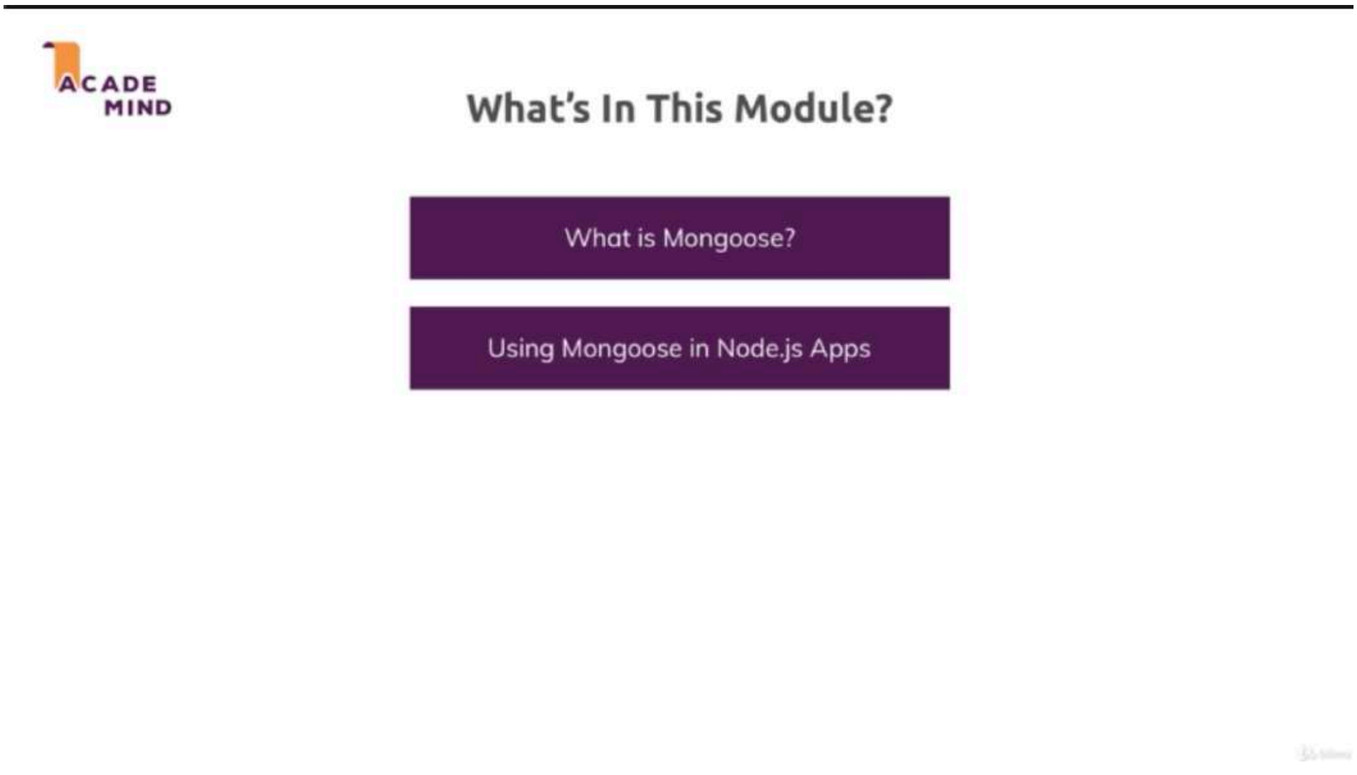
13. Working With Mongoose

* Chapter 205: Module Introduction

![](images/205-module-introduction-1.png)



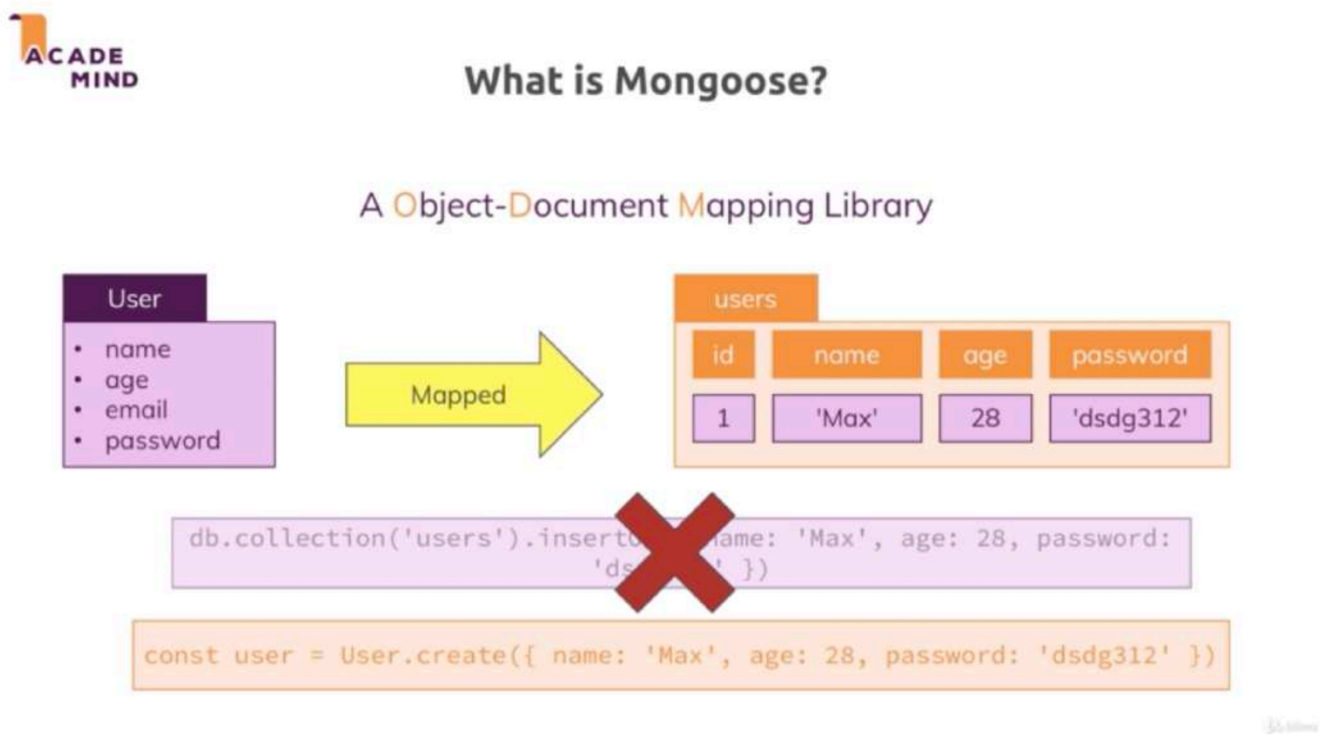
What's In This Module?

What is Mongoose?

Using Mongoose in Node.js Apps

* Chapter 206: What is Mongoose?

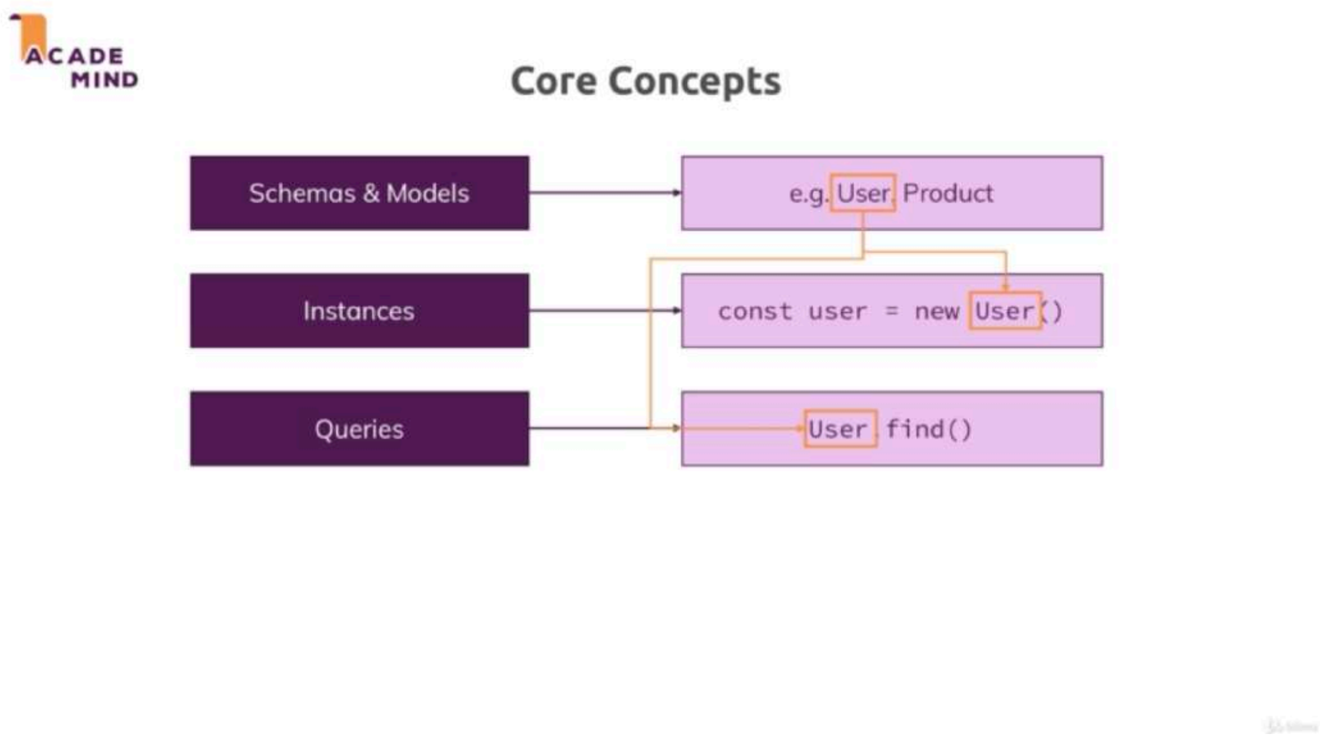
![](images/206-what-is-mongoose-1.png)



```
db.collection('users').insertOne({name: 'Max', age: 28, password: 'dsdg312' })
```

```
const user = User.create({ name: 'Max', age: 28, password: 'dsdg312' })
```

- Mongoose is an ORM standing for ‘Object-Document Mapping’ Library which is really similar to sequelize which was an ORM standing for 'Object-Relational Mapping’ Library.
- difference between is that MongoDB is not relational database. it’s document database.
- Mongoose allows us to define models with which we then work and where all the quries are done behind the scene which doesn’t means that we can’t influence and that we can’t change somethings.
![](images/206-what-is-mongoose-2.png)



```
const user = new User()
```

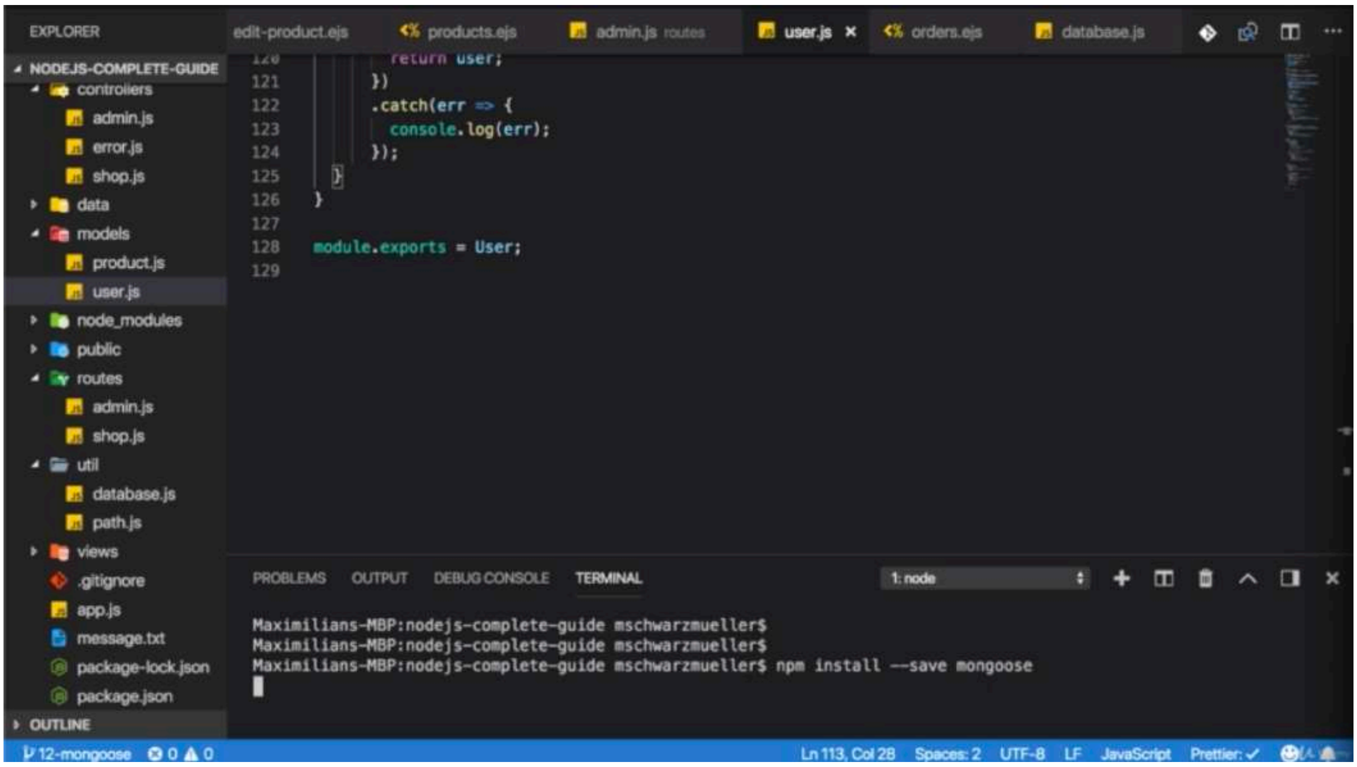
```
User.find()
```

* Chapter 207: Connecting To The MongoDB Server With Mongoose

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

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

![](images/207-connecting-to-the-mongodb-server-with-mongoose-1.png)



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

  module.exports = User;
```

```
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save mongoose
```

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

```
//app.js

const path = require('path');

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

const errorController = require('./controllers/error');
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) => {
  User.findById('5cb7d12855fbe74b129c0b7c')
    .then(user => {
```

```javascript
      req.user = new User(user.namem, user.email, user.cart, user._id);
      next();
    })
    .catch(err => console.log(err));
});

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

app.use(errorController.get404);

/**we already have everything in place we need to connect
 * and mongoose will manage that one connection behind the scenes.
 * so taht in other places where we start using mongoose from the mongoose package,
 * we use that same connection we set up here.
 */
mongoose
  .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/shop?retryWrites=true')
  .then(result => {
    app.listen(3000)
  })
  .catch(err => {
    console.log(err)
  })
```

```javascript
// ./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;

```

```javascript
// ./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;
```

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

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

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;
*/
```

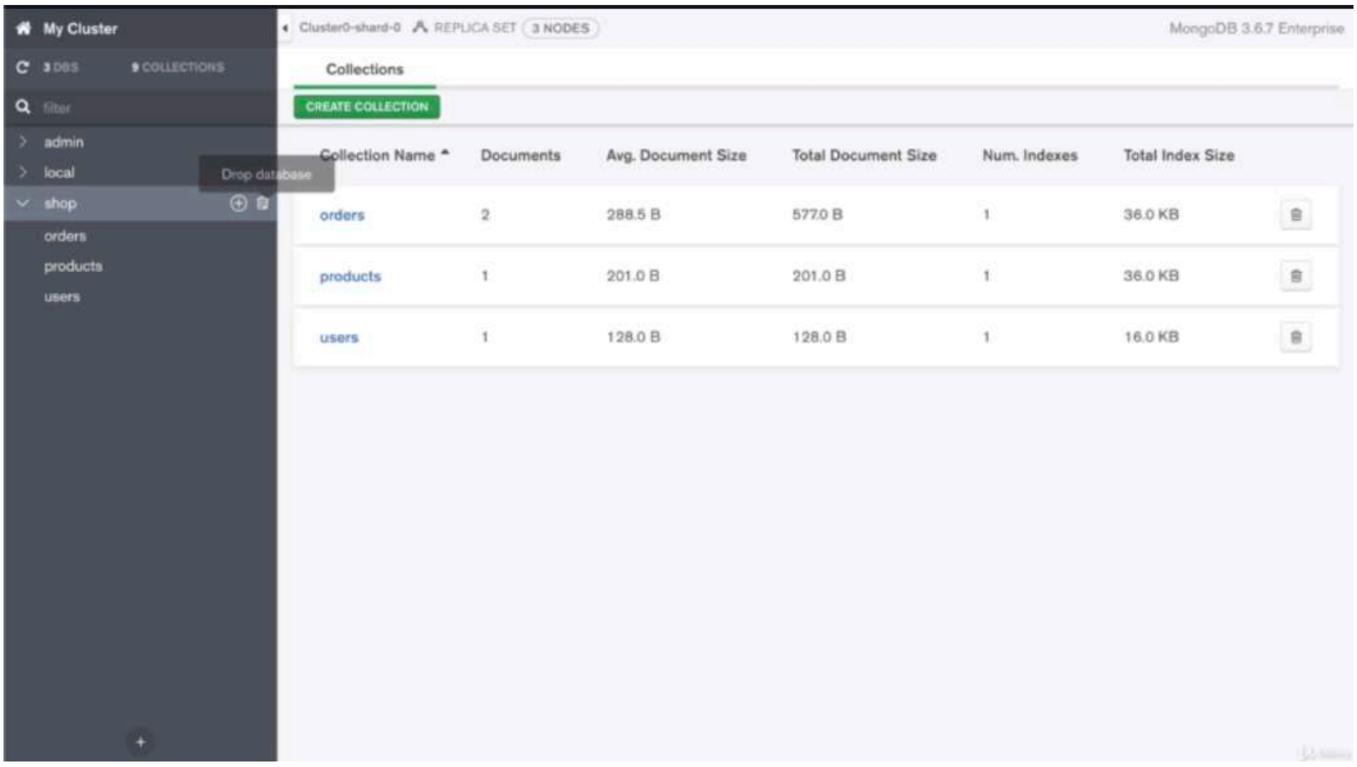
```
//./models/user.js
```

```

/*
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()
    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;
*/
```

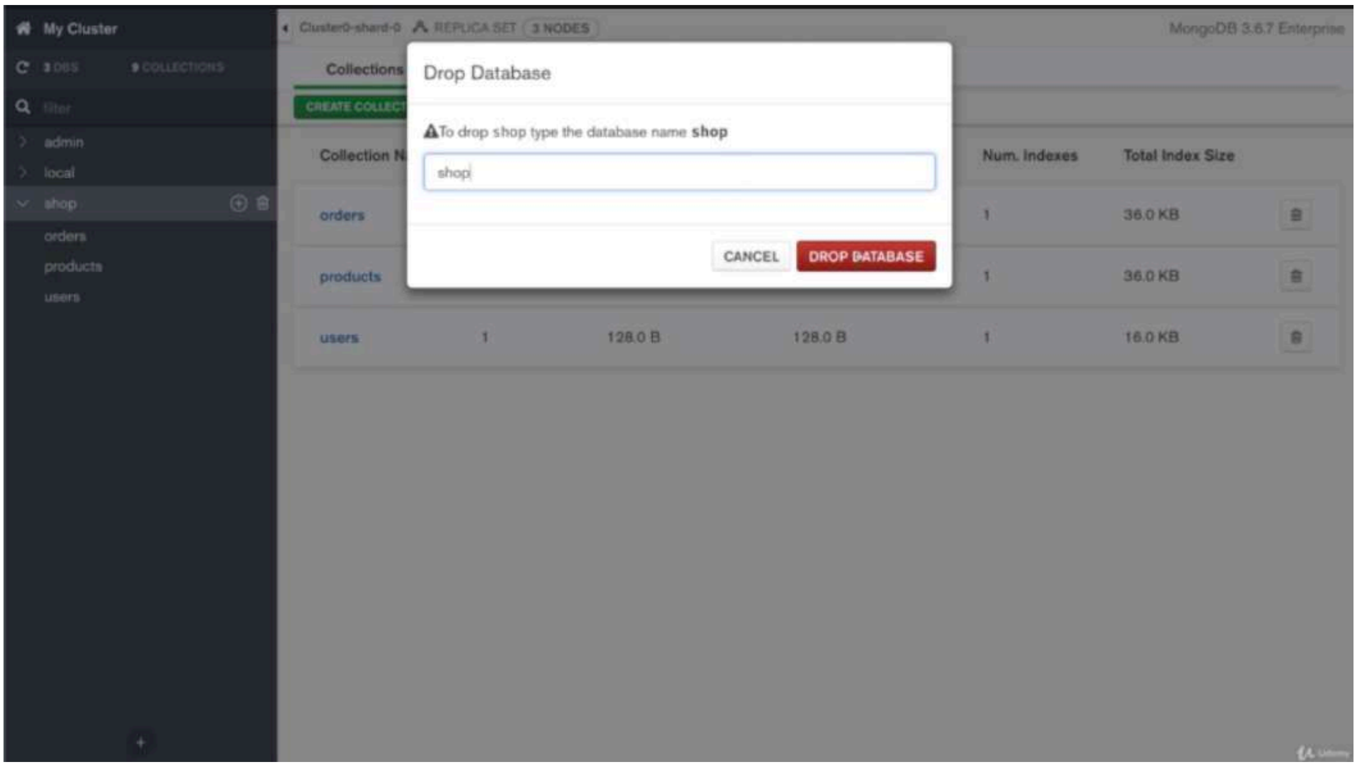
* Chapter 208: Creating The Product Schema

1. update
- ./models/product.js

![](images/208-creating-the-product-schema-1.png)
![](images/208-creating-the-product-schema-2.png)
![](images/208-creating-the-product-schema-3.png)

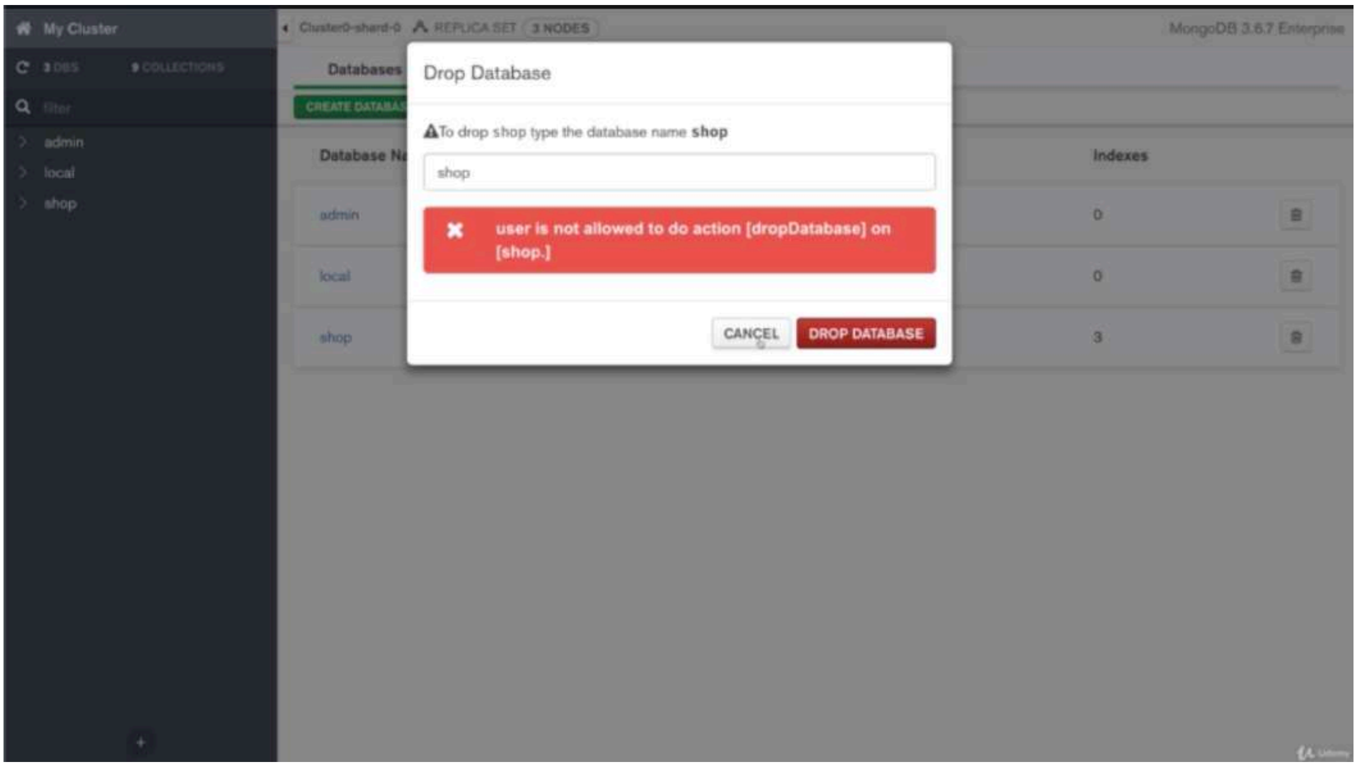


Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
orders	2	288.5 B	577.0 B	1	36.0 KB
products	1	201.0 B	201.0 B	1	36.0 KB
users	1	128.0 B	128.0 B	1	16.0 KB


Drop Database

To drop shop type the database name **shop**

shop

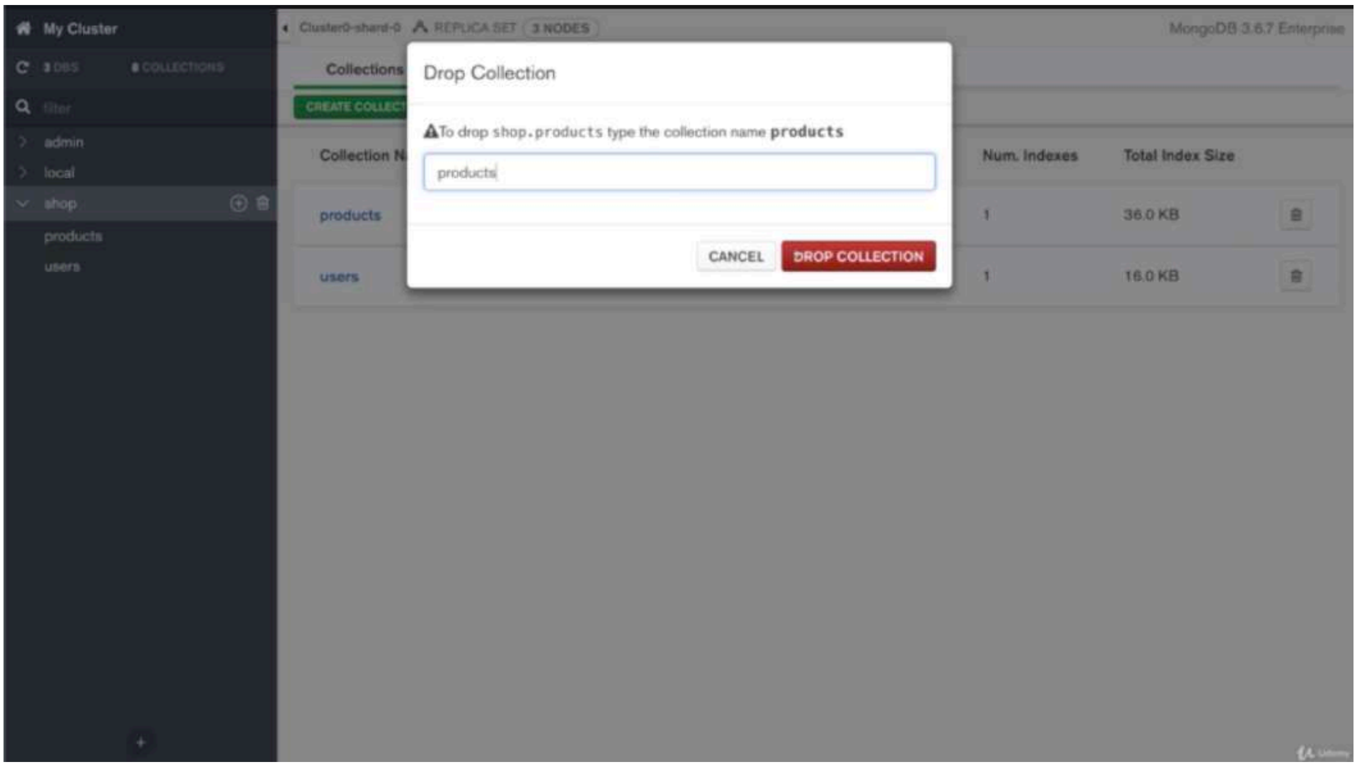

Drop Database

To drop shop type the database name **shop**

shop

user is not allowed to do action [dropDatabase] on [shop.]

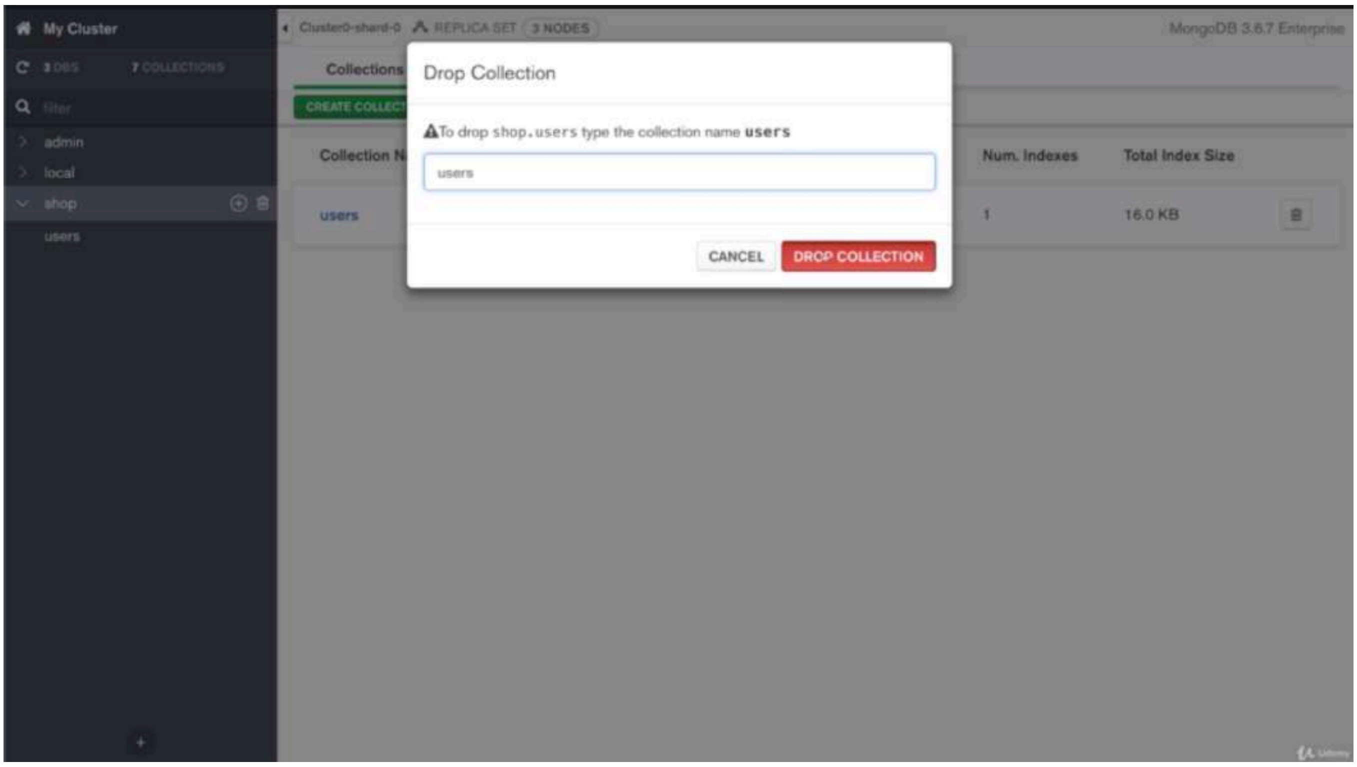
- time to fix our code and make it work again. for that first of all, i connected to my MongoDB server with MongoDB Compass again and there i wamnna clear everything so that we can start from scratch.
- therefore, i will go to my 'shop' database and simply delete that entire database.
- but we got problem that i connected with the wrong user where i'm not allowed to delete a database because i connected with a user who has only read or write access.
![](images/208-creating-the-product-schema-4.png)
![](images/208-creating-the-product-schema-5.png)
![](images/208-creating-the-product-schema-6.png)



Drop Collection

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

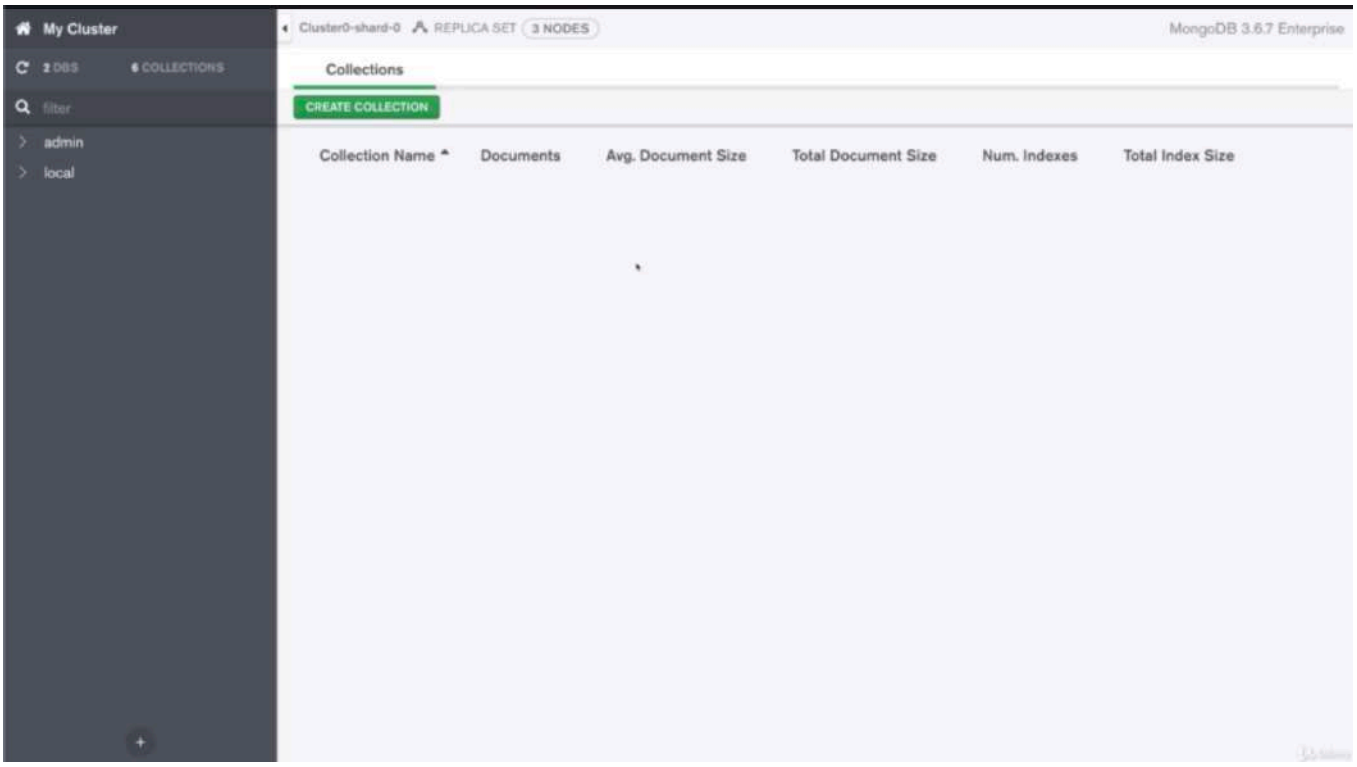
products



Drop Collection

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

users



- so i will just delete the collections here. the alternative would be to simply connect with a user where i'm allowed to manage the overall database.
- i got rid of the shop database and now we can start working from scratch again.

