

12. Working With NoSQL & Using MongoDB

* Chapter 172: Module Introduction



What's In This Module?

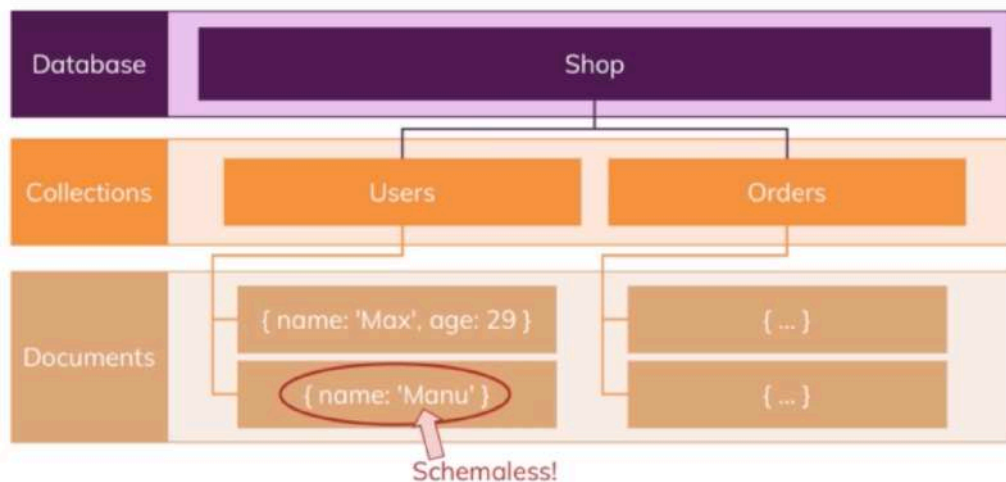
What is MongoDB?

Using the MongoDB Driver in Node.js Apps

Academind

* Chapter 173: What Is MongoDB?

How it works



- for example a shop database, now in such a database in the SQL world, we would have multiple tables.
 - in the NoSQL MongoDB world, we have multiple collections like the users and orders collection for example.
 - inside of each collection, we don't have so-called 'records' but we have a couple of 'documents'. 'documents' also look different than records did and it's not just about different names being used. the core philosophy behind the database is totally different one.
 - for example, MongoDB is schemaless, inside of one collection, your documents which is your data, your entry don't have to have the same structure. but in SQL that was totally different. we had a users table and in that users table, we had an ID, a name, an email, a password. but in here we can have any kind of data in one and the same collection. often you will still end up with an at least similar structure but you are not forced to have exactly same structure. and this gives you more flexibility, also for your application to grow and to change its data requirements overtime without that being difficult to depict in your database world.
-

JSON (BSON) Data Format

```
{
  "name": "Max",
  "age": 29,
  "address": {
    "city": "Munich"
  },
  "hobbies": [
    { "name": "Cooking" },
    { "name": "Sports" }
  ]
}
```

- A Document in MongoDB looks like this.
- MongoDB uses JSON to store data in collections. to be very precise, MongoDB uses something which is called BSON for binary JSON but that only means that MongoDB kind of transforms this behind the scenes before storing it in the files. but you will basically use it as JSON.
- you can also have arrays inside of that document and that array can hold other documents, other objects or it could also just hold strings, numbers, anything of that kind.
- the existence of these nested documents also means that relations are managed a bit differently in the NoSQL MongoDB world.

* Chapter 174: Relations In NoSQL

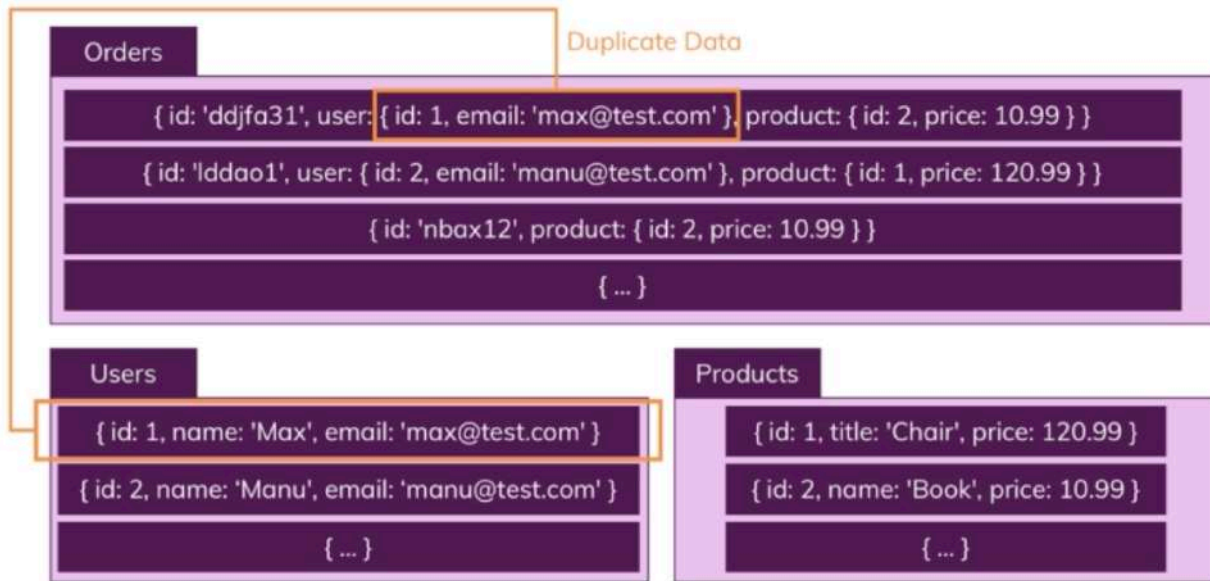


JSON (BSON) Data Format

```
{
  "name": "Max",
  "age": 29,
  "address": {
    "city": "Munich"
  },
  "hobbies": [
    { "name": "Cooking" },
    { "name": "Sports" }
  ]
}
```

- this gives you more flexibility, also regarding the storage of relations between different data.

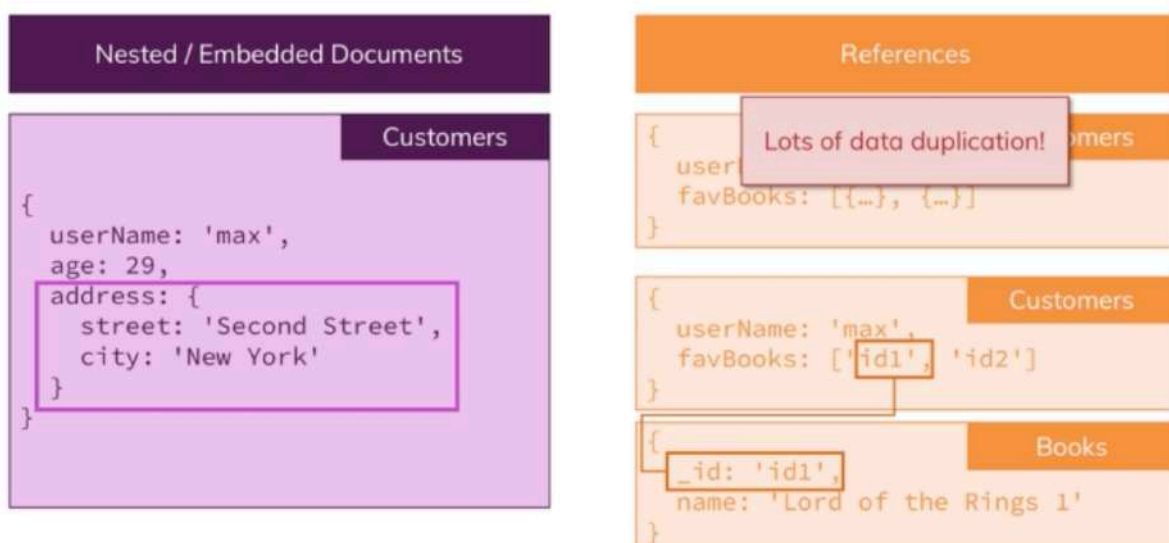
What's NoSQL?



by memory

- In NoSQL, it would be pretty normal to have something like this.
- here are 3 collections and we have some duplicate data in there.
- instead of just matching by ID as you do in the SQL world, here you can also depict a relation by embedding data into other documents. you could embed the ID which points at another document, so that you still have to merge 2 documents manually and you will have to do that pretty manually.
- but you can also just take the information that is important for you in the context of another document. in here, some user data for the orders and you copy that into the orders. and then you have that data whenever you fetch all orders without you having to fetch all orders, then look for the fitting users and fetch them too. this is part of what makes NoSQL so fast and efficient.

Relations - Options



by memory

- here's the embedded document example where the address is part of our customer document. so instead of having 2 collections, customers and address and then matching by ID, here we put the address right into the

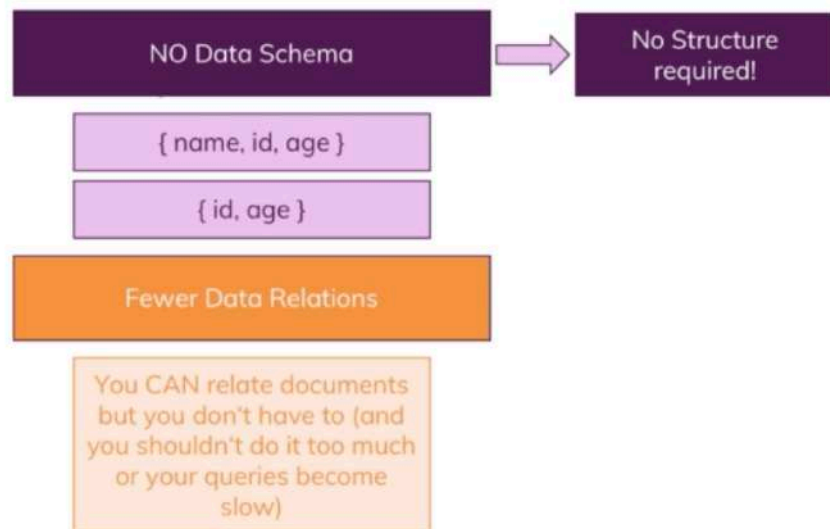
customer.

- there also are cases where you would have a lot of data duplication and where you need to work with that data a lot. and hence it would change a lot and you would have to manually update it in all duplicate places where using embedded documents is not ideal. so for example if you have some favorite books for every customer, you would have lots of data duplication because a lot of customers might have the same favorite books and these books might change a lot. you would have to go to all customers who have these books as favorites and update the entries for each customer.

- so it would be easier to go with 2 collections and only store the references to the books in a customers documents and then manually merge that with the books which are managed in a different collection

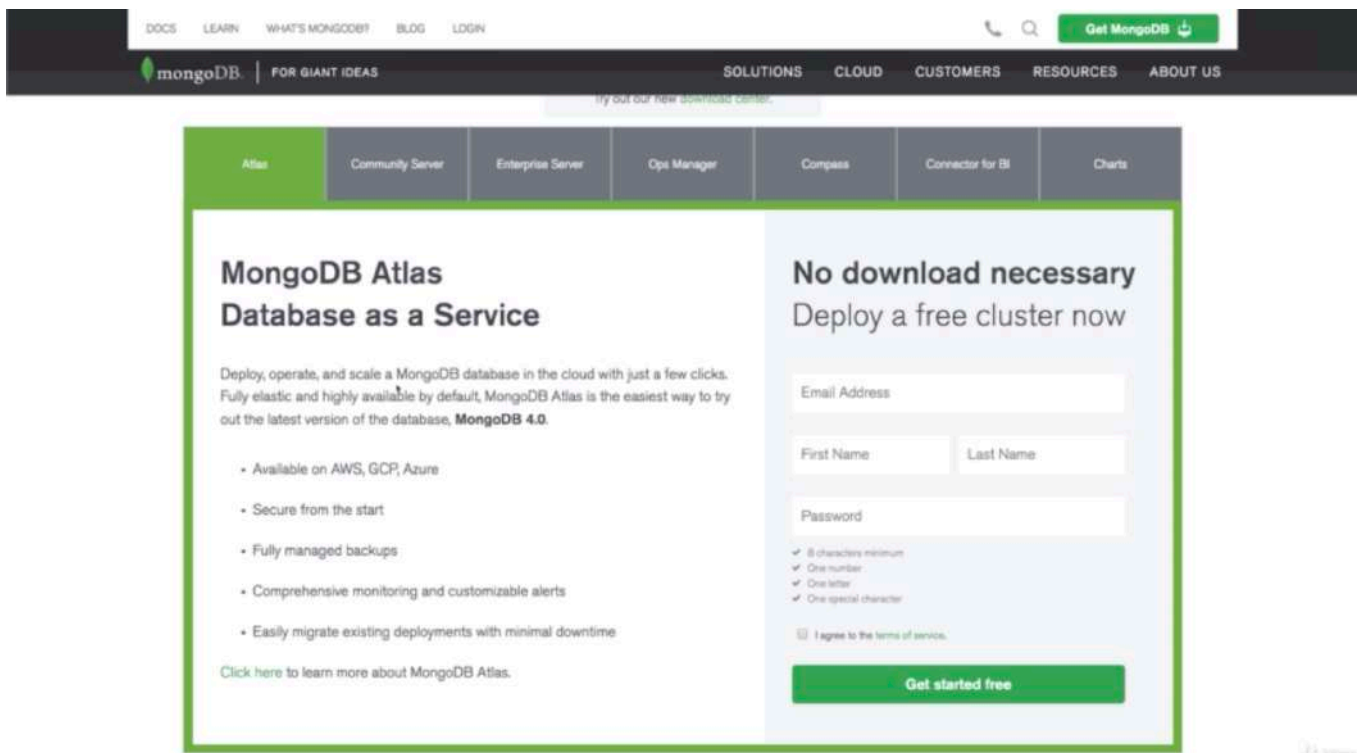


NoSQL Characteristics



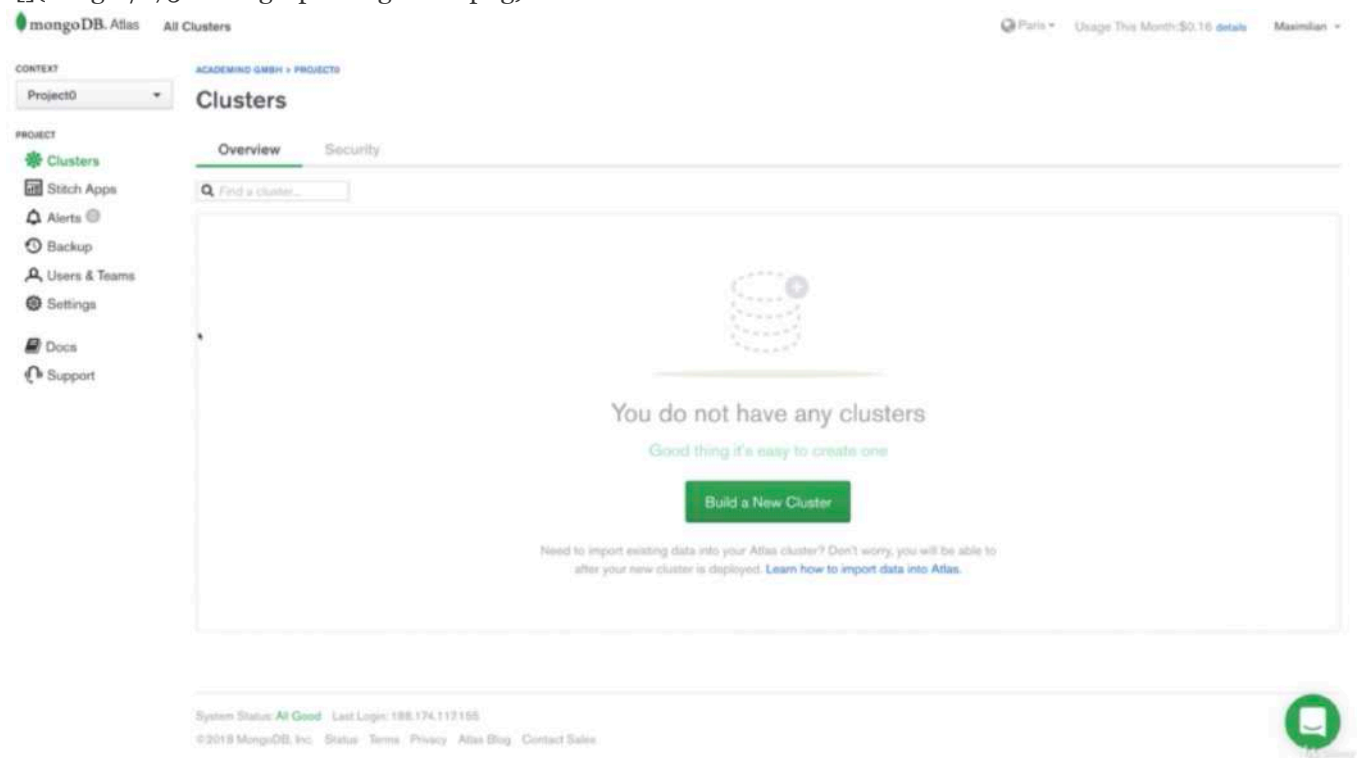
Academind

* Chapter 175: Setting Up MongoDB

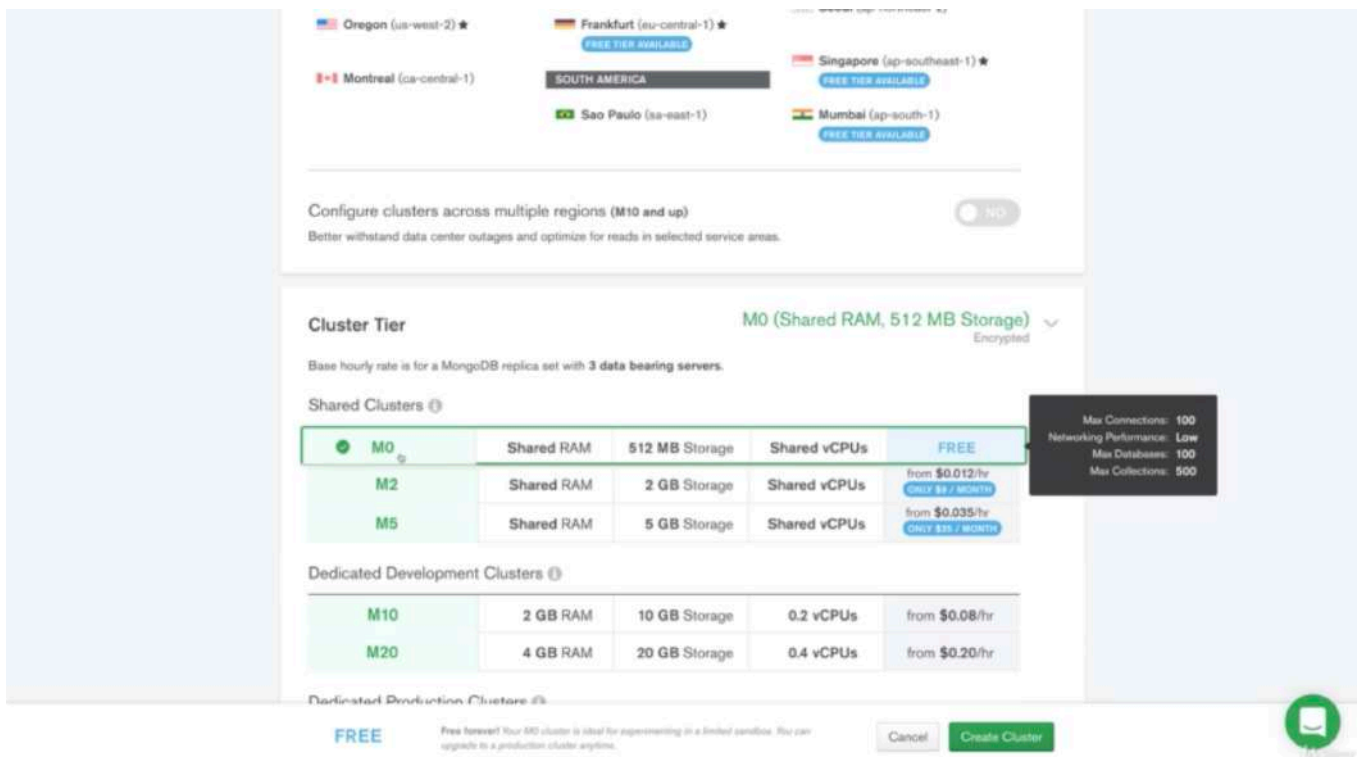


- i would go for a cloud solution because it's the more realistic environment. we would use for deployment and it's really easy to set up and it's free.

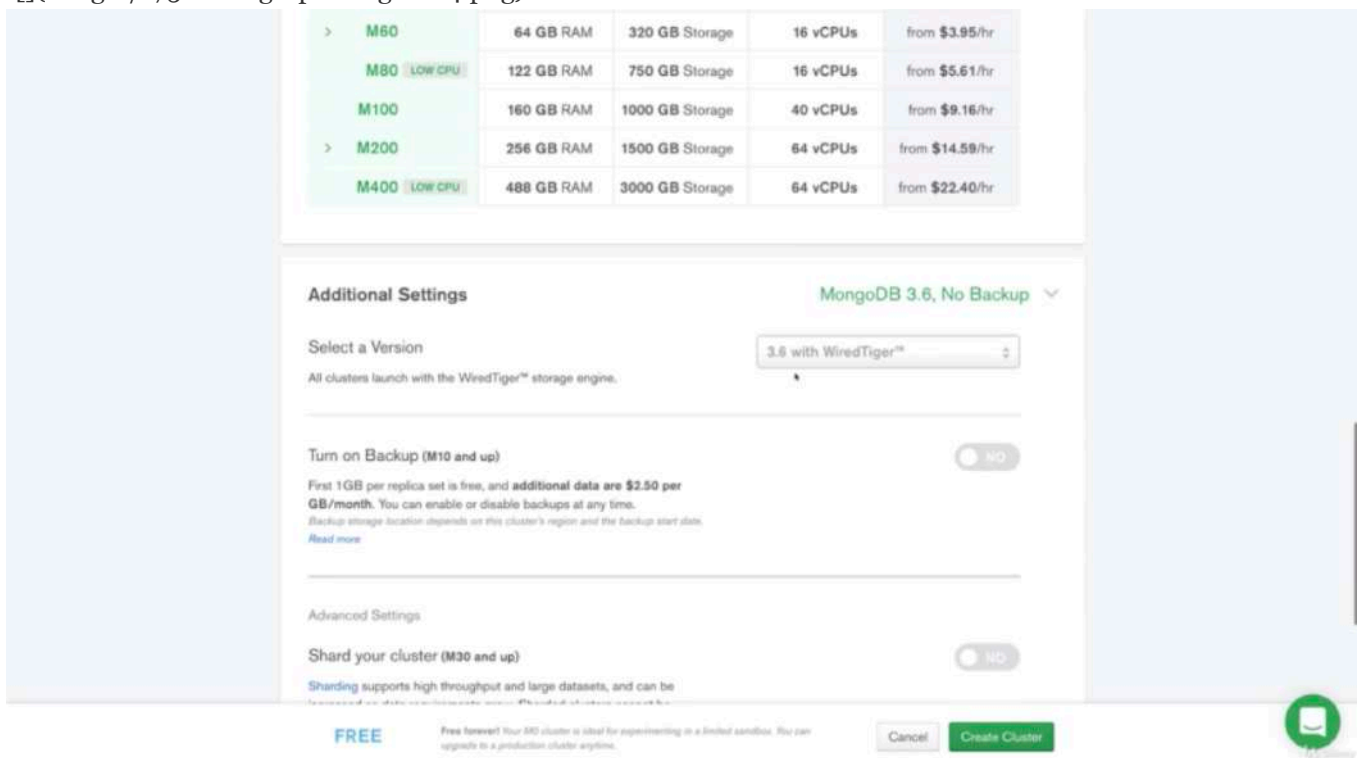
- and that will be 'Atlas'. so MongoDB Atlas.



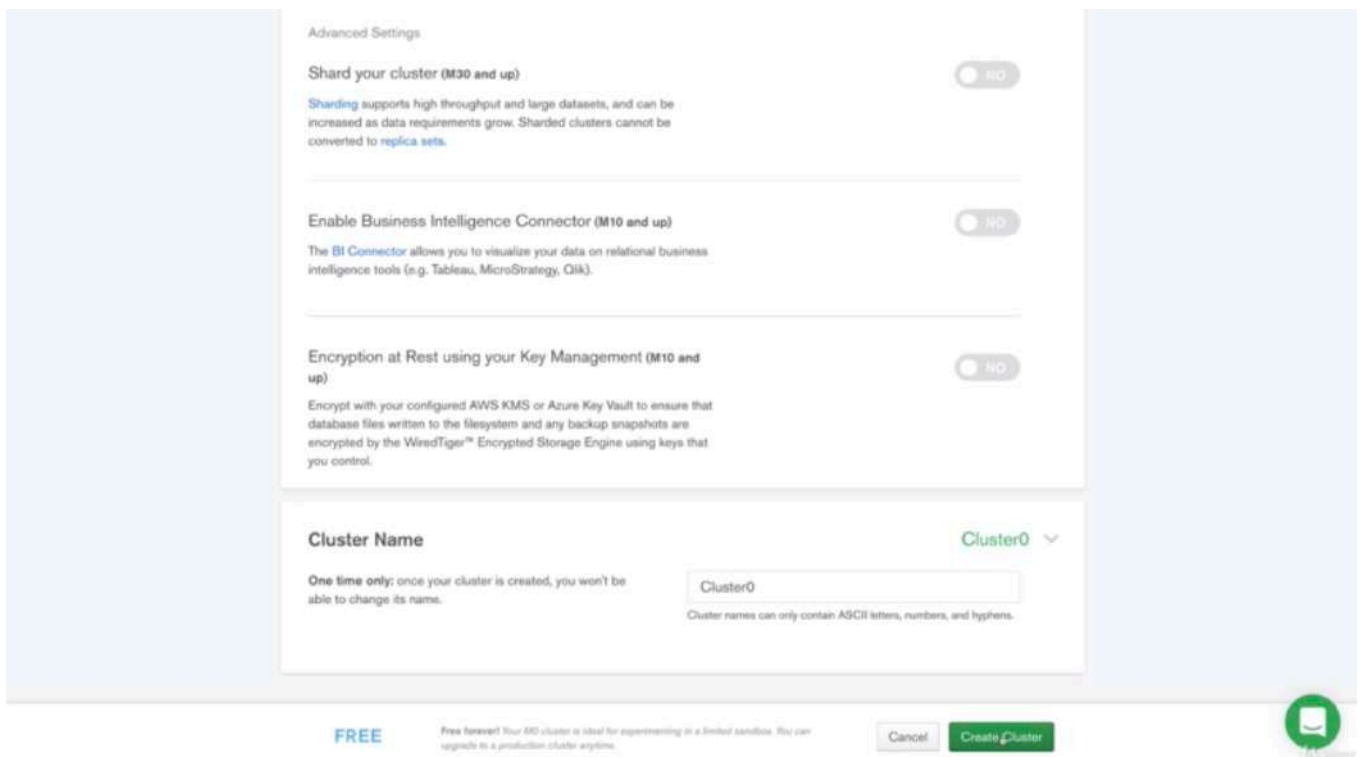
- once you did sign up, you should end up on a page that looks something like this.



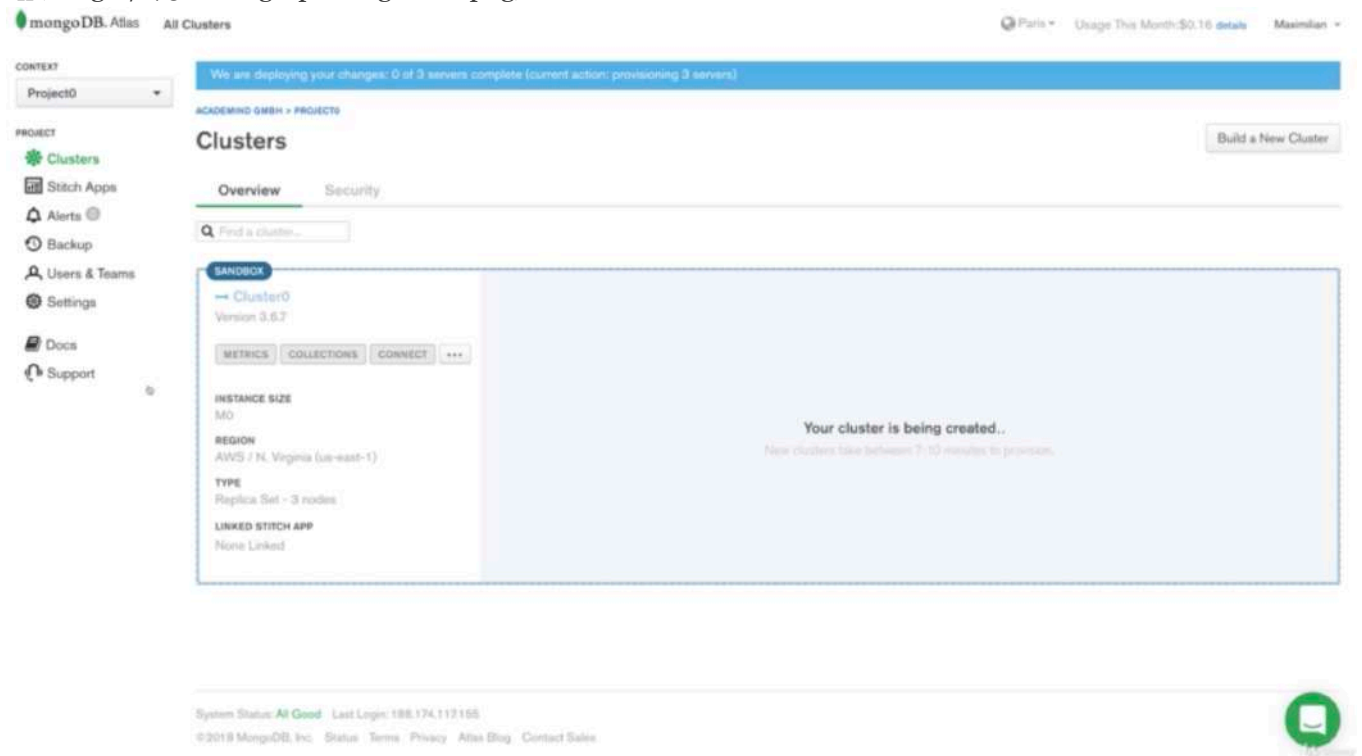
- maybe you need to create a new project first, give it any name you want and then you should be on that.
 - or you can build a new cluster. you might end up in that wizard right from the start.
 - you can generally leave all the default settings. make sure that you choose a regime where free tier is available.
-



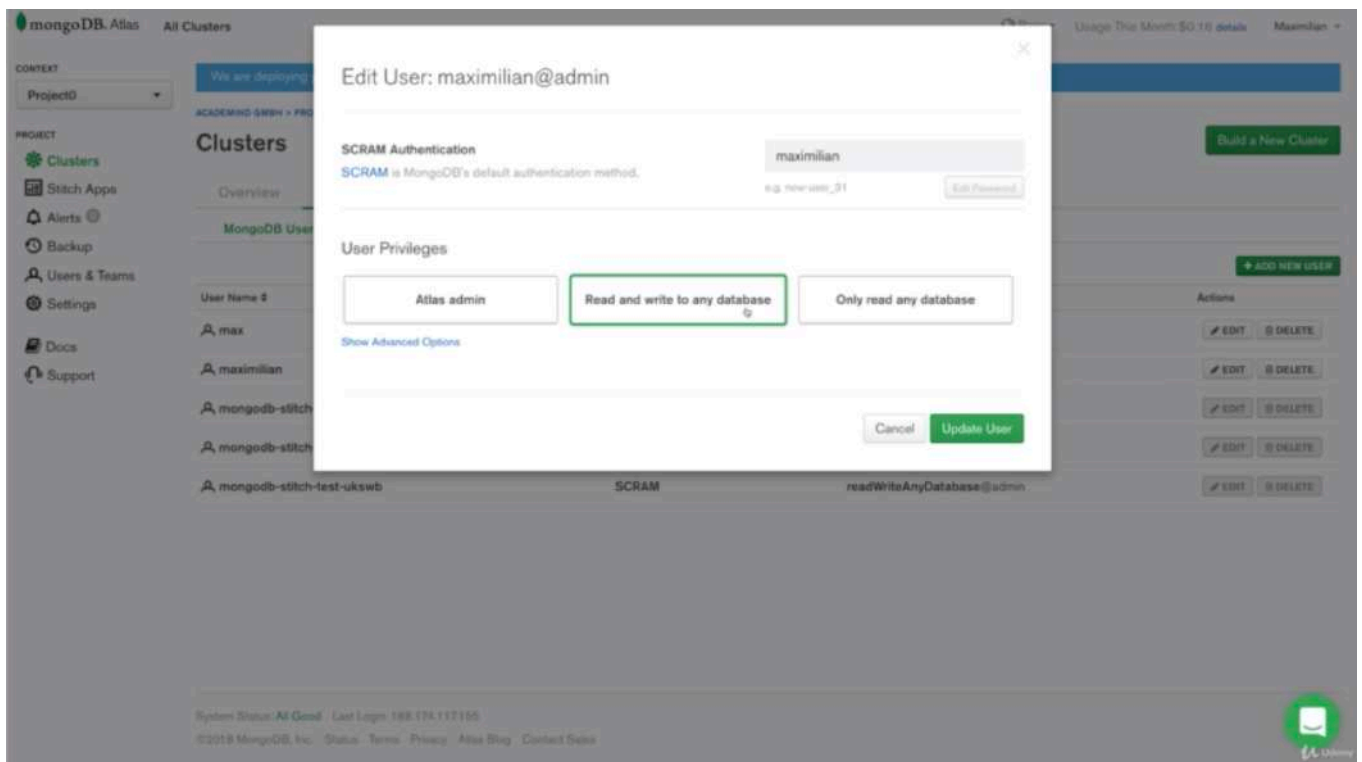
- and then you choose the cluster tier and there you should use the free one. Mo.
-



- Under additional settings, you can leave all the defaults

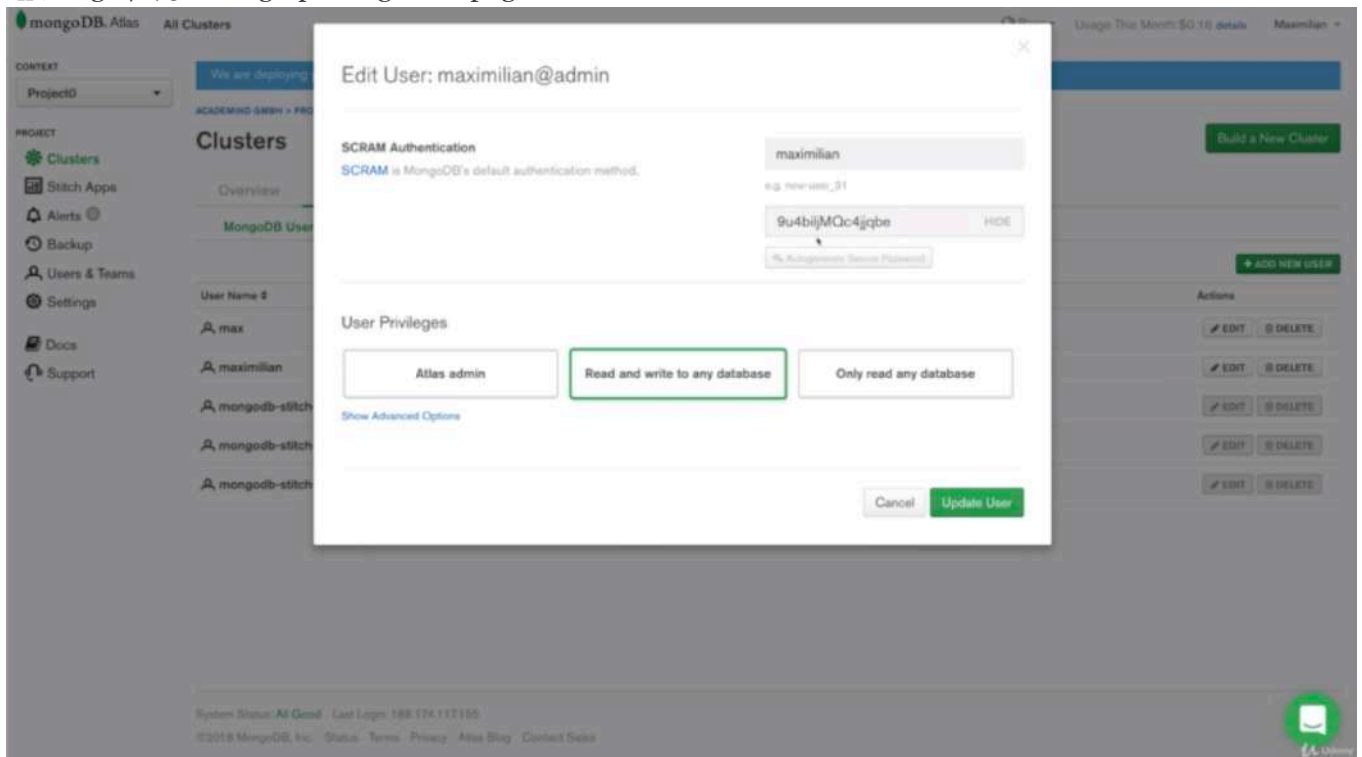


- you can change the cluster name and then you can click create cluster



- while this is getting set up, you can already click on security and make sure that you add at least one new user which has 'Read and write to any database'.

