a nested product field
20                so we could also name this just 'p' to avoid confusion
21                this will be 'p'
22
23                we have the product field with the title
24                but directly on the top level 'p' object,
25                so directly in the object that is stored in the products
26                array,
27
28                we have the quantity
29                so we can still access 'p.quantity' directly on 'p'
30                which is the part directly in order products
31                but then the product data itself is nested in one
32                additional embedded document product
33
34                -->
35                <li class="orders__products-item"><%= p.product.title %>
36                  (<%= p.quantity %>)</li>
37              <% }); %>
38            </ul>
39          </li>
40        <% }); %>
41      </ul>
42    <% } %>
43  </main>
44  <%- include('../includes/end.ejs') %>

```