

15. Adding Authentication

* Chapter 246: Module Introduction



What's In This Module?

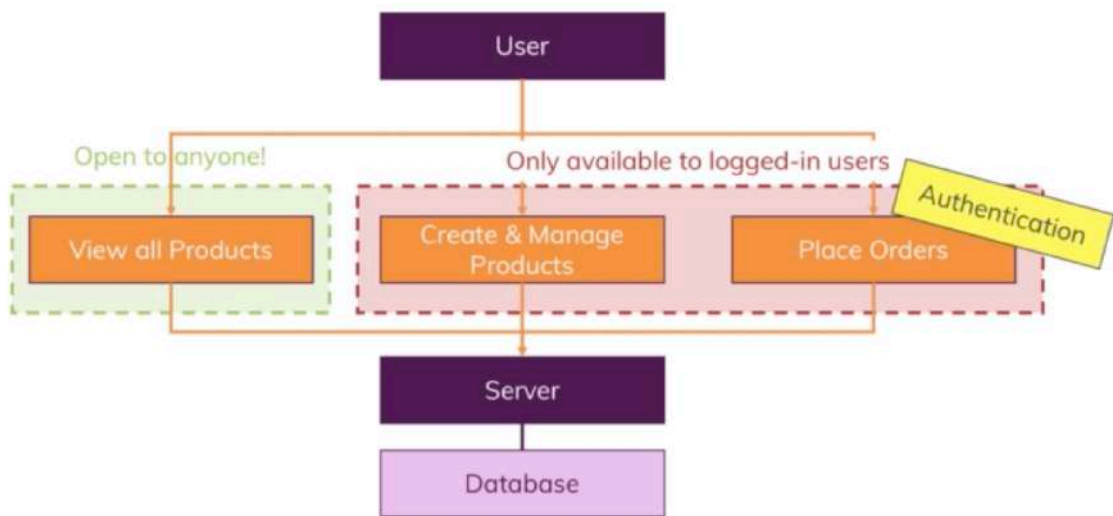
What exactly is "Authentication"?

Storing & Using Credentials

Protecting Routes

* Chapter 247: What Is Authentication?

What is "Authentication"?

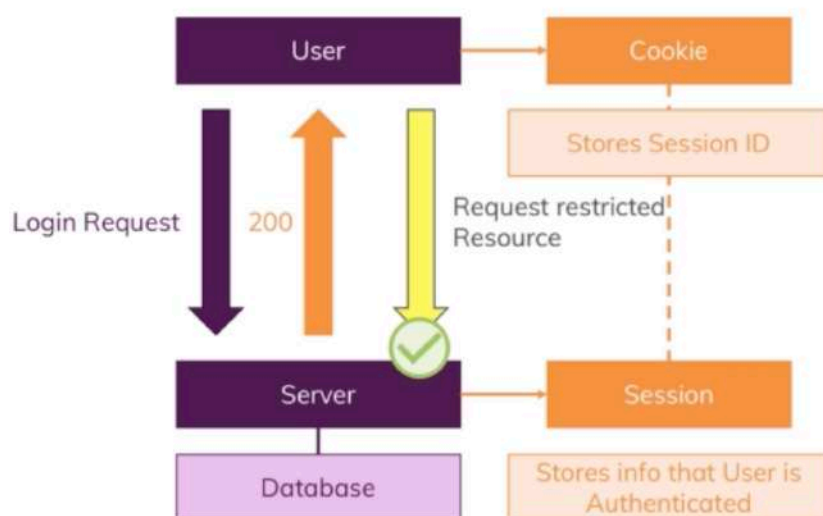


Udemy

- The idea behind authentication is that all these actions are available to every user of our application. i don't mean logged-in user, i mean a person visiting our page, visiting localhost:3000

* Chapter 248: How Is Authentication Implemented

How is Authentication Implemented?



Udemy

- we check whether that email and password combination is valid, whether we have a user with that email and that password in our database.
- if that is the case, we create a session for this user and this session then identifies this user. this is required because otherwise without a session, even if we find out that the credentials are valid, for the next request the

user would be logged out again because requests interact separated from each other, they don't know anything about each other. we need a session to connect them.

- then server send 200 response which means a success response and then we store the cookie belonging to that session on the client. so that we established a session

- thereafter, the user is able to visit our restricted routes because this cookie sent with every request, on the server we can connect this cookie to a session and in the session we have the information whether that user is signed in or not.

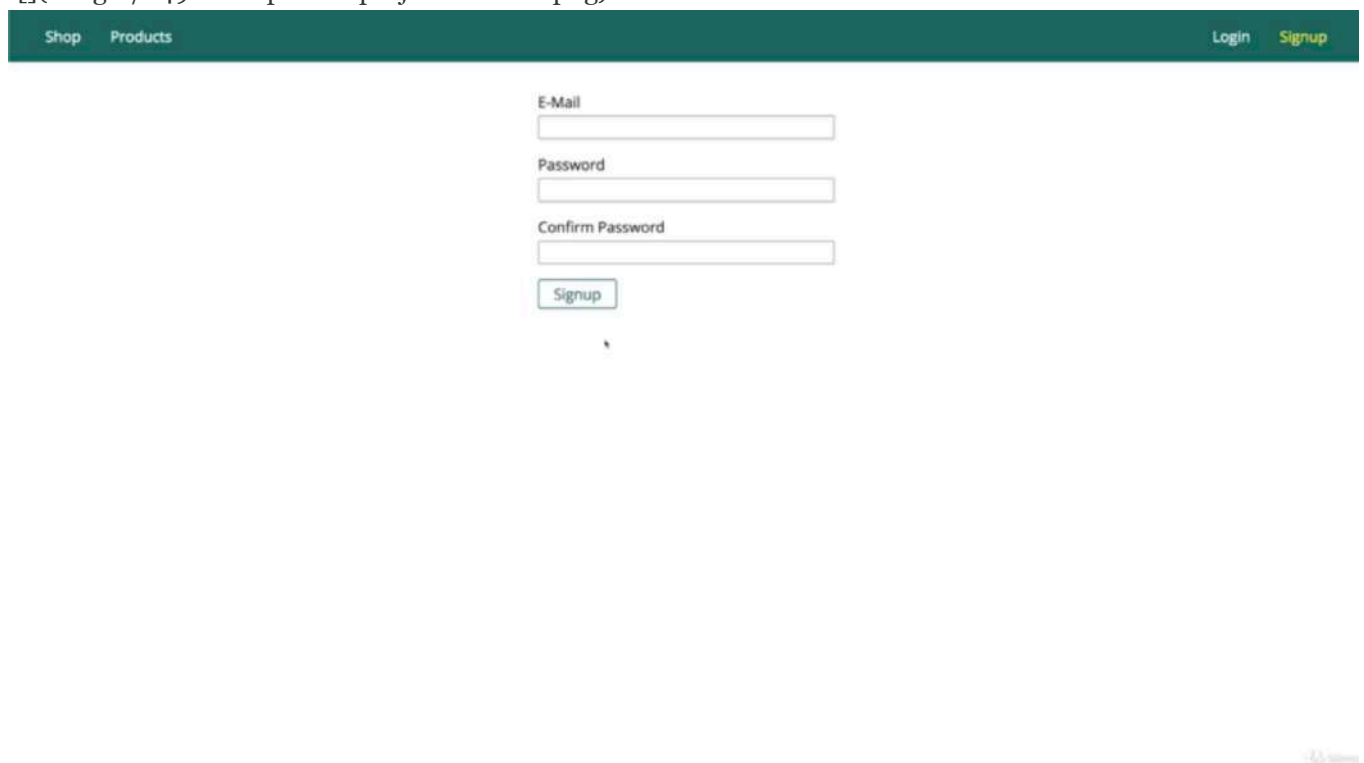
* Chapter 249: Our Updated Project Status

1. update

- changed views folder with new thing

- ./routes/auth.js

- ./controllers/auth.js



```
1 // ./routes/auth.js
2
3 const express = require('express')
4
5 const authController = require('../controllers/auth')
6
7 const router = express.Router()
8
9 router.get('/login', authController.getLogin)
10
11 router.get('/signup', authController.getSignup)
12
13 router.post('/login', authController.postLogin)
14
15 router.post('/signup', authController.postSignup)
16
```

```

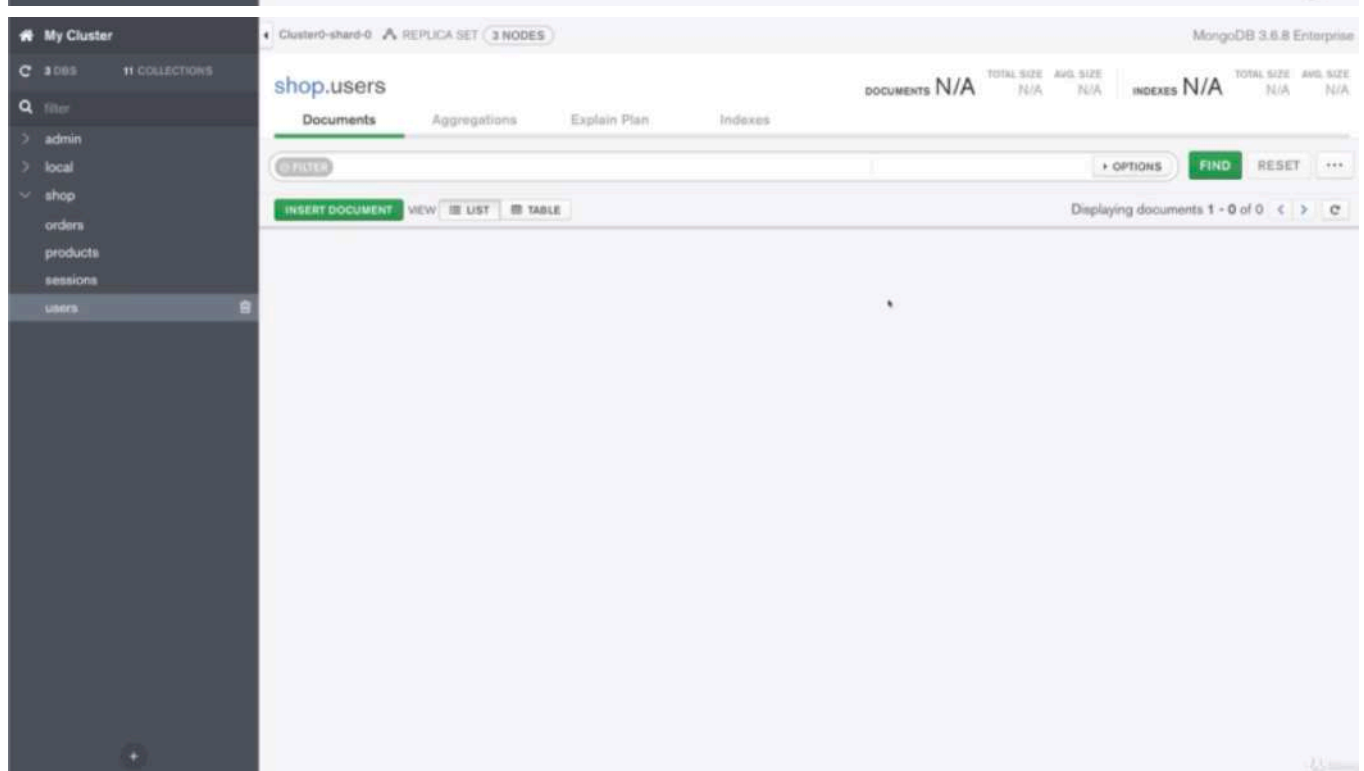
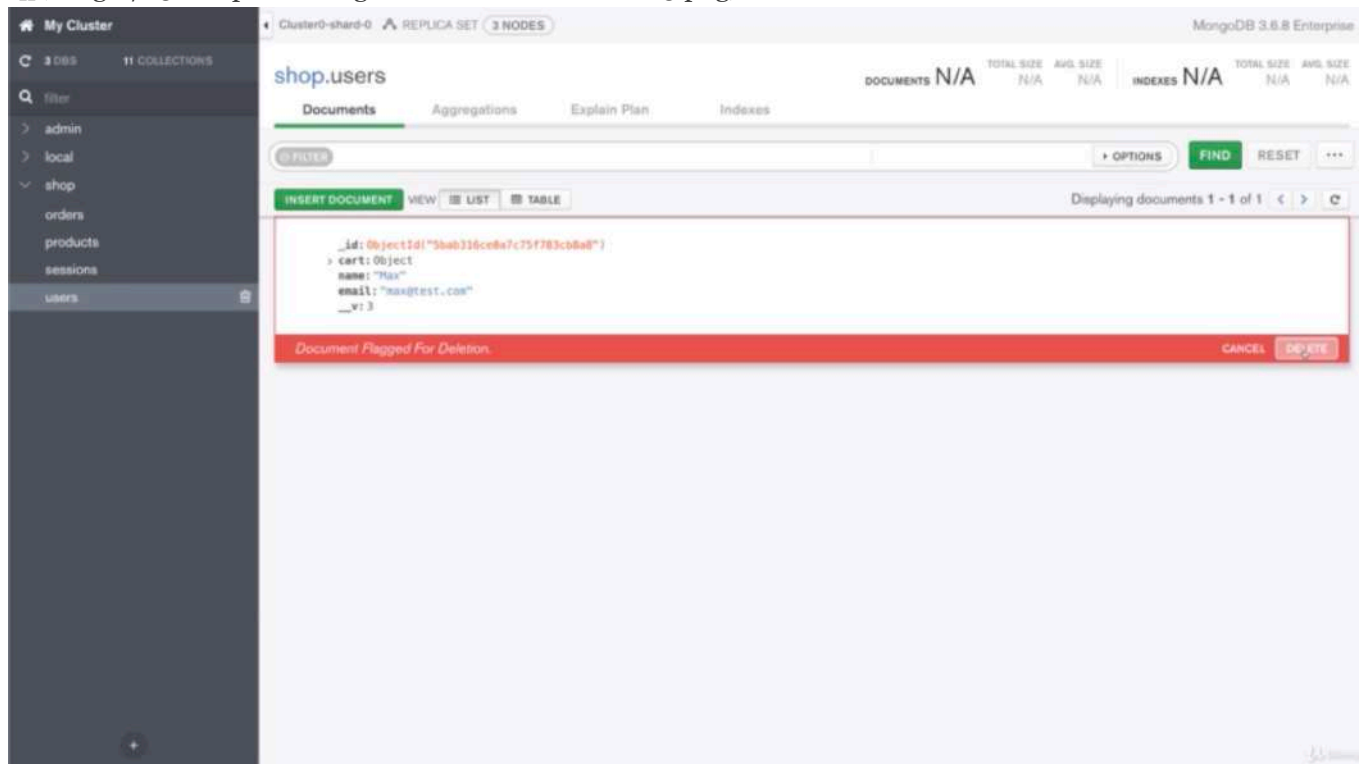
17 router.post('/logout', authController.postLogout)
18
19 module.exports = router

1 //./controllers/auth.js
2
3 const User = require('../models/user');
4
5 exports.getLogin = (req, res, next) => {
6   res.render('auth/login', {
7     path: '/login',
8     pageTitle: 'Login',
9     isAuthenticated: false
10  });
11 };
12
13 exports.getSignup = (req, res, next) => {
14   res.render('auth/signup', {
15     path: '/signup',
16     pageTitle: 'Signup',
17     isAuthenticated: false
18  });
19 };
20
21 exports.postLogin = (req, res, next) => {
22   User.findById('5cbb2b2c80bd7193adb9eeeb')
23     .then(user => {
24       req.session.isLoggedIn = true;
25       req.session.user = user;
26       req.session.save(err => {
27         console.log(err);
28         res.redirect('/');
29       });
30     })
31     .catch(err => console.log(err));
32 };
33
34 exports.postSignup = (req, res, next) => {};
35
36 exports.postLogout = (req, res, next) => {
37   req.session.destroy(err => {
38     console.log(err);
39     res.redirect('/');
40   });
41 };
42

```

* Chapter 250: Implementing An Authentication Flow

1. update
 - ./views/auth/signup.ejs
 - ./controllers/auth.js
 - ./models/user.js
 - app.js



