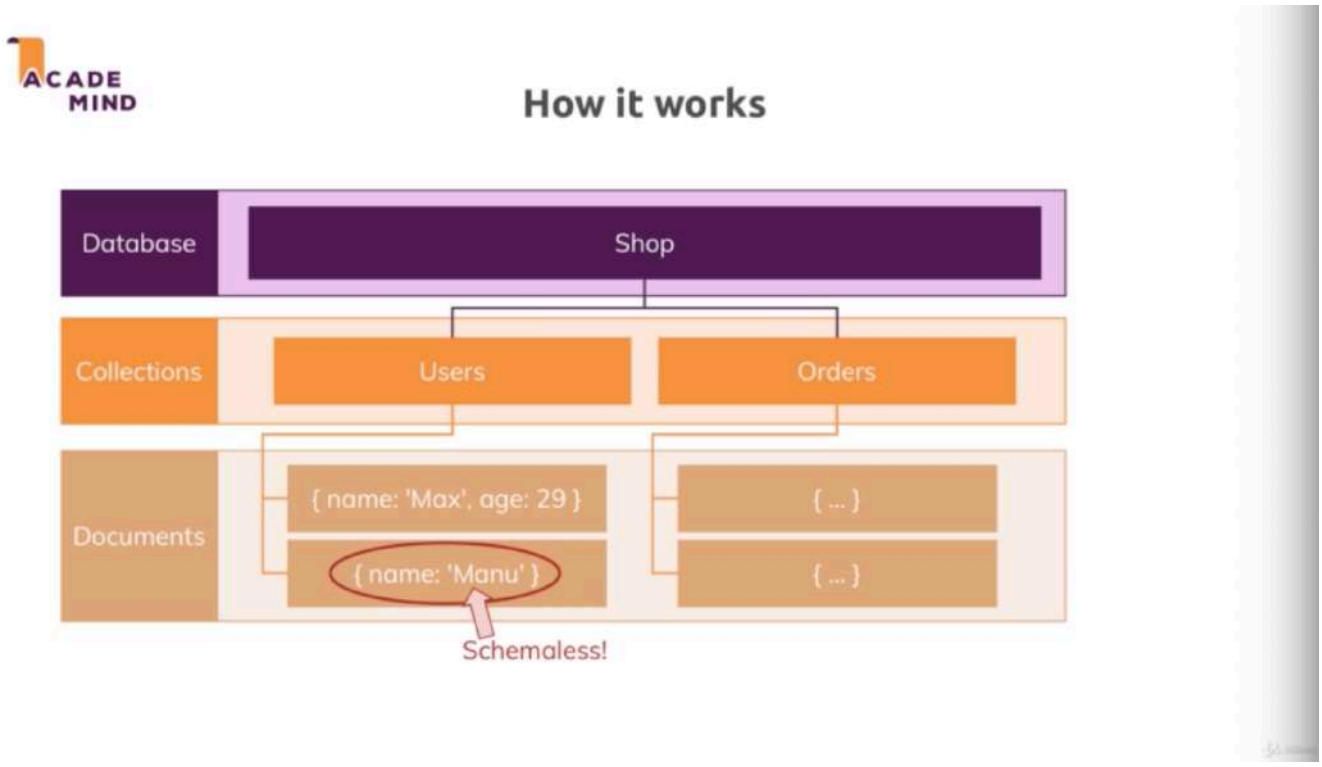
12. Working With NoSQL & Using MongoDb

* Chapter 172: Module Introduction

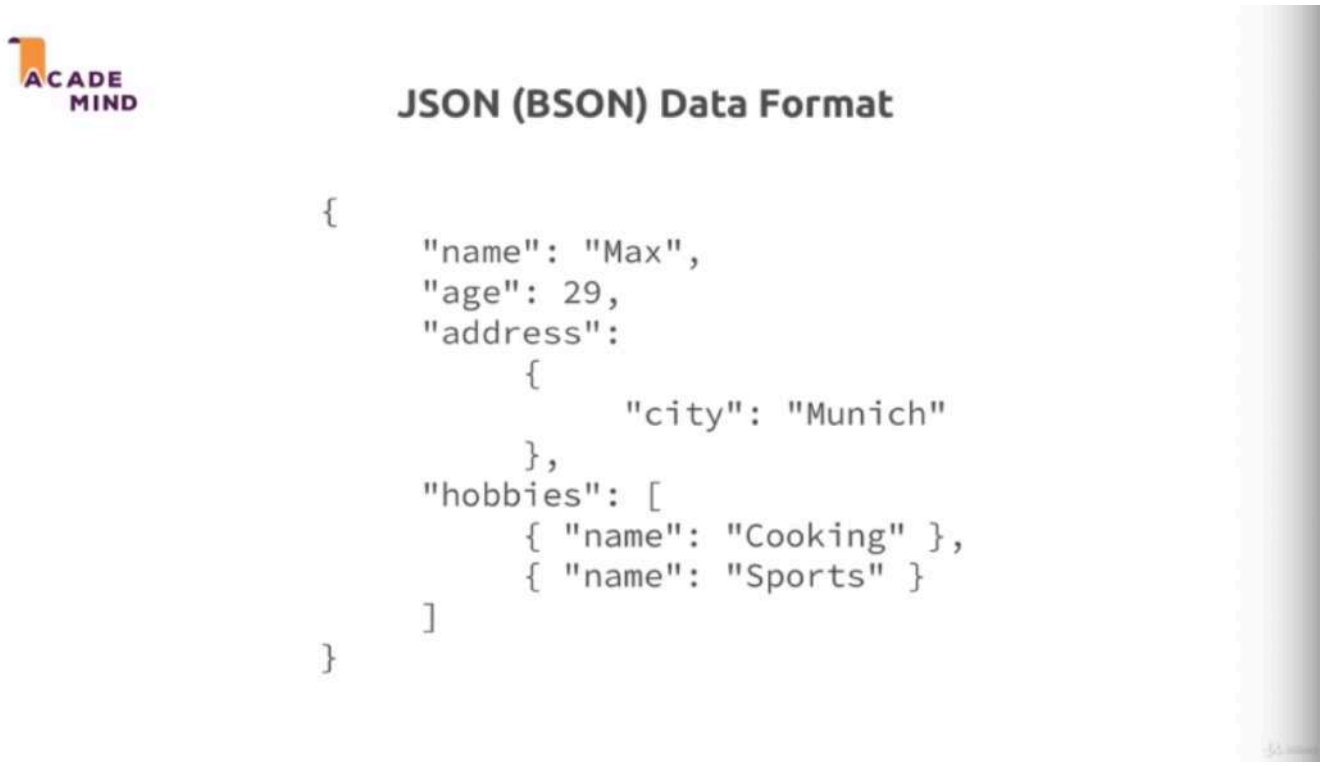
* Chapter 173: What Is MongoDB?



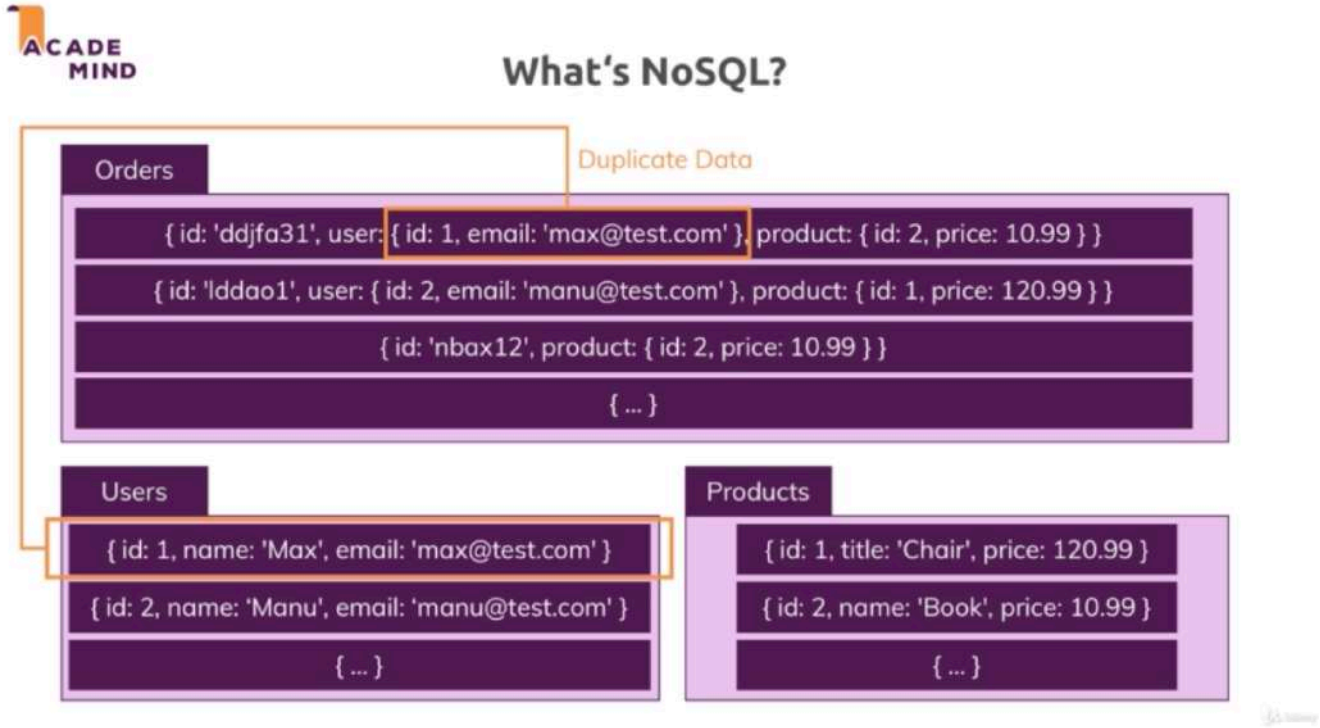
- for example a shop database, now in such a database in the SQL world, we would have multiple tables.
- in the NoSQL MongoDB world, we have multiple collections like the users and orders collection for example.
- inside of each collection, we don't have so-called 'records' but we have a couple of 'documents'. '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.

- 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.

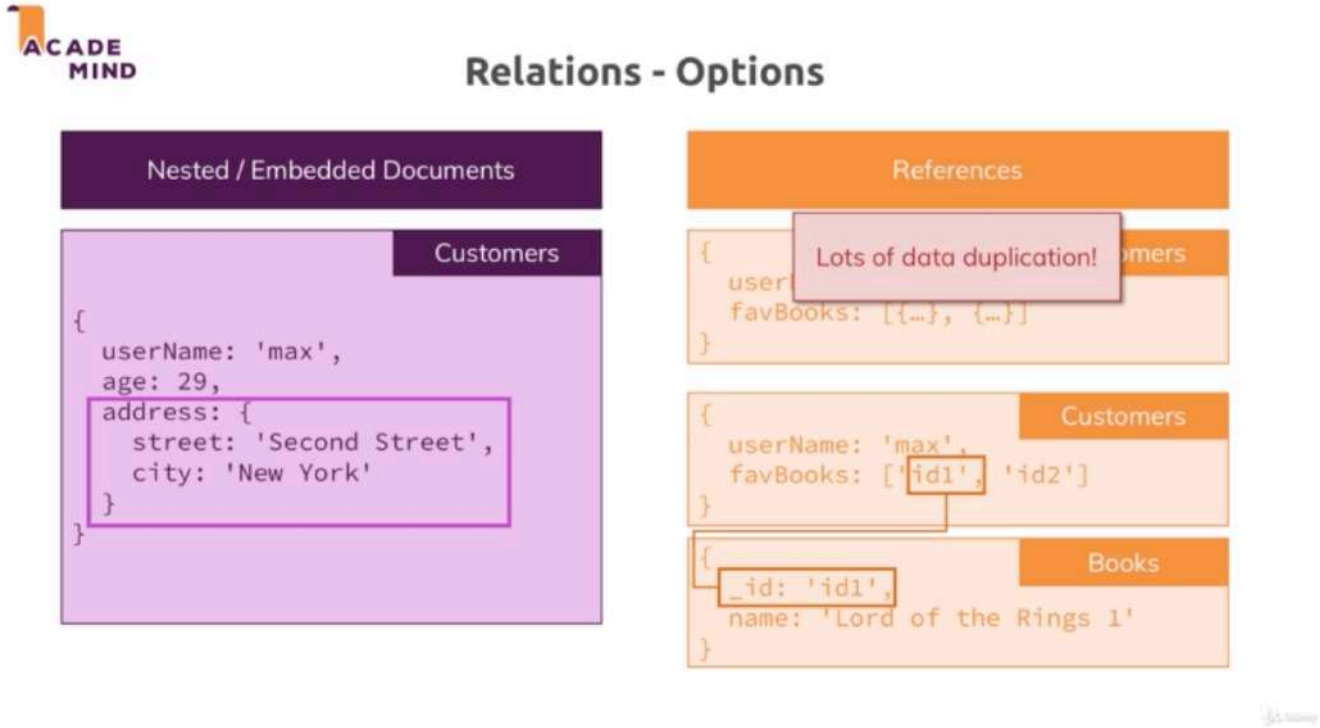
* Chapter 174: Relations In NoSQL



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

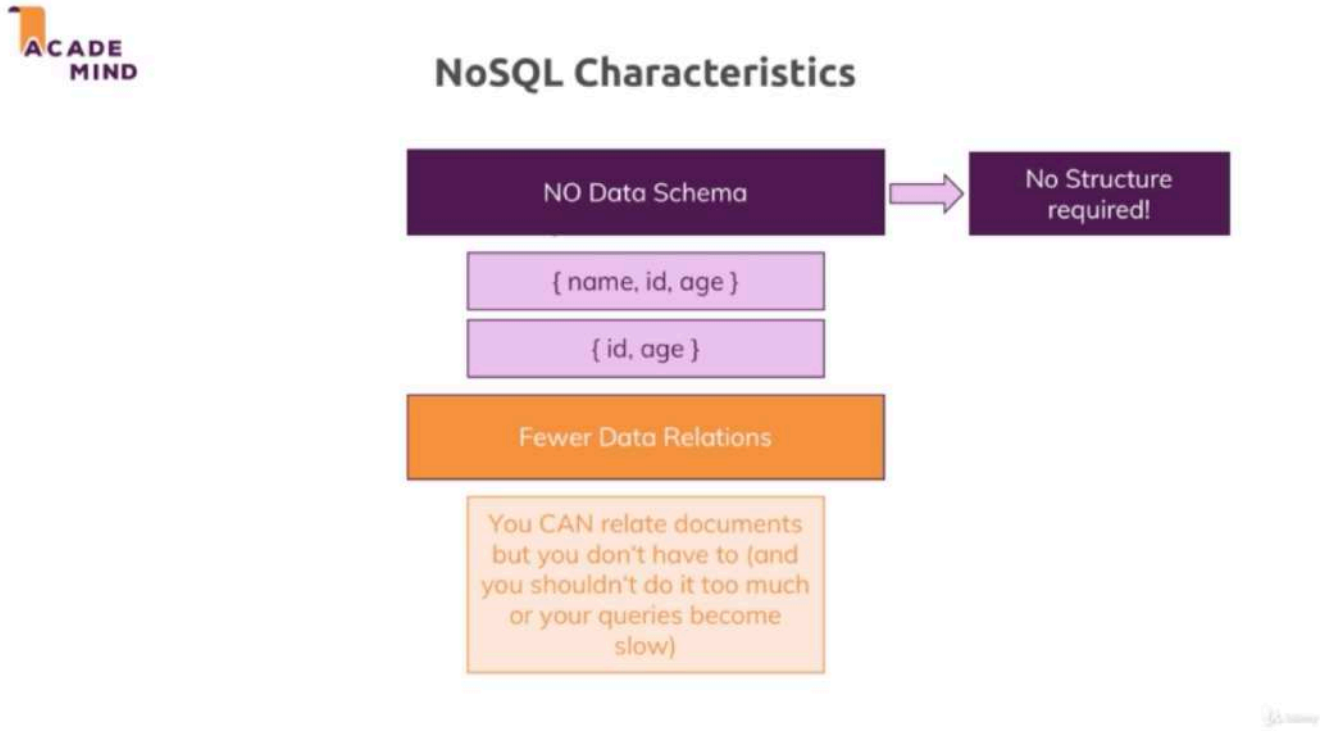


- In NoSQL, it would be pretty normal to have something like this.
- here are 3 collections and we have some duplicate data in there.
- instead of just matching by ID as you do in the SQL world, here you can also depict a relation by embeddnig data into other documents. you could embedded the ID which 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.

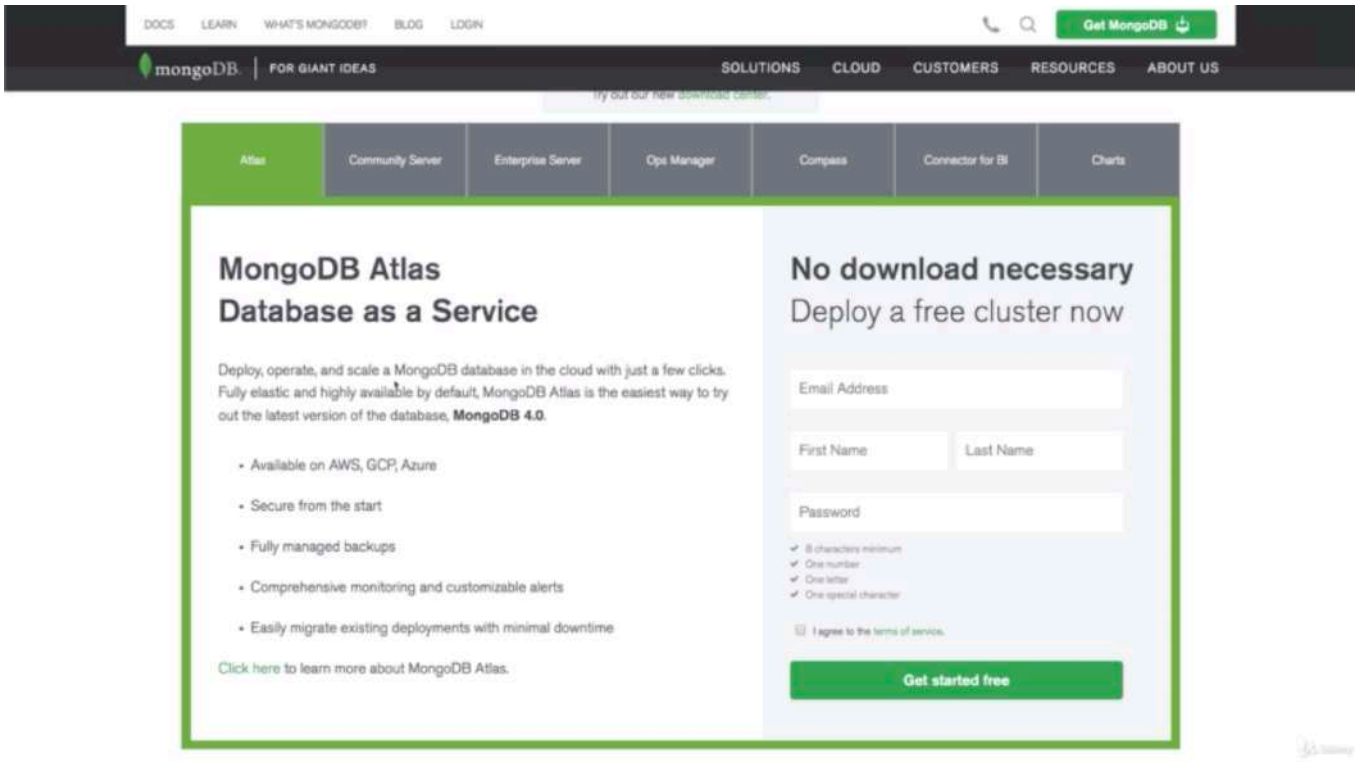


- here's the embedded document example where the address is part of our customer document. so instead of having 2 collections, customers and address and then matching by ID, here we put the address right into the

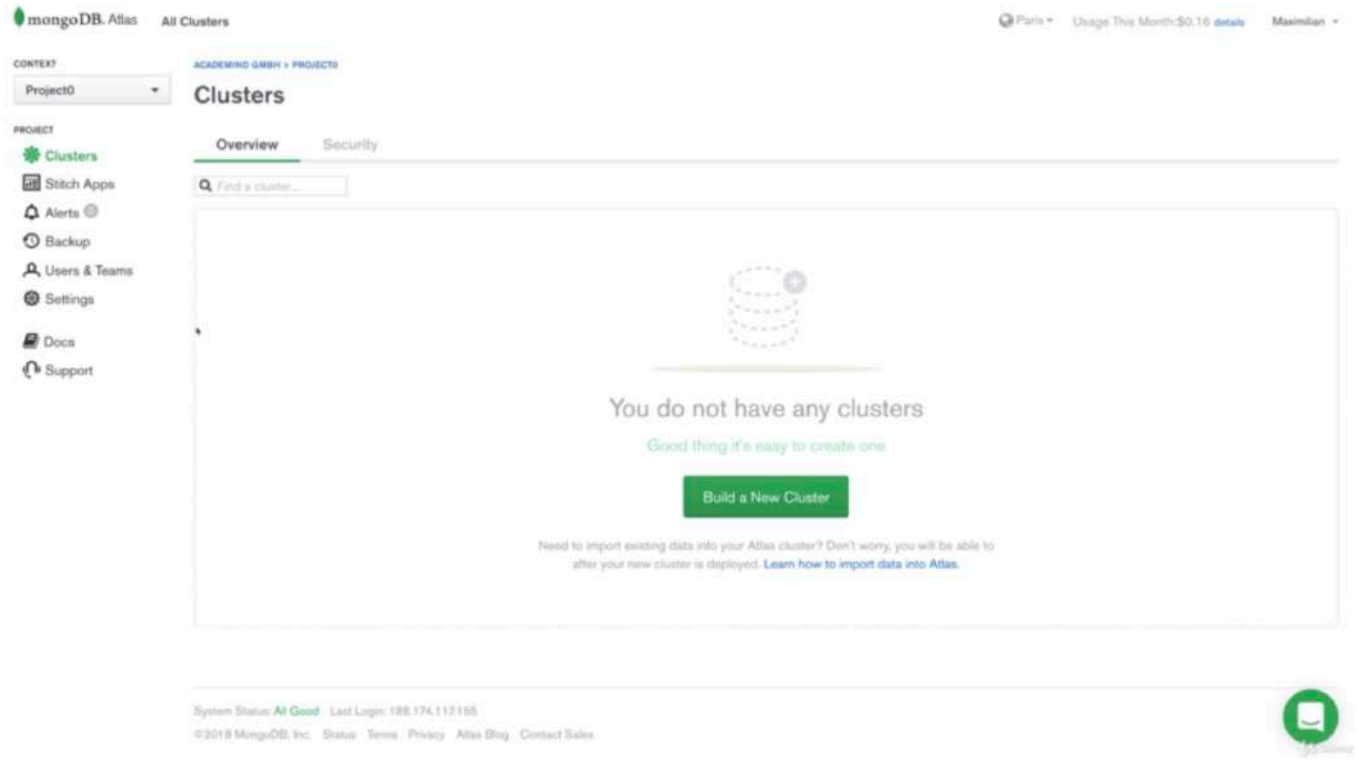
customer.
- there also are cases where you would have a lot of data duplication and where you need to work with that data a lot. and hence it would change a lot and you would have to manually update it in all duplicate plcaes where using embedded documents is not ideal. so for example if you have some 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



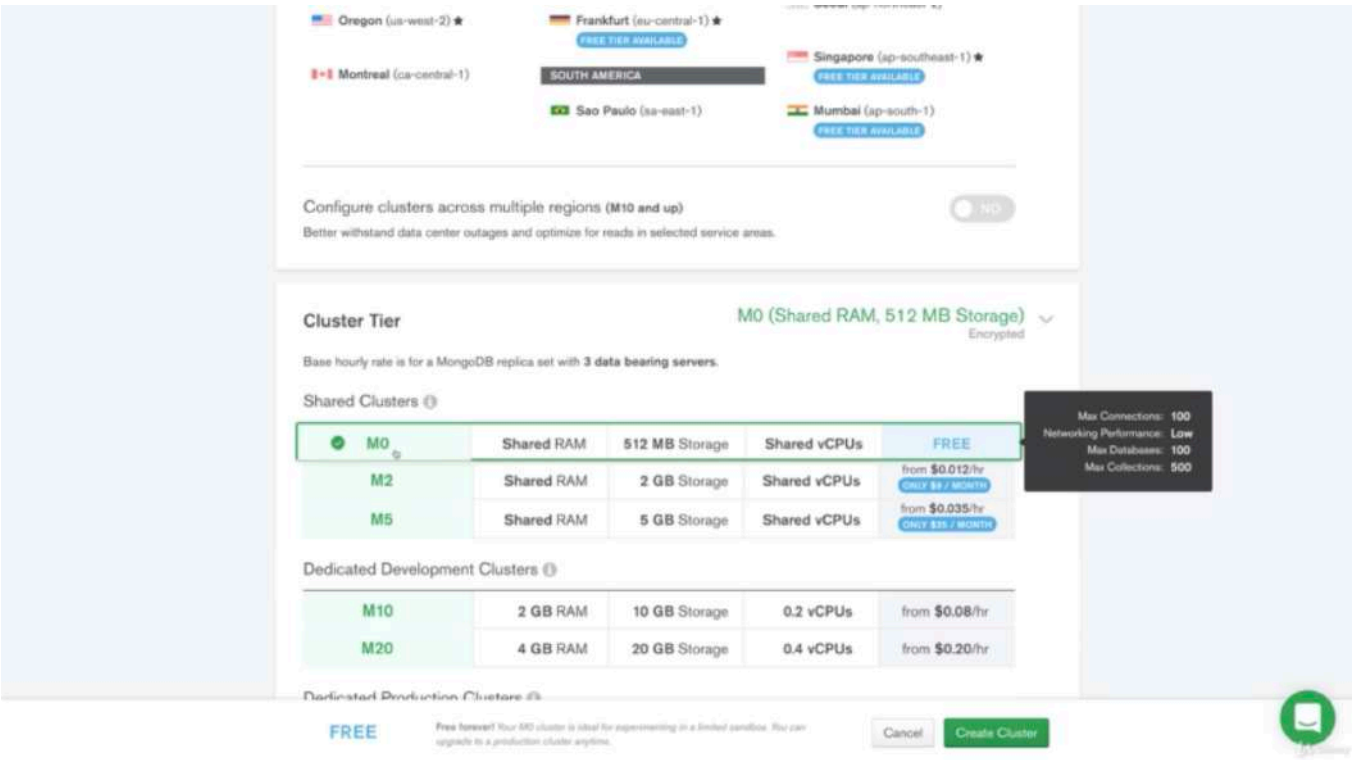
* Chapter 175: Setting Up MongoDB



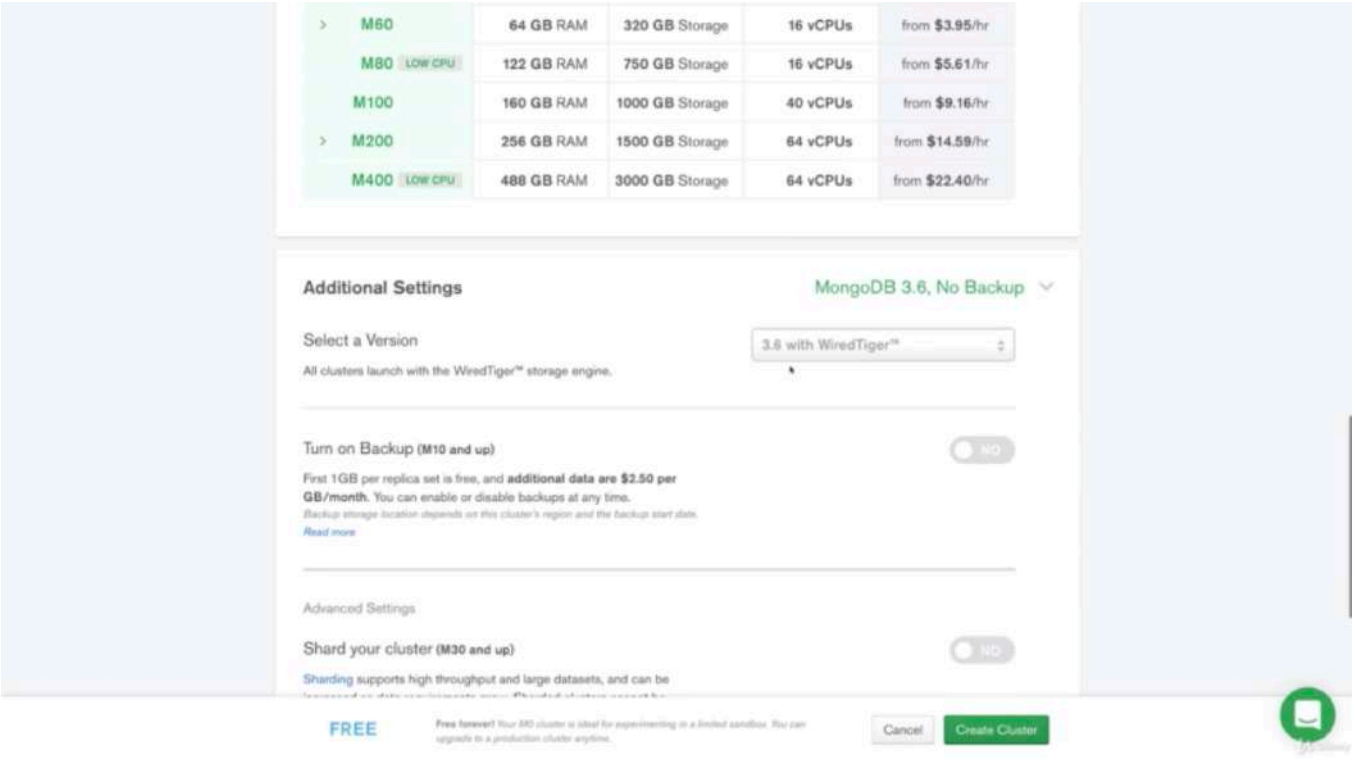
- i would go for a cloud solution becasue it's the more realistic environment. we would use for deployment and it's really easy to set up and it's free.
- and that will be 'Atlas'. so MongoDB Atlas.



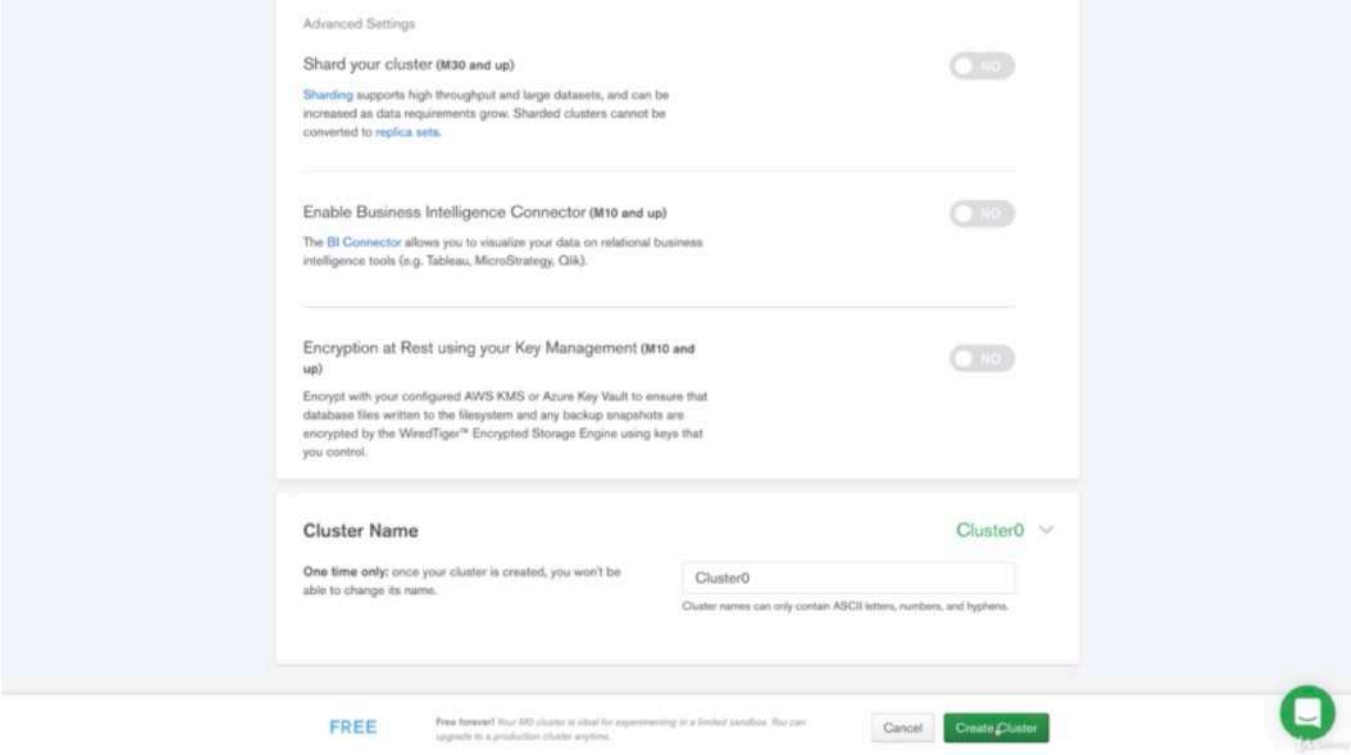
- once you did sign up, you should end up on a page that looks something like this.



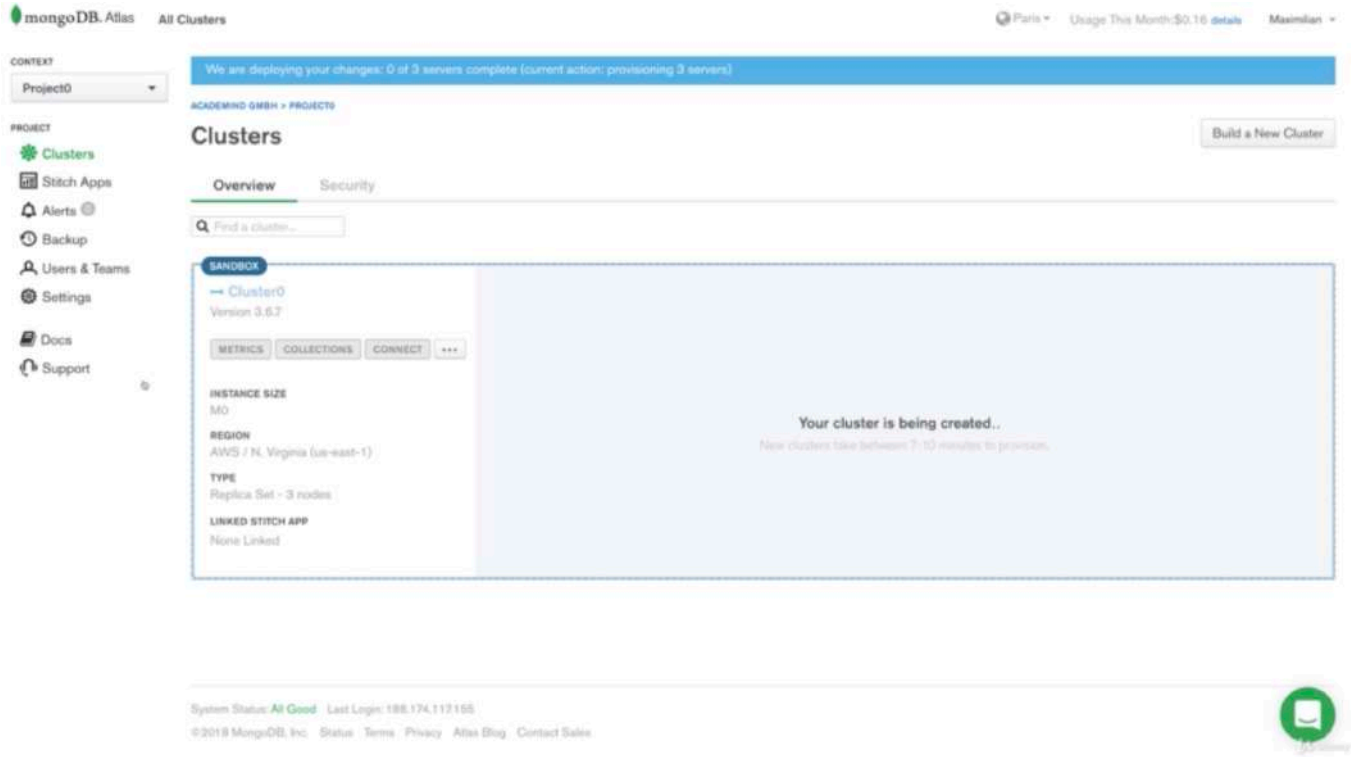
- maybe you need to create a new project first, give it any name you want and then you should be on that.
- or you can build a new cluster. you might end up in that wizard right from the start.
- you can generally leave all the default settings. make sure that you choose a regine where free tier is available.