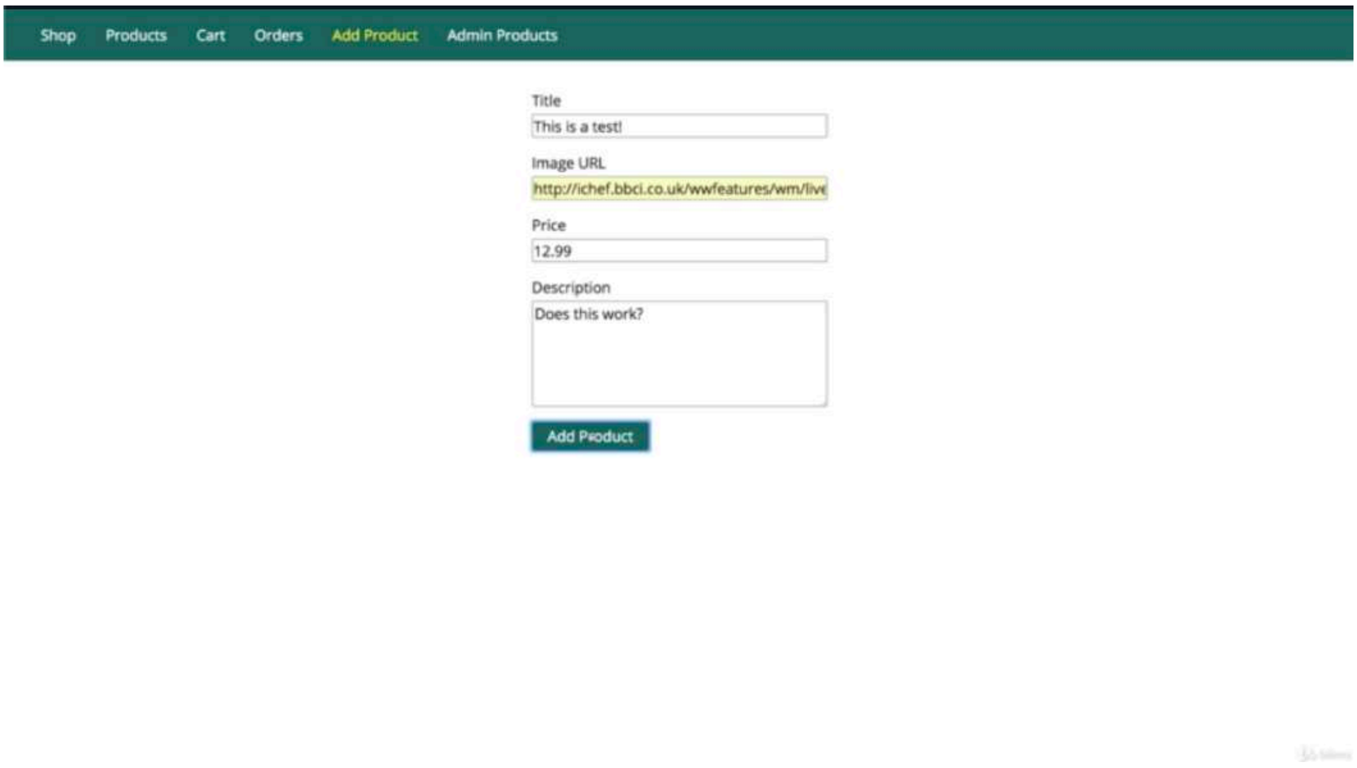
```
//./models/product.js

const mongoose = require('mongoose')

/**'Schema' constructor allows me to create new schemas */
const Schema = mongoose.Schema;

/**you now define the data schema of a product in our case here.
 * you don't just define which keys you have
 * but also which type these keys will have.
 */
const productSchema = new Schema({
  /** this would say
   * OK so i create a schema for an object
   * which i will eventually be able to work with
   * which must have or which will have a title that is of type 'String'
   *
   * MongoDB is Schemaless,
   * so why do we start to create Schemas?
   * the idea is that whilst we have the flexibility of not being restricted to specific
schema,
   * we will have a certain structure in the data we work with
   * and therefore Mongoose wanna give you the advantage of focusing on your data
   * but for that, it needs to know how your data looks like
   * and therefore we define such a schema for the structure our data will have.
   *
   * we could even work with a product and create a new one and save it to the database
without setting a title
   * because we still have the flexibility of not enforcing this,
   * though what we can do is we can pass an object instead of the type as a value
   * and then set a type property which could be set to 'String'
   * and then set required to true
   * this is a more complex way of configuring the value for this key.
   *
   * we would say, the type of this is a string as before
   * but it's also required
   * and now we give up some of the flexibility we had before
   * and we force all objects to have a title
   * but in the end, in our application,
   * every product needs to have a title
   * because we will run into other errors otherwise.
   */
  title: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  imageUrl: {
    type: String,
    required: true
  }
  /**i don't add '_id'
   * because this will still be added automatically as an objectId
   * so we don't need to define here.
   */
})
```

* Chapter 209: Saving Data Through Mongoose

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

![](images/209-saving-data-through-mongoose-1.png)

![](images/209-saving-data-through-mongoose-2.png)



Title

This is a test!

Image URL

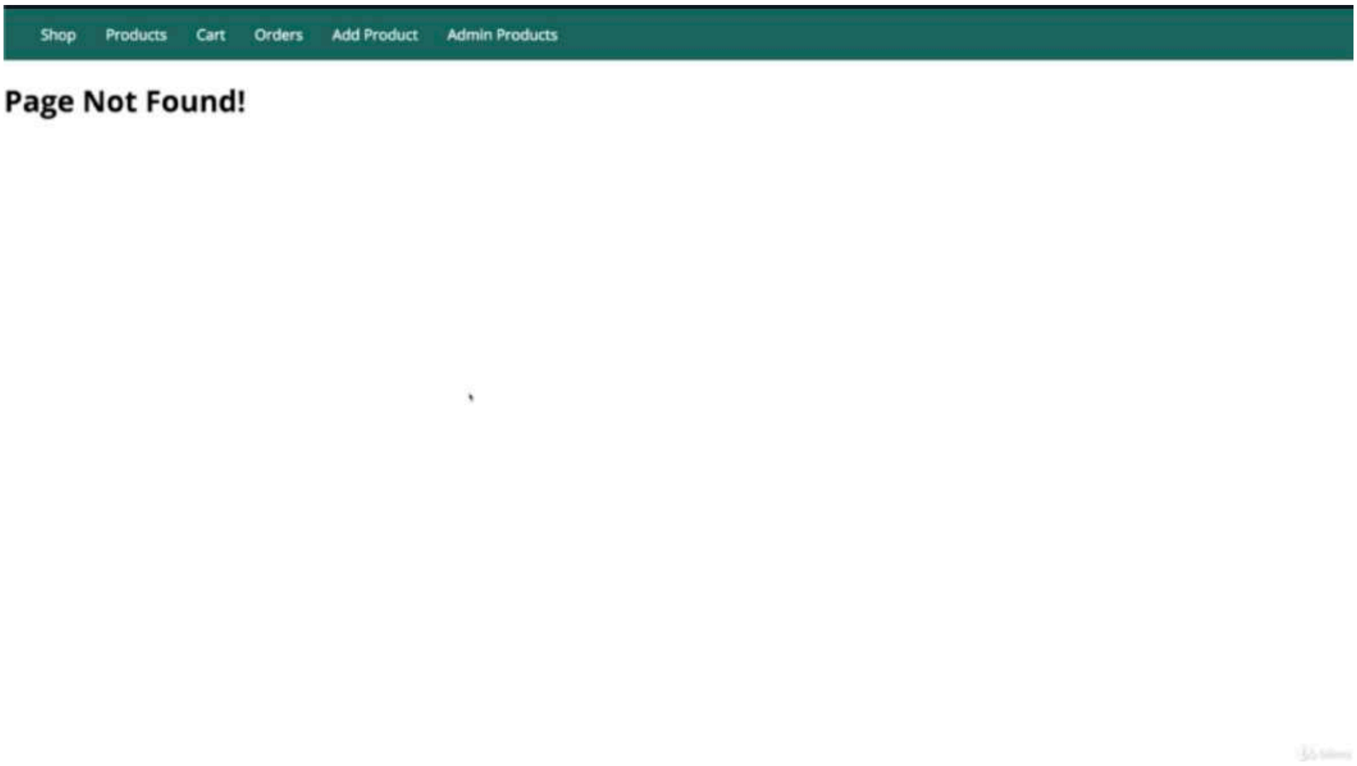
http://ichef.bbci.co.uk/wwfeatures/wm/live

Price

12.99

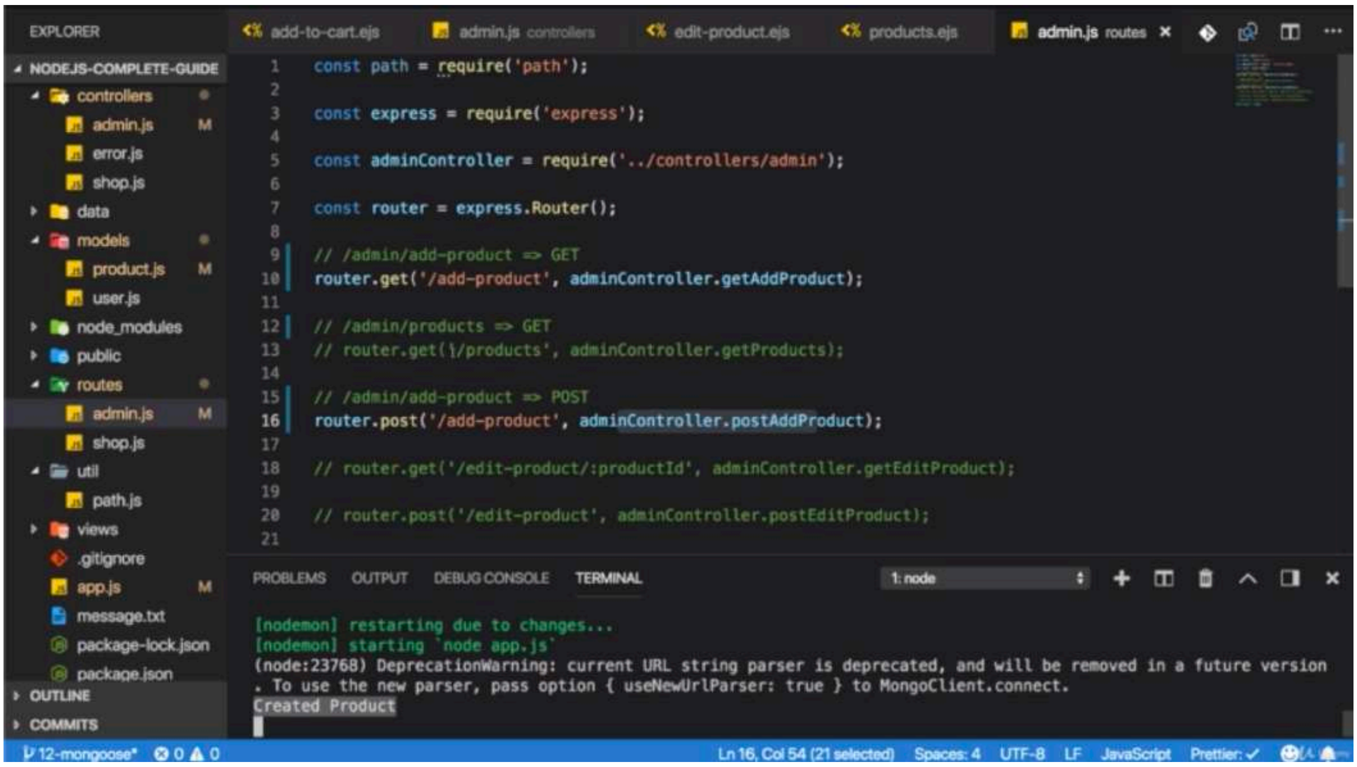
Description

Does this work?



Page Not Found!

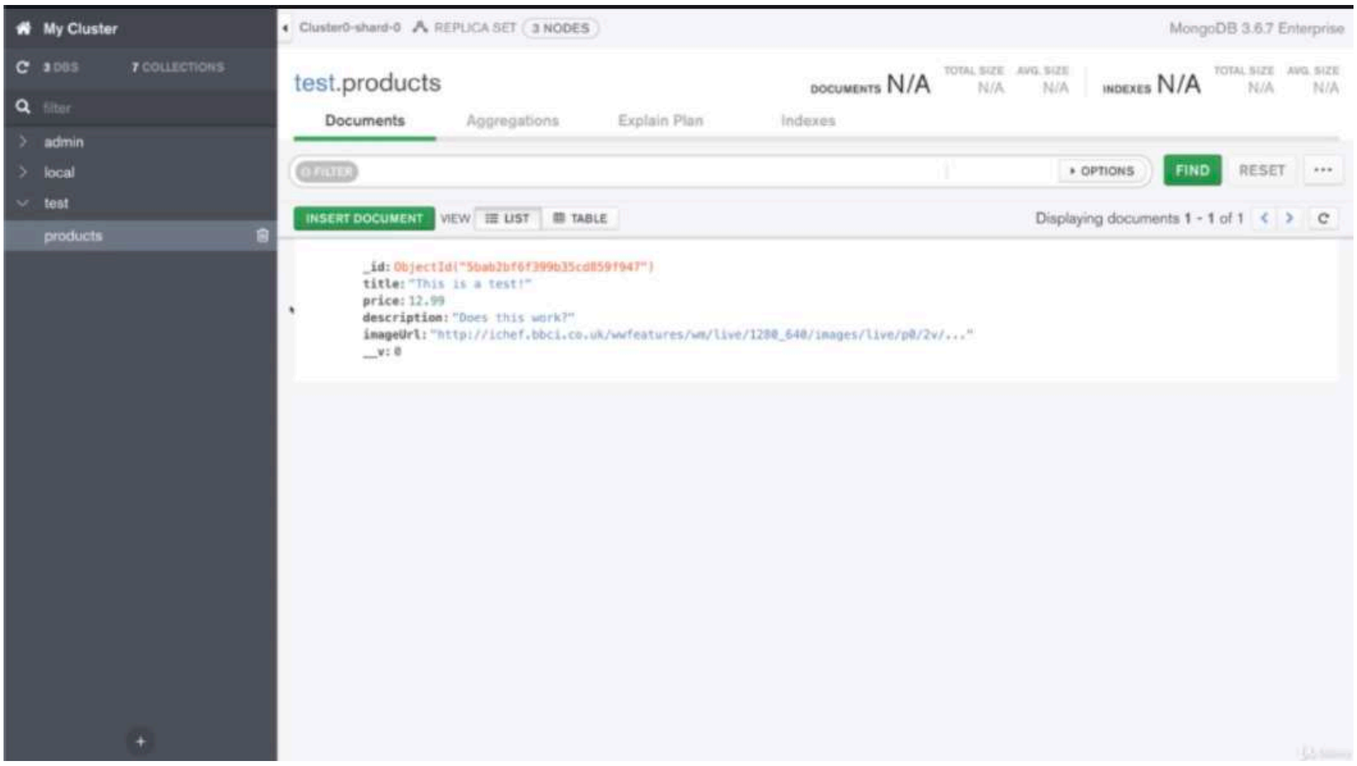
- let’s test this.
- we get a 'page not found’ because we can’t load any other pages, that is OK.
![](images/209-saving-data-through-mongoose-3.png)



```
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);
```

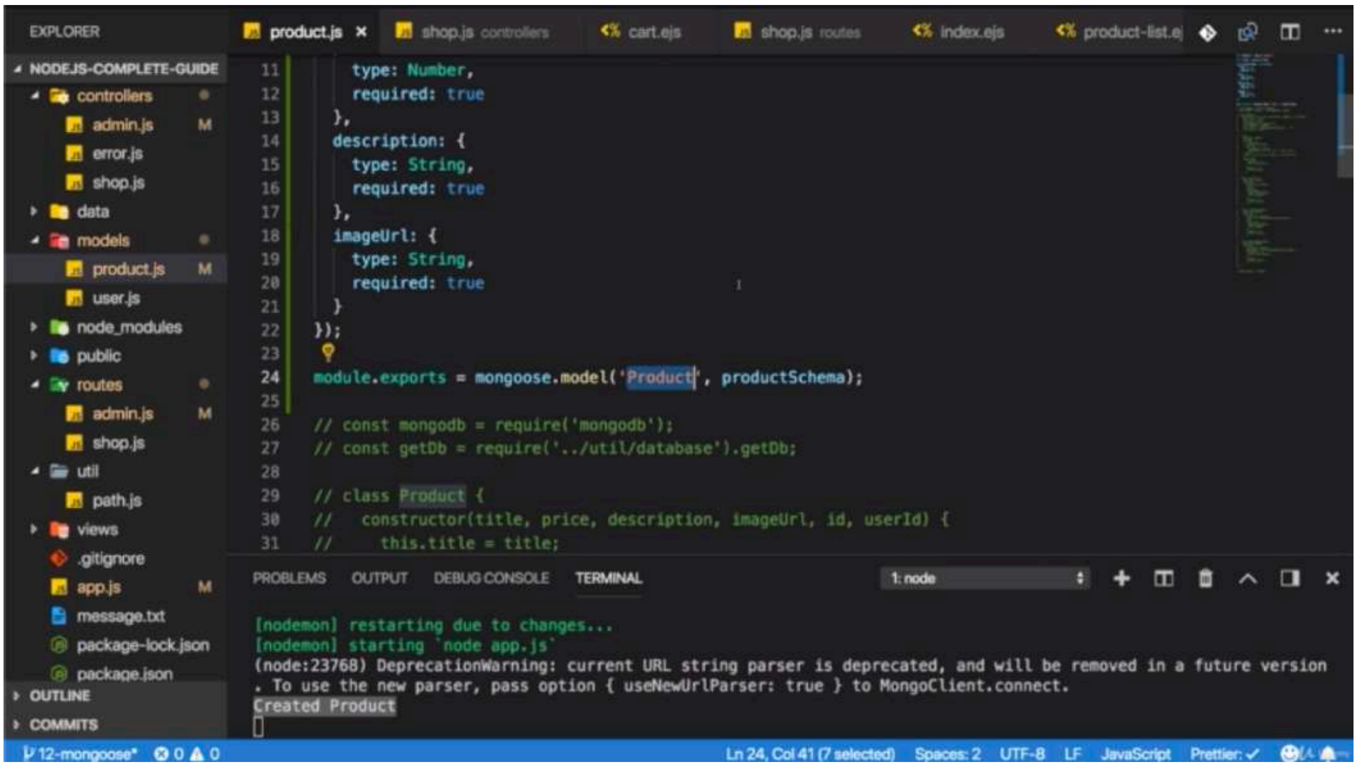
```
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:23768) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Created Product
```

- in the code, i got no error and created product which looks good
![](images/209-saving-data-through-mongoose-4.png)



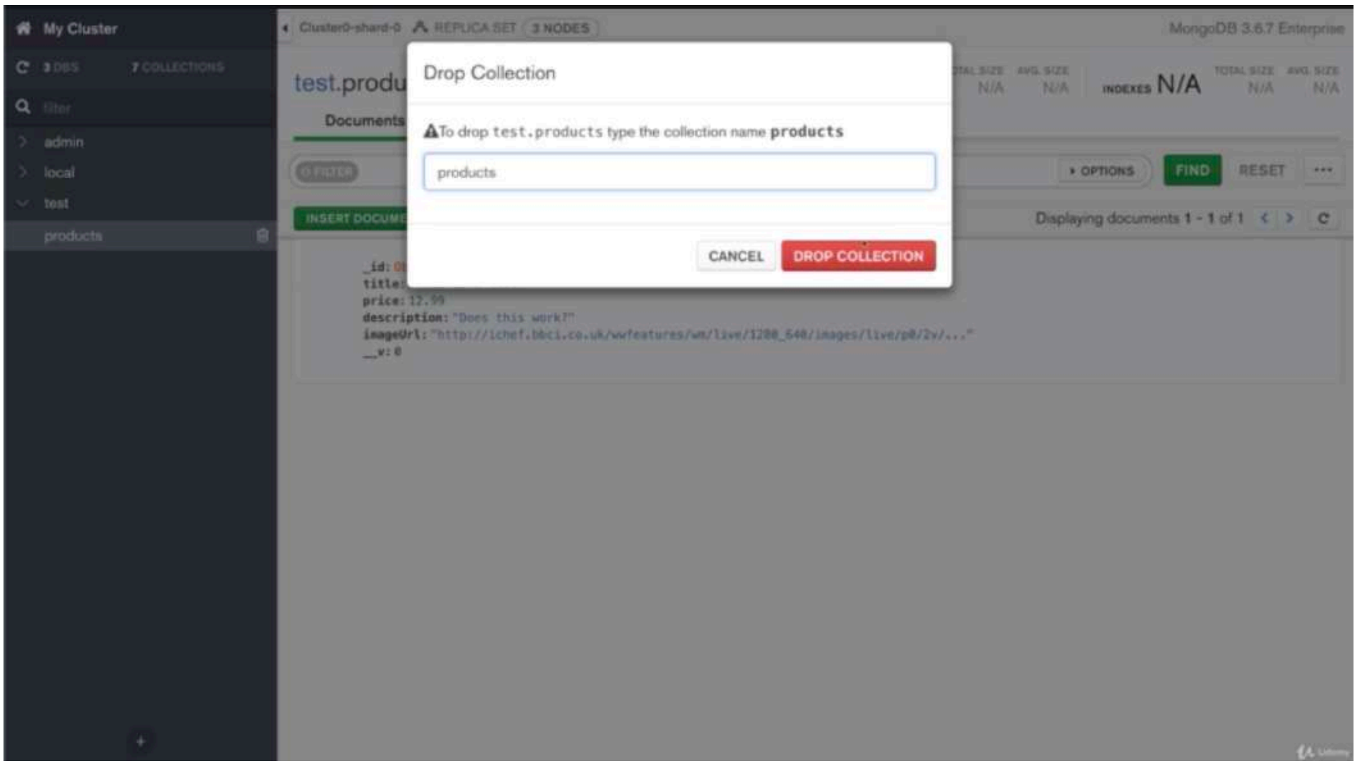
```
_id: ObjectId("5bab2bf6f399b35cd859f947")
title: "This is a test!"
price: 12.99
description: "Does this work?"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
__v: 0
```

- and in MongoDB Compass, let's refresh.
- i connected to the wrong url, i will fix that later.
- i'm connected to the 'test' database instead of the shop database. theoretically it worked. we get a product collection with the product added.
![](images/209-saving-data-through-mongoose-5.png)



```
    type: Number,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  imageUrl: {
    type: String,
    required: true
  }
});

module.exports = mongoose.model('Product', productSchema);

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

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

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

- where is the products collection coming from? we never defined that name.
- Mongoose takes your model name, so ‘Product’, turns it to all lowercase and takes the plural form of that. that will then be used as a collection name.
![](images/209-saving-data-through-mongoose-6.png)


Drop Collection

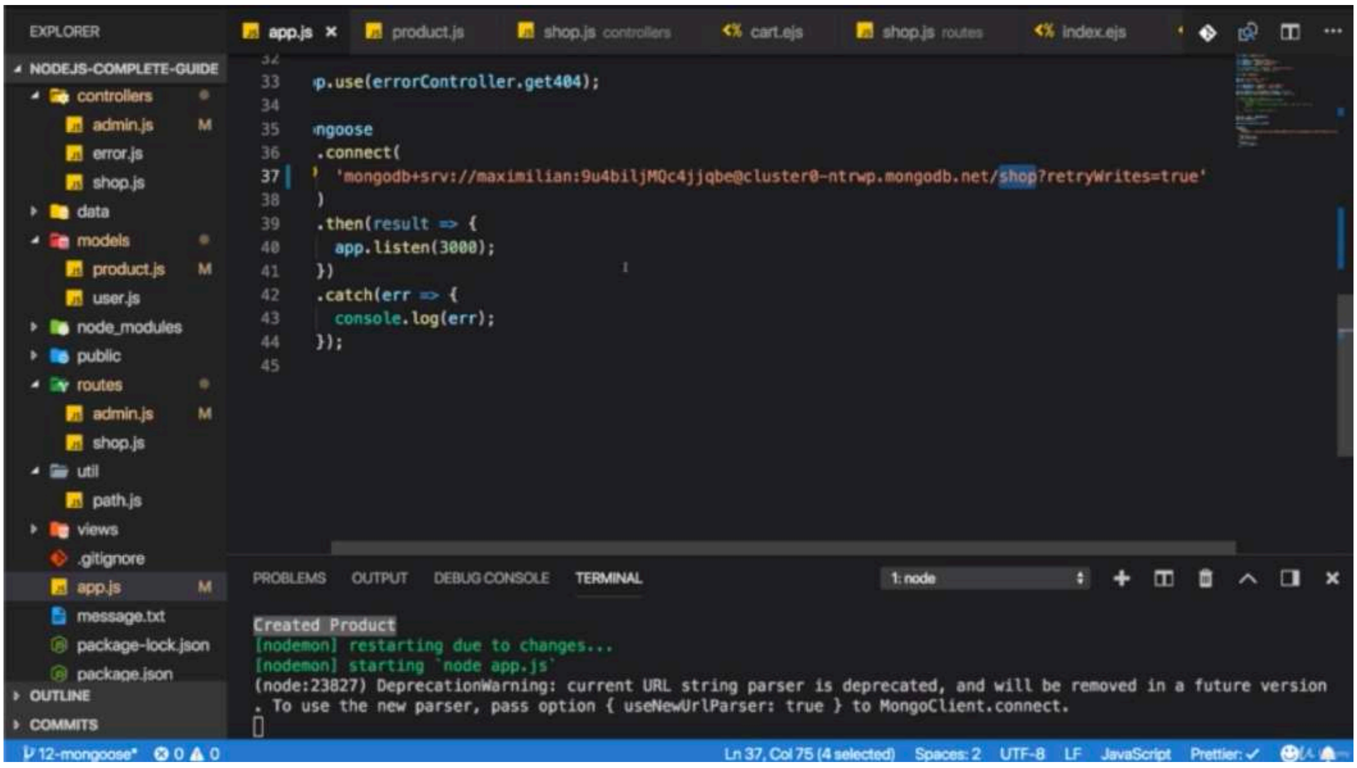
To drop test.products type the collection name **products**

products

CANCEL DROP COLLECTION

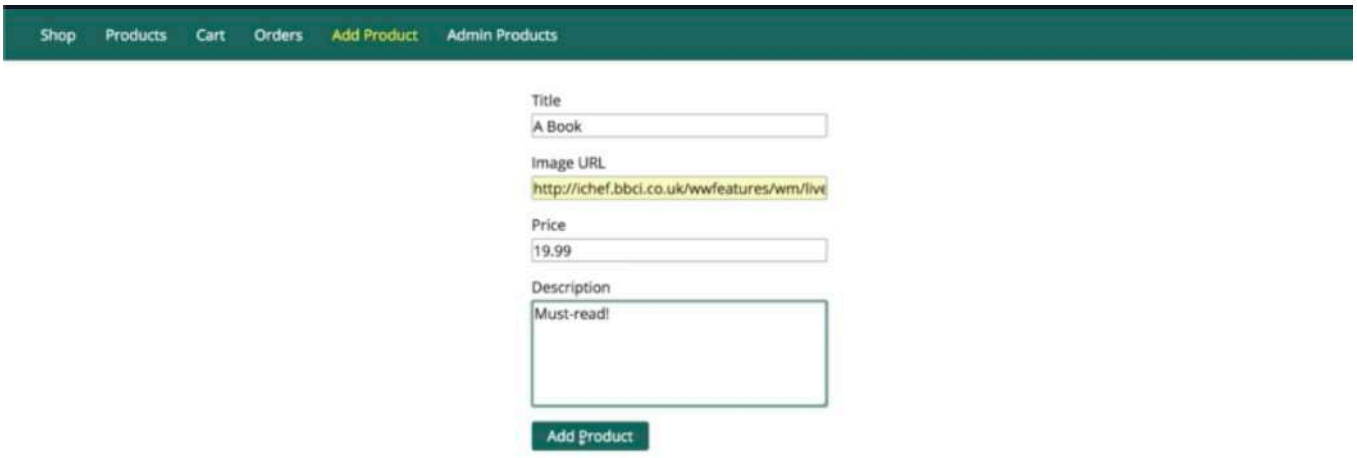
```
_id:
title:
price: 12.99
description: "Does this work?"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
__v: 0
```

- i will still drop this collection and quickly fix my connection setting in app.js file
![](images/209-saving-data-through-mongoose-7.png)
![](images/209-saving-data-through-mongoose-8.png)
![](images/209-saving-data-through-mongoose-9.png)
![](images/209-saving-data-through-mongoose-10.png)



```
p.use(errorController.get404);

ngoose
  .connect(
    'mongodb+srv://maximilian:9u4biljMQc4jjqbe@cluster0-ntrwp.mongodb.net/shop?retryWrites=true'
  )
  .then(result => {
    app.listen(3000);
  })
  .catch(err => {
    console.log(err);
  });
```

```
Created Product
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:23827) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
```

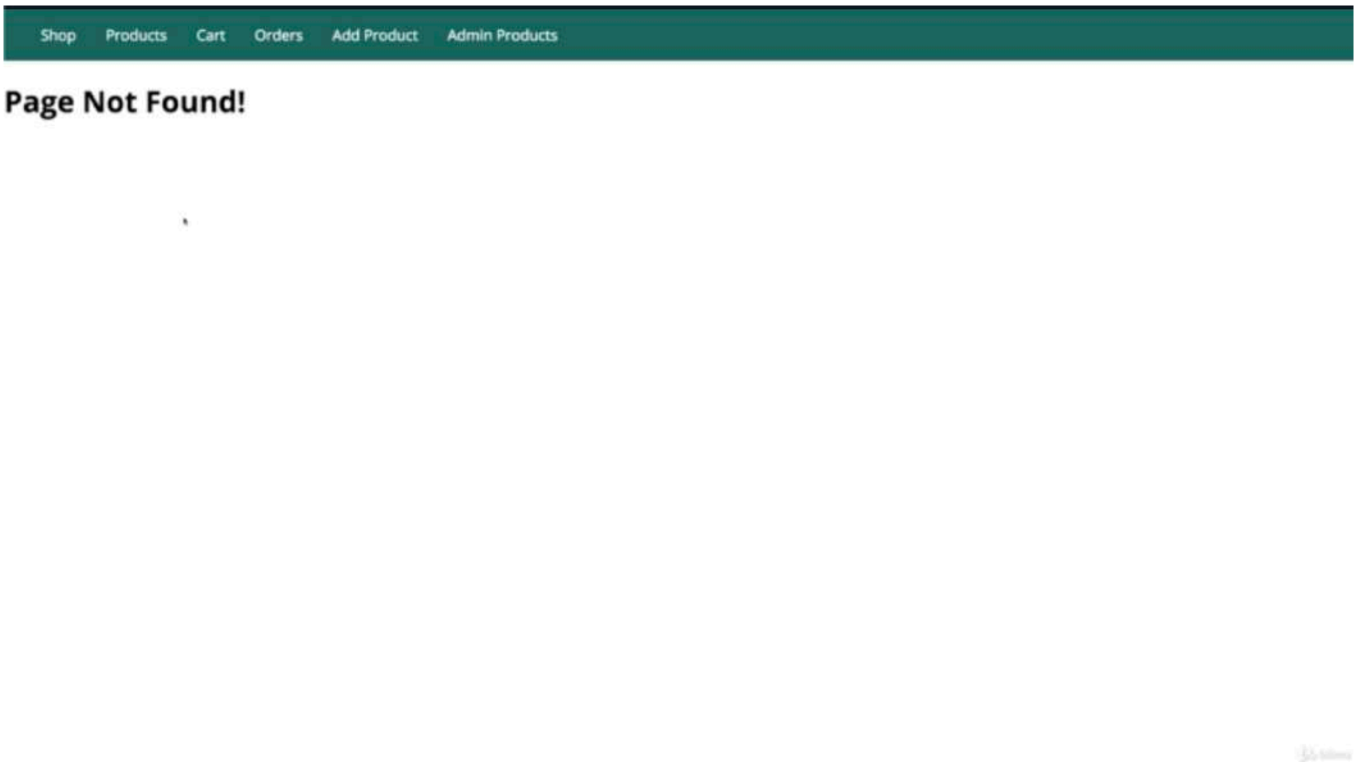


Title: A Book

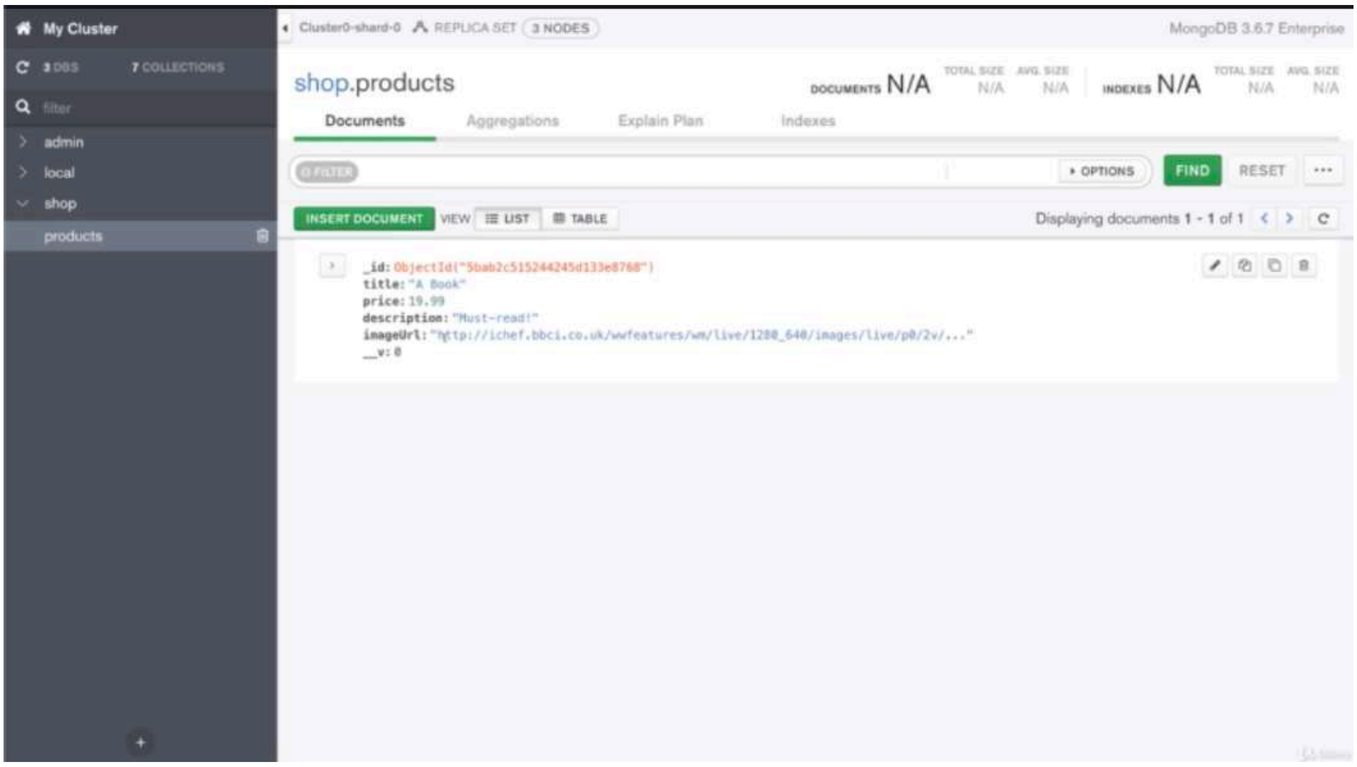
Image URL: http://ichef.bbci.co.uk/wwfeatures/wm/live

Price: 19.99

Description: Must-read!



Page Not Found!



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

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema;

const productSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  imageUrl: {
    type: String,
    required: true
  }
})

/**Mongoose also works with so-called models
 * and the model is also what we will export here.
 *
 * 'model()' is a function which is important for mongoose behind the scenes to connect a schema with a name.
 * so here you give that model a name
 * and that name would be 'Product'
 *
 * 2nd argument is the schema
 * so in my case 'productSchema' we define.
 */
module.exports = mongoose.model('Product', productSchema)
```