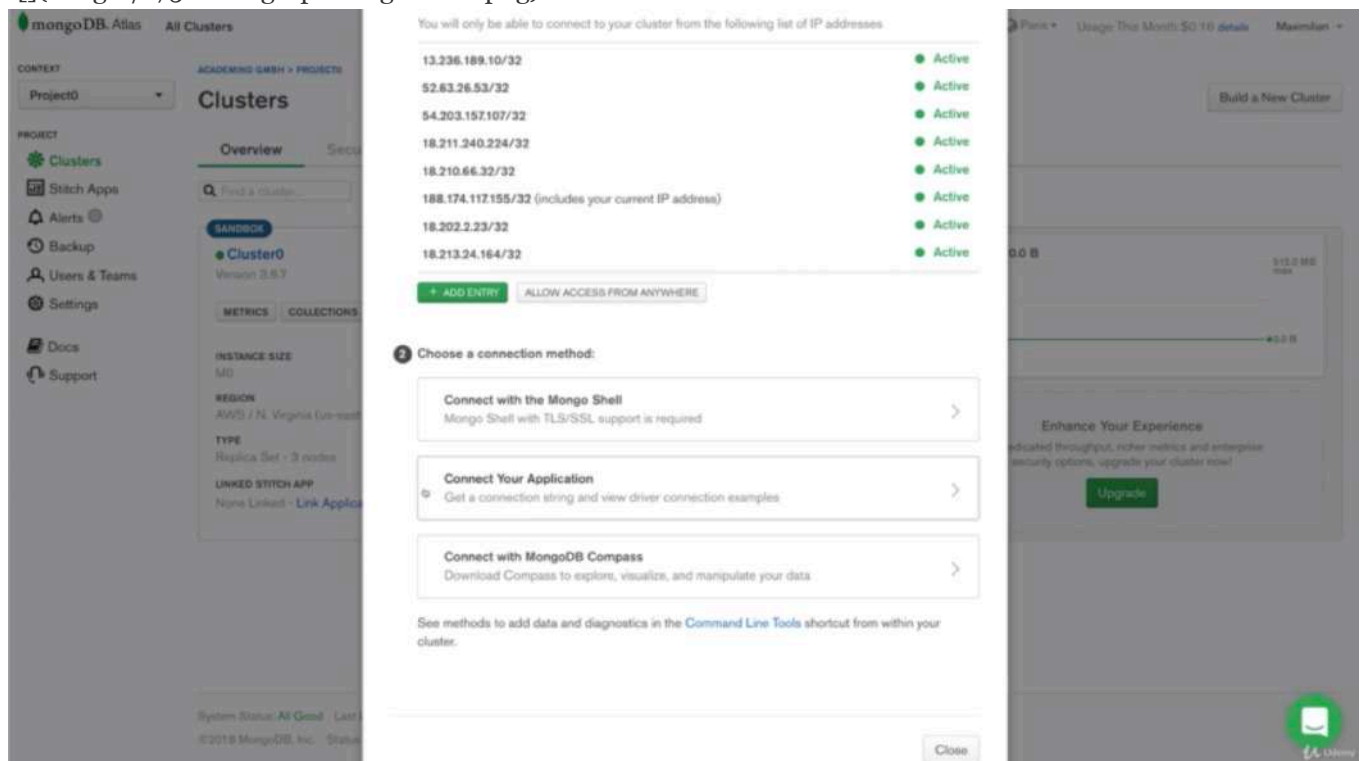
- you can turn it into 'Atlas admin' but more realistic is this 'Read and write to any database' because this will later be the role which our node.js application assumes which should be able to read and write our database but not to administrate them because we will not do database administration through node.js, that would be something our database admin does.



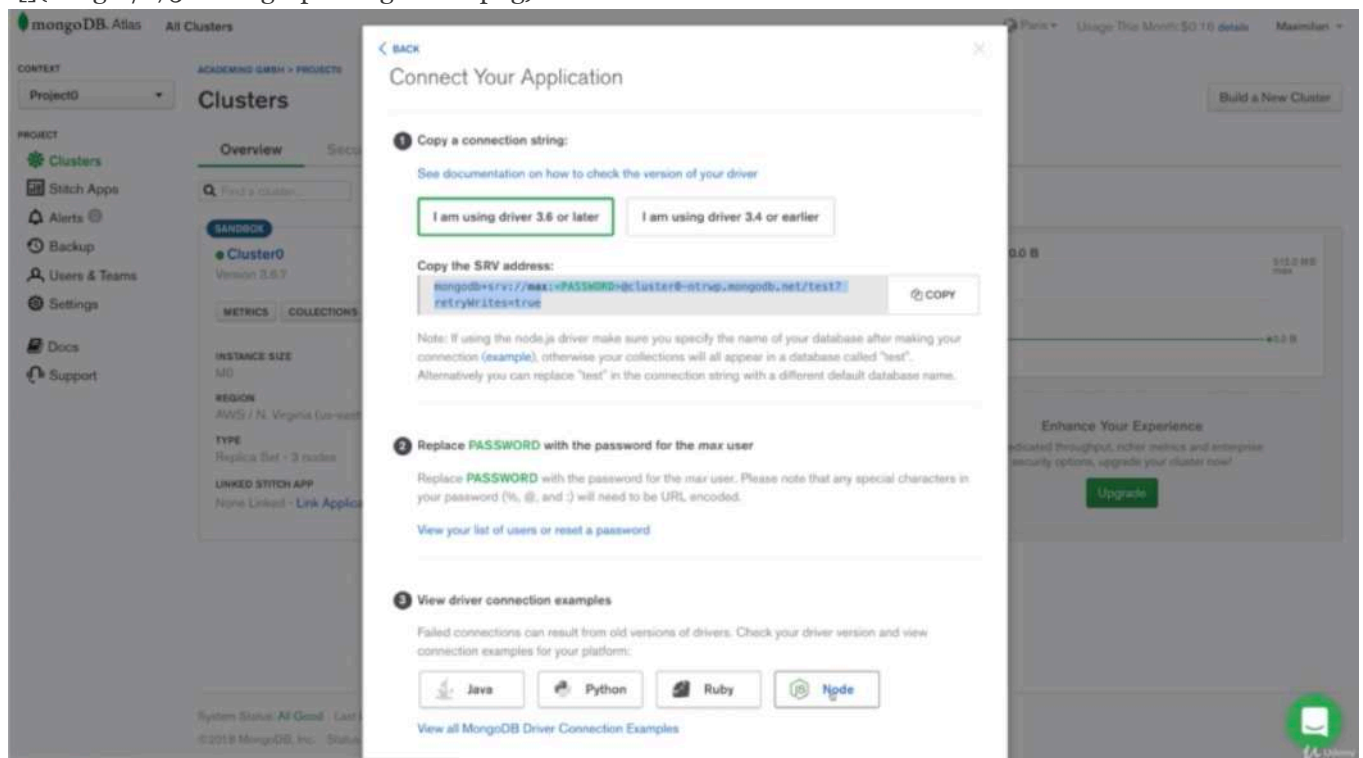
- you can also sign your own password or auto-generate a secure one. make sure you save that because you will need it later.

- and also make sure you have a look at IP whitelist. there you see all the IP addresses that are allowed to connect to your MongoDB server. these server will probably be seen less for you.

- one thing you should do here is you should add a new IP address and add your current IP address since the node app runs locally on your machine. your node app will have this IP address. later when you deploy the node app, this should use the IP address of your server where you deploy it to.
- here you can use your local one so that you can connect and your node app can connect to that server. that's a good security feature because this makes sure that no unauthorised access can happen to your database. so your database is now locked down both from a IP perspective but also from a user perspective.



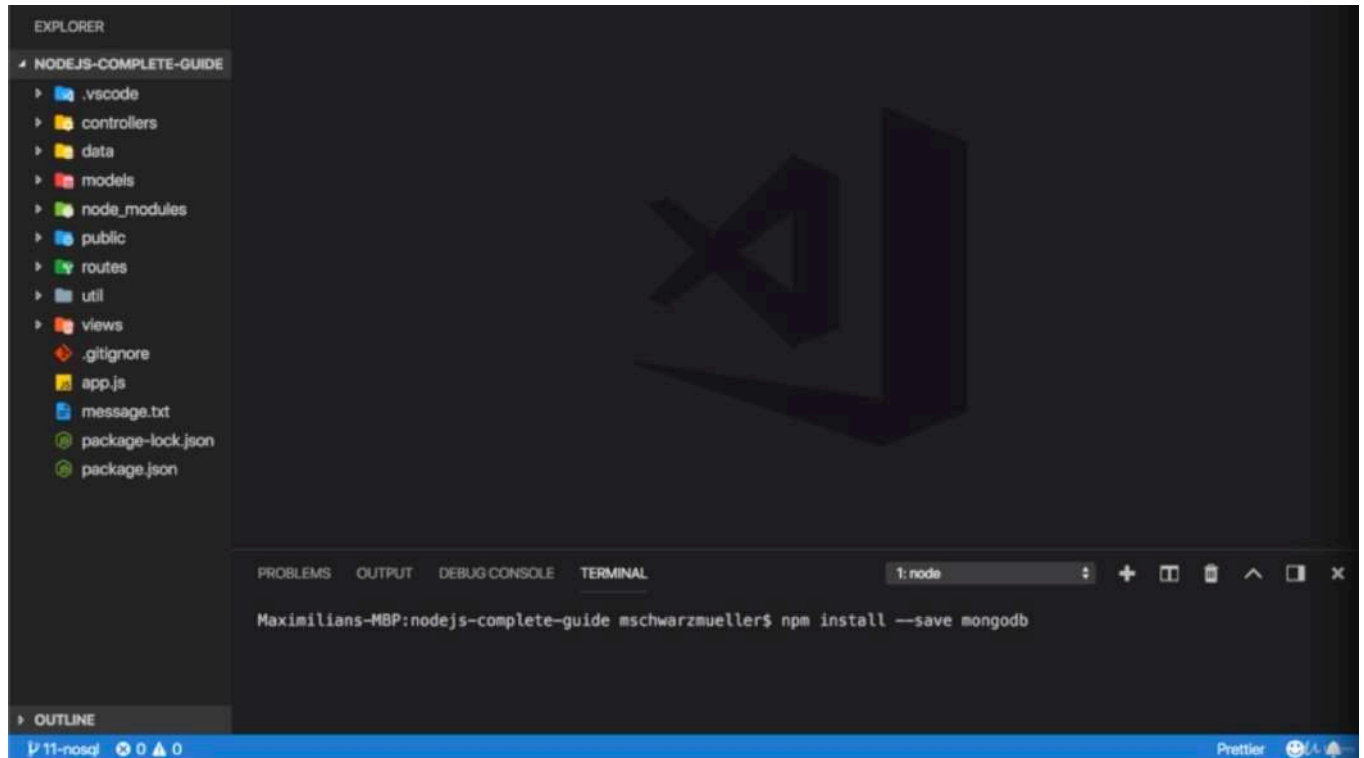
- now we can connect to our MongoDB server from inside our node app and we can click on 'connect' here and choose a connection method which in our case will be an application. so 'Connect Your Application'



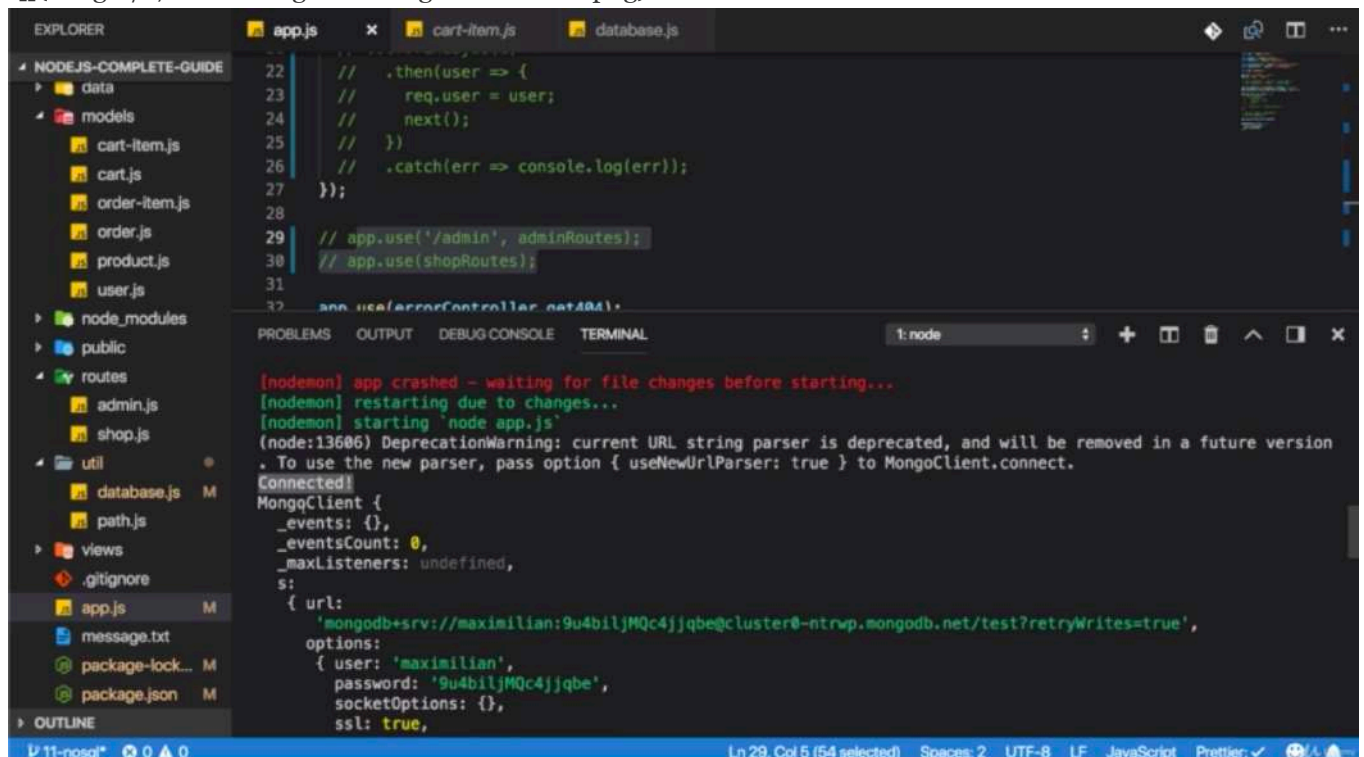
- here we can check i'm using this driver and we get this URL which we will need soon. but first of all, we need to install the MongoDB driver, in our case for Node.js.

* Chapter 176: Installing The MongoDB Driver

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



- we can start using it and we can start using it in the first file that gets executed when we bring up our server which would be the app.js file.



- we see 'connected' and then we see this 'MongoClient' object which we got with some details about the

connection and this is in the end the object which we will be able to interact with, to work with to create data in our database for example.

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database')
10
11 const app = express();
12
13 app.set('view engine', 'ejs');
14 app.set('views', 'views');
15
16 //const adminRoutes = require('./routes/admin');
17 //const shopRoutes = require('./routes/shop');
18
19 app.use(bodyParser.urlencoded({ extended: false }));
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use((req, res, next) => {
23   /*
24     User.findByPk(1)
25     .then(user => {
26       req.user = user
27       next()
28     })
29     .catch(err => console.log(err))
30   */
31 })
32
33 //app.use('/admin', adminRoutes);
34 //app.use(shopRoutes);
35
36 app.use(errorController.get404);
37
38 /**we hve to pass a callback so function that will get executed
39  * once we connect it
40  * and here i will get access to the client object
41  */
42 mongoConnect((client) => {
43   console.log(client)
44   app.listen(3000)
45 })
```

```
1 //./util/database.js
2
3 const mongodb = require('mongodb')
4 /**we can extract a Mongo Client constructor by simply accessing mongodb */
5 const MongoClient = mongodb.MongoClient
6
7 const mongoConnect = (callback) => {
8   /**now we can use that client to connect to our mongodb database.
```

```

9  * this is all we need to do to create a connection to MongoDB
10 *
11 * in 'connect()' takes a URL to connect and that URL is what you are have in the connect
    modal on the MongoDB Atlas cluster page
12 * the important thing is that you need to make sure you are using the right user
13 * and in my case that should be 'maximilian' the user you created in your MongoDB cluster
    under security
14 * and the fitting password filled when you add user.
15 */
16 MongoClient
17   .connect(
18     'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/test?
    retryWrites=true'
19   )
20   .then(client => {
21     console.log('Connected!')
22     callback(client)
23   })
24   .catch(err => console.log(err))
25 }
26
27 module.exports = mongoConnect
28

```

* Chapter 177: Creating The Database Connection

1. update
 - ./models/product.js
 - ./util/database.js
 - app.js

```

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

```

```

25     .then(user => {
26         req.user = user
27         next()
28     })
29     .catch(err => console.log(err))
30 */
31 })
32
33 //app.use('/admin', adminRoutes);
34 //app.use(shopRoutes);
35
36 app.use(errorController.get404);
37
38 mongoConnect((client) => {
39     console.log(client)
40     app.listen(3000)
41 })

```

```

1  //./models/product.js
2
3  const mongoConnect = require('./util/database')
4
5  class Product {
6      constructor(title, price, description, imageUrl){
7          this.title = title
8          this.price = price
9          this.description = description
10         this.imageUrl = imageUrl
11     }
12     /** this is a function which can be executed on this class
13     * and in here i now wanna connect to MongoDB and save my product.
14     * now to do that, to be able to connect, i will need to import MongoDB or mongoConnect
15     *
16     * so i simply import the function i created inside './util/database'
17     * where you pass a callback to,
18     * where we connect to MongoDB inside.
19     */
20     save(){
21
22     }
23 }
24
25 const Product = sequelize.define('product', {
26     id: {
27         type: Sequelize.INTEGER,
28         autoIncrement: true,
29         allowNull: false,
30         primaryKey: true
31     },
32     title: Sequelize.STRING,
33     price: {
34         type: Sequelize.DOUBLE,
35         allowNull: false
36     },
37     imageUrl: {
38         type: Sequelize.STRING,
39         allowNull: false

```

```

40   },
41   description: {
42     type: Sequelize.STRING,
43     allowNull: false
44   }
45 });
46
47 module.exports = Product;
48
49

```

```

1  //./util/database.js
2
3  const mongodb = require('mongodb')
4
5  const MongoClient = mongodb.MongoClient
6
7  /**we execute the callback and return the connected client
8   * so that we can interact with it
9   * however if you would this,
10  * we would have to connect to MongoDB for every operation we do
11  * and we would not even disconnect thereafter
12  * so this is not good way of connecting to MongoDB since we will wanna connect and interact
   with it from different places in our app.
13  *
14  * so it would be better if we could manage one connection in our database
15  * and then simply return access to the client which we set up once from there
16  * or to the different place in our app that need access
17  */
18  const mongoConnect = (callback) => {
19    MongoClient
20      .connect(
21        'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/test?
   retryWrites=true'
22      )
23      .then(client => {
24        console.log('Connected!')
25        callback(client)
26      })
27      .catch(err => console.log(err))
28  }
29
30  module.exports = mongoConnect
31

```

* Chapter 178: Finishing The Database Connection

1. update
 - ./util/database.js
 - app.js
 - ./models/product.js

```

1  //app.js
2
3  const path = require('path');

```

```

4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database')
10
11 const app = express();
12
13 app.set('view engine', 'ejs');
14 app.set('views', 'views');
15
16 const adminRoutes = require('./routes/admin');
17 //const shopRoutes = require('./routes/shop');
18
19 app.use(bodyParser.urlencoded({ extended: false }));
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use((req, res, next) => {
23   /*
24     User.findByPk(1)
25     .then(user => {
26       req.user = user
27       next()
28     })
29     .catch(err => console.log(err))
30   */
31 })
32
33 //app.use('/admin', adminRoutes);
34 //app.use(shopRoutes);
35
36 app.use(errorController.get404);
37
38 mongoConnect(() => {
39   app.listen(3000)
40 })

```

```

1 //./models/product.js
2
3 /**Using 'getDb' means that i can now call this function
4  * to get access to my database
5  * and therefore i can use it to interact with the database.
6  */
7 const getDb = require('./util/database').getDb
8
9 class Product {
10   constructor(title, price, description, imageUrl){
11     this.title = title
12     this.price = price
13     this.description = description
14     this.imageUrl = imageUrl
15   }
16   save(){
17
18   }
19 }

```



```

20
21 const Product = sequelize.define('product', {
22   id: {
23     type: Sequelize.INTEGER,
24     autoIncrement: true,
25     allowNull: false,
26     primaryKey: true
27   },
28   title: Sequelize.STRING,
29   price: {
30     type: Sequelize.DOUBLE,
31     allowNull: false
32   },
33   imageUrl: {
34     type: Sequelize.STRING,
35     allowNull: false
36   },
37   description: {
38     type: Sequelize.STRING,
39     allowNull: false
40   }
41 });
42
43 module.exports = Product;
44
45

```

```

1  //./util/database.js
2
3  const mongodb = require('mongodb')
4  const MongoClient = mongodb.MongoClient
5
6  /**instead i will add a variable _db
7   * underscore '_' is only here to signal
8   * that this will only be used internally in this file
9   * initially '_db' is undefined.
10  */
11  let _db
12
13  /**i'm exporting 2 method
14   * one for connecting and then storing the connection to the database
15   * this will keep on running
16  */
17  const mongoConnect = (callback) => {
18    MongoClient
19      .connect(
20        /**what we will do is we will connect to the 'test' database by default
21         * because that is what we specify in our connection
22         string.'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/\\test\\'?
23         retryWrites=true'
24         * so we will connect to shop (replace 'test' to 'shop')
25         'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/\\shop\\'?
26         retryWrites=true'
27         * this will give us access to the shop database to which we automatically connect
28         *
29         * on the side note, unlike in SQL
30         * we never need to create that database or the tables, the collections in there

```

```

ahead of time
27      * it will be created on the fly
28      * when we first access it
29      * which is again fitting that flexibility theme MongoDB gives us
30      *
31      * In SQL, we had to prepare everything in advance, at least when not using
sequelize
32      * which also had to do that but it did it for us.
33      * we don't need to do anything. we are just telling MongoDB
34      * hey connect me to the shop database
35      * and if that database doesn't exist yet, MongoDB will create it as soon as we
start writing data to it.
36      */
37      'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
retryWrites=true'
38    )
39    .then(client => {
40      console.log('Connected!')
41      /**i will store access to the database here*/
42      _db = client.db()
43      callback(client)
44    })
45    .catch(err => {
46      console.log(err)
47      throw err
48    })
49 }
50
51 /**and i have one method where i return access to that connected database
52 * if it exists and mongoDB behind the scenes will manage this very elegantly with something
called 'connection pooling'
53 * where mongoDB will make sure it provides sufficient connections for multiple simultaneous
interactions with the database
54 *
55 * so this is really a good pattern we should follow
56 */
57 const getDb = () => {
58   if(_db){
59     return _db
60   }
61   throw 'No database found!'
62 }
63
64 exports.mongoConnect = mongoConnect
65 exports.getDb = getDb

```

* Chapter 179: Using The Database Connection

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

This is a test

Image URL

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

Price

12.99

Description

fdsfadsf

Add Product

* Chapter 180: Creating Products

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

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

This is a test

Image URL

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

Price

12.99

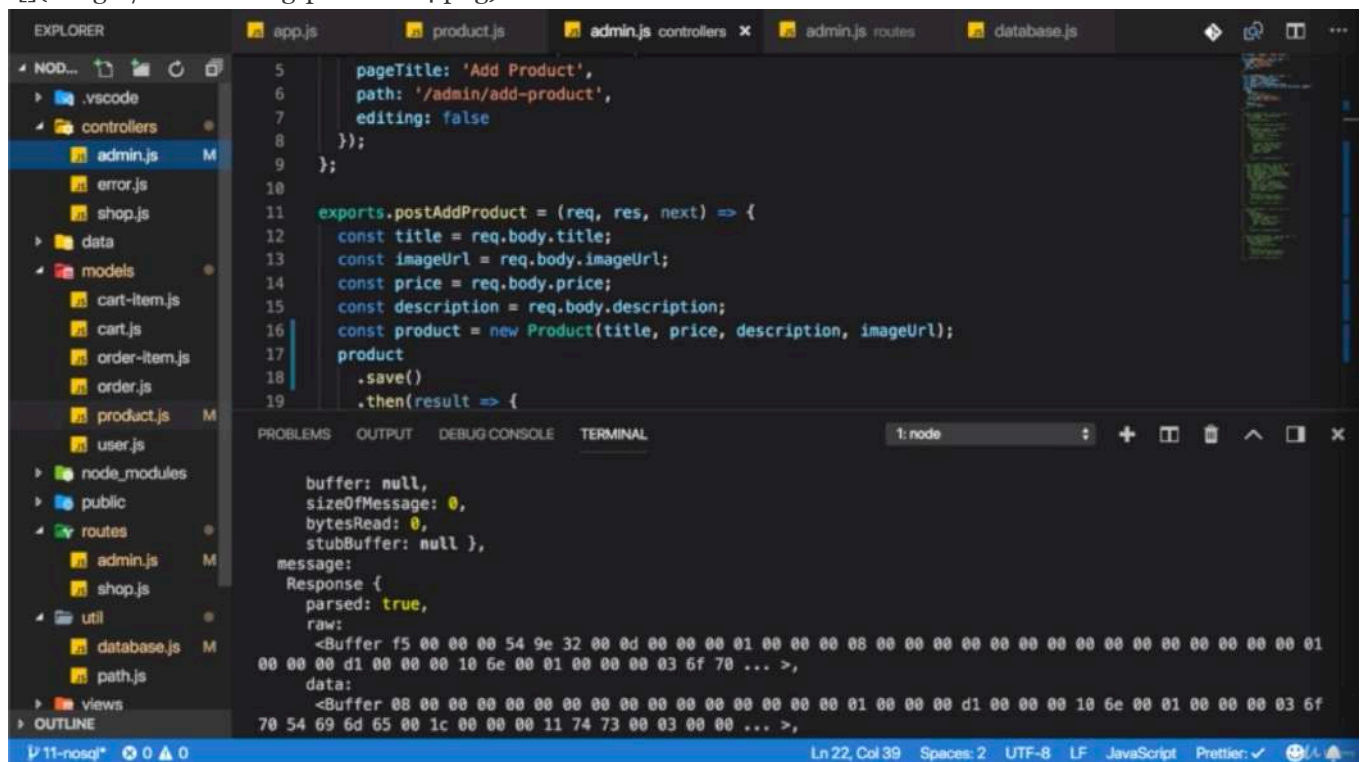
Description

fdsfadsf

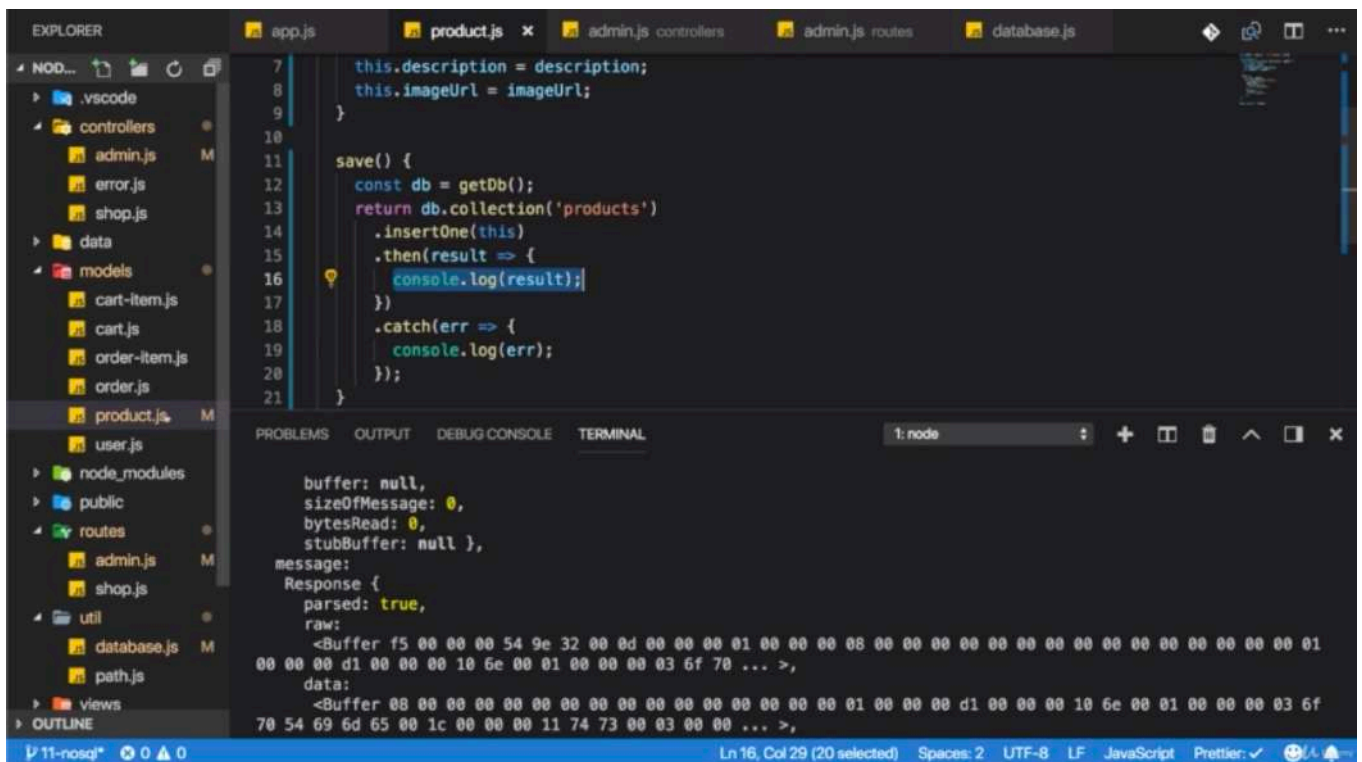
Add Product

Page Not Found!

- go back to the form and resubmit that, i'm redirected to a page which is not found because i commented out all the shop pages.



The screenshot shows a VS Code editor with a project structure on the left and a terminal window at the bottom. The project structure includes files like `app.js`, `product.js`, `admin.js`, `controllers`, `admin.js routes`, and `database.js`. The `admin.js` file is open in the editor, showing a `postAddProduct` function. The terminal window displays a Node.js error message: `buffer: null, sizeofMessage: 0, bytesRead: 0, stubBuffer: null }, message: Response { parsed: true, raw: <Buffer f5 00 00 00 54 9e 32 00 0d 00 00 00 01 00 00 08 00 00 00 00 00 00 00 00 01 00 00 d1 00 00 00 10 6e 00 01 00 00 00 03 6f 70 ... >, data: <Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 d1 00 00 00 10 6e 00 01 00 00 00 03 6f 70 54 69 6d 65 00 1c 00 00 00 11 74 73 00 03 00 00 ... >, }`. The status bar at the bottom indicates the file is `app.js` at line 22, column 39, with 2 spaces, UTF-8 encoding, and LF line endings.

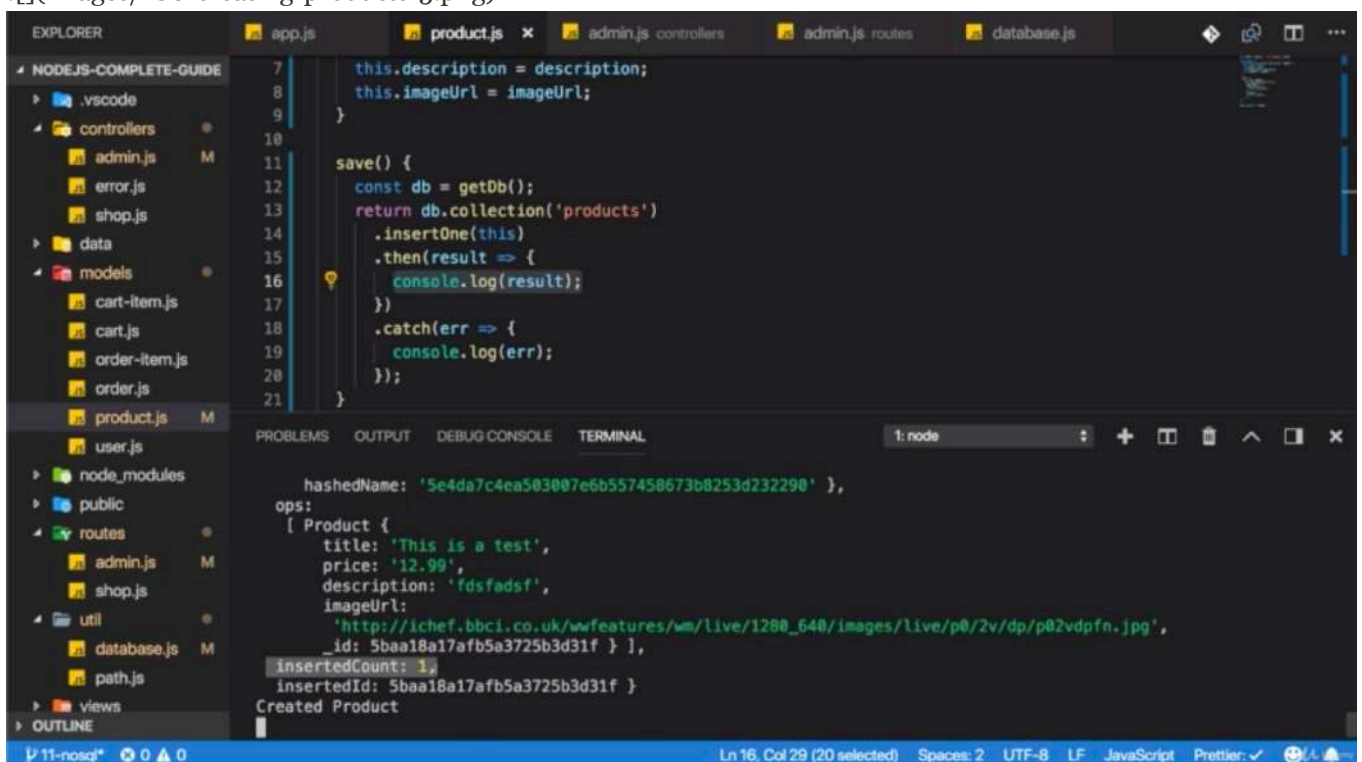


```
7   this.description = description;
8   this.imageUrl = imageUrl;
9   }
10
11  save() {
12    const db = getDb();
13    return db.collection('products')
14      .insertOne(this)
15      .then(result => {
16        console.log(result);
17      })
18      .catch(err => {
19        console.log(err);
20      });
21  }
```

```
buffer: null,
sizeOfMessage: 0,
bytesRead: 0,
stubBuffer: null },
message:
  Response {
    parsed: true,
    raw:
      <Buffer f5 00 00 00 54 9e 32 00 0d 00 00 00 01 00 00 08 00 00 00 00 00 00 00 00 00 00 01
00 00 00 d1 00 00 00 10 6e 00 01 00 00 00 03 6f 70 ... >,
    data:
      <Buffer 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 d1 00 00 00 10 6e 00 01 00 00 00 03 6f
70 54 69 6d 65 00 1c 00 00 00 11 74 73 00 03 00 00 ... >,
```

- but if we go back to our server side console, we see something interesting.

- we see that this here has to be the output of this console log line in the ./models/product.js where i print the result of the insert operation and there we see a lot of data about that operation.