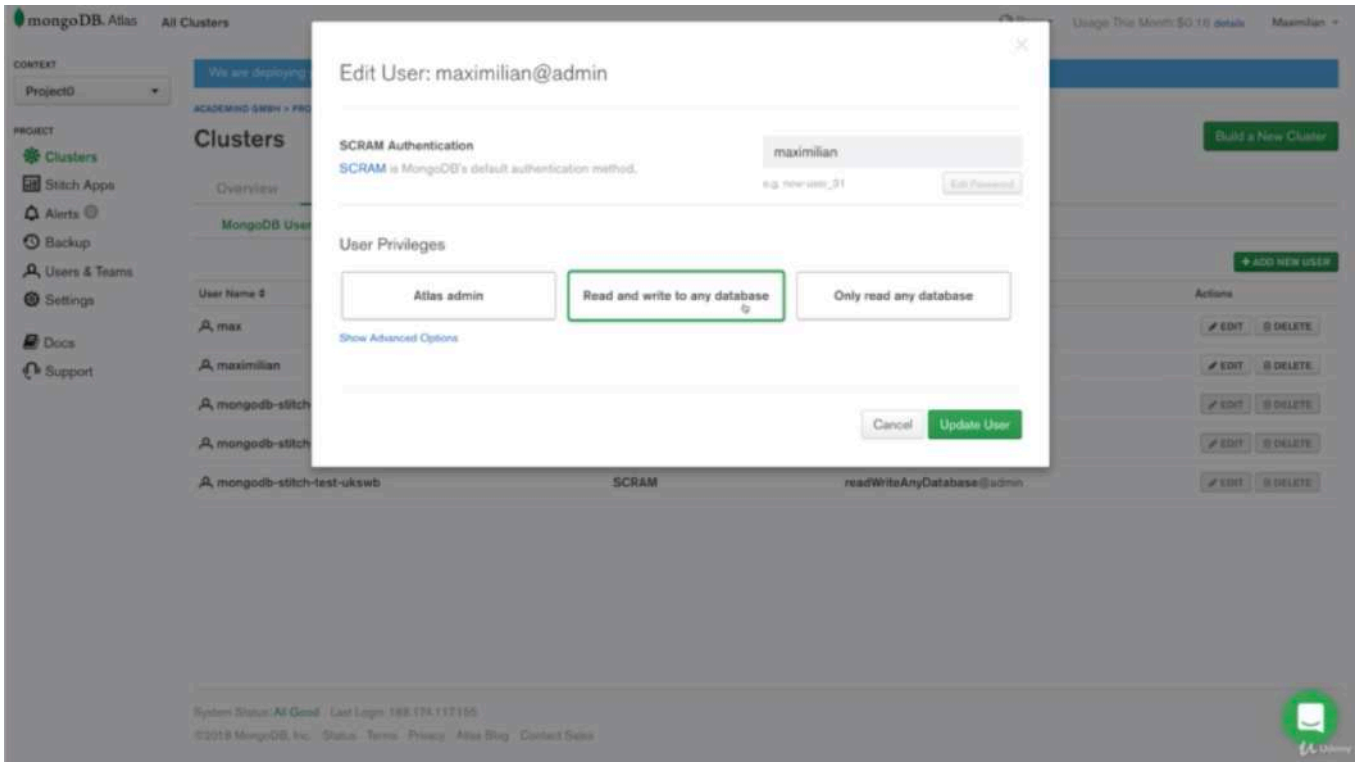
- and then you choose the cluster tier and there you should use the free one. M0.



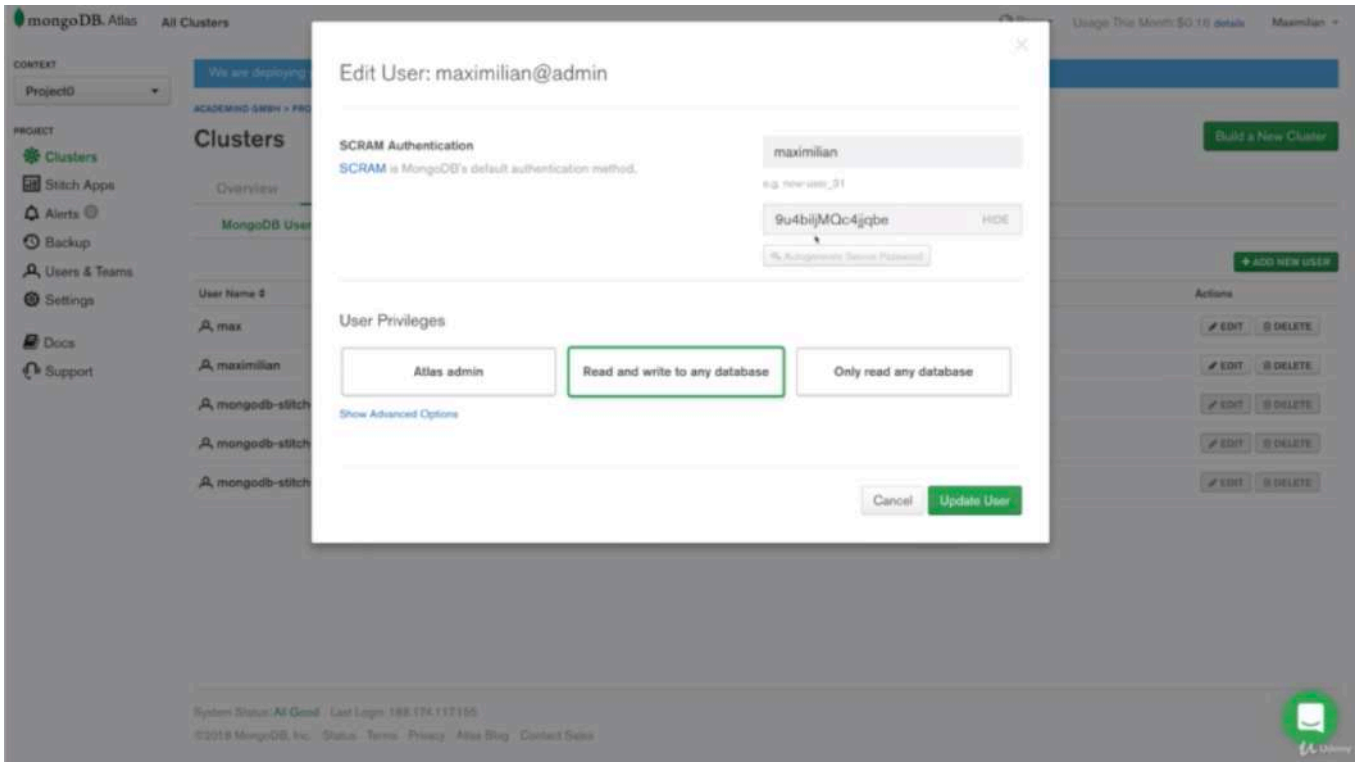
- Under additional settings, you can leave all the defaults



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



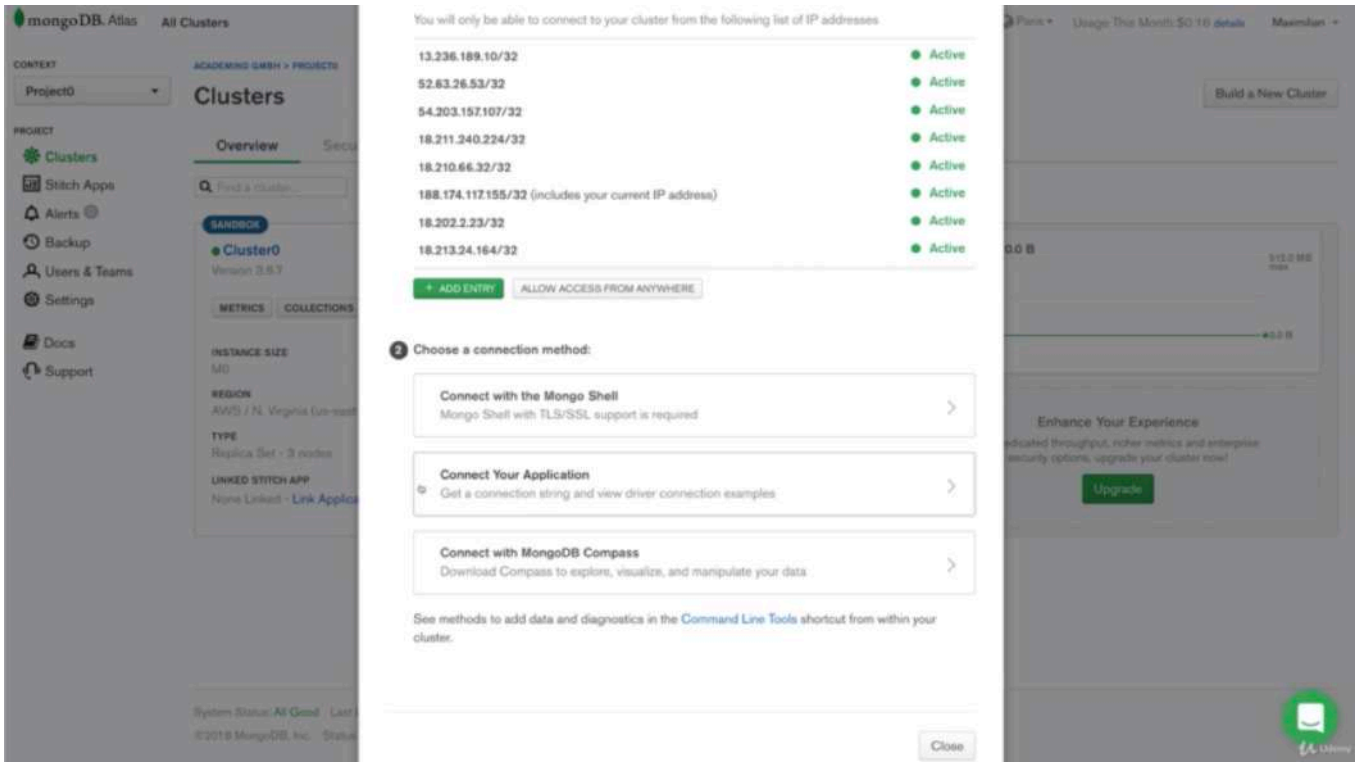
- while this is getting set up, you can already click on security and make sure that you add at least one new user which has 'Read and write to any database'.
- you can turn it into 'Atlas admin' but more realistic is this 'Read and write to any database' because this will later be the role which our 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.



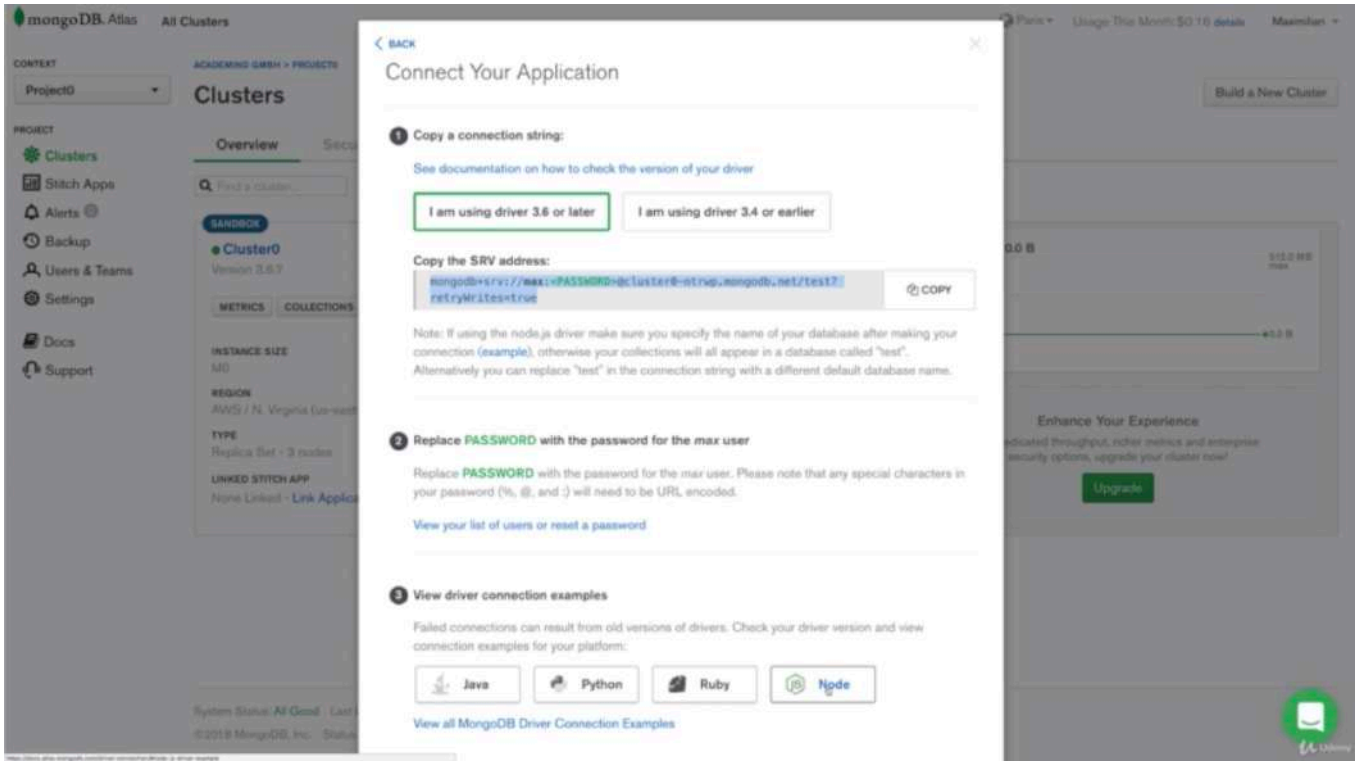
- you can also sign your own password or auto-generate a secure one. make sure you save that because you will need it later.

- 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 yo 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.



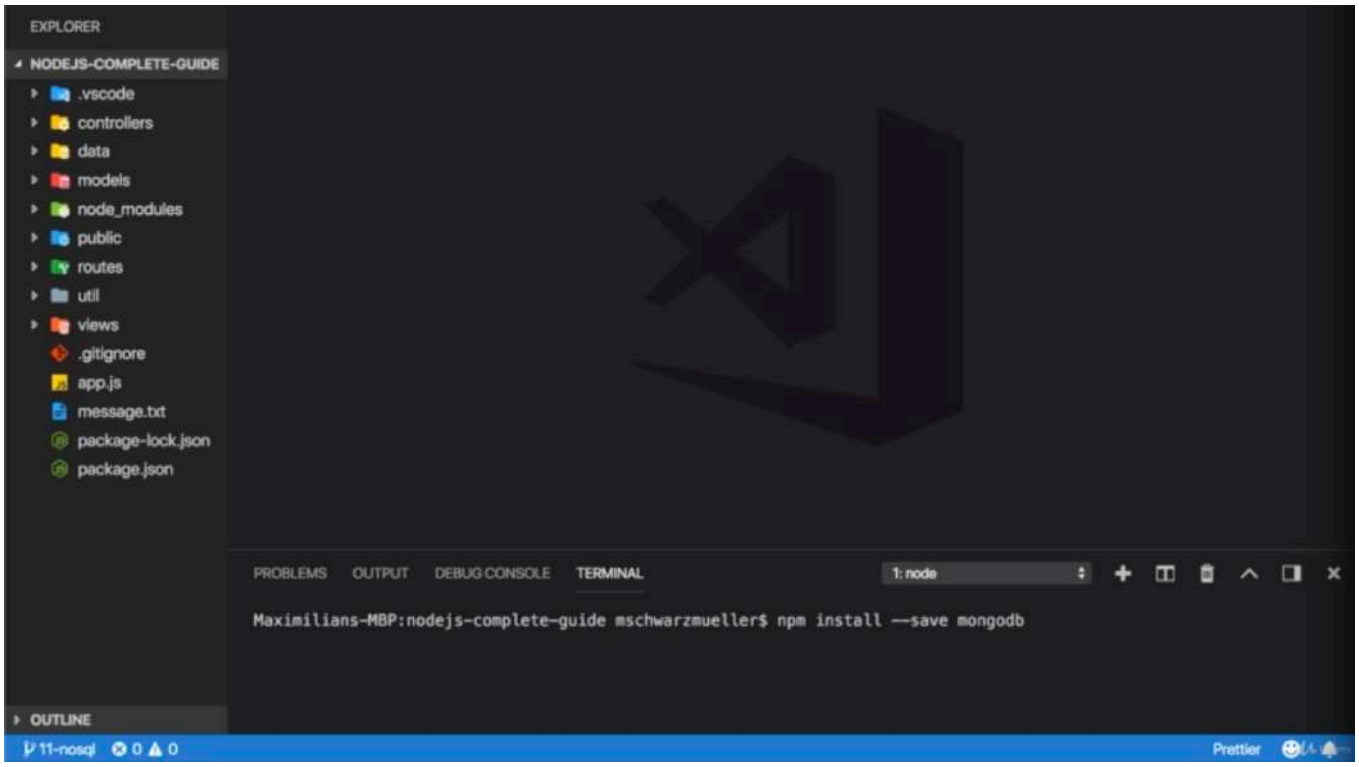
- now we can connect to our MongoDB server from insdie our node app and we can click on 'connect' here and choose a connection method which in our case will be an application. so 'Connect Your Application'



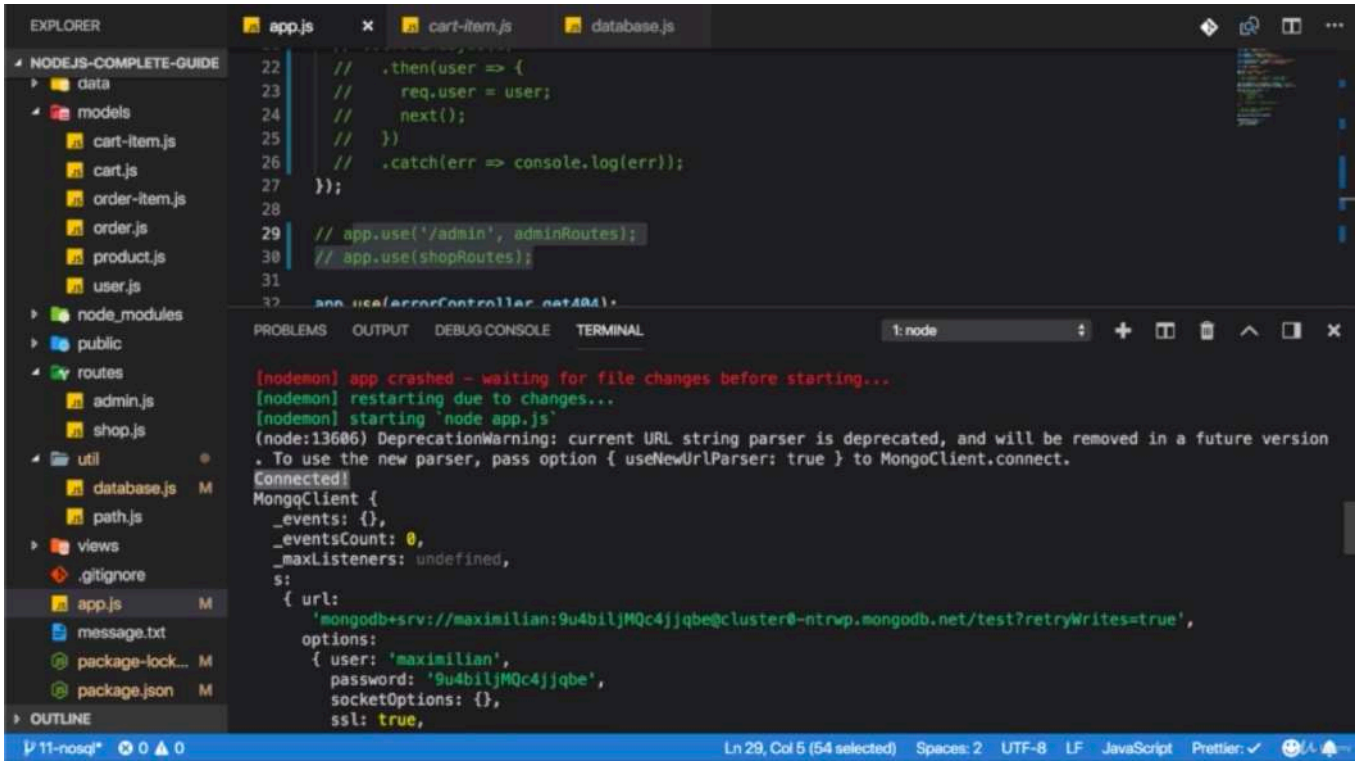
- here we can check i'm using this driver and we get this URL which we will need soon. but first of all, we need to install the MongoDB driver, in our case for Node.js.

* 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.

```
//app.js

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database')

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

//const adminRoutes = require('./routes/admin');
//const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
/*
    User.findByPk(1)
        .then(user => {
            req.user = user
            next()
        })
        .catch(err => console.log(err))
*/
})

//app.use('/admin', adminRoutes);
//app.use(shopRoutes);

app.use(errorController.get404);

/**we hve to pass a callback so function that will get executed
 * once we connect it
 * and here i will get access to the client object
 */
mongoConnect((client) => {
    console.log(client)
    app.listen(3000)
})
```

```
//./util/database.js

const mongodb = require('mongodb')
/**we can extract a Mongo Client constructor by simply accessing mongodb */
const MongoClient = mongodb.MongoClient

const mongoConnect = (callback) => {
/**now we can use that client to connect to our mongodb database.
```

```
 * this is all we need to do to create a connection to MongoDB
 *
 * 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
 * the important thing is that you need to make sure you are using the right user
 * and in my case that should be 'maximilian' the user you created in your MongoDB cluster under security
 * and the fitting password filled when you add user.
*/
MongoClient
    .connect(
        'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/test?retryWrites=true'
    )
    .then(client => {
        console.log('Connected!')
        callback(client)
    })
    .catch(err => console.log(err))
}

module.exports = mongoConnect

```

* Chapter 177: Creating The Database Connection

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

```
//app.js

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database')

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

const adminRoutes = require('./routes/admin');
//const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
/*
    User.findByPk(1)
```

```
        .then(user => {
            req.user = user
            next()
        })
        .catch(err => console.log(err))
*/
})

//app.use('/admin', adminRoutes);
//app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect((client) => {
    console.log(client)
    app.listen(3000)
})
```

```
//./models/product.js

const mongoConnect = require('./util/database')

class Product {
  constructor(title, price, description, imageUrl){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
  }
/** this is a function which can be executed on this class
 * and in here i now wanna connect to MongoDB and save my product.
 * now to do that, to be able to connect, i will need to import MongoDB or mongoConnect
 *
 * so i simply import the function i created inside './util/database'
 * where you pass a callback to,
 * where we connect to MongoDB inside.
 */
  save(){

  }
}

const Product = sequelize.define('product', {
  id: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  title: Sequelize.STRING,
  price: {
    type: Sequelize.DOUBLE,
    allowNull: false
  },
  imageUrl: {
    type: Sequelize.STRING,
    allowNull: false
```

```
  },
  description: {
    type: Sequelize.STRING,
    allowNull: false
  }
});

module.exports = Product;


```

```
//./util/database.js

const mongodb = require('mongodb')

const MongoClient = mongodb.MongoClient

/**we execute the callback and return the connected client
 * so that we can interact with it
 * however if you would this,
 * we would have to connect to MongoDB for every operation we do
 * and we would not even disconnect thereafter
 * 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.
 *
 * so it would be better if we could manage one connection in our database
 * and then simply return access to the client which we set up once from there
 * or to the different place in our app that need access
 */
const mongoConnect = (callback) => {
MongoClient
    .connect(
        'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/test?retryWrites=true'
    )
    .then(client => {
        console.log('Connected!')
        callback(client)
    })
    .catch(err => console.log(err))
}

module.exports = mongoConnect

```

* Chapter 178: Finishing The Database Connection

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

```
//app.js

const path = require('path');
```

```

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database')

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

const adminRoutes = require('./routes/admin');
//const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
/*
    User.findByPk(1)
        .then(user => {
            req.user = user
            next()
        })
        .catch(err => console.log(err))
*/
})

//app.use('/admin', adminRoutes);
//app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect(() => {
    app.listen(3000)
})
```

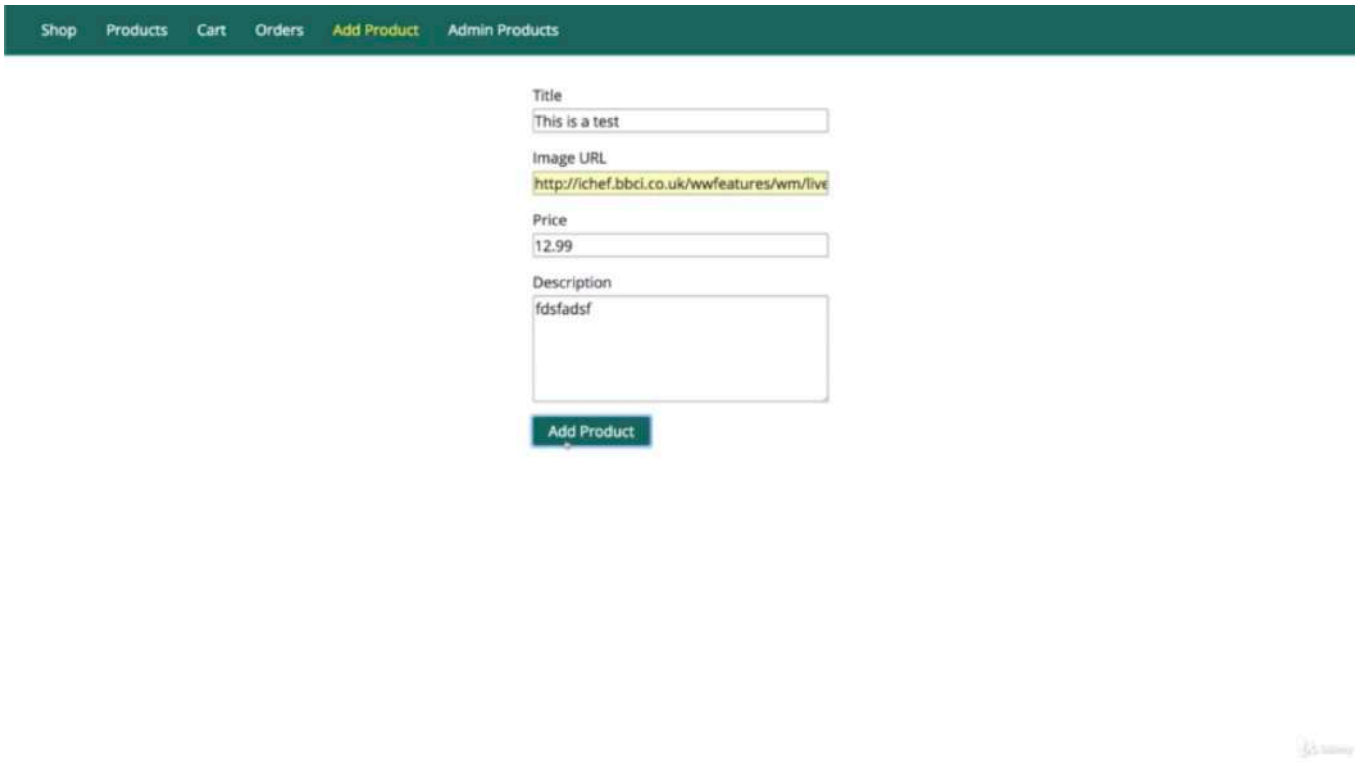
```
//./models/product.js

/**Using 'getDb' means that i can now call this function
 * to get access to my database
 * and therefore i can use it to interact with the database.
 */
const getDb = require('./util/database').getDb

class Product {
  constructor(title, price, description, imageUrl){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
  }
  save(){

  }
}
```

```

const Product = sequelize.define('product', {
  id: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  title: Sequelize.STRING,
  price: {
    type: Sequelize.DOUBLE,
    allowNull: false
  },
  imageUrl: {
    type: Sequelize.STRING,
    allowNull: false
  },
  description: {
    type: Sequelize.STRING,
    allowNull: false
  }
});

module.exports = Product;


```

```
//./util/database.js

const mongodb = require('mongodb')
const MongoClient = mongodb.MongoClient

/**instead i willl add a variable _db
 * underscore '_' is only here to signal
 * that this will only be used internally in this file
 * initially '_db' is undefined.
 */
let _db

/**i'm exporting 2 method
 * one for connecting and then storing the connection to the database
 * this will keep on running
 */
const mongoConnect = (callback) => {
MongoClient
    .connect(
        /**what we will do is we will connect to the 'test' database by default
         * because that is what we specify in our connection string.'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/\\\test\\\?retryWrites=true'
         * so we will connect to shop (replace 'test' to 'shop') 'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/\\\shop\\\?retryWrites=true'
         * this will give us access to the shop database to which we automatically connect
         *
         * on the side note, unlike in SQL
         * we never need to create that database or the tables, the collections in there
```

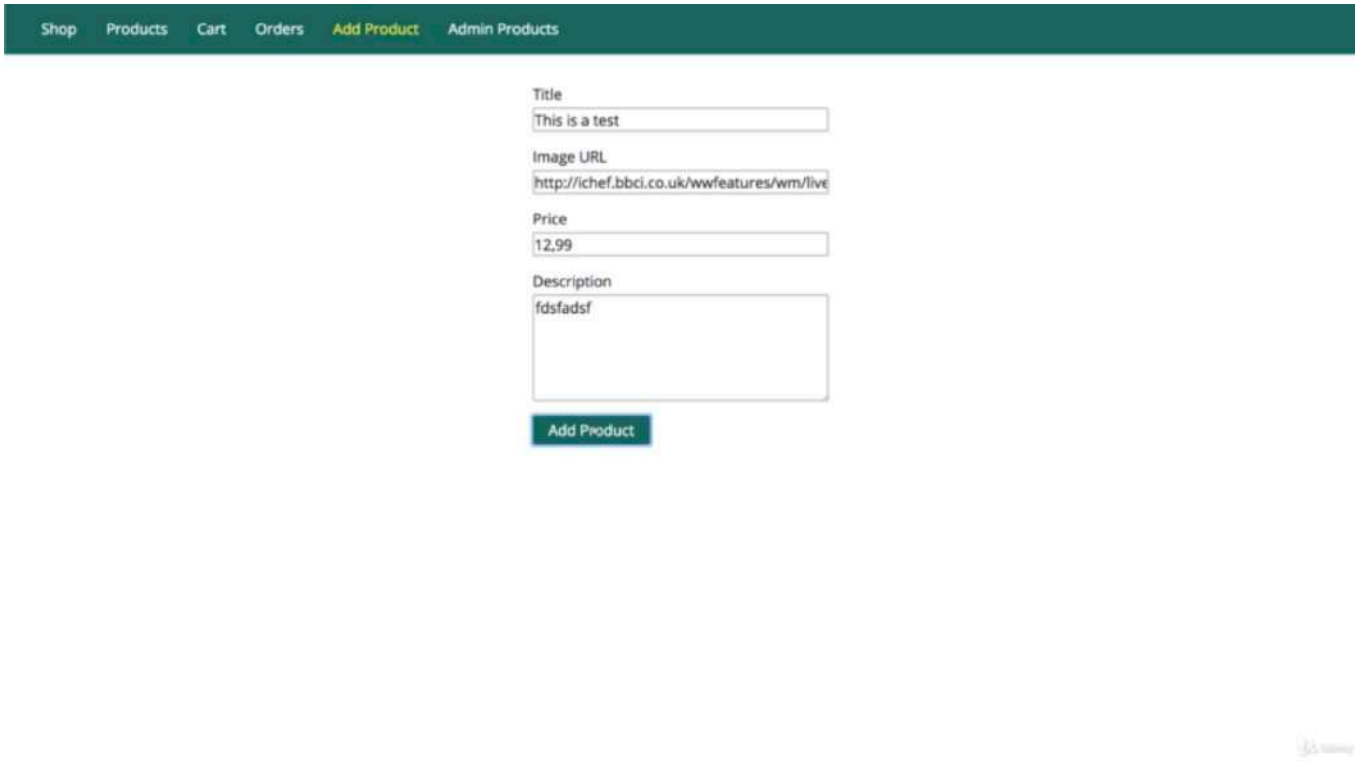
```
ahead of time
        * it will be created on the fly
        * when we first access it
        * which is again fitting that flexibility theme MongoDB gives us
        *
        * In SQL, we had to prepare everything in advance, at least when not using sequelize
        * which also had to do that but it did it for us.
        * we don't need to do anything. we are just telling MongoDB
        * hey connect me to the shop database
        * and if that database doesn't exist yet, MongoDB will create it as soon as we start writing data to it.
        */
        'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/shop?retryWrites=true'
    )
    .then(client => {
        console.log('Connected!')
        /**i will store access to the database here*/
        _db = client.db()
        callback(client)
    })
    .catch(err => {
        console.log(err)
        throw err
    })
}

/**and i have one method where i return access to that connected database
 * if it exists and mongoDB behind the scenes will manage this very elegantly with something called 'connection pooling'
 * where mongoDB will make sure it provides sufficient connections for multiple simultaneous interactions with the database
 *
 * so this is really a good pattern we should follow
*/
const getDb = () => {
    if(_db){
        return _db
    }
    throw 'No database found!'
}

exports.mongoConnect = mongoConnect
exports.getDb = getDb
```

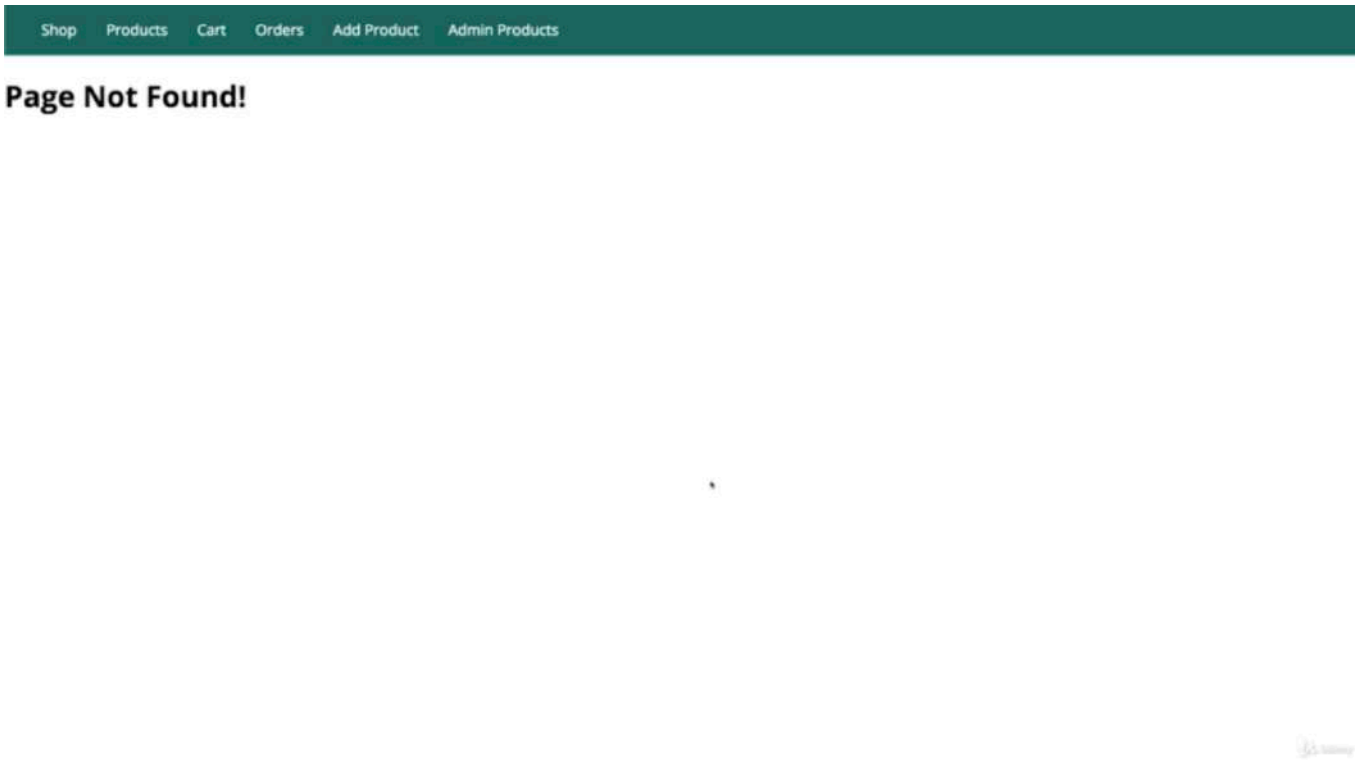
* Chapter 179: Using The Database Connection



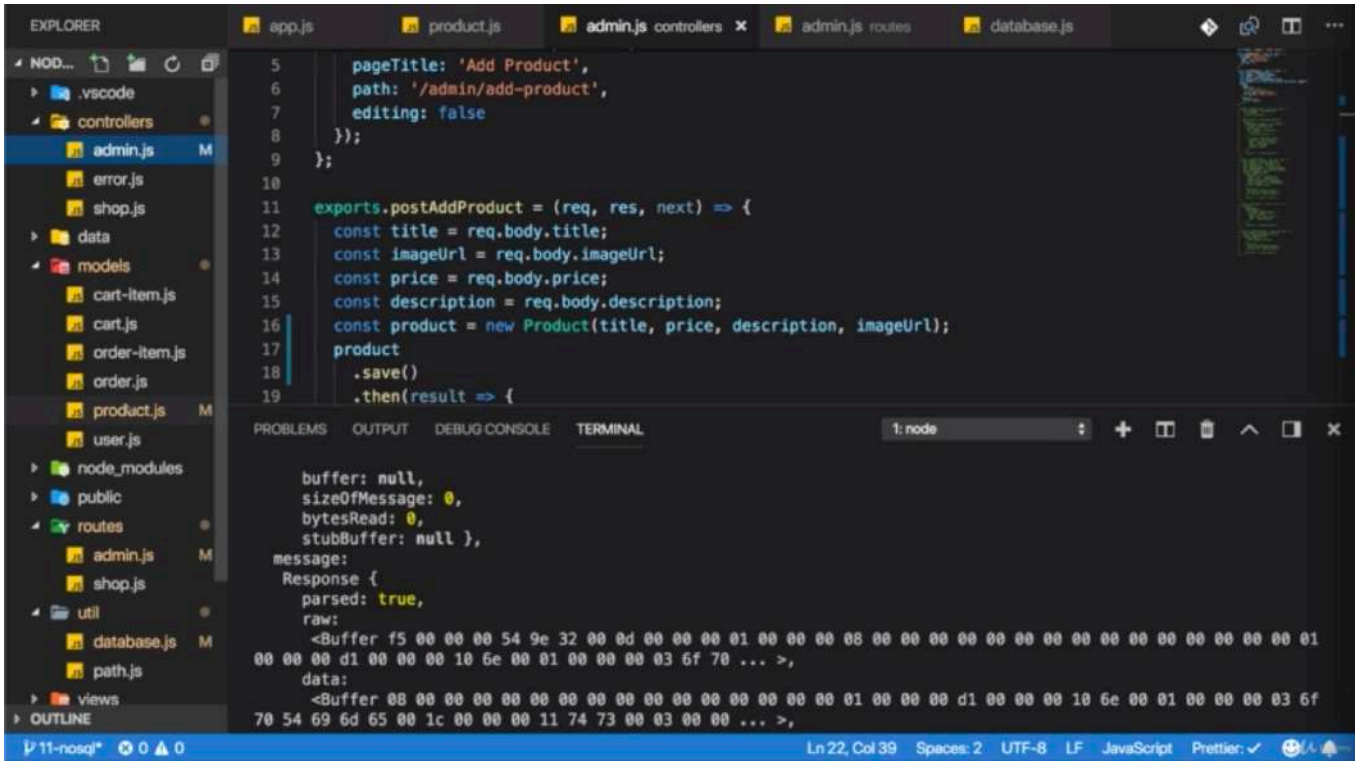
* Chapter 180: Creating Products

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





- 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.



```
EXPLORER
app.js | product.js | admin.js controllers | admin.js routes | database.js
NOD...
.vscode
controllers
admin.js
error.js
shop.js
data
models
cart-item.js
cart.js
order-item.js
order.js
product.js
user.js
node_modules
public
routes
admin.js
shop.js
util
database.js
path.js
views
OUTLINE

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     }

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   1: node

    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 00 08 00 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 ... >,
    data:
     <Buffer 08 00 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 ... >,

11-nosql*  0 0   Ln 16, Col 29 (20 selected)  Spaces: 2  UTF-8  LF  JavaScript  Prettier: ✓
```