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

/**we still import product form our ./models/product.js
 * because i export a model.
 * and we can use that in the way i used it here.
 */
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({
    /**order-matching with ./models/product.js don't matter  */
    title: title,
    price: price,
    description: description,
    imageUrl: imageUrl
  })
  /**now 'product' here is managed by Mongoose
   * and 'product' happens to have a 'save()' method provided by Mongoose
   * 'save()' is not defined by us.
   * we defined 'save()' before but now is not defined by us.
   *
   * we don't get a promise
   * but mongoose still gives us a 'then()' method
   *
   * it also still gives us a 'catch()' method
   * and therefore this code should continue to work.
   */
  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));
};
```

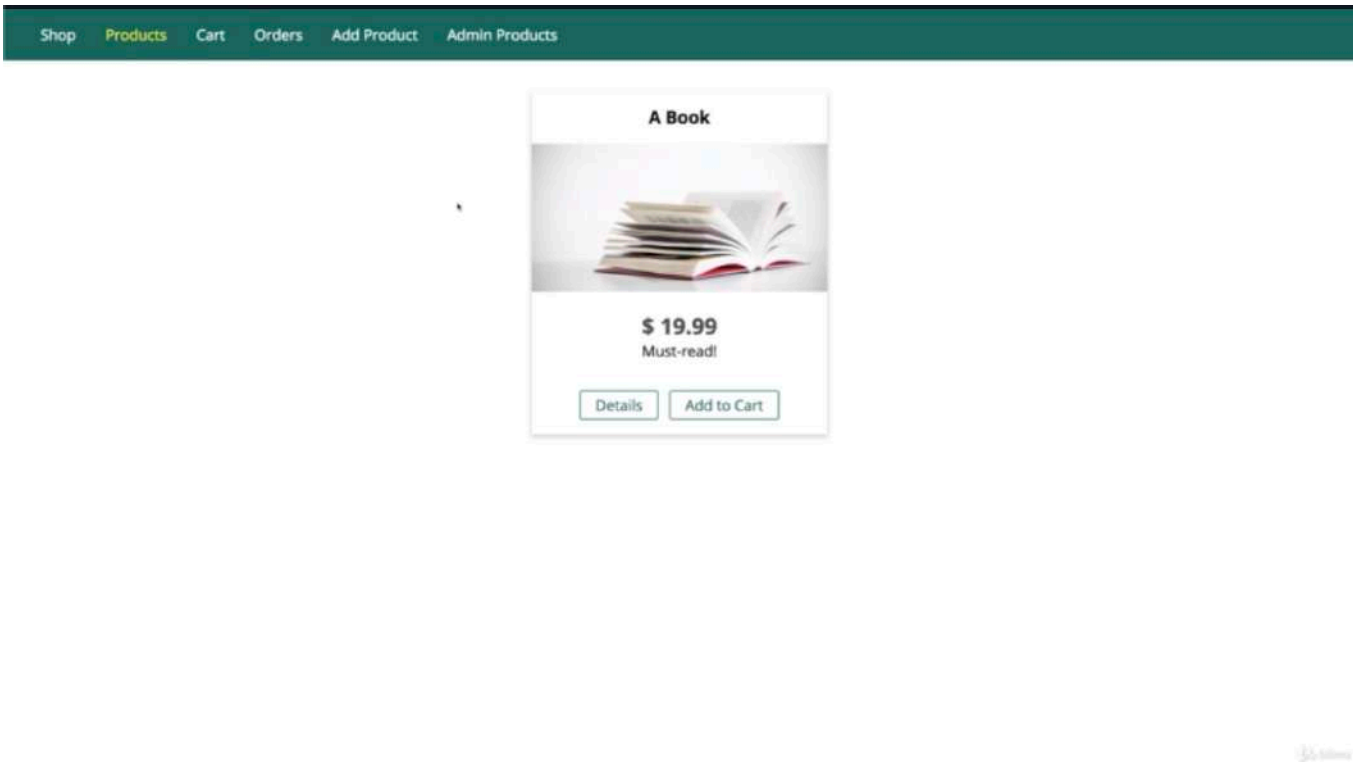
```
//app.js

const path = require('path');

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

const errorController = require('./controllers/error');
//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) => {
  User.findById('5cb7d12855fbe74b129c0b7c')
    .then(user => {
      req.user = new User(user.namem, user.email, user.cart, user._id);
      next();
    })
    .catch(err => console.log(err));
});
*/

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

app.use(errorController.get404);

/**we already have everything in place we need to connect
 * and mongoose will manage that one connection behind the scenes.
 * so taht in other places where we start using mongoose from the mongoose package,
 * we use that same connection we set up here.
 */
mongoose
  .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/shop?retryWrites=true')
  .then(result => {
    app.listen(3000)
  })
  .catch(err => {
    console.log(err)
  })
```

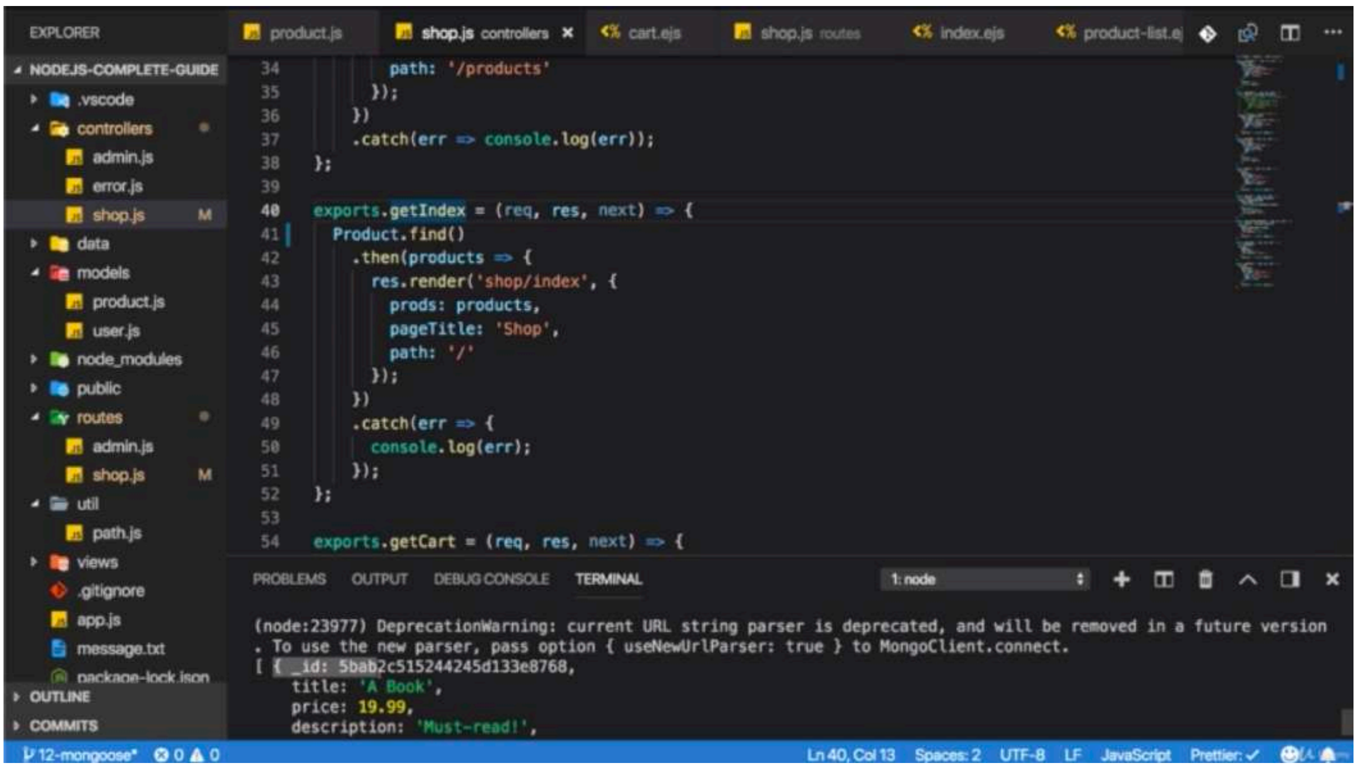
```
// ./routes/admin.js

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 210: Fetching All Products

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

![](images/210-fetching-all-products-1.png)
![](images/210-fetching-all-products-2.png)





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

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

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

```
(node:23977) 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.
[ { _id: 5bab2c515244245d133e8768,
    title: 'A Book',
    price: 19.99,
    description: 'Must-read!',
```

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

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

```

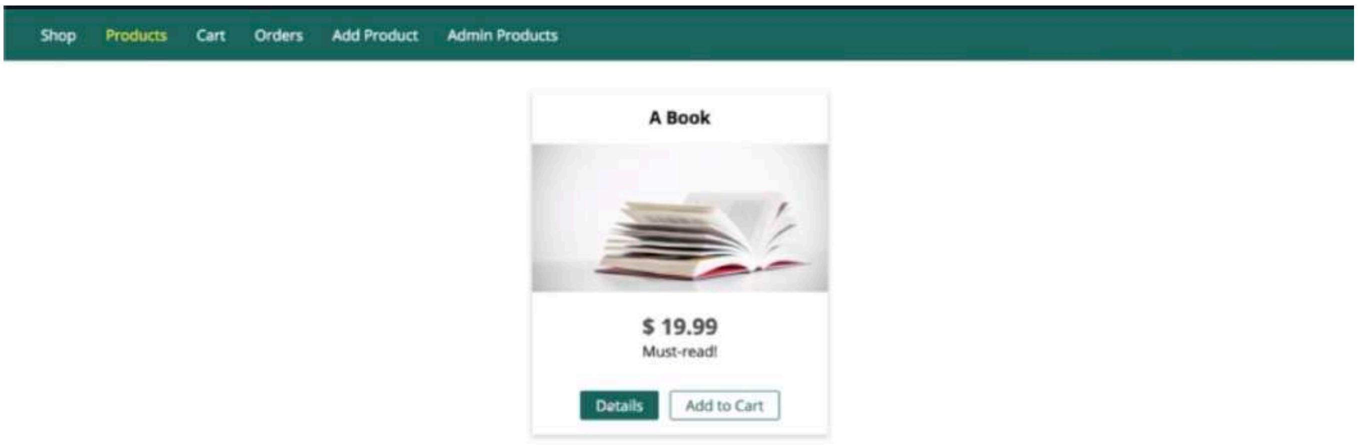
```
const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
  /**'find()' method works a bit differently when used with mongoose
   * it doesn't give us a cursor.
   * it does give us the products,
   * we could add '.cursor'
   * and call this to get access to the cursor
   * and then use each async which would allow us to loop through them
   * or 'next()' to get the next element.
   * but i will just use 'find()'
   * and this will essentially give me all my products automatically
   */
  Product.find()
    .then(products => {
      console.log(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.find()
    .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));
};

```

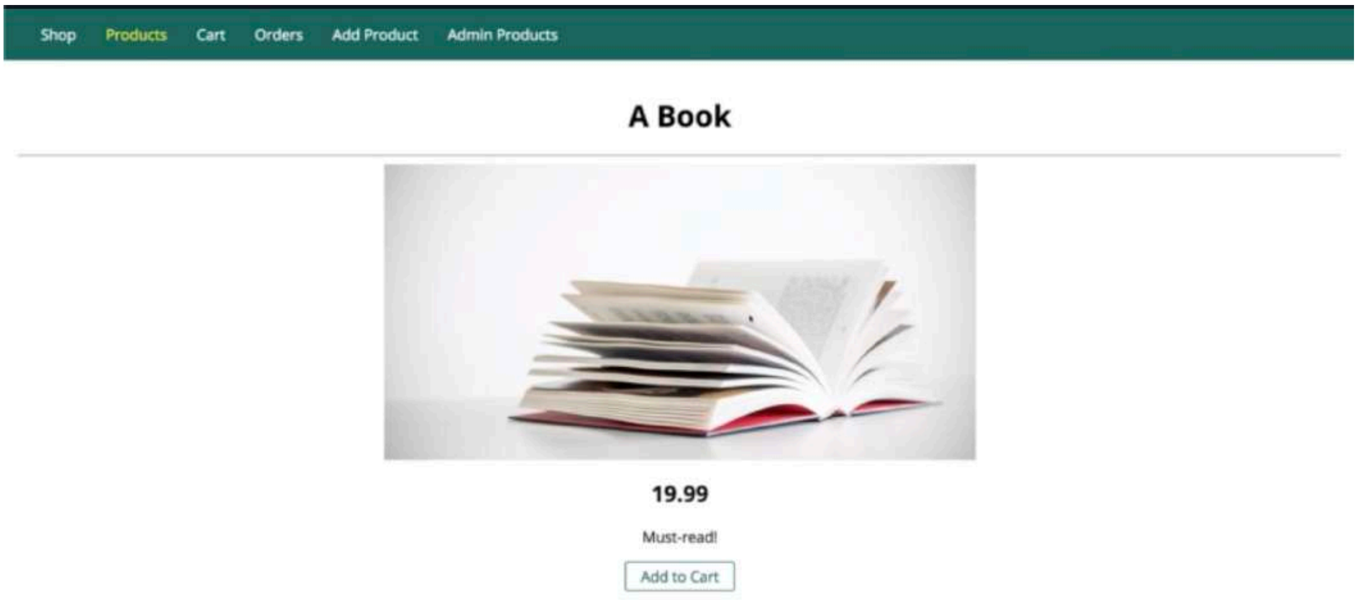
```
// ./routes/shop.js

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;

```

* Chapter 211: Fetching A Single Product

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

![](images/211-fetching-a-single-product-1.png)
![](images/211-fetching-a-single-product-2.png)





```javascript
// ./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;

```

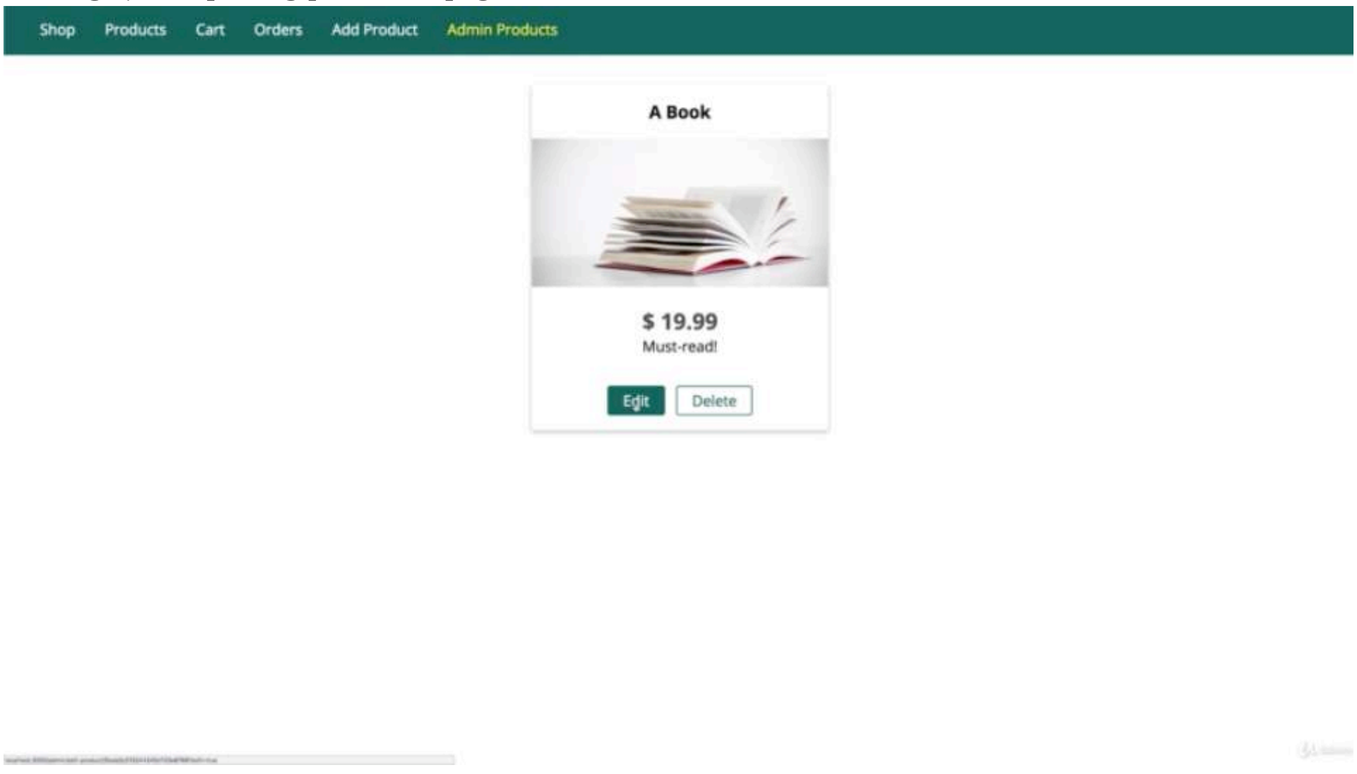
```javascript
//./controllers/shop.js

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

exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      console.log(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;
  /**mongoose has a 'findById()' method
   * so little convenience method that defines for us.
   * so again 'findById()' method is not our own method,
   * it's defined by Mongoose
   *
   * and even pass a string to 'findById()'
   * and mongoose will automatically convert this to an objectId
   * so it will handle that for us as well.
  */
  Product.findById(prodId)
    .then(product => {
      res.render('shop/product-detail', {
        product: product,
        pageTitle: product.title,
        path: '/products'
      });
    })
    .catch(err => console.log(err));
};
```

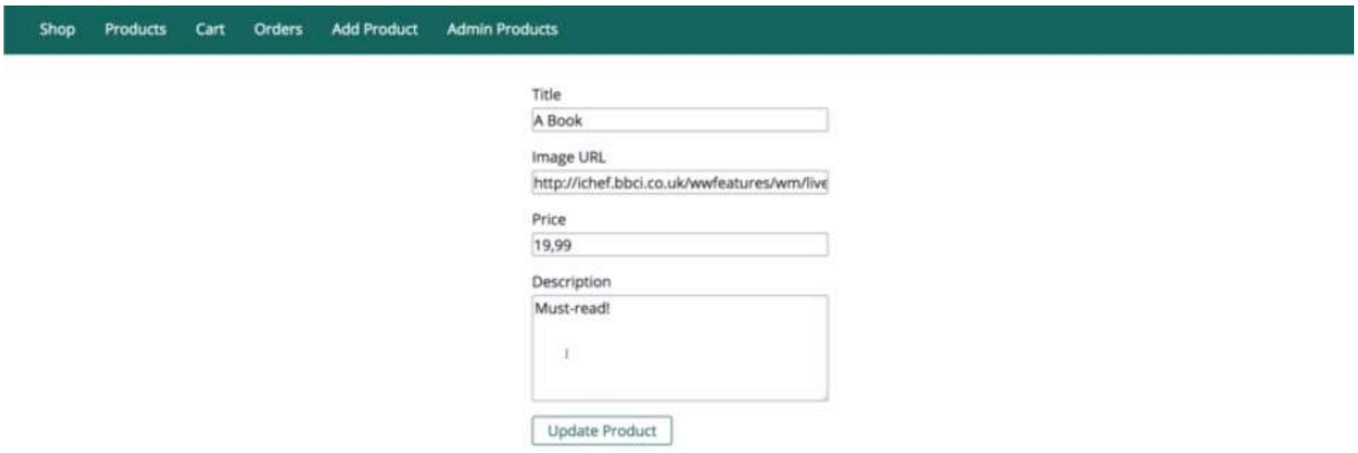
```javascript

exports.getIndex = (req, res, next) => {
  Product.find()
    .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));
};

```

* Chapter 212: Updating Products

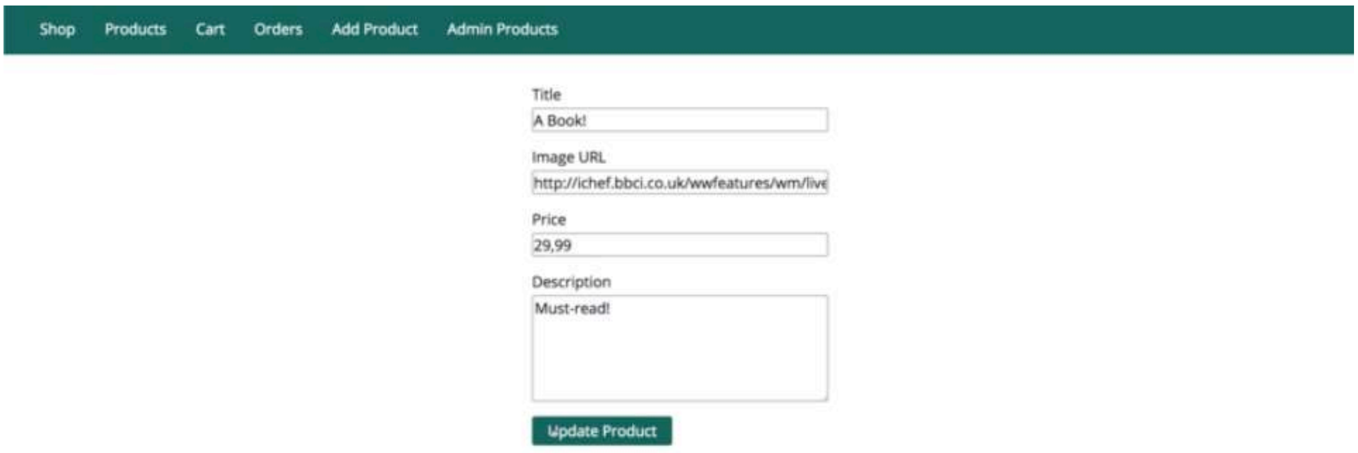
1. update
- ./controllers/admin.js
- ./routes/admin.js

![](images/212-updating-products-1.png)
![](images/212-updating-products-2.png)
![](images/212-updating-products-3.png)
![](images/212-updating-products-4.png)
![](images/212-updating-products-5.png)

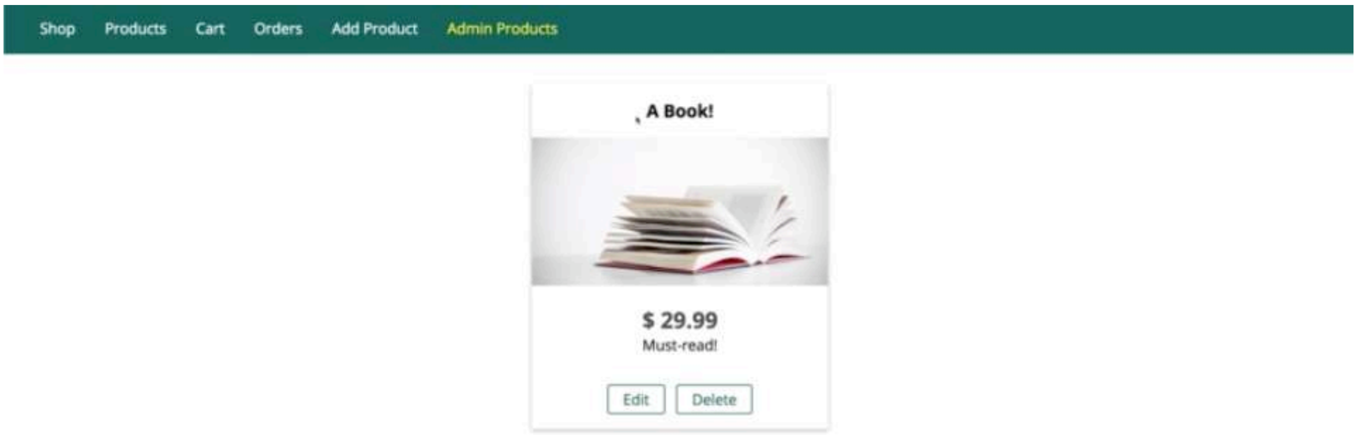


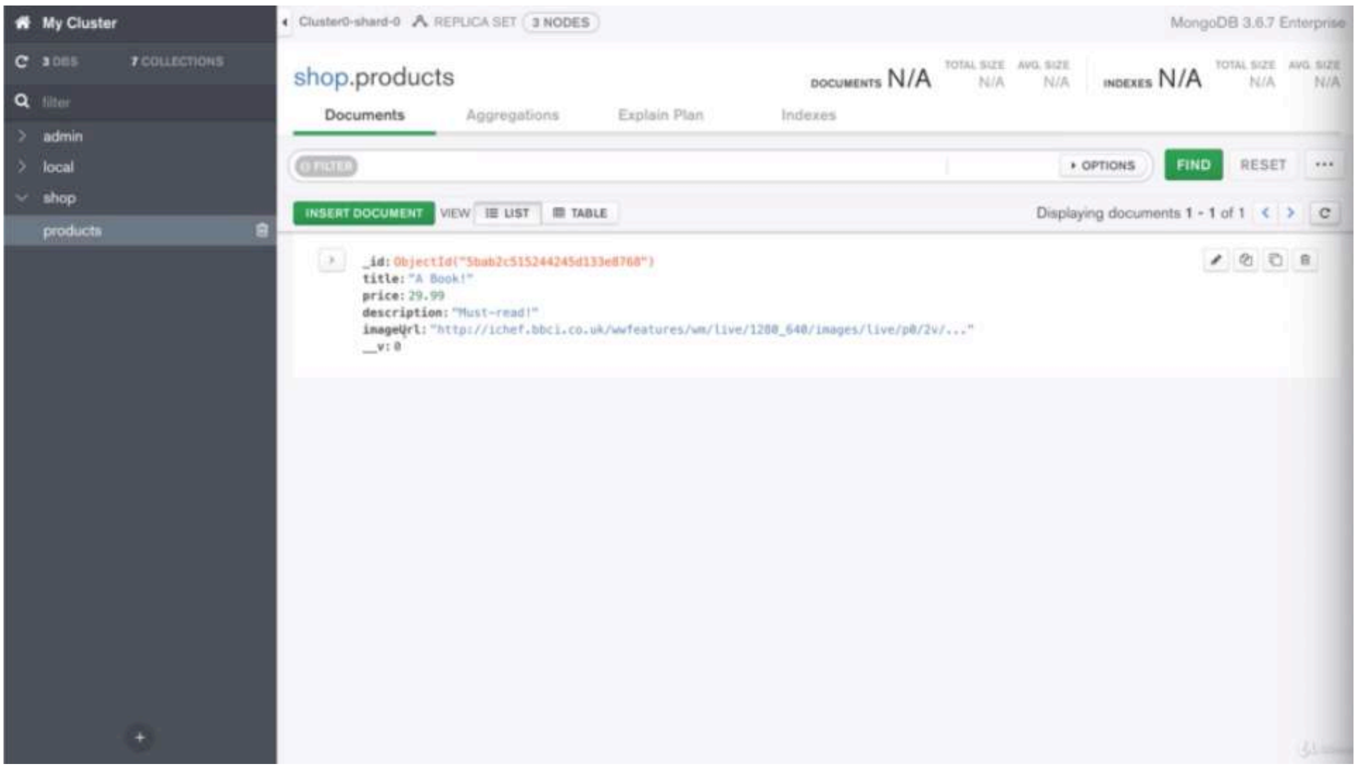


Title
A Book
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
19,99
Description
Must-read!



Title
A Book!
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
29,99
Description
Must-read!





```
_id: ObjectId("5bab2c515244245d133e8768")
title: "A Book!"
price: 29.99
description: "Must-read!"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
__v: 0
```

```javascript
// ./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: title,
    price: price,
    description: description,
    imageUrl: 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)
    .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;
/**i first of all find the product
 * and i get back a full mongoose object
 * hence i can manipulate it and call save again
 * i return the result of that
 * and then call 'then()' on that to redirect once the saving was done.
 */
  Product
    .findById(prodId)
```

```javascript
    .then(product => {
      product.title = updatedTitle
      product.price = updatedPrice
      product.description = updatedDesc
      product.imageUrl = updatedImageUrl
      /**i can move 'product.save()' into my function
       * and call 'save()' on the product that was fetched from the database
       * because thanks to mongoose,
       * this will now not be a javascript object with the data
       * but we will have a full mongoose object here with all the mongoose methods like
'save()'
       * if we call 'save()' on an existing object,
       * it will not be saved as a new one,
       * but the changes will be saved.
       * so it will automatically do an update behind the scenes.
       */
      return product
      .save()
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.find()
    .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));
};
```

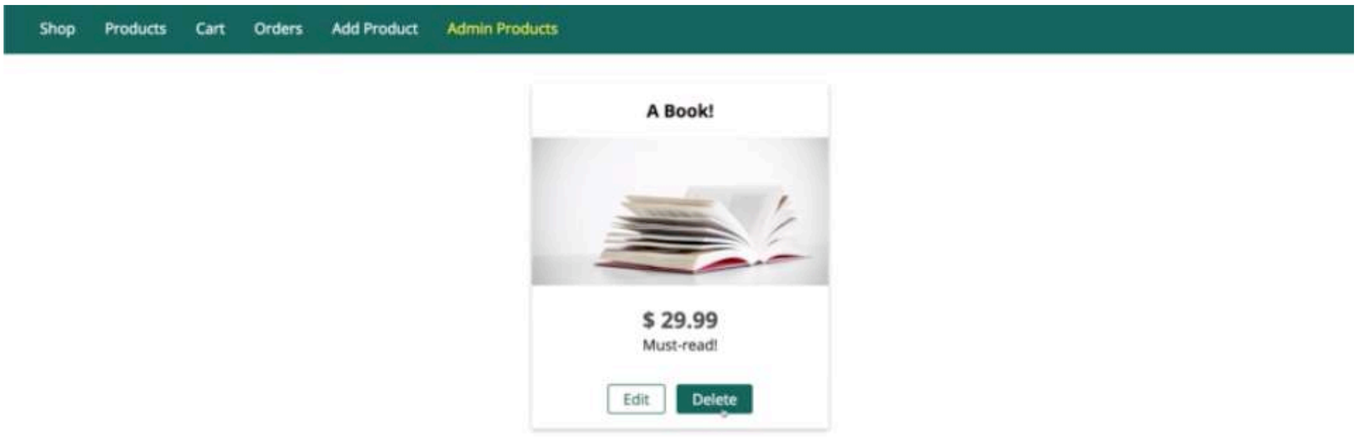
```javascript
// ./routes/admin.js

const path = require('path');

const express = require('express');

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

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

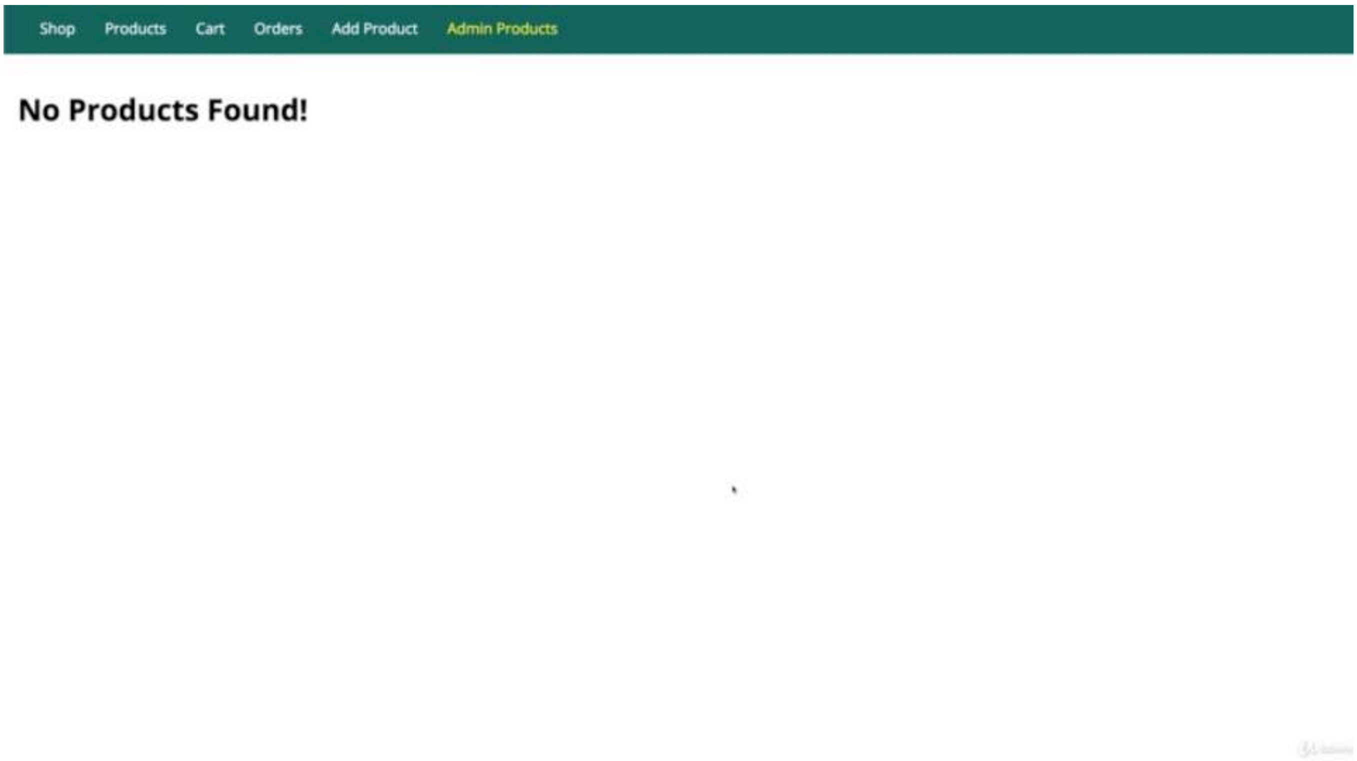
```

// /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 213: Deleting Products

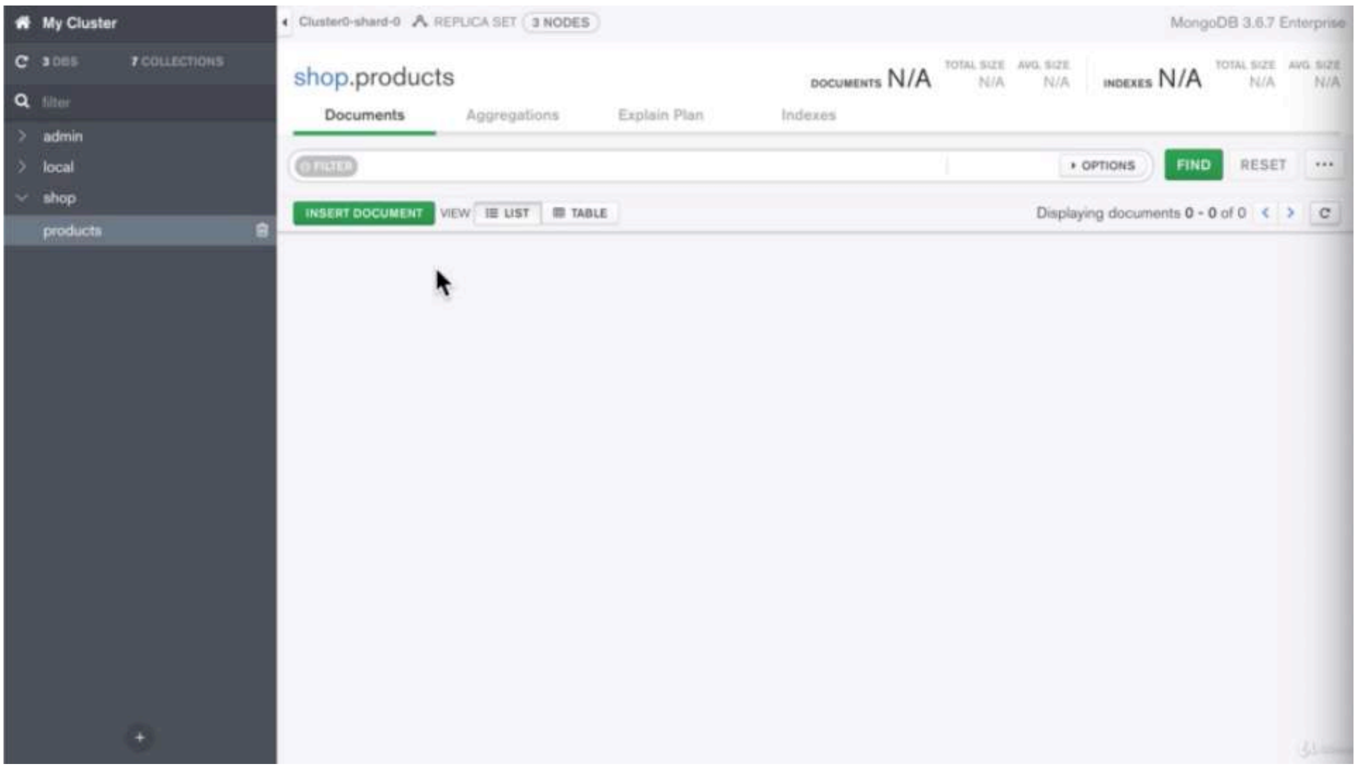
1. update
- ./controllers/admin.js
- ./routes/admin.js

![](images/213-deleting-products-1.png)
![](images/213-deleting-products-2.png)
![](images/213-deleting-products-3.png)





No Products Found!