E-Mail

test@test.com

Password

Confirm Password

Signup

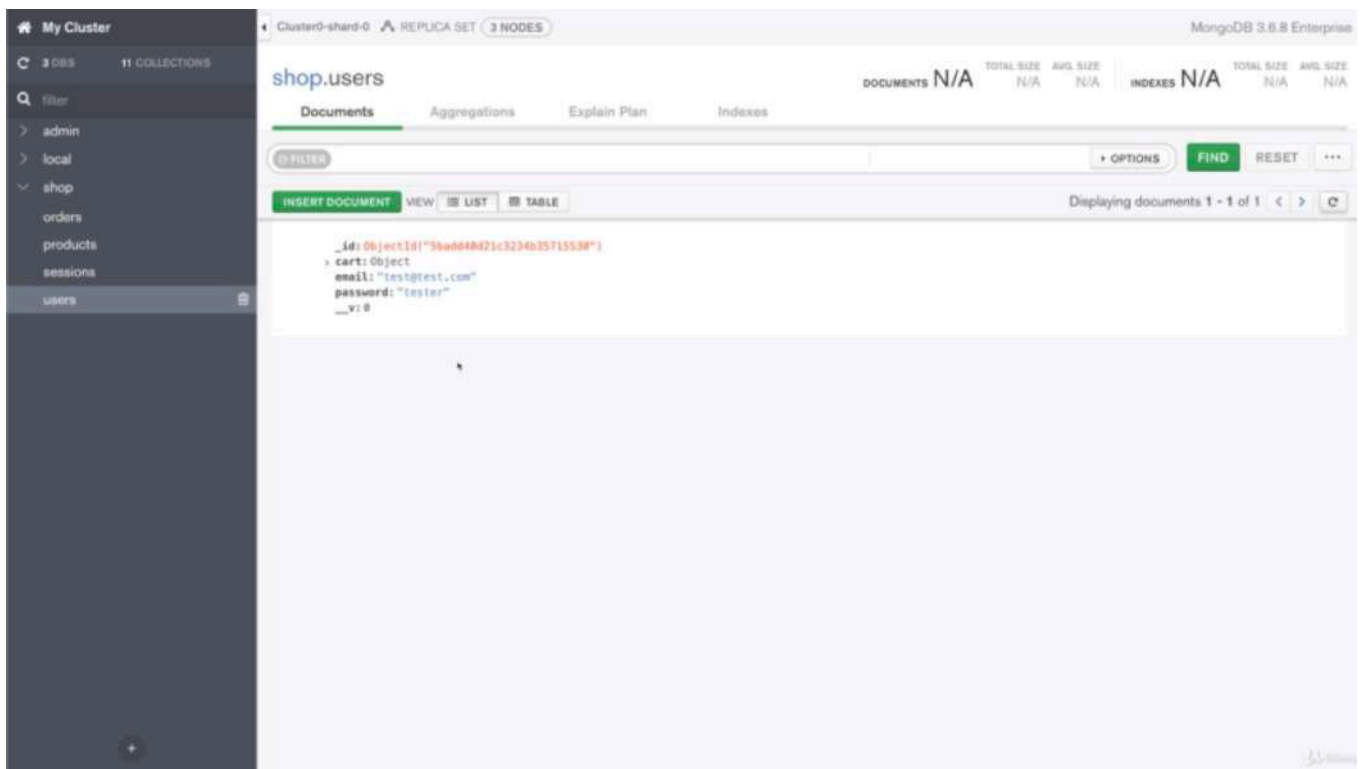


E-Mail

Password

Login





```

1 <!--../views/auth/signup.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4     <link rel="stylesheet" href="/css/forms.css">
5     <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9     <%- include('../includes/navigation.ejs') %>
10
11     <main>
12         <!--in the action now '/signup' send a POST request to reach that sign up the POST
13         signup route
14         which triggers the POST sign up action-->
15         <form class="login-form" action="/signup" method="POST">
16             <div class="form-control">
17                 <label for="email">E-Mail</label>
18                 <input type="email" name="email" id="email">
19             </div>
20             <div class="form-control">
21                 <label for="password">Password</label>
22                 <input type="password" name="password" id="password">
23             </div>
24             <div class="form-control">
25                 <label for="confirmPassword">Confirm Password</label>
26                 <input type="password" name="confirmPassword" id="confirmPassword">
27             </div>
28             <button class="btn" type="submit">Signup</button>
29         </form>
30     </main>
31 <%- include('../includes/end.ejs') %>

```

```

1 //../controllers/auth.js
2
3 const User = require('../models/user');

```

```

4
5 exports.getLogin = (req, res, next) => {
6   res.render('auth/login', {
7     path: '/login',
8     pageTitle: 'Login',
9     isAuthenticated: false
10  });
11 };
12
13 exports.getSignup = (req, res, next) => {
14   res.render('auth/signup', {
15     path: '/signup',
16     pageTitle: 'Signup',
17     isAuthenticated: false
18   });
19 };
20
21 exports.postLogin = (req, res, next) => {
22   User.findById('5cbb2b2c80bd7193adb9eeeb')
23     .then(user => {
24       req.session.isLoggedIn = true;
25       req.session.user = user;
26       req.session.save(err => {
27         console.log(err);
28         res.redirect('/');
29       });
30     })
31     .catch(err => console.log(err));
32 };
33
34 exports.postSignup = (req, res, next) => {
35   /**we have to make sure how these inputs are named in the ./views/auth/signup.ejs
36    * because you retrieve the values on req.body by these name.
37    *
38    */
39   const email = req.body.email
40   const password = req.body.password
41   const confirmPassword = req.body.confirmPassword
42   /**you could create an index in the MongoDB database on your email field
43    * and give that index the unique property.
44    *
45    * the alternative is that you try to find a user with that e-mail.
46    * for that we will use our user model, a mongoose user model
47    * we will use 'findOne()' because we already know we don't wanna create a user with that
48    user
49    *
50    * 'email' on the right side is the email we are extracting
51    * 'email' on the left side is the email we are looking for in the database.
52    *
53    */
54   User.findOne({email: email})
55     .then(userDoc => {
56       if(userDoc){
57         return res.redirect('/signup')
58       }
59       const user = new User({

```



```

59     email: email,
60     password: password,
61     cart: { items: [] }
62   })
63   /**we use 'return' so that we can chain another 'then()' block
64    * which will be called once this 'save()' action completed.
65    */
66   return user.save()
67 })
68 .then(result => {
69   res.redirect('/login')
70 })
71 .catch(err => {
72   console.log(err)
73 })
74 };
75
76 exports.postLogout = (req, res, next) => {
77   req.session.destroy(err => {
78     console.log(err);
79     res.redirect('/');
80   });
81 };
82

```

```

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

```

```

33     return cp.productId.toString() === product._id.toString();
34 });
35 let newQuantity = 1;
36 const updatedCartItems = [...this.cart.items];
37 if (cartProductIndex >= 0) {
38     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
39     updatedCartItems[cartProductIndex].quantity = newQuantity;
40 } else {
41     updatedCartItems.push({
42         productId: product._id,
43         quantity: newQuantity
44     });
45 }
46 const updatedCart = {
47     items: updatedCartItems
48 }
49 this.cart = updatedCart
50 return this.save()
51 }
52
53 userSchema.methods.removeFromCart = function(productId){
54     const updatedCartItems = this.cart.items.filter(item => {
55         return item.productId.toString() !== productId.toString()
56     })
57     this.cart.items = updatedCartItems
58     return this.save()
59 }
60
61 userSchema.methods.clearCart = function(){
62     this.cart = { items: [] }
63     return this.save()
64 }
65
66 module.exports = mongoose.model('User', userSchema)
67

```

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7  const mongoose = require('mongoose')
8  const session = require('express-session')
9  const MongoDBStore = require('connect-mongodb-session')(session)
10
11 const errorController = require('./controllers/error');
12 const User = require('./models/user')
13
14 const MONGODB_URI = 'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-
z3v1k.mongodb.net/shop'
15
16 const app = express();
17 const store = new MongoDBStore({
18     uri: MONGODB_URI,
19     collection: 'sessions'
20 })

```

```

21
22 app.set('view engine', 'ejs');
23 app.set('views', 'views');
24
25 const adminRoutes = require('./routes/admin');
26 const shopRoutes = require('./routes/shop');
27 const authRoutes = require('./routes/auth');
28
29 /**your session data will be stored in there */
30 app.use(bodyParser.urlencoded({ extended: false }));
31 app.use(express.static(path.join(__dirname, 'public')));
32 app.use(
33   session({
34     secret: 'my secret',
35     resave: false,
36     saveUninitialized: false,
37     store: store
38   })
39 )
40
41 app.use((req, res, next) => {
42   if(!req.session.user){
43     return next()
44   }
45   User.findById(req.session.user._id)
46   .then(user => {
47     req.user = user
48     next()
49   })
50   .catch(err => console.log(err));
51 })
52
53 app.use((req, res, next) => {
54   User.findById('5cbb2b2c80bd7193adb9eeeb')
55   .then(user => {
56     req.user = user
57     next();
58   })
59   .catch(err => console.log(err));
60 });
61
62 app.use('/admin', adminRoutes);
63 app.use(shopRoutes);
64 app.use(authRoutes)
65
66 app.use(errorController.get404);
67
68 mongoose
69   .connect(
70     MONGODB_URI
71   )
72   .then(result => {
73     /**i will get rid of these code
74     * because we have a real user creation flow
75     * so we don't need that dummy user.
76     */

```

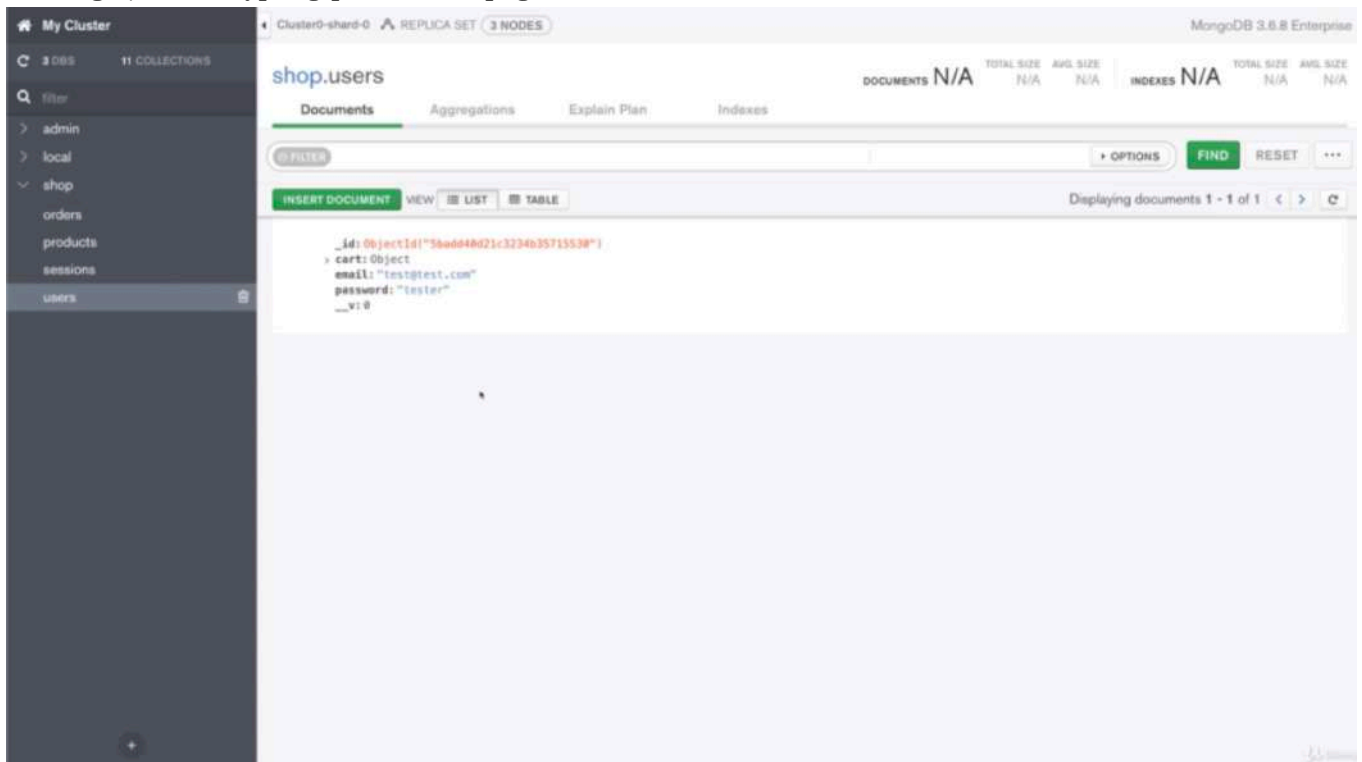
```

77  /*
78  User.findOne()
79  .then(user => {
80    if(!user){
81      const user = new User({
82        name: 'Max', email: 'max@test.com', cart: { items: [] }
83      })
84      user.save()
85    }
86  })
87  */
88  app.listen(3000)
89  })
90  .catch(err => {
91    console.log(err)
92  })

```

* Chapter 251: Encrypting Passwords

1. update
- ./controllers/auth.js



- the problem is that we store the password in plain text. so what we should do is we should encrypt that password. we should hash it in a way that is not reversible, where people can't construct the password from. so that even if you get access to the database, you might be able to see the e-mail but you are not able to see the password, the real password that belongs to the email.


```
34 const password = req.body.password;
35 const confirmPassword = req.body.confirmPassword;
36 User.findOne({ email: email })
37   .then(userDoc => {
38     if (userDoc) {
39       return res.redirect('/signup');
40     }
41     const user = new User({
42       email: email,
43       password: password,
44       cart: { items: [] }
45     });
46     return user.save();
47   })
48   .then(result => {
49     res.redirect('/login');
50   })
51   .catch(err => {
52     console.log(err);
53   });
```

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$
Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$
Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$ npm install --save bcryptjs

- the package 'bcryptjs' help us with encryption and that will help us with encrypting the password.



E-Mail
test2@test.com

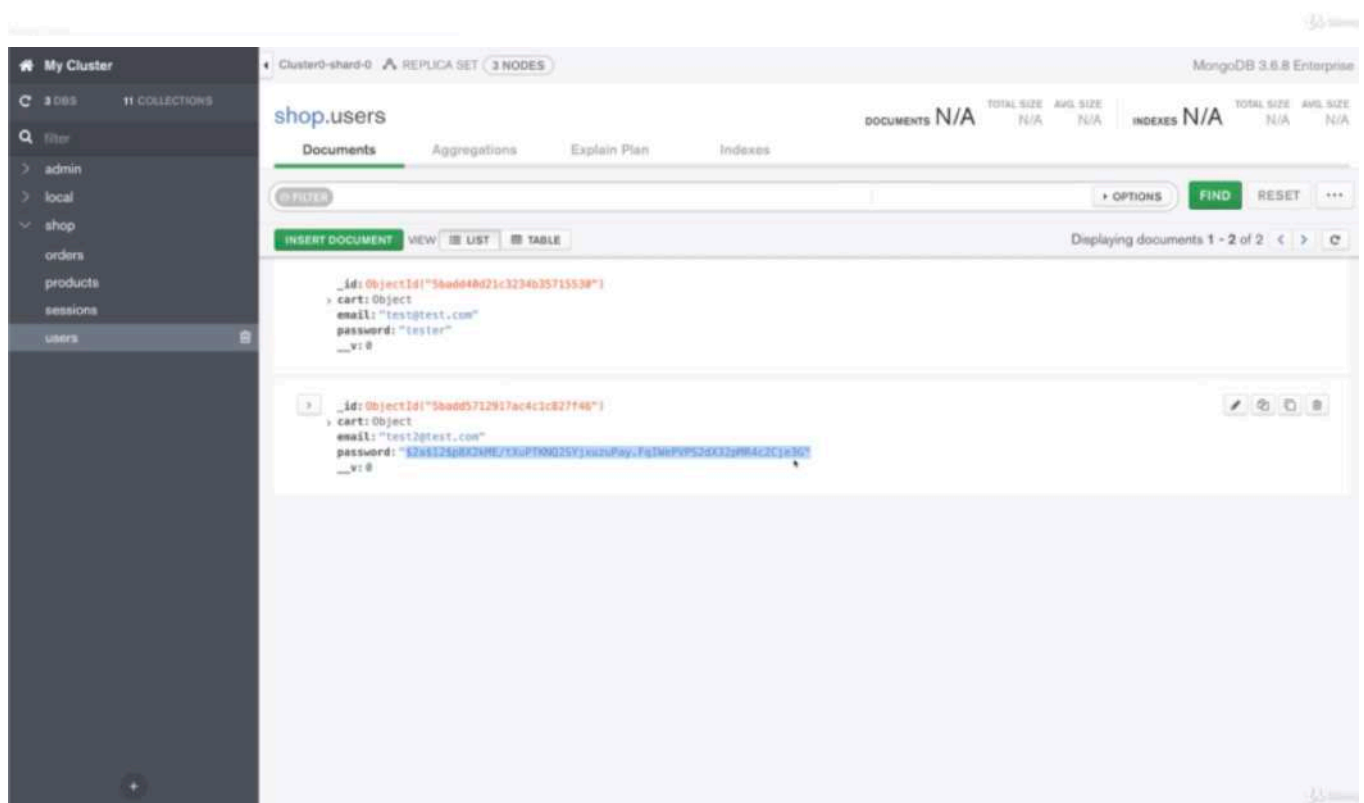
Password

Confirm Password

Signup

E-Mail

Password



- this is the hashed value and the important thing is you can't reconstruct the password i use. you can't decrypt this. this is why we don't encrypt the e-mail because we won't be able to decrypt this. so if we need to send messages to that e-mail, that would not work if we encrypt the e-mail as well because we could not revert that. so we need to store the e-mail like this but the password is secured. it's not readable.

My Cluster Cluster0-shard-0 REPLICASET 3 NODES MongoDB 3.6.8 Enterprise

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

Documents Aggregations Explain Plan Indexes

0 FILTER + OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 2 of 2