- 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.


```
EXPLORER
app.js | product.js | admin.js controllers | admin.js routes | database.js
NODEJS-COMPLETE-GUIDE
.vscode
controllers
admin.js
error.js
shop.js
data
models
cart-item.js
cart.js
order-item.js
order.js
product.js
user.js
node_modules
public
routes
admin.js
shop.js
util
database.js
path.js
views
OUTLINE

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     }

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   1: node

    hashedName: '5e4da7c4ea503007e6b557458673b8253d232290' },
  ops:
   [ Product {
       title: 'This is a test',
       price: '12.99',
       description: 'fdsfadsf',
       imageUrl:
        'http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',
       _id: 5baa18a17afb5a3725b3d31f } ],
  insertedCount: 1,
  insertedId: 5baa18a17afb5a3725b3d31f }
Created Product

11-nosql*  0 0   Ln 16, Col 29 (20 selected)  Spaces: 2  UTF-8  LF  JavaScript  Prettier: ✓
```

- 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.

```
//app.js
```

```

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database').mongoConnect;

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
  // User.findById(1)
  //   .then(user => {
  //     req.user = user;
  //     next();
  //   })
  //   .catch(err => console.log(err));
  next();
});

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect(() => {
  app.listen(3000);
});

```

```
//./models/product.js

/**we are importing something
 * which allows us to get access to the database connection we set up initially
 * when starting our server which now is a concept that we can reuse
 */
const getDb = require('../util/database').getDb

class Product {
  constructor(title, price, description, imageUrl){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
  }
  save(){
    const db = getDb()
```

```
    /**you can call 'collection()'
     * to tell mongoDB into which collection you wanna insert something
     * or with which collection you wanna work
     * remember in MongoDB, you have database, collections, documents
     *
     * we have database connection here
     * so the next level is a collection
     * we can connect to any collection and just as with the database,
     * if it doesn't exist yet, it will be created the first time you insert data.
     *
     * on that collection, we can execute a couple of MongoDB commands or operations
     * if you wanna insert data,
     * you can do this with 'insertOne()'
     * or if it's multiple documents at once, 'insertMany()'
     * which takes an array of javascript objects you wanna insert
     */

     /**here our case, it's the Product which we wanna insert
      * so we could just say 'this'
      * and see how that works
      */
    return db.collection('products')
      .insertOne(this)
      .then(result => {
        console.log(result)
      })
      .catch(err => {
        console.log(err)
      })
  }
}

module.exports = Product;


```

```
// ./controllers/admin.js

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const product = new Product(title, price, description, imageUrl);
  product
    .save()
    .then(result => {
```

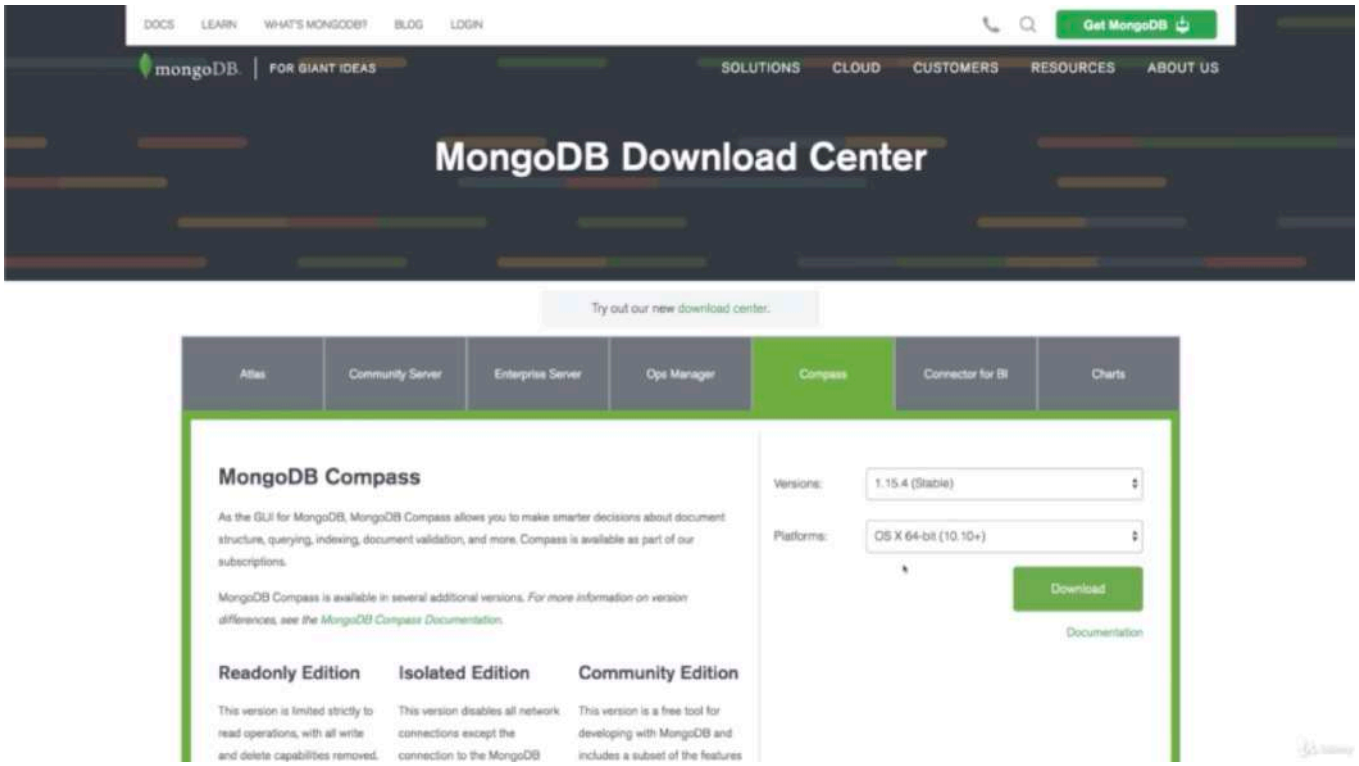
```
    // console.log(result);
    console.log('Created Product');
    res.redirect('/admin/products');
  })
  .catch(err => {
    console.log(err);
  });
};

/*
exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  req.user
    .getProducts({ where: { id: prodId } })
  //Product.findByPk(prodId)
  /**keep in mind we get back an array even if it only holds one element.
   * so we got 'products'
   * and therefore we know that one product,
   * the one we are interested in will always be the first element
   * so we have to store that separately in a new constant */
/*
  .then(products => {
    const product = products[0]
    if (!product) {
      return res.redirect('/');
    }
    res.render('admin/edit-product', {
      pageTitle: 'Edit Product',
      path: '/admin/edit-product',
      editing: editMode,
      product: product
    });
  })
  .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;
  Product.findByPk(prodId)
    .then(product => {
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save();
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
```

```
    res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  req.user
    .getProducts()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return product.destroy()
    })
    .then(result => {
      console.log('DESTROYED PRODUCT')
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err))
};
*/
```

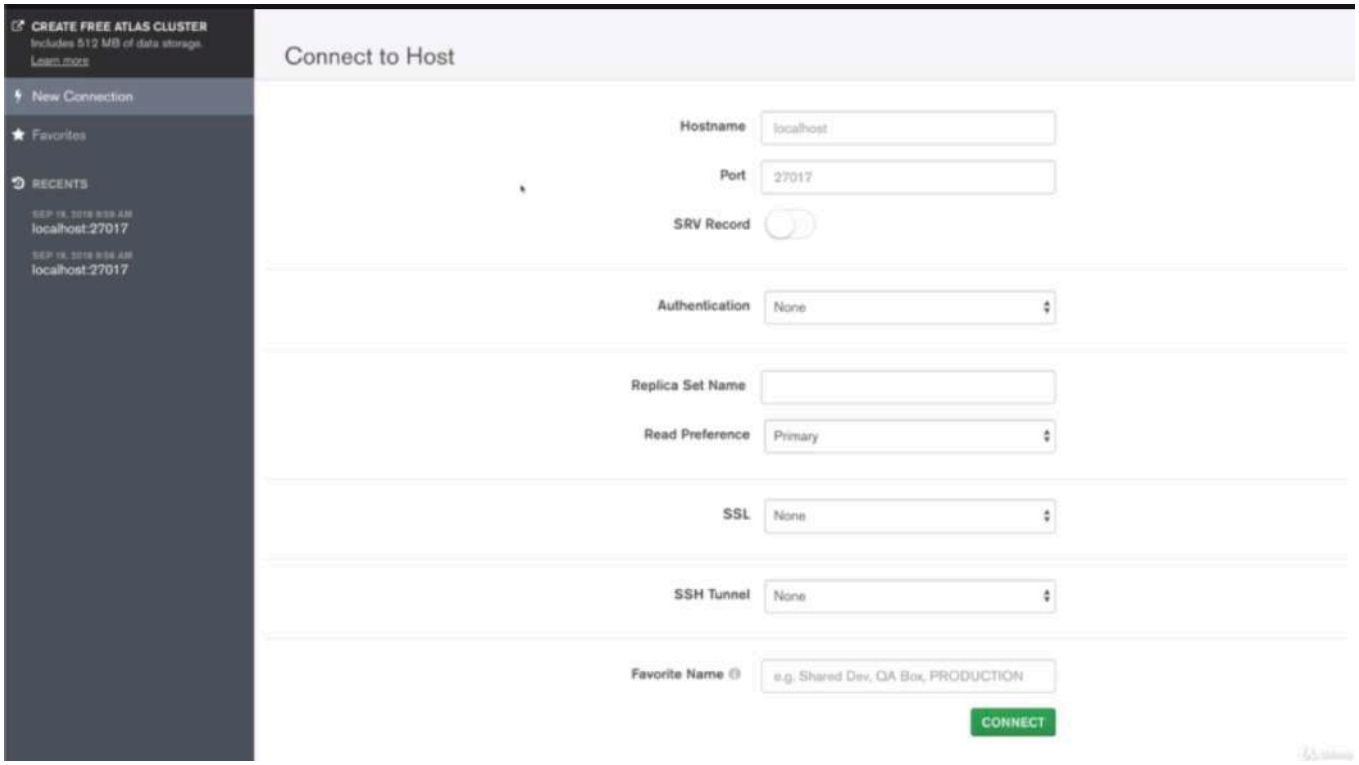
```
// ./routes/admin.js

const path = require('path');

const express = require('express');

const adminController = require('../controllers/admin');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', adminController.getAddProduct);

/*
// /admin/products => GET
router.get('/products', adminController.getProducts);
*/

// /admin/add-product => POST
router.post('/add-product', adminController.postAddProduct);

/*
router.get('/edit-product/:productId', adminController.getEditProduct);

router.post('/edit-product', adminController.postEditProduct);
```

```

router.post('/delete-product', adminController.postDeleteProduct);
*/

module.exports = router;
```

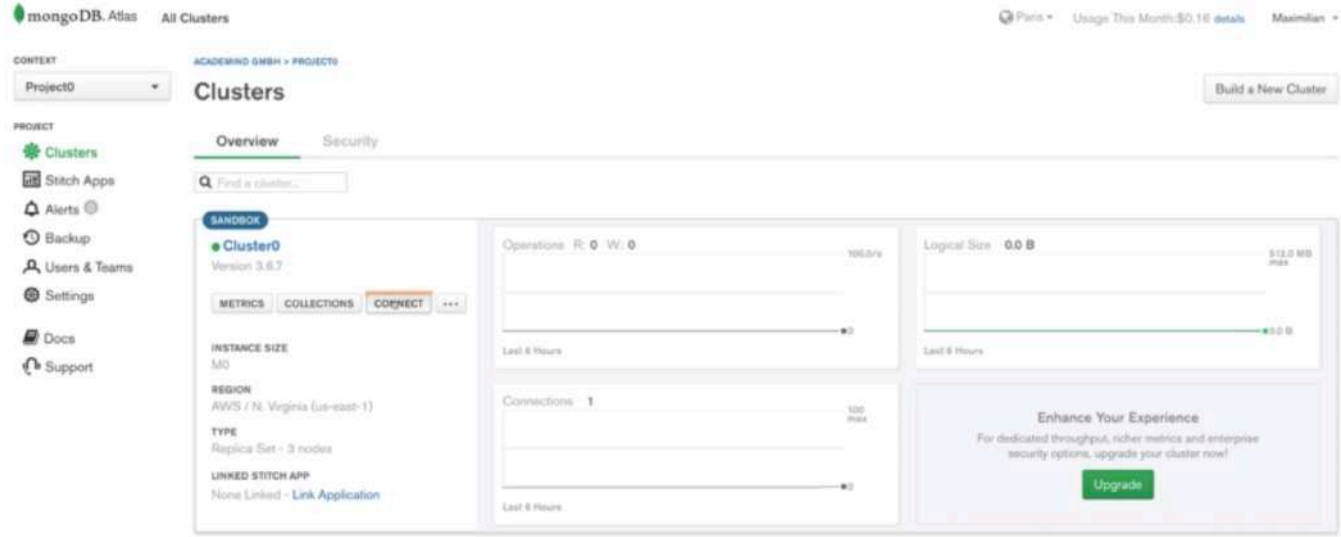
* Chapter 181: Understanding The MongoDB Compass

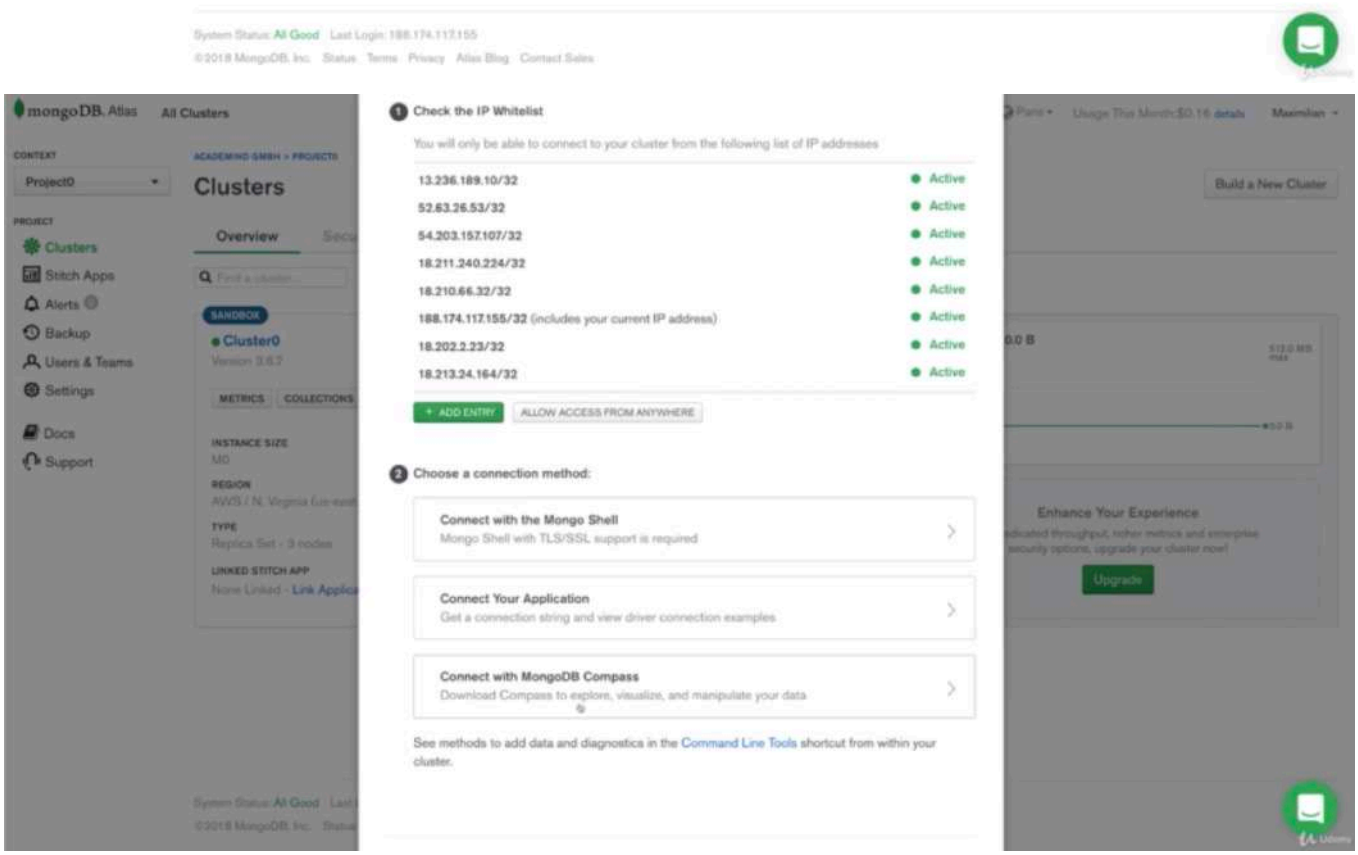


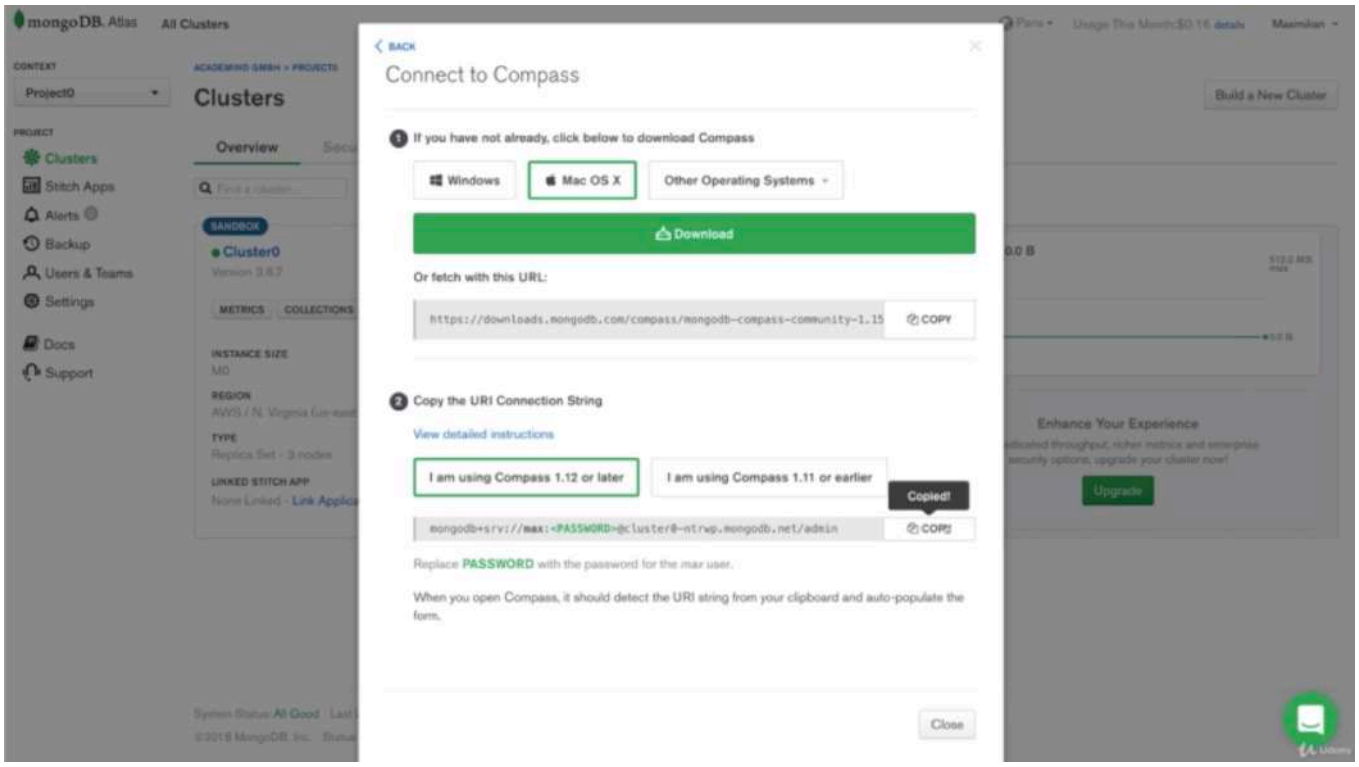
- you can choose that and you can download and install MongoDB Compass for free as well.



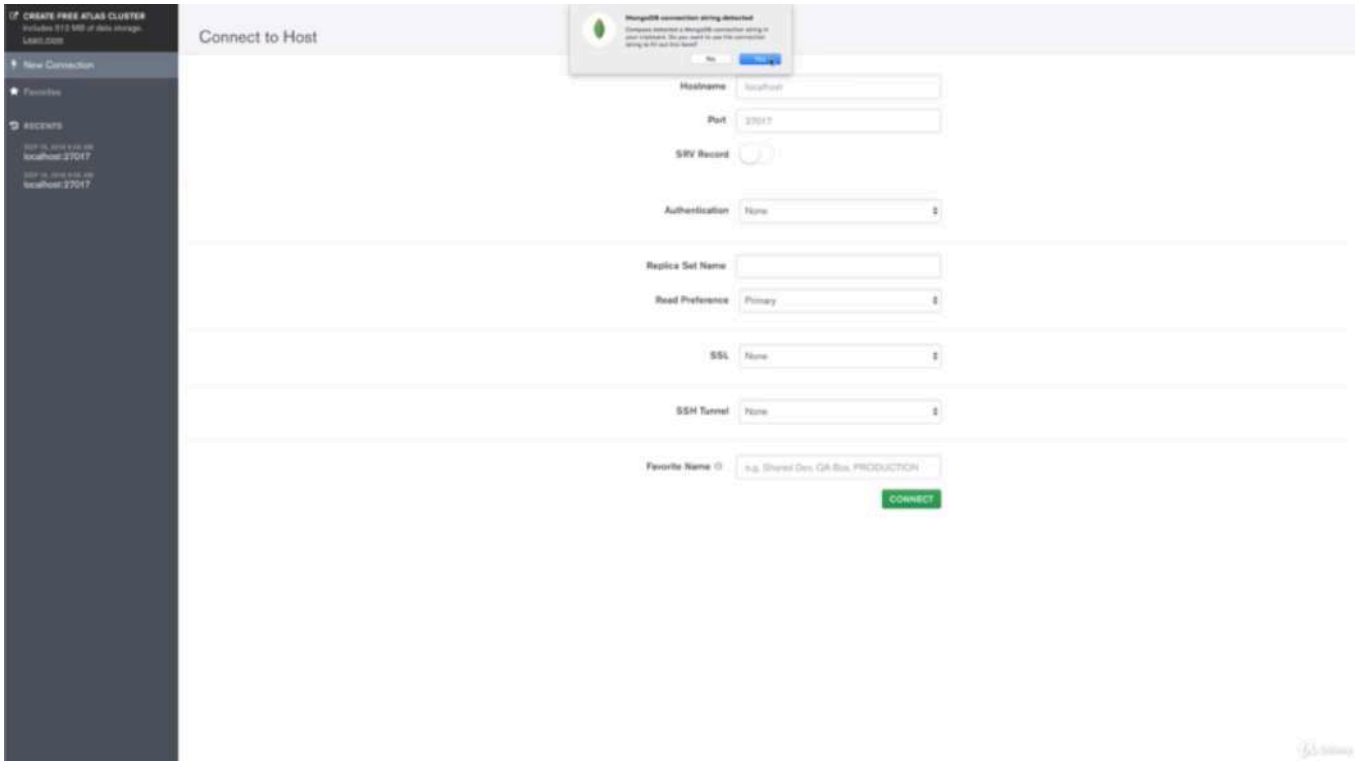
- once you get that installed, you can start your compass application on your machine. and this essentially is a tool that gives you a graphical user interface to connect to your database and to interact with it.
- once it did start up, you can connect to it.



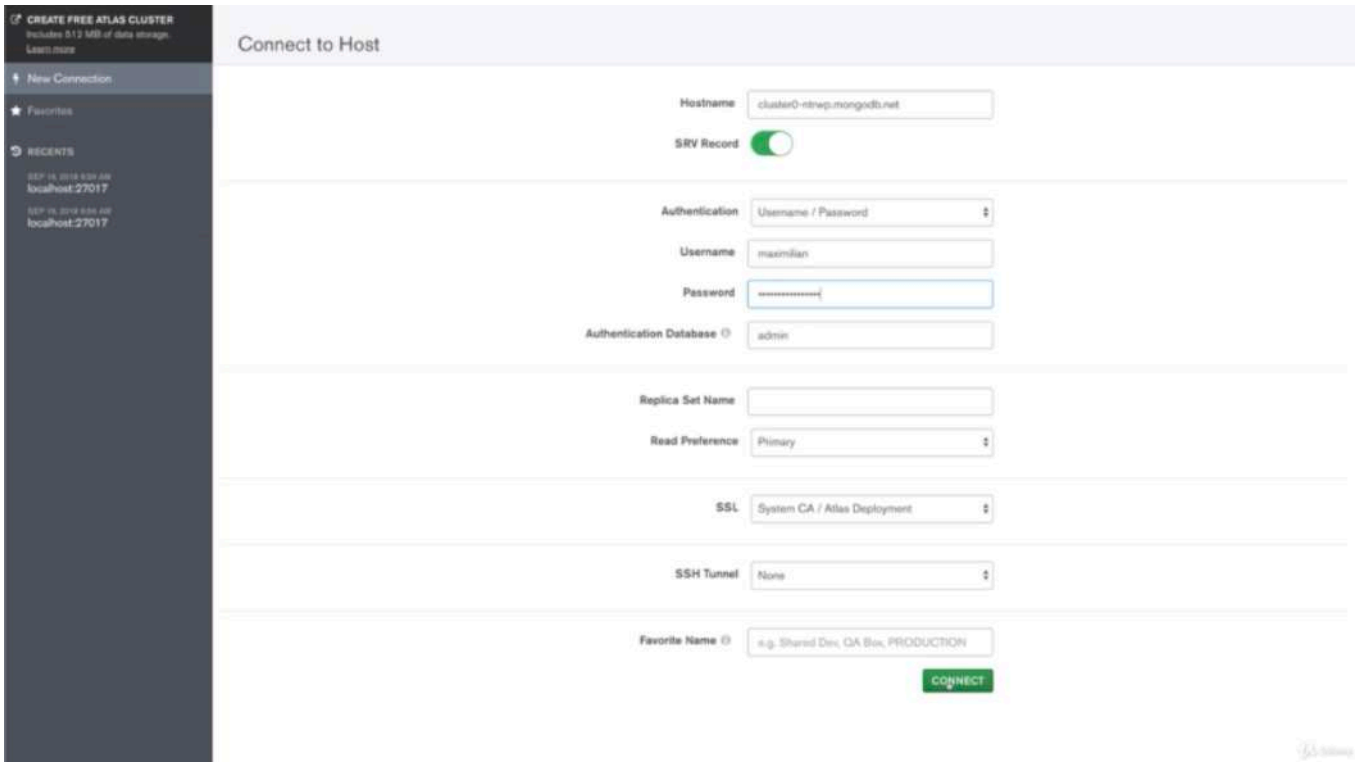




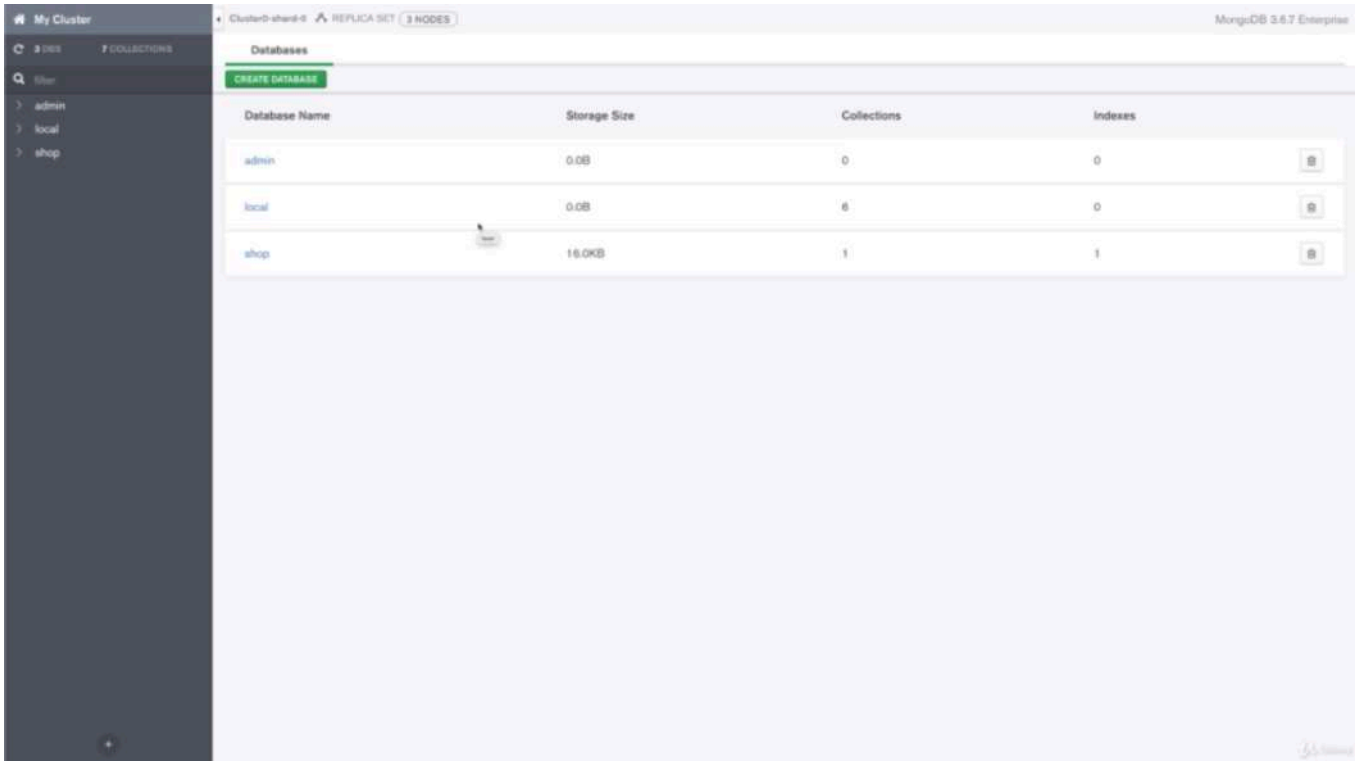
- to connect to it, go back to our MongoDB cluster and click ‘connect’ here and then click ‘connect with MongoDB Compass’ and choose your operating system and then copy that URL down there.



- 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.

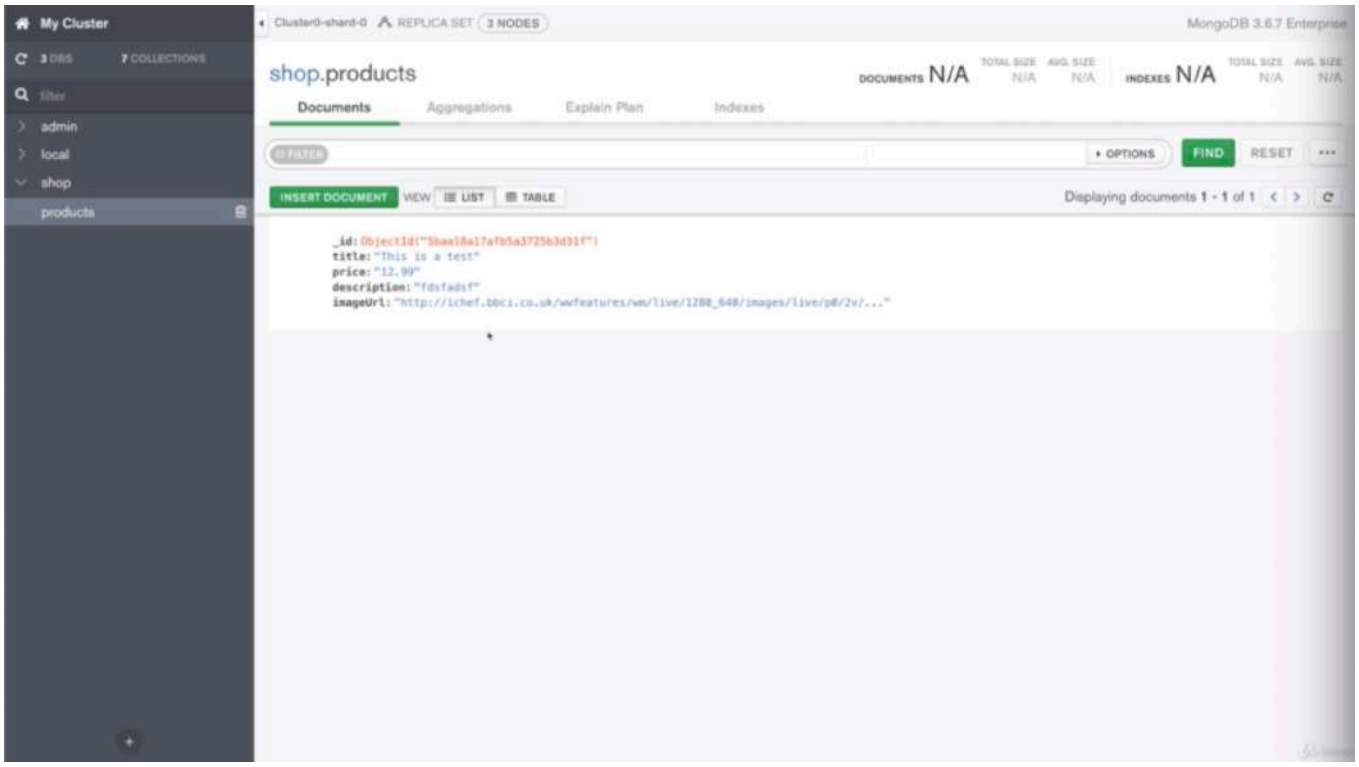


- you still need to choose how you wanna connect, make sure that your username is correct and also enter the password for this user. all the defaults can be left as they are and you should be able to now connect to your database.



- 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'

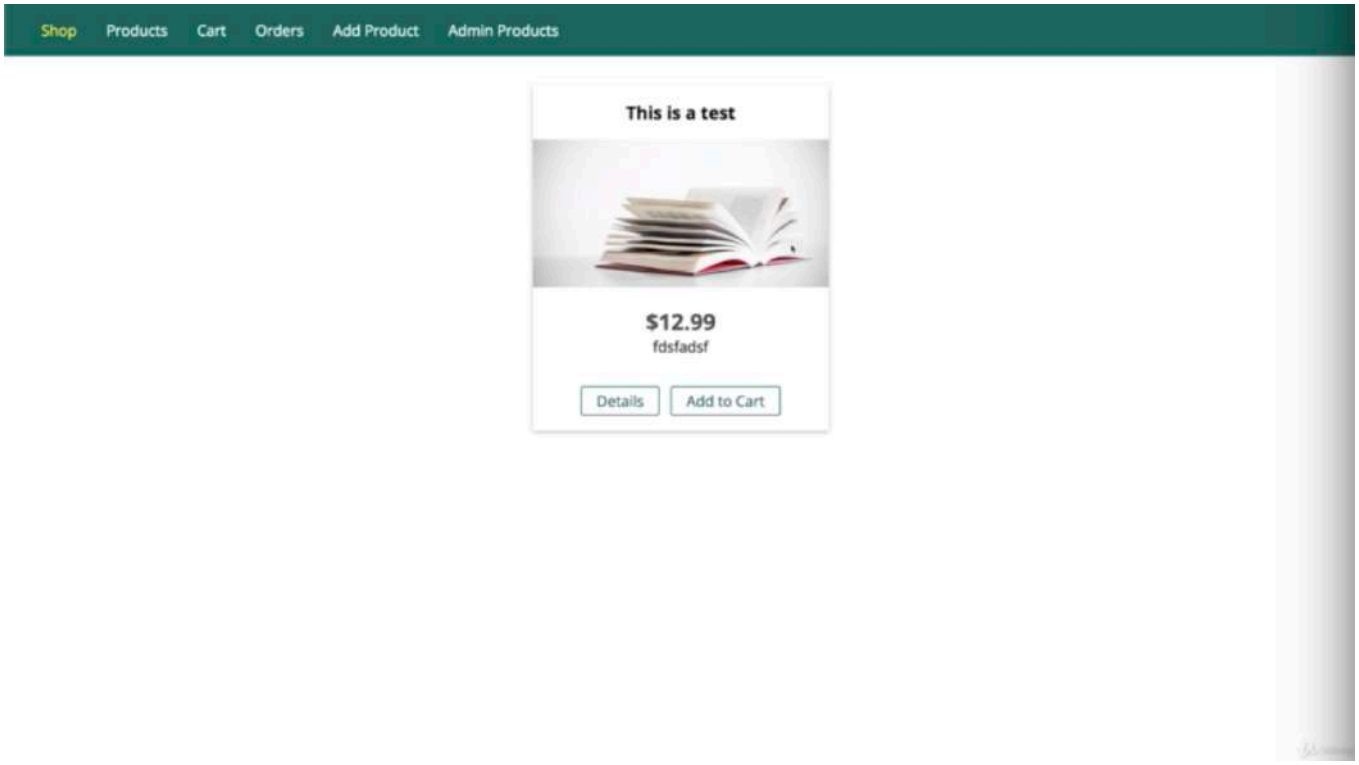
- and the shop database here has a products collection.