```
7   this.description = description;
8   this.imageUrl = imageUrl;
9   }
10
11  save() {
12    const db = getDb();
13    return db.collection('products')
14      .insertOne(this)
15      .then(result => {
16        console.log(result);
17      })
18      .catch(err => {
19        console.log(err);
20      });
21  }
```

```
hashedName: '5e4da7c4ea503007e6b557450673b8253d232290' },
ops:
  [ Product {
    title: 'This is a test',
    price: '12.99',
    description: 'fdsfadsf',
    imageUrl:
      'http://ichef.bbci.co.uk/ww/features/wm/live/1280_648/images/live/p0/2v/dp/p02vdpfn.jpg',
    _id: 5baa18a17afb5a3725b3d31f } ],
insertedCount: 1,
insertedId: 5baa18a17afb5a3725b3d31f }
Created Product
```

- if we scroll down to the bottom, we see one document was inserted. 'insertedCount: 1'. it received an ID and such an ID is managed automatically by MongoDB because every document needs to have such a _id. this is must-have and MongoDB creates it on the fly automatically if the object you entered doesn't have it. so we will use that auto-generated ID

- and then you see the data which we entered also was stored.

- while we can't look into database yet because we are not fetching anything, we see that our insert one operation was successful and did successfully add a product into the collection.

```
1 //app.js
```

```

2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database').mongoConnect;
10
11 const app = express();
12
13 app.set('view engine', 'ejs');
14 app.set('views', 'views');
15
16 const adminRoutes = require('./routes/admin');
17 const shopRoutes = require('./routes/shop');
18
19 app.use(bodyParser.urlencoded({ extended: false }));
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use((req, res, next) => {
23   // User.findById(1)
24   //   .then(user => {
25   //     req.user = user;
26   //     next();
27   //   })
28   //   .catch(err => console.log(err));
29   next();
30 });
31
32 app.use('/admin', adminRoutes);
33 app.use(shopRoutes);
34
35 app.use(errorController.get404);
36
37 mongoConnect(() => {
38   app.listen(3000);
39 });
40

```

```

1 //./models/product.js
2
3 /**we are importing something
4  * which allows us to get access to the database connection we set up initially
5  * when starting our server which now is a concept that we can reuse
6  */
7 const getDb = require('./util/database').getDb
8
9 class Product {
10   constructor(title, price, description, imageUrl){
11     this.title = title
12     this.price = price
13     this.description = description
14     this.imageUrl = imageUrl
15   }
16   save(){
17     const db = getDb()

```

```

18  /**you can call 'collection()'
19  * to tell mongoDB into which collection you wanna insert something
20  * or with which collection you wanna work
21  * remember in MongoDB, you have database, collections, documents
22  *
23  * we have database connection here
24  * so the next level is a collection
25  * we can connect to any collection and just as with the database,
26  * if it doesn't exist yet, it will be created the first time you insert data.
27  *
28  * on that collection, we can execute a couple of MongoDB commands or operations
29  * if you wanna insert data,
30  * you can do this with 'insertOne()'
31  * or if it's multiple documents at once, 'insertMany()'
32  * which takes an array of javascript objects you wanna insert
33  */
34
35  /**here our case, it's the Product which we wanna insert
36  * so we could just say 'this'
37  * and see how that works
38  */
39  return db.collection('products')
40    .insertOne(this)
41    .then(result => {
42      console.log(result)
43    })
44    .catch(err => {
45      console.log(err)
46    })
47  }
48 }
49
50 module.exports = Product;
51
52

```

```

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(title, price, description, imageUrl);
19   product
20     .save()
21     .then(result => {

```

```

22     // console.log(result);
23     console.log('Created Product');
24     res.redirect('/admin/products');
25 }
26 .catch(err => {
27     console.log(err);
28 });
29 };
30
31 /*
32 exports.getEditProduct = (req, res, next) => {
33     const editMode = req.query.edit;
34     if (!editMode) {
35         return res.redirect('/');
36     }
37     const prodId = req.params.productId;
38     req.user
39         .getProducts({ where: { id: prodId } })
40     //Product.findByPk(prodId)
41     /**keep in mind we get back an array even if it only holds one element.
42      * so we got 'products'
43      * and therefore we know that one product,
44      * the one we are interested in will always be the first element
45      * so we have to store that separately in a new constant */
46     /*
47     .then(products => {
48         const product = products[0]
49         if (!product) {
50             return res.redirect('/');
51         }
52         res.render('admin/edit-product', {
53             pageTitle: 'Edit Product',
54             path: '/admin/edit-product',
55             editing: editMode,
56             product: product
57         });
58     })
59     .catch(err => console.log(err));
60 };
61
62 exports.postEditProduct = (req, res, next) => {
63     const prodId = req.body.productId;
64     const updatedTitle = req.body.title;
65     const updatedPrice = req.body.price;
66     const updatedImageUrl = req.body.imageUrl;
67     const updatedDesc = req.body.description;
68     Product.findByPk(prodId)
69         .then(product => {
70             product.title = updatedTitle;
71             product.price = updatedPrice;
72             product.description = updatedDesc;
73             product.imageUrl = updatedImageUrl;
74             return product.save();
75         })
76         .then(result => {
77             console.log('UPDATED PRODUCT!');

```



```

78     res.redirect('/admin/products');
79   })
80   .catch(err => console.log(err));
81 };
82
83 exports.getProducts = (req, res, next) => {
84   req.user
85     .getProducts()
86     .then(products => {
87       res.render('admin/products', {
88         prods: products,
89         pageTitle: 'Admin Products',
90         path: '/admin/products'
91       });
92     })
93     .catch(err => console.log(err));
94 };
95
96 exports.postDeleteProduct = (req, res, next) => {
97   const prodId = req.body.productId;
98   Product.findById(prodId)
99     .then(product => {
100     return product.destroy()
101   })
102     .then(result => {
103       console.log('DESTROYED PRODUCT')
104       res.redirect('/admin/products');
105     })
106     .catch(err => console.log(err))
107 };
108 */

```

```

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);

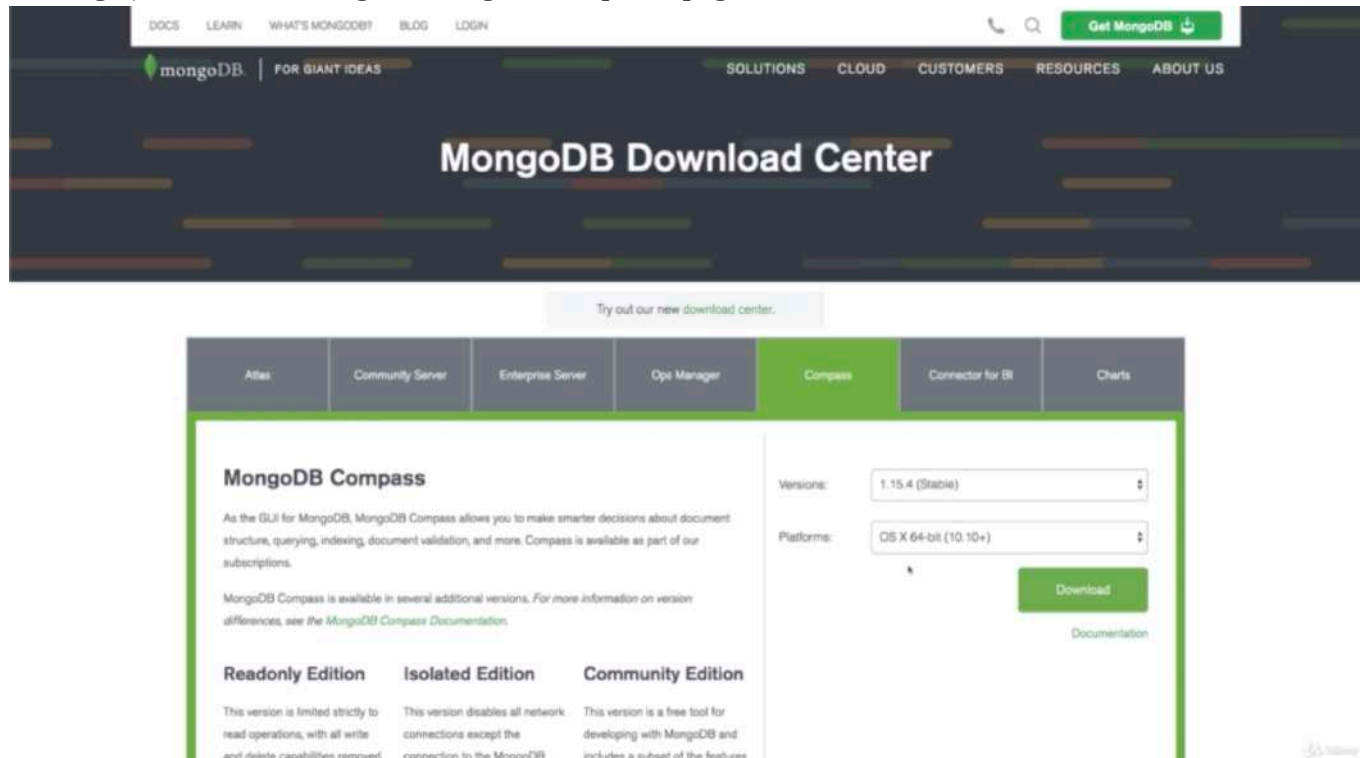
```

```

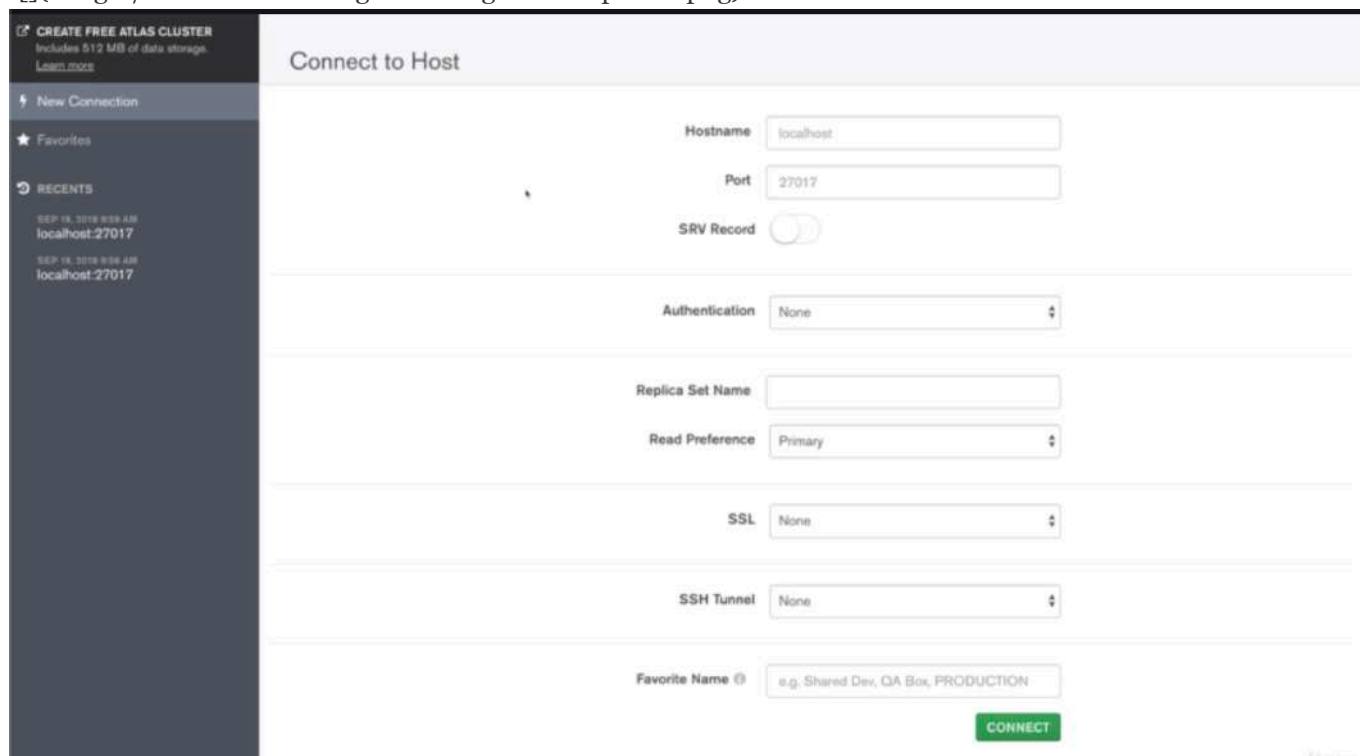
26
27 router.post('/delete-product', adminController.postDeleteProduct);
28 */
29
30 module.exports = router;

```

* Chapter 181: Understanding The MongoDB Compass



- you can choose that and you can download and install MongoDB Compass for free as well.



- once you get that installed, you can start your compass application on your machine. and this essentially is a tool that gives you a graphical user interface to connect to your database and to interact with it.

- once it did start up, you can connect to it.

The screenshot shows the MongoDB Atlas Clusters page. The top navigation bar includes the MongoDB Atlas logo, 'All Clusters', and a 'Build a New Cluster' button. The left sidebar contains navigation links for Clusters, Stitch Apps, Alerts, Backup, Users & Teams, Settings, Docs, and Support. The main content area is titled 'Clusters' and shows the 'Overview' tab for 'Cluster0'. The cluster is in the 'SANDBOX' environment, version 3.6.7, with an instance size of 'M0', region 'AWS / N. Virginia (us-east-1)', and type 'Replica Set - 3 nodes'. It is not linked to a stitch app. The 'Metrics' section shows 'Operations: R: 0 W: 0' and 'Logical Size: 0.0 B'. The 'Instance Size' section shows 'M0'. The 'Region' section shows 'AWS / N. Virginia (us-east-1)'. The 'Type' section shows 'Replica Set - 3 nodes'. The 'Linked Stitch App' section shows 'None Linked - Link Application'. The 'Connections' section shows '1'. The 'Last 6 Hours' section shows a graph. The 'Enhance Your Experience' section has an 'Upgrade' button. The bottom of the page shows the system status 'All Good' and the last login '188.174.112.155'.

1 Check the IP Whitelist

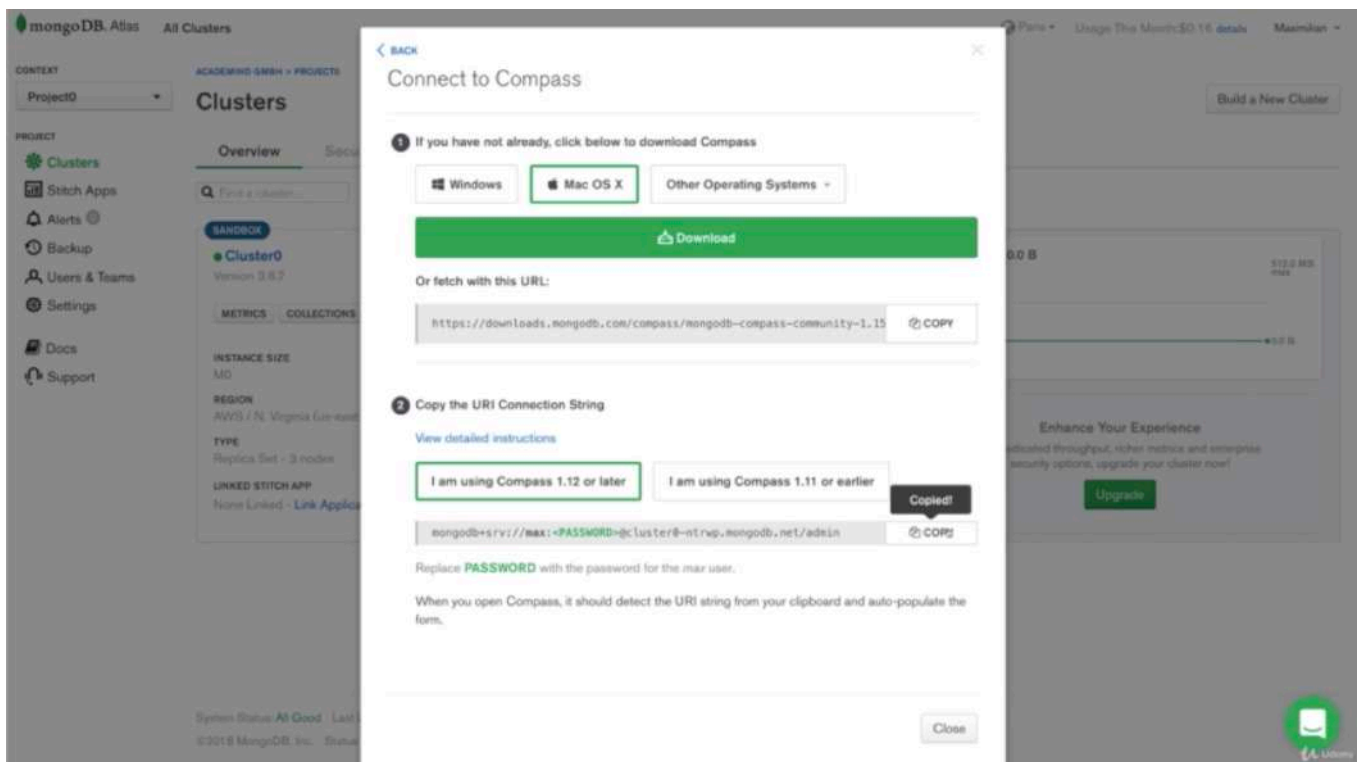
You will only be able to connect to your cluster from the following list of IP addresses

IP Address	Status
13.236.189.10/32	Active
52.63.26.53/32	Active
54.203.157.107/32	Active
18.211.240.224/32	Active
18.210.66.32/32	Active
188.174.112.155/32 (includes your current IP address)	Active
18.202.2.23/32	Active
18.213.24.164/32	Active

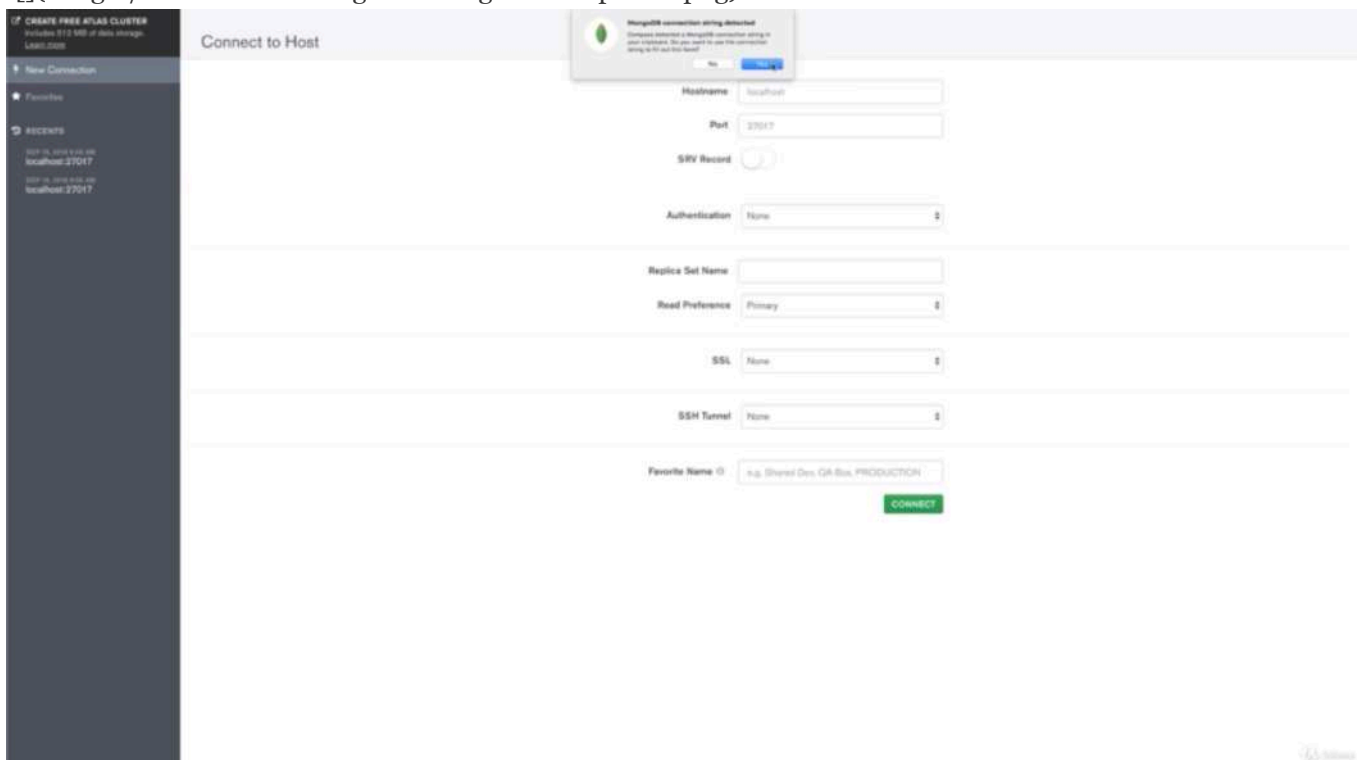
2 Choose a connection method:

- Connect with the Mongo Shell
Mongo Shell with TLS/SSL support is required
- Connect Your Application
Get a connection string and view driver connection examples
- Connect with MongoDB Compass
Download Compass to explore, visualize, and manipulate your data

See methods to add data and diagnostics in the [Command Line Tools](#) shortcut from within your cluster.



- to connect to it, go back to our MongoDB cluster and click 'connect' here and then click 'connect with MongoDB Compass' and choose your operating system and then copy that URL down there.



- the one cool thing is if you now quickly close compass and again you restart it after you copied that URL, it should tell you that it detected a connection string. and if you click yes, it will insert the most important pieces here.

Connect to Host

Hostname:

SRV Record: ☒

Authentication:

Username:

Password:

Authentication Database:

Replica Set Name:

Read Preference:

SSL:

SSH Tunnel:

Favorite Name:

CONNECT

- you still need to choose how you wanna connect, make sure that your username is correct and also enter the password for this user. all the defaults can be left as they are and you should be able to now connect to your database.

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

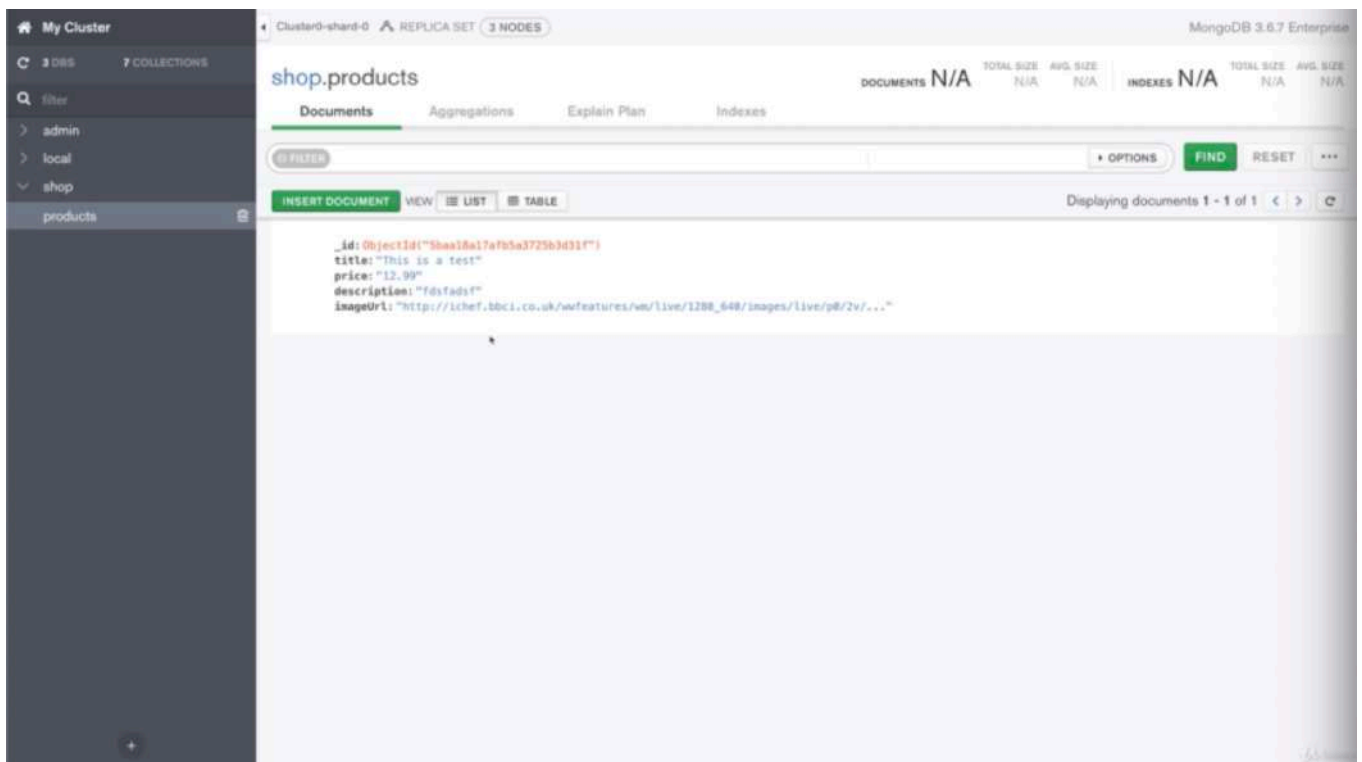
Databases

[CREATE DATABASE](#)

Database Name	Storage Size	Collections	Indexes
admin	0.0B	0	0
local	0.0B	6	0
shop	16.0KB	1	1

- you are now connected to the database server. and you can see a couple of databases, 2 default ones which you don't need to touch but then also your own one, 'shop'

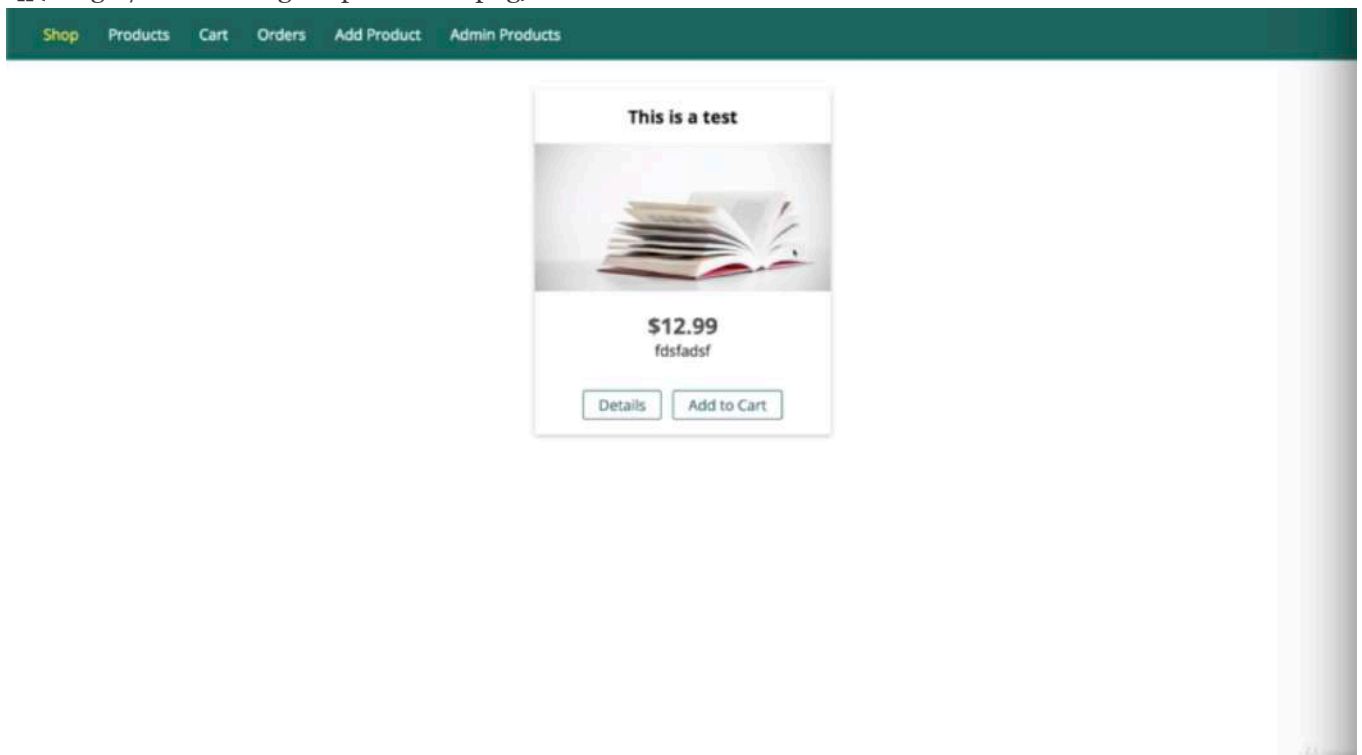
- and the shop database here has a products collection.



- if we look into that products collection, in there, we can see the documents that are stored in there. here's that one document we inserted. so it is one product we added in the last lecture.

* Chapter 182: Fetching All Products

1. update
- ./models/product.js
- ./controllers/shop.js
- ./routes/shop.js
- app.js



```

2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database').mongoConnect;
10
11 const app = express();
12
13 app.set('view engine', 'ejs');
14 app.set('views', 'views');
15
16 const adminRoutes = require('./routes/admin');
17 const shopRoutes = require('./routes/shop');
18
19 app.use(bodyParser.urlencoded({ extended: false }));
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use((req, res, next) => {
23   // User.findById(1)
24   //   .then(user => {
25   //     req.user = user;
26   //     next();
27   //   })
28   //   .catch(err => console.log(err));
29   next();
30 });
31
32 app.use('/admin', adminRoutes);
33 app.use(shopRoutes);
34
35 app.use(errorController.get404);
36
37 mongoConnect(() => {
38   app.listen(3000);
39 });
40

```

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.fetchAll()
7     .then(products => {
8       res.render('shop/product-list', {
9         prods: products,
10         pageTitle: 'All Products',
11         path: '/products'
12       });
13     })
14     .catch(err => {
15       console.log(err);
16     });
17 };

```

```

18
19 exports.getProduct = (req, res, next) => {
20   const prodId = req.params.productId;
21   // Product.findAll({ where: { id: prodId } })
22   //   .then(products => {
23   //     res.render('shop/product-detail', {
24   //       product: products[0],
25   //       pageTitle: products[0].title,
26   //       path: '/products'
27   //     });
28   //   })
29   //   .catch(err => console.log(err));
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
38     .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(cart => {
59       return cart
60         .getProducts()
61         .then(products => {
62           res.render('shop/cart', {
63             path: '/cart',
64             pageTitle: 'Your Cart',
65             products: products
66           });
67         })
68         .catch(err => console.log(err));
69     })
70     .catch(err => console.log(err));
71 };
72
73 exports.postCart = (req, res, next) => {