```

{ "_id": ObjectId("5badd48d21c3234b35715538"),
  "cart": Object,
  "email": "test@test.com",
  "password": "tester",
  "__v": 0 }

```

Document Flagged For Deletion. CANCEL DELETE

```

{ "_id": ObjectId("5badd5712917ac4c3c827f46"),
  "cart": Object,
  "email": "test2@test.com",
  "password": "$2e$12$pb8X2NME/1XuPTXNQ25YjxuzuPay.Fq[WePvPS2dX32pR84c2C]e3G",
  "__v": 0 }

```

My Cluster Cluster0-shard-0 REPLICASET 3 NODES MongoDB 3.6.8 Enterprise

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

Documents Aggregations Explain Plan Indexes

0 FILTER + OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```

{ "_id": ObjectId("5badd5712917ac4c3c827f46"),
  "cart": Object,
  "email": "test2@test.com",
  "password": "$2e$12$pb8X2NME/1XuPTXNQ25YjxuzuPay.Fq[WePvPS2dX32pR84c2C]e3G",
  "__v": 0 }

```

```

1 //./controllers/auth.js
2
3 const bcrypt = require('bcryptjs')
4
5 const User = require('../models/user');
6
7 exports.getLogin = (req, res, next) => {
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };

```

```

14
15 exports.getSignup = (req, res, next) => {
16   res.render('auth/signup', {
17     path: '/signup',
18     pageTitle: 'Signup',
19     isAuthenticated: false
20   });
21 };
22
23 exports.postLogin = (req, res, next) => {
24   User.findById('5cbb2b2c80bd7193adb9eeeb')
25     .then(user => {
26       req.session.isLoggedIn = true;
27       req.session.user = user;
28       req.session.save(err => {
29         console.log(err);
30         res.redirect('/');
31       });
32     })
33     .catch(err => console.log(err));
34 };
35
36 exports.postSignup = (req, res, next) => {
37   const email = req.body.email
38   const password = req.body.password
39   const confirmPassword = req.body.confirmPassword
40   User.findOne({email: email})
41     .then(userDoc => {
42       if(userDoc){
43         return res.redirect('/signup')
44       }
45       /**'hash()' method as a first value takes the string which you wanna hash
46        * so in our case 'password'. so we will pass the password here.
47        *
48        * the 2nd argument then is the salt value.
49        * this is specifying how many rounds of hashing will be applied.
50        * the higher the value, the longer it will take but the more secure that will be
51        * currently a value of 12 is accepted as highly secure
52        *
53        * this will generate hash password.
54        * this is asynchronous task and therefore gives us back a promise
55        * so i will return this so that i can chain another 'then()' block
56        */
57       return bcrypt.hash(password, 12)
58     })
59     .then(hashPassword => {
60       const user = new User({
61         email: email,
62         password: hashPassword,
63         cart: { items: [] }
64       })
65       return user.save()
66     })
67     .then(result => {
68       res.redirect('/login')
69     })

```



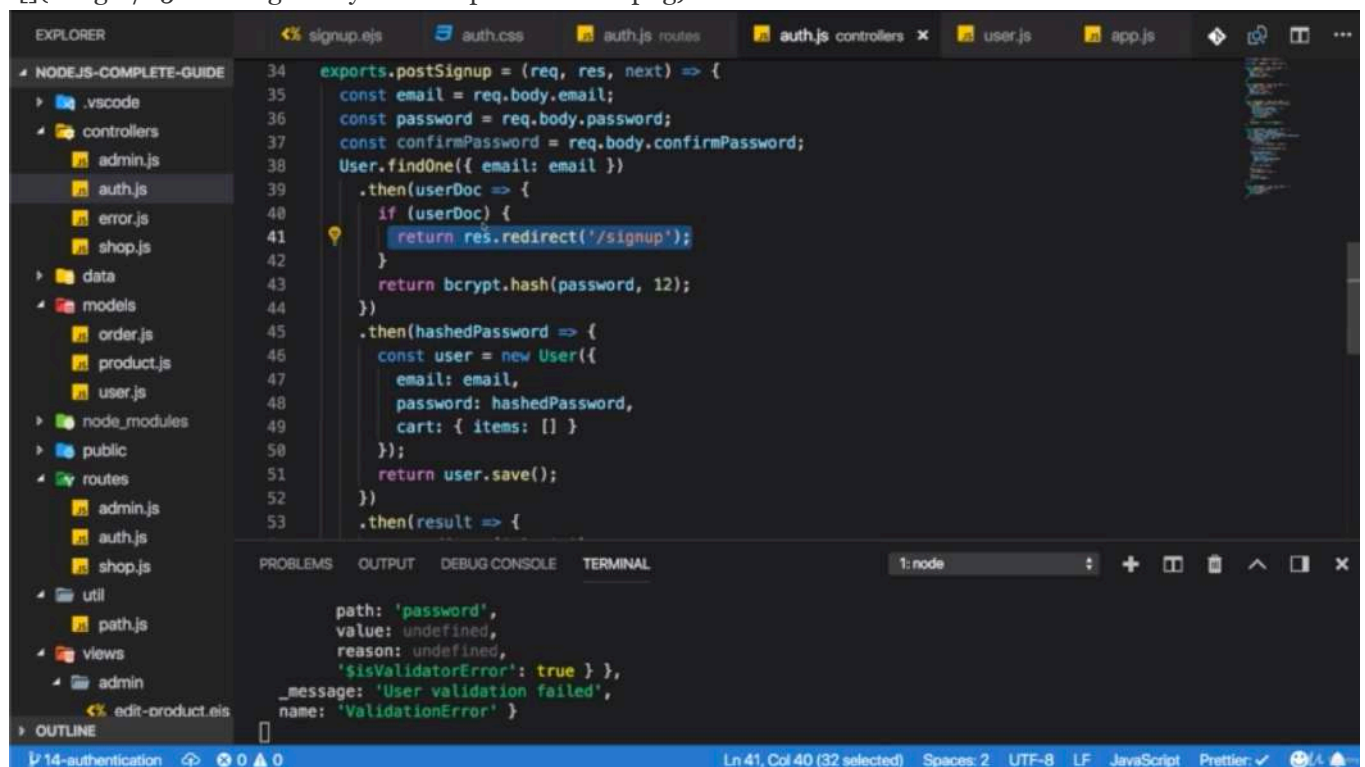
```

70     .catch(err => {
71         console.log(err)
72     })
73 };
74
75 exports.postLogout = (req, res, next) => {
76     req.session.destroy(err => {
77         console.log(err);
78         res.redirect('/');
79     });
80 };
81

```

* Chapter 252: Adding A Tiny Code Improvement

1. update
- /controllers/auth.js



```
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

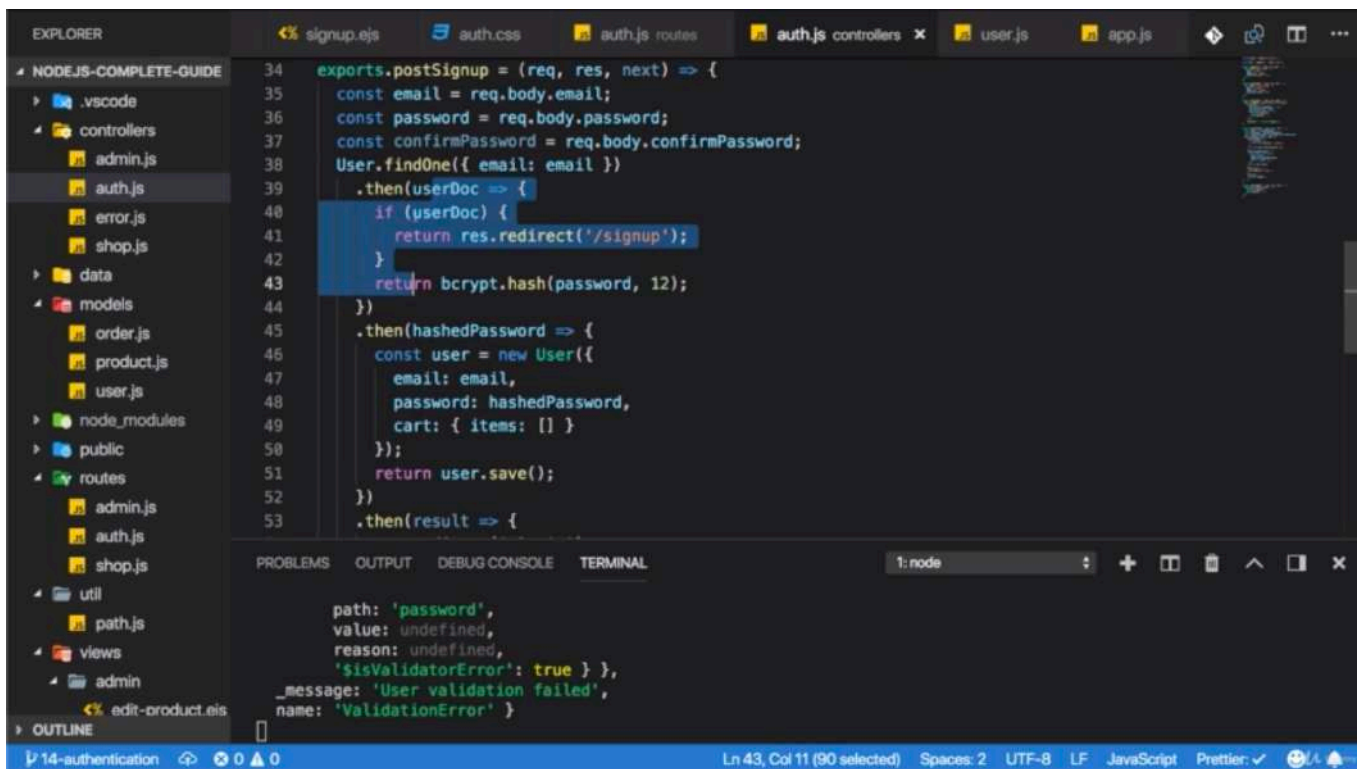
```
path: 'password',
value: undefined,
reason: undefined,
'$isValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- since i return my redirect to '/singup' here and we do this in 'then()' block, this will redirect, this is correct


```
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

```
path: 'password',
value: undefined,
reason: undefined,
'$isValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- but it will still execute the next 'then()' block. this is how promises work.

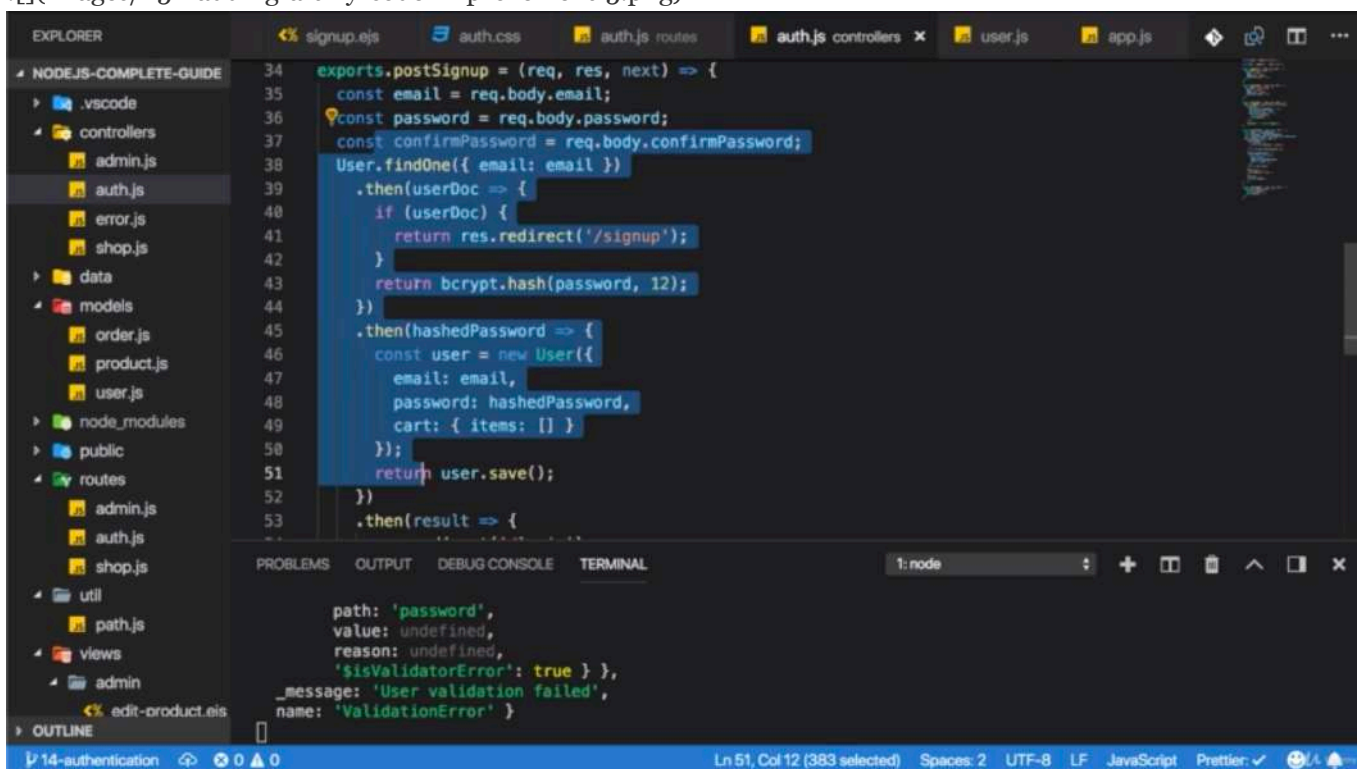


The screenshot shows the VS Code editor with the file explorer on the left displaying a project structure for '14-authentication'. The main editor window shows the file 'signup.js' with the following code:

```
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

The line `return res.redirect('/signup');` on line 41 is highlighted in blue. The bottom status bar shows 'Ln 43, Col 11 (90 selected)'.

- so this code execution in this function does finish because i return 'return res.redirect('/signup')'



The screenshot shows the VS Code editor with the file explorer on the left displaying a project structure for '14-authentication'. The main editor window shows the file 'signup.js' with the following code:

```
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

The line `return res.redirect('/signup');` on line 41 is highlighted in blue. The bottom status bar shows 'Ln 51, Col 12 (383 selected)'.