- 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




```
//app.js
```

```

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database').mongoConnect;

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
  // User.findById(1)
  //   .then(user => {
  //     req.user = user;
  //     next();
  //   })
  //   .catch(err => console.log(err));
  next();
});

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect(() => {
  app.listen(3000);
});

```

```
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};
```

```
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.findByPk(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) => {
```

```
  const prodId = req.body.productId;
  let fetchedCart;
  let newQuantity = 1;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      let product;
      if (products.length > 0) {
        product = products[0];
      }

      if (product) {
        const oldQuantity = product.cartItem.quantity;
        newQuantity = oldQuantity + 1;
        return product;
      }
      return Product.findByPk(prodId);
    })
    .then(product => {
      return fetchedCart.addProduct(product, {
        through: { quantity: newQuantity }
      });
    })
    .then(() => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .getCart()
    .then(cart => {
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      const product = products[0];
      return product.cartItem.destroy();
    })
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
```

```
    return cart.getProducts();
  })
  .then(products => {
    return req.user
      .createOrder()
      .then(order => {
        return order.addProducts(
          products.map(product => {
            product.orderItem = { quantity: product.cartItem.quantity };
            return product;
          })
        );
      })
      .catch(err => console.log(err));
  })
  .then(result => {
    return fetchedCart.setProducts(null);
  })
  .then(result => {
    res.redirect('/orders');
  })
  .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({include: ['products']})
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

```
// ./routes/shop.js

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

/*
router.get('/products/:productId', shopController.getProduct);

router.get('/cart', shopController.getCart);

```

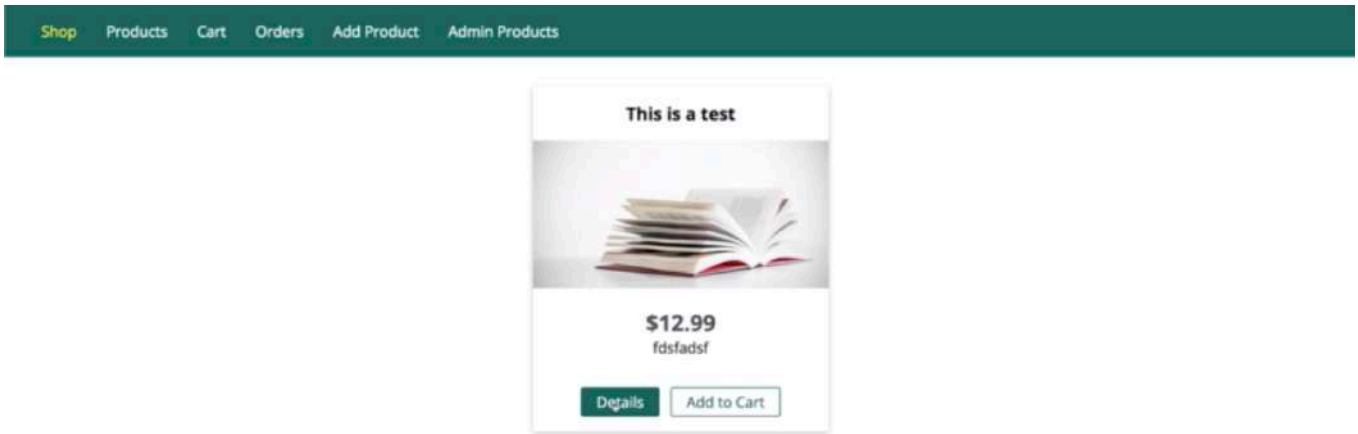
```
router.post('/cart', shopController.postCart);

router.post('/cart-delete-item', shopController.postCartDeleteProduct);

router.post('/create-order', shopController.postOrder)

router.get('/orders', shopController.getOrders);
*/

module.exports = router;

```

```
//./models/product.js

const getDb = require('../util/database').getDb

class Product {
  constructor(title, price, description, imageUrl){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
  }
  save(){
    const db = getDb()
    return db.collection('products')
      .insertOne(this)
      .then(result => {
        console.log(result)
      })
      .catch(err => {
        console.log(err)
      })
  }

  static fetchAll(){
    const db = getDb()
    /**here i wanna interact with my MongoDB database
     * to 'fetchAll()' products
     *
     * MongoDB has a method for finding data
     * which is called 'find()'
     * 'find()' could be configured to also use a filter
     * i wanna find all products which i can do
     * by calling 'find()'
     *
     * the important thing is that
     * 'find()' doesn't immediately returna promise.
     * instead it returns so-called 'cursor' which is an object provieded by MongoDB
     * a 'cursor' is an object provided by MongoDB
     * which allows us to go through our elements, our documents step by step
     * because theoretically in a collection,
     * 'find()' could return millions of documents
     * and you don't wanna transfer them over the wire all at once.
     * instead 'find()' gives you a handle which you can use to tell MongoDB
     * OK give me the next document, OK give me the next document and so on.
     *
```

```
    * there's 'toArray()' method you can execute to tell MongoDB
    * to get all documents and turn them into a javascript array.
    * but you should only use that if we are talking about a couple of dozens or maybe 1 hundred documents.
    * otherwise it's better to implemenet pagination
    * which is something we will implement at a later point of time.
    *
    * */
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products)
        return products
      })
      .catch(err => {
        console.log(err)
      })
  }
}

module.exports = Product;


```

* 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



EXPLORER

NODEJS-COMPLETE-GUIDE

- views
  - admin
    - edit-product.ejs
    - products.ejs
  - includes
    - add-to-cart.ejs
    - end.ejs
    - head.ejs
    - navigation.ejs
  - shop
    - cart.ejs
    - checkout.ejs
    - index.ejs M
    - orders.ejs
    - product-detail...
    - product-li... M
  - 404.ejs
- .gitignore
- app.js
- message.txt
- package-lock.json

OUTLINE

app.js | product.js | shop.js controllers | shop.js routes | index.ejs | product-list.ejs

```
21                  <div class="card__content">
22                      <h2 class="product__price">$
23                          <%= product.price %>
24                      </h2>
25                      <p class="product__description">
26                          <%= product.description %>
27                      </p>
28                  </div>
29                  <div class="card__actions">
30                      <a href="/products/<%= product._id %>" class="btn">Details</a
31                      <%- include('../includes/add-to-cart.ejs', {product: product}
32                  </div>
33              </article>
34              <% } %>
35          </div>
36          <% } else { %>
37              <h1>No Products Found!</h1>
38              <% } %>
39      </main>
40      <%- include('../includes/end.ejs') %>
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | 1: node

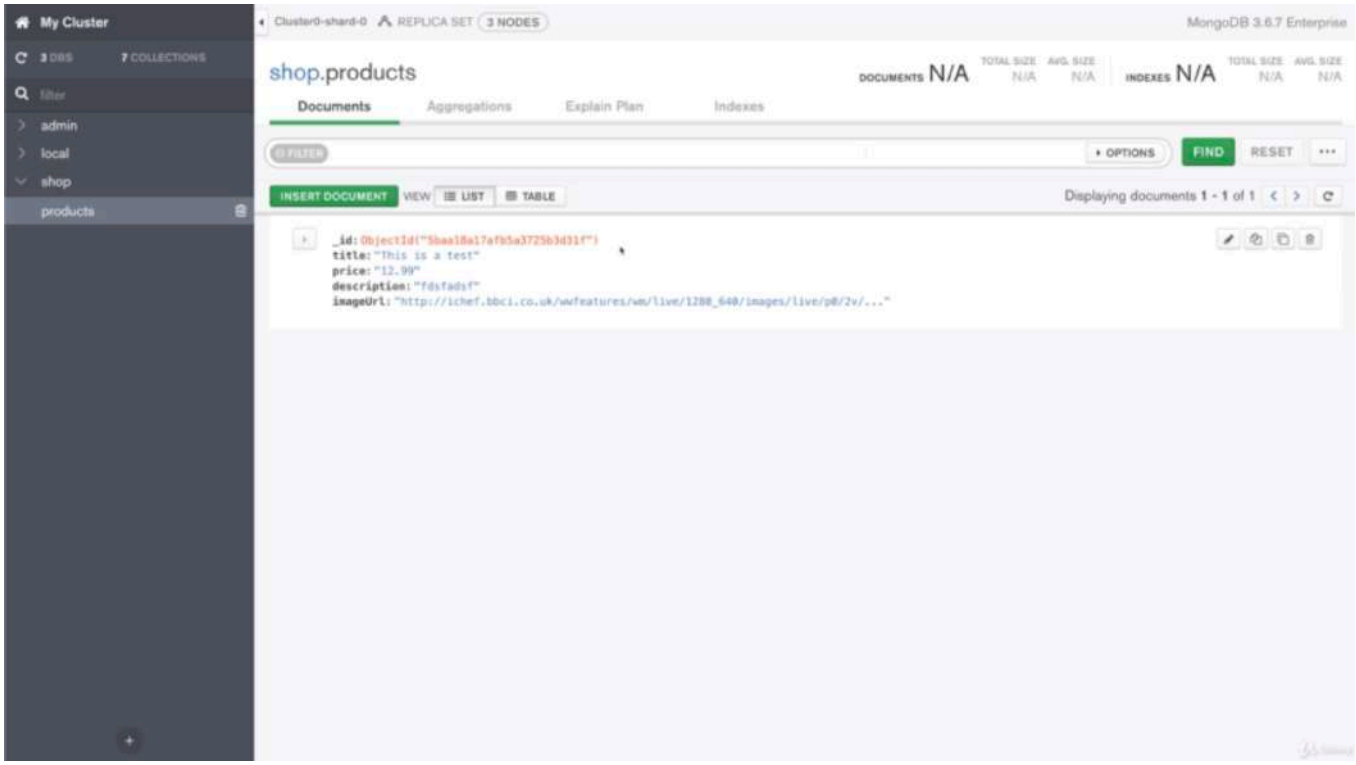
```
null
TypeError: Cannot read property 'title' of null
    at Product.findById.then.product (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/con
trollers/shop.js:32:28)
    at process._tickCallback (internal/process/next_tick.js:68:7)
```

11-nosql* ⊗ 0 ⚠ 0 | Ln 30, Col 69 | Spaces: 4 | UTF-8 | LF | Django Template

```
      .catch(err => {
        console.log(err);
      });
  }

  static findById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .find({ _id: prodId })
      .next()
      .then(product => {
        console.log(product);
        return product;
      })
      .catch(err => {
        console.log(err);
      });
  }
}
```

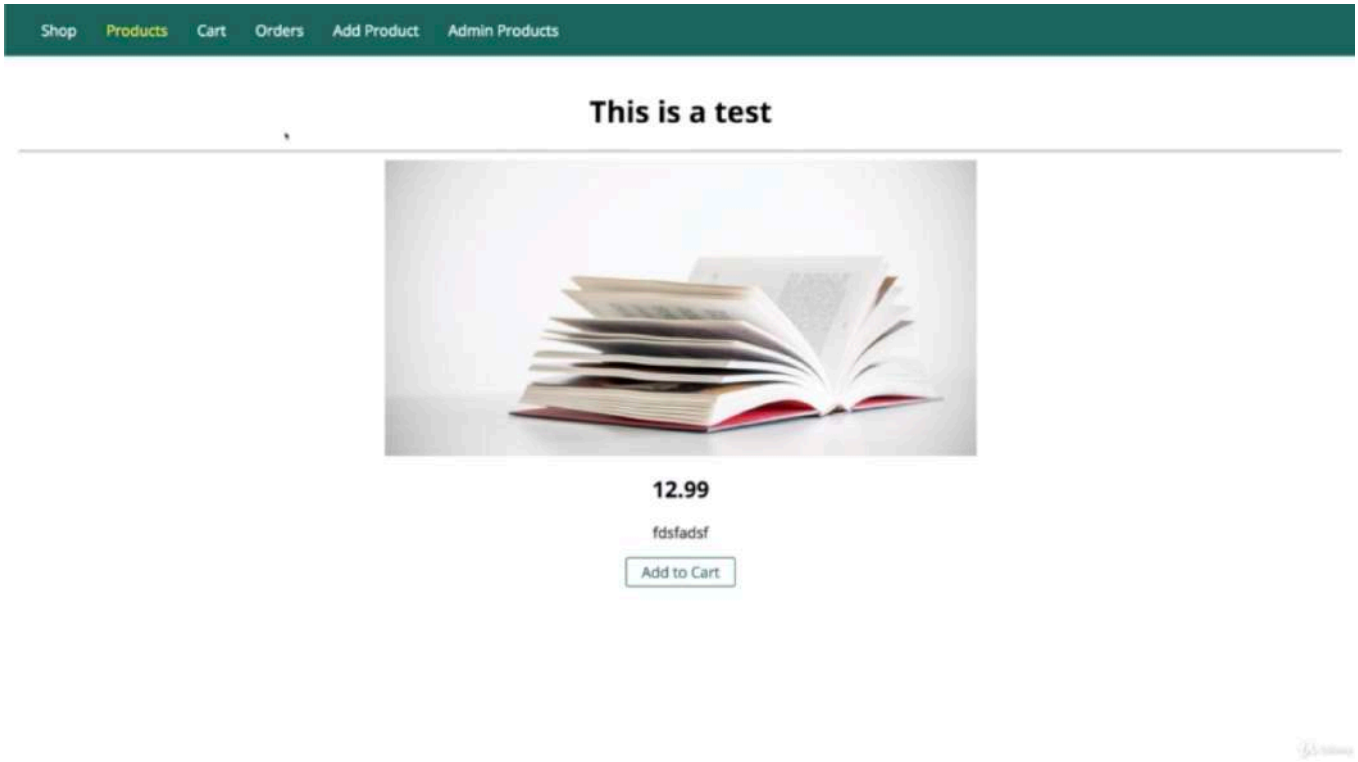
```
null
TypeError: Cannot read property 'title' of null
    at Product.findById.then.product (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/controllers/shop.js:32:28)
    at process._tickCallback (internal/process/next_tick.js:68:7)
```

- 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.

```
// ./routes/shop.js

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);
/*
router.get('/cart', shopController.getCart);

router.post('/cart', shopController.postCart);

router.post('/cart-delete-item', shopController.postCartDeleteProduct);

router.post('/create-order', shopController.postOrder)

```

```
router.get('/orders', shopController.getOrders);
*/

module.exports = router;

```

```
//./models/product.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

class Product {
  constructor(title, price, description, imageUrl){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
  }
  save(){
    const db = getDb()
    return db.collection('products')
      .insertOne(this)
      .then(result => {
        console.log(result)
      })
      .catch(err => {
        console.log(err)
      })
  }

  static fetchAll(){
    const db = getDb()
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products)
        return products
      })
      .catch(err => {
        console.log(err)
      })
  }

  static findById(prodId){
    const db = getDb()
    /**i will find a product here
     * but i will find only one product
     * and to do that,
     * i will narrow down the result set with 'find()'
     * and then i will pass a javascript object to it
     * which allow me to configure a filter and here i wanna look for a product where _id is equal to 'prodId'
     * because that's the ID i'm looking for
     *
     * 'find()' will still give me a cursor
```

```
    * because MongoDB doesn't know that i will only get one
    * an here we can use 'next()' function to get next
    * and in this case also the last document that was returned by 'find()' here.
    */
    return db
      .collection('products')
/** 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)}'
  * 'new' is a constructor
  * and a new objectId to which i pass a string which will be wrapped by that.
  * */
      .find({_id: new mongodb.ObjectId(prodId)})
      .next()
      .then(product => {
        console.log(product)
        return product
      })
      .catch(err => {
        console.log(err)
      })
  }
}

module.exports = Product;


```

```
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      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',
```

```
            products: products
          });
        })
        .catch(err => console.log(err));
    })
    .catch(err => console.log(err));
};

exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  let fetchedCart;
  let newQuantity = 1;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      let product;
      if (products.length > 0) {
        product = products[0];
      }

      if (product) {
        const oldQuantity = product.cartItem.quantity;
        newQuantity = oldQuantity + 1;
        return product;
      }
      return Product.findByPk(prodId);
    })
    .then(product => {
      return fetchedCart.addProduct(product, {
        through: { quantity: newQuantity }
      });
    })
    .then(() => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .getCart()
    .then(cart => {
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      const product = products[0];
      return product.cartItem.destroy();
    })
    .then(result => {
      res.redirect('/cart');
    })
```

```
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts();
    })
    .then(products => {
      return req.user
        .createOrder()
        .then(order => {
          return order.addProducts(
            products.map(product => {
              product.orderItem = { quantity: product.cartItem.quantity };
              return product;
            })
          );
        })
        .catch(err => console.log(err));
    })
    .then(result => {
      return fetchedCart.setProducts(null);
    })
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({include: ['products']})
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

```
<!--./views/shop/index.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/product.css">
</head>

<body>
    <%- include('../includes/navigation.ejs') %>

    <main>
```

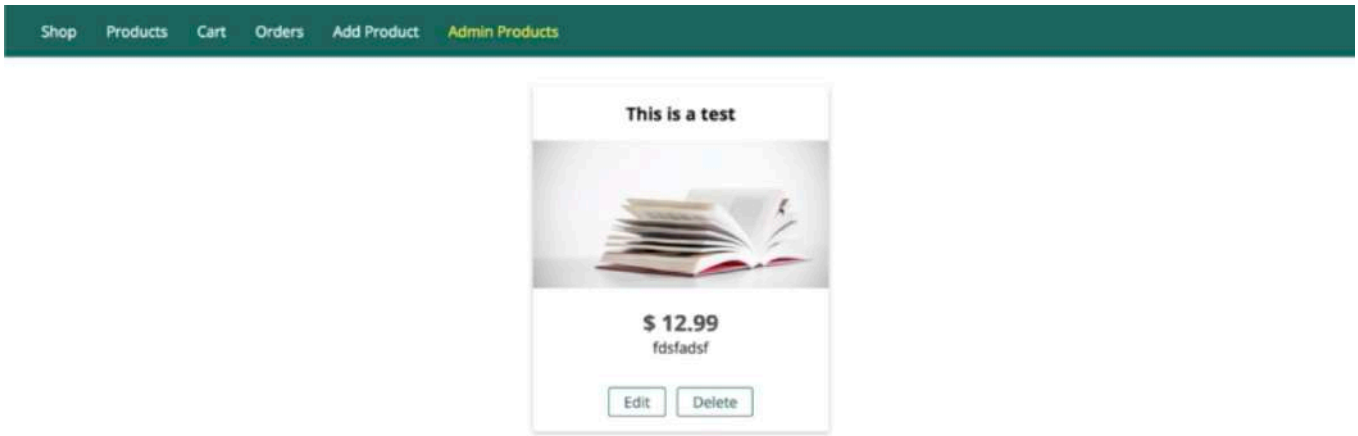
```
        <% if (prods.length > 0) { %>
            <div class="grid">
                <% for (let product of prods) { %>
                    <article class="card product-item">
                        <header class="card__header">
                            <h1 class="product__title"><%= product.title %></h1>
                        </header>
                        <div class="card__image">
                            <img src="<%= product.imageUrl %>"
                                alt="<%= product.title %>">
                        </div>
                        <div class="card__content">
                            <h2 class="product__price">$<%= product.price %></h2>
                            <p class="product__description"><%= product.description %></p>
                        </div>
                        <div class="card__actions">
                            <a href="/products/<%= product._id %>" class="btn">Details</a>
                            <%- include('../includes/add-to-cart.ejs', {product: product}) %>
                        </div>
                    </article>
                <% } %>
            </div>
        <% } else { %>
            <h1>No Products Found!</h1>
        <% } %>
    </main>
<%- include('../includes/end.ejs') %>
```

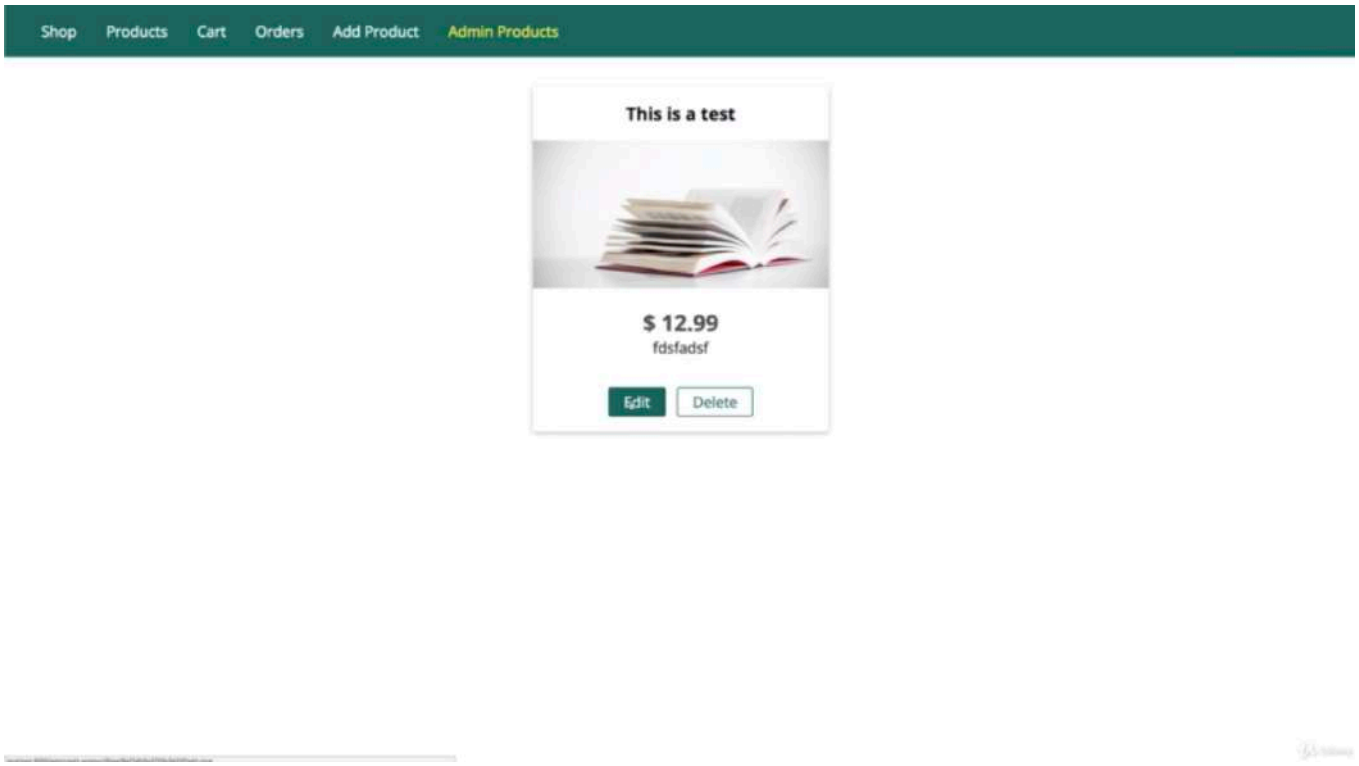
```
<!--./views/shop/product-list.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/product.css">
    </head>

    <body>
        <%- include('../includes/navigation.ejs') %>

            <main>
                <% if (prods.length > 0) { %>
                    <div class="grid">
                        <% for (let product of prods) { %>
                            <article class="card product-item">
                                <header class="card__header">
                                    <h1 class="product__title">
                                        <%= product.title %>
                                    </h1>
                                </header>
                                <div class="card__image">
                                    <img src="<%= product.imageUrl %>" alt="<%= product.title %>">
                                </div>
                                <div class="card__content">
                                    <h2 class="product__price">$
                                        <%= product.price %>
                                    </h2>
                                    <p class="product__description">
```

```
                                <%= product.description %>
                            </p>
                        </div>
                        <div class="card__actions">
                            <a href="/products/<%= product._id %>" class="btn">Details</a>
                            <%- include('../includes/add-to-cart.ejs', {product: product}) %>
                        </div>
                    </article>
                    <% } %>
                </div>
                <% } else { %>
                    <h1>No Products Found!</h1>
                    <% } %>
            </main>
            <%- include('../includes/end.ejs') %>
```

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

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

```
// ./routes/admin.js

const path = require('path');

const express = require('express');

const adminController = require('../controllers/admin');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', adminController.getAddProduct);

// /admin/products => GET
router.get('/products', adminController.getProducts);

// /admin/add-product => POST
router.post('/add-product', adminController.postAddProduct);

/*
router.get('/edit-product/:productId', adminController.getEditProduct);

router.post('/edit-product', adminController.postEditProduct);

router.post('/delete-product', adminController.postDeleteProduct);
*/

module.exports = router;
```

```
// ./controllers/admin.js

const Product = require('../models/product');

```

```
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 };
```

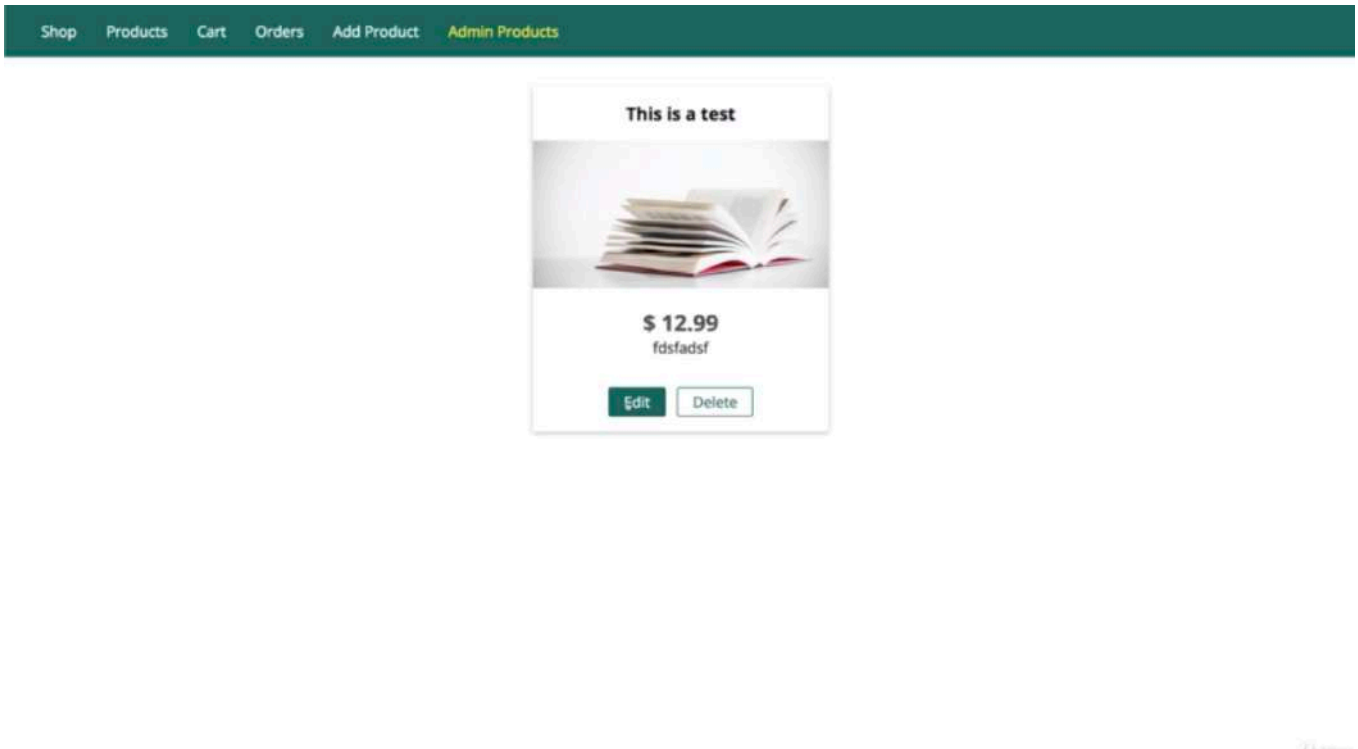
```

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;
  Product.findByPk(prodId)
    .then(product => {
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save();
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
*/

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

/*
exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return product.destroy()
    })
    .then(result => {
      console.log('DESTROYED PRODUCT')
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err))
};
*/
```

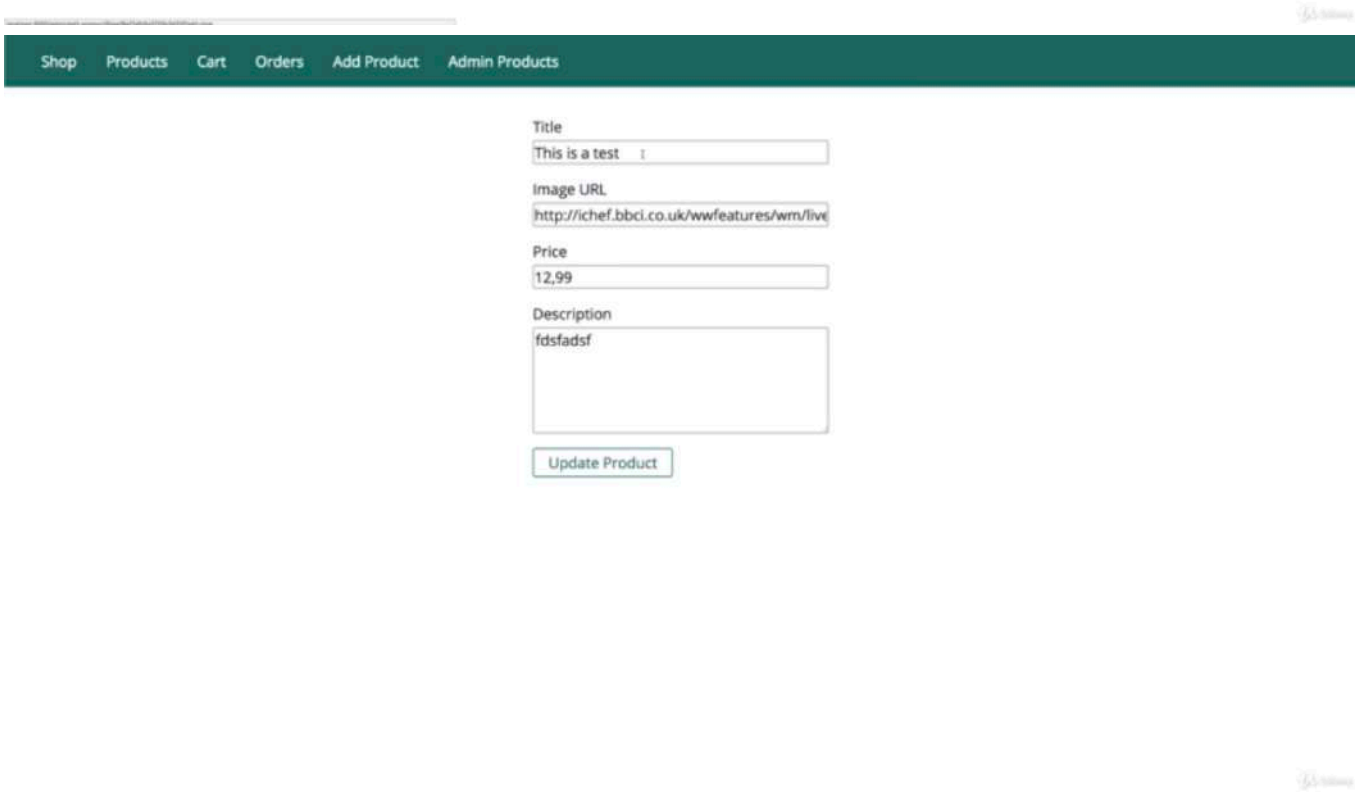
```
<!--./views/admin/products.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/product.css">
    </head>

    <body>
```

```
    <%- include('../includes/navigation.ejs') %>

        <main>
            <% if (prods.length > 0) { %>
                <div class="grid">
                    <% for (let product of prods) { %>
                        <article class="card product-item">
                            <header class="card__header">
                                <h1 class="product__title">
                                    <%= product.title %>
                                </h1>
                            </header>
                            <div class="card__image">
                                <img src="<%= product.imageUrl %>" alt="<%= product.title %>">
                            </div>
                            <div class="card__content">
                                <h2 class="product__price">$
                                    <%= product.price %>
                                </h2>
                                <p class="product__description">
                                    <%= product.description %>
                                </p>
                            </div>
                            <div class="card__actions">
                                <a href="/admin/edit-product/<%= product._id %>?edit=true" class="btn">Edit</a>
                                <form action="/admin/delete-product" method="POST">
                                    <!--set the value to productId using ejs templating syntax and the name to productId
                                        so that we can extract that information by that name.
                                    -->
                                    <input type="hidden" value="<%= product._id %>" name="productId">
                                    <button class="btn" type="submit">Delete</button>
                                </form>

                            </div>
                        </article>
                        <% } %>
                </div>
                <% } else { %>
                    <h1>No Products Found!</h1>
                    <% } %>
        </main>
        <%- 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






```
// ./controllers/admin.js

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
```

```
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.findByPk(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.findByPk(prodId)
61     .then(product => {
62       product.title = updatedTitle;
63       product.price = updatedPrice;
64       product.description = updatedDesc;
65       product.imageUrl = updatedImageUrl;
66       return product.save();
```

```
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

/*
exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return product.destroy()
    })
    .then(result => {
      console.log('DESTROYED PRODUCT')
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err))
};
*/
```

```
<!--./views/admin/edit-product.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/product.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else { %>add-product<% } %>" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title" value="<% if (editing) { %><%= product.title %><% } %>">
            </div>
            <div class="form-control">
                <label for="imageUrl">Image URL</label>
                <input type="text" name="imageUrl" id="imageUrl" value="<% if (editing) { %><%= product.imageUrl %><% } %>">
```

```
            </div>
            <div class="form-control">
                <label for="price">Price</label>
                <input type="number" name="price" id="price" step="0.01" value="<% if (editing) { %><%= product.price %><% } %>">
            </div>
            <div class="form-control">
                <label for="description">Description</label>
                <textarea name="description" id="description" rows="5"><% if (editing) { %><%= product.description %><% } %></textarea>
            </div>
            <% if (editing) { %>
                <input type="hidden" value="<%= product._id %>" name="productId">
            <% } %>

            <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else { %>Add Product<% } %></button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

```
// ./routes/admin.js

const path = require('path');

const express = require('express');

const adminController = require('../controllers/admin');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', adminController.getAddProduct);

// /admin/products => GET
router.get('/products', adminController.getProducts);

// /admin/add-product => POST
router.post('/add-product', adminController.postAddProduct);

router.get('/edit-product/:productId', adminController.getEditProduct);
/*
router.post('/edit-product', adminController.postEditProduct);

router.post('/delete-product', adminController.postDeleteProduct);
*/

module.exports = router;
```

```
//./models/product.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

class Product {
  /**how could we update our product which is stored in the database?
   * let's add a fifth arguement here, the ID
```

```
   * and then i will say 'this._id = id'
   * now we accept a kind of optional fifth argument.
   */
  constructor(title, price, description, imageUrl, id){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
    this._id = id
  }
  save(){
    const db = getDb()
    let dbOp
    if(this._id){
      //Update the product
      /**i use 'updateOne()'
       * and as the name suggests,
       * 'updateOne()' will update exactly one element
       * there's also 'updateMany()' where you can update multiple elements at once.
       */
      dbOp = db.collection('products')
      /**'updateOne()' takes at least 2 arguments
       * the first one is that we add a filter
       * that defines which element or which document we wanna update
       * so i will pass a javascript object
       * and we can filter for equality also or run more complex queries
       * anyway, here i only wanna find a document where the '_id' is equal to
       * and now again '_id: new mongodb.ObjectId(this._id)'
       * so i'm looking for a document where the ID matches the ID i have here in my product i'm currently working with
       *
       * and we now as a second argument to 'updateOne()',
       * we now specify how to update that document.
       * this again is a javascript object where we describe the update
       * and this is now not the new object.
       * so we don't say 'this' here as you could imagine that we tell MongoDB find me the existing document and replace it with this.
       * 'updateOne()' doesn't replace
       *
       * instead we have to describe the operation and we do this by using a special property name
       * which is understood by MongoDB, kind of a reserved name. '$set'
       * '$set' again takes an object as a value
       * and here we describe the changes we wanna make to the existing document which we found with this filter
       * and here you could say 'this' and you would instruct MongoDB to set these key value field like 'this.title = title' 'this.price = price' above.
       * which you have in your object to the document it found in the database
       * it will update the values of the document in the database with your new values.
       * we wanna replace all fields, we can just say {$set: this}
       */
        .updateOne({_id: new mongodb.ObjectId(this._id)}, {$set: this})
    } else {
      dbOp = db
              .collection('products')
              .insertOne(this)
```

```
    }
    return dbOp
      .then(result => {
        console.log(result)
      })
      .catch(err => {
        console.log(err)
      })
  }

  static fetchAll(){
    const db = getDb()
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products)
        return products
      })
      .catch(err => {
        console.log(err)
      })
  }

  static findById(prodId){
    const db = getDb()
    return db
      .collection('products')
      .find({_id: new mongodb.ObjectId(prodId)})
      .next()
      .then(product => {
        console.log(product)
        return product
      })
      .catch(err => {
        console.log(err)
      })
  }
}

module.exports = Product;


```

* Chapter 186: Finishing The “Update Product” Code

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

Shop Products Cart Orders Add Product Admin Products

Title

This is a test

Image URL

http://ichef.bbci.co.uk/wwfeatures/wm/live

Price

19,99

Description

fdsfadsf

Update Product

Shop Products Cart Orders Add Product Admin Products

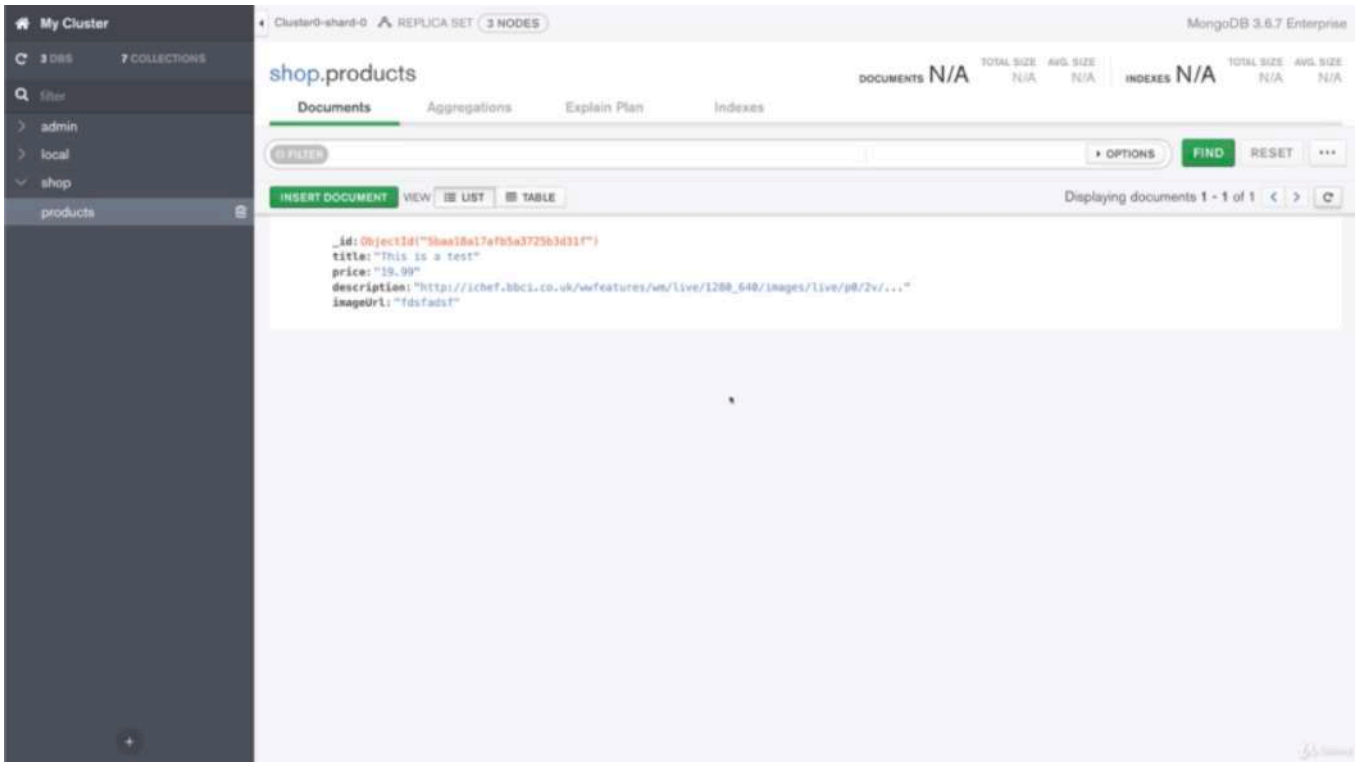
This is a test

This is a test

$ 19.99

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

Edit Delete