```



```

74  const prodId = req.body.productId;
75  let fetchedCart;
76  let newQuantity = 1;
77  req.user
78    .getCart()
79    .then(cart => {
80      fetchedCart = cart;
81      return cart.getProducts({ where: { id: prodId } });
82    })
83    .then(products => {
84      let product;
85      if (products.length > 0) {
86        product = products[0];
87      }
88
89      if (product) {
90        const oldQuantity = product.cartItem.quantity;
91        newQuantity = oldQuantity + 1;
92        return product;
93      }
94      return Product.findByPk(prodId);
95    })
96    .then(product => {
97      return fetchedCart.addProduct(product, {
98        through: { quantity: newQuantity }
99      });
100    })
101    .then(() => {
102      res.redirect('/cart');
103    })
104    .catch(err => console.log(err));
105  });
106
107  exports.postCartDeleteProduct = (req, res, next) => {
108    const prodId = req.body.productId;
109    req.user
110      .getCart()
111      .then(cart => {
112        return cart.getProducts({ where: { id: prodId } });
113      })
114      .then(products => {
115        const product = products[0];
116        return product.cartItem.destroy();
117      })
118      .then(result => {
119        res.redirect('/cart');
120      })
121      .catch(err => console.log(err));
122  });
123
124  exports.postOrder = (req, res, next) => {
125    let fetchedCart;
126    req.user
127      .getCart()
128      .then(cart => {
129        fetchedCart = cart;

```

```

130     return cart.getProducts();
131   })
132   .then(products => {
133     return req.user
134       .createOrder()
135       .then(order => {
136         return order.addProducts(
137           products.map(product => {
138             product.orderItem = { quantity: product.cartItem.quantity };
139             return product;
140           })
141         );
142       })
143       .catch(err => console.log(err));
144   })
145   .then(result => {
146     return fetchedCart.setProducts(null);
147   })
148   .then(result => {
149     res.redirect('/orders');
150   })
151   .catch(err => console.log(err));
152 };
153
154 exports.getOrders = (req, res, next) => {
155   req.user
156     .getOrders({include: ['products']})
157     .then(orders => {
158       res.render('shop/orders', {
159         path: '/orders',
160         pageTitle: 'Your Orders',
161         orders: orders
162       });
163     })
164     .catch(err => console.log(err));
165 };
166

```

```

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

```

```

1  ../models/product.js
2
3  const getDb = require('../util/database').getDb
4
5  class Product {
6    constructor(title, price, description, imageUrl){
7      this.title = title
8      this.price = price
9      this.description = description
10     this.imageUrl = imageUrl
11   }
12   save(){
13     const db = getDb()
14     return db.collection('products')
15       .insertOne(this)
16       .then(result => {
17         console.log(result)
18       })
19       .catch(err => {
20         console.log(err)
21       })
22   }
23
24   static fetchAll(){
25     const db = getDb()
26     /**here i wanna interact with my MongoDB database
27     * to 'fetchAll()' products
28     *
29     * MongoDB has a method for finding data
30     * which is called 'find()'
31     * 'find()' could be configured to also use a filter
32     * i wanna find all products which i can do
33     * by calling 'find()'
34     *
35     * the important thing is that
36     * 'find()' doesn't immediately return a promise.
37     * instead it returns so-called 'cursor' which is an object provided by MongoDB
38     * a 'cursor' is an object provided by MongoDB
39     * which allows us to go through our elements, our documents step by step
40     * because theoretically in a collection,
41     * 'find()' could return millions of documents
42     * and you don't wanna transfer them over the wire all at once.
43     * instead 'find()' gives you a handle which you can use to tell MongoDB
44     * OK give me the next document, OK give me the next document and so on.
45     *

```

```

46      * there's 'toArray()' method you can execute to tell MongoDB
47      * to get all documents and turn them into a javascript array.
48      * but you should only use that if we are talking about a couple of dozens or maybe 1
hundred documents.
49      * otherwise it's better to implemenet pagination
50      * which is something we will implement at a later point of time.
51      *
52      * */
53      return db
54        .collection('products')
55        .find()
56        .toArray()
57        .then(products => {
58          console.log(products)
59          return products
60        })
61        .catch(err => {
62          console.log(err)
63        })
64    }
65  }
66
67  module.exports = Product;
68
69

```

* Chapter 183: Fetching A Single Product

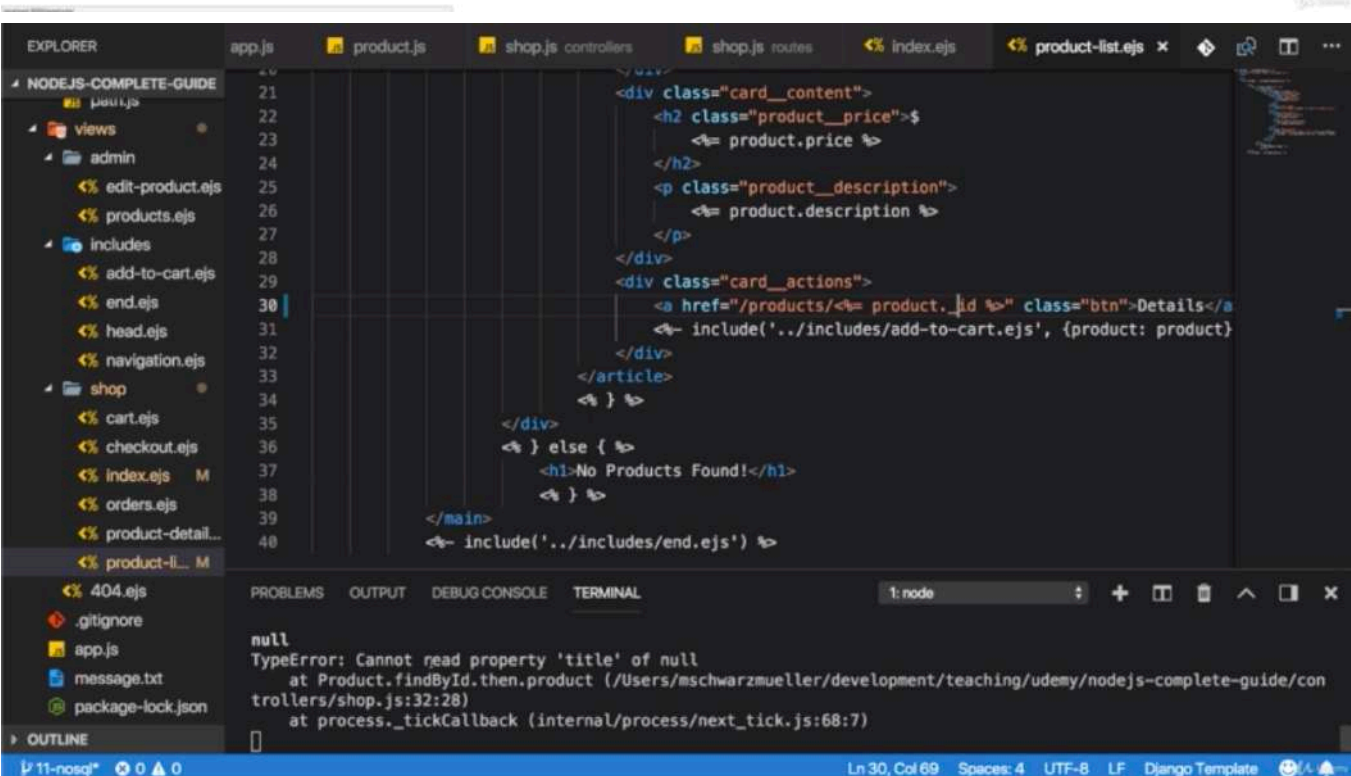
1. update
 - ./models/product.js
 - ./routes/shop.js
 - ./views/shop/index.ejs
 - ./views/product-list.ejs
 - ./controllers/shop.js

\$12.99
fdsfadsf

[Details](#) [Add to Cart](#)

[Details](#) [Add to Cart](#)

[Details](#) [Add to Cart](#)



```
34 .catch(err => {
35   console.log(err);
36 });
37 }
38
39 static findById(prodId) {
40   const db = getDb();
41   return db
42     .collection('products')
43     .find({ _id: prodId })
44     .next()
45     .then(product => {
46       console.log(product);
47       return product;
48     })
49     .catch(err => {
50       console.log(err);
51     });
52 }
53 }
54
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

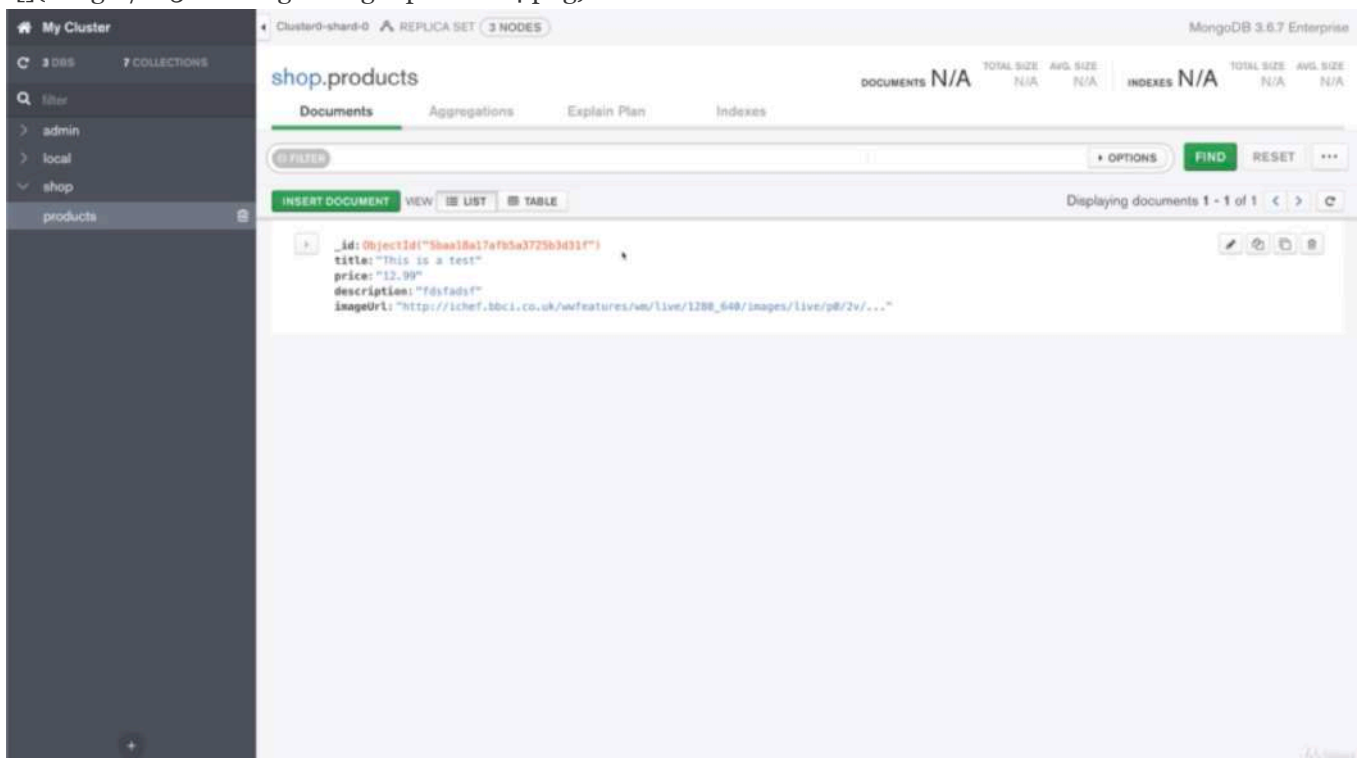
1: node

null
TypeError: Cannot read property 'title' of null
at Product.findById.then.product (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/controllers/shop.js:32:28)
at process._tickCallback (internal/process/next_tick.js:68:7)

Ln 46, Col 30 (22 selected) Spaces: 2 UTF-8 LF JavaScript Prettier: ✓

- why do i get cannot read property 'title' of null? for one it's worth noting that null is printed here as well and that null should be stemming from my product model from findById() when i console.log the product.

- so it looks like we didn't find any product for that ID and what could be the reason for that?

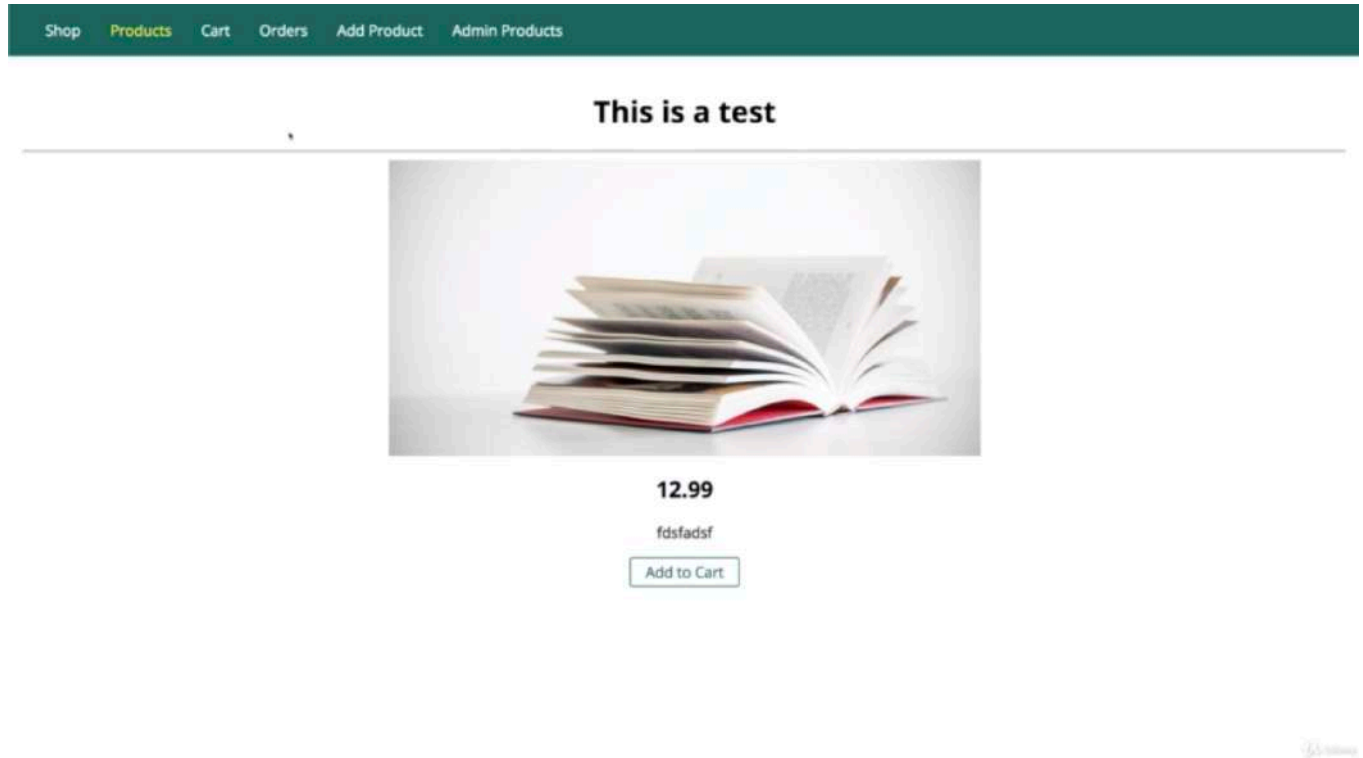


- the reason for that is that The ID in MongoDB is stored a bit differently and we can see this in MongoDB Compass. the ID is such an object id thing.

- MongoDB stores data in BSON format and this binary format of JSON is not used because it's bit faster to work with. but also because MongoDB can store some special types of data in there and object id is such a type.

- it's a type added by MongoDB, it's not a default Javascript type, it doesn't exist in javascript at all. it's simply an ID object which MongoDB uses because this generates and manages IDs which looks random but actually not. so IDs are created in a way that if you create an ID not and an ID one second later, will alphabetically be a higher value than the previous one. that's the one thing

- object id is an object provided by MongoDB.
 - we can't compare '_id' which in the database will only hold object id values with a string because a string is not equal to the object id and the string in here doesn't count, MongoDB will not compare this. it compares the entire object, the entire object ID.
 - so fix this, we simply go into our ./models/product.js, i will import 'const mongodb = require('mongodb')' and i can use mongodb to get access to that object _id type
 - so i can use '{_id: mongodb.ObjectId(prodId)}'
-



- now if i save that, now you see this works.

```

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
28 module.exports = router;
29

```

```

1  //./models/product.js
2
3  const mongodb = require('mongodb')
4  const getDb = require('../util/database').getDb
5
6  class Product {
7    constructor(title, price, description, imageUrl){
8      this.title = title
9      this.price = price
10     this.description = description
11     this.imageUrl = imageUrl
12   }
13   save(){
14     const db = getDb()
15     return db.collection('products')
16       .insertOne(this)
17       .then(result => {
18         console.log(result)
19       })
20       .catch(err => {
21         console.log(err)
22       })
23   }
24
25   static fetchAll(){
26     const db = getDb()
27     return db
28       .collection('products')
29       .find()
30       .toArray()
31       .then(products => {
32         console.log(products)
33         return products
34       })
35       .catch(err => {
36         console.log(err)
37       })
38   }
39
40   static findById(prodId){
41     const db = getDb()
42     /**i will find a product here
43      * but i will find only one product
44      * and to do that,
45      * i will narrow down the result set with 'find()'
46      * and then i will pass a javascript object to it
47      * which allow me to configure a filter and here i wanna look for a product where _id is
equal to 'prodId'
48      * because that's the ID i'm looking for
49      *
50      * 'find()' will still give me a cursor

```



```

51     * because MongoDB doesn't know that i will only get one
52     * an here we can use 'next()' function to get next
53     * and in this case also the last document that was returned by 'find()' here.
54     */
55     return db
56     .collection('products')
57 /** why do i get cannot read property 'title' of null? for one it's worth noting that null
    is printed here as well and that null should be stemming from my product model from
    findById() when i console.log the product.
58     * so it looks like we didn't find any product for that ID and what could be the reason for
    that?
59
60     * the reason for that is that The ID in MongoDB is stored a bit differently and we can see
    this in MongoDB Compass. the ID is such an object id thing.
61     * MongoDB stores data in BSON format and this binary format of JSON is not used because
    it's bit faster to work with. but also because MongoDB can store some special types of data
    in there and object id is such a type.
62     * it's a type added by MongoDB, it's not a default Javascript type, it doesn't exist in
    javascript at all. it's simply an ID object which MongoDB uses because this generates and
    manages IDs which looks random but actually not. so IDs are created in a way that if you
    create an ID not and an ID one second later, will alphabetically be a higher value than the
    previous one. that's the one thing
63     * object id is an object provided by MongoDB.
64     * we can't compare '_id' which in the database will only hold object id values with a
    string because a string is not equal to the object id and the string in here doesn't count,
    MongoDB will not compare this. it compares the entire object, the entire object ID.
65     * so fix this, we simply go into our ./models/product.js, i will import 'const mongodb =
    require('mongodb')'
66     * and i can use mongodb to get access to that object _id type
67     * so i can use '{_id: mongodb.ObjectId(prodId)}'
68     * 'new' is a constructor
69     * and a new objectId to which i pass a string which will be wrapped by that.
70     */
71     .find({_id: new mongodb.ObjectId(prodId)})
72     .next()
73     .then(product => {
74         console.log(product)
75         return product
76     })
77     .catch(err => {
78         console.log(err)
79     })
80 }
81 }
82
83 module.exports = Product;
84
85

```

```

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

```

```

9      prods: products,
10     pageTitle: 'All Products',
11     path: '/products'
12   });
13 }
14 .catch(err => {
15   console.log(err);
16 });
17 };
18
19 exports.getProduct = (req, res, next) => {
20   const prodId = req.params.productId;
21   // Product.findAll({ where: { id: prodId } })
22   // .then(products => {
23   //   res.render('shop/product-detail', {
24   //     product: products[0],
25   //     pageTitle: products[0].title,
26   //     path: '/products'
27   //   });
28   // })
29   // .catch(err => console.log(err));
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
38     .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(cart => {
59       return cart
60         .getProducts()
61         .then(products => {
62           res.render('shop/cart', {
63             path: '/cart',
64             pageTitle: 'Your Cart',

```

```

65         products: products
66     });
67 })
68     .catch(err => console.log(err));
69 })
70     .catch(err => console.log(err));
71 };
72
73 exports.postCart = (req, res, next) => {
74     const prodId = req.body.productId;
75     let fetchedCart;
76     let newQuantity = 1;
77     req.user
78         .getCart()
79         .then(cart => {
80             fetchedCart = cart;
81             return cart.getProducts({ where: { id: prodId } });
82         })
83         .then(products => {
84             let product;
85             if (products.length > 0) {
86                 product = products[0];
87             }
88
89             if (product) {
90                 const oldQuantity = product.cartItem.quantity;
91                 newQuantity = oldQuantity + 1;
92                 return product;
93             }
94             return Product.findByPk(prodId);
95         })
96         .then(product => {
97             return fetchedCart.addProduct(product, {
98                 through: { quantity: newQuantity }
99             });
100         })
101         .then(() => {
102             res.redirect('/cart');
103         })
104         .catch(err => console.log(err));
105 };
106
107 exports.postCartDeleteProduct = (req, res, next) => {
108     const prodId = req.body.productId;
109     req.user
110         .getCart()
111         .then(cart => {
112             return cart.getProducts({ where: { id: prodId } });
113         })
114         .then(products => {
115             const product = products[0];
116             return product.cartItem.destroy();
117         })
118         .then(result => {
119             res.redirect('/cart');
120         })

```

```

121     .catch(err => console.log(err));
122 };
123
124 exports.postOrder = (req, res, next) => {
125     let fetchedCart;
126     req.user
127         .getCart()
128         .then(cart => {
129             fetchedCart = cart;
130             return cart.getProducts();
131         })
132         .then(products => {
133             return req.user
134                 .createOrder()
135                 .then(order => {
136                     return order.addProducts(
137                         products.map(product => {
138                             product.orderItem = { quantity: product.cartItem.quantity };
139                             return product;
140                         })
141                     );
142                 })
143                 .catch(err => console.log(err));
144         })
145         .then(result => {
146             return fetchedCart.setProducts(null);
147         })
148         .then(result => {
149             res.redirect('/orders');
150         })
151         .catch(err => console.log(err));
152 };
153
154 exports.getOrders = (req, res, next) => {
155     req.user
156         .getOrders({include: ['products']})
157         .then(orders => {
158             res.render('shop/orders', {
159                 path: '/orders',
160                 pageTitle: 'Your Orders',
161                 orders: orders
162             });
163         })
164         .catch(err => console.log(err));
165 };
166

```

```

1 <!--./views/shop/index.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4     <link rel="stylesheet" href="/css/product.css">
5 </head>
6
7 <body>
8     <%- include('../includes/navigation.ejs') %>
9
10     <main>

```

```

11 <% if (prods.length > 0) { %>
12   <div class="grid">
13     <% for (let product of prods) { %>
14       <article class="card product-item">
15         <header class="card__header">
16           <h1 class="product__title"><%= product.title %></h1>
17         </header>
18         <div class="card__image">
19           "
21         </div>
22         <div class="card__content">
23           <h2 class="product__price">$<%= product.price %></h2>
24           <p class="product__description"><%= product.description %></p>
25         </div>
26         <div class="card__actions">
27           <a href="/products/<%= product._id %>" class="btn">Details</a>
28           <%- include('../includes/add-to-cart.ejs', {product: product})
29         </div>
30       </article>
31     <% } %>
32   </div>
33 <% } else { %>
34   <h1>No Products Found!</h1>
35 <% } %>
36 </main>
37 <%- include('../includes/end.ejs') %>

```

```

1 <!--./views/shop/product-list.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4 <link rel="stylesheet" href="/css/product.css">
5 </head>
6
7 <body>
8   <%- include('../includes/navigation.ejs') %>
9
10  <main>
11    <% if (prods.length > 0) { %>
12      <div class="grid">
13        <% for (let product of prods) { %>
14          <article class="card product-item">
15            <header class="card__header">
16              <h1 class="product__title">
17                <%= product.title %>
18              </h1>
19            </header>
20            <div class="card__image">
21              "
22            </div>
23            <div class="card__content">
24              <h2 class="product__price">$
25                <%= product.price %>
26              </h2>
27              <p class="product__description">

```

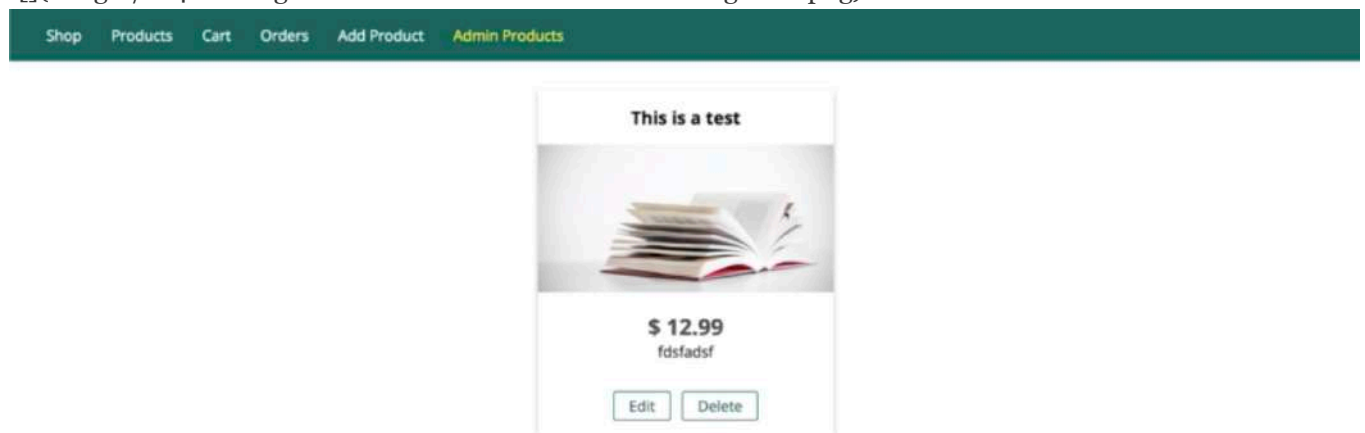
```

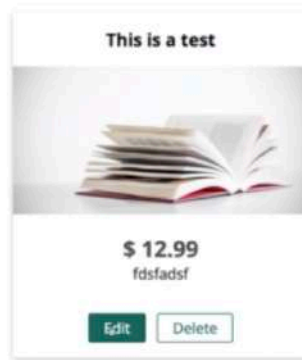
28         <%= product.description %>
29     </p>
30 </div>
31 <div class="card_actions">
32     <a href="/products/<%= product._id %>"
class="btn">Details</a>
33     <%= include('../includes/add-to-cart.ejs', {product:
product}) %>
34 </article>
35 <% } %>
36 </div>
37 <% } else { %>
38     <h1>No Products Found!</h1>
39 <% } %>
40 </main>
41 <%= include('../includes/end.ejs') %>

```

* Chapter 184: Making The “Edit” & “Delete” Buttons Work Again

1. update
 - ./routes/admin.js
 - ./controllers/admin.js
 - ./views/admin/products.ejs





- now with that in place, if we reload that page, our edit and our delete button should work again and should make sure that we edit or delete the right product.

```

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 /*
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 // ./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(title, price, description, imageUrl);
19   product
20     .save()
21     .then(result => {
22       // console.log(result);
23       console.log('Created Product');
24       res.redirect('/admin/products');
25     })
26     .catch(err => {
27       console.log(err);
28     });
29 };
30
31 /*
32 exports.getEditProduct = (req, res, next) => {
33   const editMode = req.query.edit;
34   if (!editMode) {
35     return res.redirect('/');
36   }
37   const prodId = req.params.productId;
38   req.user
39     .getProducts({ where: { id: prodId } })
40     //Product.findByPk(prodId)
41     /**keep in mind we get back an array even if it only holds one element.
42      * so we got 'products'
43      * and therefore we know that one product,
44      * the one we are interested in will always be the first element
45      * so we have to store that separately in a new constant */
46     /*
47     .then(products => {
48       const product = products[0]
49       if (!product) {
50         return res.redirect('/');
51       }
52       res.render('admin/edit-product', {
53         pageTitle: 'Edit Product',
54         path: '/admin/edit-product',
55         editing: editMode,
56         product: product
57       });
58     })
59     .catch(err => console.log(err));
60 };

```



```

61
62 exports.postEditProduct = (req, res, next) => {
63   const prodId = req.body.productId;
64   const updatedTitle = req.body.title;
65   const updatedPrice = req.body.price;
66   const updatedImageUrl = req.body.imageUrl;
67   const updatedDesc = req.body.description;
68   Product.findById(prodId)
69     .then(product => {
70       product.title = updatedTitle;
71       product.price = updatedPrice;
72       product.description = updatedDesc;
73       product.imageUrl = updatedImageUrl;
74       return product.save();
75     })
76     .then(result => {
77       console.log('UPDATED PRODUCT!');
78       res.redirect('/admin/products');
79     })
80     .catch(err => console.log(err));
81 };
82 */
83
84 exports.getProducts = (req, res, next) => {
85   Product.fetchAll()
86     .then(products => {
87       res.render('admin/products', {
88         prods: products,
89         pageTitle: 'Admin Products',
90         path: '/admin/products'
91       });
92     })
93     .catch(err => console.log(err));
94 };
95
96 /*
97 exports.postDeleteProduct = (req, res, next) => {
98   const prodId = req.body.productId;
99   Product.findById(prodId)
100     .then(product => {
101       return product.destroy()
102     })
103     .then(result => {
104       console.log('DESTROYED PRODUCT')
105       res.redirect('/admin/products');
106     })
107     .catch(err => console.log(err))
108 };
109 */

```



```

1 <!--./views/admin/products.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/product.css">
5   </head>
6
7   <body>

```

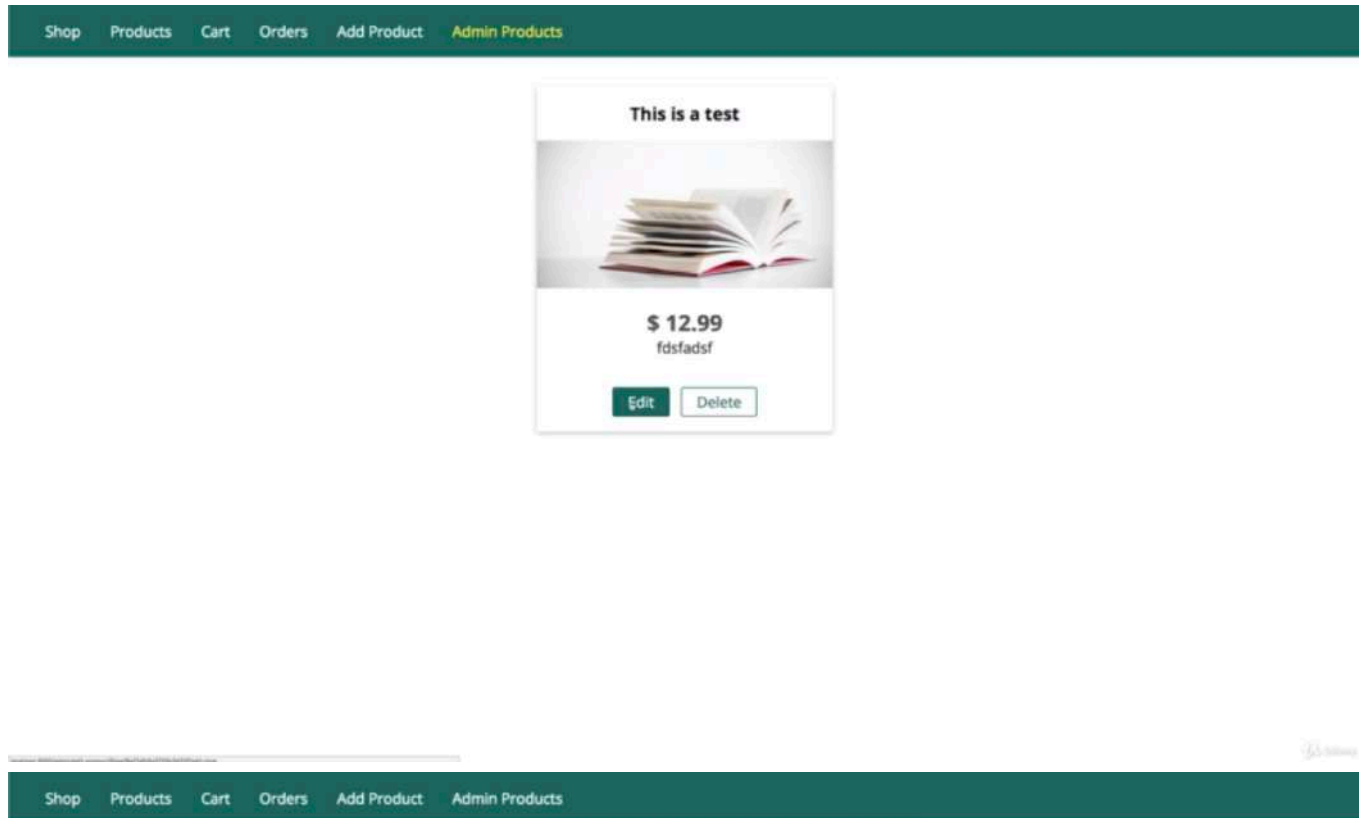
```

8      <%- include('../includes/navigation.ejs') %>
9
10     <main>
11         <% if (prods.length > 0) { %>
12             <div class="grid">
13                 <% for (let product of prods) { %>
14                     <article class="card product-item">
15                         <header class="card__header">
16                             <h1 class="product_title">
17                                 <%= product.title %>
18                             </h1>
19                         </header>
20                         <div class="card__image">
21                             ">
22                         </div>
23                         <div class="card__content">
24                             <h2 class="product__price">$
25                                 <%= product.price %>
26                             </h2>
27                             <p class="product__description">
28                                 <%= product.description %>
29                             </p>
30                         </div>
31                         <div class="card__actions">
32                             <a href="/admin/edit-product/<%= product._id %>?
edit=true" class="btn">Edit</a>
33
34                             <form action="/admin/delete-product" method="POST">
35                                 <!--set the value to productId using ejs templating
syntax and the name to productId
36                                 so that we can extract that information by that
37                                 name.
38                                 -->
39                                 <input type="hidden" value="<%= product._id %>"
name="productId">
40
41                                 <button class="btn" type="submit">Delete</button>
42                             </form>
43
44                         </div>
45                     </article>
46                     <% } %>
47                 </div>
48             <% } else { %>
49                 <h1>No Products Found!</h1>
50             <% } %>
51         </main>
52         <%- include('../includes/end.ejs') %>

```

* Chapter 185: Working On The Product Model To Edit Our Product

1. update
 - ./controllers/admin.js
 - ./views/admin/edit-product.ejs
 - ./routes/admin.js
 - ./models/product.js