- but the overall code execution does not.


```
33
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
path: 'password',
value: undefined,
reason: undefined,
'sisValidatorError': true } },
_message: 'User validation failed',
name: 'ValidationError' }
```

- therefore we reached the next 'then()' block even if we redirect.


```
33
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
e-guide/node_modules/mongoose/lib/schematype.js:846:19)
at /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/mongoose/lib/document.js:1825:9
at process._tickCallback (internal/process/next_tick.js:61:11)
message: 'Path `password` is required.',
name: 'ValidatorError',
properties: [Object],
```

- that is why we get this "password" is required' error when we use an existing email


```
33
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

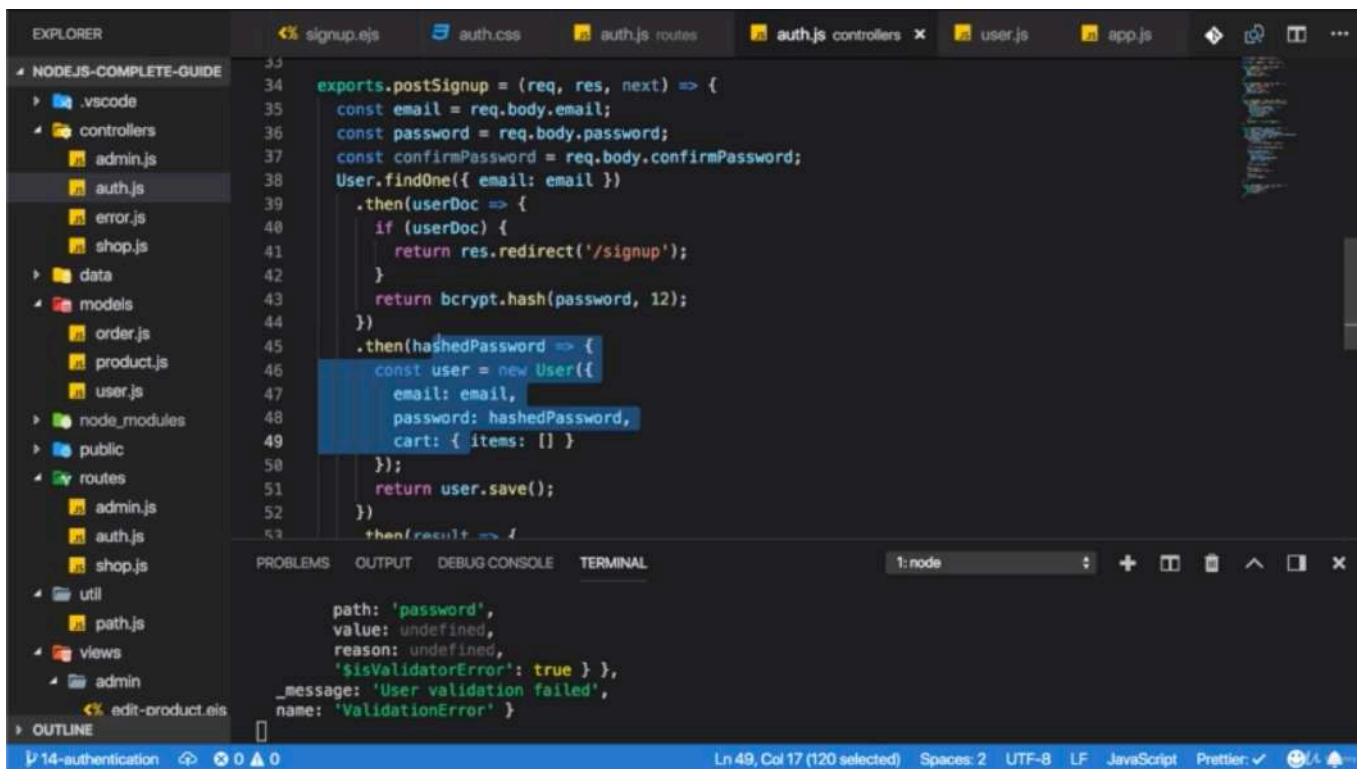
```
path: 'password',
value: undefined,
reason: undefined,
'sisValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- because we reached this 'then()' block and the function in there but hashedPassword will be undefined


```
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashedPassword => {
46       const user = new User({
47         email: email,
48         password: hashedPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
```

```
path: 'password',
value: undefined,
reason: undefined,
'sisValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- because since we return after redirecting, we never execute the hash function. 'bcrypt.hash(password, 12)'



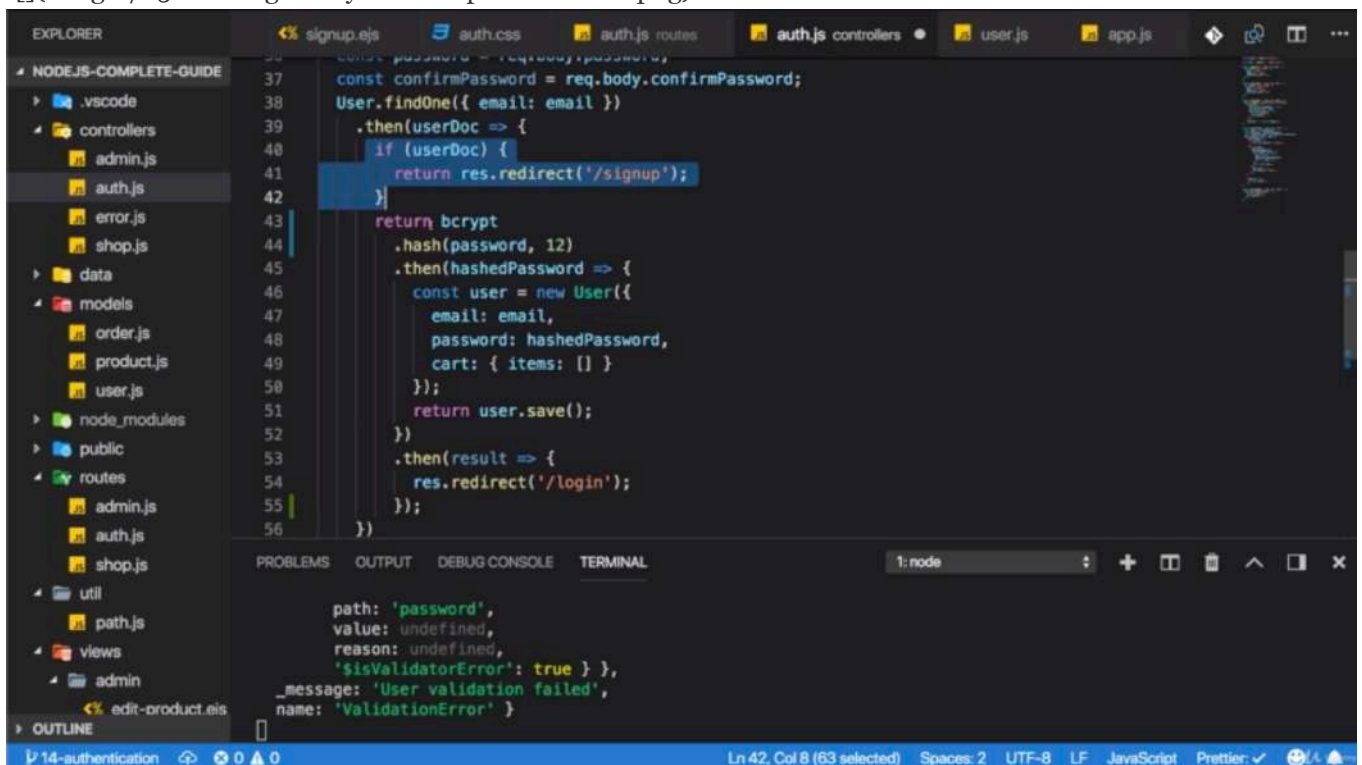
The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure with folders like 'controllers', 'models', 'routes', and 'views'. The 'auth.js' file in the 'controllers' folder is selected. The code in 'auth.js' is as follows:

```
33
34 exports.postSignup = (req, res, next) => {
35   const email = req.body.email;
36   const password = req.body.password;
37   const confirmPassword = req.body.confirmPassword;
38   User.findOne({ email: email })
39     .then(userDoc => {
40       if (userDoc) {
41         return res.redirect('/signup');
42       }
43       return bcrypt.hash(password, 12);
44     })
45     .then(hashPassword => {
46       const user = new User({
47         email: email,
48         password: hashPassword,
49         cart: { items: [] }
50       });
51       return user.save();
52     })
53     .then(result => {
54       res.redirect('/login');
55     });
56 }
```

The terminal at the bottom shows the following error:

```
path: 'password',
value: undefined,
reason: undefined,
'isValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- so we reach this function without the hashedPassword, this is why we get this error.



The screenshot shows the same VS Code editor with the 'auth.js' file. The code is now as follows:

```
37 const password = req.body.password;
38 const confirmPassword = req.body.confirmPassword;
39 User.findOne({ email: email })
40   .then(userDoc => {
41     if (userDoc) {
42       return res.redirect('/signup');
43     }
44     return bcrypt
45       .hash(password, 12)
46       .then(hashPassword => {
47         const user = new User({
48           email: email,
49           password: hashPassword,
50           cart: { items: [] }
51         });
52         return user.save();
53       })
54       .then(result => {
55         res.redirect('/login');
56       });
57   });
58 }
```

The terminal at the bottom shows the same error as before:

```
path: 'password',
value: undefined,
reason: undefined,
'isValidatorError': true },
_message: 'User validation failed',
name: 'ValidationError' }
```

- if you want to avoid this, you can take that code and chain it in here.

- that is a tiny improvement that logically makes more sense here.

```
1 //../controllers/auth.js
2
3 const bcrypt = require('bcryptjs');
4
5 const User = require('../models/user');
6
7 exports.getLogin = (req, res, next) => {
8   res.render('auth/login', {
9     path: '/login',
```

```

10     pageTitle: 'Login',
11     isAuthenticated: false
12   });
13 };
14
15 exports.getSignup = (req, res, next) => {
16   res.render('auth/signup', {
17     path: '/signup',
18     pageTitle: 'Signup',
19     isAuthenticated: false
20   });
21 };
22
23 exports.postLogin = (req, res, next) => {
24   User.findById('5cbe722a2de68db0d19186c1')
25     .then(user => {
26       req.session.isLoggedIn = true;
27       req.session.user = user;
28       req.session.save(err => {
29         console.log(err);
30         res.redirect('/');
31       });
32     })
33     .catch(err => console.log(err));
34 };
35
36 exports.postSignup = (req, res, next) => {
37   const email = req.body.email;
38   const password = req.body.password;
39   const confirmPassword = req.body.confirmPassword;
40   User.findOne({ email: email })
41     .then(userDoc => {
42       if (userDoc) {
43         return res.redirect('/signup');
44       }
45       return bcrypt
46         .hash(password, 12)
47         .then(hashPassword => {
48           const user = new User({
49             email: email,
50             password: hashPassword,
51             cart: { items: [] }
52           });
53           return user.save();
54         })
55         .then(result => {
56           res.redirect('/login');
57         });
58     })
59     .catch(err => {
60       console.log(err);
61     });
62 };
63
64 exports.postLogout = (req, res, next) => {
65   req.session.destroy(err => {

```

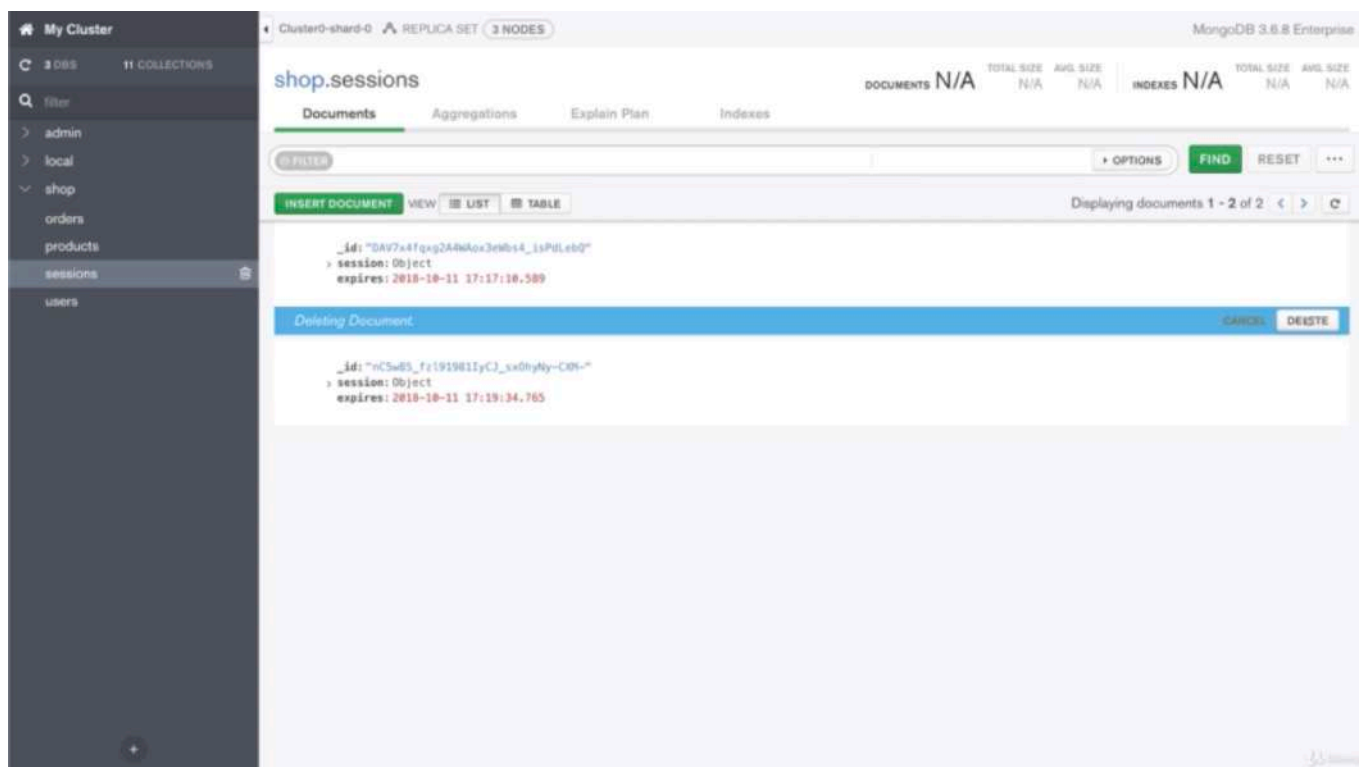
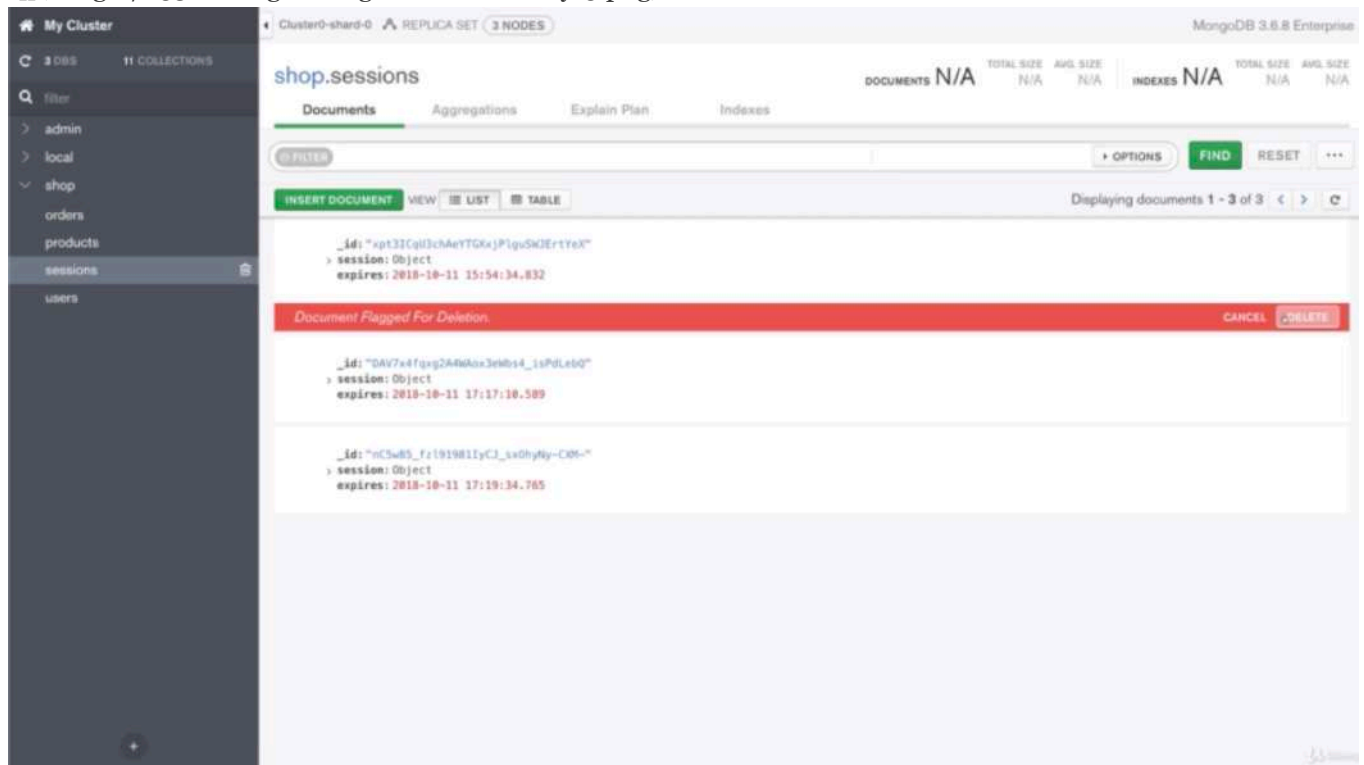
```
66     console.log(err);
67     res.redirect('/');
68   });
69 };
70
```