```
// ./controllers/admin.js

const mongodb = require('mongodb')
const Product = require('../models/product');

const ObjectId = mongodb.ObjectId

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const product = new Product(title, price, description, imageUrl);
  product
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};


exports.getEditProduct = (req, res, next) => {
```

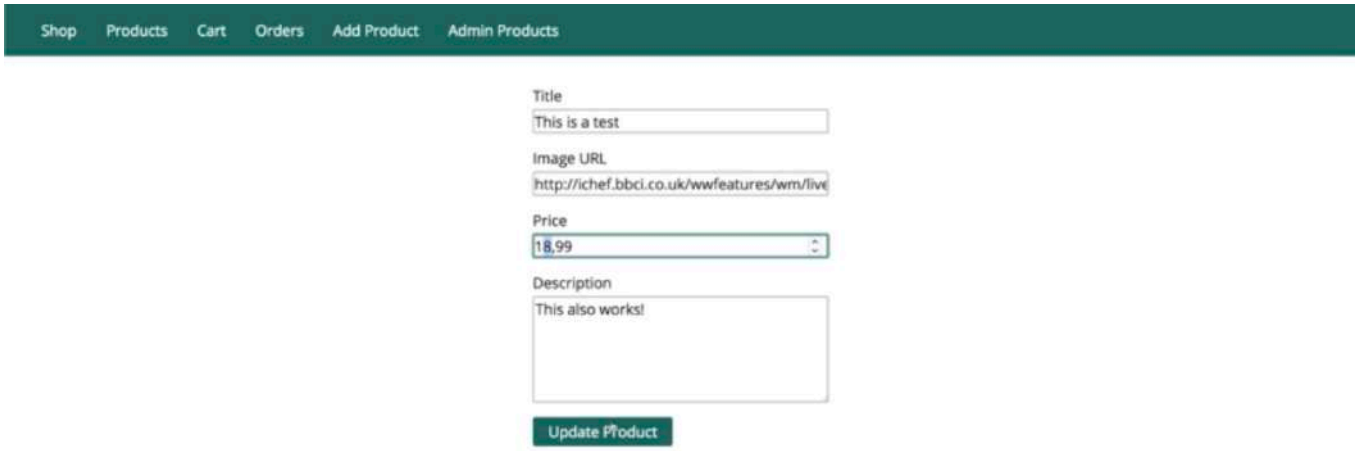
```
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
//Product.findByPk(prodId)
  .then(product => {
      if (!product) {
        return res.redirect('/');
      }
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product
      });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;
  /**we already have code where we find a product by id
   * and this will work because 'findById()' is a method i created in my model
   *
   */
      /**we don't even need 'findById()' and 'then()' anymore
       * we can just get rid of that
       * and for now also create a new product with all the updated information
       * and we also need to pass the product ID,
       * there you just need to make sure that you create a 'new mongodb.ObjectId' object
       * so make sure that at the top of the file,
       * you require 'mongodb'
       * and you can create a new constant, name it 'ObjectId'
       * and extract that 'ObjectId' constructor out of mongodb
       * and then you could write new ObjectId
       * and reference this.
       * so we can now go down to postEditProduct
       */
      const product = new Product(updatedTitle, updatedPrice, updatedDesc, updatedImageUrl, new ObjectId(prodId))
      product
      /**we call 'product.save()'
       * because we modified the 'save()' method to support both creation and updating
       */
        .save()
        .then(result => {
        console.log('UPDATED PRODUCT!');
        res.redirect('/admin/products');
      })
      .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




```
//./models/product.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

class Product {
  constructor(title, price, description, imageUrl, id){
    this.title = title
    this.price = price
    this.description = description
    this.imageUrl = imageUrl
    this._id = new mongodb.ObjectId(id)
  }
  save(){
    const db = getDb()
    let dbOp
    if(this._id){
      dbOp = db.collection('products')
      /**if we go to the ./models/product.js file,
       * and we have a look at the save() method,
       * i'm looking for the right object
       * but i will have a problem with updating it
```

```
     * because i will try set my obejct id to a different object id to a string
     * because i'm referring to these things like 'this.title = title' 'this.price = price' and so on
     * which will hold the unmodified objectId
     * so i should automatically convert the 'objectId' the id which is the string to an obejct
     * and the object in the constructor
     * so that we can remove down there. 'new mongodb.ObjectId'
     * because '_id' will always be an OjbectId field
     * no matter if i'm using it in a filter or if i'm using it for updating.
     */
      .updateOne(
        {_id: this._id},
        {$set: this}
      )
    } else {
      dbOp = db
            .collection('products')
            .insertOne(this)
    }
    return dbOp
      .then(result => {
        console.log(result)
      })
      .catch(err => {
        console.log(err)
      })
  }

  static fetchAll(){
    const db = getDb()
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products)
        return products
      })
      .catch(err => {
        console.log(err)
      })
  }

  static findById(prodId){
    const db = getDb()
    return db
      .collection('products')
      .find({_id: new mongodb.ObjectId(prodId)})
      .next()
      .then(product => {
        console.log(product)
        return product
      })
      .catch(err => {
        console.log(err)
```

```
    })
  }
}

module.exports = Product;


```

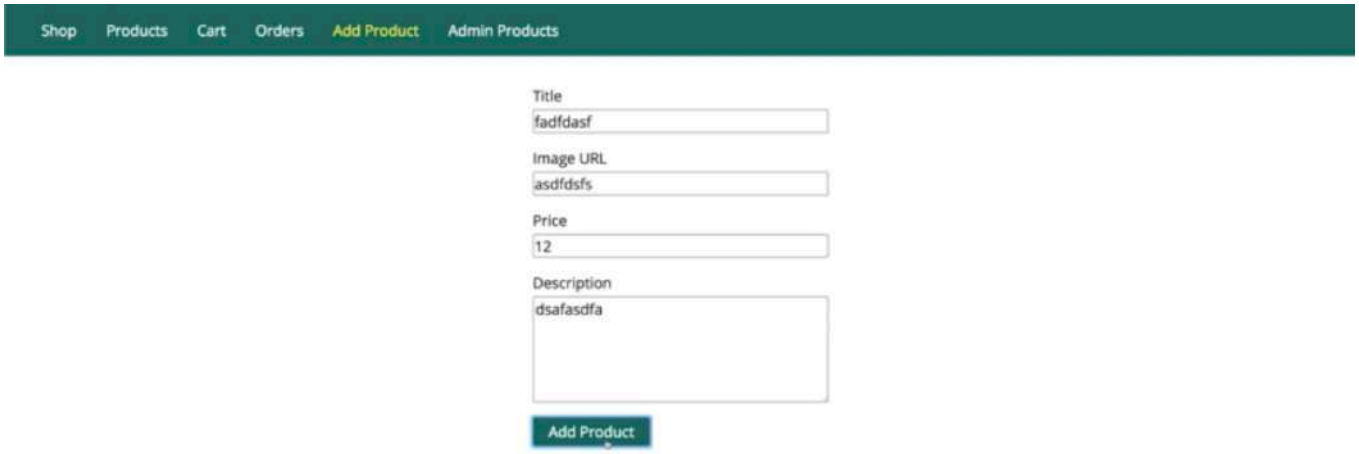
```
// ./controllers/admin.js

const mongodb = require('mongodb')
const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const product = new Product(
    title,
    price,
    description,
    imageUrl
  );
  product
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};


exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
//Product.findByPk(prodId)
  .then(product => {
      if (!product) {
        return res.redirect('/');
      }
```

```
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product
      });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;
      const product = new Product(
        updatedTitle,
        updatedPrice,
        updatedDesc,
        updatedImageUrl,
        /**i passed an 'ObjectId' in ./controllers/admin.js to my Product constructor
         * now i could also just pass 'prodId' as a string
         * and therefore remove my 'ObjectId' constant above import
         * all of that is happening in the controller now
         *
         * this is an important note
         * you don't have to convert the ID in the controller file
         * you can leave that untouched
         * but you can do a general conversion in the ./models/product.js file
         * which is the better approach of doing that.
         */
        prodId
      )
      product
        .save()
        .then(result => {
        console.log('UPDATED PRODUCT!');
        res.redirect('/admin/products');
      })
      .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

/*
```

```
exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return product.destroy()
    })
    .then(result => {
      console.log('DESTROYED PRODUCT')
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err))
};
*/
```

* Chapter 188: Deleting Products

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



Shop Products Cart Orders Add Product Admin Products

This is a test

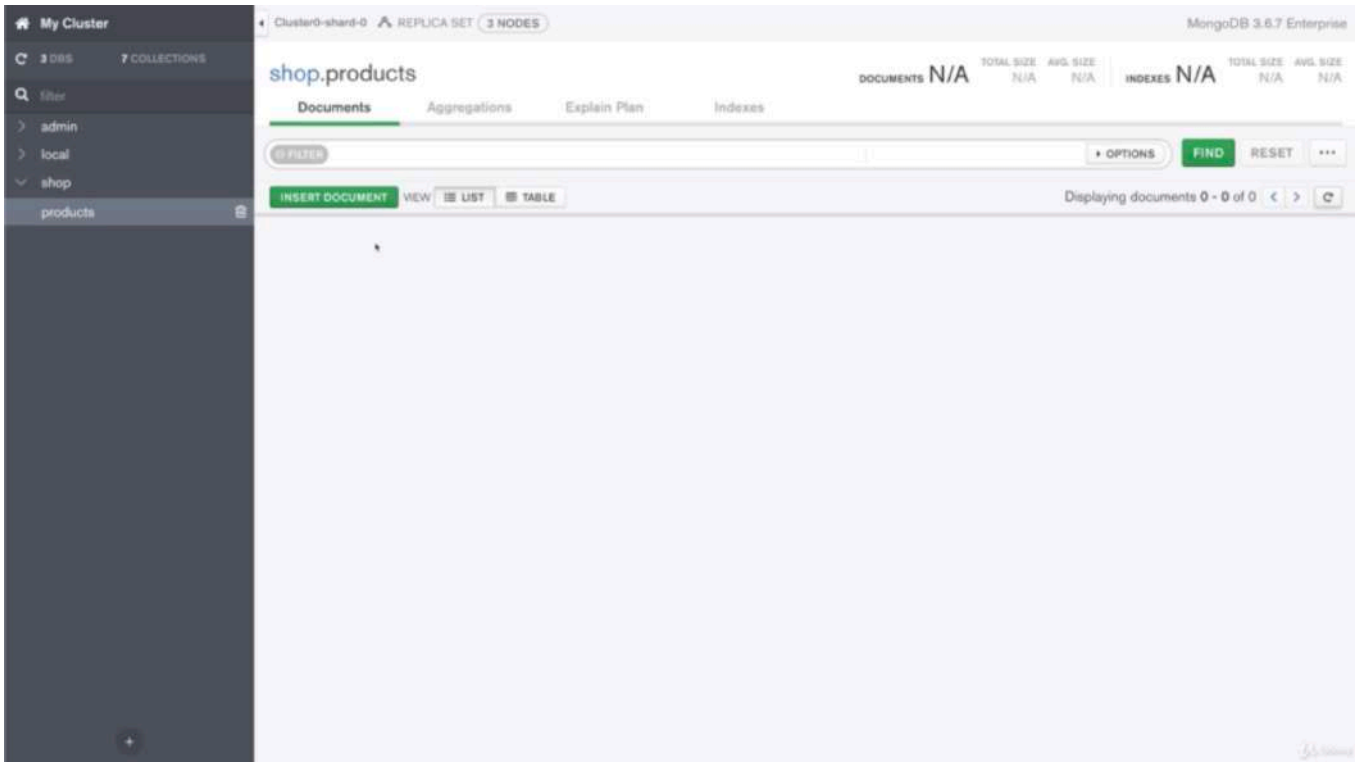
$ 18.99

This also works!

Edit Delete

Shop Products Cart Orders Add Product Admin Products

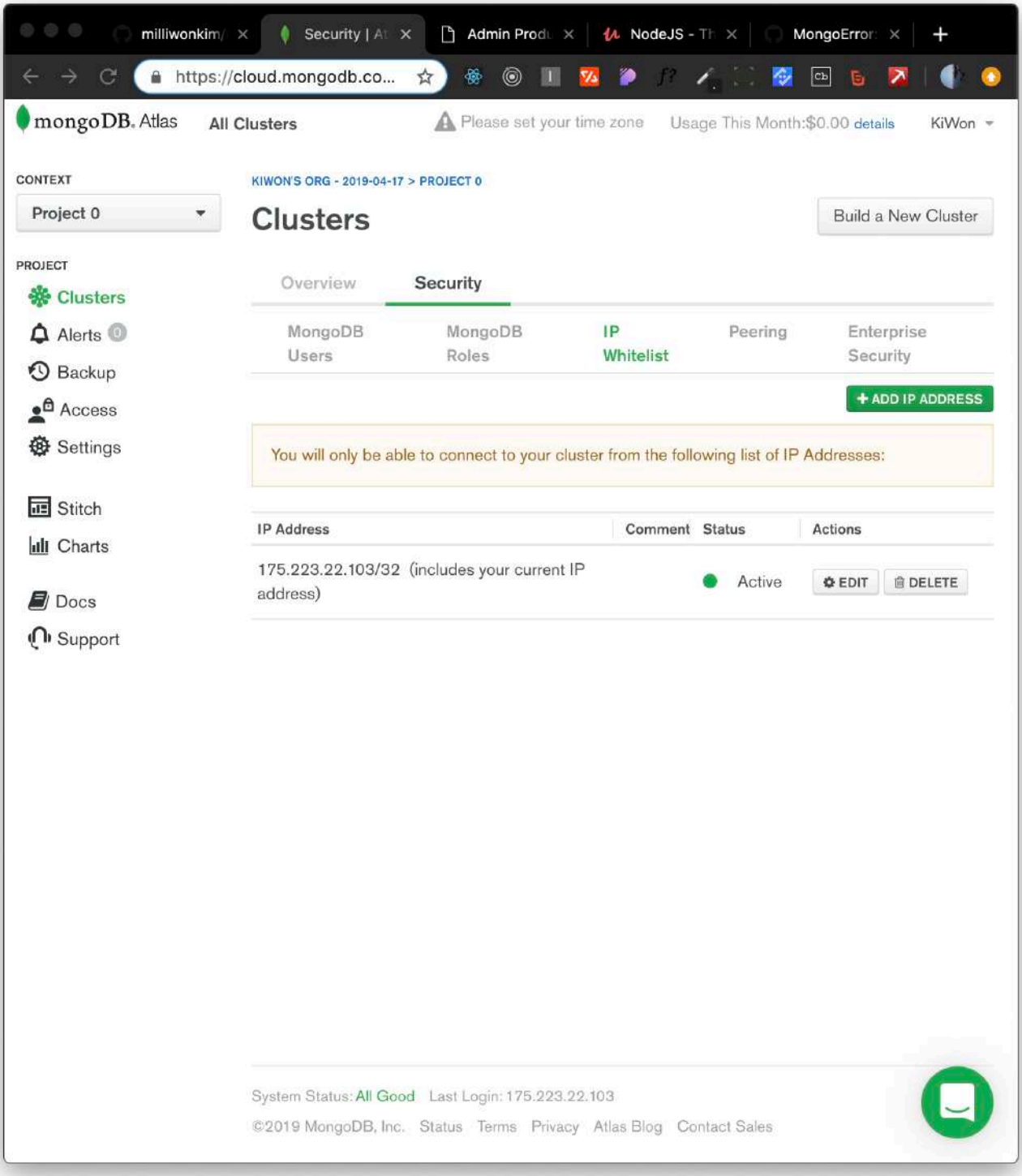
No Products Found!



- 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-z3vlk.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-z3vlk.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 periodly 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 whiltelist. as soon as you edit it, Status of IP Address will be on 'pending' and after that go to 'Active' then Error is gone.




```
//./models/product.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

class Product {
  constructor(title, price, description, imageUrl, id) {
    this.title = title;
```

```
    this.price = price;
    this.description = description;
    this.imageUrl = imageUrl;
    this._id = new mongodb.ObjectId(id);
  }

  save() {
    const db = getDb();
    let dbOp;
    if (this._id) {
      // Update the product
      dbOp = db
        .collection('products')
        .updateOne({ _id: this._id }, { $set: this });
    } else {
      dbOp = db.collection('products').insertOne(this);
    }
    return dbOp
      .then(result => {
        console.log(result);
      })
      .catch(err => {
        console.log(err);
      });
  }

  static fetchAll() {
    const db = getDb();
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products);
        return products;
      })
      .catch(err => {
        console.log(err);
      });
  }

  static findById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .find({ _id: new mongodb.ObjectId(prodId) })
      .next()
      .then(product => {
        console.log(product);
        return product;
      })
      .catch(err => {
        console.log(err);
      });
  }

```

```
  static deleteById(prodId) {
    const db = getDb();
    /**you need to specify a filter now
     * so pass in object where you describe how to find that product
     * and again it will be our _id equal to check here
     * here product id is an argument
     * so we need to convert it to an Object.Id manually again
     * by passing it to the ObjectId constructor.
     * now mongodb will go ahead and delete the first element it finds that has this criteria fulfilled.
     */
    return db
      .collection('products')
      .deleteOne({ _id: new mongodb.ObjectId(prodId) })
      .then(result => {
        console.log('Deleted');
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = Product;
```

```
// ./controllers/admin.js

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const product = new Product(title, price, description, imageUrl);
  product
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
```

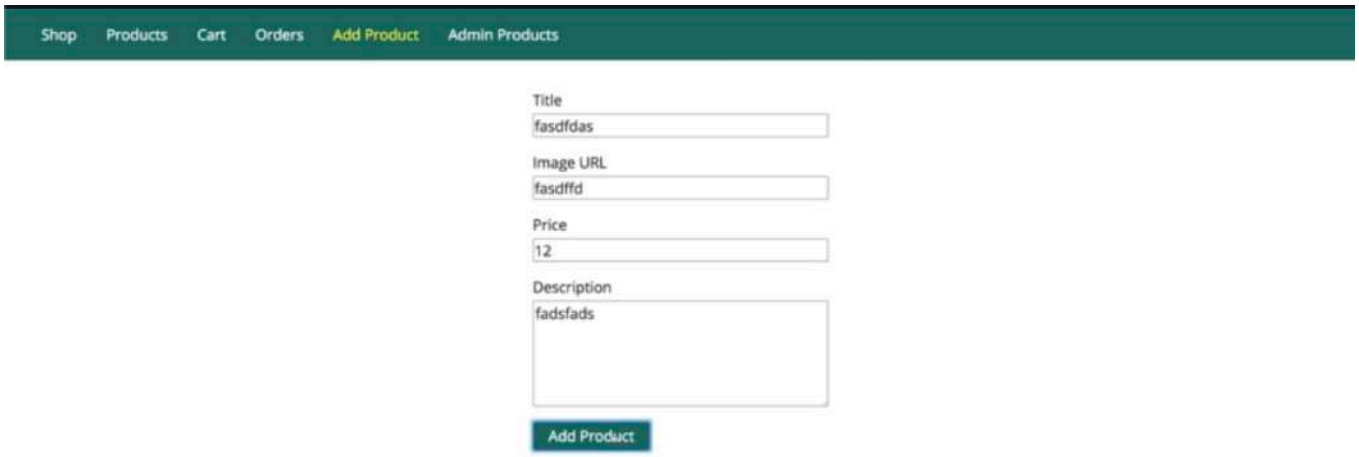
```
33   if (!editMode) {
34     return res.redirect('/');
35   }
36   const prodId = req.params.productId;
37   Product.findById(prodId)
38     // Product.findById(prodId)
39     .then(product => {
40       if (!product) {
41         return res.redirect('/');
42       }
43       res.render('admin/edit-product', {
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) => {
```

```
  /**we extract prodId
   * and here we had a different flow
   * i first of all found the product here
   * we just call 'Product.deleteById(prodId)'
   * we pass in the 'prodId' as a string
   * and then we just have our 'then()' block which won't receive an argument
   */
  const prodId = req.body.productId;
  Product.deleteById(prodId)
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
```

```
// ./routes/admin.js

const path = require('path');

const express = require('express');

const adminController = require('../controllers/admin');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', adminController.getAddProduct);

// /admin/products => GET
router.get('/products', adminController.getProducts);

// /admin/add-product => POST
router.post('/add-product', adminController.postAddProduct);

router.get('/edit-product/:productId', adminController.getEditProduct);

router.post('/edit-product', adminController.postEditProduct);

router.post('/delete-product', adminController.postDeleteProduct);

module.exports = router;
```

* Chapter 189: Fixing The “Add Product” Functionality

1. update
- ./models/product.js



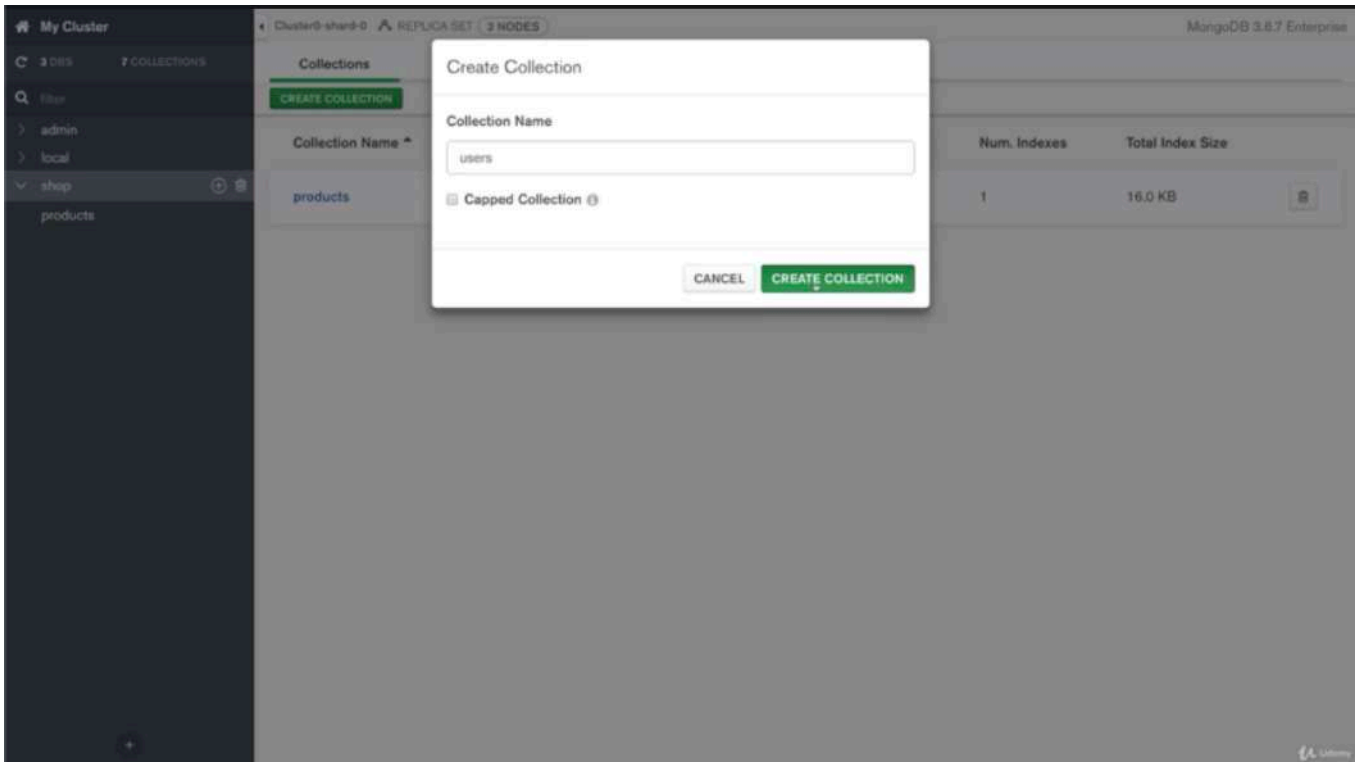

```
//./models/product.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

/**even if we pass no 'id' as an argument and this therefore is undefined,
 * this will create an object and store it in '_id'
 * so 'this._id' down there will always be defined.
 * and if it's just such an empty or automatically generated ObjectId object,
 * this should be the issue here.
 */
class Product {
  constructor(title, price, description, imageUrl, id) {
    this.title = title;
```

```
    this.price = price;
    this.description = description;
    this.imageUrl = imageUrl;
    /**for example with a ternary expression,
     * we can check if ID exists,
     * so if this returns true in an if statement
     * and if it's the case,
     * then i wanna create my ObjectId object
     * otherwise i will store null and null will be treated as false down there.
     *
     */
    this._id = id ? new mongodb.ObjectId(id) : null
  }

  save() {
    const db = getDb();
    let dbOp;
    if (this._id) {
      // Update the product
      dbOp = db
        .collection('products')
        .updateOne({ _id: this._id }, { $set: this });
    } else {
      dbOp = db.collection('products').insertOne(this);
    }
    return dbOp
      .then(result => {
        console.log(result);
      })
      .catch(err => {
        console.log(err);
      });
  }

  static fetchAll() {
    const db = getDb();
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products);
        return products;
      })
      .catch(err => {
        console.log(err);
      });
  }

  static findById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .find({ _id: new mongodb.ObjectId(prodId) })
      .next()
      .then(product => {
```

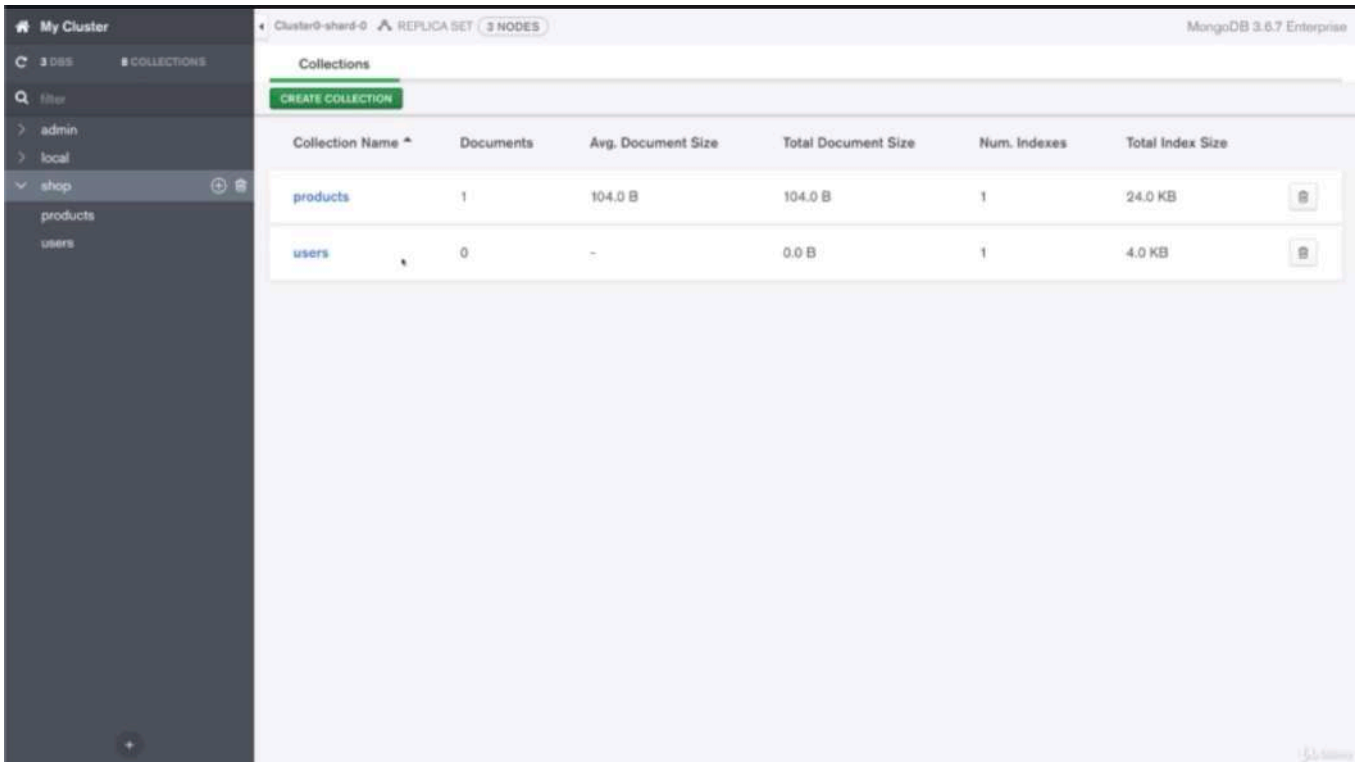
```
      console.log(product);
      return product;
    })
    .catch(err => {
      console.log(err);
    });
  }

  static deleteById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .deleteOne({ _id: new mongodb.ObjectId(prodId) })
      .then(result => {
        console.log('Deleted');
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = Product;
```

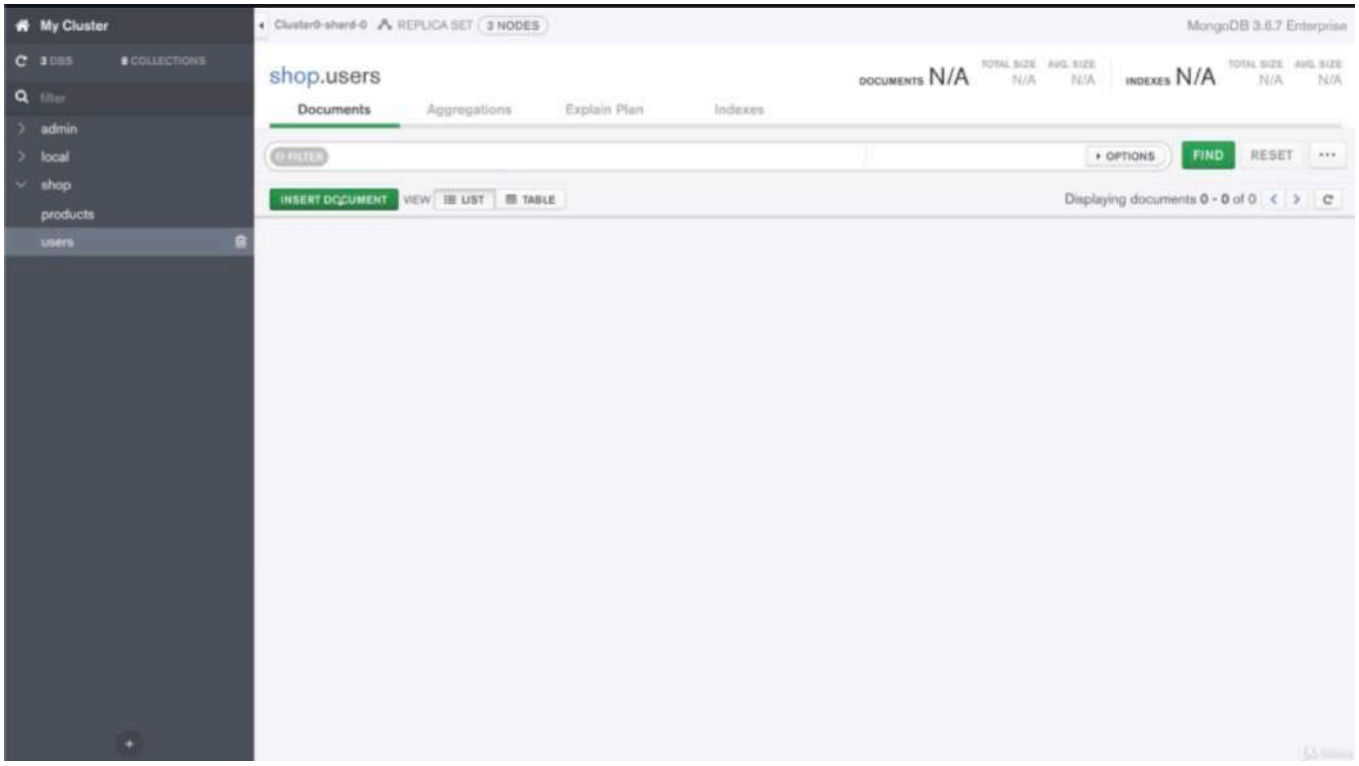
* Chapter 190: Creating New Users

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

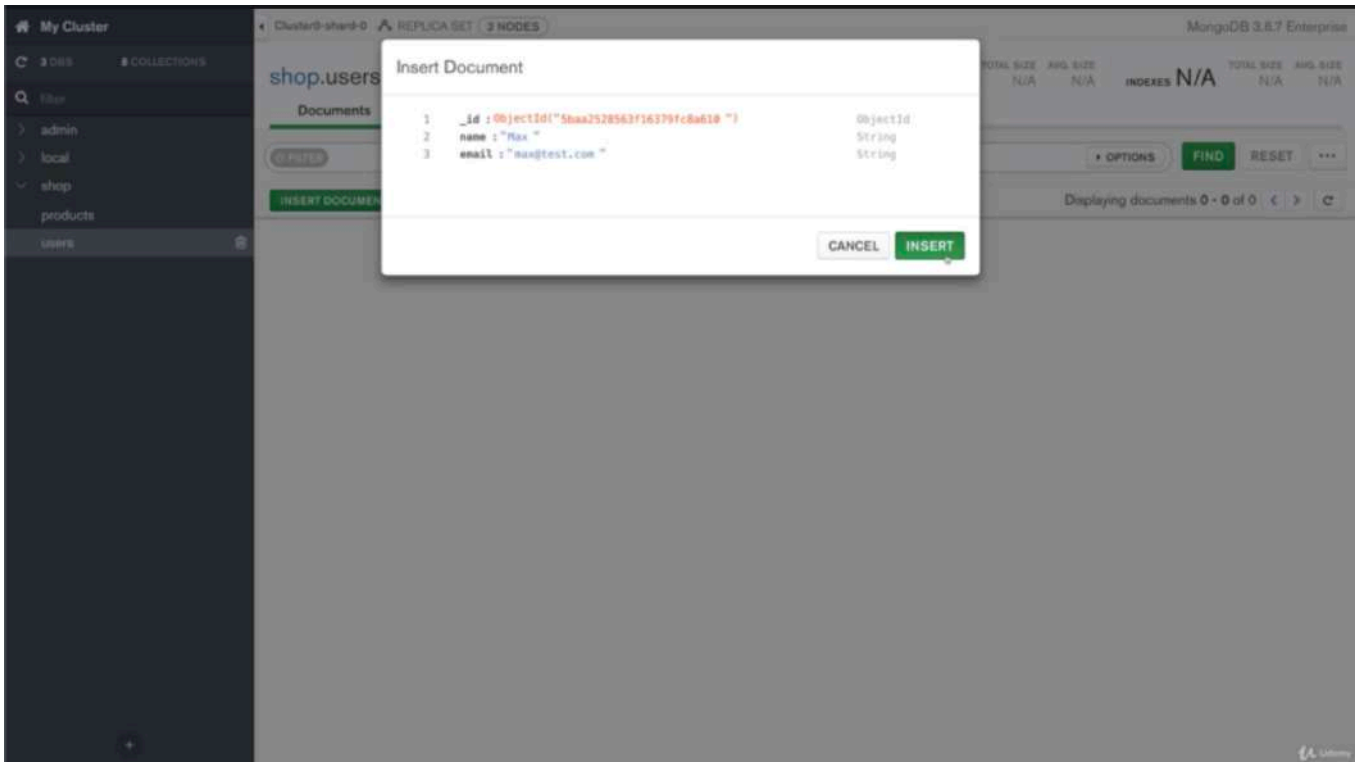


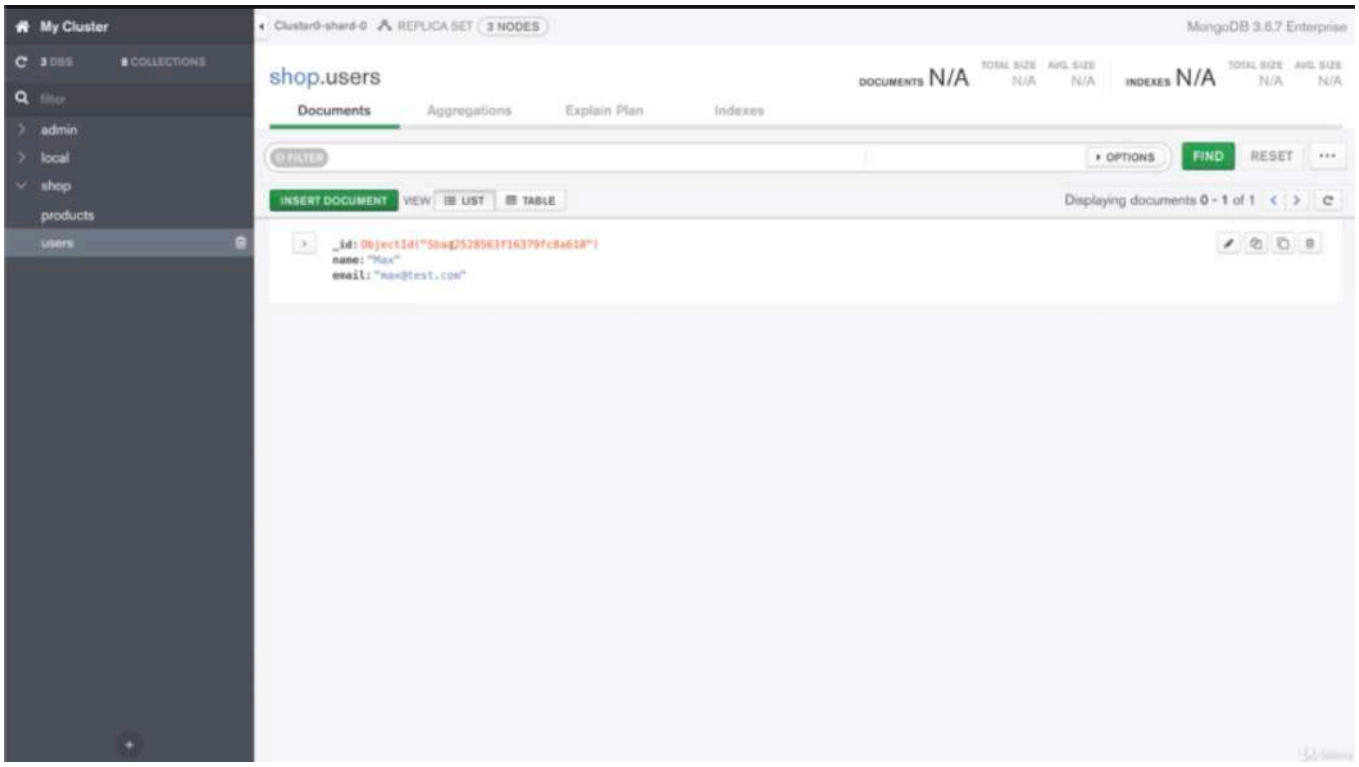
- i will connect to the shop database and i will create a new collection called ‘users’ and of course this collection name here should be the collection name you chose in your code.