```
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 }
```

```

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

```

```

67     })
68     .then(result => {
69         console.log('UPDATED PRODUCT!');
70         res.redirect('/admin/products');
71     })
72     .catch(err => console.log(err));
73 };
74
75 exports.getProducts = (req, res, next) => {
76     Product.fetchAll()
77         .then(products => {
78             res.render('admin/products', {
79                 prods: products,
80                 pageTitle: 'Admin Products',
81                 path: '/admin/products'
82             });
83         })
84         .catch(err => console.log(err));
85 };
86
87 /*
88 exports.postDeleteProduct = (req, res, next) => {
89     const prodId = req.body.productId;
90     Product.findById(prodId)
91         .then(product => {
92             return product.destroy()
93         })
94         .then(result => {
95             console.log('DESTROYED PRODUCT')
96             res.redirect('/admin/products');
97         })
98         .catch(err => console.log(err))
99 };
100 */

```

```

1  <!--./views/admin/edit-product.ejs-->
2
3  <%- include('../includes/head.ejs') %>
4      <link rel="stylesheet" href="/css/forms.css">
5      <link rel="stylesheet" href="/css/product.css">
6  </head>
7
8  <body>
9      <%- include('../includes/navigation.ejs') %>
10
11      <main>
12          <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else
13 { %>add-product<% } %>" method="POST">
14              <div class="form-control">
15                  <label for="title">Title</label>
16                  <input type="text" name="title" id="title" value="<% if (editing) { %><%=
17 product.title %><% } %>">
18              </div>
19              <div class="form-control">
20                  <label for="imageUrl">Image URL</label>
21                  <input type="text" name="imageUrl" id="imageUrl" value="<% if (editing) { %>
22 <%= product.imageUrl %><% } %>">

```

```

20     </div>
21     <div class="form-control">
22         <label for="price">Price</label>
23         <input type="number" name="price" id="price" step="0.01" value="<%= if
(editing) { %><%= product.price %><% } %>">
24     </div>
25     <div class="form-control">
26         <label for="description">Description</label>
27         <textarea name="description" id="description" rows="5"><% if (editing) { %>
<%= product.description %><% } %></textarea>
28     </div>
29     <% if (editing) { %>
30         <input type="hidden" value="<%= product._id %>" name="productId">
31     <% } %>
32
33     <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else {
%>Add Product<% } %></button>
34 </form>
35 </main>
36 <%- include('../includes/end.ejs') %>

```

```

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
27 module.exports = router;

```

```

1 //./models/product.js
2
3 const mongodb = require('mongodb')
4 const getDb = require('../util/database').getDb
5
6 class Product {
7     /**how could we update our product which is stored in the database?
8     * let's add a fifth argument here, the ID

```

```

9      * and then i will say 'this._id = id'
10     * now we accept a kind of optional fifth argument.
11     */
12     constructor(title, price, description, imageUrl, id){
13         this.title = title
14         this.price = price
15         this.description = description
16         this.imageUrl = imageUrl
17         this._id = id
18     }
19     save(){
20         const db = getDb()
21         let dbOp
22         if(this._id){
23             //Update the product
24             /**i use 'updateOne()'
25              * and as the name suggests,
26              * 'updateOne()' will update exactly one element
27              * there's also 'updateMany()' where you can update multiple elements at once.
28              */
29             dbOp = db.collection('products')
30             /**'updateOne()' takes at least 2 arguments
31              * the first one is that we add a filter
32              * that defines which element or which document we wanna update
33              * so i will pass a javascript object
34              * and we can filter for equality also or run more complex queries
35              * anyway, here i only wanna find a document where the '_id' is equal to
36              * and now again '_id: new mongodb.ObjectId(this._id)'
37              * so i'm looking for a document where the ID matches the ID i have here in my product
38              *
39              * and we now as a second argument to 'updateOne()',
40              * we now specify how to update that document.
41              * this again is a javascript object where we describe the update
42              * and this is now not the new object.
43              * so we don't say 'this' here as you could imagine that we tell MongoDB find me the
44              existing document and replace it with this.
45              * 'updateOne()' doesn't replace
46              *
47              * instead we have to describe the operation and we do this by using a special
48              property name
49              * which is understood by MongoDB, kind of a reserved name. '$set'
50              * '$set' again takes an object as a value
51              * and here we describe the changes we wanna make to the existing document which we
52              found with this filter
53              * and here you could say 'this' and you would instruct MongoDB to set these key value
54              field like 'this.title = title' 'this.price = price' above.
55              * which you have in your object to the document it found in the database
56              * it will update the values of the document in the database with your new values.
57              * we wanna replace all fields, we can just say {$set: this}
58              */
59              .updateOne({_id: new mongodb.ObjectId(this._id)}, {$set: this})
60         } else {
61             dbOp = db
62                 .collection('products')
63                 .insertOne(this)

```

```

60     }
61     return dbOp
62     .then(result => {
63         console.log(result)
64     })
65     .catch(err => {
66         console.log(err)
67     })
68 }
69
70 static fetchAll(){
71     const db = getDb()
72     return db
73     .collection('products')
74     .find()
75     .toArray()
76     .then(products => {
77         console.log(products)
78         return products
79     })
80     .catch(err => {
81         console.log(err)
82     })
83 }
84
85 static findById(prodId){
86     const db = getDb()
87     return db
88     .collection('products')
89     .find({_id: new mongodb.ObjectId(prodId)})
90     .next()
91     .then(product => {
92         console.log(product)
93         return product
94     })
95     .catch(err => {
96         console.log(err)
97     })
98 }
99 }
100
101 module.exports = Product;
102
103

```

* Chapter 186: Finishing The “Update Product” Code

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

Title

This is a test

Image URL

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

Price

19,99

Description

fdsfadsf

Update Product

This is a test

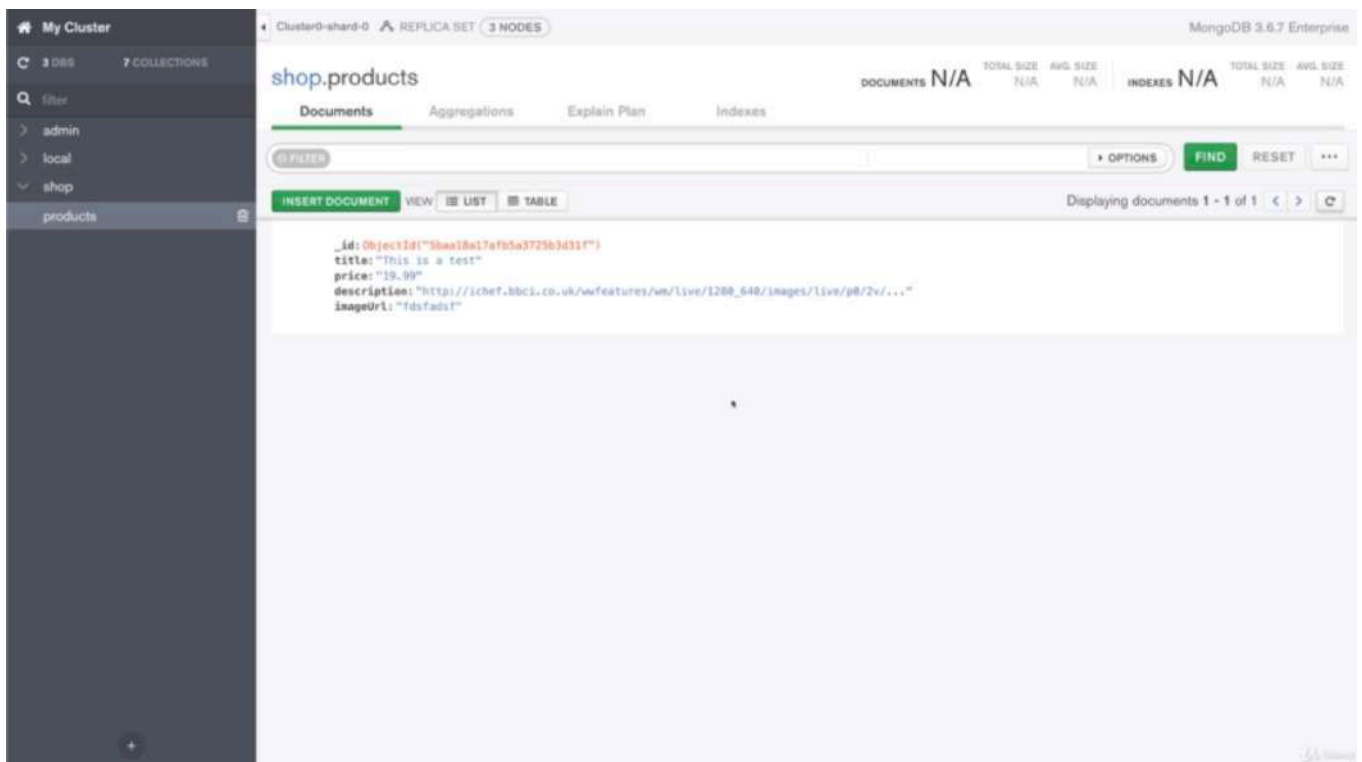
 This is a test

\$ 19.99

http://ichef.bbci.co.uk/ww/features/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg

Edit

Delete



```

1 // ./controllers/admin.js
2
3 const mongoose = require('mongoose')
4 const Product = require('../models/product');
5
6 const ObjectId = mongoose.Types.ObjectId
7
8 exports.getAddProduct = (req, res, next) => {
9   res.render('admin/edit-product', {
10     pageTitle: 'Add Product',
11     path: '/admin/add-product',
12     editing: false
13   });
14 };
15
16 exports.postAddProduct = (req, res, next) => {
17   const title = req.body.title;
18   const imageUrl = req.body.imageUrl;
19   const price = req.body.price;
20   const description = req.body.description;
21   const product = new Product(title, price, description, imageUrl);
22   product
23     .save()
24     .then(result => {
25       // console.log(result);
26       console.log('Created Product');
27       res.redirect('/admin/products');
28     })
29     .catch(err => {
30       console.log(err);
31     });
32 };
33
34
35 exports.getEditProduct = (req, res, next) => {

```

```

36 const editMode = req.query.edit;
37 if (!editMode) {
38     return res.redirect('/');
39 }
40 const prodId = req.params.productId;
41 Product.findById(prodId)
42 //Product.findByIdPk(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     /**we already have code where we find a product by id
64      * and this will work because 'findById()' is a method i created in my model
65      *
66      */
67     /**we don't even need 'findById()' and 'then()' anymore
68      * we can just get rid of that
69      * and for now also create a new product with all the updated information
70      * and we also need to pass the product ID,
71      * there you just need to make sure that you create a 'new mongodb.ObjectId' object
72      * so make sure that at the top of the file,
73      * you require 'mongodb'
74      * and you can create a new constant, name it 'ObjectId'
75      * and extract that 'ObjectId' constructor out of mongodb
76      * and then you could write new ObjectId
77      * and reference this.
78      * so we can now go down to postEditProduct
79      */
80     const product = new Product(updatedTitle, updatedPrice, updatedDesc, updatedImageUrl,
new ObjectId(prodId))
81     product
82     /**we call 'product.save()'
83      * because we modified the 'save()' method to support both creation and updating
84      */
85     .save()
86     .then(result => {
87         console.log('UPDATED PRODUCT!');
88         res.redirect('/admin/products');
89     })
90     .catch(err => console.log(err));

```

```

91 };
92
93 exports.getProducts = (req, res, next) => {
94   Product.fetchAll()
95     .then(products => {
96       res.render('admin/products', {
97         prods: products,
98         pageTitle: 'Admin Products',
99         path: '/admin/products'
100       });
101     })
102     .catch(err => console.log(err));
103 };
104
105 /*
106 exports.postDeleteProduct = (req, res, next) => {
107   const prodId = req.body.productId;
108   Product.findById(prodId)
109     .then(product => {
110       return product.destroy()
111     })
112     .then(result => {
113       console.log('DESTROYED PRODUCT')
114       res.redirect('/admin/products');
115     })
116     .catch(err => console.log(err))
117 };
118 */

```



```

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
27 module.exports = router;

```

* Chapter 187: One Note About Updating Products

1. update

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

Shop Products Cart Orders Add Product Admin Products

Title
This is a test

Image URL
http://ichef.bbci.co.uk/ww/features/wm/live

Price
18.99

Description
This also works!

Update Product

187

```
1 //./models/product.js
2
3 const mongodb = require('mongodb')
4 const getDb = require('./util/database').getDb
5
6 class Product {
7   constructor(title, price, description, imageUrl, id){
8     this.title = title
9     this.price = price
10    this.description = description
11    this.imageUrl = imageUrl
12    this._id = new mongodb.ObjectId(id)
13  }
14  save(){
15    const db = getDb()
16    let dbOp
17    if(this._id){
18      dbOp = db.collection('products')
19      /**if we go to the ./models/product.js file,
20       * and we have a look at the save() method,
21       * i'm looking for the right object
22       * but i will have a problem with updating it
```

```

23      * because i will try set my object id to a different object id to a string
24      * because i'm referring to these things like 'this.title = title' 'this.price =
price' and so on
25      * which will hold the unmodified objectId
26      * so i should automatically convert the 'objectId' the id which is the string to an
object
27      * and the object in the constructor
28      * so that we can remove down there. 'new mongodb.ObjectId'
29      * because '_id' will always be an ObjectId field
30      * no matter if i'm using it in a filter or if i'm using it for updating.
31      */
32      .updateOne(
33          {_id: this._id},
34          {$set: this}
35      )
36  } else {
37      dbOp = db
38          .collection('products')
39          .insertOne(this)
40  }
41  return dbOp
42      .then(result => {
43          console.log(result)
44      })
45      .catch(err => {
46          console.log(err)
47      })
48  }
49
50  static fetchAll(){
51      const db = getDb()
52      return db
53          .collection('products')
54          .find()
55          .toArray()
56          .then(products => {
57              console.log(products)
58              return products
59          })
60          .catch(err => {
61              console.log(err)
62          })
63  }
64
65  static findById(prodId){
66      const db = getDb()
67      return db
68          .collection('products')
69          .find({_id: new mongodb.ObjectId(prodId)})
70          .next()
71          .then(product => {
72              console.log(product)
73              return product
74          })
75          .catch(err => {
76              console.log(err)

```

```

77     })
78   }
79 }
80
81 module.exports = Product;
82
83

```

```

1  // ./controllers/admin.js
2
3  const mongoose = require('mongoose')
4  const Product = require('../models/product');
5
6  exports.getAddProduct = (req, res, next) => {
7    res.render('admin/edit-product', {
8      pageTitle: 'Add Product',
9      path: '/admin/add-product',
10     editing: false
11   });
12 };
13
14 exports.postAddProduct = (req, res, next) => {
15   const title = req.body.title;
16   const imageUrl = req.body.imageUrl;
17   const price = req.body.price;
18   const description = req.body.description;
19   const product = new Product(
20     title,
21     price,
22     description,
23     imageUrl
24   );
25   product
26     .save()
27     .then(result => {
28       // console.log(result);
29       console.log('Created Product');
30       res.redirect('/admin/products');
31     })
32     .catch(err => {
33       console.log(err);
34     });
35 };
36
37
38 exports.getEditProduct = (req, res, next) => {
39   const editMode = req.query.edit;
40   if (!editMode) {
41     return res.redirect('/');
42   }
43   const prodId = req.params.productId;
44   Product.findById(prodId)
45   //Product.findByIdPk(prodId)
46   .then(product => {
47     if (!product) {
48       return res.redirect('/');
49     }

```

```

50     res.render('admin/edit-product', {
51         pageTitle: 'Edit Product',
52         path: '/admin/edit-product',
53         editing: editMode,
54         product: product
55     });
56 })
57 .catch(err => console.log(err));
58 };
59
60 exports.postEditProduct = (req, res, next) => {
61     const prodId = req.body.productId;
62     const updatedTitle = req.body.title;
63     const updatedPrice = req.body.price;
64     const updatedImageUrl = req.body.imageUrl;
65     const updatedDesc = req.body.description;
66     const product = new Product(
67         updatedTitle,
68         updatedPrice,
69         updatedDesc,
70         updatedImageUrl,
71         /**i passed an 'ObjectId' in ./controllers/admin.js to my Product constructor
72         * now i could also just pass 'prodId' as a string
73         * and therefore remove my 'ObjectId' constant above import
74         * all of that is happening in the controller now
75         *
76         * this is an important note
77         * you don't have to convert the ID in the controller file
78         * you can leave that untouched
79         * but you can do a general conversion in the ./models/product.js file
80         * which is the better approach of doing that.
81         */
82         prodId
83     )
84     product
85     .save()
86     .then(result => {
87         console.log('UPDATED PRODUCT!');
88         res.redirect('/admin/products');
89     })
90     .catch(err => console.log(err));
91 };
92
93 exports.getProducts = (req, res, next) => {
94     Product.fetchAll()
95     .then(products => {
96         res.render('admin/products', {
97             prods: products,
98             pageTitle: 'Admin Products',
99             path: '/admin/products'
100         });
101     })
102     .catch(err => console.log(err));
103 };
104
105 /*

```



```

106 exports.postDeleteProduct = (req, res, next) => {
107   const prodId = req.body.productId;
108   Product.findById(prodId)
109     .then(product => {
110       return product.destroy()
111     })
112     .then(result => {
113       console.log('DESTROYED PRODUCT')
114       res.redirect('/admin/products');
115     })
116     .catch(err => console.log(err))
117   };
118 */

```

* Chapter 188: Deleting Products

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

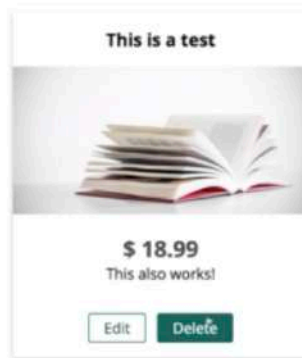
[Shop](#)
[Products](#)
[Cart](#)
[Orders](#)
[Add Product](#)
[Admin Products](#)

Title

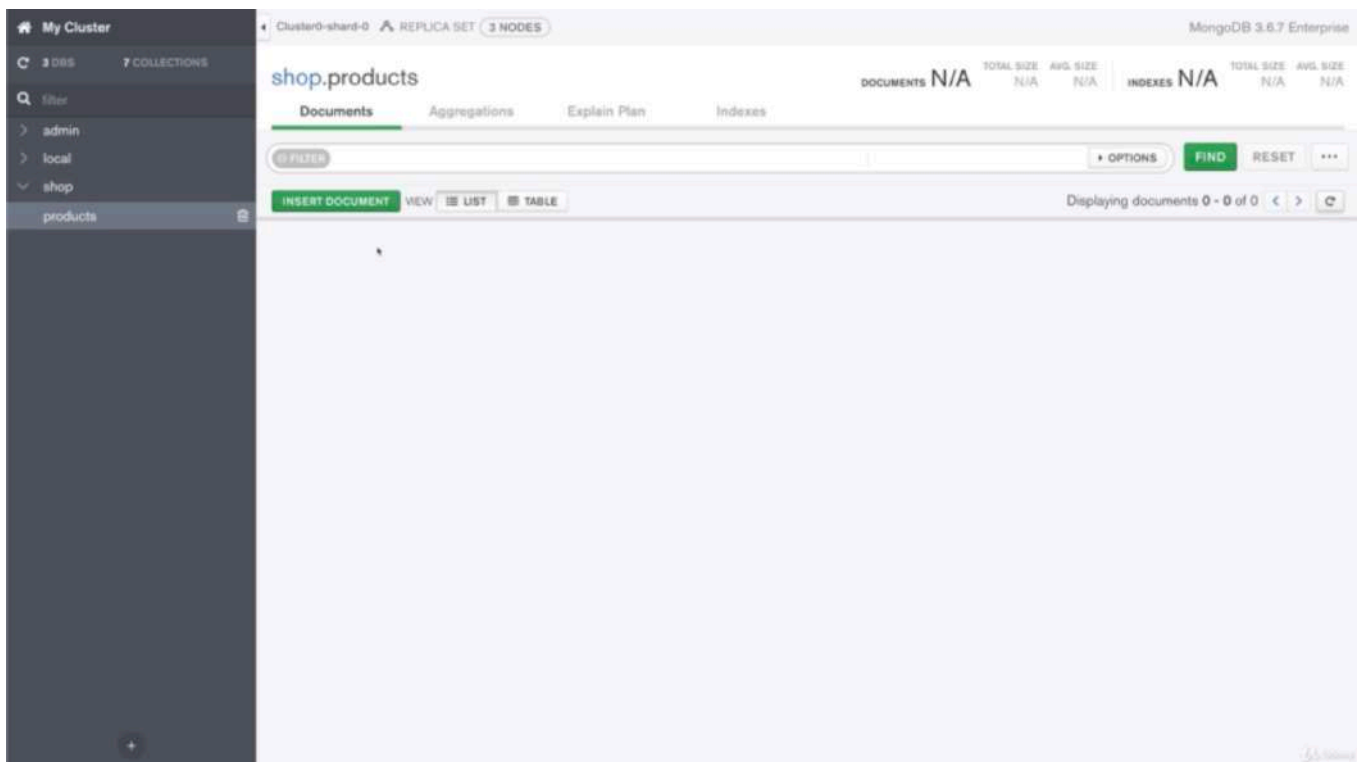
Image URL

Price

Description



No Products Found!



- let's confirm in compass by quickly refreshing that page and our product is gone. so deleting works well.

[nodemon] clean exit - waiting for changes before restart

[nodemon] restarting due to changes...

[nodemon] starting `node app.js`

(node:33113) 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.

{ MongoNetworkError: connection 3 to cluster0-shard-00-00-z3v1k.mongodb.net:27017 closed

at TLSSocket.<anonymous> (/Users/kiwonkim/Desktop/nodejs-complete-guide/node_modules/mongodb-core/lib/connection/connection.js:352:9)

at Object.onceWrapper (events.js:273:13)

at TLSSocket.emit (events.js:182:13)

at _handle.close (net.js:610:12)

at TCP.done (_tls_wrap.js:386:7)

name: 'MongoNetworkError',

errorLabels: ['TransientTransactionError'],

[Symbol(mongoErrorContextSymbol)]: {} }

(node:33113) UnhandledPromiseRejectionWarning: MongoNetworkError: connection 3 to cluster0-shard-00-00-z3v1k.mongodb.net:27017 closed

at TLSSocket.<anonymous> (/Users/kiwonkim/Desktop/nodejs-complete-guide/node_modules/mongodb-core/lib/connection/connection.js:352:9)

at Object.onceWrapper (events.js:273:13)

at TLSSocket.emit (events.js:182:13)

at _handle.close (net.js:610:12)

at TCP.done (_tls_wrap.js:386:7)

(node:33113) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch(). (rejection id: 1)

(node:33113) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process with a non-zero exit code.

- this Error message is due to wrong IP Whitelist in MongoDB Atlas. IP Whitelist is periodically changed. so if you got error like that, you should go to your Clusters you have been using in MongoDB Atlas, go to IP whitelist and edit or replace your expired IP to new IP whitelist. as soon as you edit it, Status of IP Address will be on 'pending' and after that go to 'Active' then Error is gone.

The screenshot displays the MongoDB Atlas interface for a cluster named 'KIWON'S ORG - 2019-04-17 > PROJECT 0'. The 'Security' tab is selected, showing the 'IP Whitelist' section. A table lists the whitelisted IP addresses, with one entry: '175.223.22.103/32 (includes your current IP address)' with a status of 'Active'. A yellow warning box above the table states: 'You will only be able to connect to your cluster from the following list of IP Addresses:'. The page also features a sidebar with navigation options like Clusters, Alerts, Backup, Access, Settings, Stitch, Charts, Docs, and Support. At the bottom, there is a system status bar showing 'All Good' and a last login time.

```
1 //./models/product.js
2
3 const mongodb = require('mongodb');
4 const getDb = require('./util/database').getDb;
5
6 class Product {
7   constructor(title, price, description, imageUrl, id) {
8     this.title = title;
```

```

9     this.price = price;
10    this.description = description;
11    this.imageUrl = imageUrl;
12    this._id = new mongodb.ObjectId(id);
13  }
14
15  save() {
16    const db = getDb();
17    let dbOp;
18    if (this._id) {
19      // Update the product
20      dbOp = db
21        .collection('products')
22        .updateOne({ _id: this._id }, { $set: this });
23    } else {
24      dbOp = db.collection('products').insertOne(this);
25    }
26    return dbOp
27      .then(result => {
28        console.log(result);
29      })
30      .catch(err => {
31        console.log(err);
32      });
33  }
34
35  static fetchAll() {
36    const db = getDb();
37    return db
38      .collection('products')
39      .find()
40      .toArray()
41      .then(products => {
42        console.log(products);
43        return products;
44      })
45      .catch(err => {
46        console.log(err);
47      });
48  }
49
50  static findById(prodId) {
51    const db = getDb();
52    return db
53      .collection('products')
54      .find({ _id: new mongodb.ObjectId(prodId) })
55      .next()
56      .then(product => {
57        console.log(product);
58        return product;
59      })
60      .catch(err => {
61        console.log(err);
62      });
63  }
64

```

```

65 static deleteById(prodId) {
66     const db = getDb();
67     /**you need to specify a filter now
68     * so pass in object where you describe how to find that product
69     * and again it will be our _id equal to check here
70     * here product id is an argument
71     * so we need to convert it to an ObjectId manually again
72     * by passing it to the ObjectId constructor.
73     * now mongodb will go ahead and delete the first element it finds that has this
criteria fulfilled.
74     */
75     return db
76         .collection('products')
77         .deleteOne({ _id: new mongodb.ObjectId(prodId) })
78         .then(result => {
79             console.log('Deleted');
80         })
81         .catch(err => {
82             console.log(err);
83         });
84     }
85 }
86
87 module.exports = 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(title, price, description, imageUrl);
19     product
20         .save()
21         .then(result => {
22             // console.log(result);
23             console.log('Created Product');
24             res.redirect('/admin/products');
25         })
26         .catch(err => {
27             console.log(err);
28         });
29 };
30
31 exports.getEditProduct = (req, res, next) => {
32     const editMode = req.query.edit;

```

```

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

```

```

89  /**we extract prodId
90   * and here we had a different flow
91   * i first of all found the product here
92   * we just call 'Product.deleteById(prodId)'
93   * we pass in the 'prodId' as a string
94   * and then we just have our 'then()' block which won't receive an argument
95   */
96  const prodId = req.body.productId;
97  Product.deleteById(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 189: Fixing The “Add Product” Functionality

1. update
- ./models/product.js

Title
fasdfdas

Image URL
fasdfffd

Price
12

Description
fadsfads

Add Product

fasdfdas

 fasdfdas

\$ 12
fadsfads

Edit Delete

```

1 //./models/product.js
2
3 const mongodb = require('mongodb');
4 const getDb = require('../util/database').getDb;
5
6 /**even if we pass no 'id' as an argument and this therefore is undefined,
7 * this will create an object and store it in '_id'
8 * so 'this._id' down there will always be defined.
9 * and if it's just such an empty or automatically generated ObjectId object,
10 * this should be the issue here.
11 */
12 class Product {
13   constructor(title, price, description, imageUrl, id) {
14     this.title = title;

```

```

15     this.price = price;
16     this.description = description;
17     this.imageUrl = imageUrl;
18     /**for example with a ternary expression,
19     * we can check if ID exists,
20     * so if this returns true in an if statement
21     * and if it's the case,
22     * then i wanna create my ObjectId object
23     * otherwise i will store null and null will be treated as false down there.
24     *
25     */
26     this._id = id ? new mongodb.ObjectId(id) : null
27 }
28
29 save() {
30     const db = getDb();
31     let dbOp;
32     if (this._id) {
33         // Update the product
34         dbOp = db
35             .collection('products')
36             .updateOne({ _id: this._id }, { $set: this });
37     } else {
38         dbOp = db.collection('products').insertOne(this);
39     }
40     return dbOp
41         .then(result => {
42             console.log(result);
43         })
44         .catch(err => {
45             console.log(err);
46         });
47 }
48
49 static fetchAll() {
50     const db = getDb();
51     return db
52         .collection('products')
53         .find()
54         .toArray()
55         .then(products => {
56             console.log(products);
57             return products;
58         })
59         .catch(err => {
60             console.log(err);
61         });
62 }
63
64 static findById(prodId) {
65     const db = getDb();
66     return db
67         .collection('products')
68         .find({ _id: new mongodb.ObjectId(prodId) })
69         .next()
70         .then(product => {

```

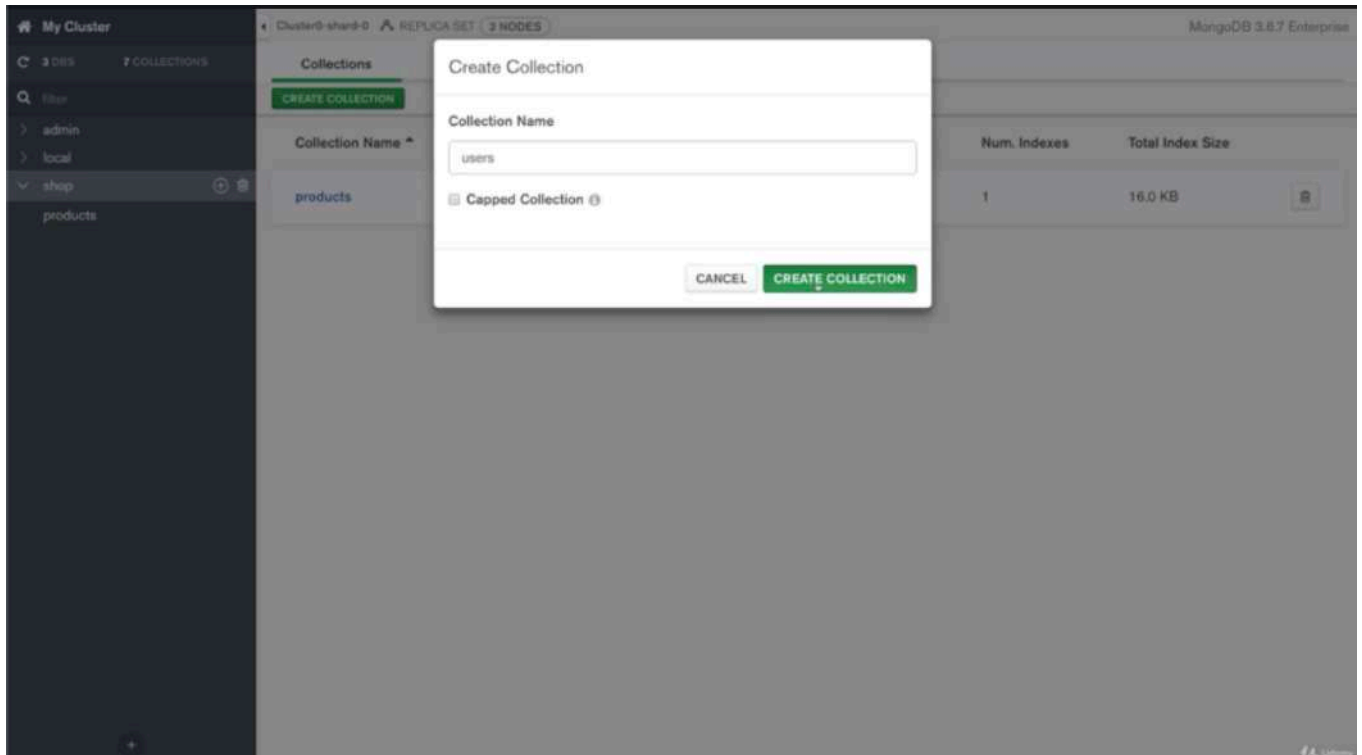
```

71     console.log(product);
72     return product;
73   })
74   .catch(err => {
75     console.log(err);
76   });
77 }
78
79 static deleteById(prodId) {
80   const db = getDb();
81   return db
82     .collection('products')
83     .deleteOne({ _id: new mongodb.ObjectId(prodId) })
84     .then(result => {
85       console.log('Deleted');
86     })
87     .catch(err => {
88       console.log(err);
89     });
90 }
91 }
92
93 module.exports = Product;

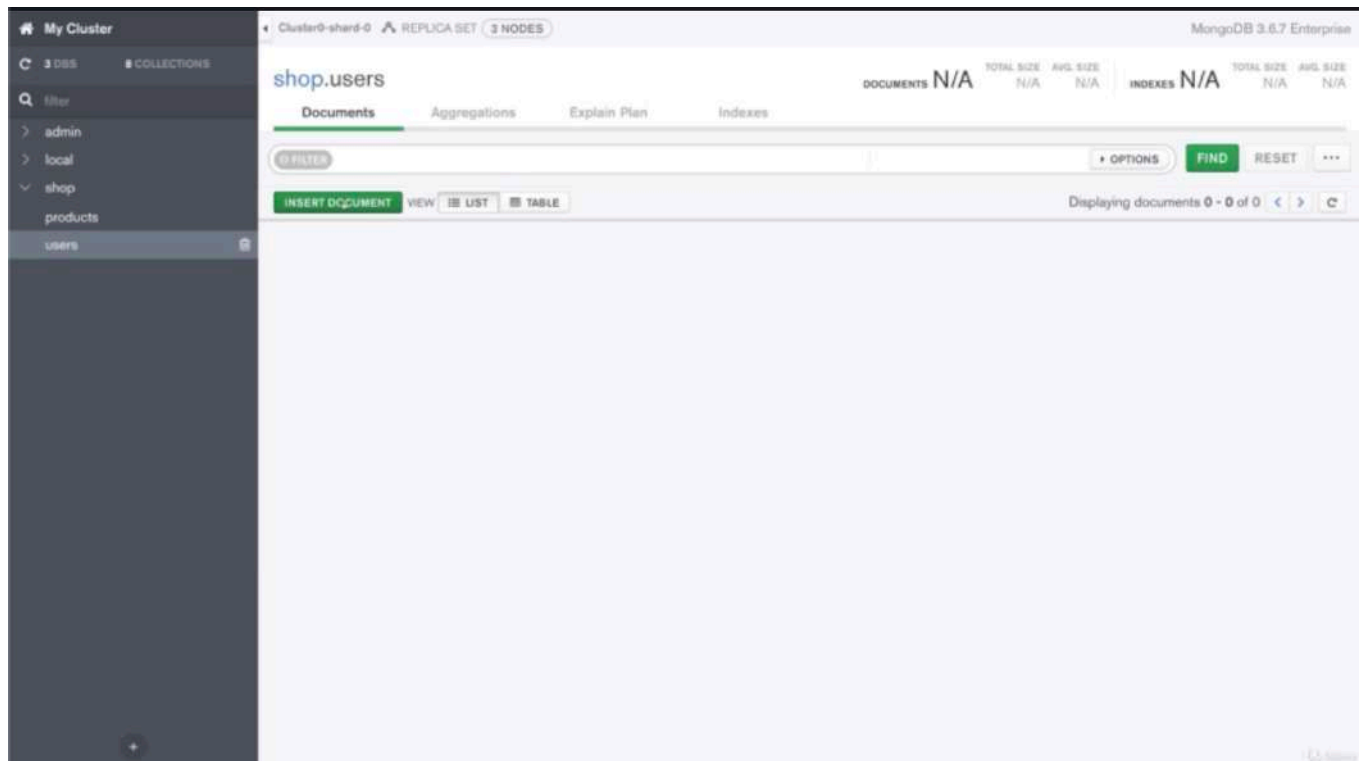
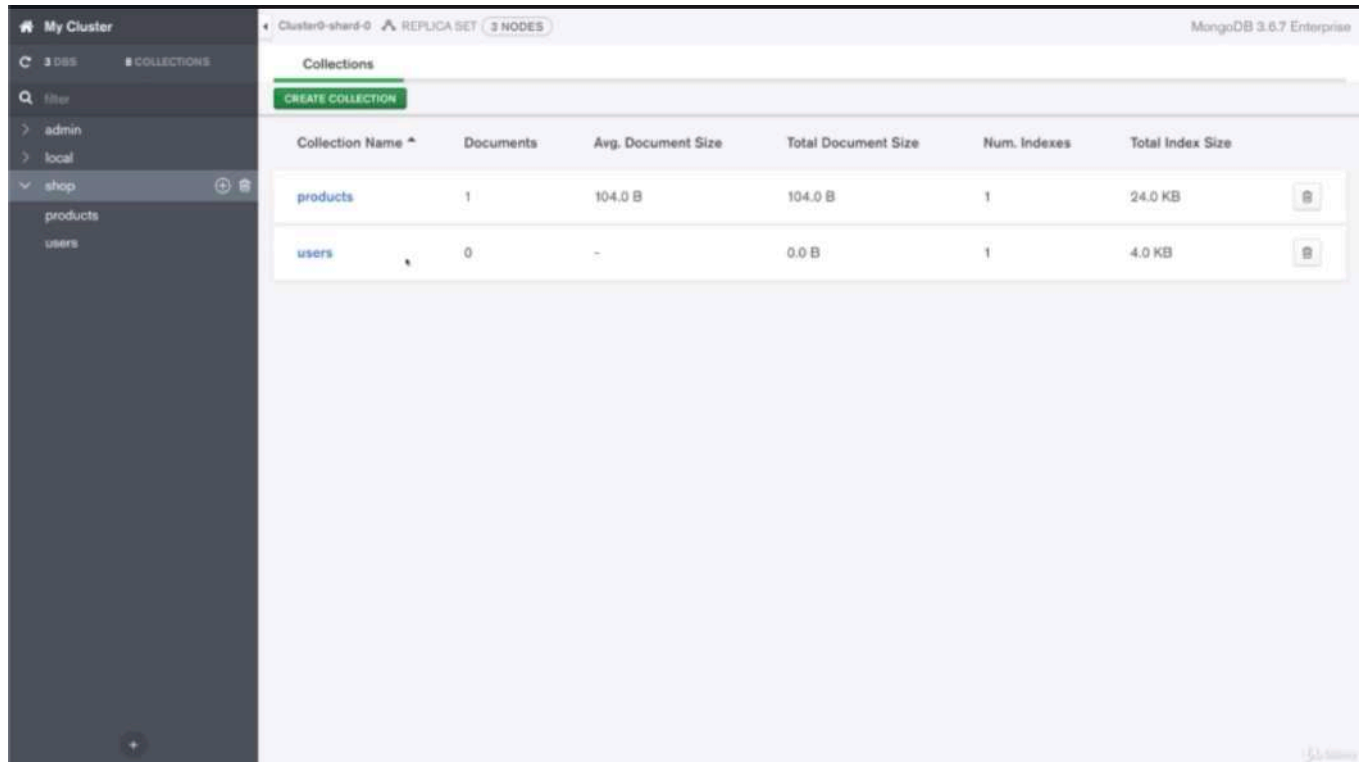
```