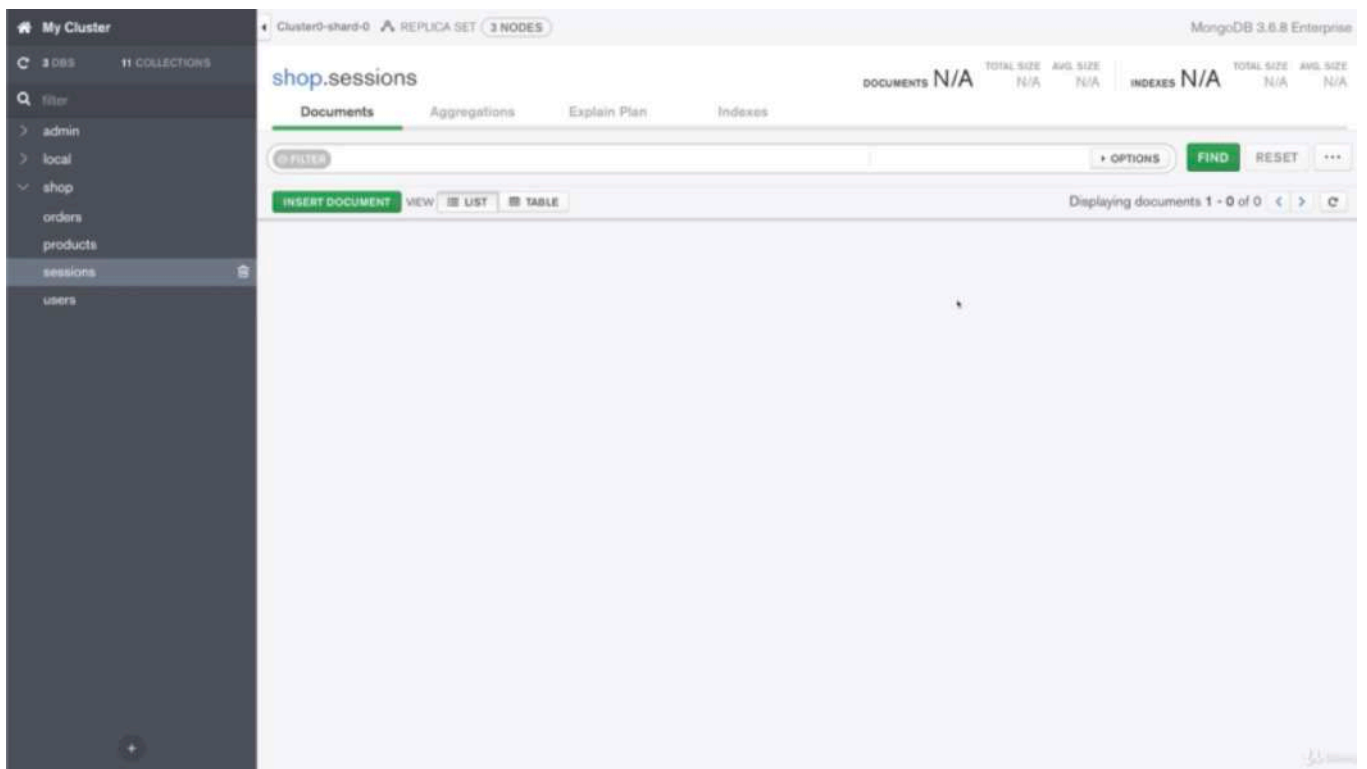
* Chapter 253: Adding The Signin Functionality

1. update
- ./controllers/auth.js

The screenshot shows a web application interface. At the top, there is a dark green header bar with the text "Shop" and "Products" on the left, and "Login" and "Signup" on the right. Below the header, the main content area is light gray. In the center, there is a login form. The form has two input fields: "E-Mail" and "Password". The "E-Mail" field contains the text "test@test.com". Below the "Password" field, there is a "Login" button. The form is styled with a light gray border and a white background.

- if i login, i'm redirected back to login, this didn't seem to succeed.





- so we can go to the sessions and clear all sessions so that we can really prove that no session was created.



E-Mail

test2@test.com

Password

Login

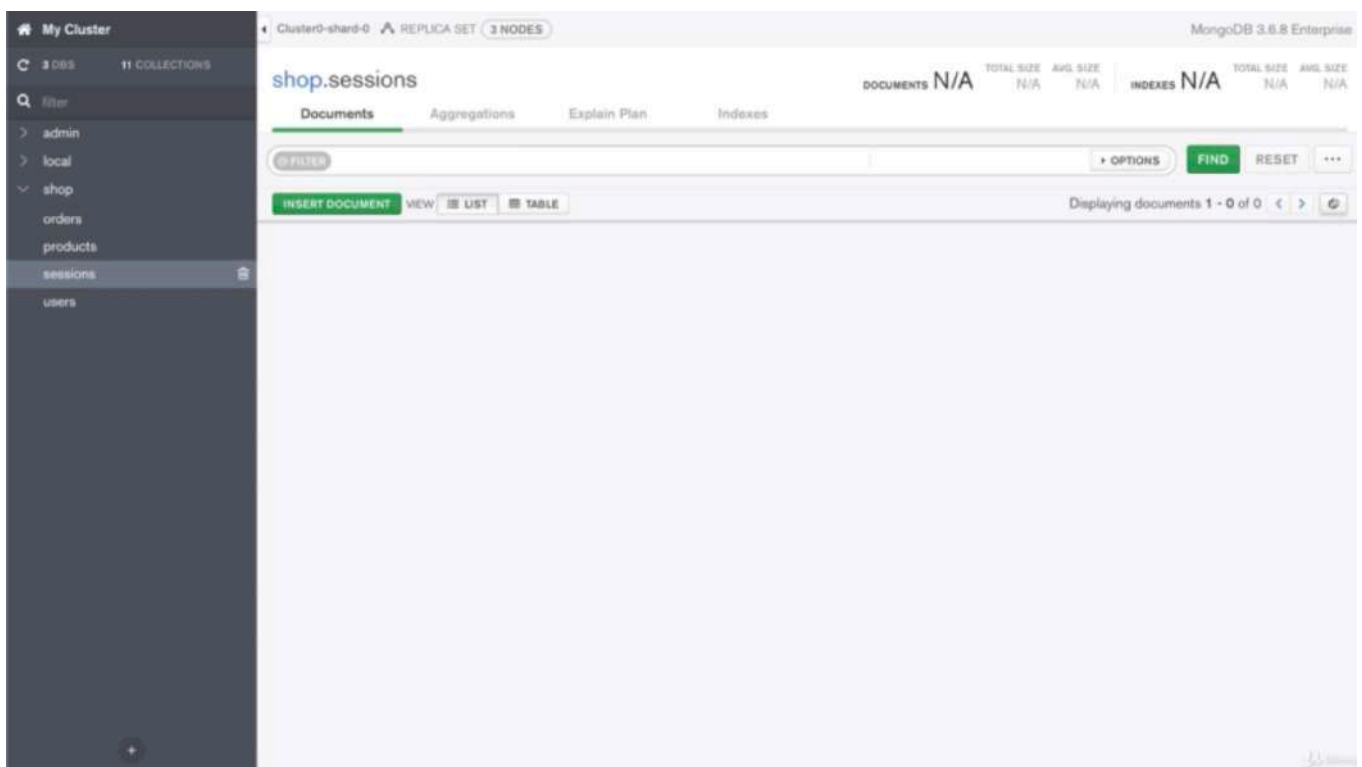
ShopProducts

LoginSignup

E-Mail

Password

Login



- let me enter a valid email but an incorrect password and i'm redirected back and no session was created.

E-Mail

test2@test.com

Password

Login

Signin

A nice Book



\$29.99

You should not miss that!

Details

Add to Cart

dasfdas



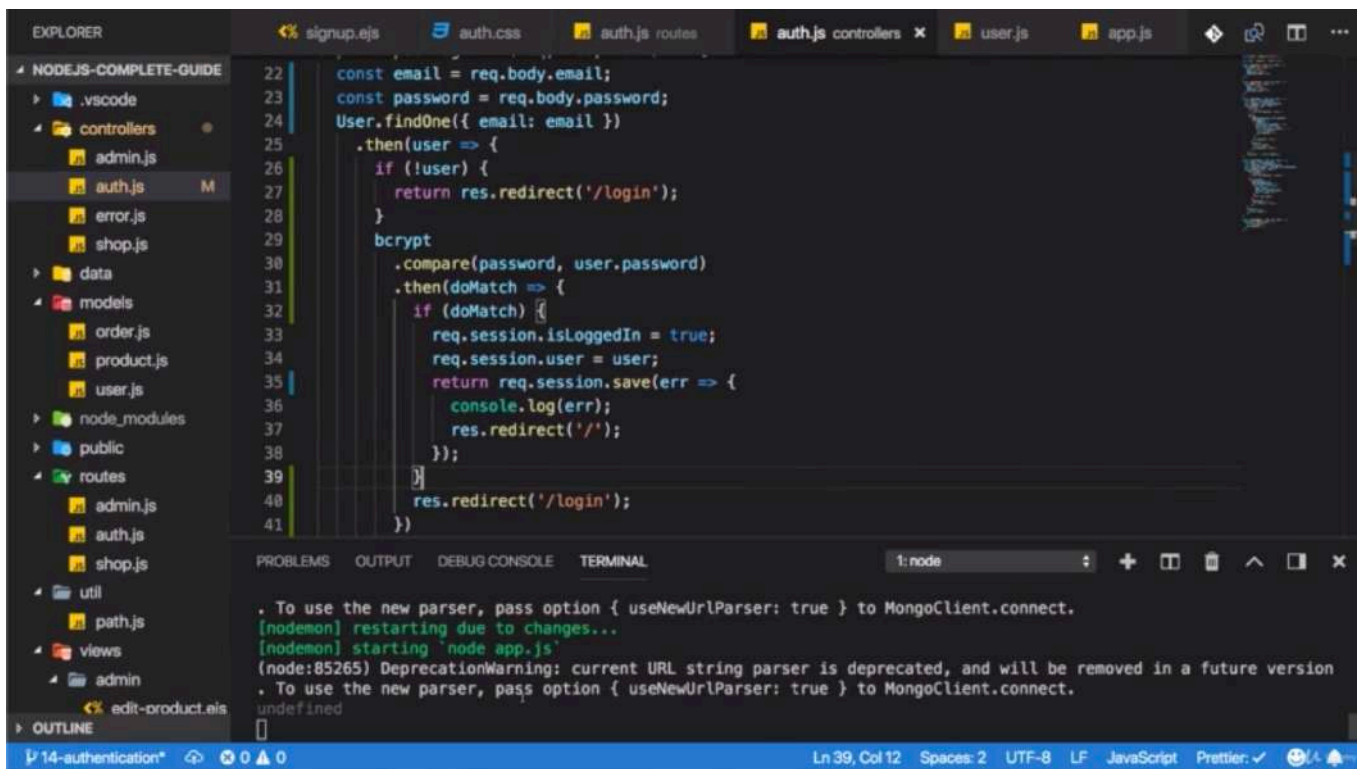
\$12

fadsf

Details

Add to Cart

Signin



```
22 const email = req.body.email;
23 const password = req.body.password;
24 User.findOne({ email: email })
25 .then(user => {
26   if (!user) {
27     return res.redirect('/login');
28   }
29   bcrypt
30     .compare(password, user.password)
31     .then(doMatch => {
32       if (doMatch) {
33         req.session.isLoggedIn = true;
34         req.session.user = user;
35         return req.session.save(err => {
36           console.log(err);
37           res.redirect('/');
38         });
39       }
40       res.redirect('/login');
41     })

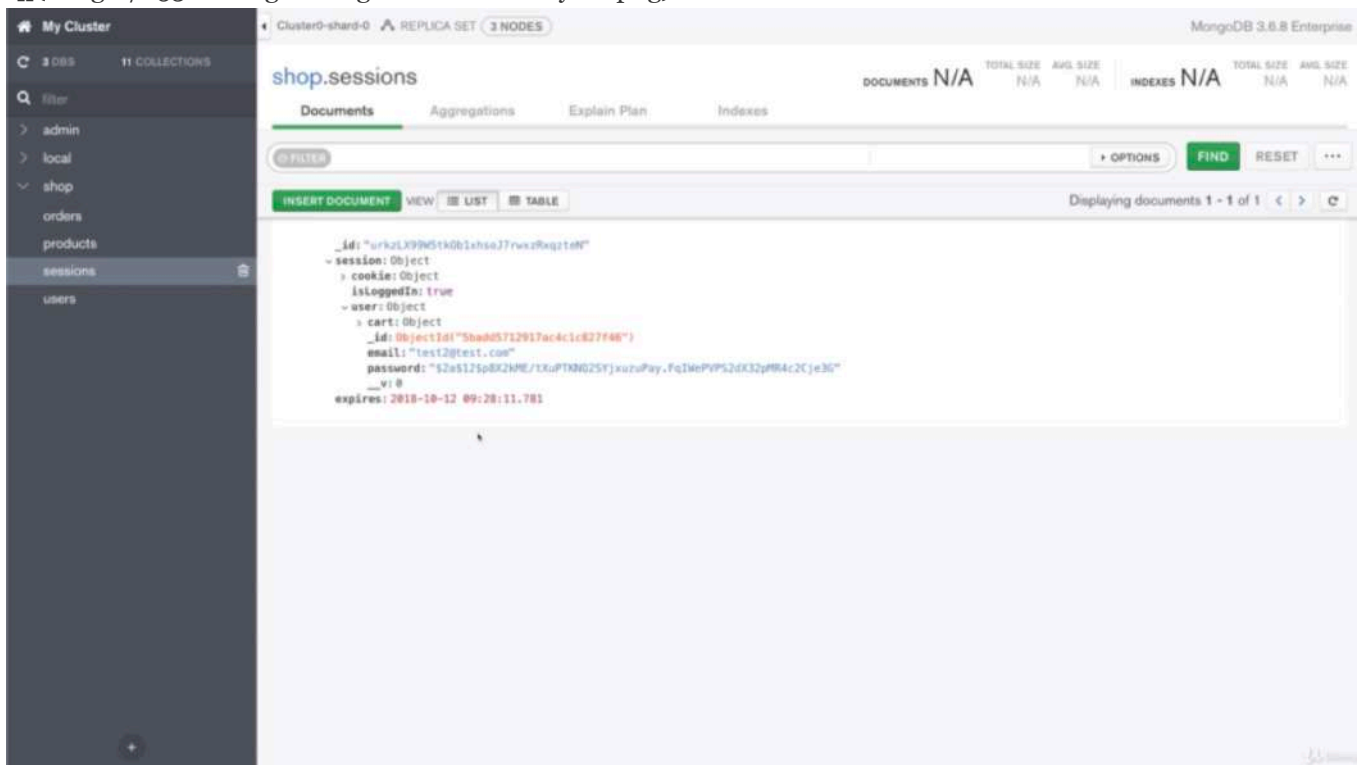
```

1: node

```
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:85265) 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.
undefined

```

- if we enter the valid email and valid password, now i'm on the starting page, i see all options. and we got no error here.