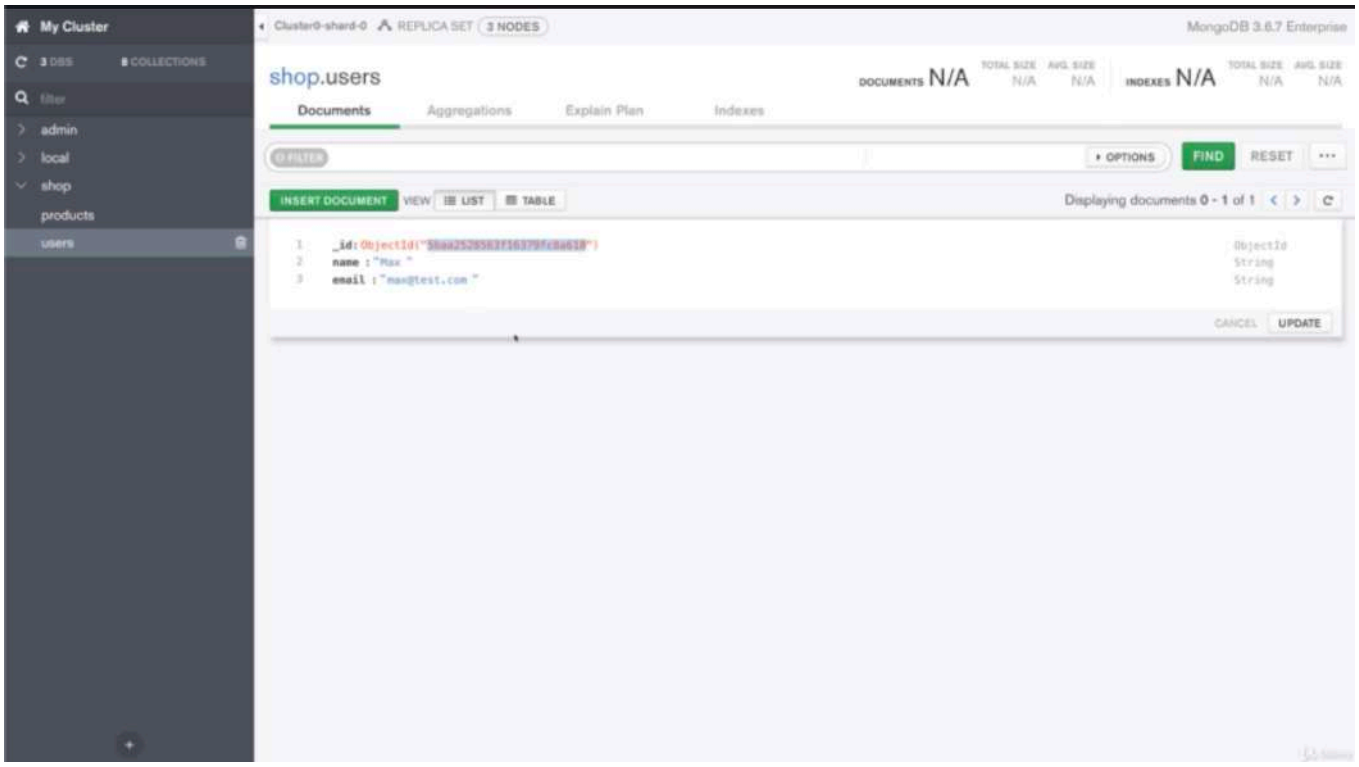


- after you create 'users' collection, then go into users collection.





- 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

```
//./models/user.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

/** i will follow a slightly different approach
 * which i already showed before
 * and i will create an ObjectId constant
 * and simply store access to it by accessing it up here.
 * but i'm not calling it, i'm not creating an object in here.
 * i'm storing the reference to the ObjectId class in my ObjectId constant
 * and then down there, i can just write 'new ObjectId'
 */
const ObjectId = mongodb.ObjectId

class User {
    constructor(username, email){
        this.name = username
        this.email = email
    }

    save(){
        const db = getDb()
        /**i wanna insert one new element
         * and that new element will be 'this'
         * so this javascript object we will in
         * an object with a name and an email property
         * this is what i wanna store as a user for now.
         */

        /**we can again use the 'then()' and 'catch()'
         * or we just return that and let whoever calls this listen to that.
```

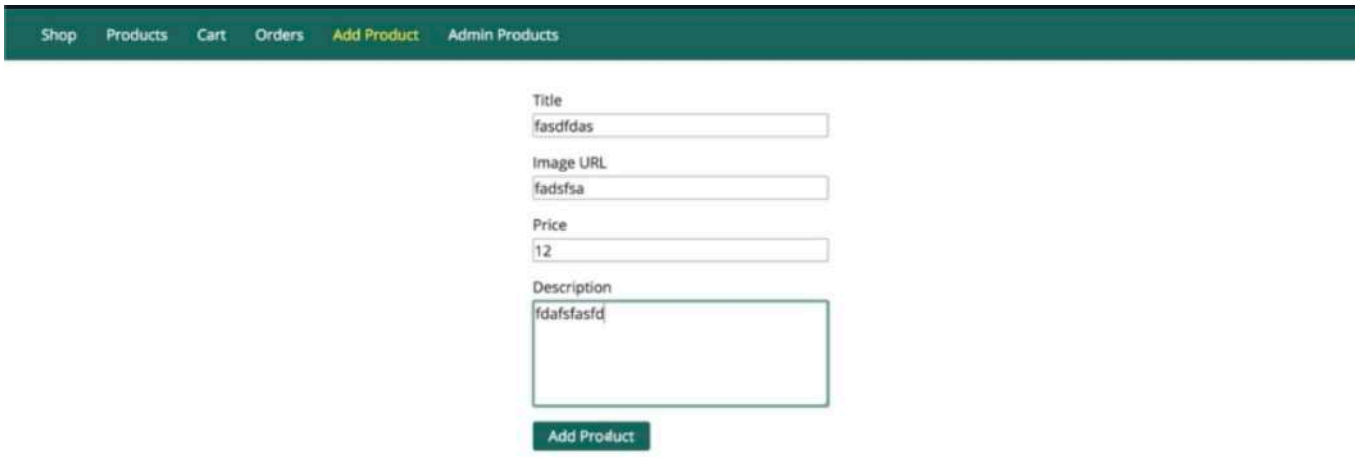
```
        */
        db.collection('users').insertOne(this)
    }

    static findById(userId){
        const db = getDb()
        /**i wanna use 'find()' to find a specific user
         * the important thing here is that you need to convert user ID which i expect to be a string to an ObjectId
         * so let me import 'mongodb' by requiring 'mongodb'
         *
         * thanks to my constant up there
         * and pass 'userId' to it.
         *
         * this should fine me all fitting users and i therefore get back a cursor.
         * and now i can call 'next()' to get the first and as we know only element that matters to us
         * so i'm returning this here
         */

         /**as a side note,
          * also use 'findOne()' if you sure that you only find one element
          * this will now not give you a cursor
          * but immediately return that one element.
          * then this would be an alternative and i will use that here.
          */
        return db
            .collection('users')
            .find({_id: new ObjectId(userId)})
            .next()
    }
}


module.exports = User
```

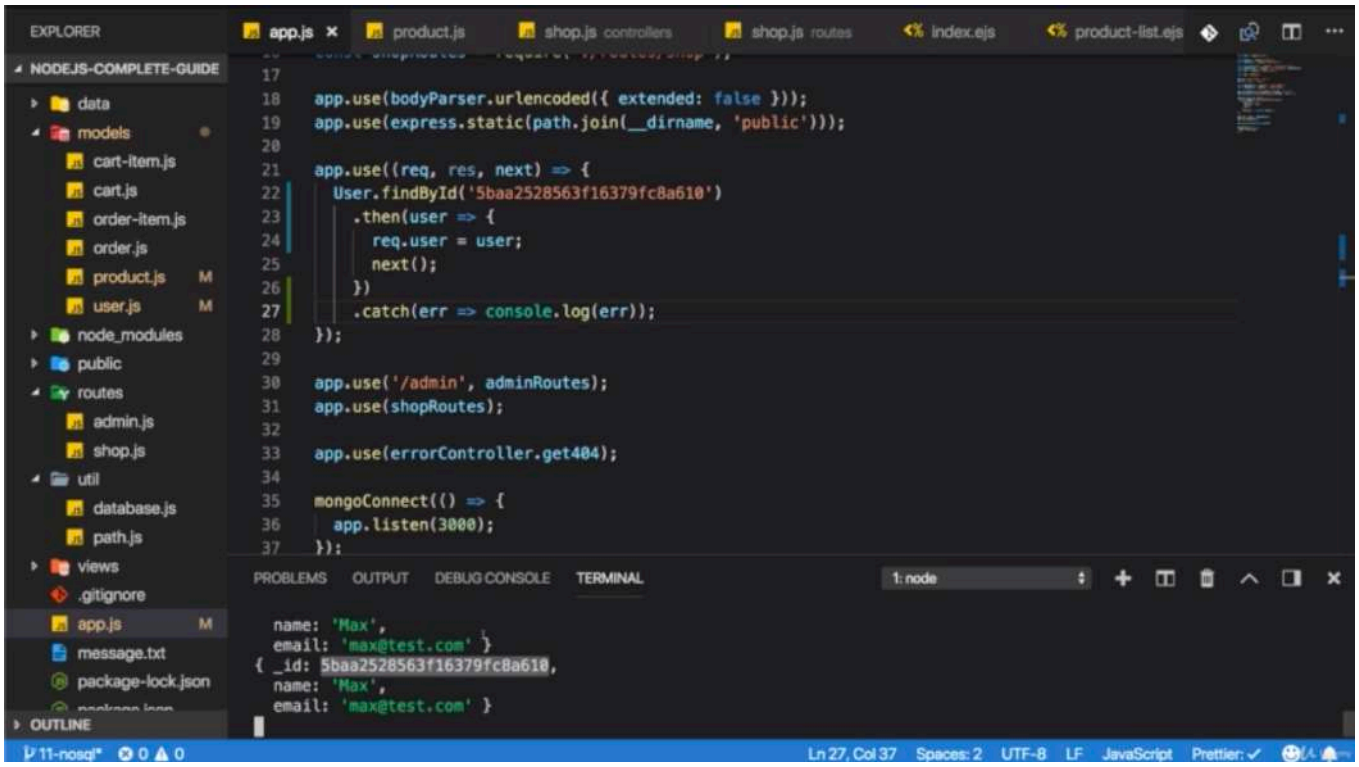
```
//app.js

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database').mongoConnect;
const User = require('./models/user')

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));
```

```

app.use((req, res, next) => {
  /** here in my middleware,
   * where i find a user by ID
   * i can search for that ID
   * and i convert that in the user model.
   * that is why i can use a string here.
   */
  User.findById('5cb7d12855fbe74b129c0b7c')
    .then(user => {
      req.user = user;
      next();
    })
    .catch(err => console.log(err));
  next();
});

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect(() => {
  app.listen(3000);
});

```

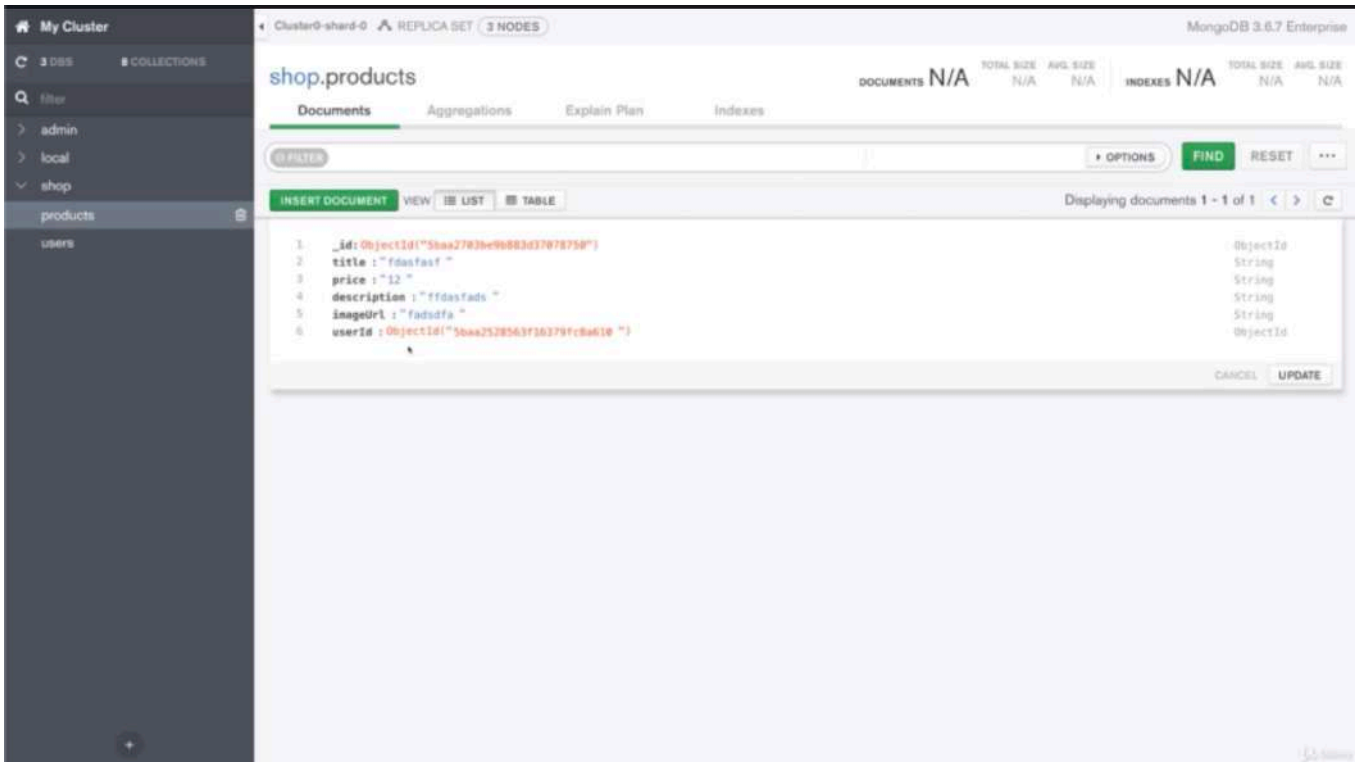
* Chapter 191: Storing The User In Our Database

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





- 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 leat, and for the product you wanna store the title and the price maybe.

```
//./models/product.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

/**i wanna use that user object when creating a new product
 * when saving a product, i wanna store a reference to a user
 * or embed the entire user data
 * however for products in users, you could in arguments for both approaches here
 * you don't wanna enclose all the user data in an embedded document
 * because that would means that if the user data changes, you need to change that data in all products
 * if you do include something which is unlikely to change very often like the username for example,
 * then you could go ahead and embed that together with the ID
 * so that you always have that ID to fetch more data about the user
 * you have got to 'findById()' in the user model
 * or that you have at least some snapshot data like the username available immediately
 * if that should change, you need to update it everywhere.
 *
 * The alternative to this is that you store the ID
 * so just a reference and therefore if you need connected data,
 * you always have to fetch it manually from 2 collections
 * but on the other hand, you might not do that too often
 * and therefore here when i fetch the product,
 * i don't really need the user data,
 * we are not displaying the user name anywhere in our app
 * i wanna store the user id so that we know who is connected even though we are not fetching that a lot
```

```
 *
 * what does this mean for our application here though?
 * it means when creating a new product,
 * we can pass the user id, we can accept the user
 */
class Product {
  constructor(title, price, description, imageUrl, id, userId) {
    this.title = title;
    this.price = price;
    this.description = description;
    this.imageUrl = imageUrl;
    this._id = id ? new mongodb.ObjectId(id) : null
    this.userId = userId
  }

  save() {
    const db = getDb();
    let dbOp;
    if (this._id) {
      // Update the product
      dbOp = db
        .collection('products')
        .updateOne({ _id: this._id }, { $set: this });
    } else {
      dbOp = db.collection('products').insertOne(this);
    }
    return dbOp
      .then(result => {
        console.log(result);
      })
      .catch(err => {
        console.log(err);
      });
  }

  static fetchAll() {
    const db = getDb();
    return db
      .collection('products')
      .find()
      .toArray()
      .then(products => {
        console.log(products);
        return products;
      })
      .catch(err => {
        console.log(err);
      });
  }

  static findById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .find({ _id: new mongodb.ObjectId(prodId) })
      .next()
```

```
    .then(product => {
      console.log(product);
      return product;
    })
    .catch(err => {
      console.log(err);
    });
  }

  static deleteById(prodId) {
    const db = getDb();
    return db
      .collection('products')
      .deleteOne({ _id: new mongodb.ObjectId(prodId) })
      .then(result => {
        console.log('Deleted');
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = Product;
```

```
// ./controllers/admin.js

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  /**when adding new products,
   * i wanna pass 'null' for the product Id
   * because we don't have that when creating a new product
   * but for the user id,
   * i wanna pass the ID of that user which we now store in our request
   *
   * 'req.user._id' will be just string
   * because when retrieving data, the object id is converted to a string.
   * so we have just the string here.
   */
  const product = new Product(
    title,
    price,
    description,
    imageUrl,
```

```
    null,
    req.user._id);
  product
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
    // Product.findById(prodId)
    .then(product => {
      if (!product) {
        return res.redirect('/');
      }
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product
      });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;

  const product = new Product(
    updatedTitle,
    updatedPrice,
    updatedDesc,
    updatedImageUrl,
    prodId
  );
  product
    .save()
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
```

```
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.deleteById(prodId)
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
```

```
//./models/user.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

/** i will follow a slightly different approach
 * which i already showed before
 * and i will create an ObjectId constant
 * and simply store access to it by accessing it up here.
 * but i'm not calling it, i'm not creating an object in here.
 * i'm storing the reference to the ObjectId class in my ObjectId constant
 * and then down there, i can just write 'new ObjectId'
 */
const ObjectId = mongodb.ObjectId

class User {
    constructor(username, email){
        this.name = username
        this.email = email
    }

    save(){
        const db = getDb()
        /**i wanna insert one new element
         * and that new element will be 'this'
         * so this javascript object we will in
         * an object with a name and an email property
         * this is what i wanna store as a user for now.
         */

        /**we can again use the 'then()' and 'catch()'
         * or we just return that and let whoever calls this listen to that.
```

```
        */
        db.collection('users').insertOne(this)
    }

    static findById(userId){
        const db = getDb()
        /**i wanna use 'find()' to find a specific user
         * the important thing here is that you need to convert user ID which i expect to be a string to an ObjectId
         * so let me import 'mongodb' by requiring 'mongodb'
         *
         * thanks to my constant up there
         * and pass 'userId' to it.
         *
         * this should fine me all fitting users and i therefore get back a cursor.
         * and now i can call 'next()' to get the first and as we know only element that matters to us
         * so i'm returning this here
         */

         /**as a side note,
          * also use 'findOne()' if you sure that you only find one element
          * this will now not give you a cursor
          * but immediately return that one element.
          * then this would be an alternative and i will use that here.
          */
        return db
            .collection('users')
            .findOne({_id: new ObjectId(userId)})
            .then(user => {
                console.log(user)
                return user
            })
            .catch(err =>
                console.log(err)
            )
    }
}


module.exports = User
```

```
//app.js

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database').mongoConnect;
const User = require('./models/user')

const app = express();

app.set('view engine', 'ejs');
app.set('views', 'views');
```

```

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use((req, res, next) => {
  /** here in my middleware,
   * where i find a user by ID
   * i can search for that ID
   * and i convert that in the user model.
   * that is why i can use a string here.
   */
  User.findById('5cb7d12855fbe74b129c0b7c')
    .then(user => {
      req.user = user;
      next();
    })
    .catch(err => console.log(err));
});

app.use('/admin', adminRoutes);
app.use(shopRoutes);

app.use(errorController.get404);

mongoConnect(() => {
  app.listen(3000);
});

```

* Chapter 192: Working On Cart Items & Orders

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

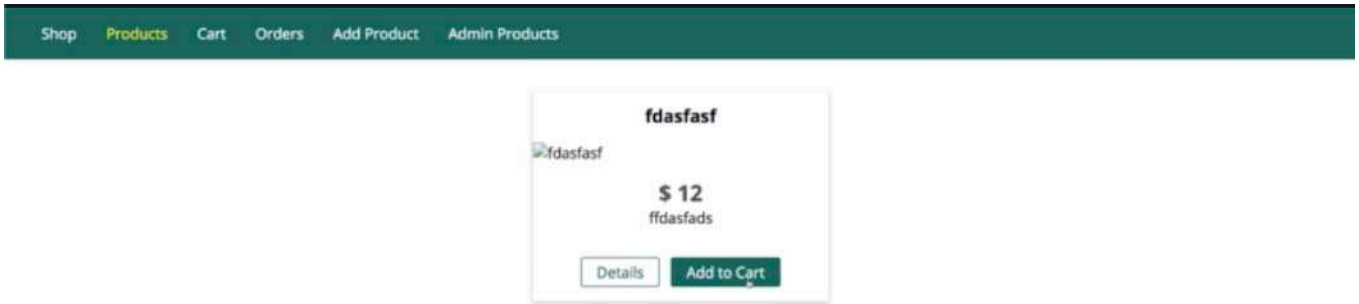
```
//./models/user.js

const mongodb = require('mongodb')
const getDb = require('../util/database').getDb

const ObjectId = mongodb.ObjectId

class User {
    constructor(username, email, cart, id){
        this.name = username
        this.email = email
        this.cart = cart //{items: []}
        this._id = id
    }

    save(){
        const db = getDb()
        db.collection('users').insertOne(this)
```

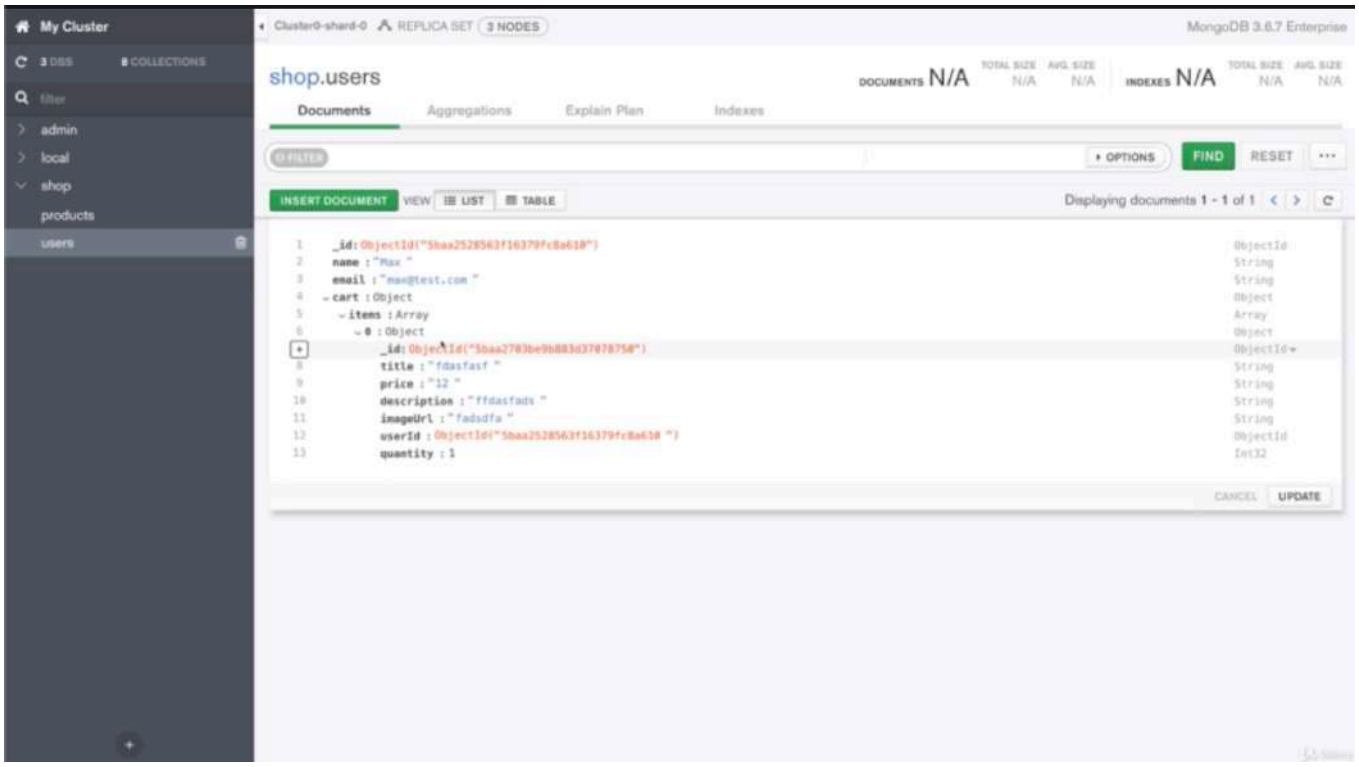
```
    }

    addToCart(product){
        /**you must not forget that 'addToCart' will be called on a user object
         * and we will create that obejct with data we fetched from the database with help of 'findById()'
         *
         * the cart will essentially be an object which has the items array
         *
         * i will pass a function to find index which is a function
         * that javascript will execute for every element in the items array
         * and here i wanna return true if i found the right product in my items array
         * and this will be the case if 'cp' which is the product in the items array
         *if 'cp._id' matches the _id of the product i'm trying to insert,
         */
        /*
        const cartProduct = this.cart.item.findIndex(cp => {
            return cp._id === prodId
        })
        */

        /**you create a object with curly braces
         * because we will add an object here
         * and then you use the javascript spread operator
         * 3 dots '...' to pull out all properties of this object
         * so of the product object
         * and then with a comma, you can add or overwrite a property
         * and here i will add the quantity property and set it to 1
         */
        const updatedCart = {items: [{...product, quantity: 1}]}
        const db = getDb()
        db
            .collection('users')
            .updateOne(
                {_id: new ObjectId(this._id)},
                /**i will describe how to update and i will use '$set'
                 * where i pass an object which holds all the information about which field to update in which way.
                 *
                 * here i wanna keep everything as it is,
                 * i don't wanna change the user name or anything like that.
                 * so 'cart' which i expect to have in a user in the database will now receive 'updatedCart'
                 * so 'updatedCart' object as a new value which will overwrite the old one
                 * and this is important. it will not merge this with the old one,
                 * it will not merge the elements in teh items array,
                 * it will overwrite the old cart with the new cart.
                 */
                {$set: {cart: updateCart}}
            )
    }

    static findById(userId){
        const db = getDb()
        return db
            .collection('users')
```

```
            .findOne({_id: new ObjectId(userId)})
            .then(user => {
                console.log(user)
                return user
            })
            .catch(err =>
                console.log(err)
            )
    }
}


module.exports = User
```

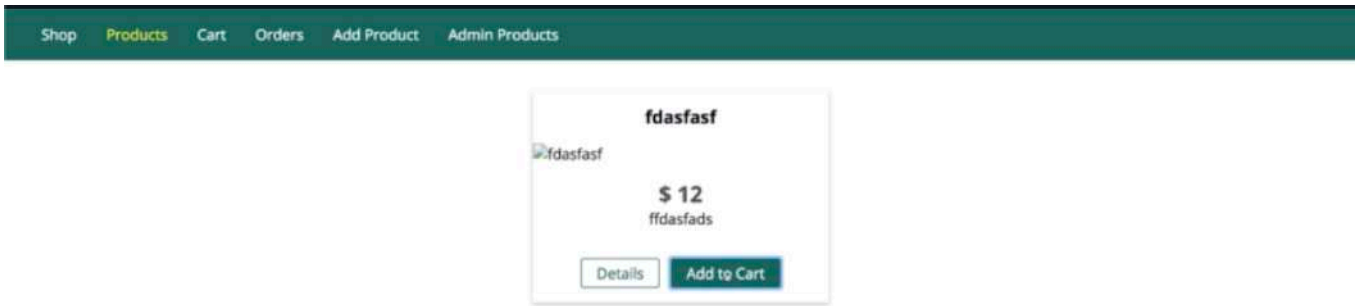
* Chapter 193: Adding The “Add To Cart” Functionality

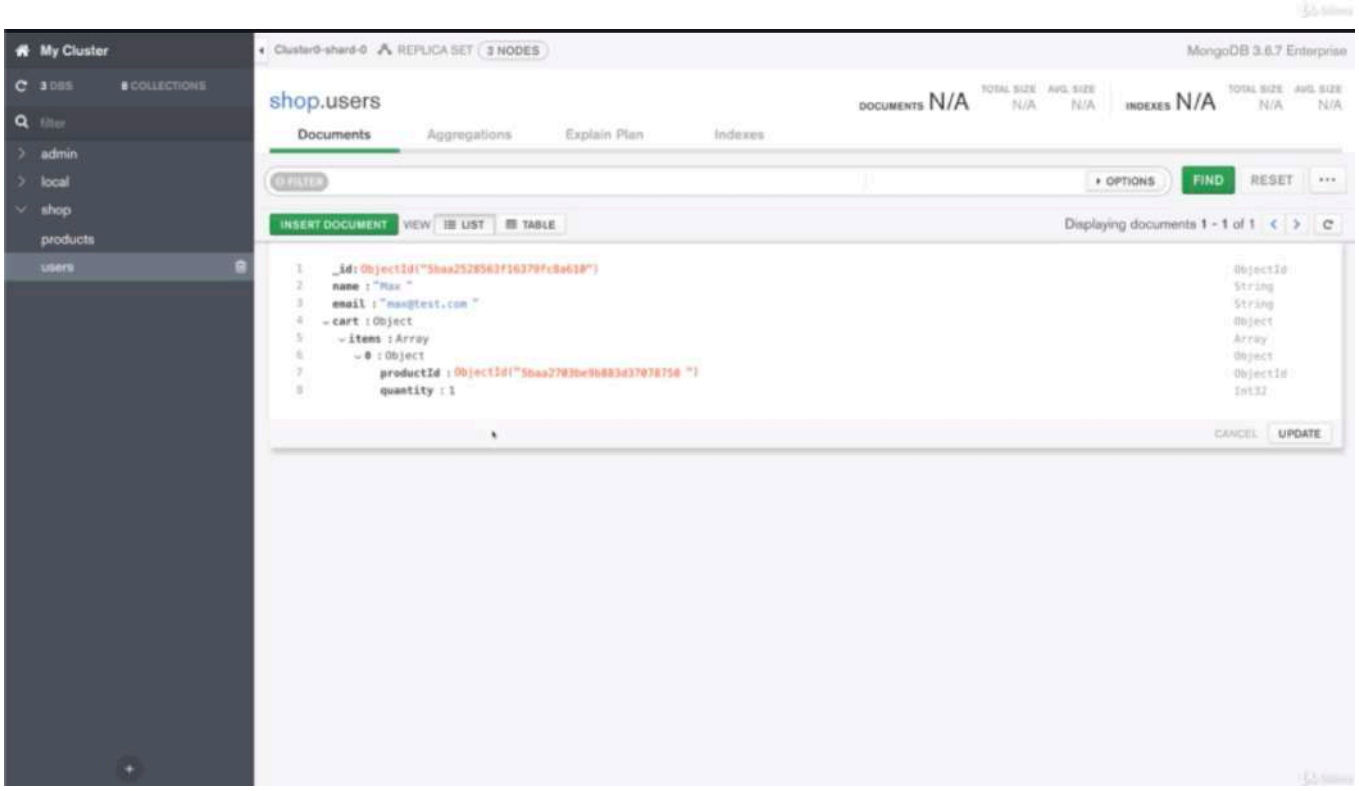
1. update
- app.js
- ./controllers/shop.js
- ./routes/shop.js
- ./views/includes/add-to-cart.ejs
- ./models/user.js





- 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 were already are 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 thisn 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 becasue 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 becasue 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 imformation i want.

```
//app.js

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const errorController = require('./controllers/error');
const mongoConnect = require('./util/database').mongoConnect;
const User = require('./models/user')

```

```
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('5cb7d12855fbe74b129c0b7c')
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 obejct 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.namem, 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 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
```

```
        this.cart = cart //{items: []}
        this._id = id
    }

    save(){
        const db = getDb()
        db.collection('users').insertOne(this)
    }

    addToCart(product){
        /**i don't wanna store all product data in this object and a quantity
         * i wanna store the product ID by creating a new ObjectId
         * and passing product _Id as an argument and the quantity
         * now only the reference and the quantity and not the rest of the data
         */
        const updatedCart = {
            items: [
                    {
                        productId: new ObjectId(product._id),
                        quantity: 1
                    }
                ]}
        const db = getDb()
        db
            .collection('users')
            .updateOne(
                {_id: new ObjectId(this._id)},
                {$set: {cart: updatedCart}}
            )
    }

    static findById(userId){
        const db = getDb()
        return db
            .collection('users')
            .findOne({_id: new ObjectId(userId)})
            .then(user => {
                console.log(user)
                return user
            })
            .catch(err =>
                console.log(err)
            )
    }
}


module.exports = User
```

```
// ./routes/shop.js

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

```

```
const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);
/*
router.get('/cart', shopController.getCart);
*/

router.post('/cart', shopController.postCart);

/*
router.post('/cart-delete-item', shopController.postCartDeleteProduct);

router.post('/create-order', shopController.postOrder)

router.get('/orders', shopController.getOrders);
*/

module.exports = router;

```

```
<!--./views/includes/add-to-cart.ejs-->

<form action="/cart" method="post">
    <button class="btn" type="submit">Add to Cart</button>
    <input type="hidden" name="productId" value="<%= product._id %>">
</form>

```

```
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
  // Product.findAll({ where: { id: prodId } })
  //   .then(products => {
  //     res.render('shop/product-detail', {
  //       product: products[0],
  //       pageTitle: products[0].title,
```

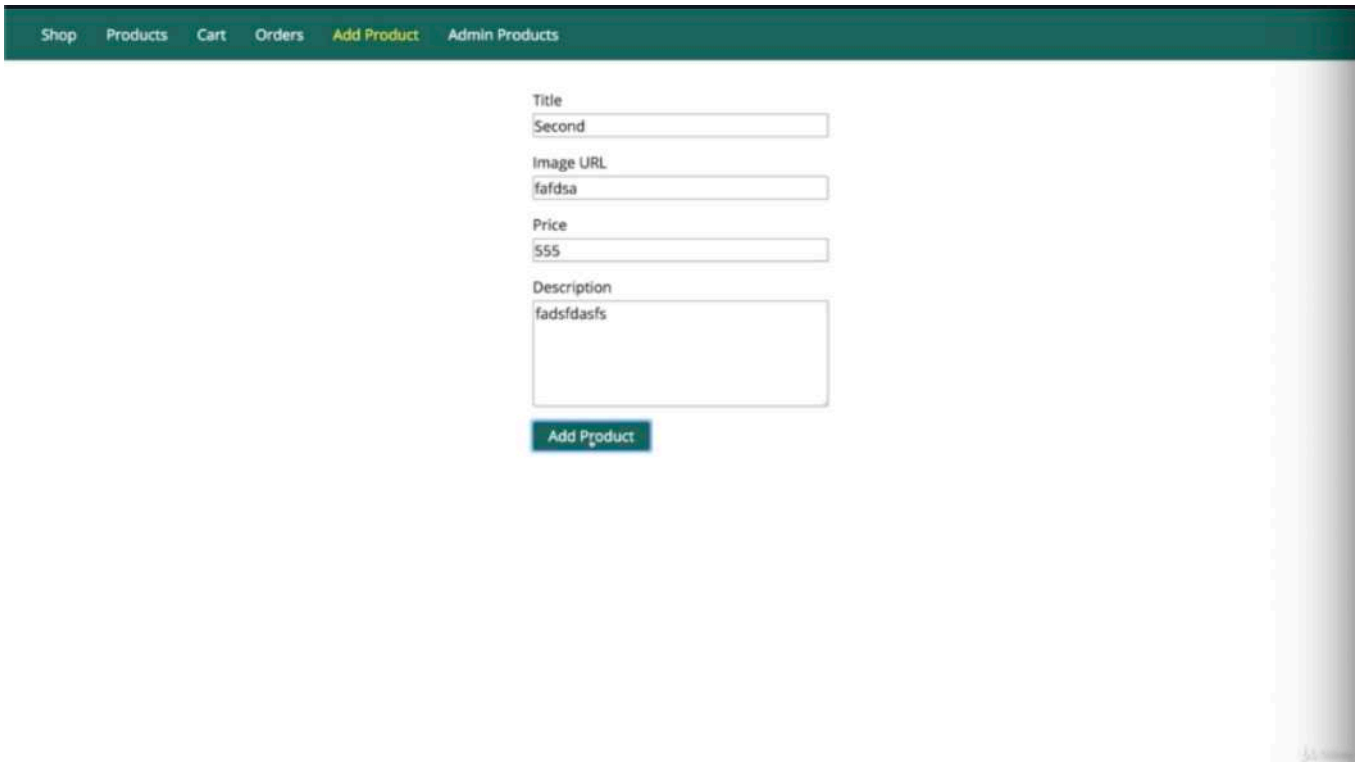
```
26  //       path: '/products'
27  //     });
28  //   })
29  //   .catch(err => console.log(err));
30  Product.findById(prodId)
31    .then(product => {
32      res.render('shop/product-detail', {
33        product: product,
34        pageTitle: product.title,
35        path: '/products'
36      });
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.
```