* Chapter 190: Creating New Users

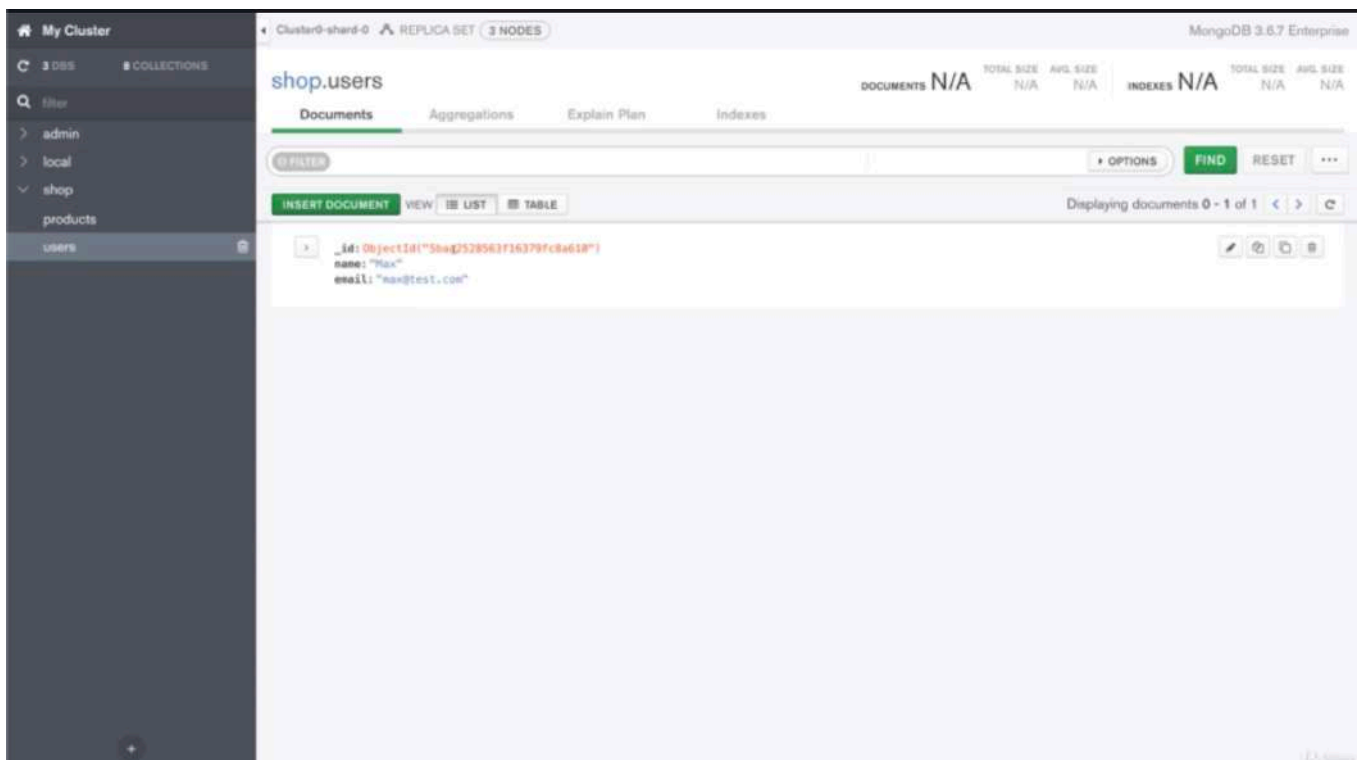
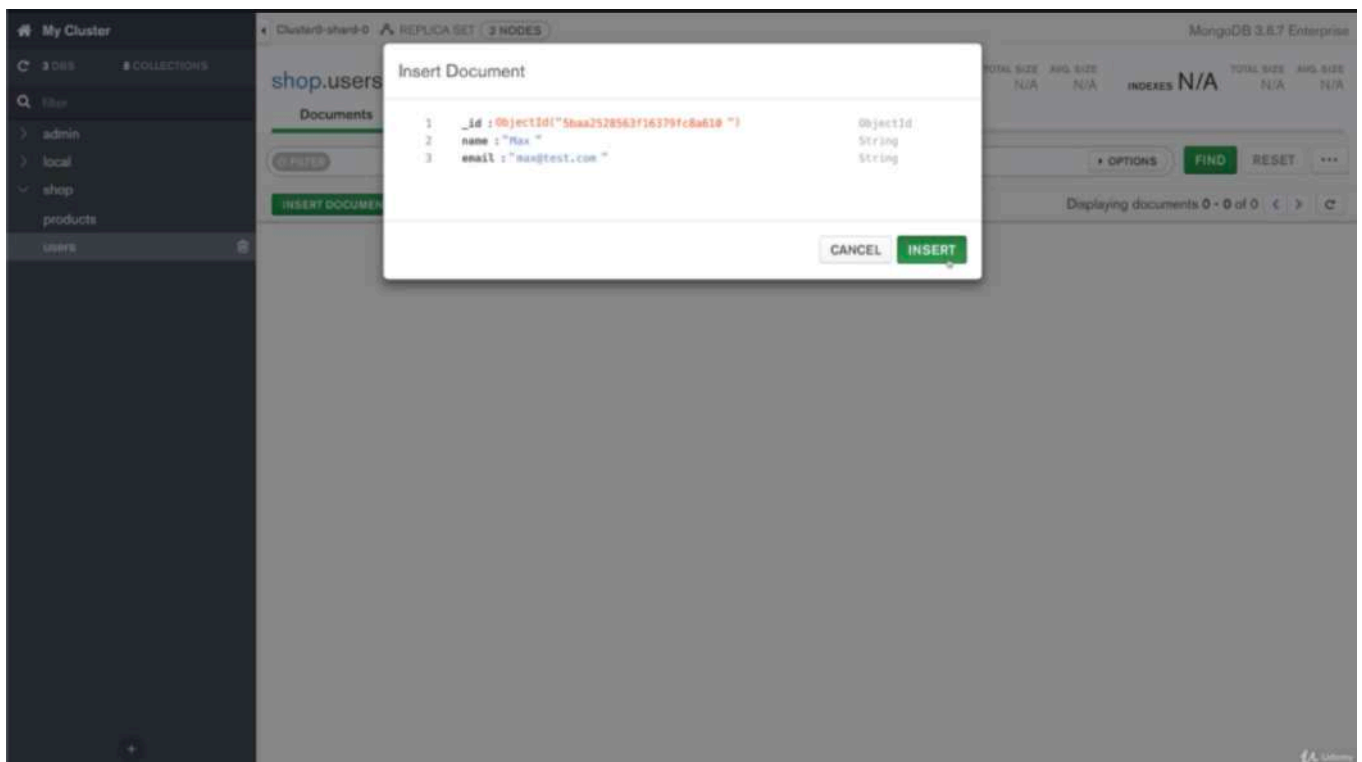
1. update
- ./models/user.js
- app.js



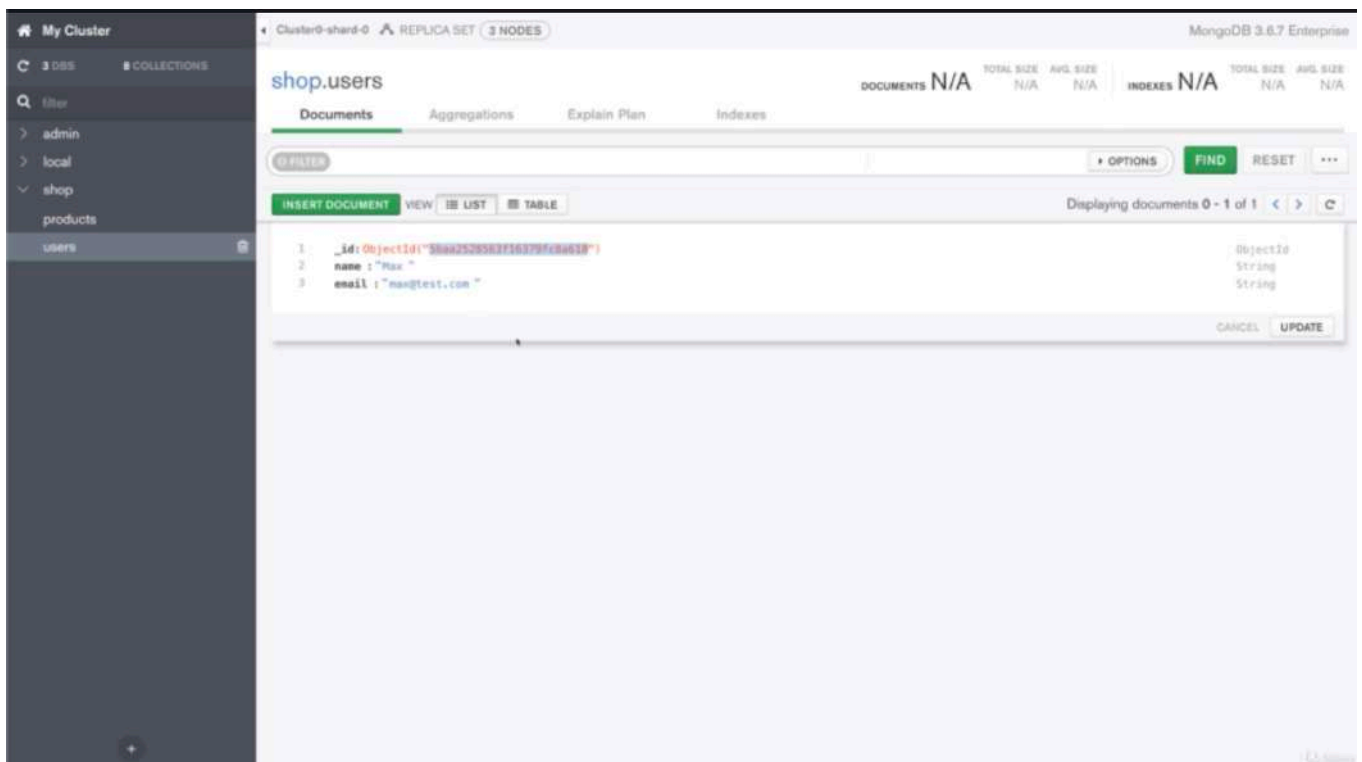
- i will connect to the shop database and i will create a new collection called 'users' and of course this collection name here should be the collection name you chose in your code.



- after you create 'users' collection, then go into users collection.



- if we wanna create one behind the scenes in compass, we can insert a new document here. and here is the automatically generated ID and i can enter a name and you can use any name you want and an email and insert that document



- and this ID, this part here between the quotation marks that matters to me. i will take that and i will use that in app.js file

```

1  //./models/user.js
2
3  const mongodb = require('mongodb')
4  const getDb = require('../util/database').getDb
5
6  /** i will follow a slightly different approach
7   * which i already showed before
8   * and i will create an ObjectId constant
9   * and simply store access to it by accessing it up here.
10  * but i'm not calling it, i'm not creating an object in here.
11  * i'm storing the reference to the ObjectId class in my ObjectId constant
12  * and then down there, i can just write 'new ObjectId'
13  */
14  const ObjectId = mongodb.ObjectId
15
16  class User {
17    constructor(username, email){
18      this.name = username
19      this.email = email
20    }
21
22    save(){
23      const db = getDb()
24      /**i wanna insert one new element
25       * and that new element will be 'this'
26       * so this javascript object we will in
27       * an object with a name and an email property
28       * this is what i wanna store as a user for now.
29       */
30
31      /**we can again use the 'then()' and 'catch()'
32       * or we just return that and let whoever calls this listen to that.

```

```

33     */
34     db.collection('users').insertOne(this)
35 }
36
37 static findById(userId){
38     const db = getDb()
39     /*i wanna use 'find()' to find a specific user
40     * the important thing here is that you need to convert user ID which i expect to be
a string to an ObjectId
41     * so let me import 'mongodb' by requiring 'mongodb'
42     *
43     * thanks to my constant up there
44     * and pass 'userId' to it.
45     *
46     * this should fine me all fitting users and i therefore get back a cursor.
47     * and now i can call 'next()' to get the first and as we know only element that
matters to us
48     * so i'm returning this here
49     */
50
51     /**as a side note,
52     * also use 'findOne()' if you sure that you only find one element
53     * this will now not give you a cursor
54     * but immediately return that one element.
55     * then this would be an alternative and i will use that here.
56     */
57     return db
58         .collection('users')
59         .find({_id: new ObjectId(userId)})
60         .next()
61 }
62 }
63
64
65 module.exports = User

```

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database').mongoConnect;
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   /** here in my middleware,
25    * where i find a user by ID
26    * i can search for that ID
27    * and i convert that in the user model.
28    * that is why i can use a string here.
29    */
30   User.findById('5cb7d12855fbe74b129c0b7c')
31     .then(user => {
32       req.user = user;
33       next();
34     })
35     .catch(err => console.log(err));
36   next();
37 });
38
39 app.use('/admin', adminRoutes);
40 app.use(shopRoutes);
41
42 app.use(errorController.get404);
43
44 mongoConnect(() => {
45   app.listen(3000);
46 });
47

```

* Chapter 191: Storing The User In Our Database

1. update

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

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

fasdfdas

Image URL

fadsfsa

Price

12

Description

fdafsfasfd

Add Product

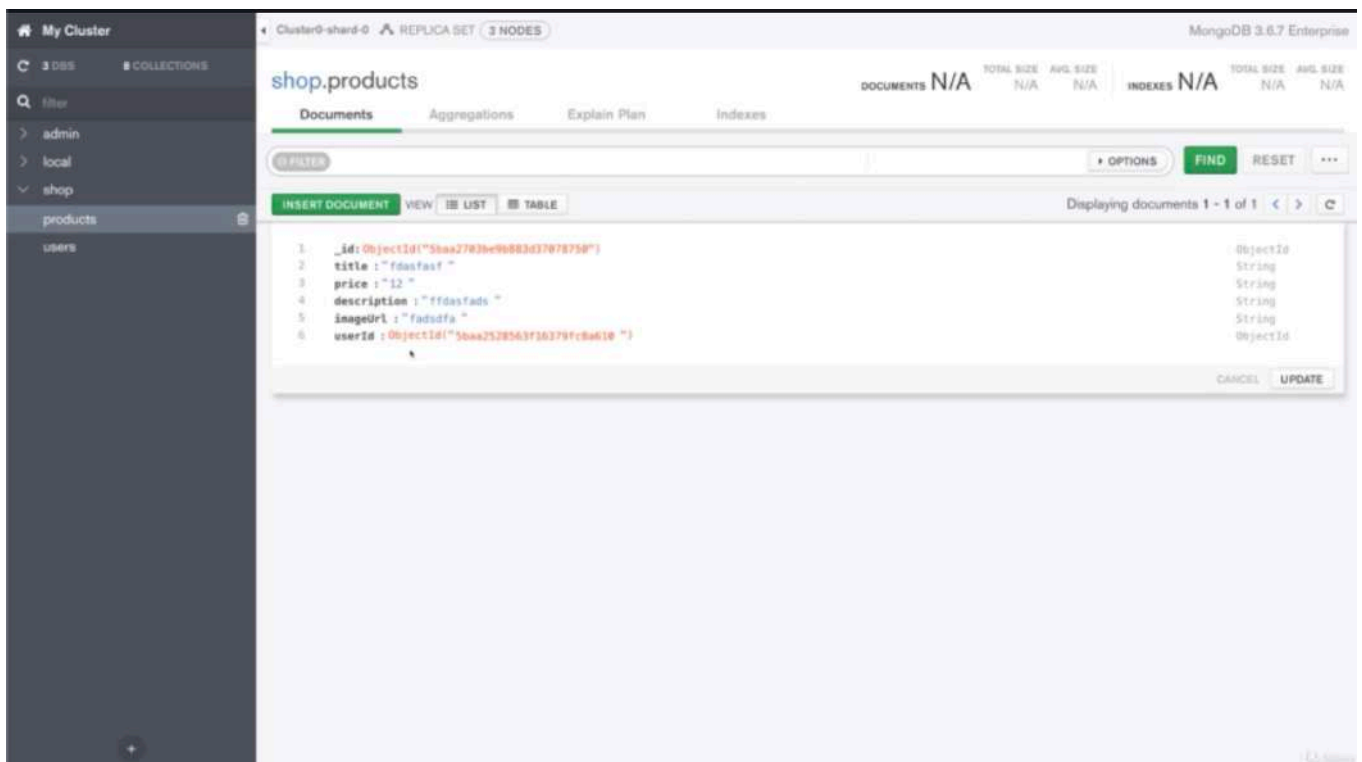
The screenshot shows a VS Code editor with the following components:

- EXPLORER:** A file tree on the left showing the project structure. The `models` folder is expanded, showing `cart-item.js`, `cart.js`, `order-item.js`, `order.js`, `product.js`, and `user.js`. The `app.js` file is selected in the `views` folder.
- EDITOR:** The `app.js` file is open, showing the following code:

```
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('5baa2528563f16379fc8a610')
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);
32
33 app.use(errorController.get404);
34
35 mongoConnect(() => {
36   app.listen(3000);
37 });
```
- TERMINAL:** The terminal at the bottom shows the output of the application, indicating that a user was found by ID:

```
name: 'Max',
email: 'max@test.com' }
{ _id: 5baa2528563f16379fc8a610,
  name: 'Max',
  email: 'max@test.com' }
```

- i just fetch the user with the ID which is string here, so that is valid.



- if we have a look at our products here, we see that products also have a user Id which is just a reference pointing at the user who did create that product which is one way of establishing relations.
- when we are fetching products, we don't really need any user information. hence we do it just like that.
- this will change once we start storing orders, it does make sense to store information about the user, for example the e-mail at least, and for the product you wanna store the title and the price maybe.

```

1 //./models/product.js
2
3 const mongodb = require('mongodb');
4 const getDb = require('./util/database').getDb;
5
6 /**i wanna use that user object when creating a new product
7  * when saving a product, i wanna store a reference to a user
8  * or embed the entire user data
9  * however for products in users, you could in arguments for both approaches here
10 * you don't wanna enclose all the user data in an embedded document
11 * because that would means that if the user data changes, you need to change that data in
    all products
12 * if you do include something which is unlikely to change very often like the username for
    example,
13 * then you could go ahead and embed that together with the ID
14 * so that you always have that ID to fetch more data about the user
15 * you have got to 'findById()' in the user model
16 * or that you have at least some snapshot data like the username available immediately
17 * if that should change, you need to update it everywhere.
18 *
19 * The alternative to this is that you store the ID
20 * so just a reference and therefore if you need connected data,
21 * you always have to fetch it manually from 2 collections
22 * but on the other hand, you might not do that too often
23 * and therefore here when i fetch the product,
24 * i don't really need the user data,
25 * we are not displaying the user name anywhere in our app
26 * i wanna store the user id so that we know who is connected even though we are not
    fetching that a lot

```

```

27 *
28 * what does this mean for our application here though?
29 * it means when creating a new product,
30 * we can pass the user id, we can accept the user
31 */
32 class Product {
33   constructor(title, price, description, imageUrl, id, userId) {
34     this.title = title;
35     this.price = price;
36     this.description = description;
37     this.imageUrl = imageUrl;
38     this._id = id ? new mongodb.ObjectId(id) : null
39     this.userId = userId
40   }
41
42   save() {
43     const db = getDb();
44     let dbOp;
45     if (this._id) {
46       // Update the product
47       dbOp = db
48         .collection('products')
49         .updateOne({ _id: this._id }, { $set: this });
50     } else {
51       dbOp = db.collection('products').insertOne(this);
52     }
53     return dbOp
54       .then(result => {
55         console.log(result);
56       })
57       .catch(err => {
58         console.log(err);
59       });
60   }
61
62   static fetchAll() {
63     const db = getDb();
64     return db
65       .collection('products')
66       .find()
67       .toArray()
68       .then(products => {
69         console.log(products);
70         return products;
71       })
72       .catch(err => {
73         console.log(err);
74       });
75   }
76
77   static findById(prodId) {
78     const db = getDb();
79     return db
80       .collection('products')
81       .find({ _id: new mongodb.ObjectId(prodId) })
82       .next()

```

```

83     .then(product => {
84         console.log(product);
85         return product;
86     })
87     .catch(err => {
88         console.log(err);
89     });
90 }
91
92 static deleteById(prodId) {
93     const db = getDb();
94     return db
95         .collection('products')
96         .deleteOne({ _id: new mongodb.ObjectId(prodId) })
97         .then(result => {
98             console.log('Deleted');
99         })
100        .catch(err => {
101            console.log(err);
102        });
103    }
104 }
105
106 module.exports = 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     /**when adding new products,
19      * i wanna pass 'null' for the product Id
20      * because we don't have that when creating a new product
21      * but for the user id,
22      * i wanna pass the ID of that user which we now store in our request
23      *
24      * 'req.user._id' will be just string
25      * because when retrieving data, the object id is converted to a string.
26      * so we have just the string here.
27      */
28     const product = new Product(
29         title,
30         price,
31         description,
32         imageUrl,

```

```

33     null,
34     req.user._id);
35 product
36     .save()
37     .then(result => {
38         // console.log(result);
39         console.log('Created Product');
40         res.redirect('/admin/products');
41     })
42     .catch(err => {
43         console.log(err);
44     });
45 };
46
47 exports.getEditProduct = (req, res, next) => {
48     const editMode = req.query.edit;
49     if (!editMode) {
50         return res.redirect('/');
51     }
52     const prodId = req.params.productId;
53     Product.findById(prodId)
54         // Product.findById(prodId)
55         .then(product => {
56             if (!product) {
57                 return res.redirect('/');
58             }
59             res.render('admin/edit-product', {
60                 pageTitle: 'Edit Product',
61                 path: '/admin/edit-product',
62                 editing: editMode,
63                 product: product
64             });
65         })
66         .catch(err => console.log(err));
67 };
68
69 exports.postEditProduct = (req, res, next) => {
70     const prodId = req.body.productId;
71     const updatedTitle = req.body.title;
72     const updatedPrice = req.body.price;
73     const updatedImageUrl = req.body.imageUrl;
74     const updatedDesc = req.body.description;
75
76     const product = new Product(
77         updatedTitle,
78         updatedPrice,
79         updatedDesc,
80         updatedImageUrl,
81         prodId
82     );
83     product
84         .save()
85         .then(result => {
86             console.log('UPDATED PRODUCT!');
87             res.redirect('/admin/products');
88         })

```

```

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

```

```

1  //./models/user.js
2
3  const mongodb = require('mongodb')
4  const getDb = require('../util/database').getDb
5
6  /** i will follow a slightly different approach
7   * which i already showed before
8   * and i will create an ObjectId constant
9   * and simply store access to it by accessing it up here.
10  * but i'm not calling it, i'm not creating an object in here.
11  * i'm storing the reference to the ObjectId class in my ObjectId constant
12  * and then down there, i can just write 'new ObjectId'
13  */
14  const ObjectId = mongodb.ObjectId
15
16  class User {
17      constructor(username, email){
18          this.name = username
19          this.email = email
20      }
21
22      save(){
23          const db = getDb()
24          /**i wanna insert one new element
25           * and that new element will be 'this'
26           * so this javascript object we will in
27           * an object with a name and an email property
28           * this is what i wanna store as a user for now.
29           */
30
31          /**we can again use the 'then()' and 'catch()'
32           * or we just return that and let whoever calls this listen to that.

```

```

33     */
34     db.collection('users').insertOne(this)
35 }
36
37 static findById(userId){
38     const db = getDb()
39     /**i wanna use 'find()' to find a specific user
40     * the important thing here is that you need to convert user ID which i expect to be
a string to an ObjectId
41     * so let me import 'mongoose' by requiring 'mongoose'
42     *
43     * thanks to my constant up there
44     * and pass 'userId' to it.
45     *
46     * this should find me all fitting users and i therefore get back a cursor.
47     * and now i can call 'next()' to get the first and as we know only element that
matters to us
48     * so i'm returning this here
49     */
50
51     /**as a side note,
52     * also use 'findOne()' if you sure that you only find one element
53     * this will now not give you a cursor
54     * but immediately return that one element.
55     * then this would be an alternative and i will use that here.
56     */
57     return db
58         .collection('users')
59         .findOne({_id: new ObjectId(userId)})
60         .then(user => {
61             console.log(user)
62             return user
63         })
64         .catch(err =>
65             console.log(err)
66         )
67 }
68 }
69
70
71 module.exports = User

```

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database').mongoConnect;
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   /** here in my middleware,
25    * where i find a user by ID
26    * i can search for that ID
27    * and i convert that in the user model.
28    * that is why i can use a string here.
29    */
30   User.findById('5cb7d12855fbe74b129c0b7c')
31     .then(user => {
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 mongoConnect(() => {
44   app.listen(3000);
45 });
46

```

* Chapter 192: Working On Cart Items & Orders

1. update

- delete ./models/cart.js, cart-item.js
- ./models/user.js

```

1 //./models/user.js
2
3 const mongodb = require('mongodb')
4 const getDb = require('../util/database').getDb
5
6 const ObjectId = mongodb.ObjectId
7
8 class User {
9   constructor(username, email, cart, id){
10     this.name = username
11     this.email = email
12     this.cart = cart //{items: []}
13     this._id = id
14   }
15
16   save(){
17     const db = getDb()
18     db.collection('users').insertOne(this)

```



```

19     }
20
21     addToCart(product){
22         /**you must not forget that 'addToCart' will be called on a user object
23         * and we will create that object with data we fetched from the database with help
of 'findById()'
24         *
25         * the cart will essentially be an object which has the items array
26         *
27         * i will pass a function to find index which is a function
28         * that javascript will execute for every element in the items array
29         * and here i wanna return true if i found the right product in my items array
30         * and this will be the case if 'cp' which is the product in the items array
31         *if 'cp._id' matches the _id of the product i'm trying to insert,
32         */
33         /*
34         const cartProduct = this.cart.item.findIndex(cp => {
35             return cp._id === prodId
36         })
37         */
38
39         /**you create a object with curly braces
40         * because we will add an object here
41         * and then you use the javascript spread operator
42         * 3 dots '...' to pull out all properties of this object
43         * so of the product object
44         * and then with a comma, you can add or overwrite a property
45         * and here i will add the quantity property and set it to 1
46         */
47         const updatedCart = {items: [{...product, quantity: 1}]}
48         const db = getDb()
49         db
50             .collection('users')
51             .updateOne(
52                 {_id: new ObjectId(this._id)},
53                 /**i will describe how to update and i will use '$set'
54                 * where i pass an object which holds all the information about which field
to update in which way.
55                 *
56                 * here i wanna keep everything as it is,
57                 * i don't wanna change the user name or anything like that.
58                 * so 'cart' which i expect to have in a user in the database will now
receive 'updatedCart'
59                 * so 'updatedCart' object as a new value which will overwrite the old one
60                 * and this is important. it will not merge this with the old one,
61                 * it will not merge the elements in the items array,
62                 * it will overwrite the old cart with the new cart.
63                 */
64                 {$set: {cart: updateCart}}
65             )
66     }
67
68     static findById(userId){
69         const db = getDb()
70         return db
71             .collection('users')

```

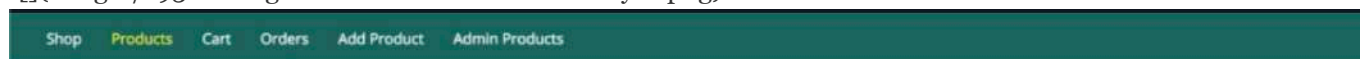
```

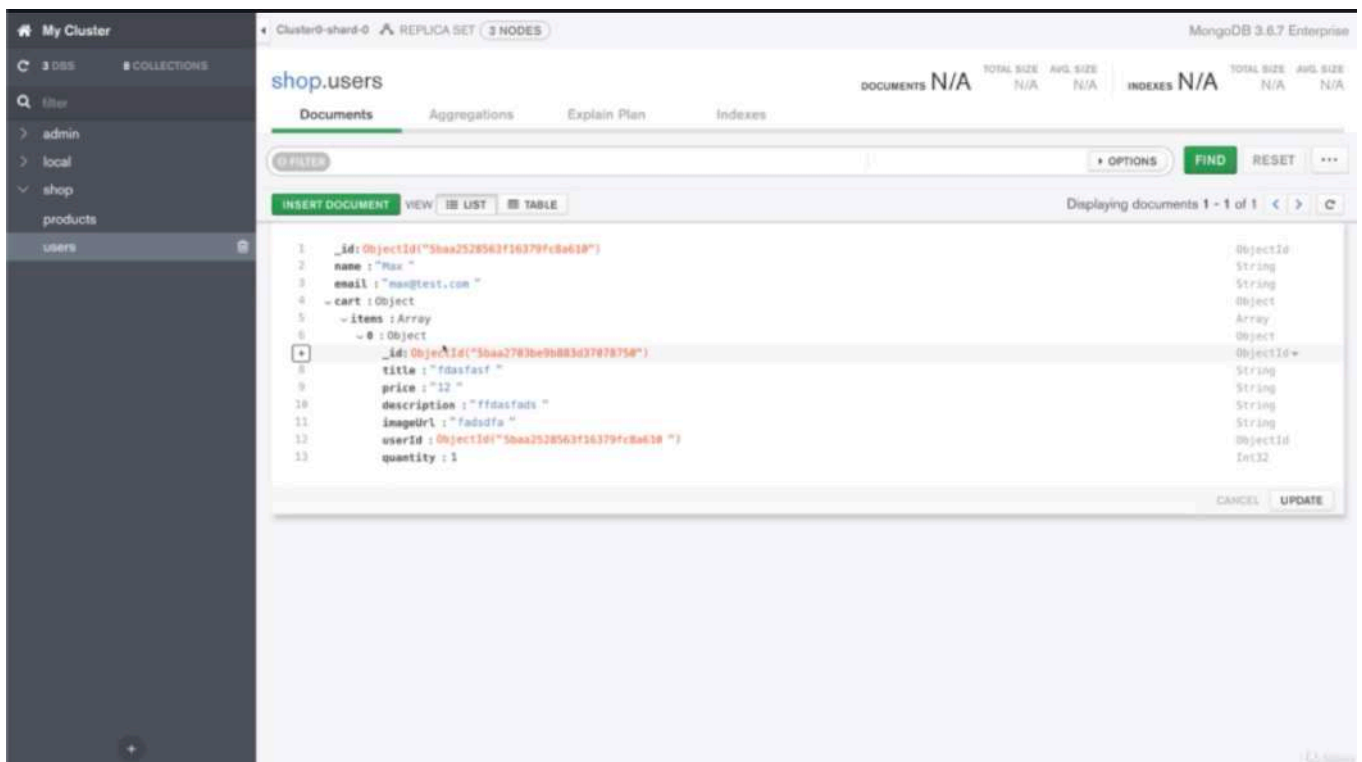
72     .findOne({_id: new ObjectId(userId)})
73     .then(user => {
74         console.log(user)
75         return user
76     })
77     .catch(err =>
78         console.log(err)
79     )
80 }
81 }
82
83
84 module.exports = User

```

* Chapter 193: Adding The “Add To Cart” Functionality

1. update
 - app.js
 - ./controllers/shop.js
 - ./routes/shop.js
 - ./views/includes/add-to-cart.ejs
 - ./models/user.js



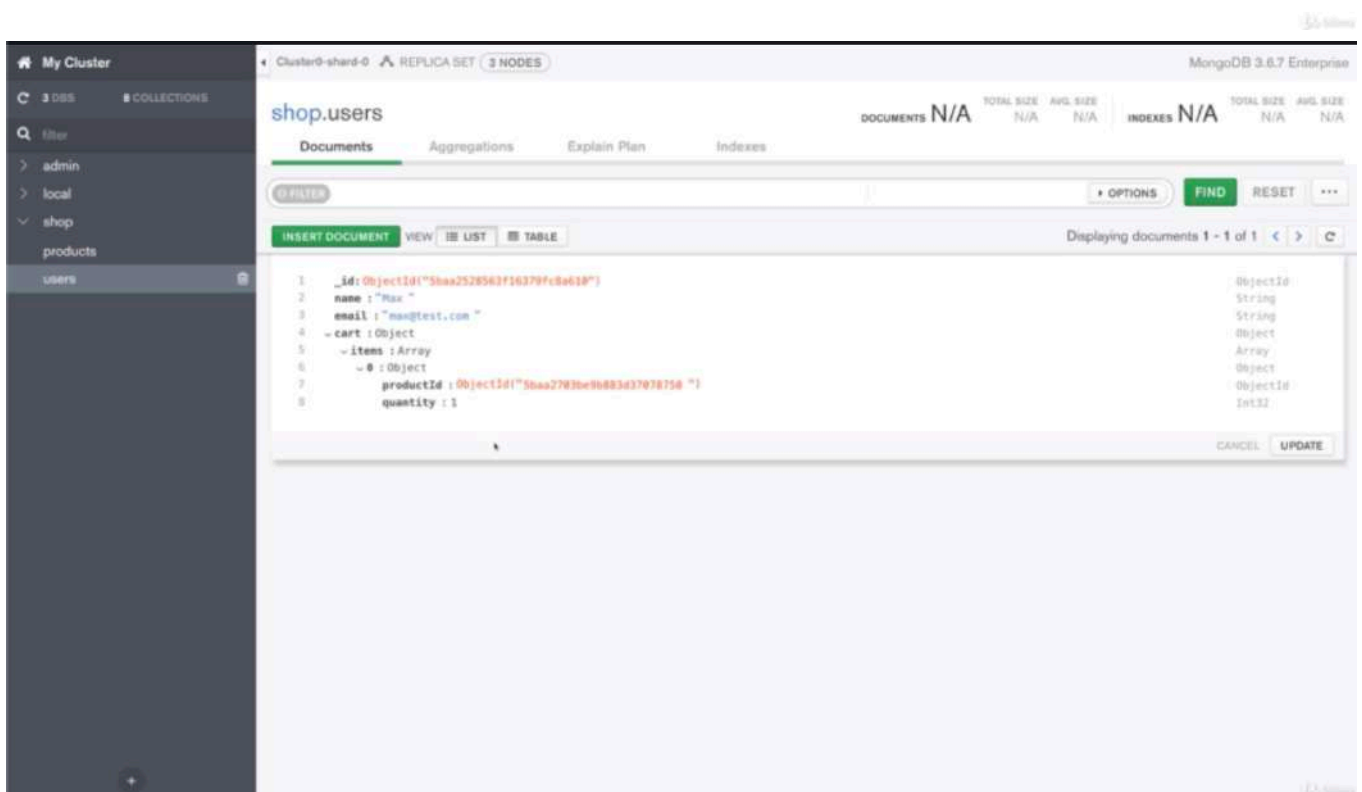


- we did modify something as our result tells us and if we go to MongoDB Compass and have a look at our users, you see there is an embedded document, a cart document with items with an object which holds product data. now that user ID here is a bit redundant because we're already in that user, we could strip that out and only store what we want but it also that doesn't matter too much.