```javascript
// ./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: title,
    price: price,
    description: description,
    imageUrl: 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)
    .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;

  Product
    .findById(prodId)
    .then(product => {
      product.title = updatedTitle
      product.price = updatedPrice
      product.description = updatedDesc
      product.imageUrl = updatedImageUrl
```

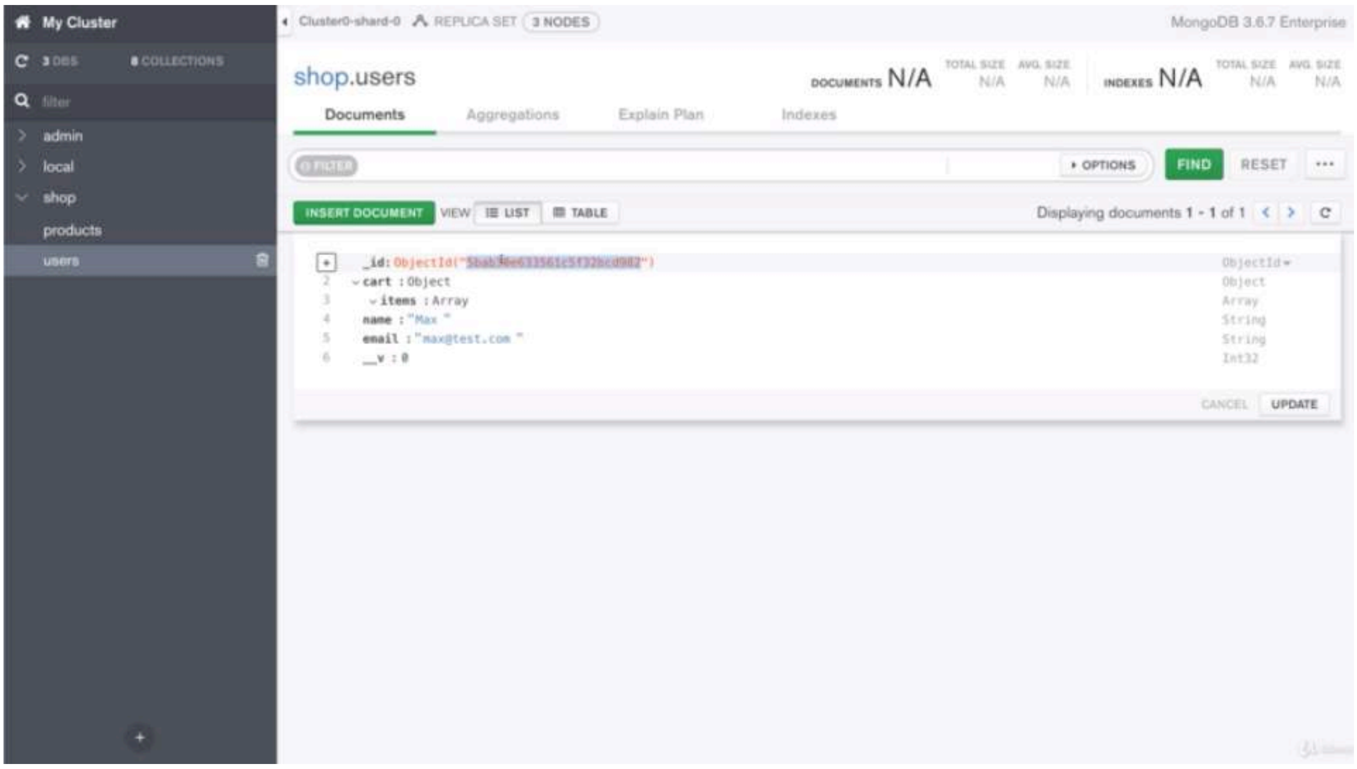
```javascript
      return product
      .save()
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.find()
    .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;
  /**'findByIdAndRemove()' is a built-in method provided by mongoose that should remove a document
   */
  Product.findByIdAndRemove(prodId)
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
```

```javascript
// ./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;
```

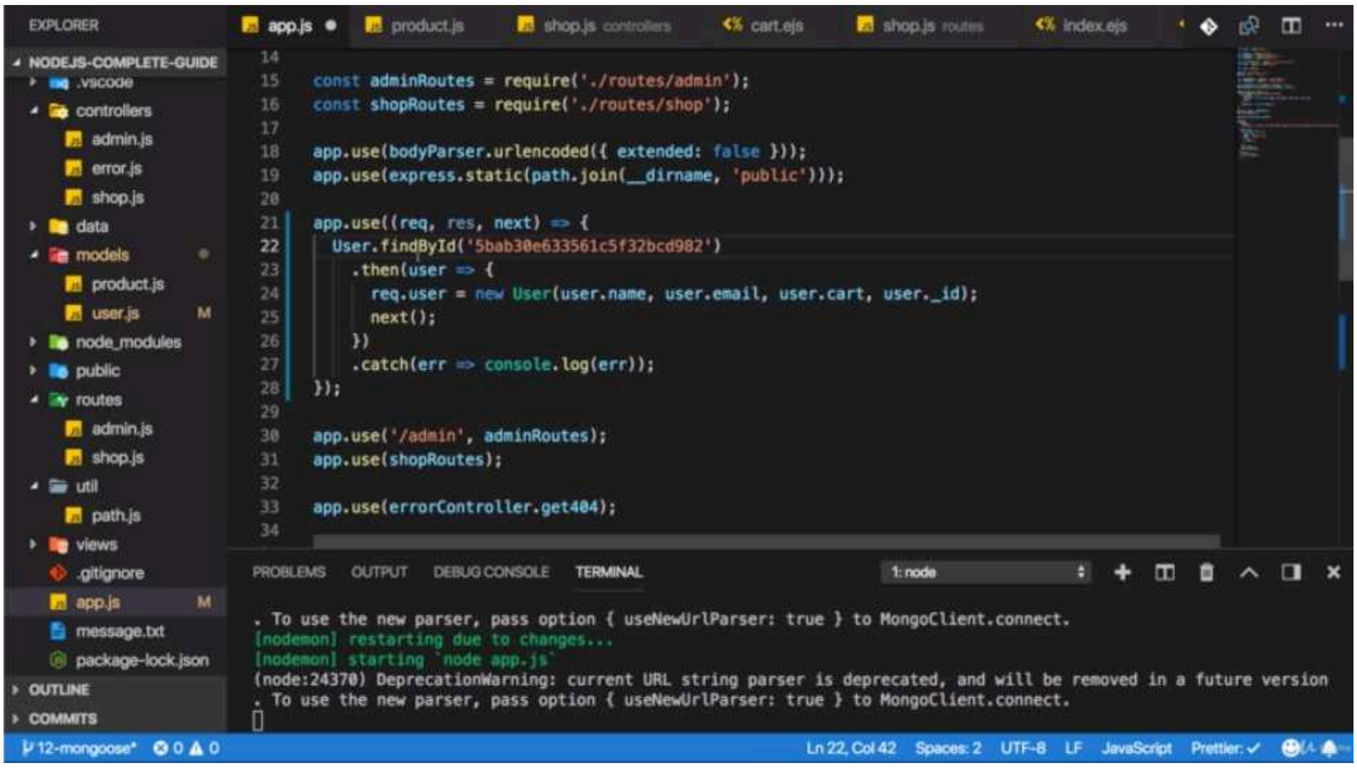
* Chapter 214: Adding And Using A User Model

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

![](images/214-adding-and-using-a-user-model-1.png)
![](images/214-adding-and-using-a-user-model-2.png)



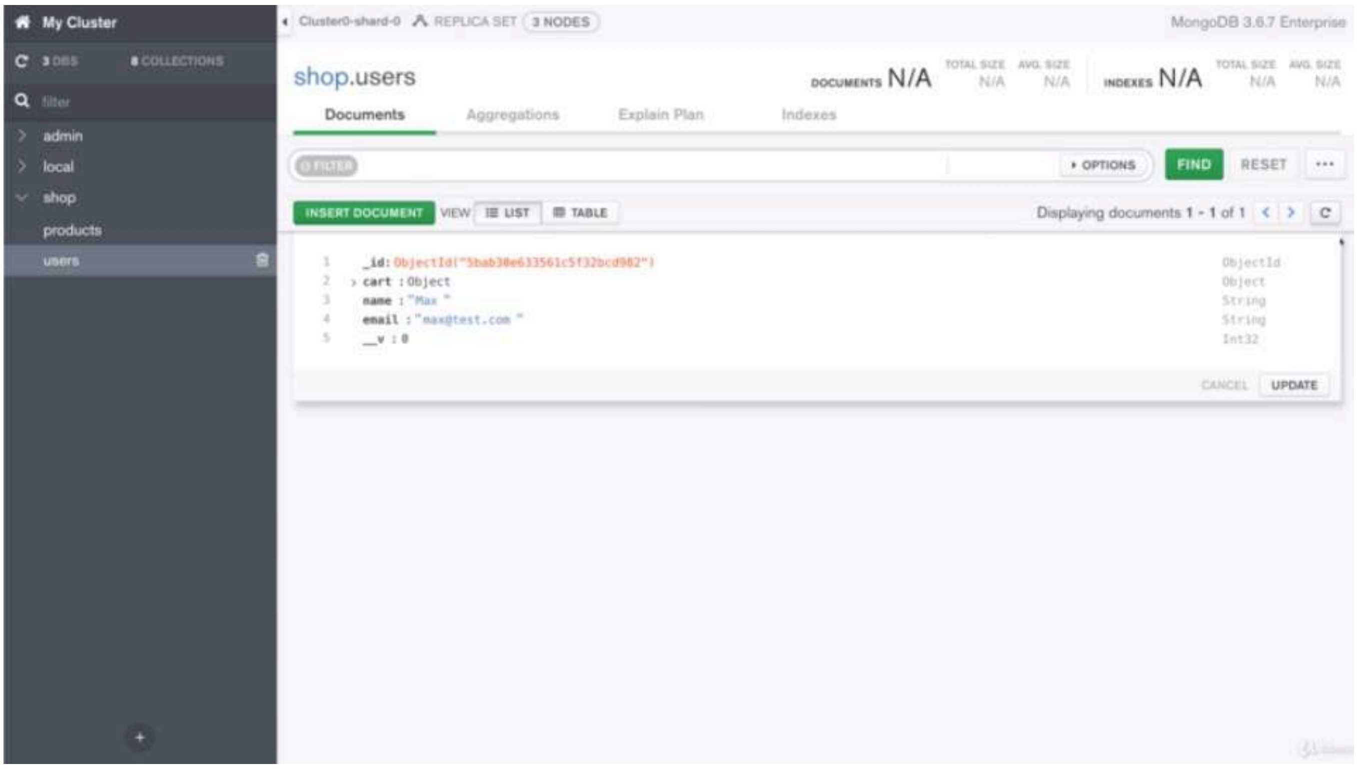
```
_id: ObjectId("5bab30e633561c5f32bcd982")     ObjectId
cart : Object                                  Object
  items : Array                                Array
name : "Max "                                  String
email : "max@test.com "                        String
__v : 0                                        Int32
```



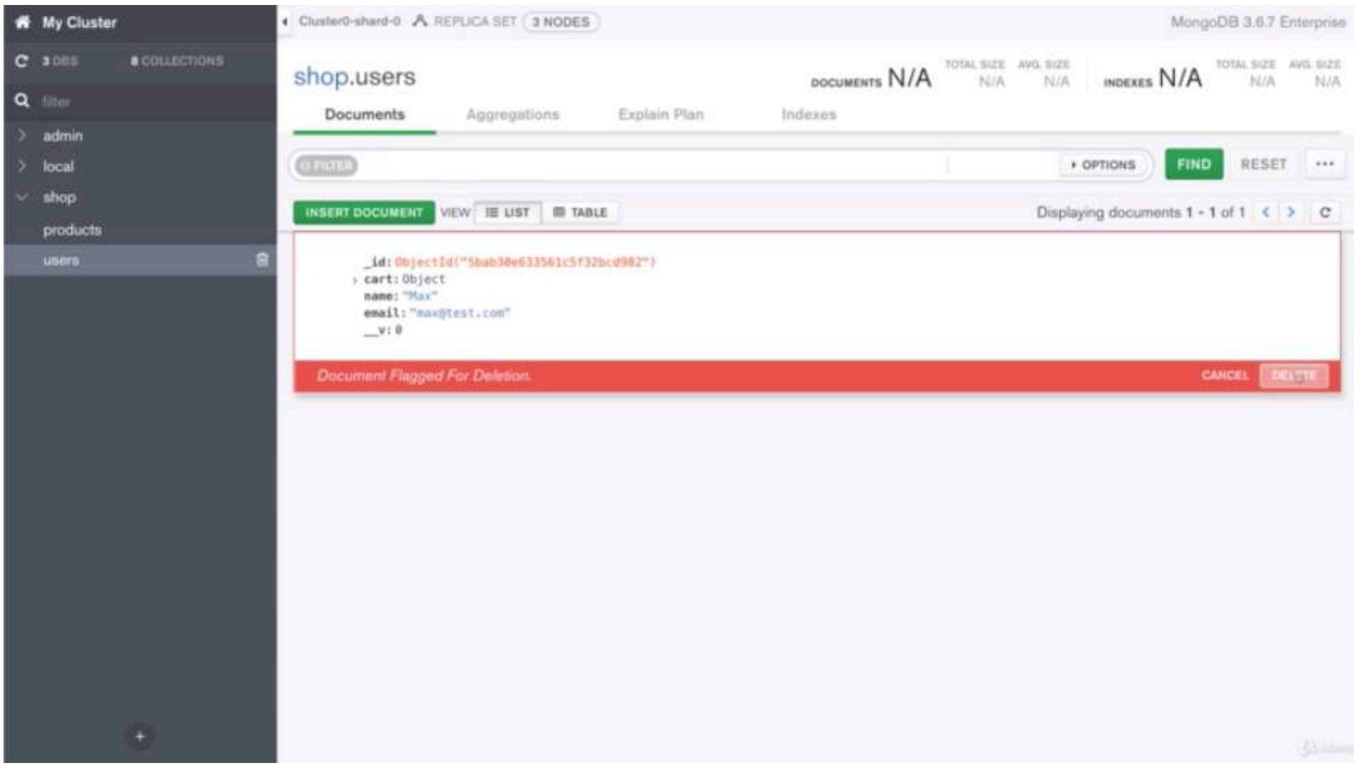
```

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('5bab30e633561c5f32bcd982')
    .then(user => {
      req.user = new User(user.name, user.email, user.cart, user._id);
      next();
    })
    .catch(err => console.log(err));
});

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

app.use(errorController.get404);

```

```
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:24370) 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.
```

- with that _id, let me copy that ID and go back to the app.js file and comment this middleware back in here.
- i just need to paste in that ID of that user we just created. 'findById()' is a method provided by mongoose so this will work.
![](images/214-adding-and-using-a-user-model-3.png)
![](images/214-adding-and-using-a-user-model-4.png)
![](images/214-adding-and-using-a-user-model-5.png)
![](images/214-adding-and-using-a-user-model-6.png)
![](images/214-adding-and-using-a-user-model-7.png)

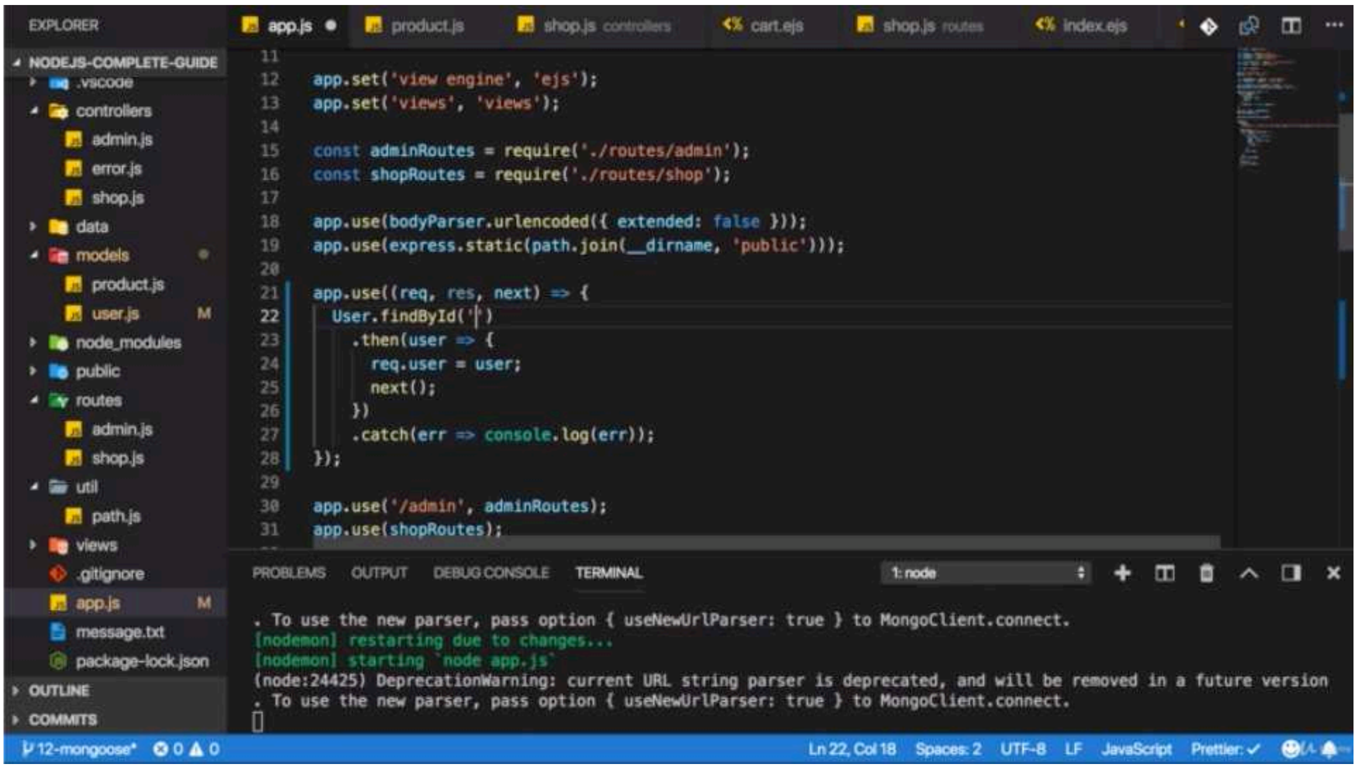


```
_id: ObjectId("5bab30e633561c5f32bcd982")    ObjectId
> cart :Object                               Object
name :"Max "                                 String
email :"max@test.com "                       String
__v :0                                       Int32
```



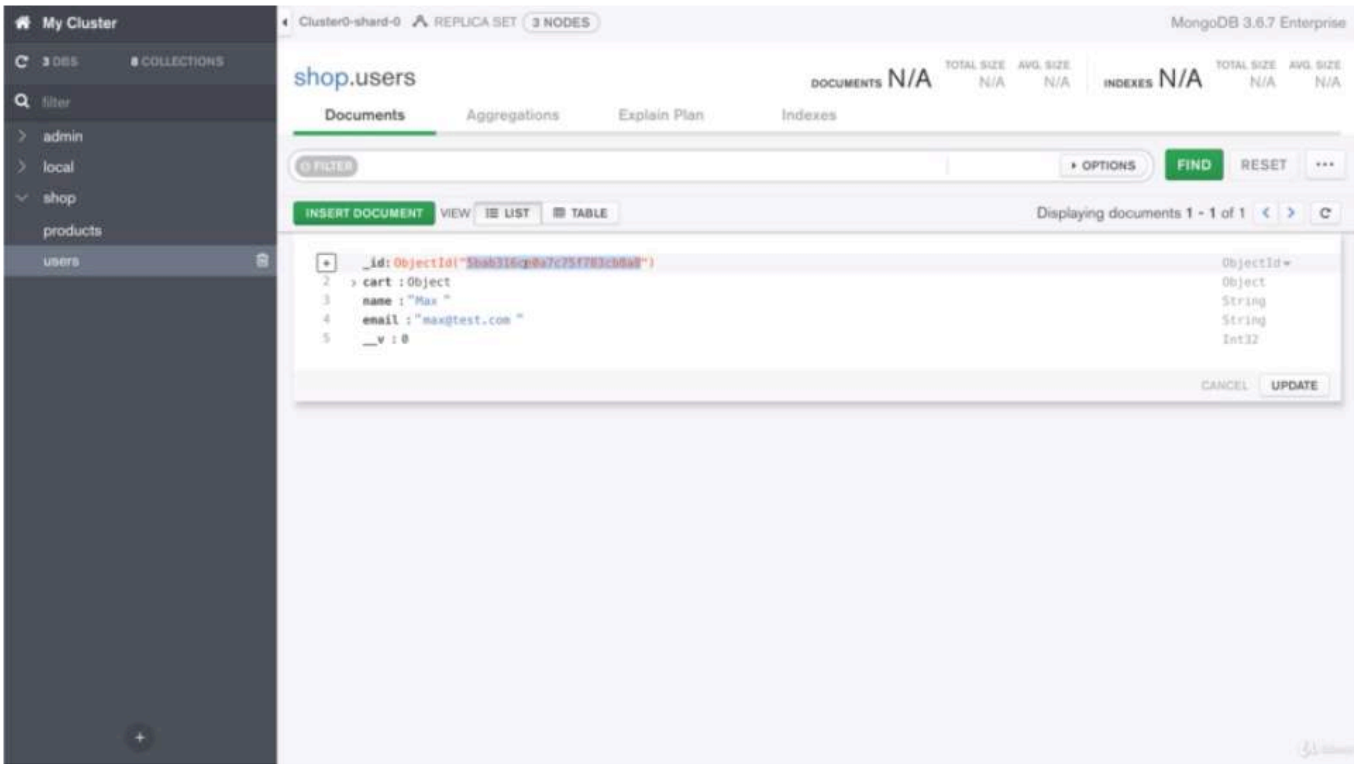
```
_id: ObjectId("5bab30e633561c5f32bcd982")
> cart: Object
name: "Max"
email: "max@test.com"
__v: 0
```

Document Flagged For Deletion.

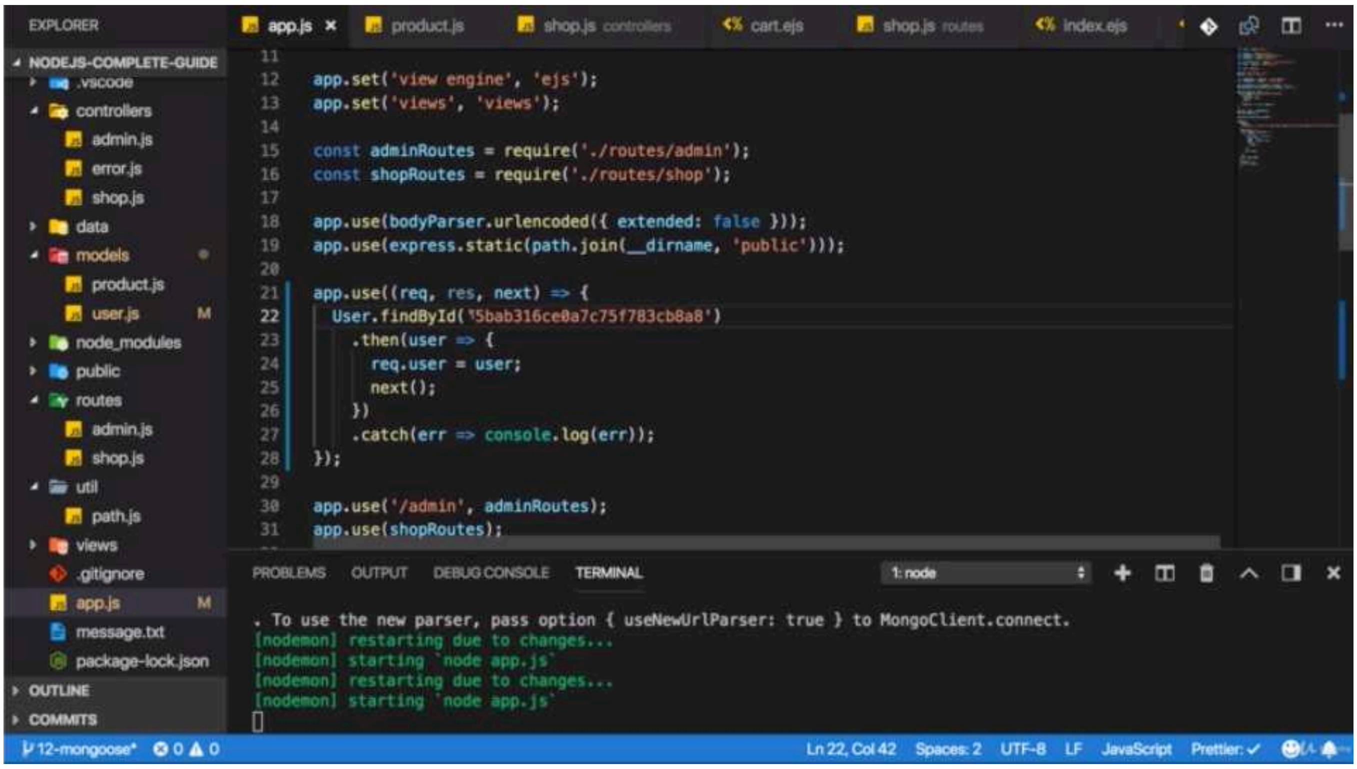


```
app.set('view engine', 'ejs');
app.set('views', 'views');

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('')
    .then(user => {
      req.user = user;
      next();
    })
    .catch(err => console.log(err));
});

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

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



```
_id: ObjectId("5bab316e90a7c75f783cb8a8")      ObjectId
> cart : Object                                 Object
name : "Max "                                   String
email : "max@test.com "                         String
__v : 0                                         Int32
```



```javascript

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('5bab316ce0a7c75f783cb8a8')
    .then(user => {
      req.user = user;
      next();
    })
    .catch(err => console.log(err));
});

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

```
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
```

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  cart: {
    /**i wanna have an array of documents where i have a productId
     * which i will configure with this document.
     *
     * we have a 'Types' field
     * and there we got all these special types like ObjectId
     * so i'm telling mongoose that
     * this will actually store an objectId
     * because it will store a reference to a product
     */
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }]
  }
})

/**i will export this by calling mongoose model,
 * give this a name and the name will be 'User'
 * hence this will be stored in a 'Users'
 * because mongoose will automatically take the plural lowercase version of that as a collection name
 */
module.exports = mongoose.model('User', userSchema)
```

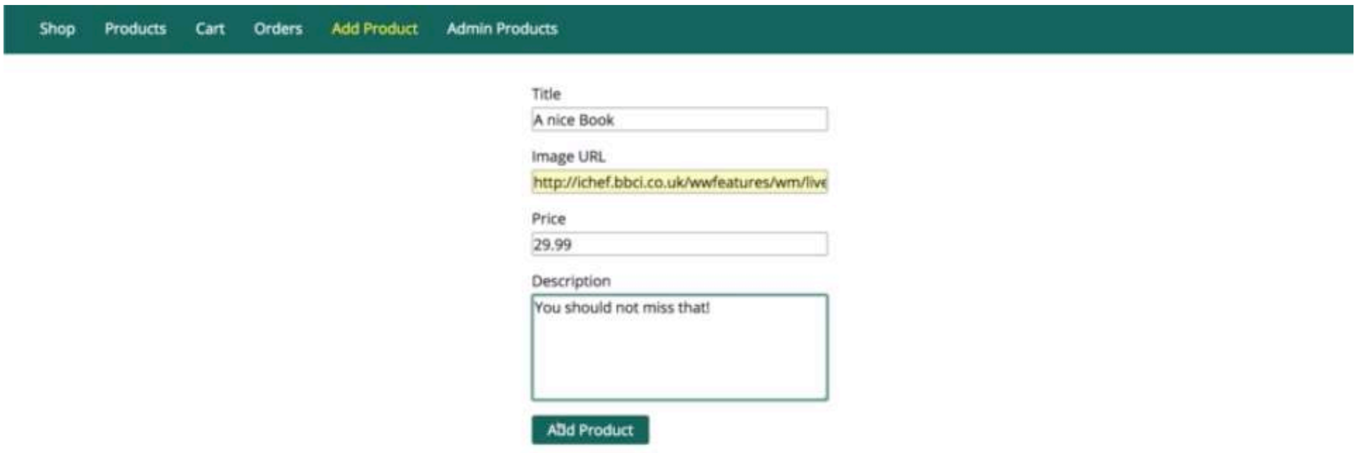
```
//app.js

const path = require('path');

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

const errorController = require('./controllers/error');
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) => {
  User.findById('5cbb2b2c80bd7193adb9eeeb')
    .then(user => {
      /**i can store that user in my request
       * and keep in mind, 'user' on right side is a full mongoose model
       * so we can call all these mongoose model functions or methods on that user object
       * and therefore also on the user object which i store here.
       */
      req.user = user
      next();
    })
    .catch(err => console.log(err));
});

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

app.use(errorController.get404);

mongoose
  .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3vlk.mongodb.net/shop?retryWrites=true')
  .then(result => {
```

```
  /**if i give 'findOne()' no arguments,
   * it will always give me back the first user it finds
   */
  User
    .findOne()
    .then(user => {
      if(!user){
        const user = new User({
          name: 'Max',
          email: 'max@test.com',
          cart: {
            items: []
          }
        })
        user.save()
      }
    })
  app.listen(3000)
})
.catch(err => {
  console.log(err)
})
```

* Chapter 215: Using Relations In Mongoose

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

![](images/215-using-relations-in-mongoose-1.png)
![](images/215-using-relations-in-mongoose-2.png)
![](images/215-using-relations-in-mongoose-3.png)