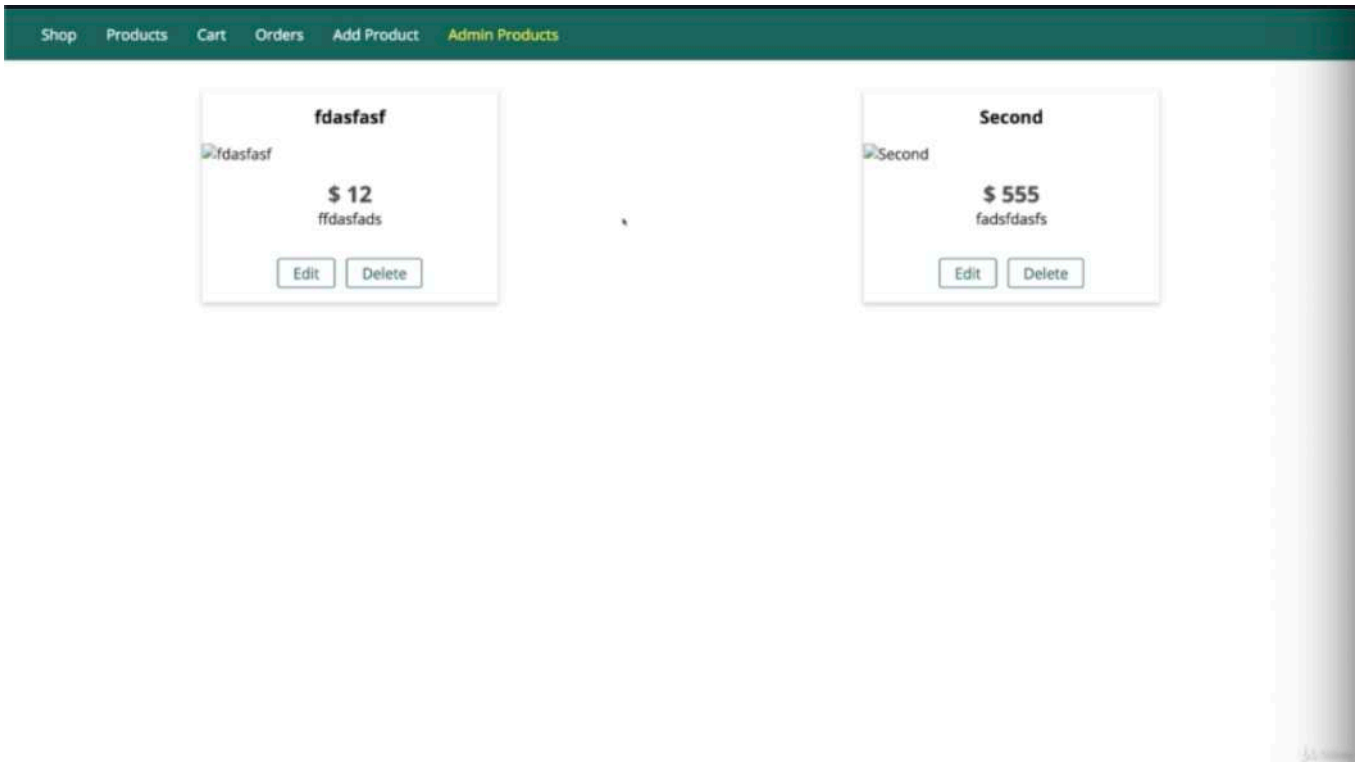
```
     *
     */
    return req.user.addToCart(product)
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
  console.log(err)
})
/*
let fetchedCart;
let newQuantity = 1;
req.user
  .getCart()
  .then(cart => {
    fetchedCart = cart;
    return cart.getProducts({ where: { id: prodId } });
  })
  .then(products => {
    let product;
    if (products.length > 0) {
      product = products[0];
    }

    if (product) {
      const oldQuantity = product.cartItem.quantity;
      newQuantity = oldQuantity + 1;
      return product;
    }
    return Product.findByPk(prodId);
  })
  .then(product => {
    return fetchedCart.addProduct(product, {
      through: { quantity: newQuantity }
    });
  })
  .then(() => {
    res.redirect('/cart');
  })
  .catch(err => console.log(err));
*/
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .getCart()
    .then(cart => {
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      const product = products[0];
      return product.cartItem.destroy();
    })
    .then(result => {
```

```
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts();
    })
    .then(products => {
      return req.user
        .createOrder()
        .then(order => {
          return order.addProducts(
            products.map(product => {
              product.orderItem = { quantity: product.cartItem.quantity };
              return product;
            })
          );
        })
        .catch(err => console.log(err));
    })
    .then(result => {
      return fetchedCart.setProducts(null);
    })
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({include: ['products']})
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

* 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
fadsfdasfs
Add Product


Shop
Products
Cart
Orders
Add Product
Admin Products
fdasfasf
fdasfasf
$ 12
ffdasfads
Edit
Delete
Second
Second
$ 555
fadsfdasfs
Edit
Delete

Shop Products Cart Orders Add Product Admin Products

fdasfasf

fdasfasf

$ 12

ffdasfads

Details | Add to Cart

Second

Second

$ 555

fadsfdasfs

Details | Add to Cart

My Cluster

3 DBS 8 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

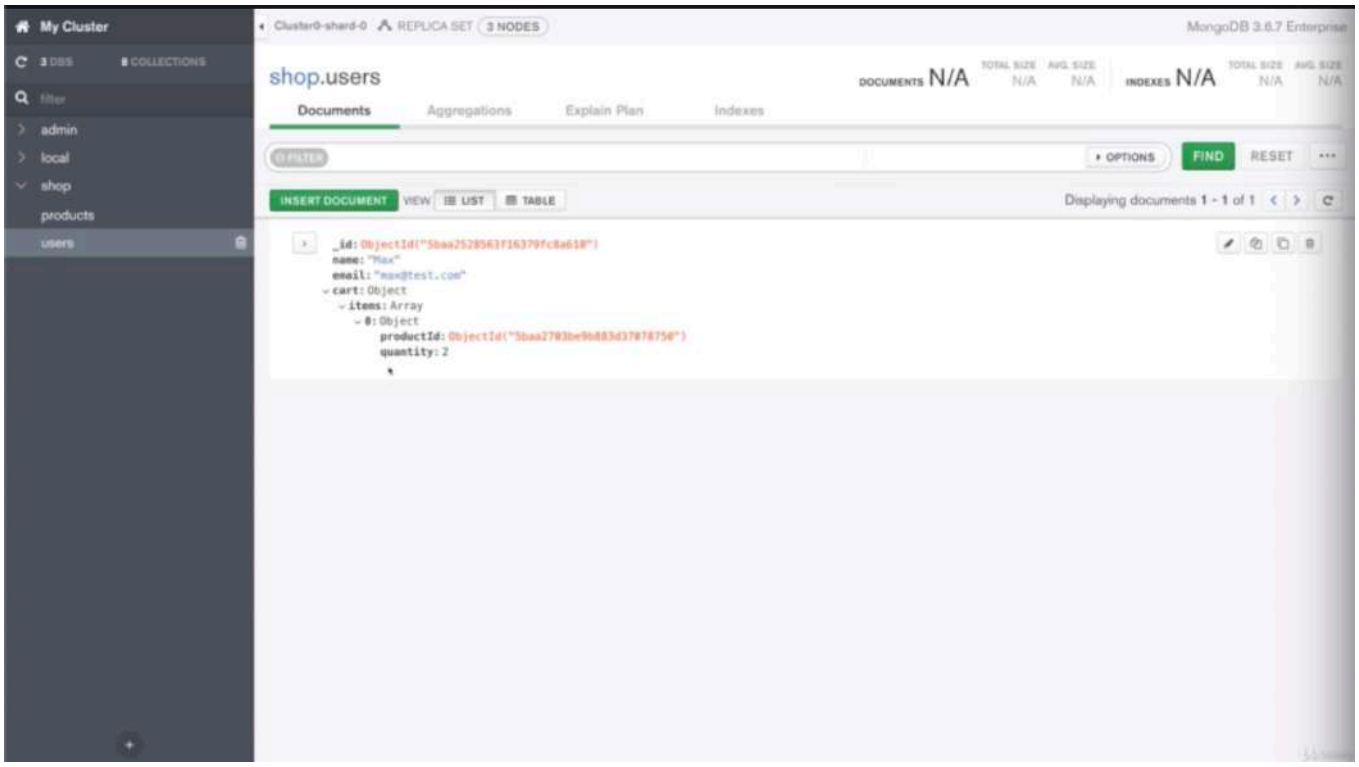
FILTER OPTIONS FIND RESET

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

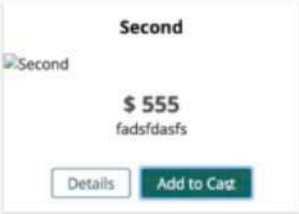
```
1  _id: ObjectId("5baa2528563f16379fc8a610")          ObjectId
2  name : "Max "                                      String
3  email : "max@test.com "                            String
4  cart : Object                                      Object
5    items : Array                                    Array
6      0 : Object                                     Object
7        productId : ObjectId("5baa2703be9b883d37078750 ")   ObjectId
8        quantity : 1                                 Int32
9      1 : Object                                     Object
10       productId : ObjectId("5baa2703be9b883d37078750 ")   ObjectId
11       quantity : 1                                 Int32
```

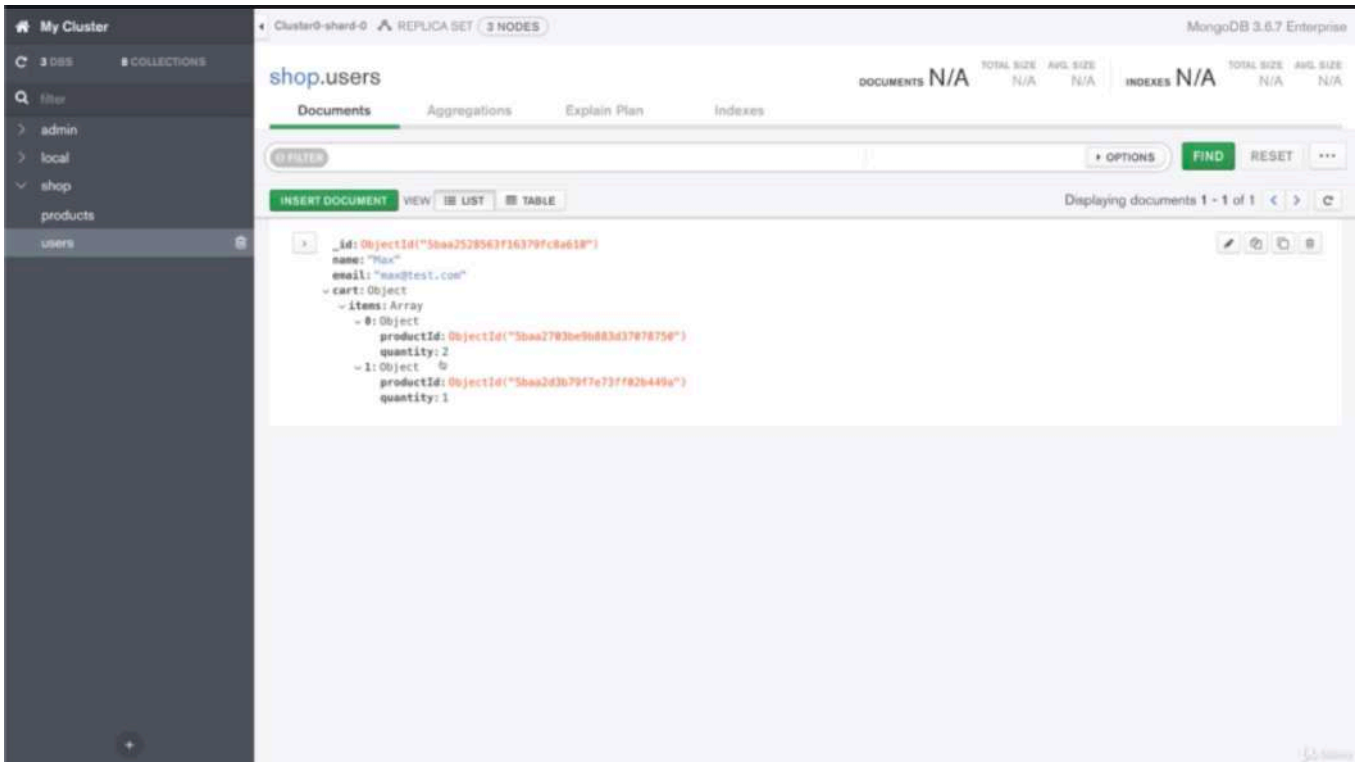
CANCEL UPDATE

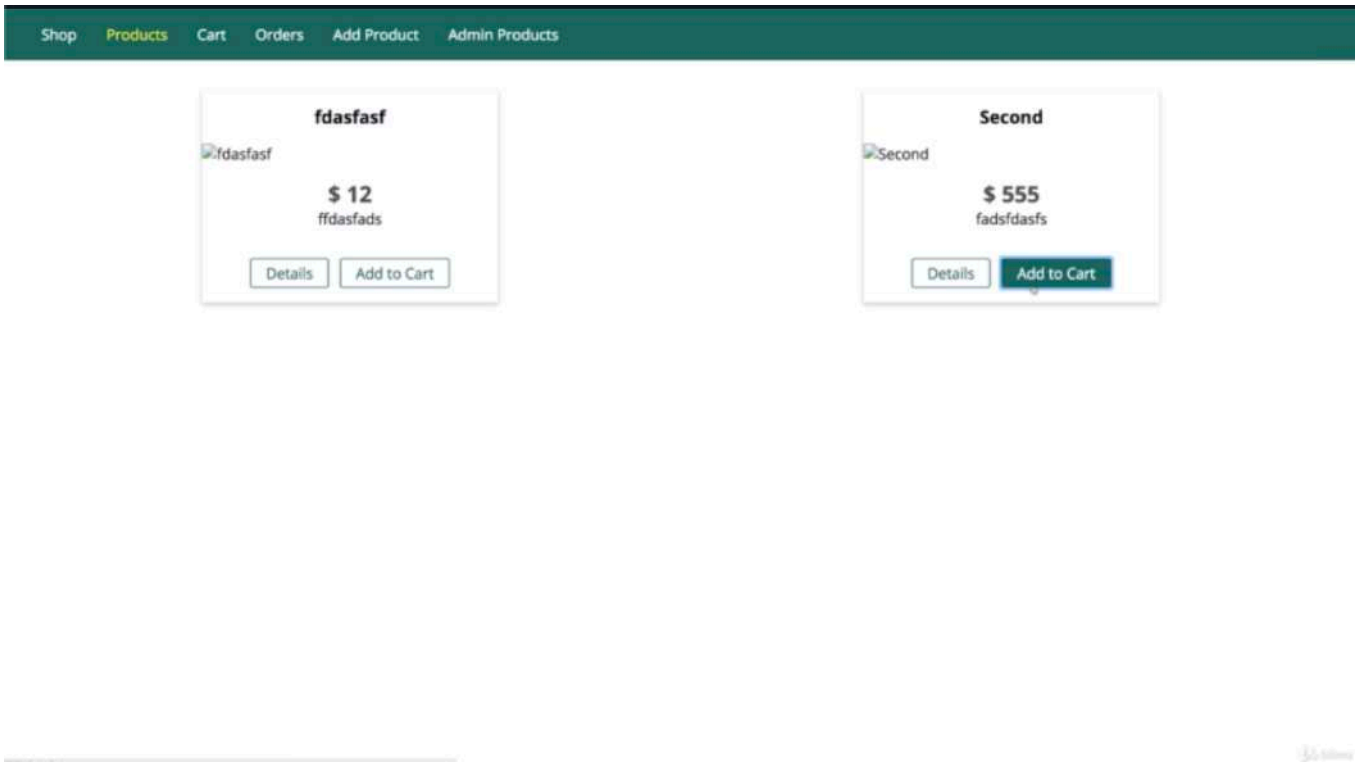

My Cluster
3 DBS
8 COLLECTIONS
admin
local
shop
products
users
Cluster0-shard-0
REPLICA SET
3 NODES
MongoDB 3.6.7 Enterprise
shop.users
DOCUMENTS N/A
INDEXES N/A
Documents
Aggregations
Explain Plan
Indexes
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 1 of 1
_id: ObjectId("5baa2528563f16379fc8a618")
name: "Max"
email: "max@test.com"
cart: Object
items: Array
0: Object
productId: ObjectId("5baa2703be9b883d37074750")
quantity: 2

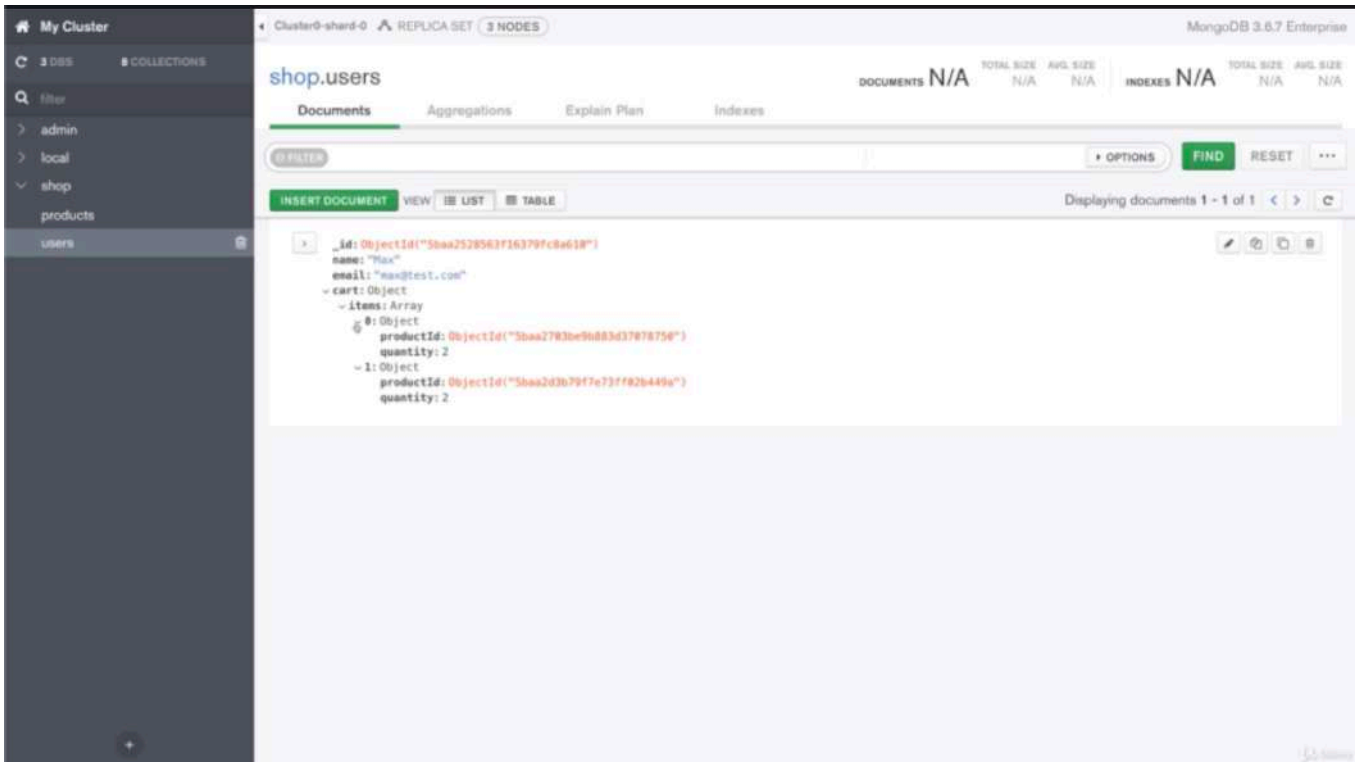

Shop
Products
Cart
Orders
Add Product
Admin Products


fdasfasf
fdasfasf
$ 12
ffdasfads
Details
Add to Cart


Second
Second
$ 555
fadsfdasfs
Details
Add to Cart


My Cluster
3 DBS
8 COLLECTIONS
admin
local
shop
products
users
Cluster0-shard-0
REPLICA SET
3 NODES
MongoDB 3.6.7 Enterprise
shop.users
DOCUMENTS N/A
INDEXES N/A
Documents
Aggregations
Explain Plan
Indexes
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 1 of 1
_id: ObjectId("5baa2528563f16379fc8a618")
name: "Max"
email: "max@test.com"
cart: Object
items: Array
0: Object
productId: ObjectId("5baa2703be9b883d37078750")
quantity: 2
1: Object
productId: ObjectId("5baa2d3b79f7e73ff82b449a")
quantity: 1


Shop
Products
Cart
Orders
Add Product
Admin Products
fdasfasf
fdasfasf
$ 12
ffdasfads
Details
Add to Cart
Second
Second
$ 555
fadsfdasfs
Details
Add to Cart



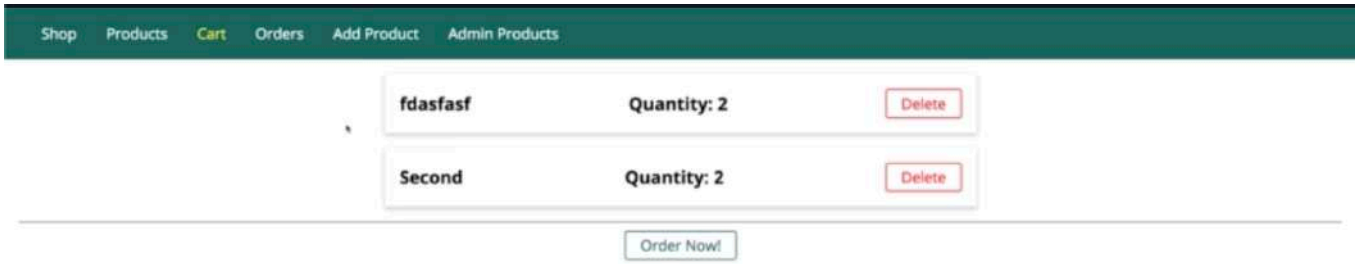

```javascript
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }

  save() {
    const db = getDb();
    return db.collection('users').insertOne(this);
  }

  addToCart(product) {
    const cartProductIndex = this.cart.items.findIndex(cp => {
      /**'product' in 'addToCart(product)' is a 'product' i just retrieved from the database
       * '_id' in here is treated as a string in javascript
       * but is not exactly of type string
       * since i'm using 3 equal signs'===' in my check here
       * however i'm telling javascript that this should only return true
       * if these 2 don't only match by value but also by type
       * and technically '_id' is not string even though we can use it as such.
       *
       * so one solution is to use 2 equal signs or to use 'toString()' on both here.
       * make sure we only works with strings here in both cases.
       */
      return cp.productId.toString() === product._id.toString();
    });
```

```
    let newQuantity = 1;
    /**this gives me a new array with all the items that are in the cart
     * and they are now stored here.
     *
     * now i can edit my updatedCartItems
     * and now i need to differentiate,
     * do we already have that item in cart or not
     * so 'updatedCartItems' position is here, before my if check
     */
    const updatedCartItems = [...this.cart.items];
    /**if i make it into this if statement,
     * i know that we have this product already.
     * i can access updatedCartItems for the cartProductIndex i found
     * now i have access to that item i'm interested in.
     * i know it already existed
     * so i can set its quantity equal to the 'newQuantity'
     */
    if (cartProductIndex >= 0) {
      newQuantity = this.cart.items[cartProductIndex].quantity + 1;
    /**and i can edit this array without touching the old array
     * due to the way javascript works with reference and primitive types
     */
      updatedCartItems[cartProductIndex].quantity = newQuantity;
    } else {
    /**if the item didn't exist before,
     * i will take my updatedCartItems and add a new one with 'push()'
     * i will add a new cartItem and i will add a new cartItem
     * which looks exactly as described down there. '{ productId: new ObjectId(product._id), quantity: newQuantity }'
     * so i will grab that code and add it here.
     */
      updatedCartItems.push({
        productId: new ObjectId(product._id),
        quantity: newQuantity
      });
    }
    const updatedCart = {
      /**i can always set my items equal to the updatedCartItems
       * because that will always be an array with all the old elements '[...this.cart.items]'
       * because i copy '[...this.cart.items]' first and then with the update added
       * so either with the quantity increated for the existing element
       * or with a new element added to the cart.
       *
       * so i can safely have my updatedCart down there and save that to the database with all the updatedItems in there.
       */
      items: updatedCartItems
    };
    const db = getDb();
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        { $set: { cart: updatedCart } }
      );
```

```
  }

  static findById(userId) {
    const db = getDb();
    return db
      .collection('users')
      .findOne({ _id: new ObjectId(userId) })
      .then(user => {
        console.log(user);
        return user;
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = User;
```

* Chapter 195: Displaying The Cart Items

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



Shop Products Cart Orders Add Product Admin Products

Title

Third!

Image URL

http://ichef.bbci.co.uk/wwfeatures/wm/live

Price

12

Description

fdasfdasf

Add Product

Shop Products Cart Orders Add Product Admin Products

fdasfasf

fdasfasf

$ 12

ffdasfads

Edit Delete

Second

Second

$ 555

fadsfdasfs

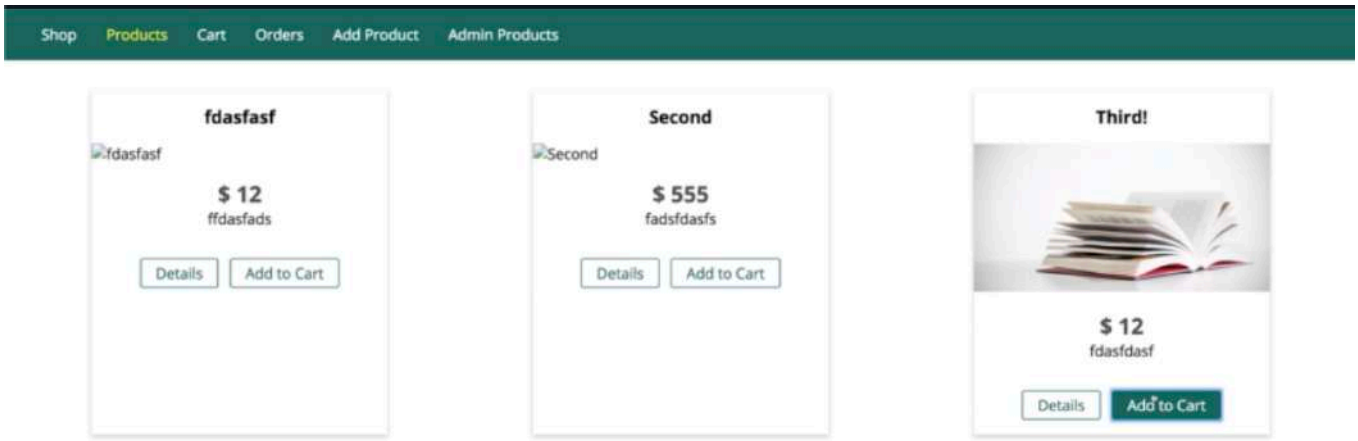
Edit Delete

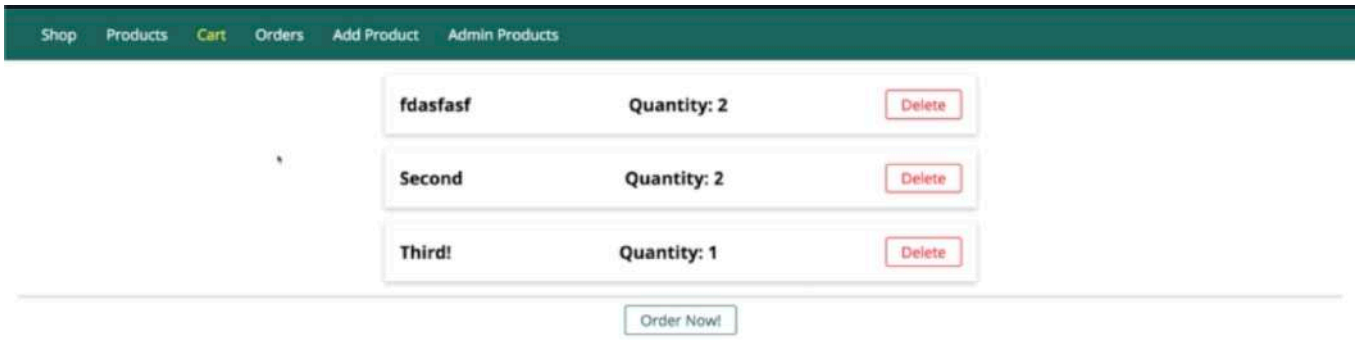
Third!

$ 12

fdasfdasf

Edit Delete






```
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }
```

```

  save() {
    const db = getDb();
    return db.collection('users').insertOne(this);
  }

  addToCart(product) {
    const cartProductIndex = this.cart.items.findIndex(cp => {
      return cp.productId.toString() === product._id.toString();
    });
    let newQuantity = 1;
    const updatedCartItems = [...this.cart.items];
    if (cartProductIndex >= 0) {
      newQuantity = this.cart.items[cartProductIndex].quantity + 1;
      updatedCartItems[cartProductIndex].quantity = newQuantity;
    } else {
      updatedCartItems.push({
        productId: new ObjectId(product._id),
        quantity: newQuantity
      });
    }
    const updatedCart = {
      items: updatedCartItems
    };
    const db = getDb();
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        { $set: { cart: updatedCart } }
      );
  }
/**'getCart()' should return products which are enriched with all the data that is stored in a product's collection
 * because in users, in the cart we will only store the reference
 * and this is what we need to do in MongoDB
 * if we then have a connection between 2 collections with a reference,
 * we need to merge them manually as we are doing it here
 * with that merging being done manually,
 * we can now use that data.
 * so 'getCart()' should return a cart with all the information we need.
 */

  getCart() {
    const db = getDb();
    /**object have a productID and quantity
     * but we are only interested in product ID
     * so 'map()' this, this is a default javascript function
     * i will map this to transform every item in there
     * and i wanna return the product ID
     *
     * so what i'm doing is that
     * i'm mapping an array of items where every item is a javascript object into an array of just string, of the product IDs
     * and this is then stored in this new 'productIds' constant
     */
```

```
    const productIds = this.cart.items.map(i => {
      return i.productId;
    });
    /**i reach out to the products collection
     * now because i have all the user data
     * i have all the cart data
     * now i need to fill it with some live from the products database
     *
     * in there, i wanna find all products that are in my cart
     * for this, we can use special query syntax MongoDB supports.
     * in 'find()', i can tell i wanna find all products where '_id' is equal to
     * and i don't pass an ID here because i'm not looking for a single ID
     * instead i pass an object becasue this allows me to use some special MongoDB query operator
     * we are looking for '$in' operator
     * and this operator takes an array of IDs
     * and therefore every ID which is in the array will be accepted
     * and will get back a cursor which holds references to all products with one of the IDs mentioned in this array
     *
     * then my array which i wanna use to tell MongoDB
     * give me all elements where the ID is one of the IDs mentioned in this array here.
     * this gives me a cursor with all the matching products
     *
     * and i will again use 'toArray()' to get quickly that converted to a javascript array
     * and then i will add 'then()' method
     * in 'then()' method, i will have all my product data for the products that were in my cart
     * and of course we wanna add the quantity back to every product
     * because that is something that is important to us
     */
    return db
      .collection('products')
      .find({ _id: { $in: productIds } })
      .toArray()
      .then(products => {
        return products.map(p => {
          /**i wanna keep all the data i retrieved
           * but then i will add a new quantity property
           * and that quantity property needs to be populated with data i have on that product
           * we have the products stored in the cart of this user
           * and make sure you use arrow function to ensure that 'this' inside of this functio still refers to the overall class
           * with normal functions, it would not
           * and access my items and find the item with that ID at hand here
           */
          return {
            ...p,
            quantity: this.cart.items.find(i => {
            /**in the end, i have an array of 'products'
             * fresh from the database
             * then i wanna transform this which i'm doing with 'map()'
             * 'map()' takes a function that executes on every element and products
             * which describes how to transform this element
             * and in 'p', i'm returning the new value which is an object
```

```
         * where i still have all the old product properties
         * but i add a new quantity property and to get the right quantity for that given product,
         * i reach out to 'cart.items' which exist on that user
         * and i again use a built in javascript method,
         * find to look at all elements in cart.items with this function in 'i'
         * 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
         * and since with 'map()' i'm going through all these products,
         * this will also vary for every run.
         *
         * now from the cart.items i have, i extract a quantity for that given product.
         */
          return i.productId.toString() === p._id.toString();
        }).quantity
      };
    });
  });
}

static findById(userId) {
  const db = getDb();
  return db
    .collection('users')
    .findOne({ _id: new ObjectId(userId) })
    .then(user => {
      console.log(user);
      return user;
    })
    .catch(err => {
      console.log(err);
    });
}
}

module.exports = User;
```

```
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
```

```
  // Product.findAll({ where: { id: prodId } })
  //   .then(products => {
  //     res.render('shop/product-detail', {
  //       product: products[0],
  //       pageTitle: products[0].title,
  //       path: '/products'
  //     });
  //   })
  //   .catch(err => console.log(err));
  Product.findById(prodId)
    .then(product => {
      res.render('shop/product-detail', {
        product: product,
        pageTitle: product.title,
        path: '/products'
      });
    })
    .catch(err => console.log(err));
};

exports.getIndex = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .getCart()
    .then(products => {
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
    .catch(err => console.log(err));
};

exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
```

```
    });
  // let fetchedCart;
  // let newQuantity = 1;
  // req.user
  //   .getCart()
  //   .then(cart => {
  //     fetchedCart = cart;
  //     return cart.getProducts({ where: { id: prodId } });
  //   })
  //   .then(products => {
  //     let product;
  //     if (products.length > 0) {
  //       product = products[0];
  //     }

  //     if (product) {
  //       const oldQuantity = product.cartItem.quantity;
  //       newQuantity = oldQuantity + 1;
  //       return product;
  //     }
  //     return Product.findById(prodId);
  //   })
  //   .then(product => {
  //     return fetchedCart.addProduct(product, {
  //       through: { quantity: newQuantity }
  //     });
  //   })
  //   .then(() => {
  //     res.redirect('/cart');
  //   })
  //   .catch(err => console.log(err));
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .getCart()
    .then(cart => {
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      const product = products[0];
      return product.cartItem.destroy();
    })
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
```

```
    return cart.getProducts();
  })
  .then(products => {
    return req.user
      .createOrder()
      .then(order => {
        return order.addProducts(
          products.map(product => {
            product.orderItem = { quantity: product.cartItem.quantity };
            return product;
          })
        );
      })
      .catch(err => console.log(err));
  })
  .then(result => {
    return fetchedCart.setProducts(null);
  })
  .then(result => {
    res.redirect('/orders');
  })
  .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({ include: ['products'] })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

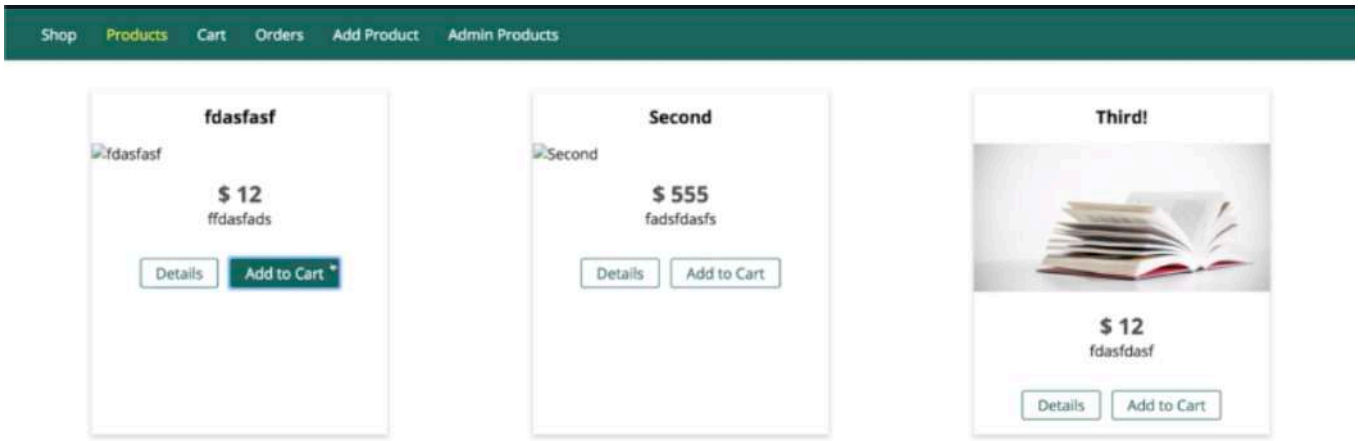
```
<!--./views/shop/cart.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/cart.css">
    </head>

    <body>
        <%- include('../includes/navigation.ejs') %>
        <main>
            <% if (products.length > 0) { %>
                <ul class="cart__item-list">
                    <% products.forEach(p => { %>
                        <li class="cart__item">
                            <h1><%= p.title %></h1>
                            <h2>Quantity: <%= p.quantity %></h2>
                            <form action="/cart-delete-item" method="POST">
                                <input type="hidden" value="<%= p._id %>" name="productId">
                                <button class="btn danger" type="submit">Delete</button>
                            </form>
```

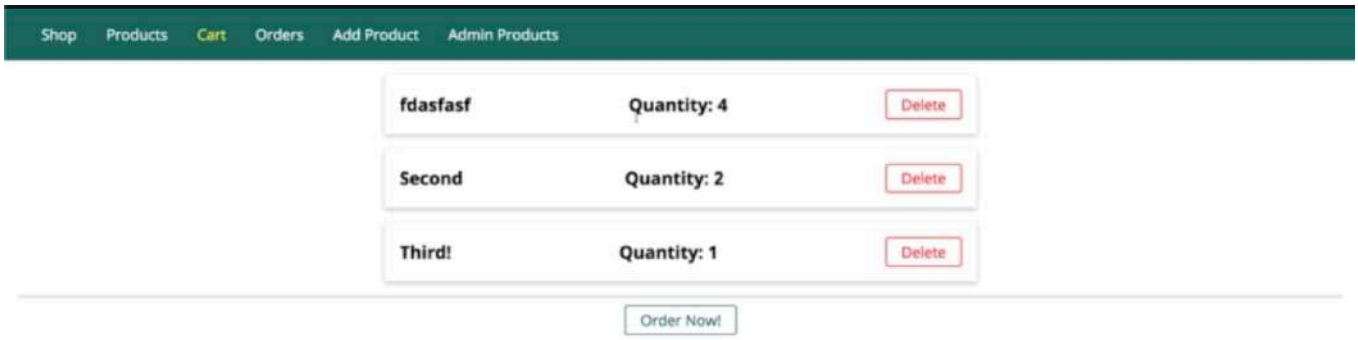
```
                    </li>
                <% }) %>
            </ul>
            <hr>
            <div class="centered">
                <form action="/create-order" method="POST">
                    <button type="submit" class="btn">Order Now!</button>
                </form>
            </div>

        <% } else { %>
            <h1>No Products in Cart!</h1>
        <% } %>
    </main>
    <%- include('../includes/end.ejs') %>
```

```
// ./routes/shop.js

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);

router.get('/cart', shopController.getCart);

router.post('/cart', shopController.postCart);

// router.post('/cart-delete-item', shopController.postCartDeleteProduct);

// router.post('/create-order', shopController.postOrder);

// router.get('/orders', shopController.getOrders);

module.exports = router;
```

* Chapter 196: Fixing A Bug

1. update
- ./controllers/shop.js





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

```
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
```

```
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;
```

```
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
    });
  // let fetchedCart;
  // let newQuantity = 1;
  // req.user
  //   .getCart()
  //   .then(cart => {
  //     fetchedCart = cart;
  //     return cart.getProducts({ where: { id: prodId } });
  //   })
  //   .then(products => {
  //     let product;
  //     if (products.length > 0) {
  //       product = products[0];
  //     }

  //     if (product) {
  //       const oldQuantity = product.cartItem.quantity;
  //       newQuantity = oldQuantity + 1;
  //       return product;
  //     }
  //     return Product.findById(prodId);
  //   })
  //   .then(product => {
  //     return fetchedCart.addProduct(product, {
  //       through: { quantity: newQuantity }
  //     });
  //   })
  //   .then(() => {
  //     res.redirect('/cart');
  //   })
  //   .catch(err => console.log(err));
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .getCart()
    .then(cart => {
      return cart.getProducts({ where: { id: prodId } });
    })
    .then(products => {
      const product = products[0];
      return product.cartItem.destroy();
    })
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};
```

```

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts();
    })
    .then(products => {
      return req.user
        .createOrder()
        .then(order => {
          return order.addProducts(
            products.map(product => {
              product.orderItem = { quantity: product.cartItem.quantity };
              return product;
            })
          );
        })
        .catch(err => console.log(err));
    })
    .then(result => {
      return fetchedCart.setProducts(null);
    })
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({ include: ['products'] })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

* Chapter 197: Deleting Cart Items

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

Shop Products Cart Orders Add Product Admin Products

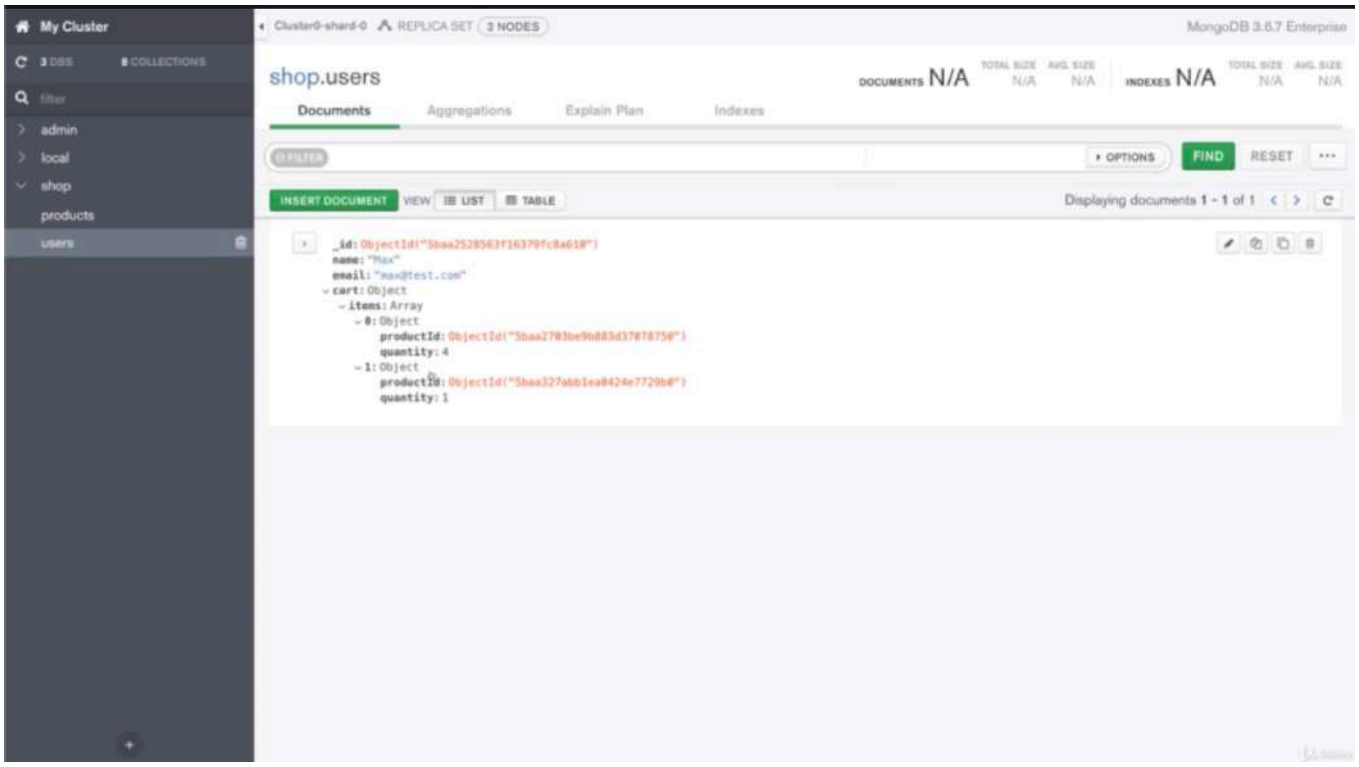
fdasfasf	Quantity: 4	Delete
Second	Quantity: 2	Delete
Third!	Quantity: 1	Delete

Order Now!

Shop Products Cart Orders Add Product Admin Products

fdasfasf	Quantity: 4	Delete
Third!	Quantity: 1	Delete

Order Now!