- The important thing is that we now store a whole product which we store in a separate collection as part of an embedded document in our user.

- so we clearly have duplicate data here. we have the same product here as an embedded document and we have it in products.

- this is maybe something which we should change because if we change the product, if we change the title or the price, this will not be reflected in the cart and in the cart, we should have correct data because if the price changes, we can show the wrong price in our cart.



- if i click 'Add To Cart' and go to MongoDB Atlas and reload, then you see i'm only storing the reference and the quantity and this is the information i want.

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7
8 const errorController = require('./controllers/error');
9 const mongoConnect = require('./util/database').mongoConnect;
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('5cb7d12855f5e74b129c0b7c')
25     .then(user => {
26       /**it's important to understand that
27        * the user as i'm storing it will just be an object with the property
28        * so the data we have in the database
29        * all the methods of our ./models/user will not be in there
30        * because the user i'm getting here is data i'm getting out of the database
31        * and the methods are not stored there.
32        * they couldn't be stored there.
33        *
34        * to have a real user object with which we can interact,
35        * i should create a new user
36        * and pass 'user.name', 'user.email', 'user.cart', 'user._id'
37        * so i should create such a user object
38        * and store that in the request
39        * because now this enables me to work with all the user data
40        * or with the whole user model
41        * and this allows me to also call all these methods like 'addToCart()' on it. */
42       req.user = new User(user.name, user.email, user.cart, user._id);
43       next();
44     })
45     .catch(err => console.log(err));
46 });
47
48 app.use('/admin', adminRoutes);
49 app.use(shopRoutes);
50
51 app.use(errorController.get404);
52
53 mongoConnect(() => {
54   app.listen(3000);
55 });
56

```

```

1 //./models/user.js
2
3 const mongoose = require('mongoose')
4 const getDb = require('../util/database').getDb
5
6 const ObjectId = mongoose.Types.ObjectId
7
8 class User {
9   constructor(username, email, cart, id){
10     this.name = username
11     this.email = email

```

```

12     this.cart = cart //{items: []}
13     this._id = id
14 }
15
16 save(){
17     const db = getDb()
18     db.collection('users').insertOne(this)
19 }
20
21 addToCart(product){
22     /**i don't wanna store all product data in this object and a quantity
23     * i wanna store the product ID by creating a new ObjectId
24     * and passing product _Id as an argument and the quantity
25     * now only the reference and the quantity and not the rest of the data
26     */
27     const updatedCart = {
28         items: [
29             {
30                 productId: new ObjectId(product._id),
31                 quantity: 1
32             }
33         ]
34     }
35     const db = getDb()
36     db
37         .collection('users')
38         .updateOne(
39             {_id: new ObjectId(this._id)},
40             {$set: {cart: updatedCart}}
41         )
42
43     static findById(userId){
44         const db = getDb()
45         return db
46             .collection('users')
47             .findOne({_id: new ObjectId(userId)})
48             .then(user => {
49                 console.log(user)
50                 return user
51             })
52             .catch(err =>
53                 console.log(err)
54             )
55     }
56 }
57
58
59 module.exports = User

```



```

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

```

```

1  <!--./views/includes/add-to-cart.ejs-->
2
3  <form action="/cart" method="post">
4      <button class="btn" type="submit">Add to Cart</button>
5      <input type="hidden" name="productId" value="<%= product._id %>">
6  </form>
7

```

```

1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4
5  exports.getProducts = (req, res, next) => {
6      Product.fetchAll()
7          .then(products => {
8              res.render('shop/product-list', {
9                  prods: products,
10                 pageTitle: 'All Products',
11                 path: '/products'
12             });
13         })
14         .catch(err => {
15             console.log(err);
16         });
17 };
18
19 exports.getProduct = (req, res, next) => {
20     const prodId = req.params.productId;
21     // Product.findAll({ where: { id: prodId } })
22     // .then(products => {
23     //     res.render('shop/product-detail', {
24     //         product: products[0],
25     //         pageTitle: products[0].title,

```

```

26 //     path: '/products'
27 //   });
28 // })
29 // .catch(err => console.log(err));
30 Product.findById(prodId)
31   .then(product => {
32     res.render('shop/product-detail', {
33       product: product,
34       pageTitle: product.title,
35       path: '/products'
36     });
37   })
38   .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(cart => {
59       return cart
60         .getProducts()
61         .then(products => {
62           res.render('shop/cart', {
63             path: '/cart',
64             pageTitle: 'Your Cart',
65             products: products
66           });
67         })
68         .catch(err => console.log(err));
69     })
70     .catch(err => console.log(err));
71 };
72
73 exports.postCart = (req, res, next) => {
74   const prodId = req.body.productId;
75   Product
76     .findById(prodId)
77     .then(product => {
78       /** pass in 'product' in 'addToCart()'
79        * because in the ./models/user.js file,
80        * 'addToCart()' does expect 'product'
81        * and then i return the result of 'updateOne' which will be a promise.

```



```

82     *
83     */
84     return req.user.addToCart(product)
85   })
86   .then(result => {
87     console.log(result)
88   })
89   .catch(err => {
90     console.log(err)
91   })
92   /*
93   let fetchedCart;
94   let newQuantity = 1;
95   req.user
96     .getCart()
97     .then(cart => {
98       fetchedCart = cart;
99       return cart.getProducts({ where: { id: prodId } });
100     })
101     .then(products => {
102       let product;
103       if (products.length > 0) {
104         product = products[0];
105       }
106
107       if (product) {
108         const oldQuantity = product.cartItem.quantity;
109         newQuantity = oldQuantity + 1;
110         return product;
111       }
112       return Product.findByPk(prodId);
113     })
114     .then(product => {
115       return fetchedCart.addProduct(product, {
116         through: { quantity: newQuantity }
117       });
118     })
119     .then(() => {
120       res.redirect('/cart');
121     })
122     .catch(err => console.log(err));
123   */
124 };
125
126 exports.postCartDeleteProduct = (req, res, next) => {
127   const prodId = req.body.productId;
128   req.user
129     .getCart()
130     .then(cart => {
131       return cart.getProducts({ where: { id: prodId } });
132     })
133     .then(products => {
134       const product = products[0];
135       return product.cartItem.destroy();
136     })
137     .then(result => {

```

```

138     res.redirect('/cart');
139   })
140   .catch(err => console.log(err));
141 };
142
143 exports.postOrder = (req, res, next) => {
144   let fetchedCart;
145   req.user
146     .getCart()
147     .then(cart => {
148       fetchedCart = cart;
149       return cart.getProducts();
150     })
151     .then(products => {
152       return req.user
153         .createOrder()
154         .then(order => {
155           return order.addProducts(
156             products.map(product => {
157               product.orderItem = { quantity: product.cartItem.quantity };
158               return product;
159             })
160           );
161         })
162         .catch(err => console.log(err));
163     })
164     .then(result => {
165       return fetchedCart.setProducts(null);
166     })
167     .then(result => {
168       res.redirect('/orders');
169     })
170     .catch(err => console.log(err));
171 };
172
173 exports.getOrders = (req, res, next) => {
174   req.user
175     .getOrders({include: ['products']})
176     .then(orders => {
177       res.render('shop/orders', {
178         path: '/orders',
179         pageTitle: 'Your Orders',
180         orders: orders
181       });
182     })
183     .catch(err => console.log(err));
184 };
185

```

* Chapter 194: Storing Multiple Products In The Cart

1. update
- ./models/user.js

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Title

Second

Image URL

fafdsa

Price

555


Description

fadsfadsfs

Add Product

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

fdasfasf




\$ 12

ffdasfads

Edit

Delete

Second



\$ 555

fadsfadsfs

Edit

Delete

fdasfasf



\$ 12

ffdasfads

Details

Add to Cart

Second



\$ 555

fadsfadsfs

Details

Add to Cart

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES

MongoDB 3.6.7 Enterprise

3 DBS

COLLECTIONS

filter

admin

local

shop

products

users

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

2

3

4

5

6

7

8

9

10

11

ObjectId

String

String

Object

Array

Object

Object

Object

Object

Object

Object

Int32

CANCEL

UPDATE

My Cluster

3 DBS

COLLECTIONS

filter

admin

local

shop

products

users

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

Filter

Options

Find

Reset

...

Insert Document

View

List

Table

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5baa2528963f26379fc8a618"),
  "name": "Max",
  "email": "max@test.com",
  "cart": {
    "items": Array
    - 0: Object
      productId: ObjectId("5baa27830e9a883d37878756")
      quantity: 2
  }
}
```

ShopProductsCartOrdersAdd ProductAdmin Products

fdasfasf



\$ 12

ffdasfads

Details

Add to Cart

Second



\$ 555

fadsfdasfs

Details

Add to Cart

My Cluster

3 DBS

COLLECTIONS

filter

admin

local

shop

products

users

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

Filter

Options

Find

Reset

More

Insert Document

View

List

Table

Displaying documents 1 - 1 of 1

<

>

Copy

Share

Print

```
{ "_id": "ObjectId(\"5baa2528563f16379fc8a618\")",
  "name": "Max",
  "email": "max@test.com",
  "cart": {
    "items": [
      {
        "product": {
          "productId": "ObjectId(\"5baa27830e9b883d37878756\")",
          "quantity": 2
        },
        "product": {
          "productId": "ObjectId(\"5baa2d3b79f7e737f82b449a\")",
          "quantity": 1
        }
      ]
    }
  }
}
```

ShopProductsCartOrdersAdd ProductAdmin Products

fdasfasf




\$ 12

ffdasfads

Details

Add to Cart

Second

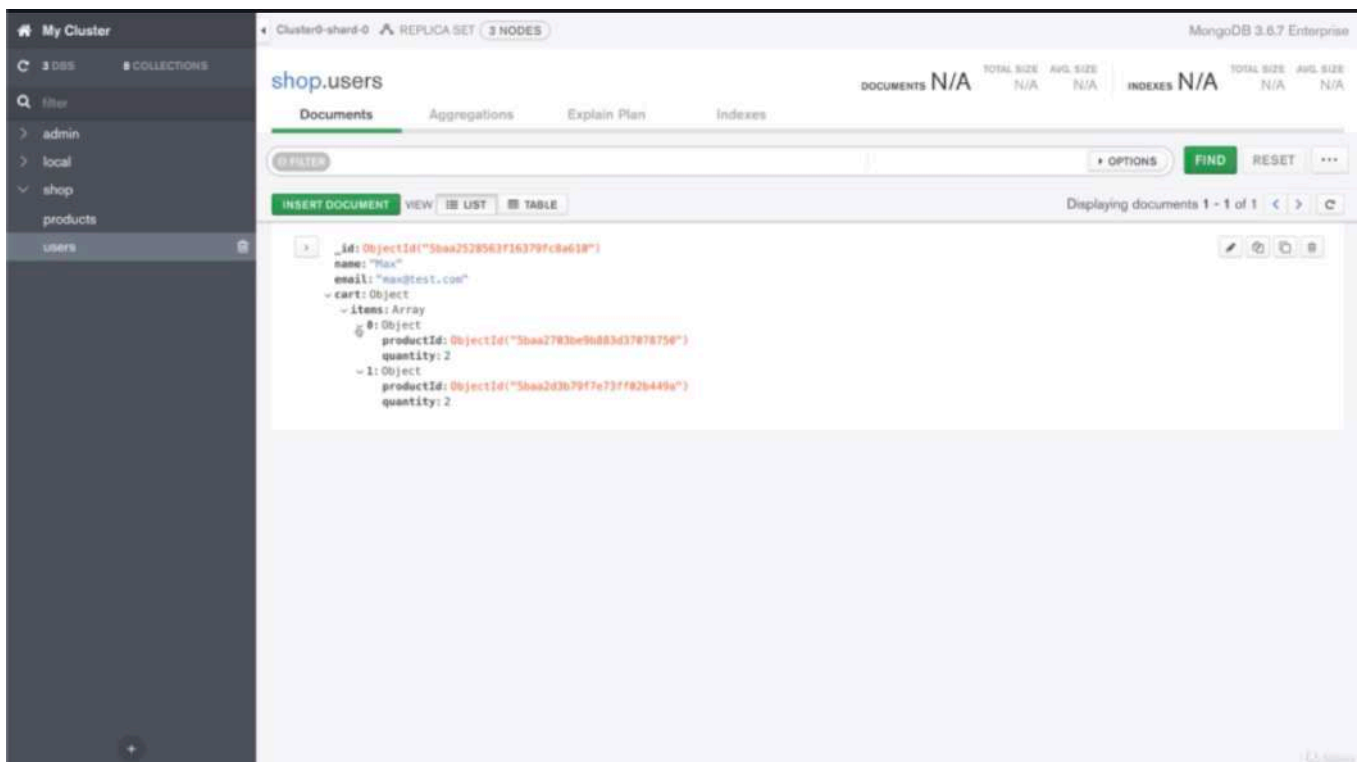


\$ 555

fadsfdasfs

Details

Add to Cart



```

1  ../models/user.js
2
3  const mongodb = require('mongodb');
4  const getDb = require('../util/database').getDb;
5
6  const ObjectId = mongodb.ObjectId;
7
8  class User {
9    constructor(username, email, cart, id) {
10      this.name = username;
11      this.email = email;
12      this.cart = cart; // {items: []}
13      this._id = id;
14    }
15
16    save() {
17      const db = getDb();
18      return db.collection('users').insertOne(this);
19    }
20
21    addToCart(product) {
22      const cartProductIndex = this.cart.items.findIndex(cp => {
23        /**'product' in 'addToCart(product)' is a 'product' i just retrieved from the database
24         * '_id' in here is treated as a string in javascript
25         * but is not exactly of type string
26         * since i'm using 3 equal signs===' in my check here
27         * however i'm telling javascript that this should only return true
28         * if these 2 don't only match by value but also by type
29         * and technically '_id' is not string even though we can use it as such.
30         *
31         * so one solution is to use 2 equal signs or to use 'toString()' on both here.
32         * make sure we only works with strings here in both cases.
33         */
34        return cp.productId.toString() === product._id.toString();
35      });

```

```

36     let newQuantity = 1;
37     /**this gives me a new array with all the items that are in the cart
38      * and they are now stored here.
39      *
40      * now i can edit my updatedCartItems
41      * and now i need to differentiate,
42      * do we already have that item in cart or not
43      * so 'updatedCartItems' position is here, before my if check
44      */
45     const updatedCartItems = [...this.cart.items];
46     /**if i make it into this if statement,
47      * i know that we have this product already.
48      * i can access updatedCartItems for the cartProductIndex i found
49      * now i have access to that item i'm interested in.
50      * i know it already existed
51      * so i can set its quantity equal to the 'newQuantity'
52      */
53     if (cartProductIndex >= 0) {
54         newQuantity = this.cart.items[cartProductIndex].quantity + 1;
55         /**and i can edit this array without touching the old array
56          * due to the way javascript works with reference and primitive types
57          */
58         updatedCartItems[cartProductIndex].quantity = newQuantity;
59     } else {
60         /**if the item didn't exist before,
61          * i will take my updatedCartItems and add a new one with 'push()'
62          * i will add a new cartItem and i will add a new cartItem
63          * which looks exactly as described down there. '{ productId: new ObjectId(product._id),
quantity: newQuantity }'
64          * so i will grab that code and add it here.
65          */
66         updatedCartItems.push({
67             productId: new ObjectId(product._id),
68             quantity: newQuantity
69         });
70     }
71     const updatedCart = {
72         /**i can always set my items equal to the updatedCartItems
73          * because that will always be an array with all the old elements
74         '[...this.cart.items]'
75         * because i copy '[...this.cart.items]' first and then with the update added
76         * so either with the quantity increased for the existing element
77         * or with a new element added to the cart.
78         *
79         * so i can safely have my updatedCart down there and save that to the database with
all the updatedItems in there.
80         */
81         items: updatedCartItems
82     };
83     const db = getDb();
84     return db
85         .collection('users')
86         .updateOne(
87             { _id: new ObjectId(this._id) },
88             { $set: { cart: updatedCart } }
89         );

```



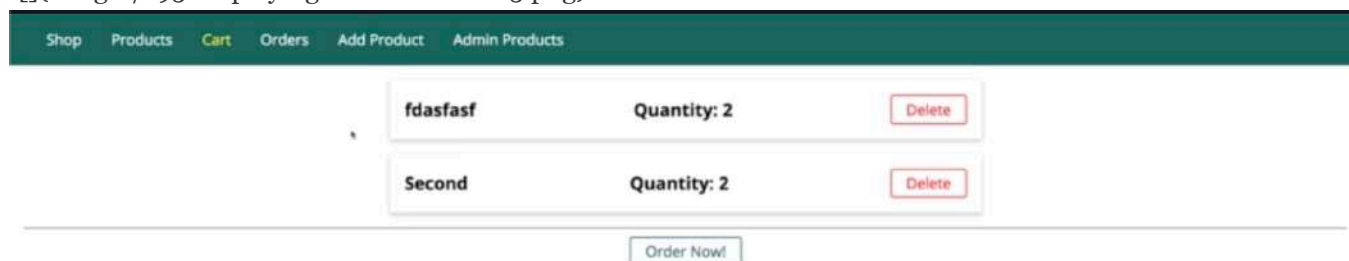
```

89   }
90
91   static findById(userId) {
92     const db = getDb();
93     return db
94       .collection('users')
95       .findOne({ _id: new ObjectId(userId) })
96       .then(user => {
97         console.log(user);
98         return user;
99       })
100    .catch(err => {
101      console.log(err);
102    });
103  }
104 }
105
106 module.exports = User;

```

* Chapter 195: Displaying The Cart Items

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



Title
Third!


Image URL
http://ichef.bbci.co.uk/ww/features/wm/live

Price
12

Description
fdasfdasf

Add Product

fdasfasf




\$ 12

ffdasfads

Edit

Delete

Second




\$ 555

fadsfdasfs

Edit

Delete

Third!

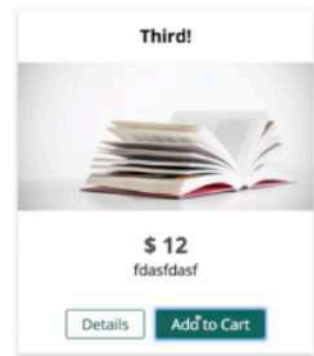
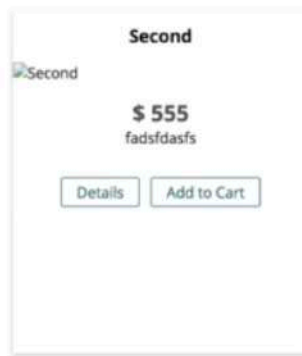
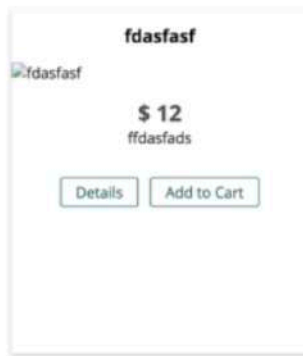


\$ 12

fdasfdasf

Edit

Delete



35 items

fdasfasf	Quantity: 2	Delete
Second	Quantity: 2	Delete
Third!	Quantity: 1	Delete

Order Now!

35 items

```

1  ../models/user.js
2
3  const mongodb = require('mongodb');
4  const getDb = require('../util/database').getDb;
5
6  const ObjectId = mongodb.ObjectId;
7
8  class User {
9    constructor(username, email, cart, id) {
10      this.name = username;
11      this.email = email;
12      this.cart = cart; // {items: []}
13      this._id = id;
14    }

```

```

15
16 save() {
17     const db = getDb();
18     return db.collection('users').insertOne(this);
19 }
20
21 addToCart(product) {
22     const cartProductIndex = this.cart.items.findIndex(cp => {
23         return cp.productId.toString() === product._id.toString();
24     });
25     let newQuantity = 1;
26     const updatedCartItems = [...this.cart.items];
27     if (cartProductIndex >= 0) {
28         newQuantity = this.cart.items[cartProductIndex].quantity + 1;
29         updatedCartItems[cartProductIndex].quantity = newQuantity;
30     } else {
31         updatedCartItems.push({
32             productId: new ObjectId(product._id),
33             quantity: newQuantity
34         });
35     }
36     const updatedCart = {
37         items: updatedCartItems
38     };
39     const db = getDb();
40     return db
41         .collection('users')
42         .updateOne(
43             { _id: new ObjectId(this._id) },
44             { $set: { cart: updatedCart } }
45         );
46 }
47 /**'getCart()' should return products which are enriched with all the data that is stored in
48 a product's collection
49 * because in users, in the cart we will only store the reference
50 * and this is what we need to do in MongoDB
51 * if we then have a connection between 2 collections with a reference,
52 * we need to merge them manually as we are doing it here
53 * with that merging being done manually,
54 * we can now use that data.
55 * so 'getCart()' should return a cart with all the information we need.
56 */
57 getCart() {
58     const db = getDb();
59     /**object have a productId and quantity
60     * but we are only interested in product ID
61     * so 'map()' this, this is a default javascript function
62     * i will map this to transform every item in there
63     * and i wanna return the product ID
64     *
65     * so what i'm doing is that
66     * i'm mapping an array of items where every item is a javascript object into an array
67 of just string, of the product IDs
68     * and this is then stored in this new 'productIds' constant
69 */

```

```

69     const productIds = this.cart.items.map(i => {
70         return i.productId;
71     });
72     /**i reach out to the products collection
73     * now because i have all the user data
74     * i have all the cart data
75     * now i need to fill it with some live from the products database
76     *
77     * in there, i wanna find all products that are in my cart
78     * for this, we can use special query syntax MongoDB supports.
79     * in 'find()', i can tell i wanna find all products where '_id' is equal to
80     * and i don't pass an ID here because i'm not looking for a single ID
81     * instead i pass an object because this allows me to use some special MongoDB query
operator
82     * we are looking for '$in' operator
83     * and this operator takes an array of IDs
84     * and therefore every ID which is in the array will be accepted
85     * and will get back a cursor which holds references to all products with one of the IDs
mentioned in this array
86     *
87     * then my array which i wanna use to tell MongoDB
88     * give me all elements where the ID is one of the IDs mentioned in this array here.
89     * this gives me a cursor with all the matching products
90     *
91     * and i will again use 'toArray()' to get quickly that converted to a javascript array
92     * and then i will add 'then()' method
93     * in 'then()' method, i will have all my product data for the products that were in my
cart
94     * and of course we wanna add the quantity back to every product
95     * because that is something that is important to us
96     */
97     return db
98         .collection('products')
99         .find({ _id: { $in: productIds } })
100         .toArray()
101         .then(products => {
102             return products.map(p => {
103                 /**i wanna keep all the data i retrieved
104                 * but then i will add a new quantity property
105                 * and that quantity property needs to be populated with data i have on that
product
106                 * we have the products stored in the cart of this user
107                 * and make sure you use arrow function to ensure that 'this' inside of this
functio still refers to the overall class
108                 * with normal functions, it would not
109                 * and access my items and find the item with that ID at hand here
110                 */
111                 return {
112                     ...p,
113                     quantity: this.cart.items.find(i => {
114                         /**in the end, i have an array of 'products'
115                         * fresh from the database
116                         * then i wanna transform this which i'm doing with 'map()'
117                         * 'map()' takes a function that executes on every element and products
118                         * which describes how to transform this element
119                         * and in 'p', i'm returning the new value which is an object

```

```

120     * where i still have all the old product properties
121     * but i add a new quantity property and to get the right quantity for that
given product,
122     * i reach out to 'cart.items' which exist on that user
123     * and i again use a built in javascript method,
124     * find to look at all elements in cart.items with this function in 'i'
125     * then identify the one product where the productId i'm storing in my
cart.items matches the ID of the product i have fetched from the database
126     * and since with 'map()' i'm going through all these products,
127     * this will also vary for every run.
128     *
129     * now from the cart.items i have, i extract a quantity for that given product.
130     */
131     return i.productId.toString() === p._id.toString();
132   }).quantity
133   };
134   });
135   });
136   }
137
138   static findById(userId) {
139     const db = getDb();
140     return db
141       .collection('users')
142       .findOne({ _id: new ObjectId(userId) })
143       .then(user => {
144         console.log(user);
145         return user;
146       })
147       .catch(err => {
148         console.log(err);
149       });
150   }
151 }
152
153 module.exports = User;

```

```

1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4
5  exports.getProducts = (req, res, next) => {
6    Product.fetchAll()
7      .then(products => {
8        res.render('shop/product-list', {
9          prods: products,
10         pageTitle: 'All Products',
11         path: '/products'
12       });
13     })
14     .catch(err => {
15       console.log(err);
16     });
17   };
18
19   exports.getProduct = (req, res, next) => {
20     const prodId = req.params.productId;

```

```

21 // Product.findAll({ where: { id: prodId } })
22 //   .then(products => {
23 //     res.render('shop/product-detail', {
24 //       product: products[0],
25 //       pageTitle: products[0].title,
26 //       path: '/products'
27 //     });
28 //   })
29 //   .catch(err => console.log(err));
30 Product.findById(prodId)
31   .then(product => {
32     res.render('shop/product-detail', {
33       product: product,
34       pageTitle: product.title,
35       path: '/products'
36     });
37   })
38   .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(products => {
59       res.render('shop/cart', {
60         path: '/cart',
61         pageTitle: 'Your Cart',
62         products: products
63       });
64     })
65     .catch(err => console.log(err));
66 };
67
68 exports.postCart = (req, res, next) => {
69   const prodId = req.body.productId;
70   Product.findById(prodId)
71     .then(product => {
72       return req.user.addToCart(product);
73     })
74     .then(result => {
75       console.log(result);
76       res.redirect('/cart');

```

```

77     });
78     // let fetchedCart;
79     // let newQuantity = 1;
80     // req.user
81     //   .getCart()
82     //   .then(cart => {
83     //     fetchedCart = cart;
84     //     return cart.getProducts({ where: { id: prodId } });
85     //   })
86     //   .then(products => {
87     //     let product;
88     //     if (products.length > 0) {
89     //       product = products[0];
90     //     }
91
92     //     if (product) {
93     //       const oldQuantity = product.cartItem.quantity;
94     //       newQuantity = oldQuantity + 1;
95     //       return product;
96     //     }
97     //     return Product.findById(prodId);
98     //   })
99     //   .then(product => {
100    //     return fetchedCart.addProduct(product, {
101    //       through: { quantity: newQuantity }
102    //     });
103    //   })
104    //   .then(() => {
105    //     res.redirect('/cart');
106    //   })
107    //   .catch(err => console.log(err));
108  });
109
110  exports.postCartDeleteProduct = (req, res, next) => {
111    const prodId = req.body.productId;
112    req.user
113      .getCart()
114      .then(cart => {
115        return cart.getProducts({ where: { id: prodId } });
116      })
117      .then(products => {
118        const product = products[0];
119        return product.cartItem.destroy();
120      })
121      .then(result => {
122        res.redirect('/cart');
123      })
124      .catch(err => console.log(err));
125  });
126
127  exports.postOrder = (req, res, next) => {
128    let fetchedCart;
129    req.user
130      .getCart()
131      .then(cart => {
132        fetchedCart = cart;

```



```

133     return cart.getProducts();
134   })
135   .then(products => {
136     return req.user
137       .createOrder()
138       .then(order => {
139         return order.addProducts(
140           products.map(product => {
141             product.orderItem = { quantity: product.cartItem.quantity };
142             return product;
143           })
144         );
145       })
146       .catch(err => console.log(err));
147   })
148   .then(result => {
149     return fetchedCart.setProducts(null);
150   })
151   .then(result => {
152     res.redirect('/orders');
153   })
154   .catch(err => console.log(err));
155 };
156
157 exports.getOrders = (req, res, next) => {
158   req.user
159     .getOrders({ include: ['products'] })
160     .then(orders => {
161       res.render('shop/orders', {
162         path: '/orders',
163         pageTitle: 'Your Orders',
164         orders: orders
165       });
166     })
167     .catch(err => console.log(err));
168 };
169

```

```

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         <% products.forEach(p => { %>
13           <li class="cart__item">
14             <h1><%= p.title %></h1>
15             <h2>Quantity: <%= p.quantity %></h2>
16             <form action="/cart-delete-item" method="POST">
17               <input type="hidden" value="<%= p._id %>" name="productId">
18               <button class="btn danger" type="submit">Delete</button>
19             </form>

```

```

20         </li>
21     <% }) %>
22 </ul>
23 <hr>
24 <div class="centered">
25     <form action="/create-order" method="POST">
26         <button type="submit" class="btn">Order Now!</button>
27     </form>
28 </div>
29
30 <% } else { %>
31     <h1>No Products in Cart!</h1>
32 <% } %>
33 </main>
34 <%- include('../includes/end.ejs') %>

```

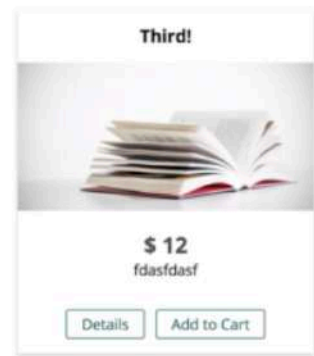
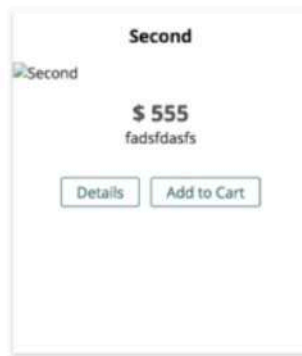
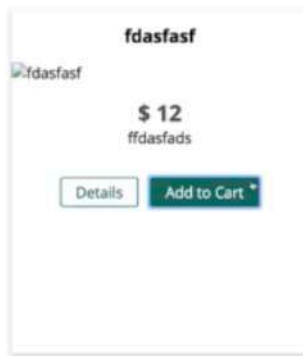
```

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;

```

* Chapter 196: Fixing A Bug

1. update
- ./controllers/shop.js



35 slides

fdasfasf	Quantity: 4	Delete
Second	Quantity: 2	Delete
Third!	Quantity: 1	Delete

Order Now!