- and if i have a look at my sessions collection, we have got a session because i'm logged in. and this is the user object with which i'm logged in.

```
1 //./controllers/auth.js
2
3 const bcrypt = require('bcryptjs');
4
5 const User = require('../models/user');
6
7 exports.getLogin = (req, res, next) => {
8   res.render('auth/login', {
```

```

9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.getSignup = (req, res, next) => {
16   res.render('auth/signup', {
17     path: '/signup',
18     pageTitle: 'Signup',
19     isAuthenticated: false
20   });
21 };
22
23 exports.postLogin = (req, res, next) => {
24   const email = req.body.email
25   const password = req.body.password
26   User.findOne({ email: email })
27     .then(user => {
28       if(!user){
29         return res.redirect('/login')
30       }
31       /**we will do this with 'bcrypt'
32        * because the password is stored in hashed form
33        * and we can't reverse this.
34        * so how we can compare the password?
35        *
36        * 'bcrypt' was responsible for creating the hash
37        * bcrypt uses a certain algorithm
38        * and we can essentially pass the password the user entered into bcrypt
39        * and let bcrypt compare it to the hashed value
40        * and bcrypt can then find out if the hashed value does make sense,
41        * taking into account the algorithm that is used
42        * for creating that for the password you entered as plain text
43        * if you would hash it could result in the hash password
44        * and if the answer is true, then the user entered a valid password
45        *
46        * and for this, bcrypt has the 'compare' function
47        * and so the password we extract from the incoming request
48        * and then the hashed value against we wanna compare it.
49        * that is found in our user document which we get from the database
50        * and there in the password field.
51        */
52       bcrypt.compare(password, user.password)
53       /**with 'compare', we will only face an error
54        * if something goes wrong, not if the password don't match
55        * in both a matching and a non-matching case,
56        * we make it into 'then' block
57        * and result will be a boolean that is true if the passwords are equal
58        * and it will be false if they are not equal
59        *
60        */
61       .then(doMatch => {
62         if(doMatch){
63           req.session.isLoggedIn = true;
64           req.session.user = user;

```

```

65     /**return this
66     * therefore, we should also return this
67     * to avoid code execution of this line "res.redirect('/login')"

```

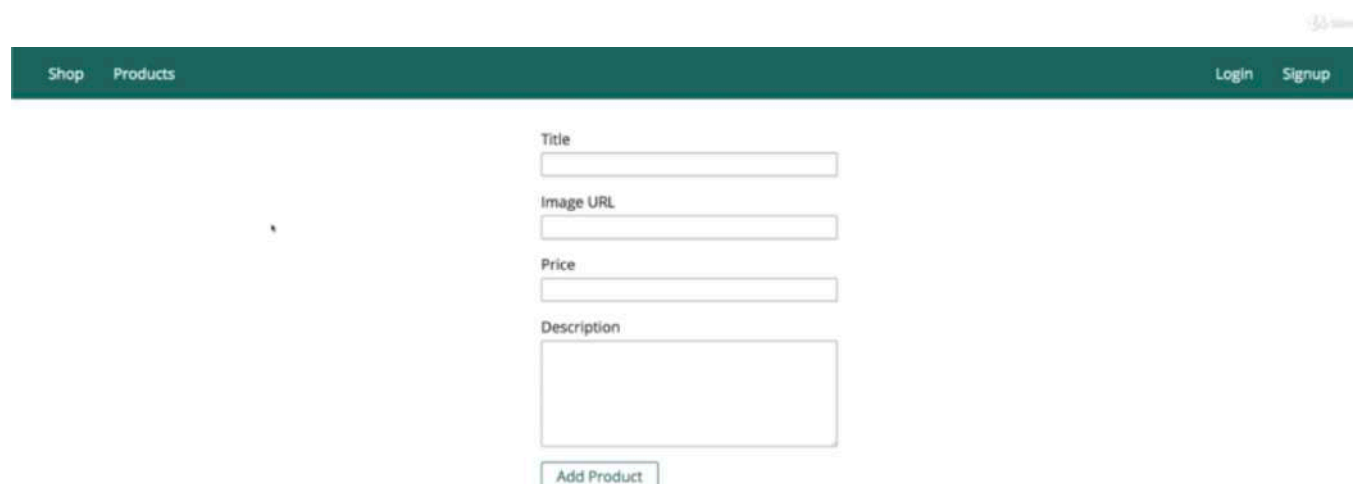
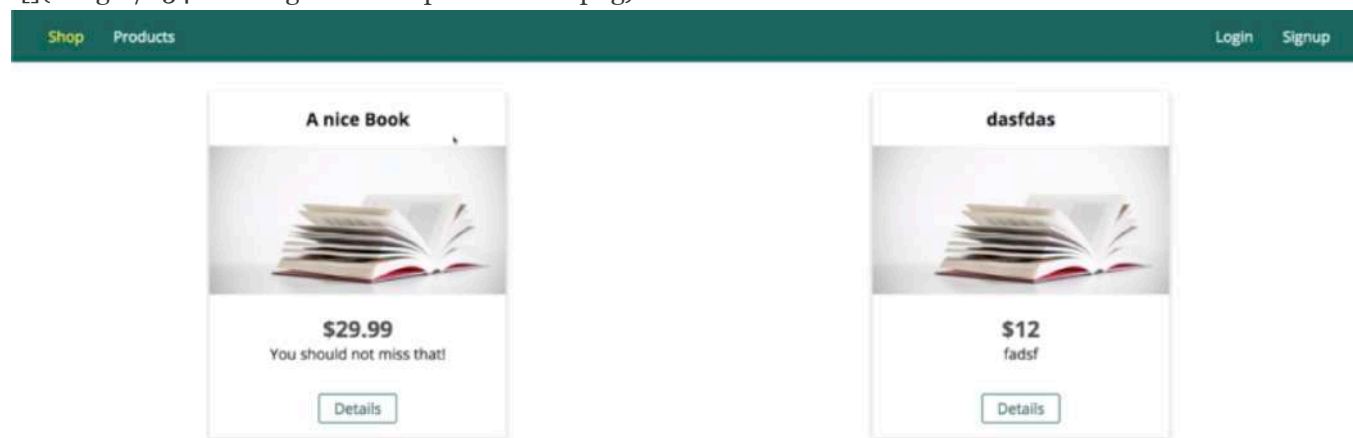
```

120     console.log(err);
121     res.redirect('/');
122   });
123 };
124

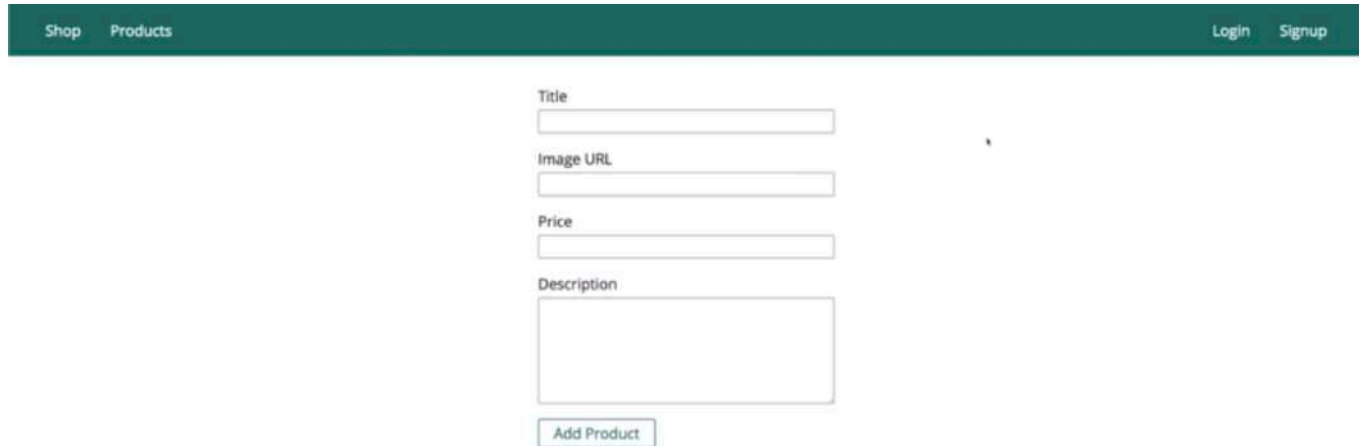
```

* Chapter 254: Working On Route Protection

1. update
- ./controllers/admin.js



- even when i logout and therefore my session is gone, even when i do that, i can still manually enter admin '/add-product' and reach that page and i could even try to creat a product here.
- but i would fail then because there would be no user object on the backend where i tried to get the logged in user. this is not the user experience you wanna offer.
- so we need to protect our route.



This screenshot shows the 'Add Product' form in a web application. The form is centered on a light gray background. At the top, there is a dark green navigation bar with 'Shop' and 'Products' on the left, and 'Login' and 'Signup' on the right. The form itself has four input fields: 'Title', 'Image URL', 'Price', and 'Description'. The 'Description' field is a larger text area. Below the input fields is a button labeled 'Add Product'.



This screenshot shows the 'Login' form in a web application. The form is centered on a light gray background. At the top, there is a dark green navigation bar with 'Shop' and 'Products' on the left, and 'Login' and 'Signup' on the right. The form has two input fields: 'E-Mail' and 'Password'. Below the input fields is a button labeled 'Login'.

- i'm not logged in. and i'm on the login page. so this works, i'm redirected here because session logged in is not set.

E-Mail

Password

[Login](#)

Title *

Image URL

Price

Description

[Add Product](#)

- on the other hand if i login now, this does work because i'm logged in now.

```
1 // ./controllers/admin.js
2
3 const Product = require('../models/product');
4
5 exports.getAddProduct = (req, res, next) => {
6   if(!req.session.isLoggedIn){
7     return res.redirect('/login')
8   }
9   res.render('admin/edit-product', {
10     pageTitle: 'Add Product',
11     path: '/admin/add-product',
12     editing: false,
```

```
13     isAuthenticated: req.isLoggedIn
14   });
15 };
16
17 exports.postAddProduct = (req, res, next) => {
18   const title = req.body.title;
19   const imageUrl = req.body.imageUrl;
20   const price = req.body.price;
21   const description = req.body.description;
22   const product = new Product({
23     title: title,
24     price: price,
25     description: description,
26     imageUrl: imageUrl,
27     userId: req.user
28   })
29   product
30     .save()
31     .then(result => {
32       // console.log(result);
33       console.log('Created Product');
34       res.redirect('/admin/products');
35     })
36     .catch(err => {
37       console.log(err);
38     });
39 };
40
41 exports.getEditProduct = (req, res, next) => {
42   const editMode = req.query.edit;
43   if (!editMode) {
44     return res.redirect('/');
45   }
46   const prodId = req.params.productId;
47   Product.findById(prodId)
48     .then(product => {
49       if (!product) {
50         return res.redirect('/');
51       }
52       res.render('admin/edit-product', {
53         pageTitle: 'Edit Product',
54         path: '/admin/edit-product',
55         editing: editMode,
56         product: product,
57         isAuthenticated: req.isLoggedIn
58       });
59     })
60     .catch(err => console.log(err));
61 };
62
63 exports.postEditProduct = (req, res, next) => {
64   const prodId = req.body.productId;
65   const updatedTitle = req.body.title;
66   const updatedPrice = req.body.price;
67   const updatedImageUrl = req.body.imageUrl;
68   const updatedDesc = req.body.description;
```

```

69
70 Product
71   .findById(prodId)
72   .then(product => {
73     product.title = updatedTitle
74     product.price = updatedPrice
75     product.description = updatedDesc
76     product.imageUrl = updatedImageUrl
77     return product
78   })
79   .save()
80   .then(result => {
81     console.log('UPDATED PRODUCT!');
82     res.redirect('/admin/products');
83   })
84   .catch(err => console.log(err));
85 };
86
87 exports.getProducts = (req, res, next) => {
88   Product.find()
89   //   .select('title price -_id')
90   //   .populate('userId', 'name')
91   .then(products => {
92     console.log(products)
93     res.render('admin/products', {
94       prods: products,
95       pageTitle: 'Admin Products',
96       path: '/admin/products',
97       isAuthenticated: req.isLoggedIn
98     });
99   })
100   .catch(err => console.log(err));
101 };
102
103 exports.postDeleteProduct = (req, res, next) => {
104   const prodId = req.body.productId;
105   Product.findByIdAndRemove(prodId)
106   .then(() => {
107     console.log('DESTROYED PRODUCT');
108     res.redirect('/admin/products');
109   })
110   .catch(err => console.log(err));
111 };

```

* Chapter 255: Using Middleware To Protect Routes

1. update
 - ./middleware/is-auth.js
 - ./routes/admin.js
 - ./routes/shop.js

[Shop](#) [Products](#) [Login](#) [Signup](#)

E-Mail

Password*

Login

- i will try to access admin/add-product in being logged in. and you see i end up on the login page
 - and if i try to access cart, i end up on the login page.
-
-
-

[Shop](#) [Products](#) [Login](#) [Signup](#)

E-Mail

test2@test.com

Password

Login

Showing 10 documents

Title
New Book

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

Price
29.99

Description
But it's still new!

Add Product



- if i login, i can add a new book with that same image. let's add this and this works

A promotional card for a book. At the top, it says "A nice Book" in bold black text. Below this is a photograph of an open book with white pages and a red cover, lying flat. Under the photo, the price "\$ 29.99" is displayed in a large, bold, black font. Below the price, the text "You should not miss that!" is written in a smaller black font. At the bottom of the card are two rounded rectangular buttons: "Edit" on the left and "Delete" on the right, both in black text.

Title	<input type="text" value="New Book"/>
Image URL	<input type="text" value="http://ichef.bbci.co.uk/ww/features/wm/live"/>
Price	<input type="text" value="39,99"/>
Description	<input type="text" value="But it's still new!"/>
<input type="button" value="Update Product"/>	



```

1 //./middleware/is-auth.js
2
3 module.exports = (req, res, next) => {
4   if(!req.session.isLoggedIn){
5     return res.redirect('/login')
6   }
7   /**i will call next
8    * because otherwise, i wanna allow the request to continue
9    * to whichever route the request wanted to go to
10  */
11   next()
12 }

```

```

1 // ./routes/admin.js

```



```

2
3 const path = require('path');
4
5 const express = require('express');
6
7 const adminController = require('../controllers/admin');
8 const isAuthenticated = require('../middleware/is-auth')
9
10 const router = express.Router();
11
12 // /admin/add-product => GET
13 /**'isAuth' can now simply be added as an argument to 'get'
14 * and you can add as many arguments as you want, as many handlers as you want
15 * they will be parsed from left to right.
16 * the request will travel through them from left to right
17 * so the request which reaches getProduct goes in to that 'isAuth' middleware first
18 * and in the 'isAuth' middleware, we might be redirecting
19 * and we don't call 'next()'
20 * hence the request would never continue to that controller action
21 *
22 * but if we make it past the if check in ./middleware/isAuth.js file,
23 * we do call 'next()',
24 * so that next middleware in line will be called
25 * and the next middleware in line would be our 'getAddProduct' controller action here.
26 *
27 * and 'isAuth' middleware to all the routes
28 * because these routes all require authentication
29 *
30 */
31 router.get('/add-product', isAuthenticated, adminController.getAddProduct);
32
33 // /admin/products => GET
34 router.get('/products', isAuthenticated, adminController.getProducts);
35
36 // /admin/add-product => POST
37 router.post('/add-product', isAuthenticated, adminController.postAddProduct);
38
39 router.get('/edit-product/:productId', isAuthenticated, adminController.getEditProduct);
40
41 router.post('/edit-product', isAuthenticated, adminController.postEditProduct);
42
43 router.post('/delete-product', isAuthenticated, adminController.postDeleteProduct);
44
45 module.exports = router;

```



```

1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8 const isAuthenticated = require('../middleware/is-auth')
9
10 const router = express.Router();
11
12 router.get('/', shopController.getIndex);

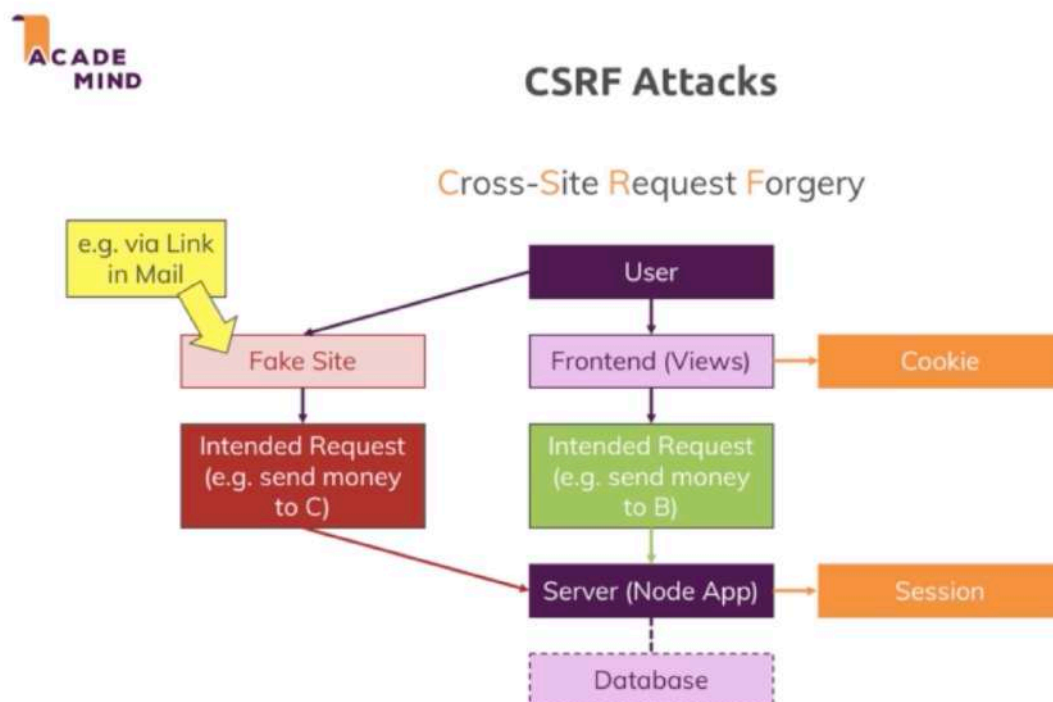
```

```

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