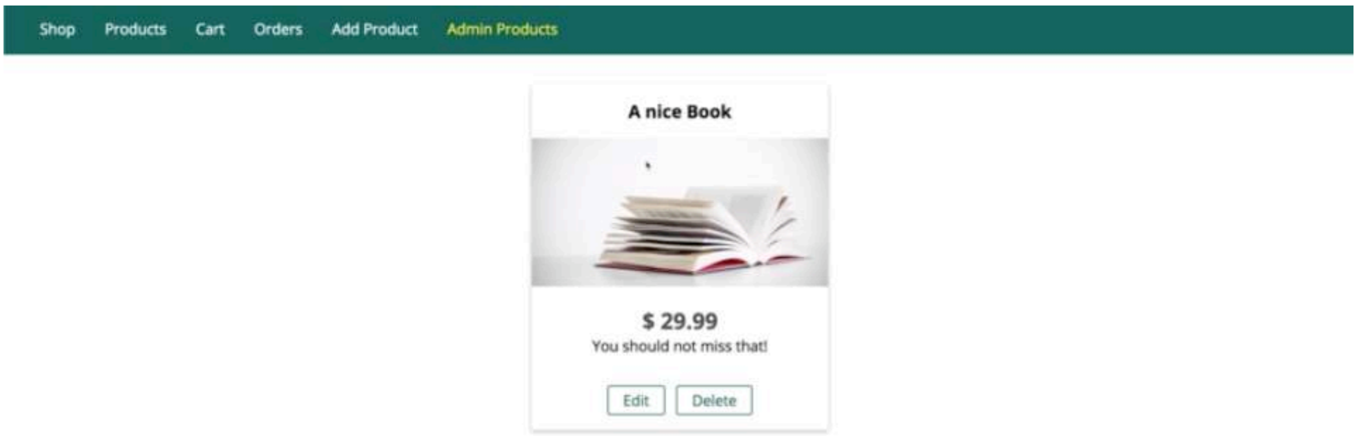


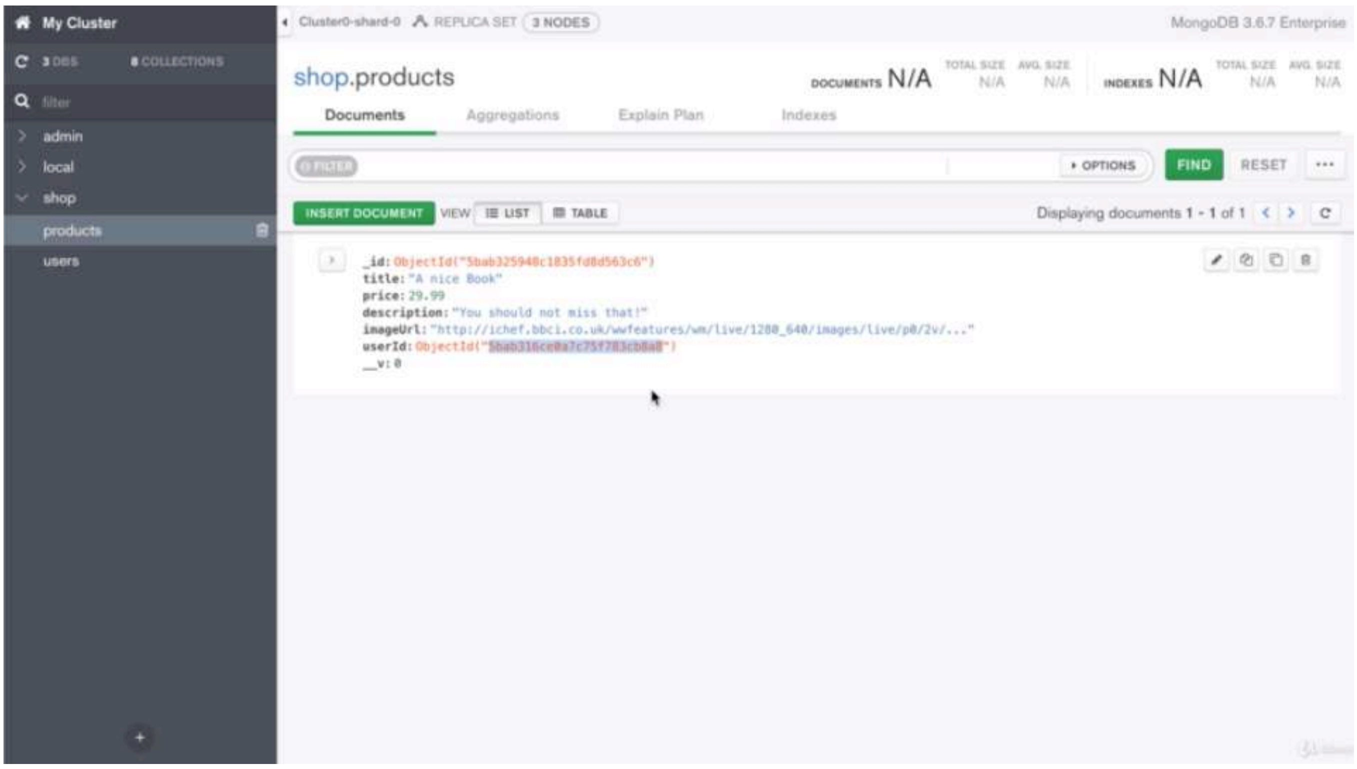
Title
A nice Book

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

Price
29.99

Description
You should not miss that!





```
_id: ObjectId("5bab325948c1835fd8d563c6")
title: "A nice Book"
price: 29.99
description: "You should not miss that!"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bab316ce0a7c75f783cb8a8")
__v: 0
```

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema;

const productSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  imageUrl: {
    type: String,
    required: true
  },
  userId: {
    /**this wiil be a reference to a user,
     * so this will be of type 'Schema.Types.ObjectId'
     *
     * and we can add a special 'ref' configuration
     * and 'ref' takes a string where we tell mongoose
     * hey which other mongoose model is related to the data in that field.
     * we know that we will store a userId
     * but because the type is objectId, this is not obvious
     * this could be any objectId of any object
     * so i will add 'User'
     * and you use the name of your model which you wanna relate this,
     * so since our model here is named user,
     * i will name it user here,
     * so i refer to my User model here
     *
     * with that, i got a relation set up.
     */
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true
  }
})

module.exports = mongoose.model('Product', productSchema)

```

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  /**this also means that in my User model where i store productId,
    * i can also add a reference here and refer to product
    * because i know that for every user in the cart items,
    * i will store products where i refer to some ID
    * and that ID happes to refer to a product stored or defined through the Product model
    * now we got relation set up with ref.
```

```
   *
   * you only need this when using references,
   * when using embedded documents as we do with the cart,
   * you don't need to do anything
   * because you use an embedded document,
   * this already has kind of an implicit relation that is managed inside of one document
   *
   * */
  cart: {
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }]
  }
})

module.exports = mongoose.model('User', userSchema)

```

```
// ./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: title,
    price: price,
    description: description,
    imageUrl: imageUrl,
    /**principle is 'req.user._id'
     * conveniently in mongoose,
     * you can store the entire user object like 'req.user' not 'req.user._id'
     * 'req.user' is the entire user object not just the ID
     * and mongoose will just pick the ID from that object
     */
    userId: req.user
  })
  product
```

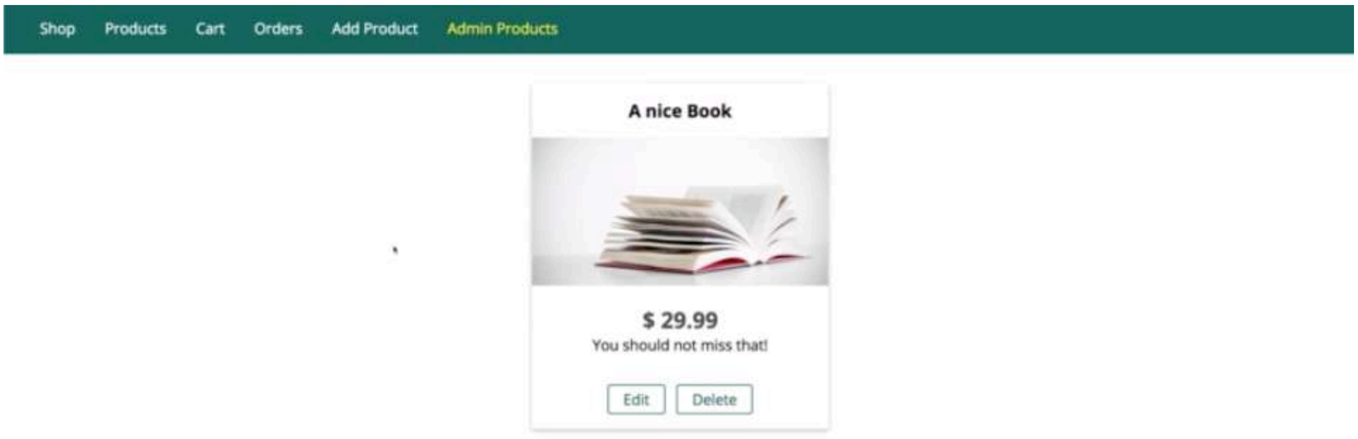
```
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};

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

  Product
    .findById(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));
};

```

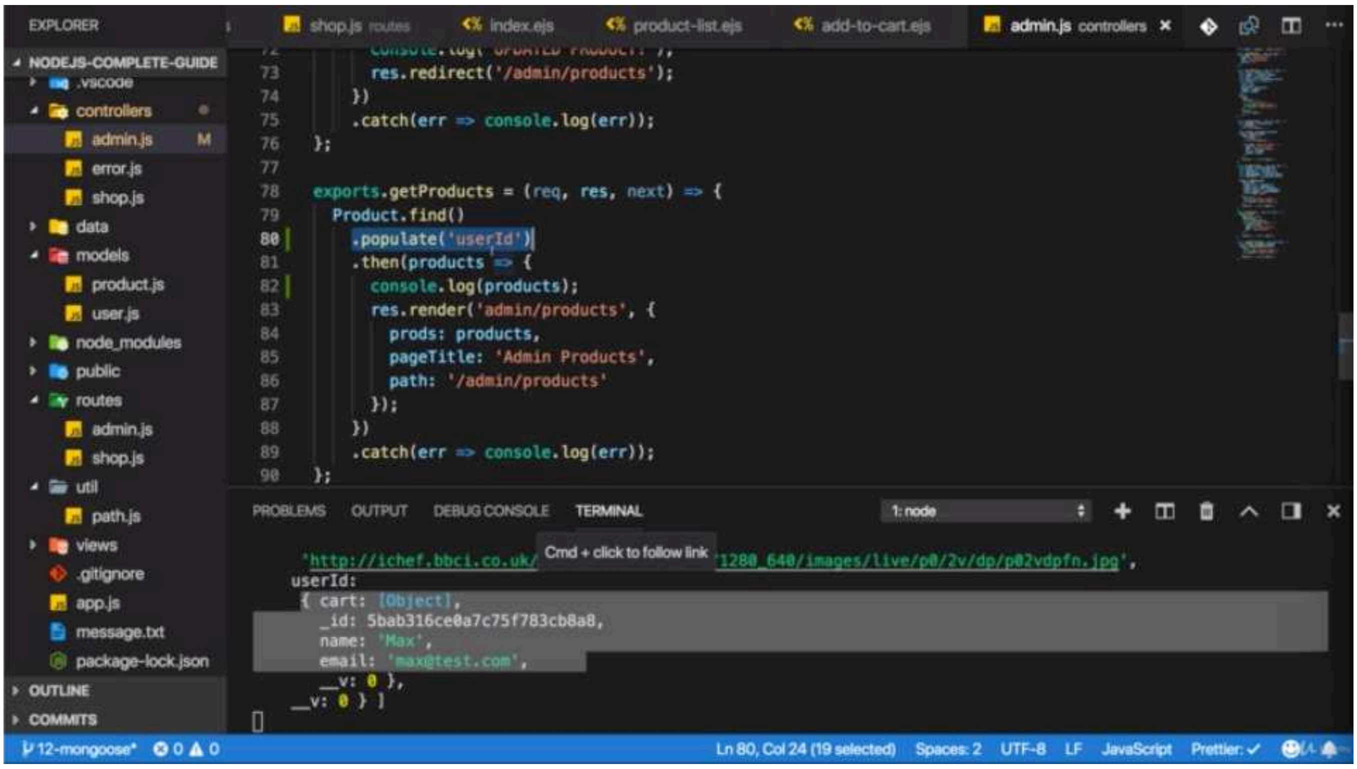
```
exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

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

* Chapter 216: One Important Thing About Fetching Relations

1. update
- ./controllers/admin.js

![](images/216-one-important-thing-about-fetching-relations-1.png)
![](images/216-one-important-thing-about-fetching-relations-2.png)

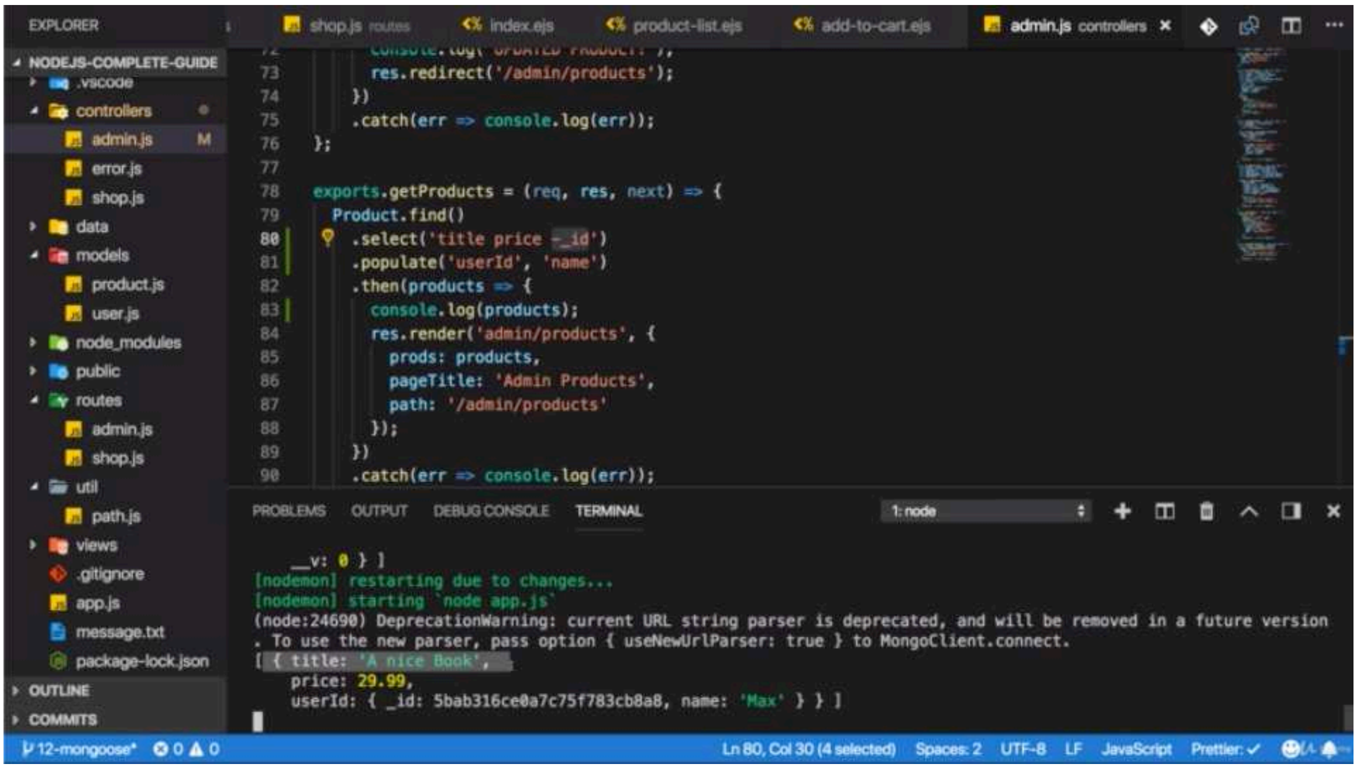




```
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

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

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

- if i reload this page, you will see the userId, but the full user object and that can be really helpful for fetching data because this gives you all the data in one step, instead of writing nested queries on your own.
![](images/216-one-important-thing-about-fetching-relations-3.png)
![](images/216-one-important-thing-about-fetching-relations-4.png)



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

exports.getProducts = (req, res, next) => {
  Product.find()
    .select('title price -_id')
    .populate('userId', 'name')
    .then(products => {
      console.log(products);
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
```

```
    __v: 0 } ]
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:24690) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[ { title: 'A nice Book',
    price: 29.99,
    userId: { _id: 5bab316ce0a7c75f783cb8a8, name: 'Max' } } ]
```

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

```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
  });
};

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: title,
    price: price,
    description: description,
    imageUrl: imageUrl,
    userId: req.user
  })
  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)
    .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;

  Product
    .findById(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));
};

/**we wanna get all the user data for the related user and not just the Id
 */
exports.getProducts = (req, res, next) => {
  Product.find()
    /**this allows you to define which fields you wanna select or unselect,
     * so which fields should be retrieved from the database
     * and there you pass a string where you could say for a product,
     * maybe you wanna get the title and the price
     * but you don't need description and anything else.
     *
     * so you could say 'title price'
     * and you could even exclude something like '_id' by '-' in front of '_id' like '-_id'
     * the same can be done on 'populate()' by passing a 2nd argument.
     *
     */
    .select('title price -_id')
    /**'populate()' allows you to tell mongoose to populate a certain field with all the
detail information and not just the Id
     * i could add 'populate()'
     * and then you first of all describe the path which you wanna populate
     * in my case, that's just the userId field
     * but you could also point at nested paths
    */
    .popultate('userId', 'name')
    .then(products => {
      console.log(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.findByIdAndRemove(prodId)
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
```

* Chapter 217: Working On The Shopping Cart

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

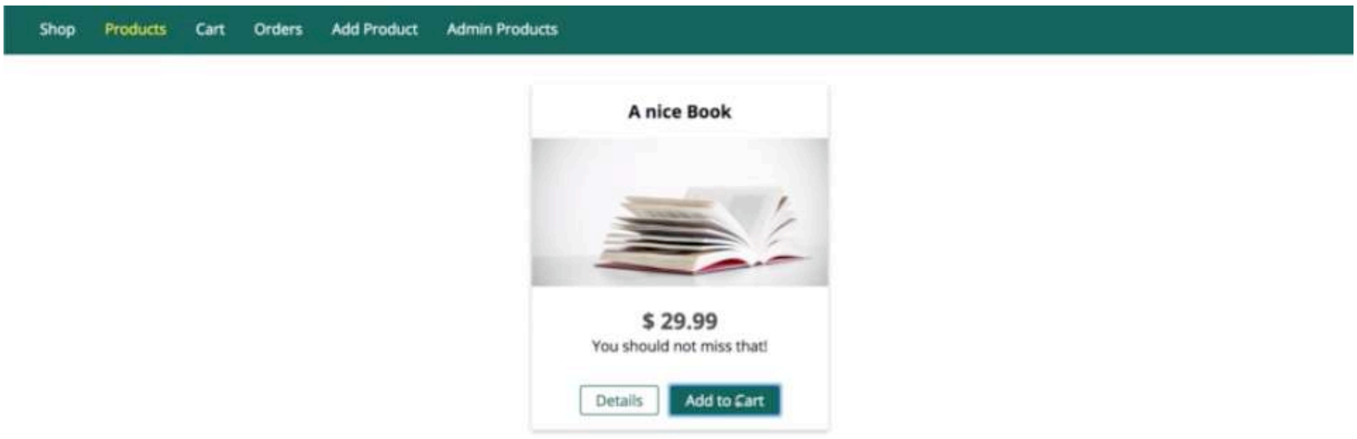
![](images/217-working-on-the-shopping-cart-1.png)

![](images/217-working-on-the-shopping-cart-2.png)

![](images/217-working-on-the-shopping-cart-3.png)

![](images/217-working-on-the-shopping-cart-4.png)

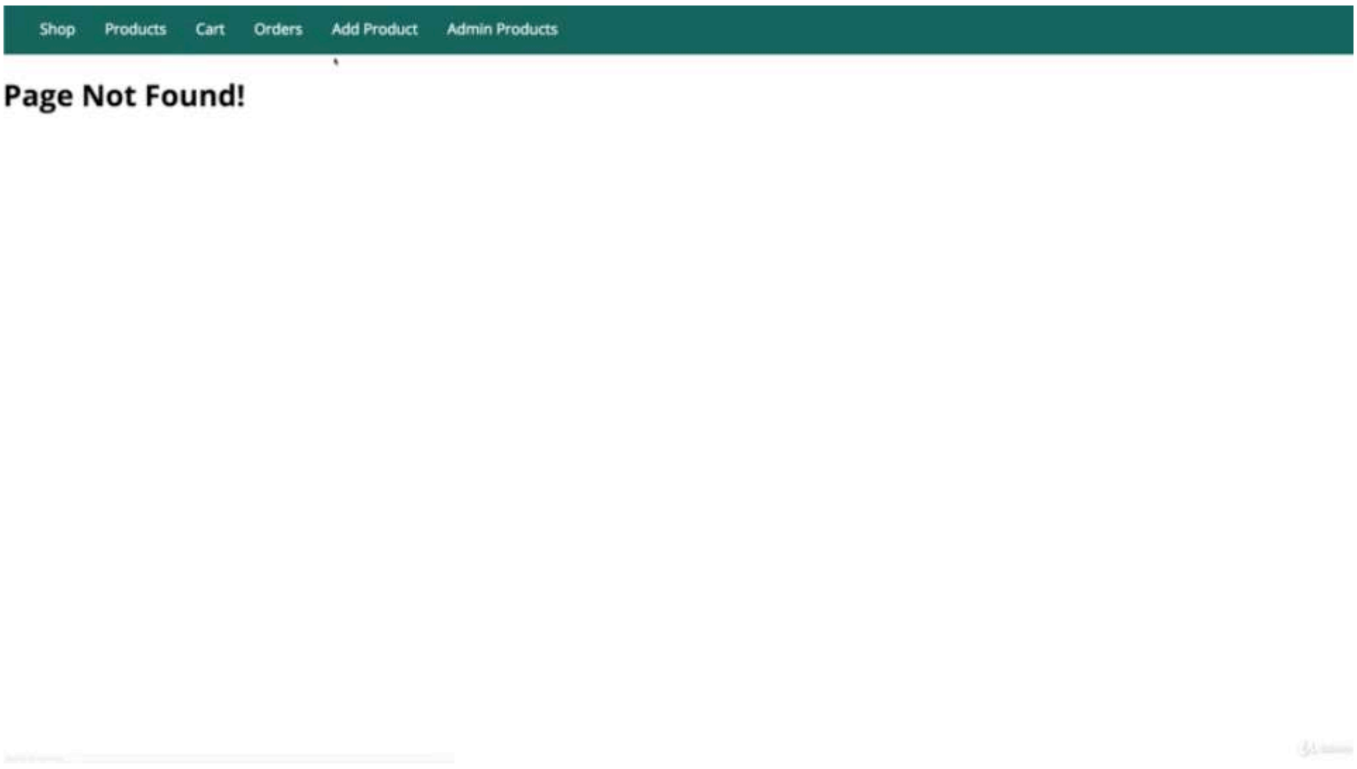
![](images/217-working-on-the-shopping-cart-5.png)



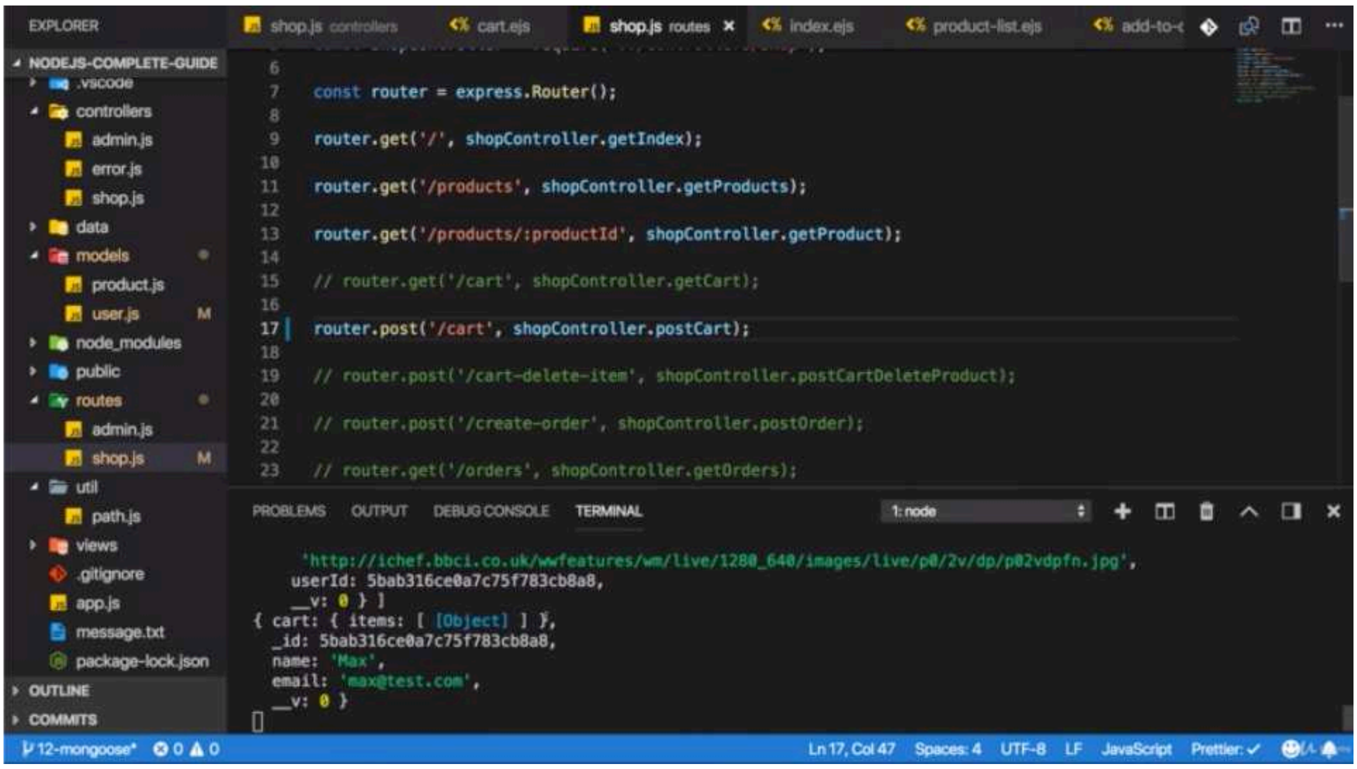
A nice Book

$ 29.99

You should not miss that!

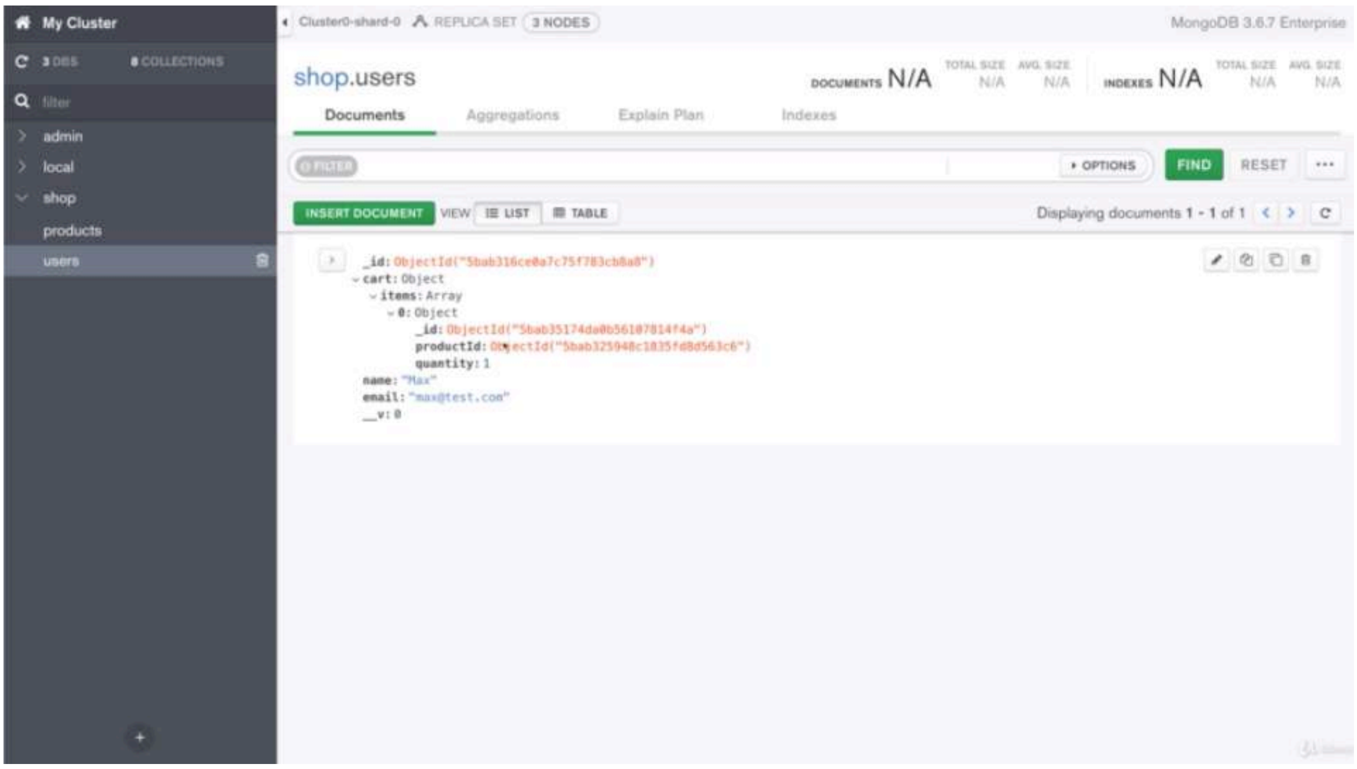


Page Not Found!



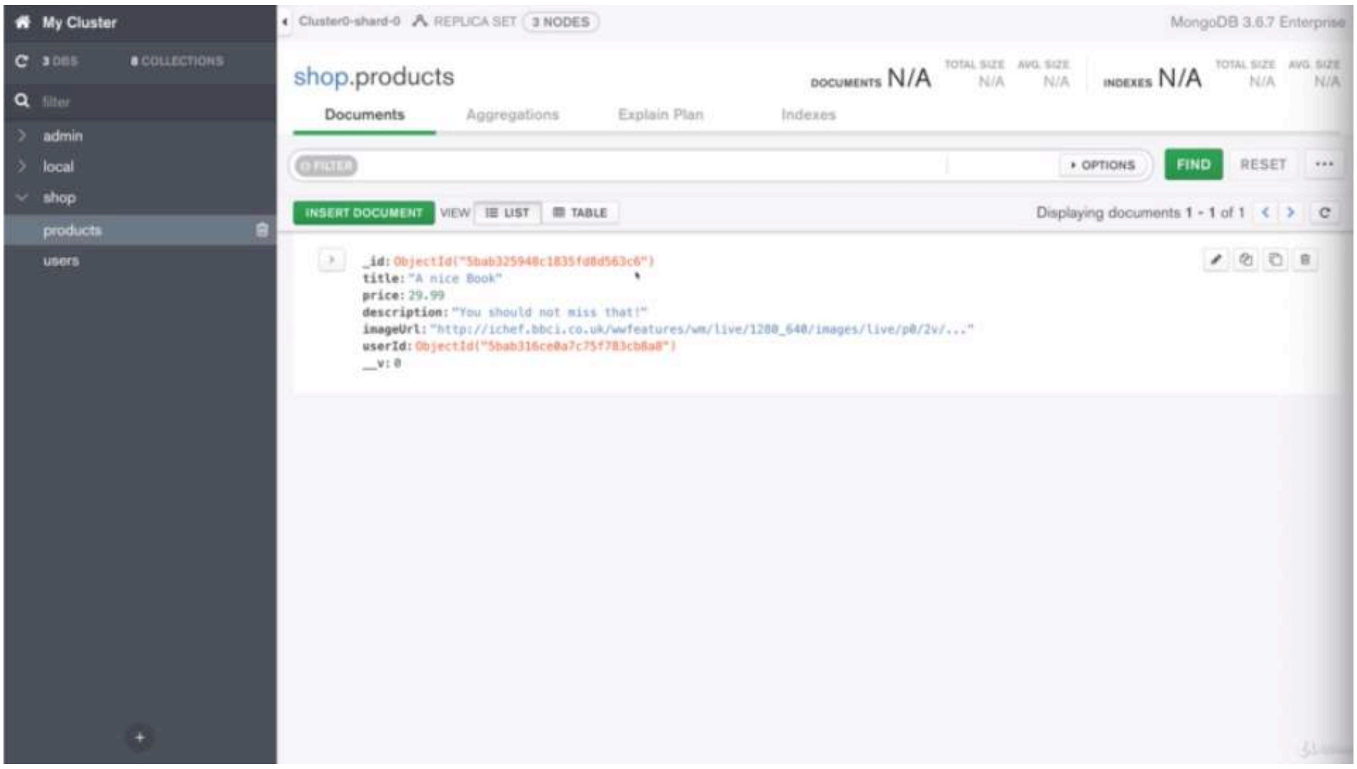
```
const router = express.Router();

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);
```

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



shop.users

```
_id: ObjectId("5bab316ce0a7c75f783cb8a8")
cart: Object
  items: Array
    0: Object
      _id: ObjectId("5bab35174da0b56107814f4a")
      productId: ObjectId("5bab325948c1835fd8d563c6")
      quantity: 1
name: "Max"
email: "max@test.com"
__v: 0
```



```
_id: ObjectId("5bab325948c1835fd8d563c6")
title: "A nice Book"
price: 29.99
description: "You should not miss that!"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bab316ce0a7c75f783cb8a8")
__v: 0
```

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

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  cart: {
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }]
  }
})

/**'methods' key is an object which allows you to add your own methods
 * by adding them to Cart
 * and it has to be a function written like this
 * so that the 'this' keyword still refers to the schema and not to something else
 * and in this function you can add your own logic
 * and that is exactly what i wanna do
 * in this method, i wanna add the logic i had in 'addToCart()' before
 *
 * the function should also receive the 'product' argument which i wanna add
 * that is something we required in the past as well.
 *
 * 'addToCart' here will be called on a real instance based on that schema.
 *
 */
userSchema.methods.addToCart = function(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({
      /**'new ObjectId' will not work
       * so i can store product Id like this 'product._id'
       *
       * make sure that the names you used up there in your schema are the naes you use down there for creating new data
       *
       */
      productId: product._id,
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  }
  this.cart = updatedCart
  /**this should be a utility method that saves itself
   * so where the object saves itself by using the built-in 'save()' method
   * where we update the cart
   */
  return this.save()
}

module.exports = mongoose.model('User', userSchema)
```

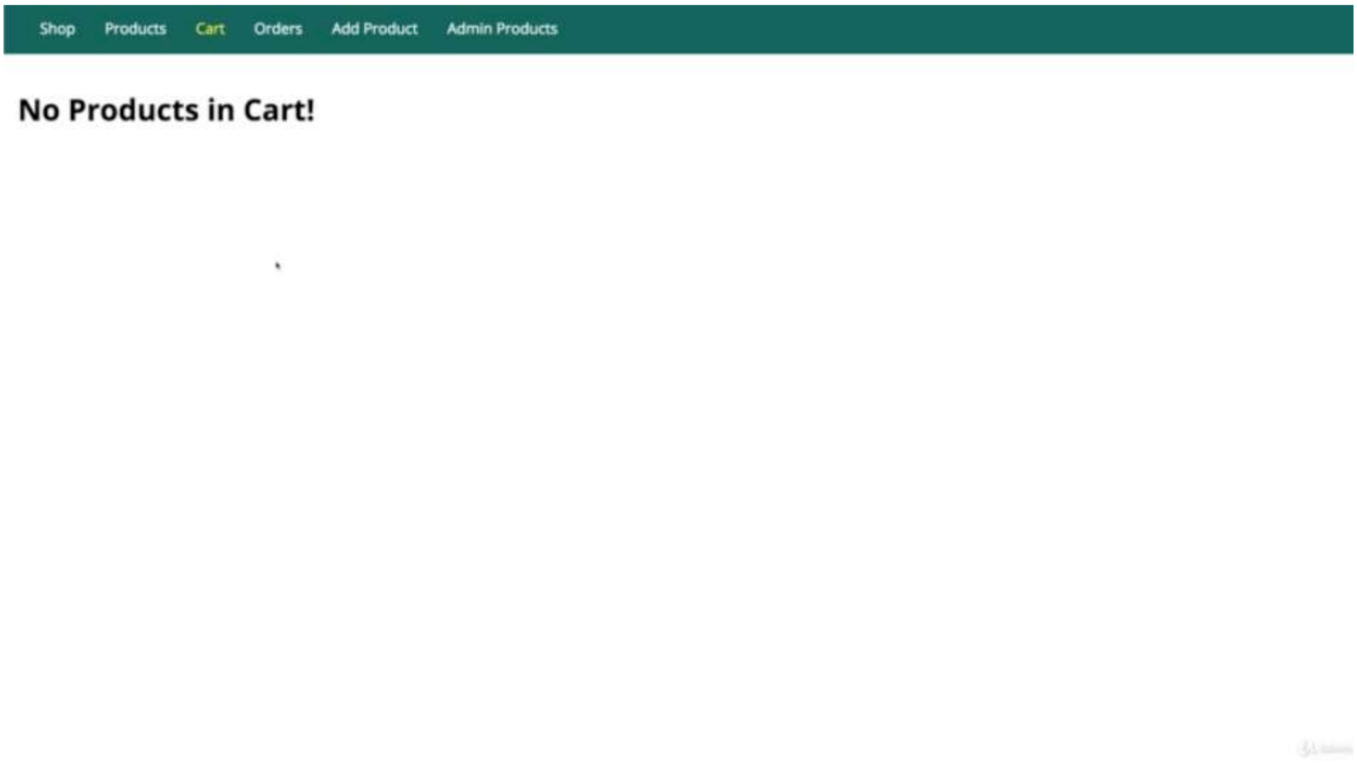
```
//./controllers/shop.js

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

exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      console.log(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.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.find()
    .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);
```