```javascript
//./controllers/shop.js

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
  // Product.findAll({ where: { id: prodId } })
  //   .then(products => {
  //     res.render('shop/product-detail', {
  //       product: products[0],
  //       pageTitle: products[0].title,
  //       path: '/products'
  //     });
  //   })
  //   .catch(err => console.log(err));
  Product.findById(prodId)
    .then(product => {
      res.render('shop/product-detail', {
        product: product,
        pageTitle: product.title,
        path: '/products'
```

```
    });
  })
  .catch(err => console.log(err));
};

exports.getIndex = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .getCart()
    .then(products => {
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
    .catch(err => console.log(err));
};

exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
    });
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .deleteItemFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
```

```
  req.user
    .getCart()
    .then(cart => {
      fetchedCart = cart;
      return cart.getProducts();
    })
    .then(products => {
      return req.user
        .createOrder()
        .then(order => {
          return order.addProducts(
            products.map(product => {
              product.orderItem = { quantity: product.cartItem.quantity };
              return product;
            })
          );
        })
        .catch(err => console.log(err));
    })
    .then(result => {
      return fetchedCart.setProducts(null);
    })
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({ include: ['products'] })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

```
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }

```

```
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 vanila javascript
71     * 'filter()' allows us to define a criteria
```

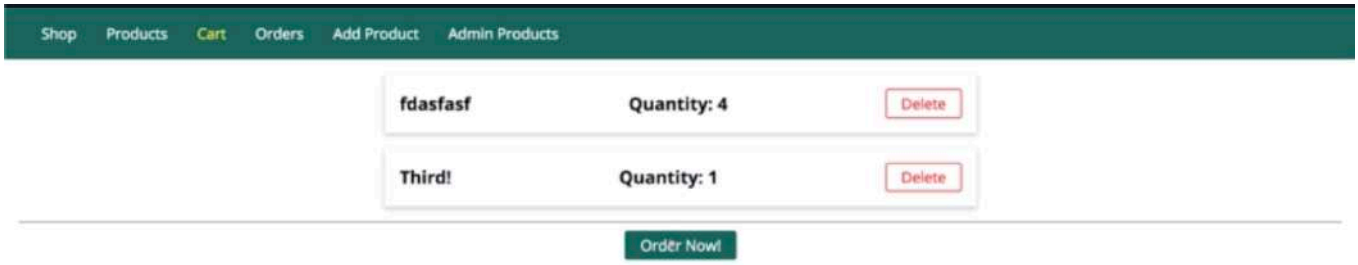
```
    * on how we wanna filter the elements in that array
    * so in this case, the elements of the items array
    * and then it will return a new array with all the filtered items
    * so all the items that make it through the filter
    *
    * 'filter()' is a function which runs on every item
    * and we return true if we wanna keep the item in the new array
    * or false if you want to get rid of it.
    */
    const updatedCartItems = this.cart.items.filter(item => {
      return item.productId.toString() !== productId.toString()
    })
    const db = getDb()
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        /**i wanna assign this to an object with an items property
         * because that was what our cart has.
         *
         */
        { $set: { cart: {items: updatedCartItems} } }
      )
  }

  static findById(userId) {
    const db = getDb();
    return db
      .collection('users')
      .findOne({ _id: new ObjectId(userId) })
      .then(user => {
        console.log(user);
        return user;
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = User;
```

```
// ./routes/shop.js

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);
```

```

router.get('/cart', shopController.getCart);

router.post('/cart', shopController.postCart);

router.post('/cart-delete-item', shopController.postCartDeleteProduct);

// router.post('/create-order', shopController.postOrder);

// router.get('/orders', shopController.getOrders);

module.exports = router;

```

* Chapter 198: Adding An Order

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



Shop Products Cart Orders Add Product Admin Products

Page Not Found!

My Cluster

3 DBS 9 COLLECTIONS

filter

admin

local

shop

orders

products

users

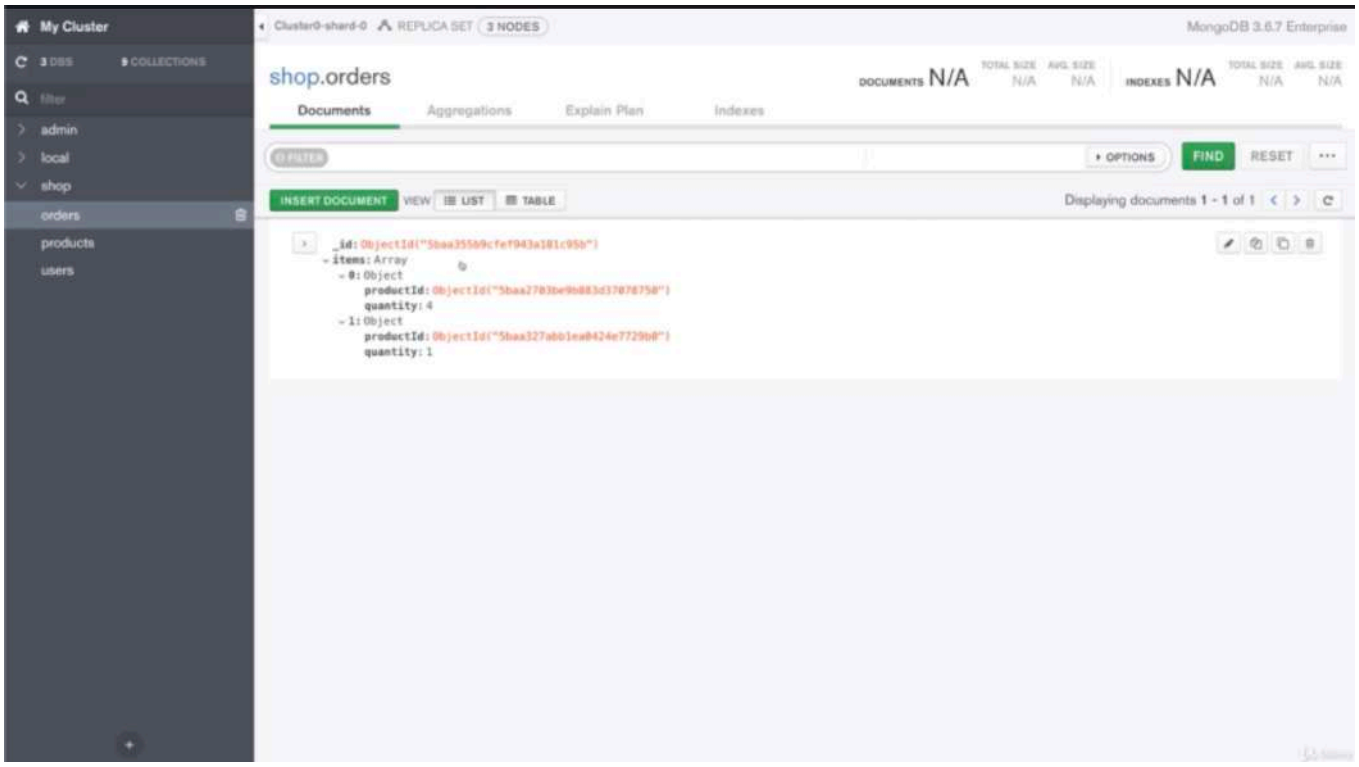
Cluster0-shard-0 REPLICA SET 3 NODES

MongoDB 3.6.7 Enterprise

Collections

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



No Products in Cart!

```
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }
```

```
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 => {
```

```
      return item.productId.toString() !== productId.toString()
    })
    const db = getDb()
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        { $set: { cart: {items: updatedCartItems} } }
      )
  }

  addOrder(){
    /**this doesn't take any arguments
     * because the cart which will be passed as an order or as the data for the order
     * is already registered on this user
     * so i need to add the orders to my user or the other way around
     */
    const db = getDb()
    /**i will return the entire thing
     * and the one new order will be well the cart i currently have
     * i wanna insert my cart which refers to the users cart
     */
    return db
      .collection('orders')
      .insertOne(this.cart)
      .then(result => {
        this.cart = {items: []}
        return db
            .collection('users')
            .updateOne(
              { _id: new ObjectId(this._id) },
              { $set: { cart: { items: [] } } }
            )
    })
  }

  static findById(userId) {
    const db = getDb();
    return db
      .collection('users')
      .findOne({ _id: new ObjectId(userId) })
      .then(user => {
        console.log(user);
        return user;
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = User;
```

```
//./controllers/shop.js

const Product = require('../models/product');

```

```
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',
```

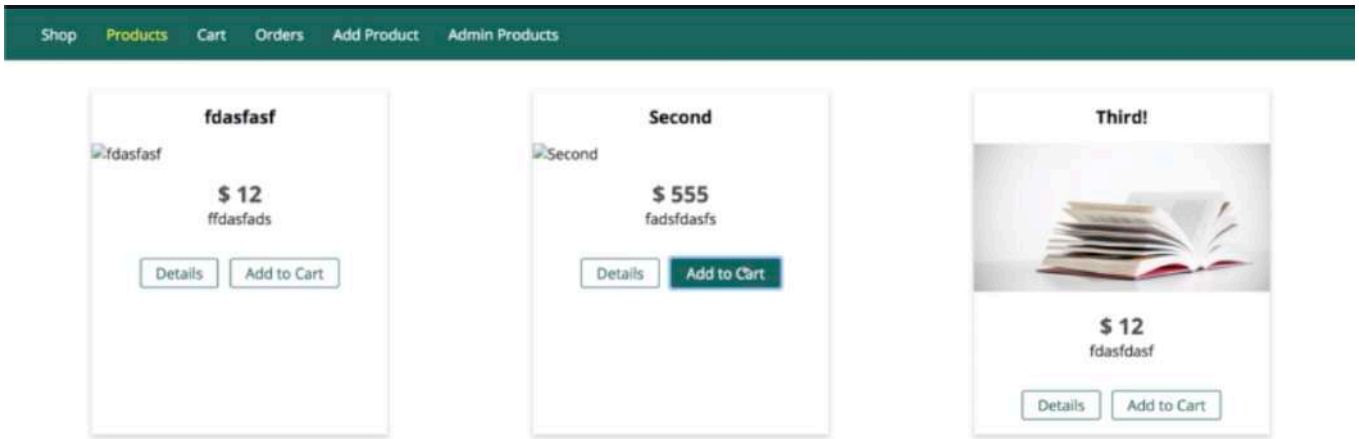
```
      pageTitle: 'Your Cart',
      products: products
    });
  })
  .catch(err => console.log(err));
};

exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
    });
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .deleteItemFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .addOrder()
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders({ include: ['products'] })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

```
// ./routes/shop.js

const path = require('path');

```

```
const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);

router.get('/cart', shopController.getCart);

router.post('/cart', shopController.postCart);

router.post('/cart-delete-item', shopController.postCartDeleteProduct);

router.post('/create-order', shopController.postOrder);

// router.get('/orders', shopController.getOrders);

module.exports = router;

```

* Chapter 199: Adding Relational Order Data

1. update
- ./models/user.js



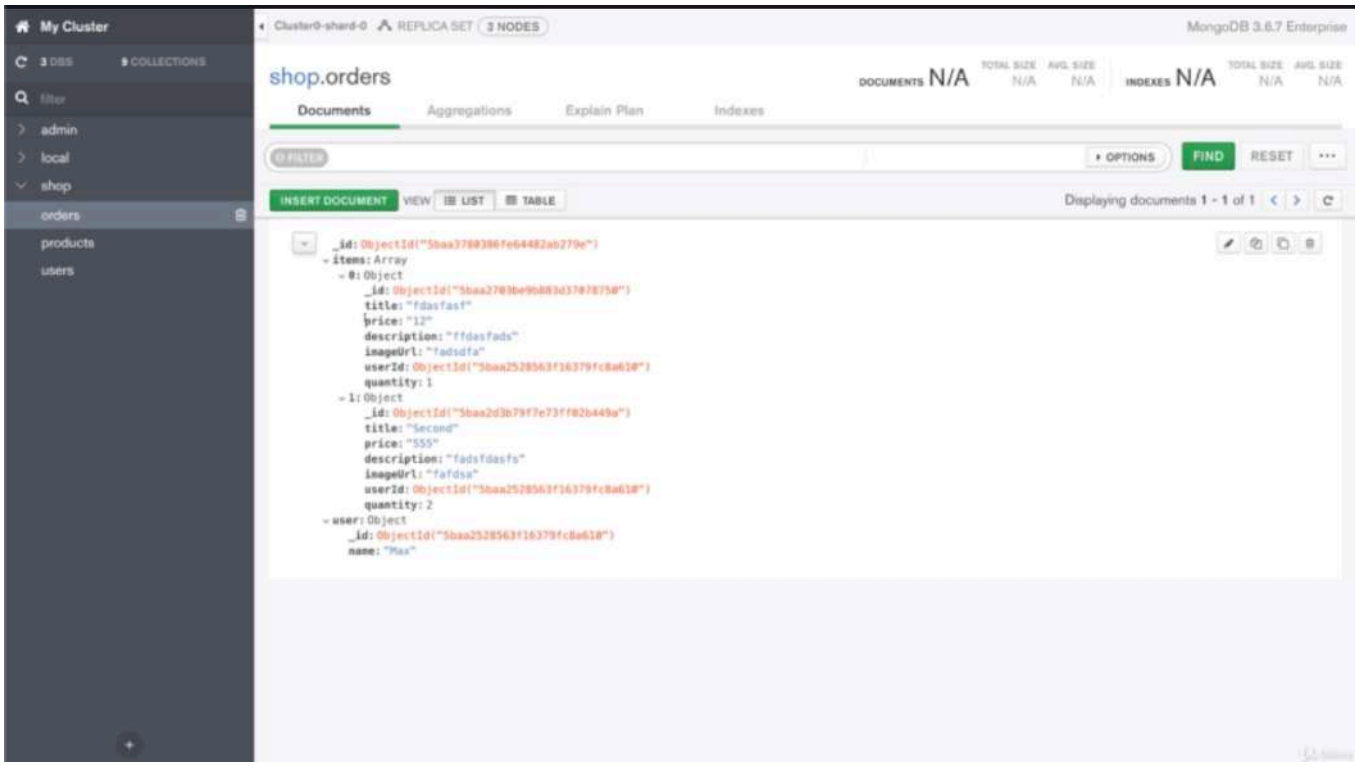
Shop Products Cart Orders Add Product Admin Products

fdasfasf	Quantity: 1	Delete
Second	Quantity: 2	Delete

Order Now!

Shop Products Cart Orders Add Product Admin 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

```
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }

  save() {
    const db = getDb();
    return db.collection('users').insertOne(this);
  }

  addToCart(product) {
    const cartProductIndex = this.cart.items.findIndex(cp => {
      return cp.productId.toString() === product._id.toString();
    });
    let newQuantity = 1;
    const updatedCartItems = [...this.cart.items];
    if (cartProductIndex >= 0) {
      newQuantity = this.cart.items[cartProductIndex].quantity + 1;
      updatedCartItems[cartProductIndex].quantity = newQuantity;
    } else {
      updatedCartItems.push({
        productId: new ObjectId(product._id),
```

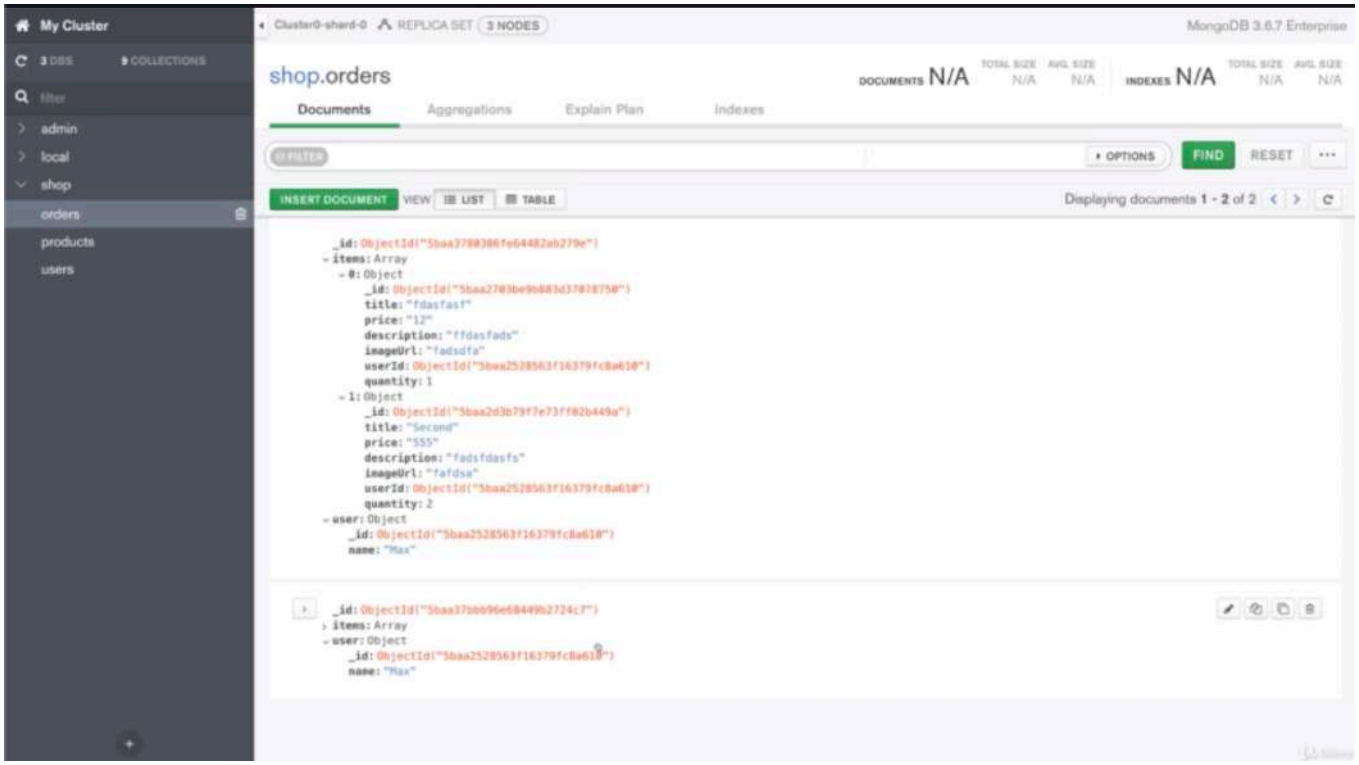
```
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  };
  const db = getDb();
  return db
    .collection('users')
    .updateOne(
      { _id: new ObjectId(this._id) },
      { $set: { cart: updatedCart } }
    );
}

getCart() {
  const db = getDb();
  const productIds = this.cart.items.map(i => {
    return i.productId;
  });
  return db
    .collection('products')
    .find({ _id: { $in: productIds } })
    .toArray()
    .then(products => {
      return products.map(p => {
        return {
          ...p,
          quantity: this.cart.items.find(i => {
            return i.productId.toString() === p._id.toString();
          }).quantity
        };
      });
    });
}

deleteItemFromCart(productId){
  const updatedCartItems = this.cart.items.filter(item => {
    return item.productId.toString() !== productId.toString()
  })
  const db = getDb()
  return db
    .collection('users')
    .updateOne(
      { _id: new ObjectId(this._id) },
      { $set: { cart: {items: updatedCartItems} } }
    )
}

addOrder(){
  const db = getDb()
  /**i have that products data
   * because outside of that 'then()' block,
   * the code would execute too early.
   *
   * i need to 'return' the result of this 'addOrder()'
```

```
    * so that outside of addOrder(), in my ./controllers/shop.js,
    * i can call 'then()' in 'postOrder()'
    */
    return this.getCart().then(products => {
      const order = {
        /**then my items will be my products
         * so an array of products with the product information and the quantity
         * so now the product information will also be part of the order
         *
         * and here i really don't care about that information changing
         * because if it should change, for orders, we need a snapshot anyways
         * if the price of a product changes, that doesn't affect the past order
         * so we wouldn't wanna update the price even if it would change
         * so for orders, such a snapshot and therefore an embedded document is a great way of relating the order and the product
         * because the product data might be duplicate but it doesn't need to change in the orders collection
         * because we want a snapshot.
         */
        items: products,
        /**we wanna have the 'cart.items' in there
         * but we also wanna have some information about the user
         * and i will add an embedded document where i add the '_id' and here,
         * i wanna create a 'new objectId' based on the ID of the user we are working with
         * but i also wanna store the name which we have as a property here and the email
         * so i will duplicate data because this will then end up in the orders collection and in this users collection
         */
        user: {
          _id: new ObjectId(this._id),
          name: this.name,
        }
      }
      return db
      .collection('orders')
      .insertOne(order)
    })
      .then(result => {
        this.cart = {items: []}
        return db
            .collection('users')
            .updateOne(
              { _id: new ObjectId(this._id) },
              { $set: { cart: { items: [] } } }
            )
    })
  }

  getOrders(){
    const db = getDb()
    //return db.collection('orders').
  }

  static findById(userId) {
    const db = getDb();
    return db
```

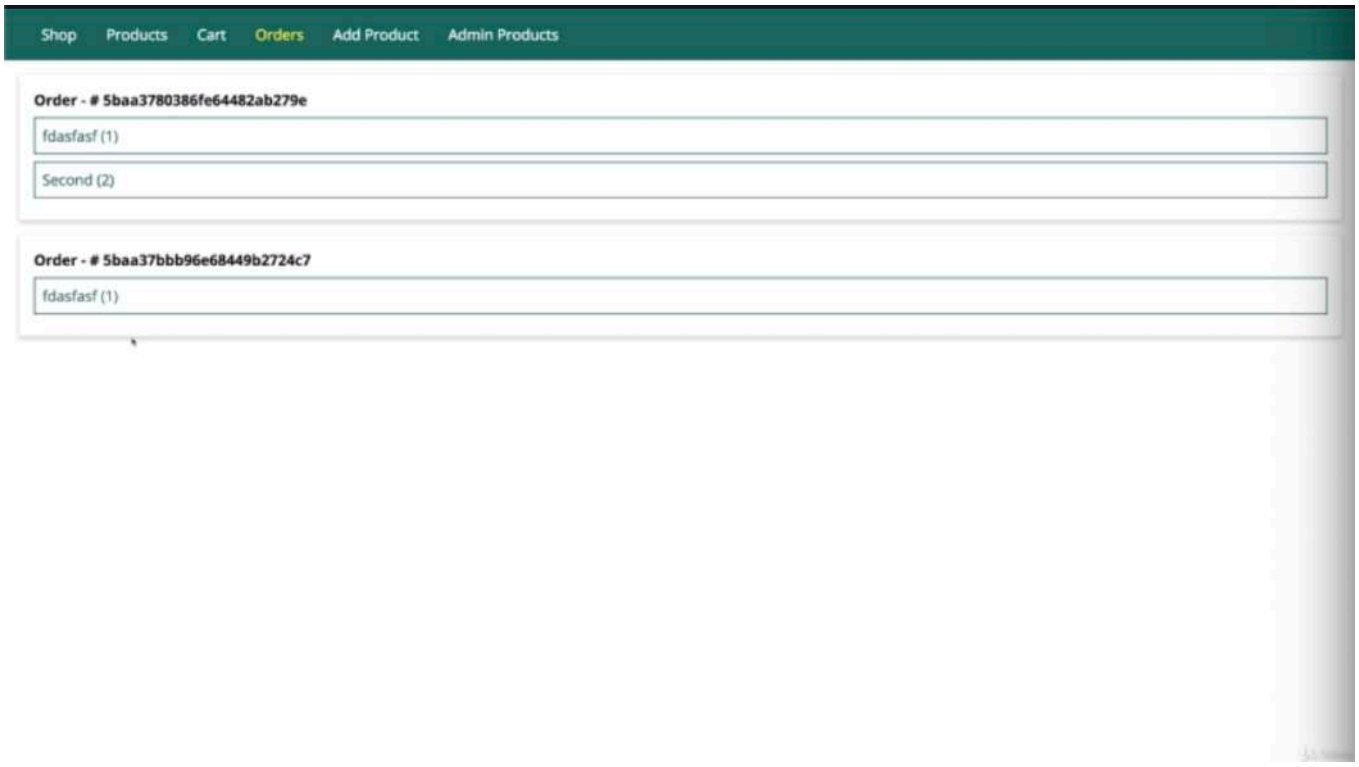
```
      .collection('users')
      .findOne({ _id: new ObjectId(userId) })
      .then(user => {
        console.log(user);
        return user;
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = User;
```

* Chapter 200: Getting Orders

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



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




```javascript
//./models/user.js

const mongodb = require('mongodb');
const getDb = require('../util/database').getDb;

const ObjectId = mongodb.ObjectId;

class User {
  constructor(username, email, cart, id) {
    this.name = username;
    this.email = email;
    this.cart = cart; // {items: []}
    this._id = id;
  }

  save() {
    const db = getDb();
    return db.collection('users').insertOne(this);
  }

  addToCart(product) {
    const cartProductIndex = this.cart.items.findIndex(cp => {
      return cp.productId.toString() === product._id.toString();
    });
    let newQuantity = 1;
    const updatedCartItems = [...this.cart.items];
    if (cartProductIndex >= 0) {
      newQuantity = this.cart.items[cartProductIndex].quantity + 1;
      updatedCartItems[cartProductIndex].quantity = newQuantity;
    } else {
      updatedCartItems.push({
        productId: new ObjectId(product._id),
        quantity: newQuantity
      });
    }
```

```
    const updatedCart = {
      items: updatedCartItems
    };
    const db = getDb();
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        { $set: { cart: updatedCart } }
      );
  }

  getCart() {
    const db = getDb();
    const productIds = this.cart.items.map(i => {
      return i.productId;
    });
    return db
      .collection('products')
      .find({ _id: { $in: productIds } })
      .toArray()
      .then(products => {
        return products.map(p => {
          return {
            ...p,
            quantity: this.cart.items.find(i => {
              return i.productId.toString() === p._id.toString();
            }).quantity
          };
        });
      });
  }

  deleteItemFromCart(productId){
    const updatedCartItems = this.cart.items.filter(item => {
      return item.productId.toString() !== productId.toString()
    })
    const db = getDb()
    return db
      .collection('users')
      .updateOne(
        { _id: new ObjectId(this._id) },
        { $set: { cart: {items: updatedCartItems} } }
      )
  }

  addOrder(){
    const db = getDb()
    return this.getCart().then(products => {
      const order = {
        items: products,
        user: {
          _id: new ObjectId(this._id),
          name: this.name,
        }
      }
```

```
      return db
      .collection('orders')
      .insertOne(order)
    })
      .then(result => {
        this.cart = {items: []}
        return db
                .collection('users')
                .updateOne(
                  { _id: new ObjectId(this._id) },
                  { $set: { cart: { items: [] } } }
                )
    })
  }

  getOrders(){
    const db = getDb()
    /**each order has a user object
     * and in that user object, we have the ID of that user
     * so we need to compare that ID to the current user ID
     * now to do that, we add a filter
     * and now in MongoDB, you can check nested properties by defining the path to them
     * the important thing to know here is that
     * you need to use quotation marks '' around the path
     * and then you can say check user and then the ID for the user
     *
     * 'user._id' will look for _id in the user property which holds an embedded document
     * and then i can compare it is to a new objectId for this ID
     * and this should give me all orders for that user
     * and this will now be more than one
     * so again we can use the 'toArray()' shortcut
     * and return that data to return an array of orders for that user
     */
    return db
      .collection('orders')
      .find({ 'user._id': new ObjectId(this._id) })
      .toArray()
  }

  static findById(userId) {
    const db = getDb();
    return db
      .collection('users')
      .findOne({ _id: new ObjectId(userId) })
      .then(user => {
        console.log(user);
        return user;
      })
      .catch(err => {
        console.log(err);
      });
  }
}

module.exports = User;
```

```
//./controllers/shop.js
```

```

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
  // Product.findAll({ where: { id: prodId } })
  //   .then(products => {
  //     res.render('shop/product-detail', {
  //       product: products[0],
  //       pageTitle: products[0].title,
  //       path: '/products'
  //     });
  //   })
  //   .catch(err => console.log(err));
  Product.findById(prodId)
    .then(product => {
      res.render('shop/product-detail', {
        product: product,
        pageTitle: product.title,
        path: '/products'
      });
    })
    .catch(err => console.log(err));
};

exports.getIndex = (req, res, next) => {
  Product.fetchAll()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .getCart()
```

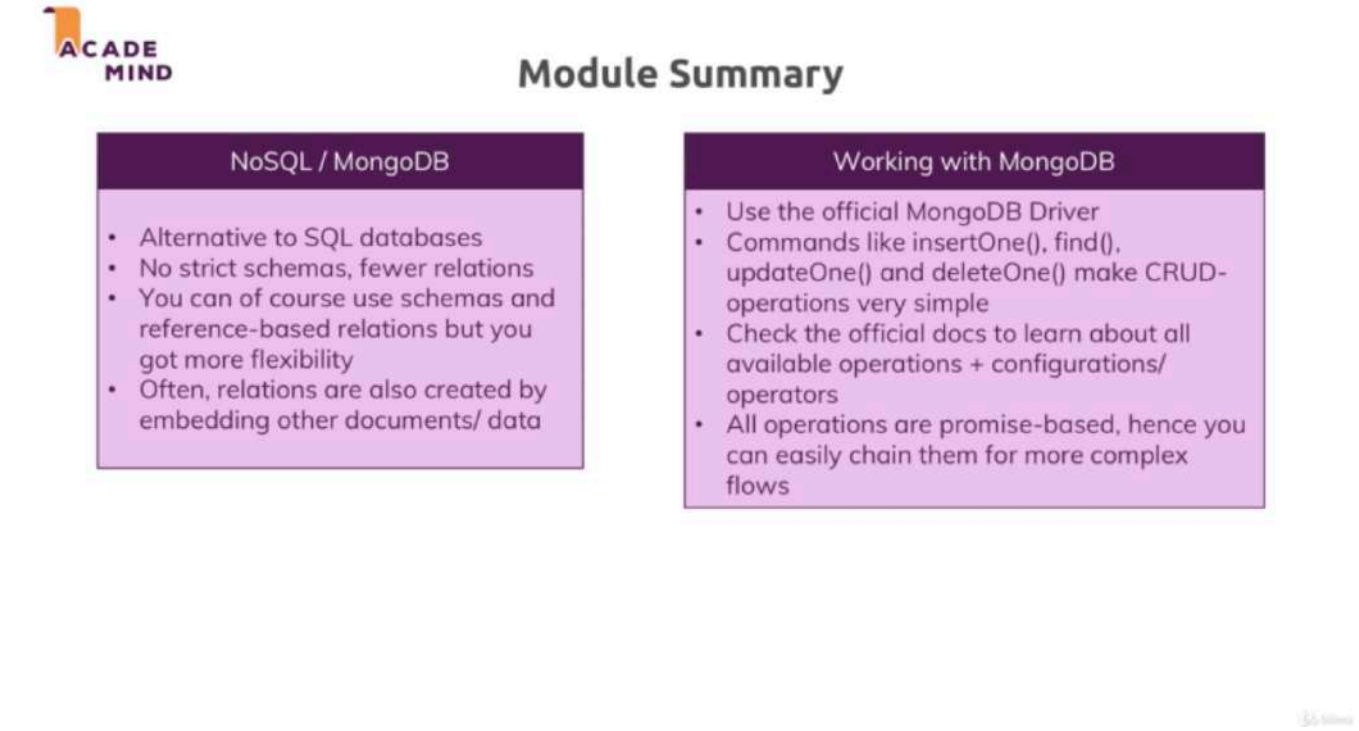
```
    .then(products => {
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
    .catch(err => console.log(err));
};

exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
    });
};

exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .deleteItemFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  let fetchedCart;
  req.user
    .addOrder()
    .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  req.user
    .getOrders()
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

```
// ./routes/shop.js
```

```

const path = require('path');

const express = require('express');

const shopController = require('../controllers/shop');

const router = express.Router();

router.get('/', shopController.getIndex);

router.get('/products', shopController.getProducts);

router.get('/products/:productId', shopController.getProduct);

router.get('/cart', shopController.getCart);

router.post('/cart', shopController.postCart);

router.post('/cart-delete-item', shopController.postCartDeleteProduct);

router.post('/create-order', shopController.postOrder);

router.get('/orders', shopController.getOrders);

module.exports = router;

```

```
<!--./views/shop/orders.ejs-->

<%- include('../includes/head.ejs') %>
    </head>

    <body>
        <%- include('../includes/navigation.ejs') %>
        <main>
            <% if (orders.length <= 0) { %>
                <h1>Nothing there!</h1>
            <% } else { %>
                <ul>
                    <% orders.forEach(order => { %>
                        <li>
                            <h1># <%= order._id %></h1>
                            <ul>
                                <% order.items.forEach(product => { %>
                                    <li><%= product.title %> (<%= product.quantity %>)</li>
                                <% }); %>
                            </ul>
                        </li>
                    <% }); %>
                </ul>
            <% } %>
        </main>
        <%- include('../includes/end.ejs') %>
```

* Chapter 202: Wrap Up



* 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