```

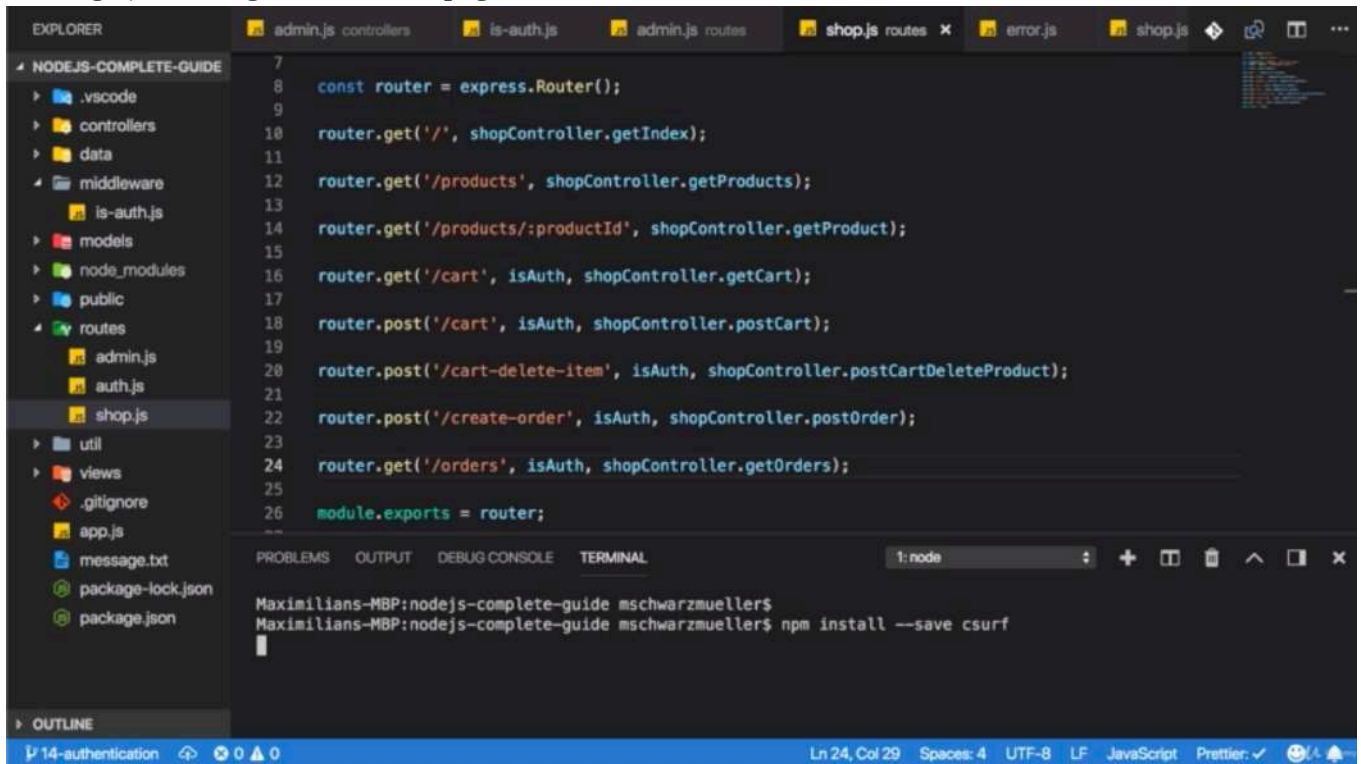
* Chapter 256: Understanding CSRF Attacks



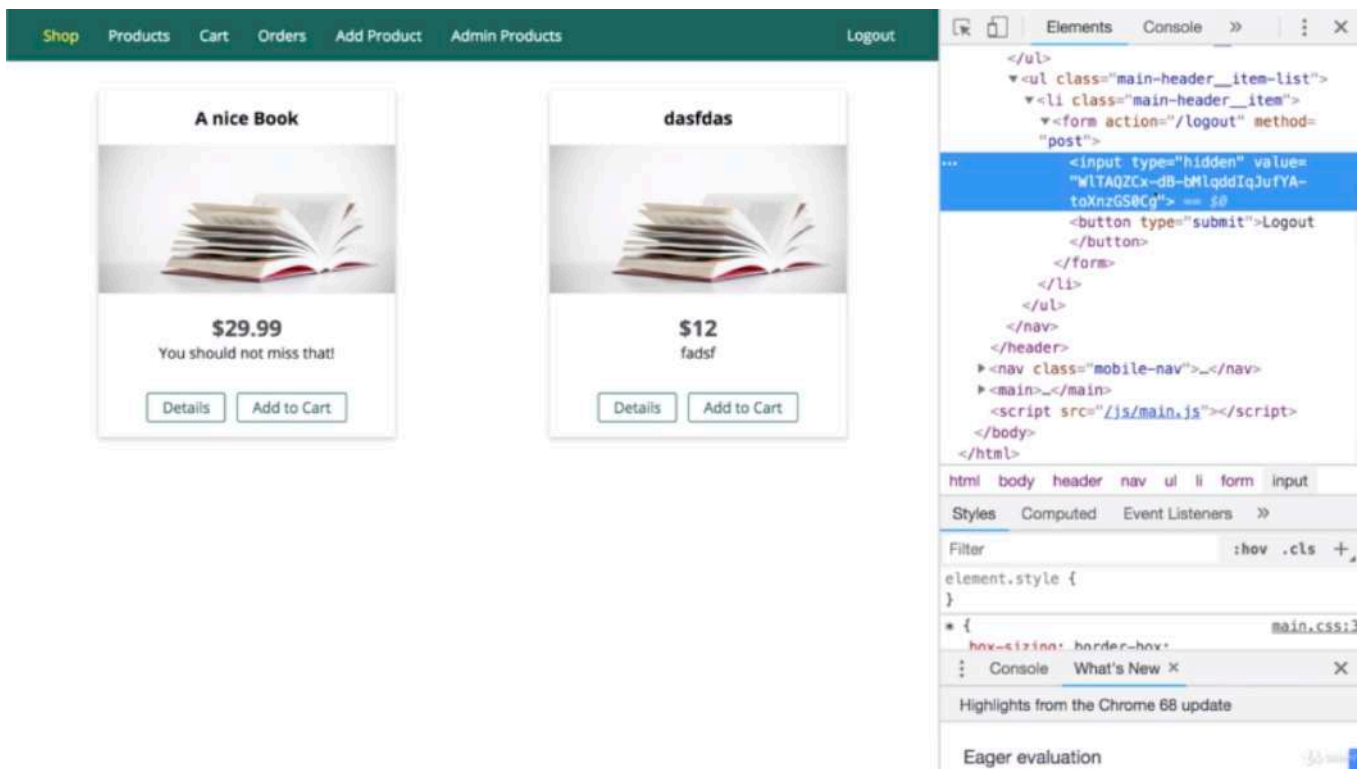
- since you got valid session for that user if you send something to your site, to your server, your sessions is used for that user and therefore that behind the scenes data that the user never sees that configures the money transferal or the order in a certain way that is not OK to the user.
- this part is invisible to the user but the valid session gets used for it because your server is used and therefore this is accepted. this is attack pattern where the session can be stolen where you can abuse the fact the users are logged in and you can simply trick them into making requests which they might not even notice and obviously we wanna protect against this attack pattern
- how we can protect from this is that we wanna ensure that people can only use your session if they are working with your views. so with the views rendered by your application, so that the session is not available on any fake page that might look like your page. but that aren't your page. and to ensure this, to add this feature, we will use a so-called CSRF token.

*Chapter 257: Using A CSRF Token

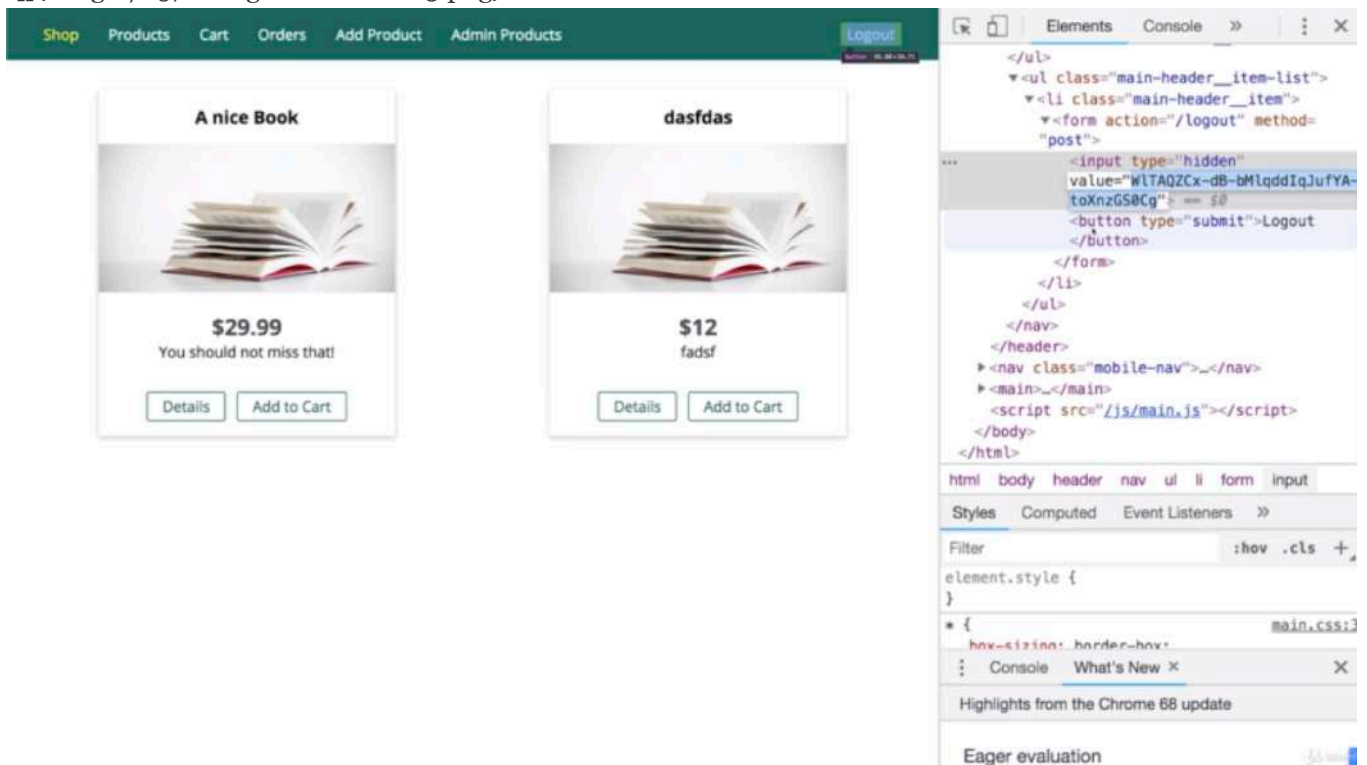
- 1. update
 - app.js
 - ./controllers/auth.js
 - ./controllers/shop.js
 - ./views/includes/navigation.ejs



- 'npm install --save csrf' is a package for node-express which allows us to generate a so-called CSRF token. basically a token, a string value, we can embed into our forms, so into our pages for every request that does something on the backend that changes the users state. so anything like ordering something, anything that does something sensitive which we wanna protect against.
- we can include such a token in our views and on the server, this package will check if the incoming request does have that valid token.
- how does this protect us? the fake sites might send a request to your backend and they could theoretically use your session but the requests will be missing the token and they can't guess the token because it's random hashed value and only one value is valid and the package which runs on the server determines whether a token is valid, so they can't guess it and they also can steal it because a new token is generated for every page you render.




- let me reload and when inspecting the logout button, you will see that there we also have that input with the value of this token and



- this here is the CSRF token which was generated by the package

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout


A nice Book



\$29.99
You should not miss that!

Details
Add to Cart

dasfdas



\$12
fadsf

Details
Add to Cart

Elements
Console

```

<!doctype html>
<html lang="en">
  <head>_</head>
  <body cz-shortcut-listen="true">
    <div class="backdrop"></div>
    <header class="main-header">
      <button id="side-menu-toggle">Menu
    </button>
    <nav class="main-header__nav">
      <ul class="main-header__item-list">_
    </ul>
    <ul class="main-header__item-list">
      <li class="main-header__item">
        <form action="/logout" method=
        "post">
          <input type="hidden" name="_csrf"
            value="YKHgfCnW-
            kRxfsNp8wUfxtpuq0jRrc0bB_jg"> ==
          <button type="submit">Logout
        </button>
      </li>
    </ul>
  </body>
</html>

```

html
body
header
nav
ul
li
form
input

Styles
Computed
Event Listeners

Filter
:hov .cls +

element.style {
}

* {
 hnx-size: 100%; border: 1px solid #ccc;
}


Console
What's New

Highlights from the Chrome 68 update

Eager evaluation

Shop
Products
Login
Signup


A nice Book



\$29.99
You should not miss that!

Details

dasfdas



\$12
fadsf

Details

Elements
Console

```

<!doctype html>
<html lang="en">
  <head>_</head>
  <body cz-shortcut-listen="true"> == $0
    <div class="backdrop"></div>
    <header class="main-header"></header>
    <nav class="mobile-nav"></nav>
    <main>_</main>
    <script src="/js/main.js"></script>
  </body>
</html>

```

html
body

Styles
Computed
Event Listeners

Filter
:hov .cls +

element.style {
}

body {
 padding: 10px;
}

Console
What's New

Highlights from the Chrome 68 update

Eager evaluation

- if i click logout, now it works because the package is able to extract that CSRF Token. now it's not just a session that matters but also the existence of this token.