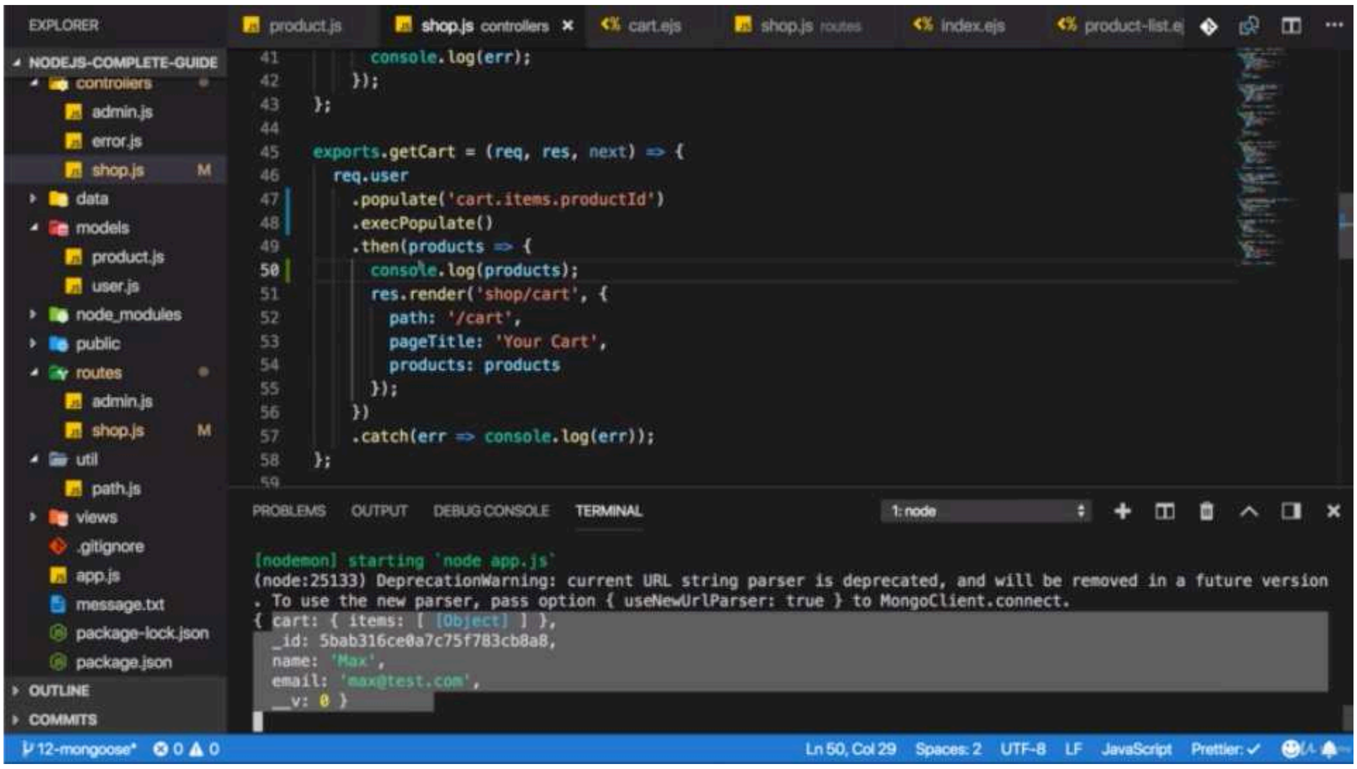
```
*/

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;

```

* Chapter 218: Loading The Cart

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

![](images/218-loading-the-cart-1.png)
![](images/218-loading-the-cart-2.png)


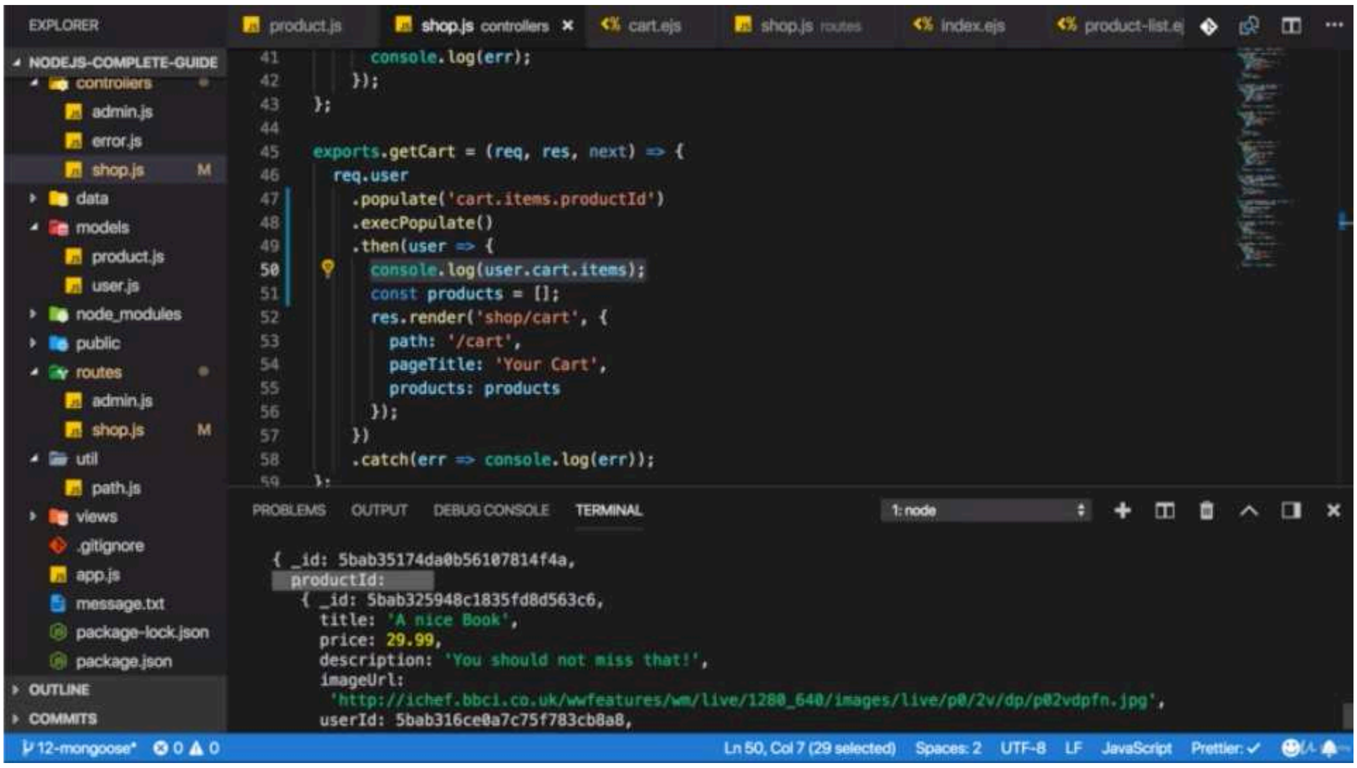
No Products in Cart!



```
      console.log(err);
    });
};

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

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

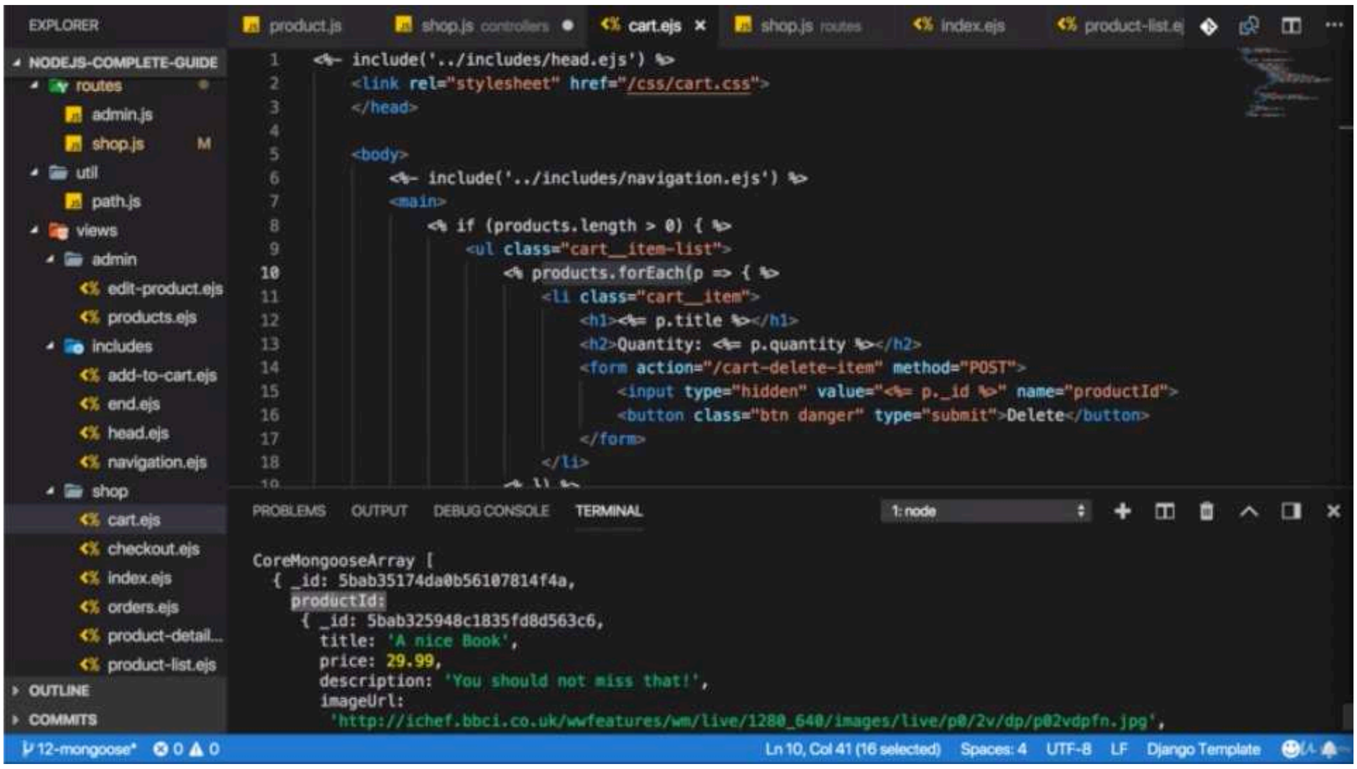
- we see what we have is the full user object which makes sense because we are not fetching products, we still work with the full user.
![](images/218-loading-the-cart-3.png)



```
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      console.log(user.cart.items);
      const products = [];
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
    .catch(err => console.log(err));
};
```

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

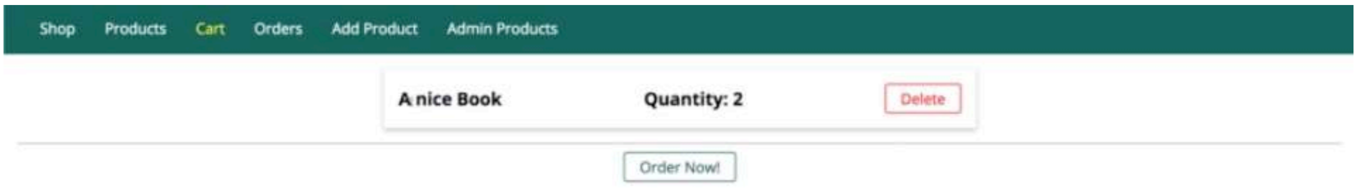
- you see what i log, 'user.cart.items' now is an array of items where the productId is populated with the product data. so now it works a bit different than before but it still gives us the data we need.
![](images/218-loading-the-cart-4.png)



```
<%- 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>
```

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

- we loop through all procuts which is fine, but our product data will then be nested in a product field and you could also rename this to just product in your schema therefore
- but i still have 'productId' here. the title is not available on the top-level object which would be this object on the log
- but on the nested productId object. so we have to say 'p.productId.title' instead 'p.title'.
![](images/218-loading-the-cart-5.png)



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

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

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

exports.getCart = (req, res, next) => {
  req.user
    /**'populate()' doesn't return a promise 'then()'
     * so calling 'then()' on it would not work
     * we have to chain 'execPopulate()' after that
     * and then we will get a promise.
     */
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
```

```javascript
    .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));
};

```

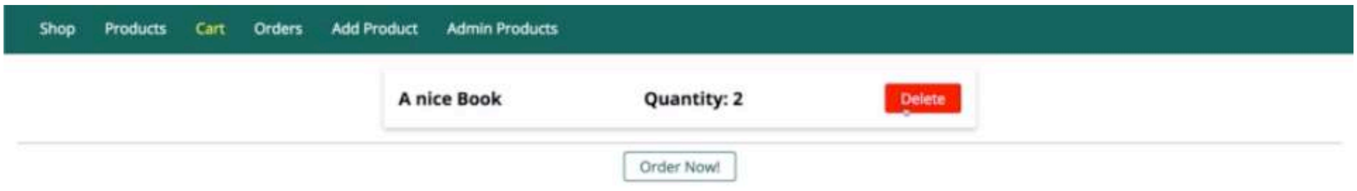
```javascript
// ./routes/shop.js

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;

```

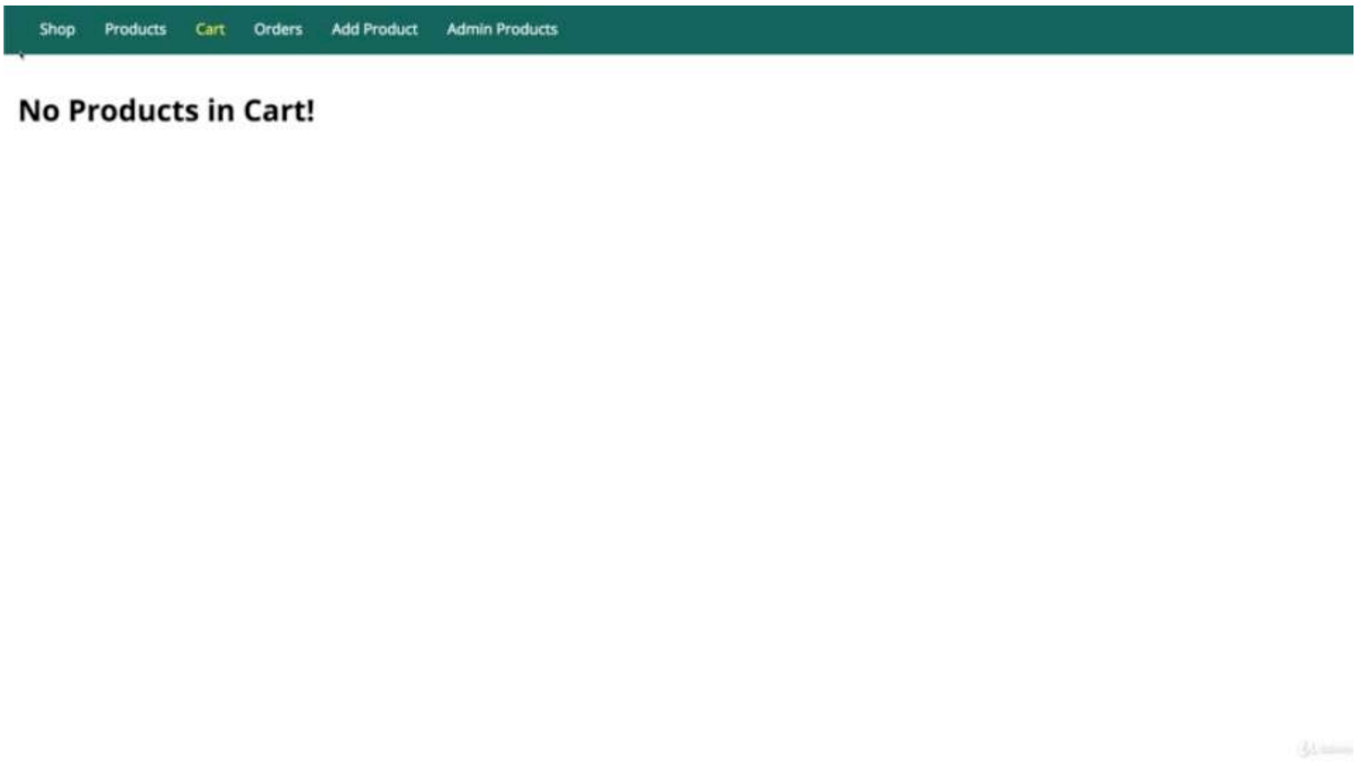
```html
<!--./views/shop/cart.ejs-->

<%- 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">
                    <!--we loop through all products which is fine
                    but our product data will then be nested in a product field
                    and you could also rename this to just 'product' in your schema
therefore

                    i still have 'productId' here the title is not available on the top-
level object which would be this object on the log
                    but on the nested productId object, so we have to say
'p.productId.title' instead 'p.title'

                    and the quantity is on the top-level object. so this is fine.
                    the productId again can be found on the productId nested or embedded
document
                    -->
                    <% products.forEach(p => { %>
                        <li class="cart__item">
                            <h1><%= p.productId.title %></h1>
                            <h2>Quantity: <%= p.quantity %></h2>
                            <form action="/cart-delete-item" method="POST">
                                <input type="hidden" value="<%= p.productId._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') %>
```

* Chapter 219: Deleting Cart Items

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

![](images/219-deleting-cart-items-1.png)
![](images/219-deleting-cart-items-2.png)





No Products in Cart!

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  cart: {
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
      }]
  }
})

userSchema.methods.addToCart = function(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: product._id,
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  }
  this.cart = updatedCart
  return this.save()
}

userSchema.methods.removeFromCart = function(productId){
  const updatedCartItems = this.cart.items.filter(item => {
    return item.productId.toString() !== productId.toString()
  })
  this.cart.items = updatedCartItems
  return this.save()
}

module.exports = mongoose.model('User', userSchema)
```

```
// ./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;

```

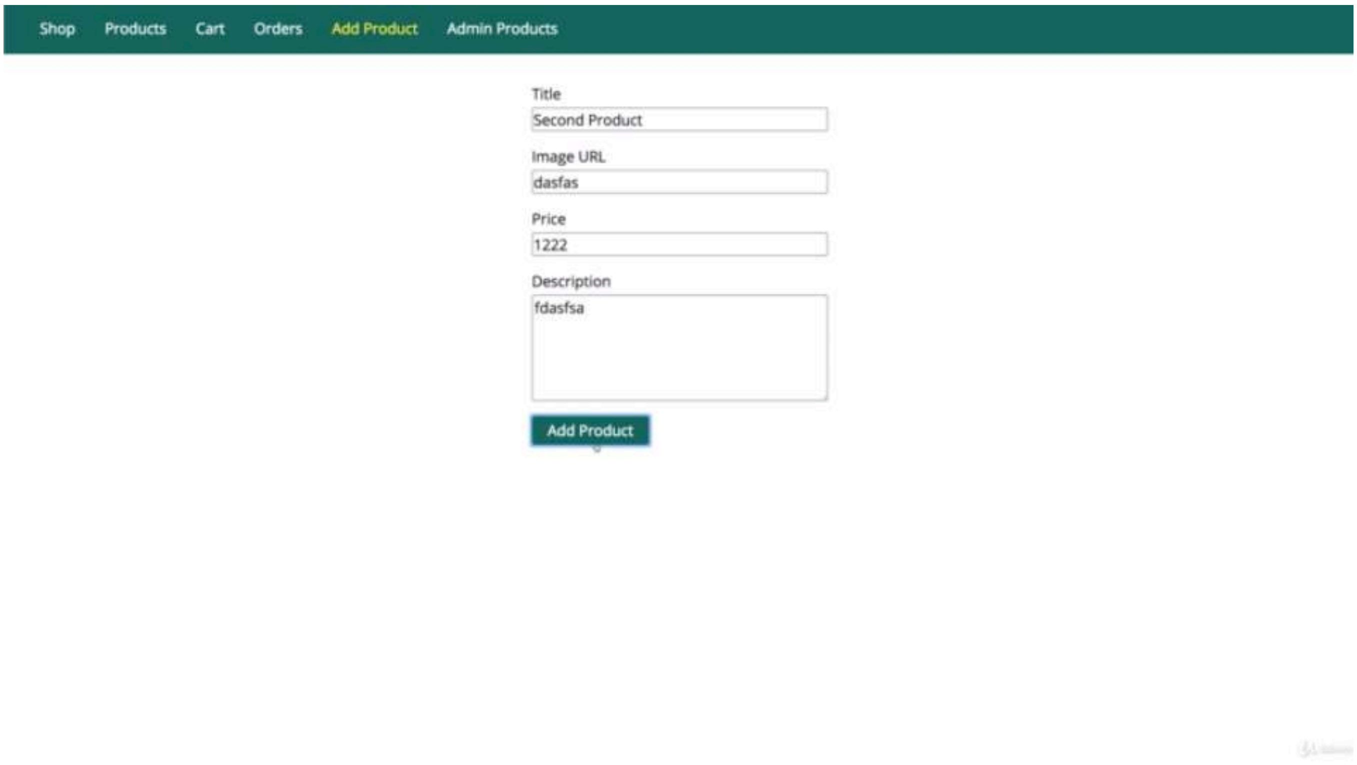
```javascript
//./controllers/shop.js