36 slides

- if i click 'Add to cart', then redirect to '/cart'

```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.fetchAll()
7     .then(products => {
8       res.render('shop/product-list', {
9         prods: products,
10        pageTitle: 'All Products',
11        path: '/products'
12      });
13    })
  
```

```

14     .catch(err => {
15         console.log(err);
16     });
17 };
18
19 exports.getProduct = (req, res, next) => {
20     const prodId = req.params.productId;
21     // Product.findAll({ where: { id: prodId } })
22     // .then(products => {
23     //     res.render('shop/product-detail', {
24     //         product: products[0],
25     //         pageTitle: products[0].title,
26     //         path: '/products'
27     //     });
28     // })
29     // .catch(err => console.log(err));
30     Product.findById(prodId)
31     .then(product => {
32         res.render('shop/product-detail', {
33             product: product,
34             pageTitle: product.title,
35             path: '/products'
36         });
37     })
38     .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42     Product.fetchAll()
43     .then(products => {
44         res.render('shop/index', {
45             prods: products,
46             pageTitle: 'Shop',
47             path: '/'
48         });
49     })
50     .catch(err => {
51         console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56     req.user
57     .getCart()
58     .then(products => {
59         res.render('shop/cart', {
60             path: '/cart',
61             pageTitle: 'Your Cart',
62             products: products
63         });
64     })
65     .catch(err => console.log(err));
66 };
67
68 exports.postCart = (req, res, next) => {
69     const prodId = req.body.productId;

```

```

70 Product.findById(prodId)
71   .then(product => {
72     return req.user.addToCart(product);
73   })
74   .then(result => {
75     console.log(result);
76     res.redirect('/cart');
77   });
78 // let fetchedCart;
79 // let newQuantity = 1;
80 // req.user
81 //   .getCart()
82 //   .then(cart => {
83 //     fetchedCart = cart;
84 //     return cart.getProducts({ where: { id: prodId } });
85 //   })
86 //   .then(products => {
87 //     let product;
88 //     if (products.length > 0) {
89 //       product = products[0];
90 //     }
91
92 //     if (product) {
93 //       const oldQuantity = product.cartItem.quantity;
94 //       newQuantity = oldQuantity + 1;
95 //       return product;
96 //     }
97 //     return Product.findById(prodId);
98 //   })
99 //   .then(product => {
100 //     return fetchedCart.addProduct(product, {
101 //       through: { quantity: newQuantity }
102 //     });
103 //   })
104 //   .then(() => {
105 //     res.redirect('/cart');
106 //   })
107 //   .catch(err => console.log(err));
108 };
109
110 exports.postCartDeleteProduct = (req, res, next) => {
111   const prodId = req.body.productId;
112   req.user
113     .getCart()
114     .then(cart => {
115       return cart.getProducts({ where: { id: prodId } });
116     })
117     .then(products => {
118       const product = products[0];
119       return product.cartItem.destroy();
120     })
121     .then(result => {
122       res.redirect('/cart');
123     })
124     .catch(err => console.log(err));
125 };

```

```

126
127 exports.postOrder = (req, res, next) => {
128   let fetchedCart;
129   req.user
130     .getCart()
131     .then(cart => {
132       fetchedCart = cart;
133       return cart.getProducts();
134     })
135     .then(products => {
136       return req.user
137         .createOrder()
138         .then(order => {
139           return order.addProducts(
140             products.map(product => {
141               product.orderItem = { quantity: product.cartItem.quantity };
142               return product;
143             })
144           );
145         })
146         .catch(err => console.log(err));
147     })
148     .then(result => {
149       return fetchedCart.setProducts(null);
150     })
151     .then(result => {
152       res.redirect('/orders');
153     })
154     .catch(err => console.log(err));
155   };
156
157 exports.getOrders = (req, res, next) => {
158   req.user
159     .getOrders({ include: ['products'] })
160     .then(orders => {
161       res.render('shop/orders', {
162         path: '/orders',
163         pageTitle: 'Your Orders',
164         orders: orders
165       });
166     })
167     .catch(err => console.log(err));
168   };
169

```

* Chapter 197: Deleting Cart Items

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

fdasfasf	Quantity: 4	Delete
Second	Quantity: 2	Delete
Third!	Quantity: 1	Delete

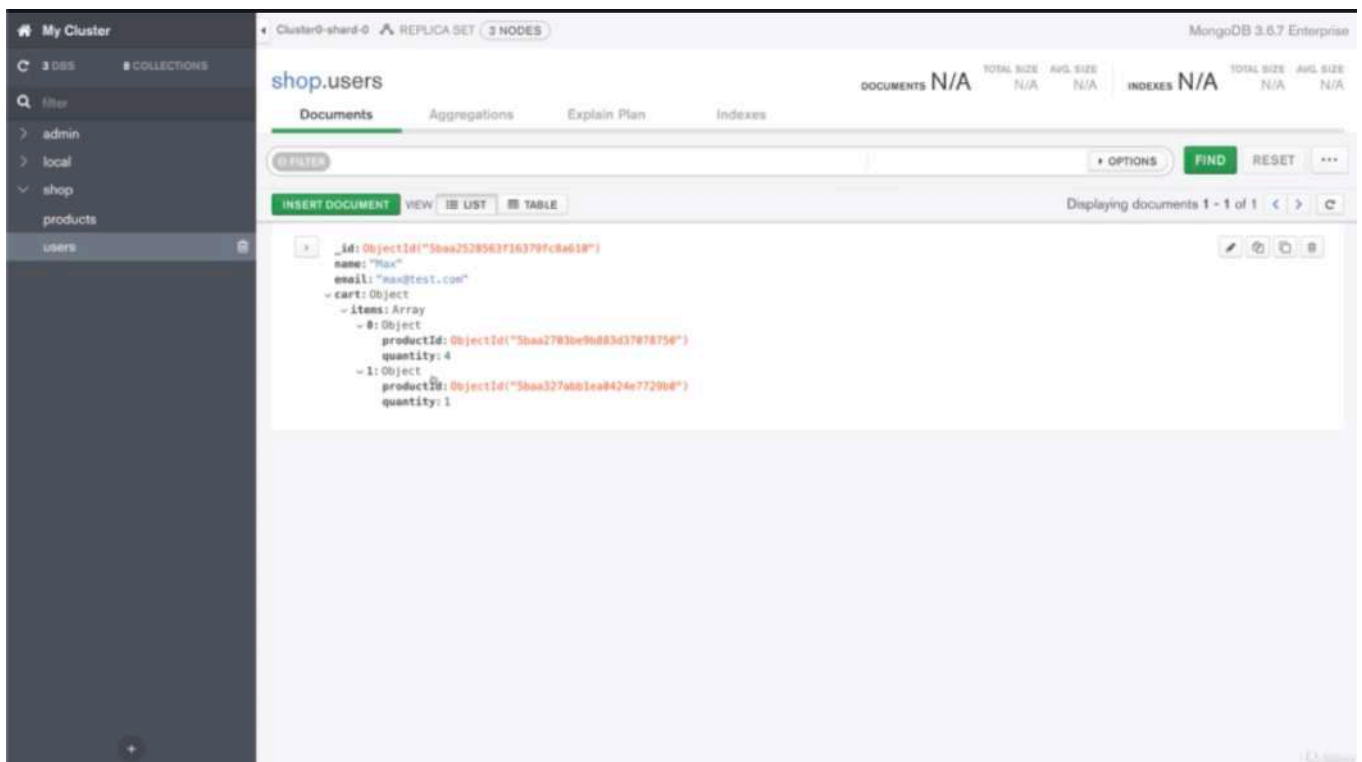
Order Now!

3 items

fdasfasf	Quantity: 4	Delete
Third!	Quantity: 1	Delete

Order Now!

2 items



```

1  ../controllers/shop.js
2
3  const Product = require('../models/product');
4
5  exports.getProducts = (req, res, next) => {
6    Product.fetchAll()
7      .then(products => {
8        res.render('shop/product-list', {
9          prods: products,
10         pageTitle: 'All Products',
11         path: '/products'
12       });
13     })
14    .catch(err => {
15      console.log(err);
16    });
17  };
18
19  exports.getProduct = (req, res, next) => {
20    const prodId = req.params.productId;
21    // Product.findAll({ where: { id: prodId } })
22    // .then(products => {
23    //   res.render('shop/product-detail', {
24    //     product: products[0],
25    //     pageTitle: products[0].title,
26    //     path: '/products'
27    //   });
28    // })
29    // .catch(err => console.log(err));
30    Product.findById(prodId)
31      .then(product => {
32        res.render('shop/product-detail', {
33          product: product,
34          pageTitle: product.title,
35          path: '/products'

```



```

36     });
37   })
38   .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(products => {
59       res.render('shop/cart', {
60         path: '/cart',
61         pageTitle: 'Your Cart',
62         products: products
63       });
64     })
65     .catch(err => console.log(err));
66 };
67
68 exports.postCart = (req, res, next) => {
69   const prodId = req.body.productId;
70   Product.findById(prodId)
71     .then(product => {
72       return req.user.addToCart(product);
73     })
74     .then(result => {
75       console.log(result);
76       res.redirect('/cart');
77     });
78 };
79
80 exports.postCartDeleteProduct = (req, res, next) => {
81   const prodId = req.body.productId;
82   req.user
83     .deleteItemFromCart(prodId)
84     .then(result => {
85       res.redirect('/cart');
86     })
87     .catch(err => console.log(err));
88 };
89
90 exports.postOrder = (req, res, next) => {
91   let fetchedCart;

```

```

92 req.user
93   .getCart()
94   .then(cart => {
95     fetchedCart = cart;
96     return cart.getProducts();
97   })
98   .then(products => {
99     return req.user
100       .createOrder()
101       .then(order => {
102         return order.addProducts(
103           products.map(product => {
104             product.orderItem = { quantity: product.cartItem.quantity };
105             return product;
106           })
107         );
108       })
109       .catch(err => console.log(err));
110   })
111   .then(result => {
112     return fetchedCart.setProducts(null);
113   })
114   .then(result => {
115     res.redirect('/orders');
116   })
117   .catch(err => console.log(err));
118 });
119
120 exports.getOrders = (req, res, next) => {
121   req.user
122     .getOrders({ include: ['products'] })
123     .then(orders => {
124       res.render('shop/orders', {
125         path: '/orders',
126         pageTitle: 'Your Orders',
127         orders: orders
128       });
129     })
130     .catch(err => console.log(err));
131 };
132

```

```

1  //./models/user.js
2
3  const mongodb = require('mongodb');
4  const getDb = require('../util/database').getDb;
5
6  const ObjectId = mongodb.ObjectId;
7
8  class User {
9    constructor(username, email, cart, id) {
10      this.name = username;
11      this.email = email;
12      this.cart = cart; // {items: []}
13      this._id = id;
14    }
15

```

```

16 save() {
17     const db = getDb();
18     return db.collection('users').insertOne(this);
19 }
20
21 addToCart(product) {
22     const cartProductIndex = this.cart.items.findIndex(cp => {
23         return cp.productId.toString() === product._id.toString();
24     });
25     let newQuantity = 1;
26     const updatedCartItems = [...this.cart.items];
27     if (cartProductIndex >= 0) {
28         newQuantity = this.cart.items[cartProductIndex].quantity + 1;
29         updatedCartItems[cartProductIndex].quantity = newQuantity;
30     } else {
31         updatedCartItems.push({
32             productId: new ObjectId(product._id),
33             quantity: newQuantity
34         });
35     }
36     const updatedCart = {
37         items: updatedCartItems
38     };
39     const db = getDb();
40     return db
41         .collection('users')
42         .updateOne(
43             { _id: new ObjectId(this._id) },
44             { $set: { cart: updatedCart } }
45         );
46 }
47
48 getCart() {
49     const db = getDb();
50     const productIds = this.cart.items.map(i => {
51         return i.productId;
52     });
53     return db
54         .collection('products')
55         .find({ _id: { $in: productIds } })
56         .toArray()
57         .then(products => {
58             return products.map(p => {
59                 return {
60                     ...p,
61                     quantity: this.cart.items.find(i => {
62                         return i.productId.toString() === p._id.toString();
63                     }).quantity
64                 };
65             });
66         });
67 }
68
69 deleteItemFromCart(productId){
70     /**'filter()' is a method provided by vanilla javascript
71     * 'filter()' allows us to define a criteria

```

```

72     * on how we wanna filter the elements in that array
73     * so in this case, the elements of the items array
74     * and then it will return a new array with all the filtered items
75     * so all the items that make it through the filter
76     *
77     * 'filter()' is a function which runs on every item
78     * and we return true if we wanna keep the item in the new array
79     * or false if you want to get rid of it.
80     */
81     const updatedCartItems = this.cart.items.filter(item => {
82         return item.productId.toString() !== productId.toString()
83     })
84     const db = getDb()
85     return db
86         .collection('users')
87         .updateOne(
88             { _id: new ObjectId(this._id) },
89             /*i wanna assign this to an object with an items property
90             * because that was what our cart has.
91             *
92             */
93             { $set: { cart: {items: updatedCartItems} } }
94         )
95     }
96
97     static findById(userId) {
98         const db = getDb();
99         return db
100             .collection('users')
101             .findOne({ _id: new ObjectId(userId) })
102             .then(user => {
103                 console.log(user);
104                 return user;
105             })
106             .catch(err => {
107                 console.log(err);
108             });
109     }
110 }
111
112 module.exports = User;

```

```

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);

```

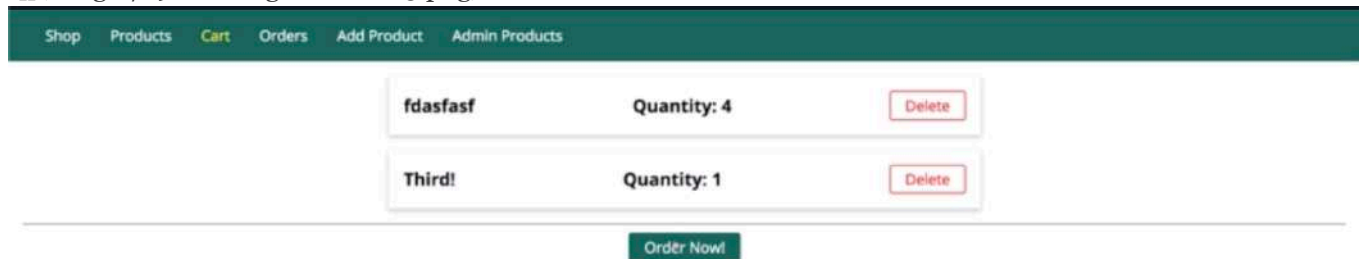
```

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

```

* Chapter 198: Adding An Order

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



Page Not Found!

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES

MongoDB 3.6.7 Enterprise

3 DBS COLLECTIONS

Filter

> admin

> local

▼ shop

orders

products

users

CREATE COLLECTION

Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
orders	1	124.0 B	124.0 B	1	16.0 KB
products	3	150.0 B	450.0 B	1	36.0 KB
users	1	83.0 B	83.0 B	1	16.0 KB

My Cluster

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("5baa355b9cfe7943a181c95b"),
  "items": Array(
    - 0: Object
      "productId": ObjectId("5baa27830e9b883d378758"),
      "quantity": 4
    - 1: Object
      "productId": ObjectId("5baa327abb1ea0424e7729b8"),
      "quantity": 1
  )
}

```

Shop Products **Cart** Orders Add Product Admin Products

No Products in Cart!

1/1

```

1 //./models/user.js
2
3 const mongodb = require('mongodb');
4 const getDb = require('../util/database').getDb;
5
6 const ObjectId = mongodb.ObjectId;
7
8 class User {
9   constructor(username, email, cart, id) {
10     this.name = username;
11     this.email = email;
12     this.cart = cart; // {items: []}
13     this._id = id;
14   }

```

```

15
16 save() {
17     const db = getDb();
18     return db.collection('users').insertOne(this);
19 }
20
21 addToCart(product) {
22     const cartProductIndex = this.cart.items.findIndex(cp => {
23         return cp.productId.toString() === product._id.toString();
24     });
25     let newQuantity = 1;
26     const updatedCartItems = [...this.cart.items];
27     if (cartProductIndex >= 0) {
28         newQuantity = this.cart.items[cartProductIndex].quantity + 1;
29         updatedCartItems[cartProductIndex].quantity = newQuantity;
30     } else {
31         updatedCartItems.push({
32             productId: new ObjectId(product._id),
33             quantity: newQuantity
34         });
35     }
36     const updatedCart = {
37         items: updatedCartItems
38     };
39     const db = getDb();
40     return db
41         .collection('users')
42         .updateOne(
43             { _id: new ObjectId(this._id) },
44             { $set: { cart: updatedCart } }
45         );
46 }
47
48 getCart() {
49     const db = getDb();
50     const productIds = this.cart.items.map(i => {
51         return i.productId;
52     });
53     return db
54         .collection('products')
55         .find({ _id: { $in: productIds } })
56         .toArray()
57         .then(products => {
58             return products.map(p => {
59                 return {
60                     ...p,
61                     quantity: this.cart.items.find(i => {
62                         return i.productId.toString() === p._id.toString();
63                     }).quantity
64                 };
65             });
66         });
67 }
68
69 deleteItemFromCart(productId){
70     const updatedCartItems = this.cart.items.filter(item => {

```



```

71     return item.productId.toString() !== productId.toString()
72   })
73   const db = getDb()
74   return db
75     .collection('users')
76     .updateOne(
77       { _id: new ObjectId(this._id) },
78       { $set: { cart: { items: updatedCartItems } } }
79     )
80   }
81
82   addOrder(){
83     /**this doesn't take any arguments
84     * because the cart which will be passed as an order or as the data for the order
85     * is already registered on this user
86     * so i need to add the orders to my user or the other way around
87     */
88     const db = getDb()
89     /**i will return the entire thing
90     * and the one new order will be well the cart i currently have
91     * i wanna insert my cart which refers to the users cart
92     */
93     return db
94       .collection('orders')
95       .insertOne(this.cart)
96       .then(result => {
97         this.cart = { items: [] }
98         return db
99           .collection('users')
100           .updateOne(
101             { _id: new ObjectId(this._id) },
102             { $set: { cart: { items: [] } } }
103           )
104       })
105   }
106
107   static findById(userId) {
108     const db = getDb();
109     return db
110       .collection('users')
111       .findOne({ _id: new ObjectId(userId) })
112       .then(user => {
113         console.log(user);
114         return user;
115       })
116       .catch(err => {
117         console.log(err);
118       });
119   }
120 }
121
122 module.exports = User;

```



```

1 //./controllers/shop.js
2
3 const Product = require('../models/product');
4

```

```
5 exports.getProducts = (req, res, next) => {
6   Product.fetchAll()
7     .then(products => {
8       res.render('shop/product-list', {
9         prods: products,
10        pageTitle: 'All Products',
11        path: '/products'
12      });
13    })
14    .catch(err => {
15      console.log(err);
16    });
17 };
18
19 exports.getProduct = (req, res, next) => {
20   const prodId = req.params.productId;
21   // Product.findAll({ where: { id: prodId } })
22   // .then(products => {
23   //   res.render('shop/product-detail', {
24   //     product: products[0],
25   //     pageTitle: products[0].title,
26   //     path: '/products'
27   //   });
28   // })
29   // .catch(err => console.log(err));
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
38     .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
58     .then(products => {
59       res.render('shop/cart', {
60         path: '/cart',
```

```

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

```

```

1 // ./routes/shop.js
2
3 const path = require('path');
4

```

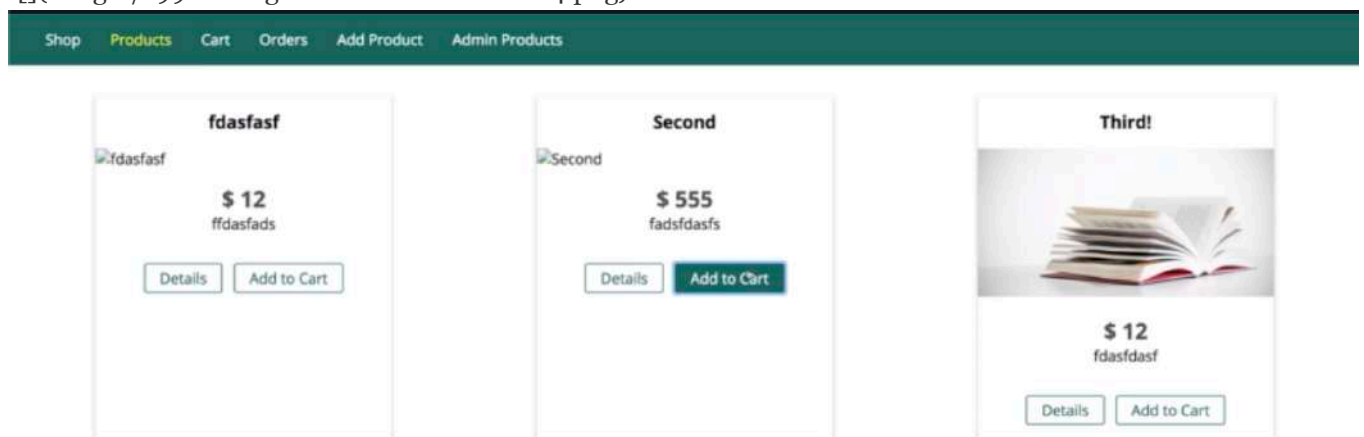
```

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

```

* Chapter 199: Adding Relational Order Data

1. update
- ./models/user.js



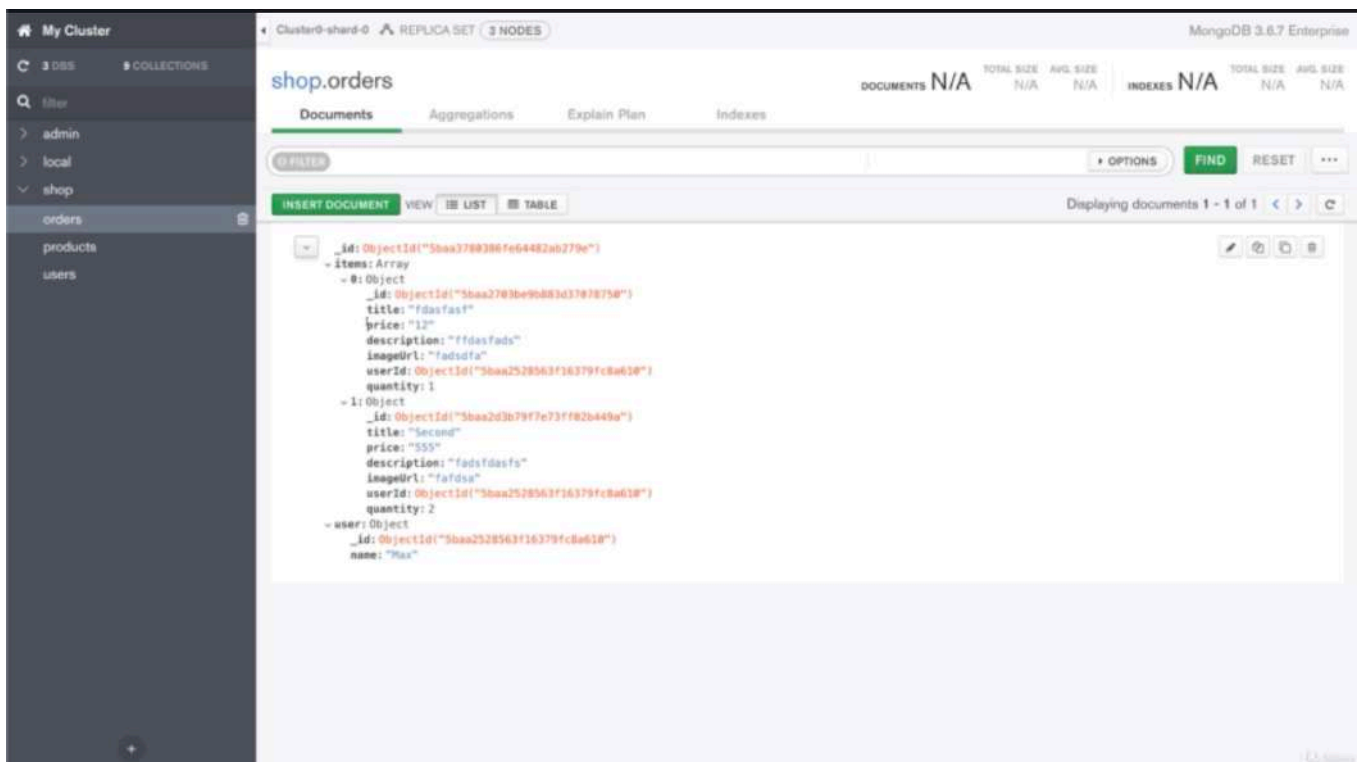
ShopProductsCartOrdersAdd ProductAdmin Products

fdasfasf	Quantity: 1	Delete
Second	Quantity: 2	Delete

Order Now!

ShopProductsCartOrdersAdd ProductAdmin Products

No Products in Cart!



- this is the new order we just added and there you see items does have all the enriched product information, the snapshots of our products and the quantity in there and we have some user data

```

1  //./models/user.js
2
3  const mongoose = require('mongoose');
4  const getDb = require('../util/database').getDb();
5
6  const ObjectId = mongoose.Types.ObjectId;
7
8  class User {
9    constructor(username, email, cart, id) {
10      this.name = username;
11      this.email = email;
12      this.cart = cart; // {items: []}
13      this._id = id;
14    }
15
16    save() {
17      const db = getDb();
18      return db.collection('users').insertOne(this);
19    }
20
21    addToCart(product) {
22      const cartProductIndex = this.cart.items.findIndex(cp => {
23        return cp.productId.toString() === product._id.toString();
24      });
25      let newQuantity = 1;
26      const updatedCartItems = [...this.cart.items];
27      if (cartProductIndex >= 0) {
28        newQuantity = this.cart.items[cartProductIndex].quantity + 1;
29        updatedCartItems[cartProductIndex].quantity = newQuantity;
30      } else {
31        updatedCartItems.push({
32          productId: new ObjectId(product._id),

```

```

33         quantity: newQuantity
34     });
35 }
36 const updatedCart = {
37     items: updatedCartItems
38 };
39 const db = getDb();
40 return db
41     .collection('users')
42     .updateOne(
43         { _id: new ObjectId(this._id) },
44         { $set: { cart: updatedCart } }
45     );
46 }
47
48 getCart() {
49     const db = getDb();
50     const productIds = this.cart.items.map(i => {
51         return i.productId;
52     });
53     return db
54         .collection('products')
55         .find({ _id: { $in: productIds } })
56         .toArray()
57         .then(products => {
58             return products.map(p => {
59                 return {
60                     ...p,
61                     quantity: this.cart.items.find(i => {
62                         return i.productId.toString() === p._id.toString();
63                     }).quantity
64                 };
65             });
66         });
67 }
68
69 deleteItemFromCart(productId){
70     const updatedCartItems = this.cart.items.filter(item => {
71         return item.productId.toString() !== productId.toString()
72     })
73     const db = getDb()
74     return db
75         .collection('users')
76         .updateOne(
77             { _id: new ObjectId(this._id) },
78             { $set: { cart: {items: updatedCartItems} } }
79         )
80 }
81
82 addOrder(){
83     const db = getDb()
84     /**i have that products data
85     * because outside of that 'then()' block,
86     * the code would execute too early.
87     *
88     * i need to 'return' the result of this 'addOrder()'

```

```

89  * so that outside of addOrder(), in my ./controllers/shop.js,
90  * i can call 'then()' in 'postOrder()'
91  */
92  return this.getCart().then(products => {
93    const order = {
94      /**then my items will be my products
95       * so an array of products with the product information and the quantity
96       * so now the product information will also be part of the order
97       *
98       * and here i really don't care about that information changing
99       * because if it should change, for orders, we need a snapshot anyways
100      * if the price of a product changes, that doesn't affect the past order
101      * so we wouldn't wanna update the price even if it would change
102      * so for orders, such a snapshot and therefore an embedded document is a great way
of relating the order and the product
103      * because the product data might be duplicate but it doesn't need to change in the
orders collection
104      * because we want a snapshot.
105      */
106      items: products,
107      /**we wanna have the 'cart.items' in there
108      * but we also wanna have some information about the user
109      * and i will add an embedded document where i add the '_id' and here,
110      * i wanna create a 'new objectId' based on the ID of the user we are working with
111      * but i also wanna store the name which we have as a property here and the email
112      * so i will duplicate data because this will then end up in the orders collection
and in this users collection
113      */
114      user: {
115        _id: new ObjectId(this._id),
116        name: this.name,
117      }
118    }
119    return db
120      .collection('orders')
121      .insertOne(order)
122    })
123    .then(result => {
124      this.cart = {items: []}
125      return db
126        .collection('users')
127        .updateOne(
128          { _id: new ObjectId(this._id) },
129          { $set: { cart: { items: [] } } }
130        )
131    })
132  }
133
134  getOrders(){
135    const db = getDb()
136    //return db.collection('orders').
137  }
138
139  static findById(userId) {
140    const db = getDb();
141    return db

```



```

142 .collection('users')
143 .findOne({ _id: new ObjectId(userId) })
144 .then(user => {
145   console.log(user);
146   return user;
147 })
148 .catch(err => {
149   console.log(err);
150 });
151 }
152 }
153
154 module.exports = User;

```

* Chapter 200: Getting Orders

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

The screenshot shows the MongoDB Enterprise interface for the 'shop.orders' collection. The left sidebar shows the database structure with 'shop' selected. The main area displays two documents in a list view. The first document is a JSON object with an '_id', an 'items' array containing two objects (each with '_id', 'title', 'price', 'description', 'imageUrl', 'userId', and 'quantity'), and a 'user' object with '_id' and 'name'. The second document is a similar JSON object with one item and one user.

- our user, for example, has 2 orders because we got 2 orders for that userId.

[Shop](#) [Products](#) [Cart](#) [Orders](#) [Add Product](#) [Admin Products](#)

Order - # 5baa3780386fe64482ab279e

fdasfasf (1)

Second (2)

Order - # 5baa37bbb96e68449b2724c7

fdasfasf (1)

```
1  ../models/user.js
2
3  const mongodb = require('mongodb');
4  const getDb = require('../util/database').getDb;
5
6  const ObjectId = mongodb.ObjectId;
7
8  class User {
9    constructor(username, email, cart, id) {
10      this.name = username;
11      this.email = email;
12      this.cart = cart; // {items: []}
13      this._id = id;
14    }
15
16    save() {
17      const db = getDb();
18      return db.collection('users').insertOne(this);
19    }
20
21    addToCart(product) {
22      const cartProductIndex = this.cart.items.findIndex(cp => {
23        return cp.productId.toString() === product._id.toString();
24      });
25      let newQuantity = 1;
26      const updatedCartItems = [...this.cart.items];
27      if (cartProductIndex >= 0) {
28        newQuantity = this.cart.items[cartProductIndex].quantity + 1;
29        updatedCartItems[cartProductIndex].quantity = newQuantity;
30      } else {
31        updatedCartItems.push({
32          productId: new ObjectId(product._id),
33          quantity: newQuantity
34        });
35      }
```

```

36     const updatedCart = {
37       items: updatedCartItems
38     };
39     const db = getDb();
40     return db
41       .collection('users')
42       .updateOne(
43         { _id: new ObjectId(this._id) },
44         { $set: { cart: updatedCart } }
45       );
46   }
47
48   getCart() {
49     const db = getDb();
50     const productIds = this.cart.items.map(i => {
51       return i.productId;
52     });
53     return db
54       .collection('products')
55       .find({ _id: { $in: productIds } })
56       .toArray()
57       .then(products => {
58         return products.map(p => {
59           return {
60             ...p,
61             quantity: this.cart.items.find(i => {
62               return i.productId.toString() === p._id.toString();
63             }).quantity
64           };
65         });
66       });
67   }
68
69   deleteItemFromCart(productId){
70     const updatedCartItems = this.cart.items.filter(item => {
71       return item.productId.toString() !== productId.toString()
72     })
73     const db = getDb()
74     return db
75       .collection('users')
76       .updateOne(
77         { _id: new ObjectId(this._id) },
78         { $set: { cart: {items: updatedCartItems} } }
79       )
80   }
81
82   addOrder(){
83     const db = getDb()
84     return this.getCart().then(products => {
85       const order = {
86         items: products,
87         user: {
88           _id: new ObjectId(this._id),
89           name: this.name,
90         }
91       }

```

```

92     return db
93     .collection('orders')
94     .insertOne(order)
95   })
96   .then(result => {
97     this.cart = {items: []}
98     return db
99       .collection('users')
100      .updateOne(
101        { _id: new ObjectId(this._id) },
102        { $set: { cart: { items: [] } } }
103      )
104   })
105 }
106
107 getOrders(){
108   const db = getDb()
109   /**each order has a user object
110    * and in that user object, we have the ID of that user
111    * so we need to compare that ID to the current user ID
112    * now to do that, we add a filter
113    * and now in MongoDB, you can check nested properties by defining the path to them
114    * the important thing to know here is that
115    * you need to use quotation marks ' ' around the path
116    * and then you can say check user and then the ID for the user
117    *
118    * 'user._id' will look for _id in the user property which holds an embedded document
119    * and then i can compare it is to a new ObjectId for this ID
120    * and this should give me all orders for that user
121    * and this will now be more than one
122    * so again we can use the 'toArray()' shortcut
123    * and return that data to return an array of orders for that user
124   */
125   return db
126     .collection('orders')
127     .find({ 'user._id': new ObjectId(this._id) })
128     .toArray()
129 }
130
131 static findById(userId) {
132   const db = getDb();
133   return db
134     .collection('users')
135     .findOne({ _id: new ObjectId(userId) })
136     .then(user => {
137       console.log(user);
138       return user;
139     })
140     .catch(err => {
141       console.log(err);
142     });
143 }
144 }
145
146 module.exports = User;

```

```
2
3 const Product = require('../models/product');
4
5 exports.getProducts = (req, res, next) => {
6   Product.fetchAll()
7     .then(products => {
8       res.render('shop/product-list', {
9         prods: products,
10        pageTitle: 'All Products',
11        path: '/products'
12      });
13    })
14    .catch(err => {
15      console.log(err);
16    });
17 };
18
19 exports.getProduct = (req, res, next) => {
20   const prodId = req.params.productId;
21   // Product.findAll({ where: { id: prodId } })
22   // .then(products => {
23   //   res.render('shop/product-detail', {
24   //     product: products[0],
25   //     pageTitle: products[0].title,
26   //     path: '/products'
27   //   });
28   // })
29   // .catch(err => console.log(err));
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
38     .catch(err => console.log(err));
39 };
40
41 exports.getIndex = (req, res, next) => {
42   Product.fetchAll()
43     .then(products => {
44       res.render('shop/index', {
45         prods: products,
46         pageTitle: 'Shop',
47         path: '/'
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .getCart()
```

```

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

```

```

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.items.forEach(product => { %>
18                   <li><%= product.title %> (<%= product.quantity %>)</li>
19                 <% }); %>
20               </ul>
21             </li>
22           <% }); %>
23         </ul>
24       <% } %>
25     </main>
26     <%- include('../includes/end.ejs') %>

```

* Chapter 202: Wrap Up



Module Summary

NoSQL / MongoDB

- Alternative to SQL databases
- No strict schemas, fewer relations
- You can of course use schemas and reference-based relations but you got more flexibility
- Often, relations are also created by embedding other documents/ data

Working with MongoDB

- Use the official MongoDB Driver
- Commands like `insertOne()`, `find()`, `updateOne()` and `deleteOne()` make CRUD-operations very simple
- Check the official docs to learn about all available operations + configurations/ operators
- All operations are promise-based, hence you can easily chain them for more complex flows

35 slides

* Chapter 204: 2 Adjustments(Behind The Scenes)

Behind the scenes, 2 files were deleted

- order-item.js
- order.js

Why? We simply don't need them anymore, the way we now structured our models