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

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

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      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
    .removeFromCart(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));
};

```

* Chapter 220: Creating & Getting Orders

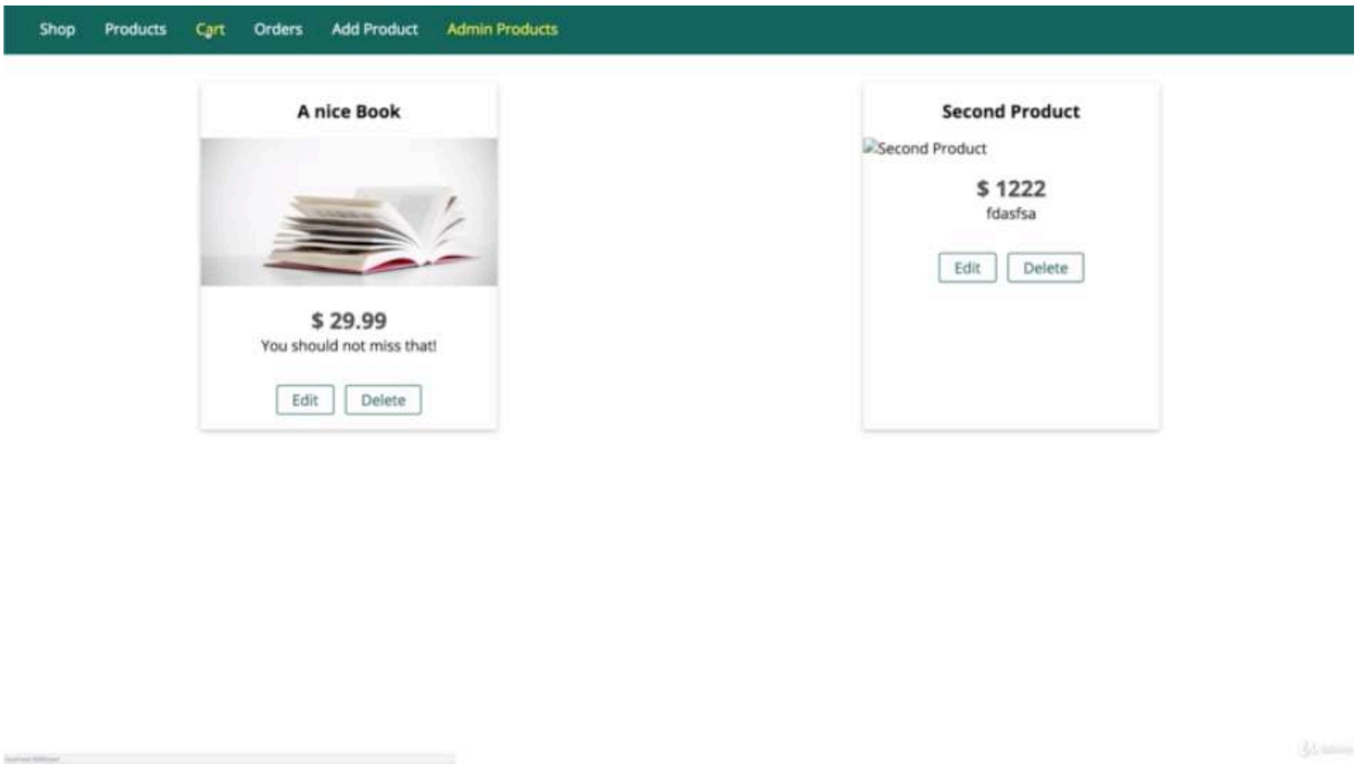
1. update
- ./controllers/shop.js
- ./models/order.js

- ./routes/shop.js

![](images/220-creating-and-getting-orders-1.png)
![](images/220-creating-and-getting-orders-2.png)
![](images/220-creating-and-getting-orders-3.png)
![](images/220-creating-and-getting-orders-4.png)
![](images/220-creating-and-getting-orders-5.png)
![](images/220-creating-and-getting-orders-6.png)
![](images/220-creating-and-getting-orders-7.png)
![](images/220-creating-and-getting-orders-8.png)
![](images/220-creating-and-getting-orders-9.png)
![](images/220-creating-and-getting-orders-10.png)
![](images/220-creating-and-getting-orders-11.png)
![](images/220-creating-and-getting-orders-12.png)
![](images/220-creating-and-getting-orders-13.png)



Title
Second Product
Image URL
dasfas
Price
1222
Description
fdasfsa



A nice Book

$ 29.99

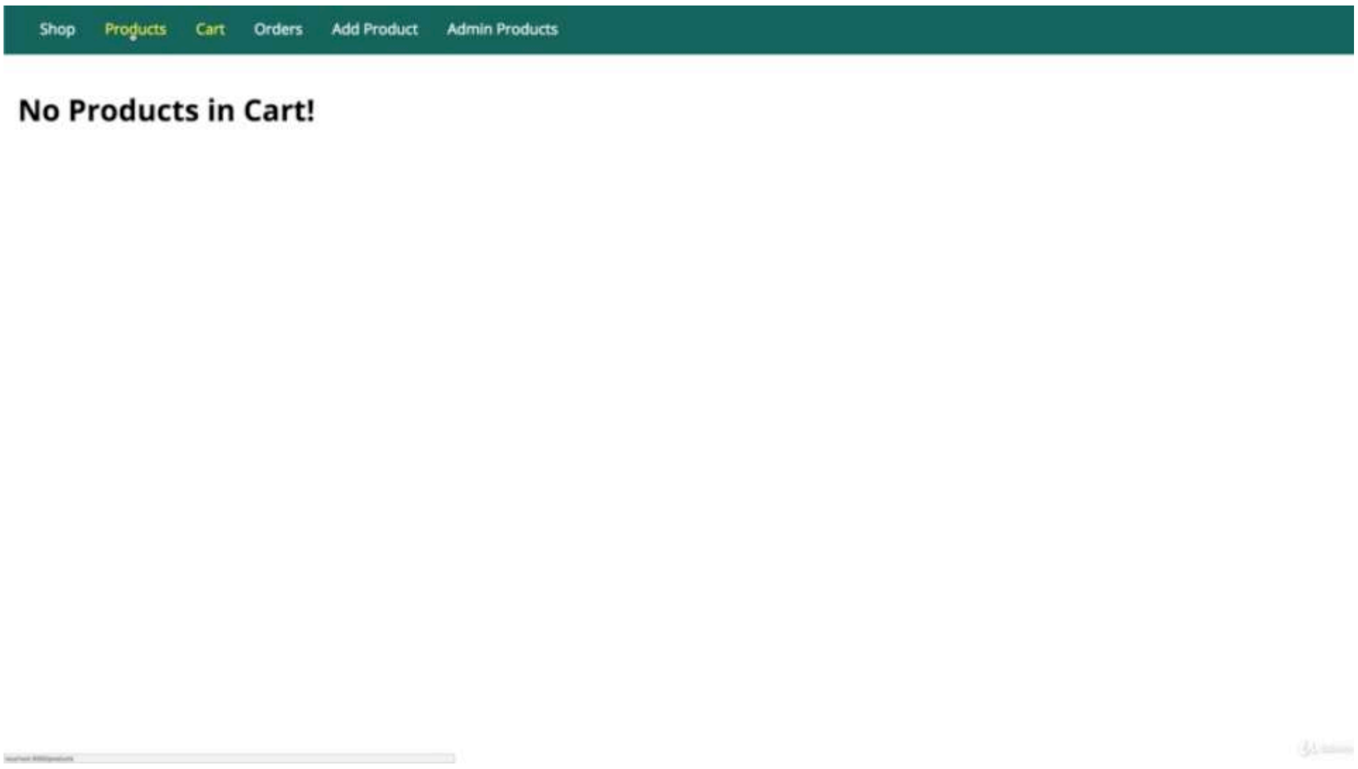
You should not miss that!

Second Product

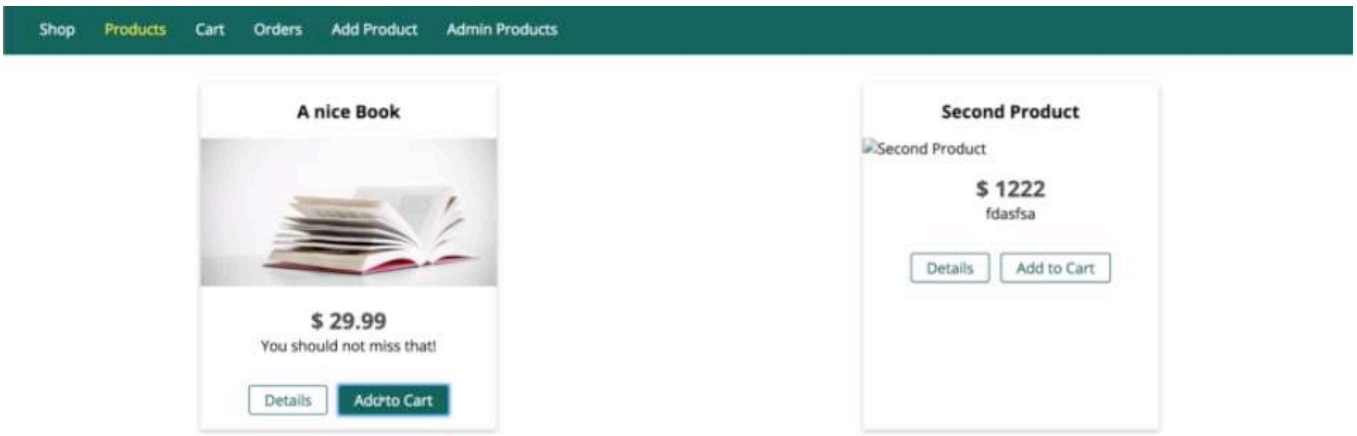
Second Product

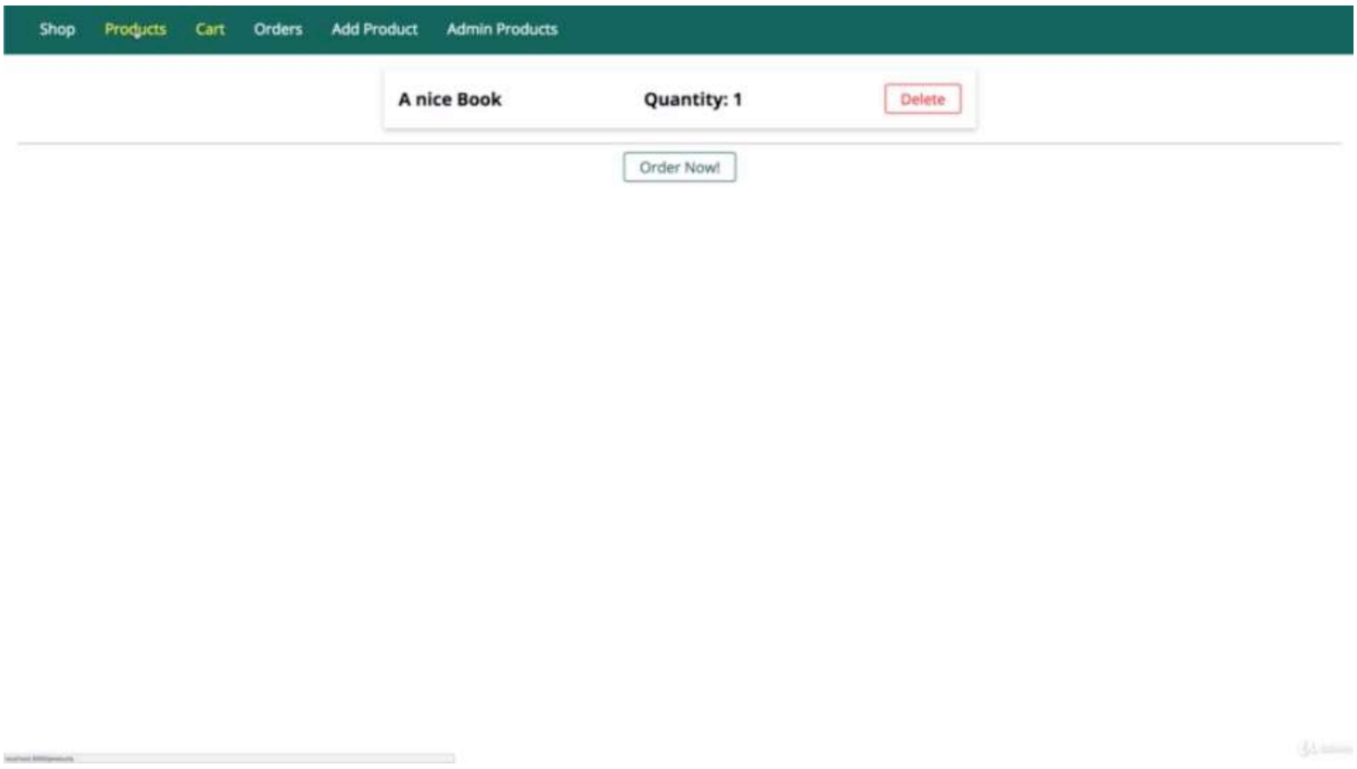
$ 1222

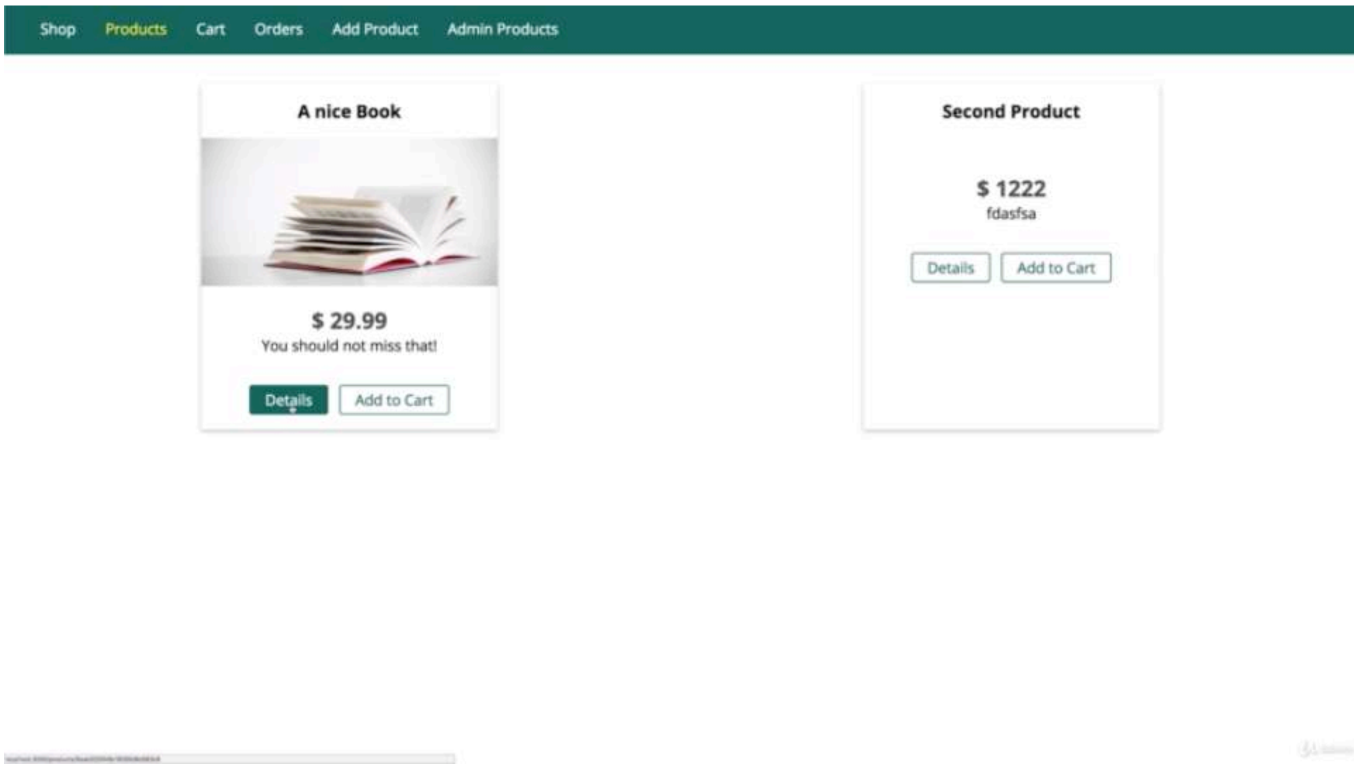
fdasfsa



No Products in Cart!


Shop
Products
Cart
Orders
Add Product
Admin Products
A nice Book
$ 29.99
You should not miss that!
Details
Add to Cart
Second Product
Second Product
$ 1222
fdasfsa
Details
Add to Cart


Shop
Products
Cart
Orders
Add Product
Admin Products
A nice Book
Quantity: 1
Delete
Order Now!


Shop Products Cart Orders Add Product Admin Products

A nice Book

$ 29.99

You should not miss that!

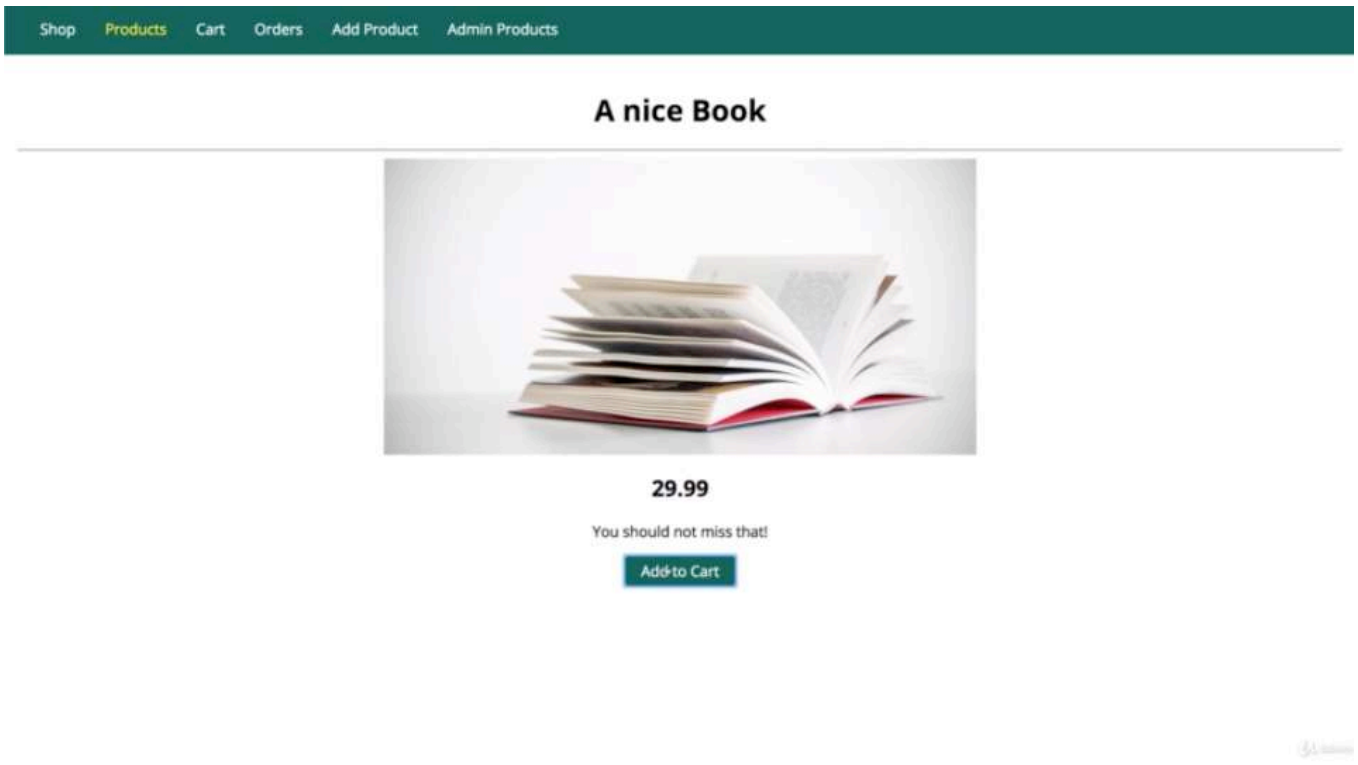
Details Add to Cart

Second Product

$ 1222

fdasfsa

Details Add to Cart

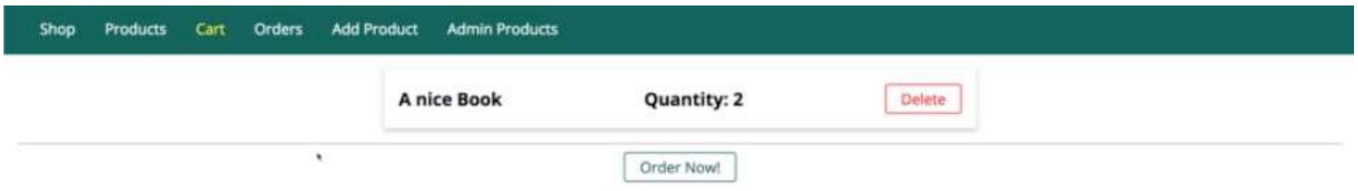

Shop Products Cart Orders Add Product Admin Products

A nice Book

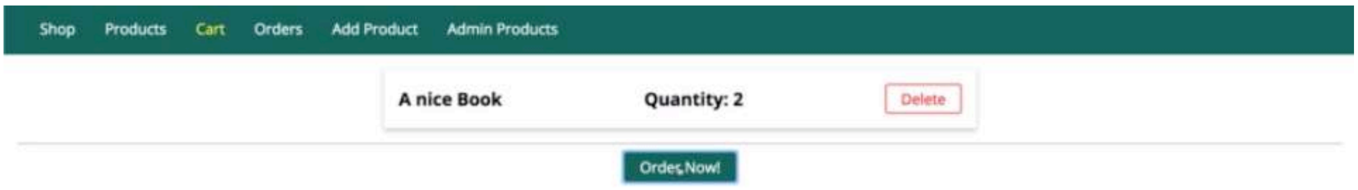
29.99

You should not miss that!

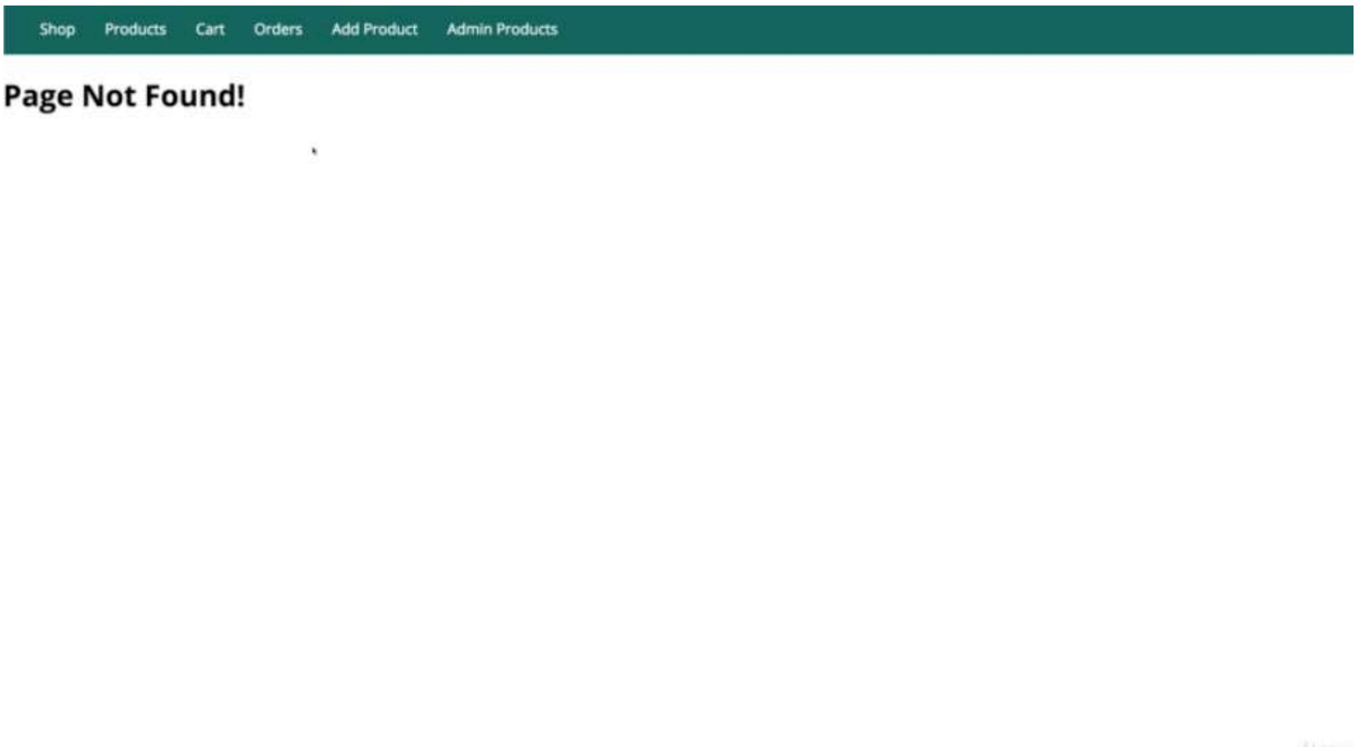
Add to Cart



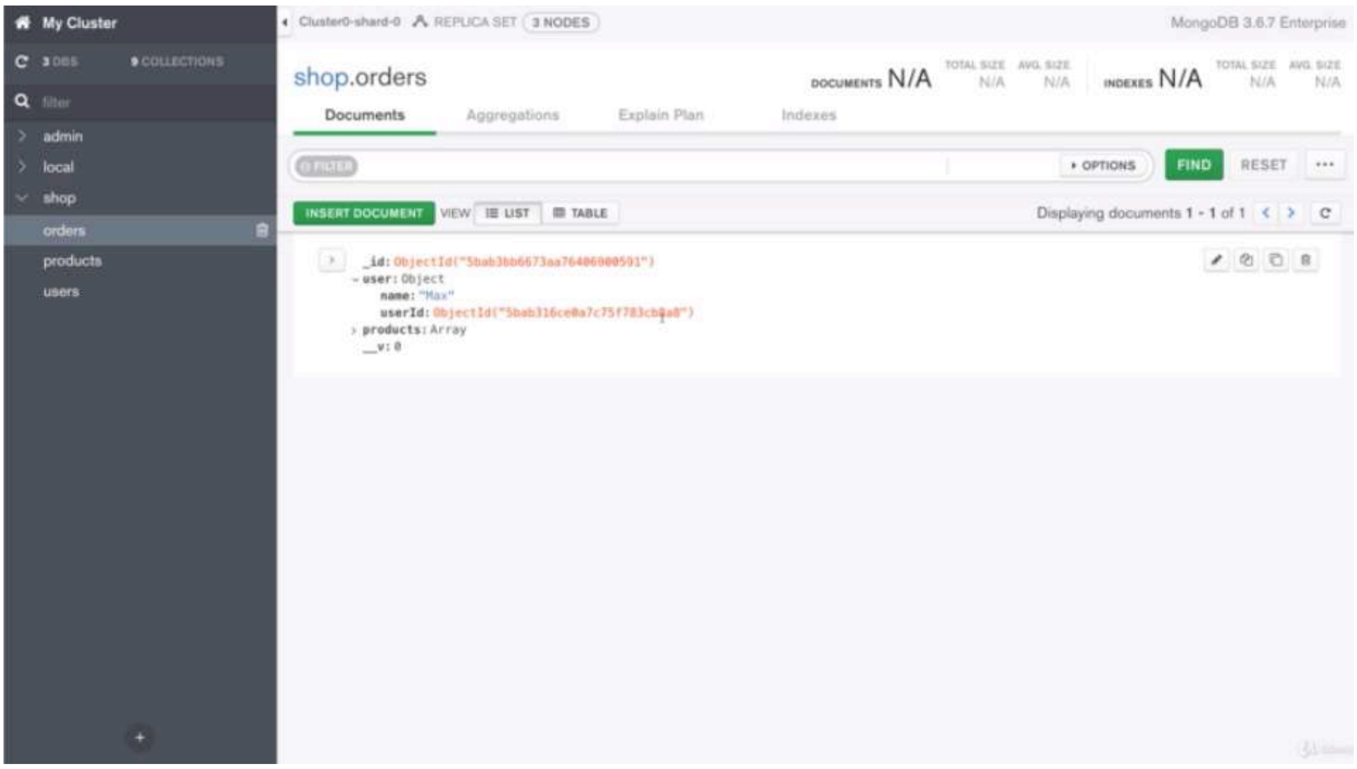
A nice Book Quantity: 2



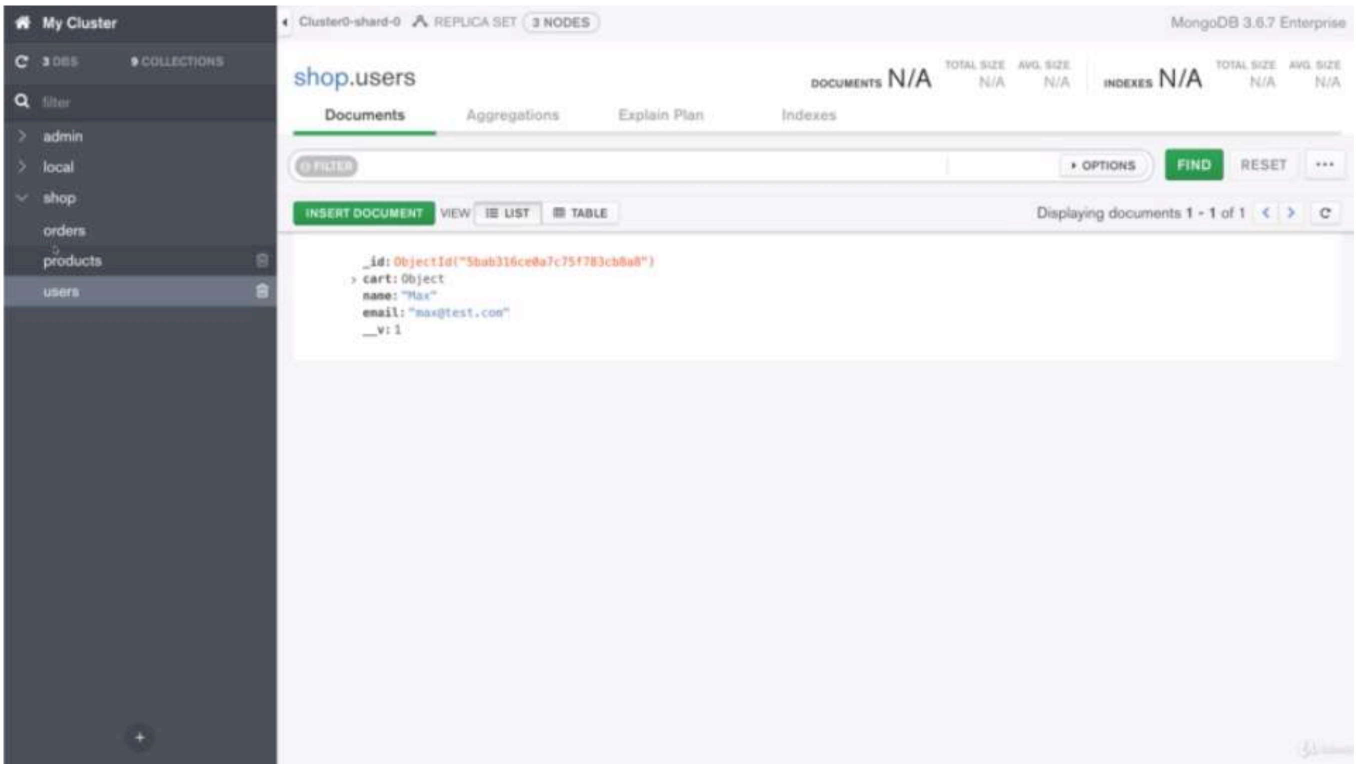
A nice Book Quantity: 2



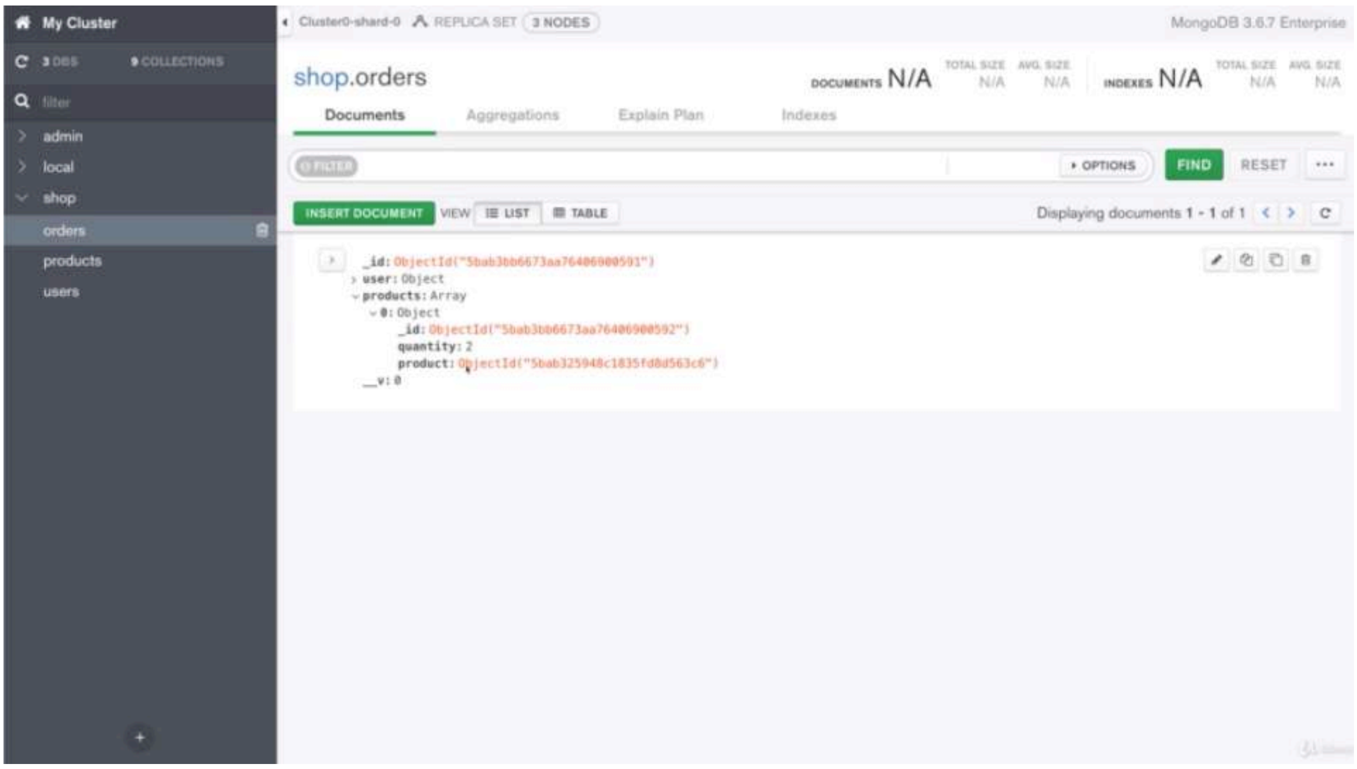
Page Not Found!



```
_id: ObjectId("5bab3bb6673aa76406900591")
⌄ user: Object
    name: "Max"
    userId: ObjectId("5bab316ce0a7c75f783cb8a8")
› products: Array
  __v: 0
```



```
_id: ObjectId("5bab316ce0a7c75f783cb8a8")
> cart: Object
name: "Max"
email: "max@test.com"
__v: 1
```



```
_id: ObjectId("5bab3bb6673aa76406900591")
> user: Object
v products: Array
  v 0: Object
      _id: ObjectId("5bab3bb6673aa76406900592")
      quantity: 2
      product: ObjectId("5bab325948c1835fd8d563c6")
__v: 0
```

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

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

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

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

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      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
    .removeFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  /**this is the approach for fetching all the products that are in the users cart*/
  req.user
  .populate('cart.items.productId')
  .execPopulate()
  .then(user => {
    const products = user.cart.items.map(i => {
      /**we will have a product field
       * and the product field should have all the product data
       * so that we will store everything that i had in i.productId before
       * because that was the old structure we had in there.
       * now we have this structure.
       * we have an array of products which just have a quantity
       * and then the product detail data which is exactly the structure we expect to get in the ./models/order.js file
       *
       */
      return { quantity: i.quantity, product: i.productId }
    })
    /**initialize */
    const order = new Order({
      user: {
        name: req.user.name,
        userId: req.user
      },
      products: products
    })
    order.save()
  })
  .then(result => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

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

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

```

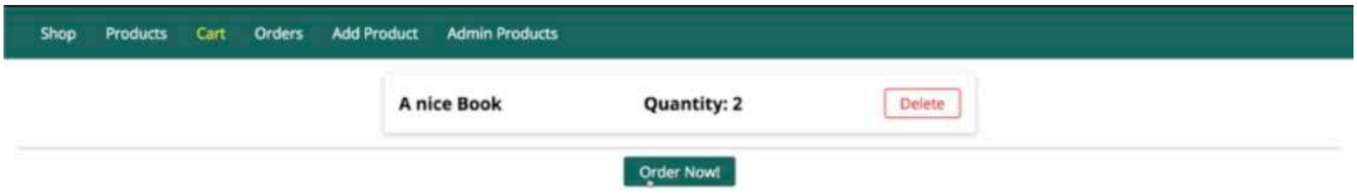
```javascript
// ./routes/shop.js

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;

```

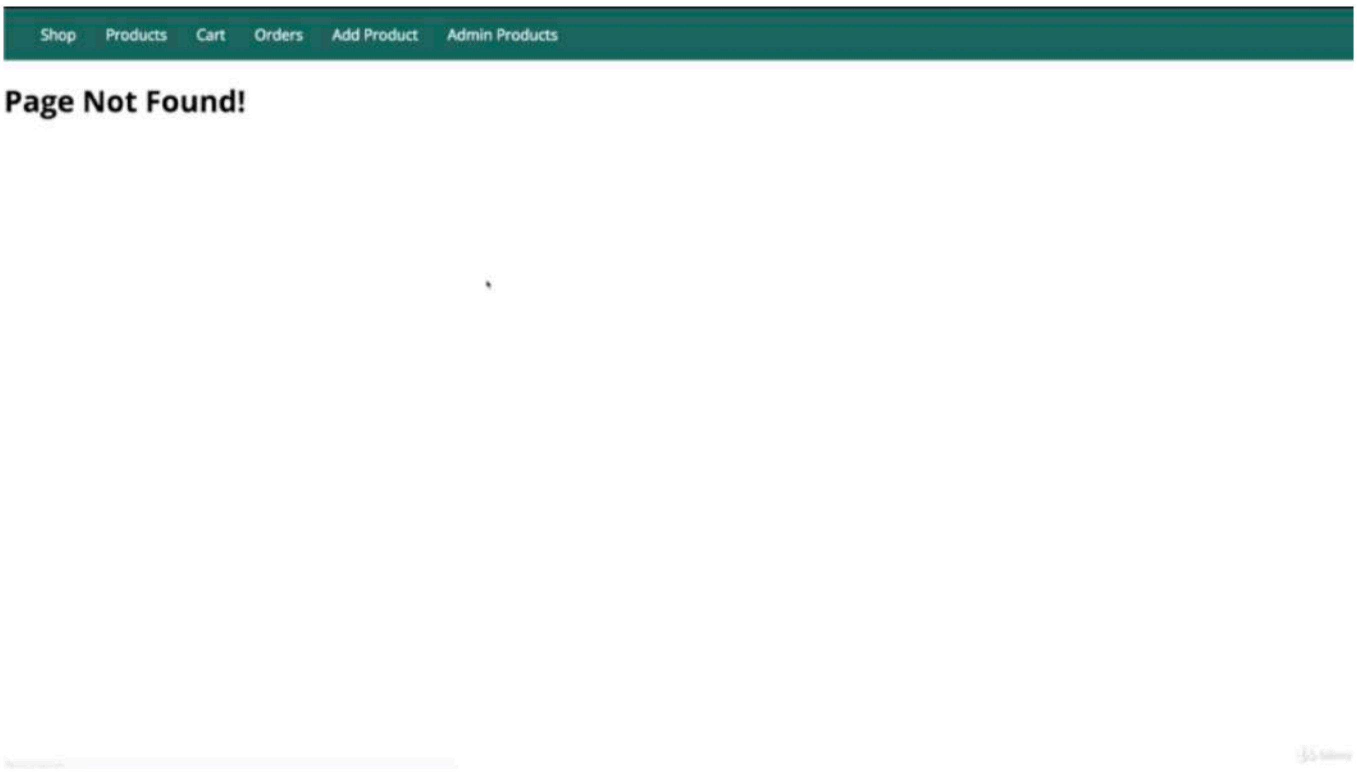
```javascript
// ./models/order.js

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const orderSchema = new Schema({
    products: [{
        product: { type: Object, required: true },
        quantity: { type: Number, required: true }
    }],
    user: {
        name: {
            type: String,
            required: true
        },
        userId: {
            type: Schema.Types.ObjectId,
            required: true,
            ref: 'User'
        }
    }
});

module.exports = mongoose.model('Order', orderSchema)
```

* Chapter 221: Storing All Order Related Data

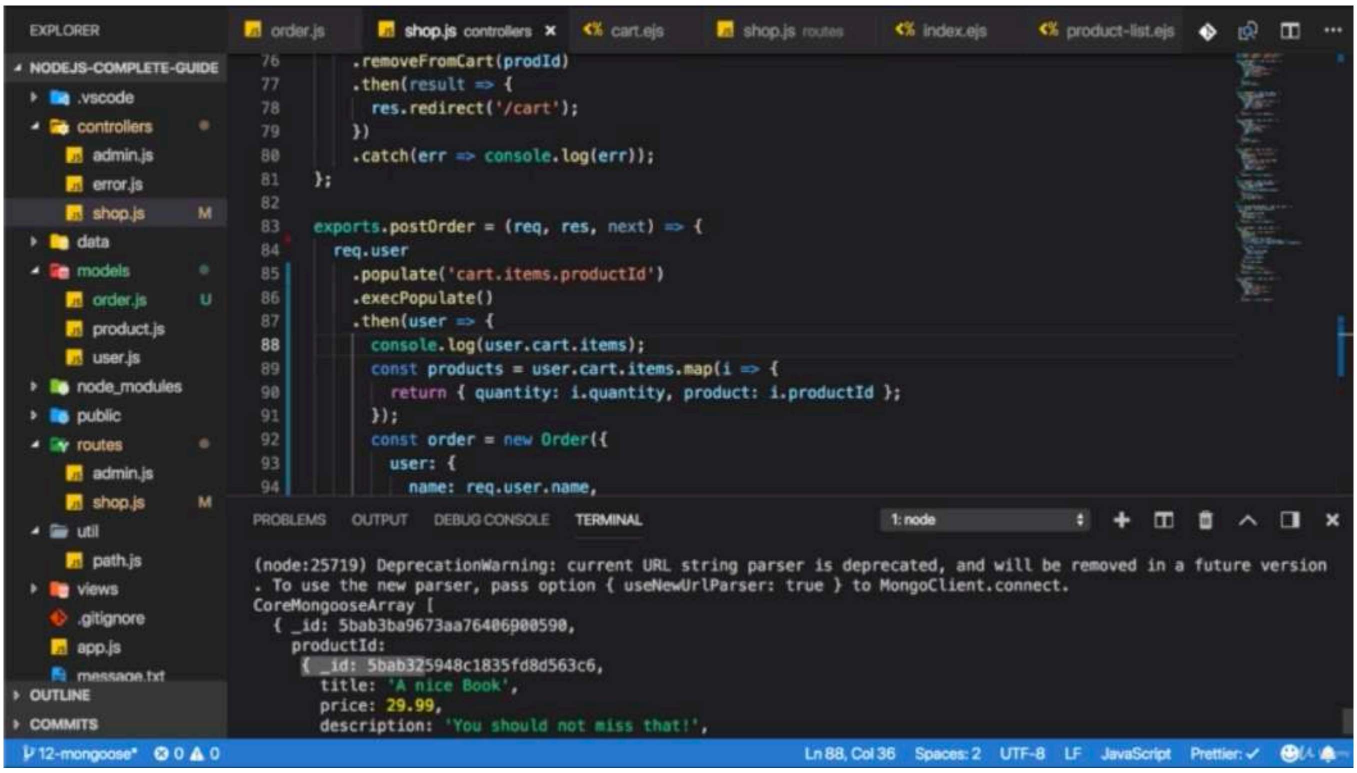
1. update
- ./controllers/shop.js

![](images/221-storing-all-order-related-data-1.png)
![](images/221-storing-all-order-related-data-2.png)
![](images/221-storing-all-order-related-data-3.png)





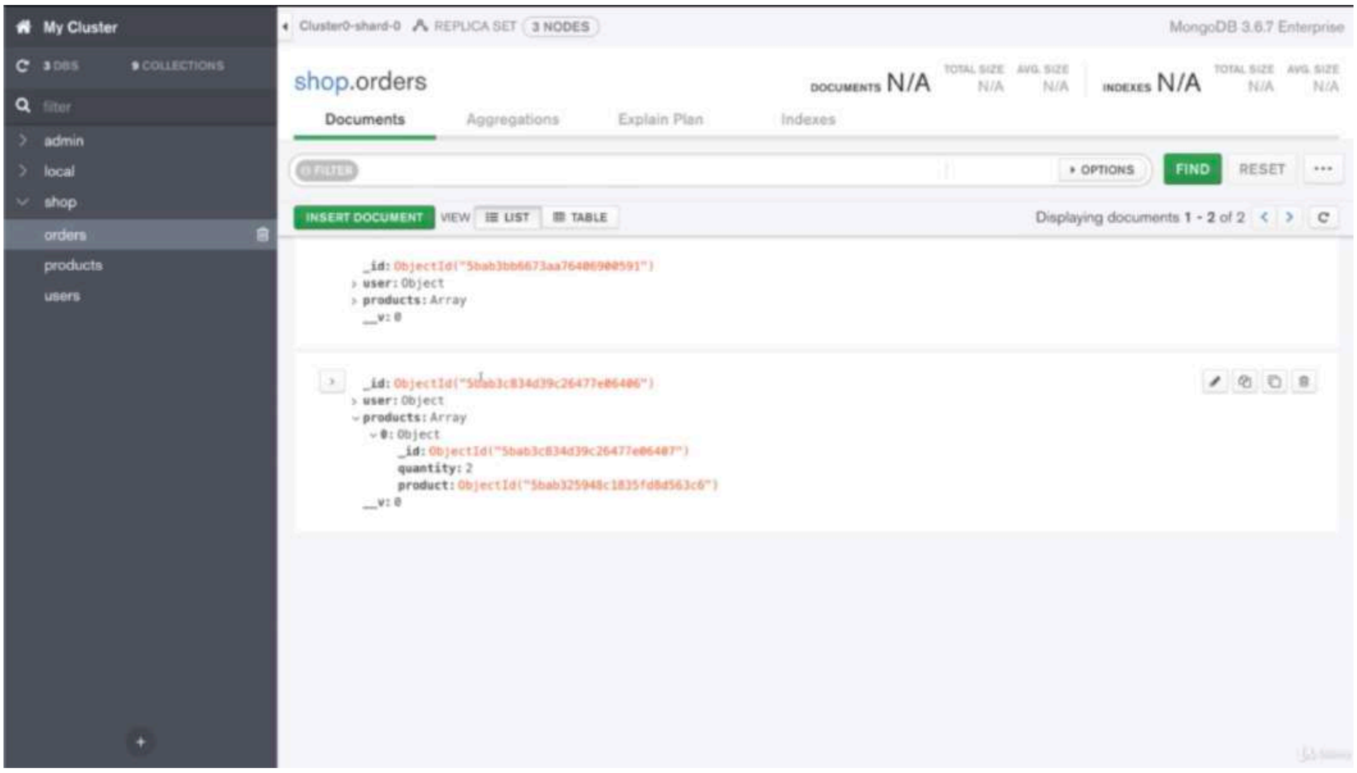
Page Not Found!



```javascript
      .removeFromCart(prodId)
      .then(result => {
        res.redirect('/cart');
      })
      .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      console.log(user.cart.items);
      const products = user.cart.items.map(i => {
        return { quantity: i.quantity, product: i.productId };
      });
      const order = new Order({
        user: {
          name: req.user.name,
```

```
(node:25719) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
CoreMongooseArray [
  { _id: 5bab3ba9673aa76406900590,
    productId:
     { _id: 5bab325948c1835fd8d563c6,
       title: 'A nice Book',
       price: 29.99,
       description: 'You should not miss that!',
```

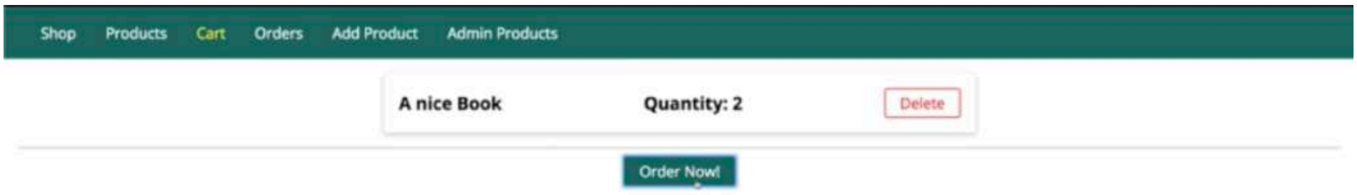
- if i click order now and i increase the console, we see productId does hold a full object and not just the Id which is what gets stored.
![](images/221-storing-all-order-related-data-4.png)

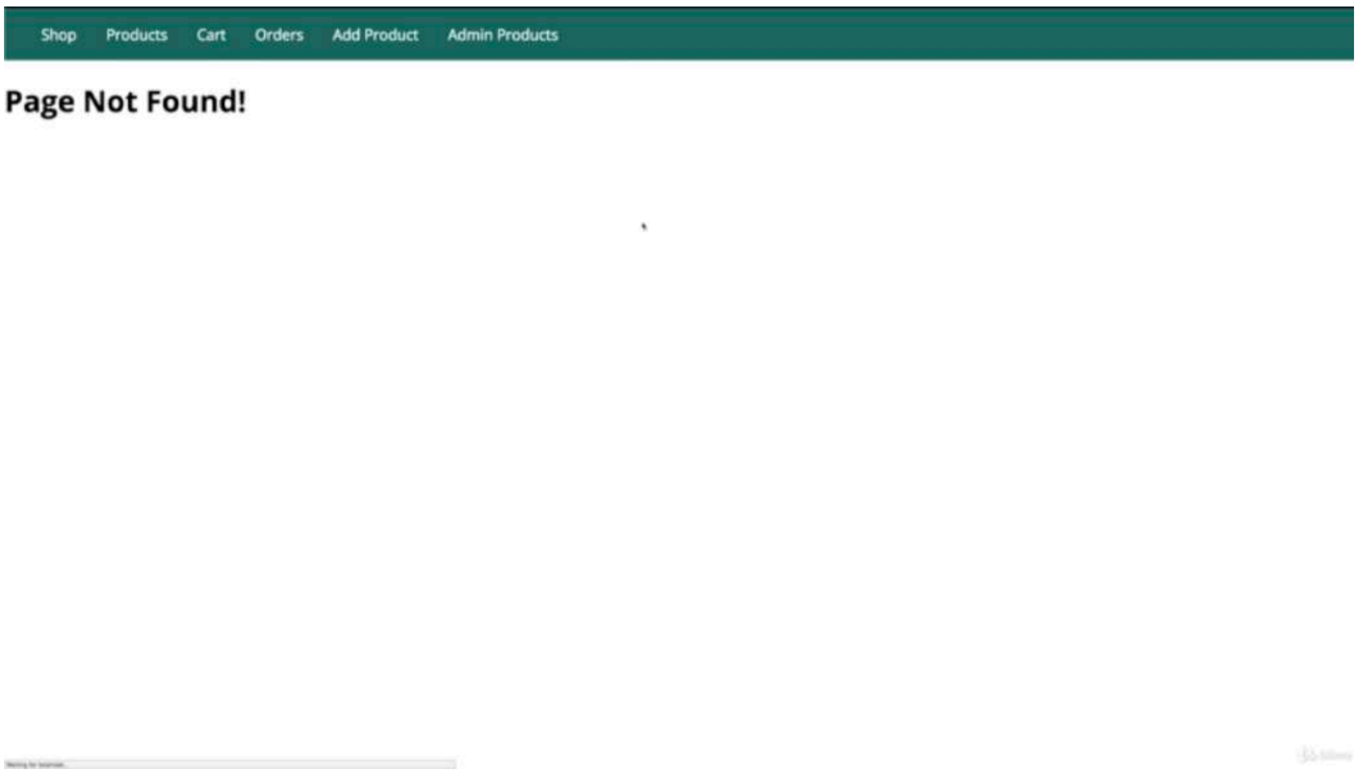


```
_id: ObjectId("5bab3bb6673aa76406900591")
> user: Object
> products: Array
  __v: 0
```

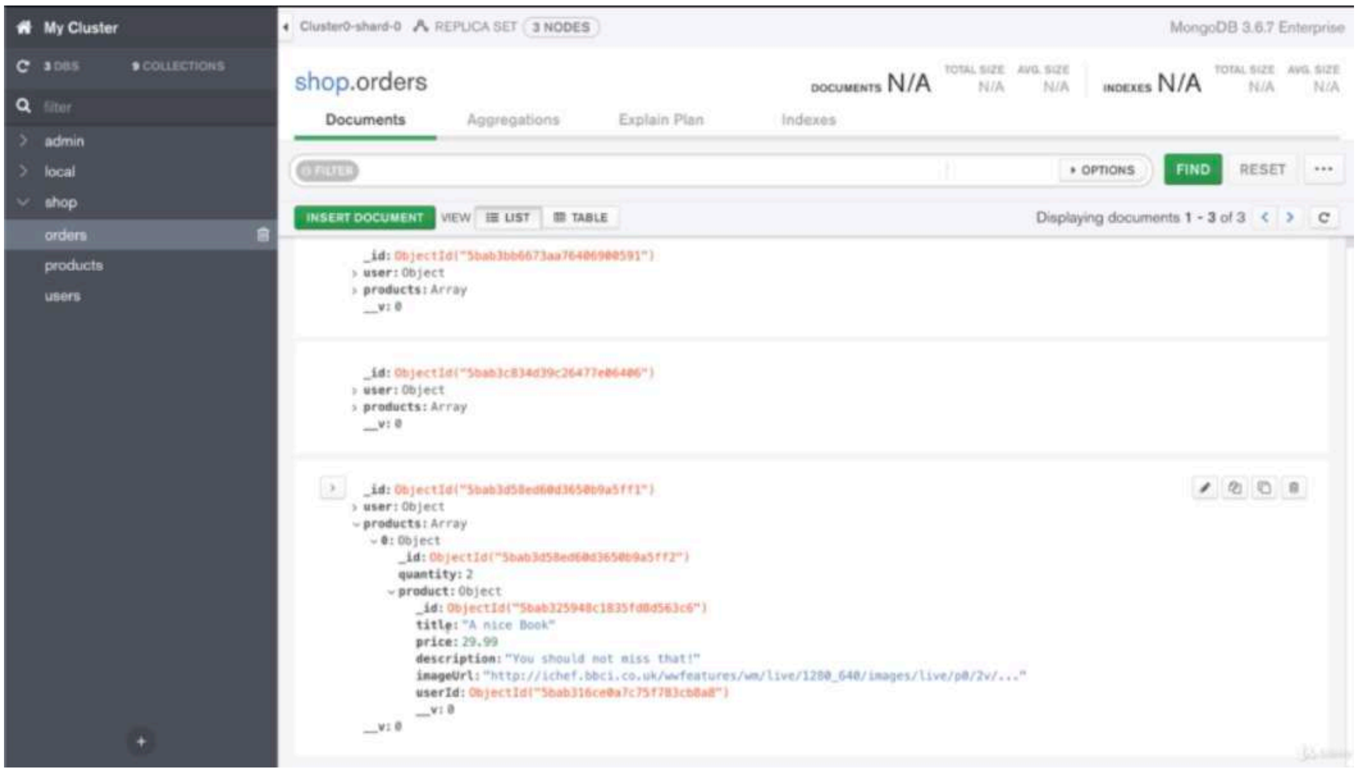
```
_id: ObjectId("5bab3c834d39c26477e06406")
> user: Object
v products: Array
  v 0: Object
      _id: ObjectId("5bab3c834d39c26477e06407")
      quantity: 2
      product: ObjectId("5bab325948c1835fd8d563c6")
  __v: 0
```

- if i refresh my orders in MongoDB Compass, i got 2 but both orders just have the productId in there, not the full product. i wanna have the full product data though.
- so when i store the productId, i can wrap that in curly braces to create a new javascript object.
![](images/221-storing-all-order-related-data-5.png)
![](images/221-storing-all-order-related-data-6.png)
![](images/221-storing-all-order-related-data-7.png)





Page Not Found!



Displaying documents 1 - 3 of 3

```
_id: ObjectId("5bab3bb6673aa76406900591")
> user: Object
> products: Array
  __v: 0
```

```
_id: ObjectId("5bab3c834d39c26477e06406")
> user: Object
> products: Array
  __v: 0
```

```
_id: ObjectId("5bab3d58ed60d3650b9a5ff1")
> user: Object
v products: Array
  v 0: Object
      _id: ObjectId("5bab3d58ed60d3650b9a5ff2")
      quantity: 2
    v product: Object
        _id: ObjectId("5bab325948c1835fd8d563c6")
        title: "A nice Book"
        price: 29.99
        description: "You should not miss that!"
        imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
        userId: ObjectId("5bab316ce0a7c75f783cb8a8")
        __v: 0
  __v: 0
```

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

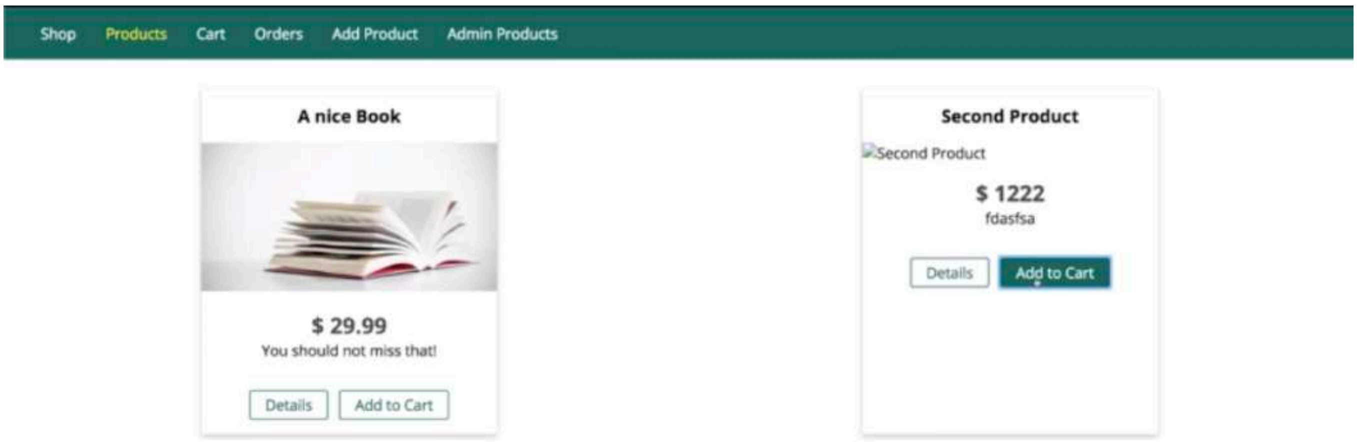
```
//./controllers/shop.js

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

exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      console.log(products)
      res.render('shop/product-list', {
```

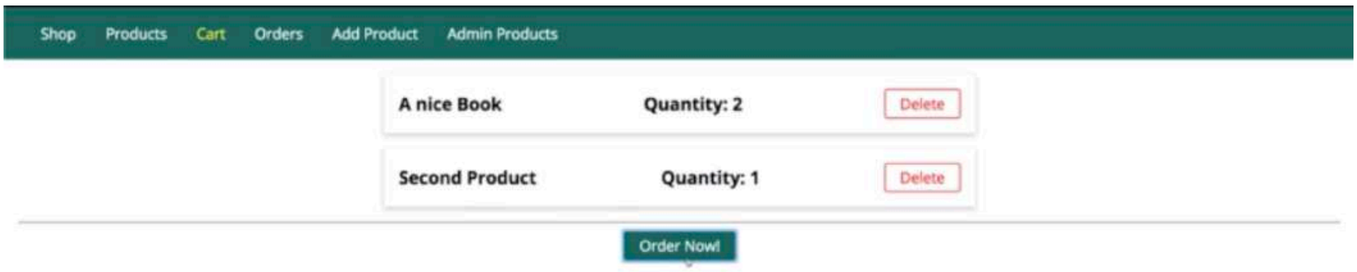
```javascript
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
  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.find()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      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
    .removeFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  req.user
  .populate('cart.items.productId')
  .execPopulate()
  .then(user => {
    /**if i click 'order now'
     * and i increase the console,
     * we see productId does hold a full object
     * and not just the Id which is what gets stored.
     *
     * if i refresh my orders in MongoDB Compass,
     * i got 2 but both orders have the productId in there,
     * not the full product,
     * but i wanna have the full product data though.
     *
     * so when i store the productId,
     * i can wrap that in curly braces to create a new javascript object.
     * and use the spread operator
     * and use that not directly on the productId
     * but on a special field,
     * Mongoose gives me '_doc'
     * i can access 'productId'
     * because productId will be an object with a lot of metadata attached to it
     * even though we can't directly see that when console logging it
     * but with '._doc',
     * we get access to the data that is in there
     * and then with the spread operator inside of a new object,
     * we pull out all the data in that document we retrieved
     * and store it in a new object which we save here as a product
     */
    console.log(user.cart.items)
    const products = user.cart.items.map(i => {

      return { quantity: i.quantity, product: { ...i.productId._doc } }
    })
    const order = new Order({
      user: {
        name: req.user.name,
        userId: req.user
      },
      products: products
    })
    order.save()
  })
  .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));
};

```

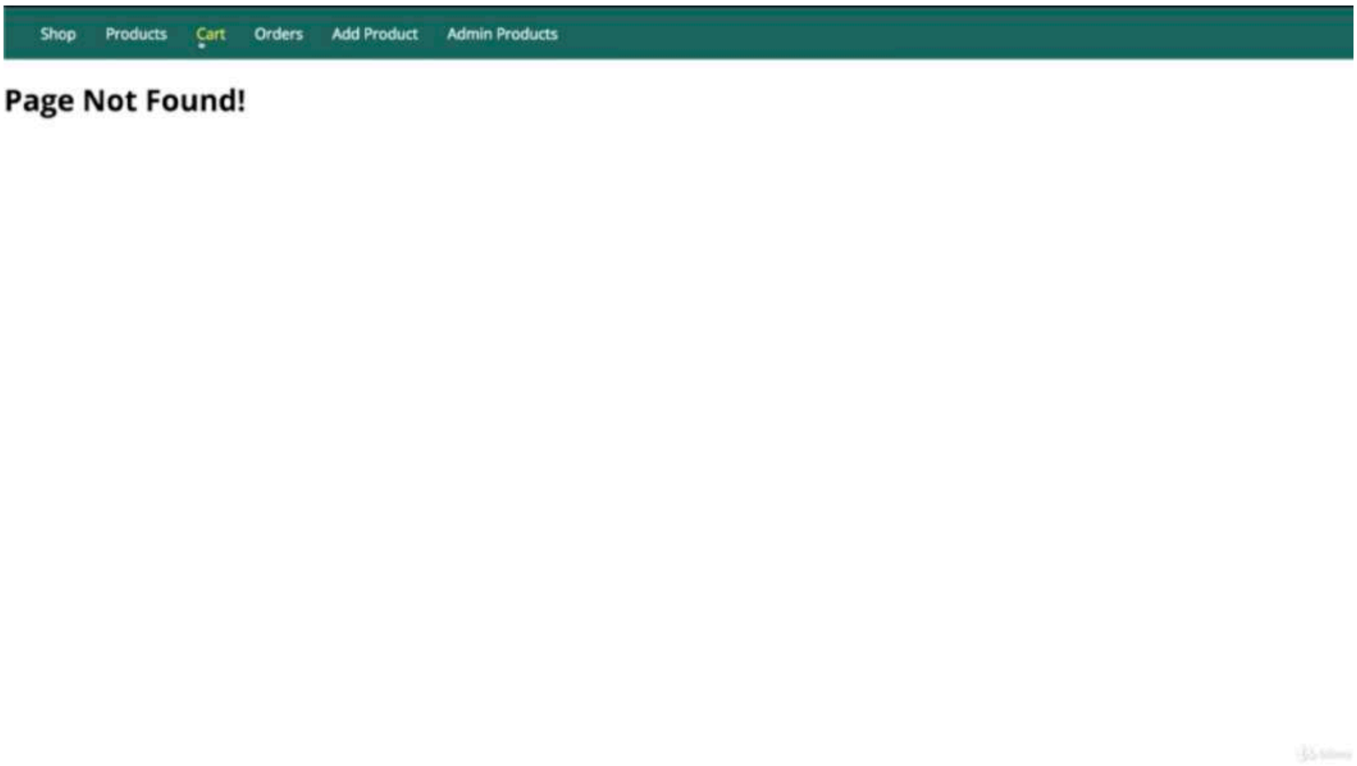
* Chapter 222: Clearing The Cart After Storing An Order

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

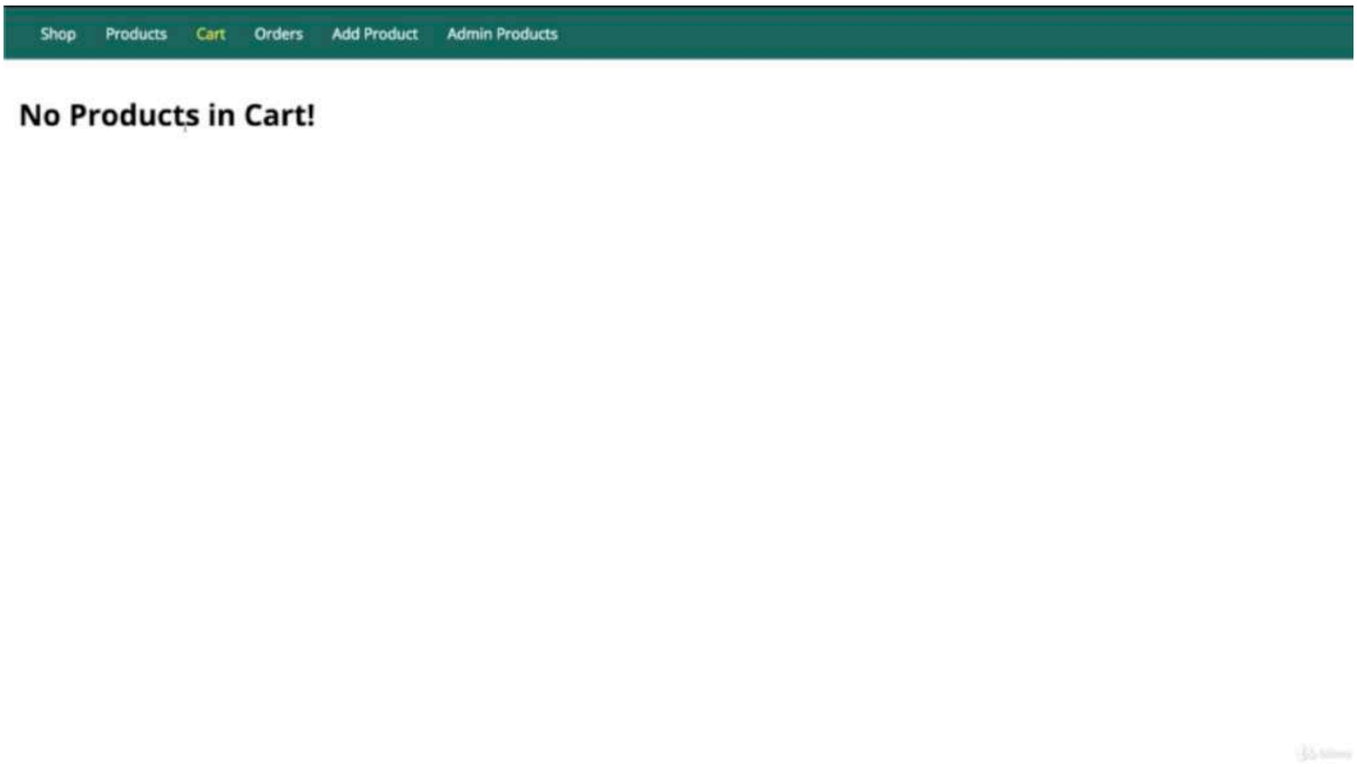
![](images/222-clearing-the-cart-after-storing-an-order-1.png)
![](images/222-clearing-the-cart-after-storing-an-order-2.png)
![](images/222-clearing-the-cart-after-storing-an-order-3.png)
![](images/222-clearing-the-cart-after-storing-an-order-4.png)
![](images/222-clearing-the-cart-after-storing-an-order-5.png)


Shop
Products
Cart
Orders
Add Product
Admin Products
A nice Book
$ 29.99
You should not miss that!
Details
Add to Cart
Second Product
Second Product
$ 1222
fdasfsa
Details
Add to Cart

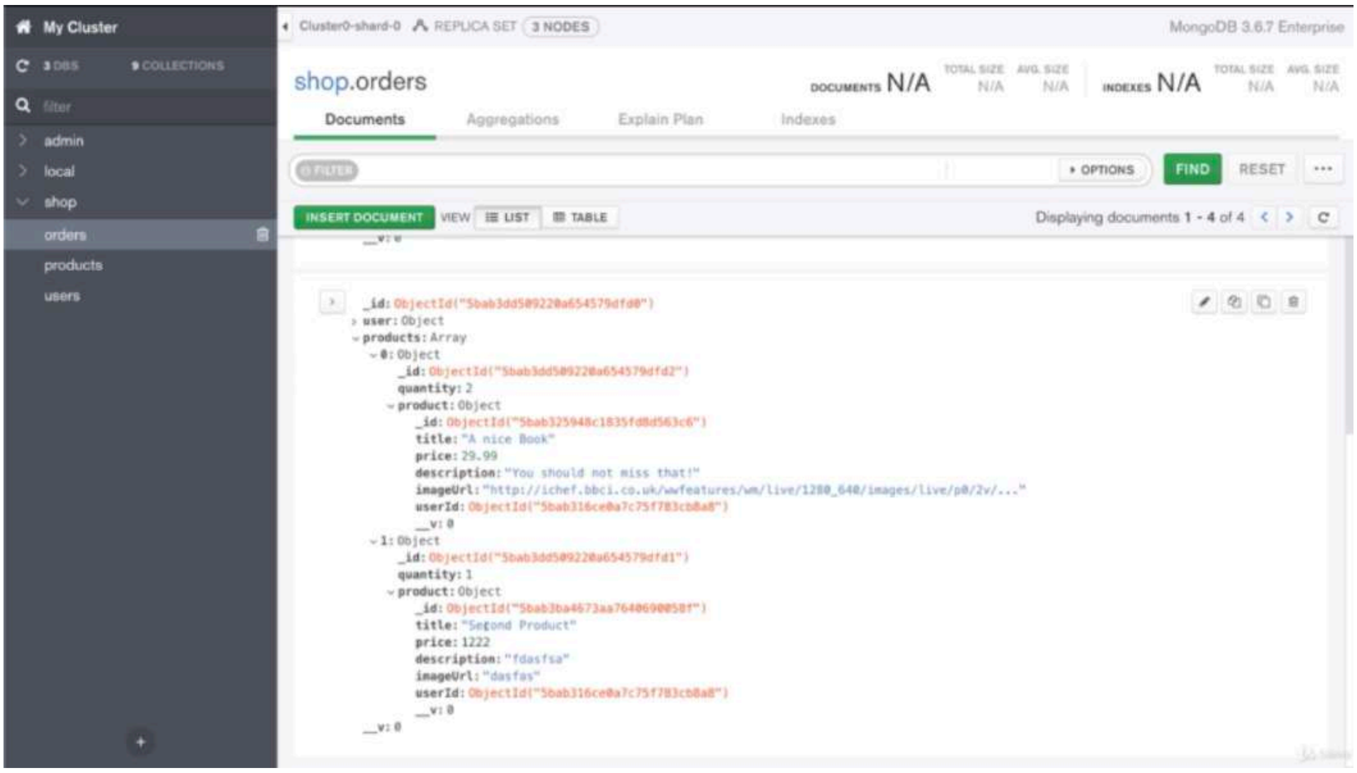

Shop
Products
Cart
Orders
Add Product
Admin Products
A nice Book
Quantity: 2
Delete
Second Product
Quantity: 1
Delete
Order Now!



Page Not Found!

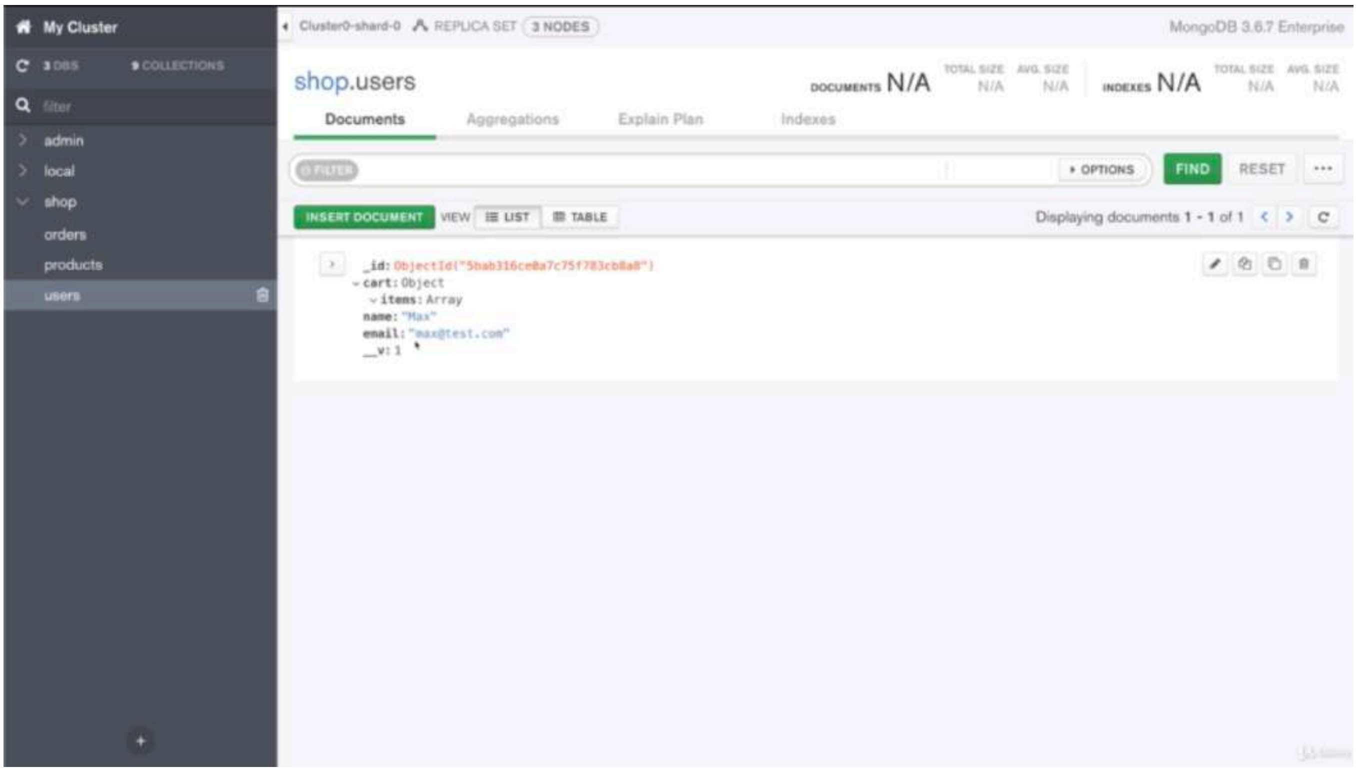


No Products in Cart!



```
_id: ObjectId("5bab3dd509220a654579dfd0")
> user: Object
v products: Array
  v 0: Object
      _id: ObjectId("5bab3dd509220a654579dfd2")
      quantity: 2
    v product: Object
        _id: ObjectId("5bab325948c1835fd0d563c6")
        title: "A nice Book"
        price: 29.99
        description: "You should not miss that!"
        imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
        userId: ObjectId("5bab316ce0a7c75f783cb8a8")
        __v: 0
  v 1: Object
      _id: ObjectId("5bab3dd509220a654579dfd1")
      quantity: 1
    v product: Object
        _id: ObjectId("5bab3ba4673aa7640690058f")
        title: "Second Product"
        price: 1222
        description: "fdasfsa"
        imageUrl: "dasfas"
        userId: ObjectId("5bab316ce0a7c75f783cb8a8")
        __v: 0
__v: 0
```

- this new order should have 2 products. one with quantity 2, that was our 'nice book' and one with quantity 1, that was the 'second product'
![](images/222-clearing-the-cart-after-storing-an-order-6.png)



```
_id: ObjectId("5bab316ce0a7c75f783cb8a8")
v cart: Object
  v items: Array
name: "Max"
email: "max@test.com"
__v: 1
```

- and in the users, the cart is empty.

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

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

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

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      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
    .removeFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  req.user
  .populate('cart.items.productId')
  .execPopulate()
  .then(user => {
    console.log(user.cart.items)
    const products = user.cart.items.map(i => {

      return { quantity: i.quantity, product: { ...i.productId._doc } }
    })
    const order = new Order({
      user: {
        name: req.user.name,
        userId: req.user
      },
      products: products
    })
    order.save()
  })
  .then(result => {
      return req.user.clearCart()
  })
  .then(() => {
    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));
};

```

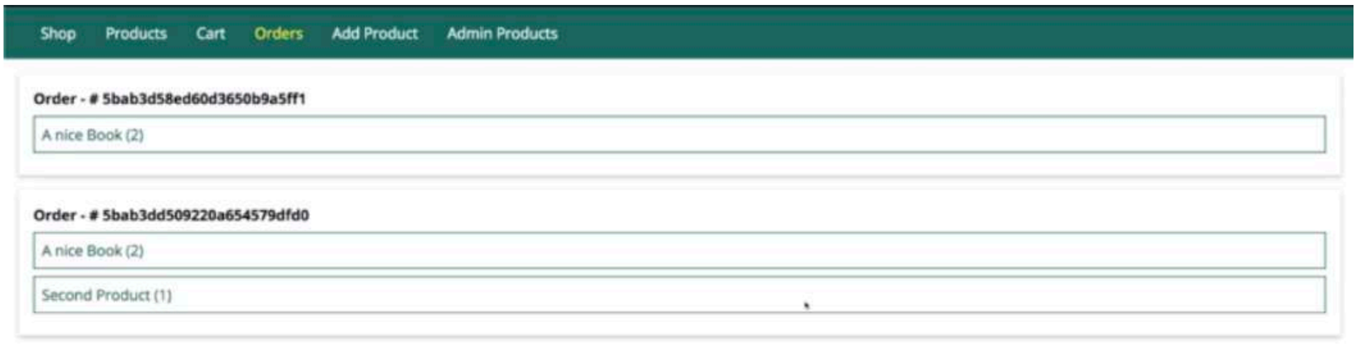
```
//./models/user.js

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  cart: {
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }]
  }
})

userSchema.methods.addToCart = function(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: product._id,
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  }
  this.cart = updatedCart
  return this.save()
}

```

```
userSchema.methods.removeFromCart = function(productId){
  const updatedCartItems = this.cart.items.filter(item => {
    return item.productId.toString() !== productId.toString()
  })
  this.cart.items = updatedCartItems
  return this.save()
}

userSchema.methods.clearCart = function(){
  this.cart = { items: [] }
  return this.save()
}

module.exports = mongoose.model('User', userSchema)

```

* Chapter 223: Getting & Displaying The Orders

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

![](images/223-getting-and-displaying-the-orders-1.png)



Order - # 5bab3d58ed60d3650b9a5ff1

A nice Book (2)

Order - # 5bab3dd509220a654579dfd0

A nice Book (2)

Second Product (1)

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

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

exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      console.log(products)
```

```javascript
      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.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.find()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items
      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)
```

```javascript
    .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
    .removeFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};

exports.postOrder = (req, res, next) => {
  req.user
  .populate('cart.items.productId')
  .execPopulate()
  .then(user => {
    console.log(user.cart.items)
    const products = user.cart.items.map(i => {

      return { quantity: i.quantity, product: { ...i.productId._doc } }
    })
    const order = new Order({
      user: {
        name: req.user.name,
        userId: req.user
      },
      products: products
    })
    order.save()
  })
  .then(result => {
      return req.user.clearCart()
  })
  .then(() => {
    res.redirect('/orders');
  })
  .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  /**i can find all orders
   * and let's have a look at the ./models/order.js
   * 'userId' is nested object in 'user' object,
   * this nested key is equal to the 'userId' of the logged-in user.
   * so '"user.userId": req.user._id'
   * this is the check i wanna make
   * and this will give me all orders that belong to that user.
   *
   *
   */
  Order
    .find({ 'user.userId': req.user._id })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};

```

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

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const userSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  cart: {
    items: [{
      productId: {
        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }]
  }
})

userSchema.methods.addToCart = function(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: product._id,
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  }
  this.cart = updatedCart
  return this.save()
}

userSchema.methods.removeFromCart = function(productId){
  const updatedCartItems = this.cart.items.filter(item => {
    return item.productId.toString() !== productId.toString()
  })
  this.cart.items = updatedCartItems
  return this.save()
}

userSchema.methods.clearCart = function(){
  this.cart = { items: [] }
  return this.save()
}

module.exports = mongoose.model('User', userSchema)

```

```
// ./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.products.forEach(p => { %>
                                    <!--inside here, we have the product
                                    and there we have a nested product field
                                    so we could also name this just 'p' to avoid confusion
                                    this will be 'p'

                                    we have the product field with the title
                                    but directly on the top level 'p' object,
                                    so directly in the object that is stored in the products array,
                                    we have the quantity
                                    so we can still access 'p.quantity' directly on 'p'
                                    which is the part directly in order products
                                    but then the product data itself is nested in one additional embedded document product
                                    -->
                                    <li class="orders__products-item"><%= p.product.title %> (<%= p.quantity %>)</li>
                                <% }); %>
                            </ul>
                        </li>
                    <% }); %>
                </ul>
            <% } %>
        </main>
        <%- include('../includes/end.ejs') %>
```