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const session = require('express-session');
9 const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf')
11

```

```

12 const errorController = require('./controllers/error');
13 const User = require('./models/user');
14
15 const MONGODB_URI =
16   'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop?retryWrites=true';
17
18 const app = express();
19 const store = new MongoDBStore({
20   uri: MONGODB_URI,
21   collection: 'sessions'
22 });
23 /**you can pass an object where you can configure somethings,
24  * for example that you wanna store the secret that is used for assigning your token,
25  * so for hashing them, that you wanna store them in cookie
26  * instead of the session which is the default
27  *
28  * but i wanna use the session, the default.
29  * we don't need to set any of the other values.
30  */
31 const csrfProtection = csrf()
32
33 app.set('view engine', 'ejs');
34 app.set('views', 'views');
35
36 const adminRoutes = require('./routes/admin');
37 const shopRoutes = require('./routes/shop');
38 const authRoutes = require('./routes/auth');
39
40 app.use(bodyParser.urlencoded({ extended: false }));
41 app.use(express.static(path.join(__dirname, 'public')));
42 app.use(
43   session({
44     secret: 'my secret',
45     resave: false,
46     saveUninitialized: false,
47     store: store
48   })
49 );
50 /**initialize the session.
51  * because the package will use that session now.
52  * after initializing the session,
53  * i will add 'csrfProtection'
54  *
55  * csrfProtection is enabled
56  * but we still need to add something to our views to use it.
57  */
58 app.use(csrfProtection)
59
60 app.use((req, res, next) => {
61   if (!req.session.user) {
62     return next();
63   }
64   User.findById(req.session.user._id)
65     .then(user => {
66       req.user = user;
67       next();

```

```

68     })
69     .catch(err => console.log(err));
70 });
71
72 app.use('/admin', adminRoutes);
73 app.use(shopRoutes);
74 app.use(authRoutes);
75
76 app.use(errorController.get404);
77
78 mongoose
79   .connect(MONGODB_URI)
80   .then(result => {
81     app.listen(3000);
82   })
83   .catch(err => {
84     console.log(err);
85   });
86

```

```

1  //./controllers/auth.js
2
3  const bcrypt = require('bcryptjs');
4
5  const User = require('../models/user');
6
7  exports.getLogin = (req, res, next) => {
8    res.render('auth/login', {
9      path: '/login',
10     pageTitle: 'Login',
11     isAuthenticated: false
12   });
13 };
14
15 exports.getSignup = (req, res, next) => {
16   res.render('auth/signup', {
17     path: '/signup',
18     pageTitle: 'Signup',
19     isAuthenticated: false
20   });
21 };
22
23 exports.postLogin = (req, res, next) => {
24   const email = req.body.email;
25   const password = req.body.password;
26   User.findOne({ email: email })
27     .then(user => {
28       if (!user) {
29         return res.redirect('/login');
30       }
31       bcrypt
32         .compare(password, user.password)
33         .then(doMatch => {
34           if (doMatch) {
35             req.session.isLoggedIn = true;
36             req.session.user = user;
37             return req.session.save(err => {

```



```

38     console.log(err);
39     res.redirect('/');
40   });
41   }
42   res.redirect('/login');
43 })
44 .catch(err => {
45   console.log(err);
46   res.redirect('/login');
47   });
48 })
49 .catch(err => console.log(err));
50 };
51
52 exports.postSignup = (req, res, next) => {
53   const email = req.body.email;
54   const password = req.body.password;
55   const confirmPassword = req.body.confirmPassword;
56   User.findOne({ email: email })
57     .then(userDoc => {
58       if (userDoc) {
59         return res.redirect('/signup');
60       }
61       return bcrypt
62         .hash(password, 12)
63         .then(hashPassword => {
64           const user = new User({
65             email: email,
66             password: hashPassword,
67             cart: { items: [] }
68           });
69           return user.save();
70         })
71         .then(result => {
72           res.redirect('/login');
73         });
74     })
75     .catch(err => {
76       console.log(err);
77     });
78 };
79
80 /**'Logout' failed because i get an invalid token
81  * because logout action was a POST request
82  * and for any non-GET request
83  * because you change data via POST requests typically
84  * for any such requests,
85  * this package will look for the existence of a CSRF token in your views in the request
86  body
87  *
88  * to make sure that such a token is there,
89  * we first of all need to ensure we have it available in our views
90  * to do that, we have to pass data into our view
91  */
92 exports.postLogout = (req, res, next) => {
93   req.session.destroy(err => {

```



```
93     console.log(err);
94     res.redirect('/');
95   });
96 };
```

```
1  //./controllers/shop.js
2
3  const Product = require('../models/product');
4  const Order = require('../models/order');
5
6  exports.getProducts = (req, res, next) => {
7    Product.find()
8      .then(products => {
9        console.log(products);
10       res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products',
14         isAuthenticated: req.session.isLoggedIn
15       });
16     })
17     .catch(err => {
18       console.log(err);
19     });
20 };
21
22 exports.getProduct = (req, res, next) => {
23   const prodId = req.params.productId;
24   Product.findById(prodId)
25     .then(product => {
26       res.render('shop/product-detail', {
27         product: product,
28         pageTitle: product.title,
29         path: '/products',
30         isAuthenticated: req.session.isLoggedIn
31       });
32     })
33     .catch(err => console.log(err));
34 };
35
36 exports.getIndex = (req, res, next) => {
37   Product.find()
38     .then(products => {
39       res.render('shop/index', {
40         prods: products,
41         pageTitle: 'Shop',
42         path: '/',
43         isAuthenticated: req.session.isLoggedIn,
44         /**'csrfToken()' is provided by the CSRF middleware which we added by this package.
45          * this will generate such a token
46          * and we will store it in CSRF token which we then can use in our view
47          */
48         csrfToken: req.csrfToken()
49       });
50     })
51     .catch(err => {
52       console.log(err);
```

```

53     });
54 };
55
56 exports.getCart = (req, res, next) => {
57   req.user
58     .populate('cart.items.productId')
59     .execPopulate()
60     .then(user => {
61       const products = user.cart.items;
62       res.render('shop/cart', {
63         path: '/cart',
64         pageTitle: 'Your Cart',
65         products: products,
66         isAuthenticated: req.session.isLoggedIn
67       });
68     })
69     .catch(err => console.log(err));
70 };
71
72 exports.postCart = (req, res, next) => {
73   const prodId = req.body.productId;
74   Product.findById(prodId)
75     .then(product => {
76       return req.user.addToCart(product);
77     })
78     .then(result => {
79       console.log(result);
80       res.redirect('/cart');
81     });
82 };
83
84 exports.postCartDeleteProduct = (req, res, next) => {
85   const prodId = req.body.productId;
86   req.user
87     .removeFromCart(prodId)
88     .then(result => {
89       res.redirect('/cart');
90     })
91     .catch(err => console.log(err));
92 };
93
94 exports.postOrder = (req, res, next) => {
95   req.user
96     .populate('cart.items.productId')
97     .execPopulate()
98     .then(user => {
99       const products = user.cart.items.map(i => {
100         return { quantity: i.quantity, product: { ...i.productId._doc } };
101       });
102       const order = new Order({
103         user: {
104           name: req.user.name,
105           userId: req.user
106         },
107         products: products
108       });

```

```

109     return order.save();
110   })
111   .then(result => {
112     return req.user.clearCart();
113   })
114   .then(() => {
115     res.redirect('/orders');
116   })
117   .catch(err => console.log(err));
118 };
119
120 exports.getOrders = (req, res, next) => {
121   Order.find({ 'user.userId': req.user._id })
122     .then(orders => {
123       res.render('shop/orders', {
124         path: '/orders',
125         pageTitle: 'Your Orders',
126         orders: orders,
127         isAuthenticated: req.session.isLoggedIn
128       });
129     })
130     .catch(err => console.log(err));
131 };
132
133

```

```

1 <!--./views/includes/navigation.ejs-->
2
3 <div class="backdrop"></div>
4 <header class="main-header">
5   <button id="side-menu-toggle">Menu</button>
6   <nav class="main-header__nav">
7     <ul class="main-header__item-list">
8       <li class="main-header__item">
9         <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10      </li>
11      <li class="main-header__item">
12        <a class="<%= path === '/products' ? 'active' : '' %>"
13        href="/products">Products</a>
14      </li>
15      <% if (isAuthenticated) { %>
16        <li class="main-header__item">
17          <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
18        </li>
19        <li class="main-header__item">
20          <a class="<%= path === '/orders' ? 'active' : '' %>"
21          href="/orders">Orders</a>
22        </li>
23        <li class="main-header__item">
24          <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
25          href="/admin/add-product">Add Product
26        </a>
27        </li>
28        <li class="main-header__item">
29          <a class="<%= path === '/admin/products' ? 'active' : '' %>"
30          href="/admin/products">Admin Products
31        </a>
32      </li>
33    </ul>
34  </nav>
35 </header>

```

```

28         </li>
29     <% } %>
30 </ul>
31 <ul class="main-header__item-list">
32     <% if (!isAuthenticated) { %>
33         <li class="main-header__item">
34             <a class="<%= path === '/login' ? 'active' : '' %>"
href="/login">Login</a>
35         </li>
36         <li class="main-header__item">
37             <a class="<%= path === '/signup' ? 'active' : '' %>"
href="/signup">Signup</a>
38         </li>
39     <% } else { %>
40         <li class="main-header__item">
41             <form action="/logout" method="post">
42                 <!--
43                 'csrfToken' in here is in 'getIndex()' in ./controllers/shop.js
44                 we have to give this a name and the name has to be '_csrf'
45                 because the package which we added will look for this name
46
47                 the same should be done in the mobile logout button
48                 -->
49                 <input type="hidden" name="_csrf" value="<%= csrfToken %>">
50                 <button type="submit">Logout</button>
51             </form>
52         </li>
53     <% } %>
54 </ul>
55 </nav>
56 </header>
57
58 <nav class="mobile-nav">
59     <ul class="mobile-nav__item-list">
60         <li class="mobile-nav__item">
61             <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
62         </li>
63         <li class="mobile-nav__item">
64             <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
65         </li>
66         <% if (isAuthenticated) { %>
67             <li class="mobile-nav__item">
68                 <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
69             </li>
70             <li class="mobile-nav__item">
71                 <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
72             </li>
73             <li class="mobile-nav__item">
74                 <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
75             </li>
76             <li class="mobile-nav__item">
77                 <a class="<%= path === '/admin/products' ? 'active' : '' %>"

```

```

79 href="/admin/products">Admin Products
80     </a>
81 </li>
82 <% } %>
83 <% if (!isAuthenticated) { %>
84     <li class="mobile-nav__item">
85         <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
86     </li>
87     <li class="mobile-nav__item">
88         <a class="<%= path === '/signup' ? 'active' : '' %>"
89 href="/signup">Signup</a>
90     </li>
91 <% } else { %>
92     <li class="mobile-nav__item">
93         <form action="/logout" method="post">
94             <input type="hidden" name="_csrf" value="<%= csrfToken %>">
95             <button type="submit">Logout</button>
96         </form>
97     </li>
98 <% } %>
99 </ul>
100 </nav>

```

* Chapter 258: Adding CSRF Protection

1. update
 - ./controllers/shop.js
 - app.js
 - ./views/includes/navigation.ejs
 - ./views/includes/add-to-cart.ejs
 - ./views/admin/edit-product.ejs
 - ./views/auth/login.ejs
 - ./views/auth/signup.ejs
 - ./views/admin/product.ejs
 - ./views/shop/cart.ejs


E-Mail

test2@test.com

Password

Logh

A nice Book




\$29.99

You should not miss that!

DetailsAdd to Cart

dasfdas




\$12

fadsf

DetailsAdd to Cart

A nice Book



\$29.99

You should not miss that!

Details

Add to Cart

dasfdas



\$12

fadsf

Details

Add to Cart

A nice Book	Quantity: 1	<div>Delete</div>
-------------	-------------	-------------------

Order Now!

No Products in Cart!

Nothing there!

- this CSRF Protection is the crucial thing which you have to add to any production ready application, it's not optional. otherwise you will have a huge security issue on your page. you need to add this to ensure that your sessions don't get stolen.

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const session = require('express-session');
9 const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf')
```



```

11
12 const errorController = require('./controllers/error');
13 const User = require('./models/user');
14
15 const MONGODB_URI =
16   'mongodb+srv://maximilian:rldnjs12@cluster0-z3vkl.mongodb.net/shop?retryWrites=true';
17
18 const app = express();
19 const store = new MongoDBStore({
20   uri: MONGODB_URI,
21   collection: 'sessions'
22 });
23 /**you can pass an object where you can configure somethings,
24  * for example that you wanna store the secret that is used for assigning your token,
25  * so for hashing them, that you wanna store them in cookie
26  * instead of the session which is the default
27  *
28  * but i wanna use the session, the default.
29  * we don't need to set any of the other values.
30  */
31 const csrfProtection = csrf()
32
33 app.set('view engine', 'ejs');
34 app.set('views', 'views');
35
36 const adminRoutes = require('./routes/admin');
37 const shopRoutes = require('./routes/shop');
38 const authRoutes = require('./routes/auth');
39
40 app.use(bodyParser.urlencoded({ extended: false }));
41 app.use(express.static(path.join(__dirname, 'public')));
42 app.use(
43   session({
44     secret: 'my secret',
45     resave: false,
46     saveUninitialized: false,
47     store: store
48   })
49 );
50 /**initialize the session.
51  * because the package will use that session now.
52  * after initializing the session,
53  * i will add 'csrfProtection'
54  *
55  * csrfProtection is enabled
56  * but we still need to add something to our views to use it.
57  */
58 app.use(csrfProtection)
59
60 app.use((req, res, next) => {
61   if (!req.session.user) {
62     return next();
63   }
64   User.findById(req.session.user._id)
65     .then(user => {
66       req.user = user;

```

```

67     next();
68   })
69   .catch(err => console.log(err));
70 });
71
72 app.use((req, res, next) => {
73   /**we can use a special feature provided by Express.js
74    * we can access a special field on the response, the 'locals' field
75    * this allows us to set local variables that are passed into the views,
76    * because they will only exist in the views which are rendered.
77    *
78    * so now for every request that is executed,
79    * these 2 fields will be set for the views that are rendered
80    *
81    * then we have to call 'next()'
82    * so that we are able to continue
83    */
84   res.locals.isAuthenticated = req.session.isLoggedIn
85   res.locals.csrfToken = req.csrfToken()
86   next()
87 })
88
89 app.use('/admin', adminRoutes);
90 app.use(shopRoutes);
91 app.use(authRoutes);
92
93 app.use(errorController.get404);
94
95 mongoose
96   .connect(MONGODB_URI)
97   .then(result => {
98     app.listen(3000);
99   })
100  .catch(err => {
101    console.log(err);
102  });
103

```

```

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

```

```

20 };
21
22 exports.getProduct = (req, res, next) => {
23   const prodId = req.params.productId;
24   Product.findById(prodId)
25     .then(product => {
26       res.render('shop/product-detail', {
27         product: product,
28         pageTitle: product.title,
29         path: '/products',
30         isAuthenticated: req.session.isLoggedIn
31       });
32     })
33     .catch(err => console.log(err));
34 };
35
36 exports.getIndex = (req, res, next) => {
37   Product.find()
38     .then(products => {
39       res.render('shop/index', {
40         prods: products,
41         pageTitle: 'Shop',
42         path: '/'
43         /**tell express.js that this is now totally unrelated from CSRF thing,
44         * tell express.js that we have some data that should be included in every rendered
45         view
46         * we will do this in app.js
47         */
48       });
49     })
50     .catch(err => {
51       console.log(err);
52     });
53 };
54
55 exports.getCart = (req, res, next) => {
56   req.user
57     .populate('cart.items.productId')
58     .execPopulate()
59     .then(user => {
60       const products = user.cart.items;
61       res.render('shop/cart', {
62         path: '/cart',
63         pageTitle: 'Your Cart',
64         products: products,
65         isAuthenticated: req.session.isLoggedIn
66       });
67     })
68     .catch(err => console.log(err));
69 };
70
71 exports.postCart = (req, res, next) => {
72   const prodId = req.body.productId;
73   Product.findById(prodId)
74     .then(product => {
75     return req.user.addToCart(product);

```

```

75     })
76     .then(result => {
77         console.log(result);
78         res.redirect('/cart');
79     });
80 };
81
82 exports.postCartDeleteProduct = (req, res, next) => {
83     const prodId = req.body.productId;
84     req.user
85         .removeFromCart(prodId)
86         .then(result => {
87             res.redirect('/cart');
88         })
89         .catch(err => console.log(err));
90 };
91
92 exports.postOrder = (req, res, next) => {
93     req.user
94         .populate('cart.items.productId')
95         .execPopulate()
96         .then(user => {
97             const products = user.cart.items.map(i => {
98                 return { quantity: i.quantity, product: { ...i.productId._doc } };
99             });
100             const order = new Order({
101                 user: {
102                     name: req.user.name,
103                     userId: req.user
104                 },
105                 products: products
106             });
107             return order.save();
108         })
109         .then(result => {
110             return req.user.clearCart();
111         })
112         .then(() => {
113             res.redirect('/orders');
114         })
115         .catch(err => console.log(err));
116 };
117
118 exports.getOrders = (req, res, next) => {
119     Order.find({ 'user.userId': req.user._id })
120         .then(orders => {
121             res.render('shop/orders', {
122                 path: '/orders',
123                 pageTitle: 'Your Orders',
124                 orders: orders,
125                 isAuthenticated: req.session.isLoggedIn
126             });
127         })
128         .catch(err => console.log(err));
129 };
130

```

```

1 <!--./views/includes/navigation.ejs-->
2
3 <div class="backdrop"></div>
4 <header class="main-header">
5   <button id="side-menu-toggle">Menu</button>
6   <nav class="main-header__nav">
7     <ul class="main-header__item-list">
8       <li class="main-header__item">
9         <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10      </li>
11      <li class="main-header__item">
12        <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
13      </li>
14      <% if (isAuthenticated) { %>
15        <li class="main-header__item">
16          <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
17        </li>
18        <li class="main-header__item">
19          <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
20        </li>
21        <li class="main-header__item">
22          <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
23          </a>
24        </li>
25        <li class="main-header__item">
26          <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
27          </a>
28        </li>
29      <% } %>
30    </ul>
31    <ul class="main-header__item-list">
32      <% if (!isAuthenticated) { %>
33        <li class="main-header__item">
34          <a class="<%= path === '/login' ? 'active' : '' %>"
href="/login">Login</a>
35        </li>
36        <li class="main-header__item">
37          <a class="<%= path === '/signup' ? 'active' : '' %>"
href="/signup">Signup</a>
38        </li>
39      <% } else { %>
40        <li class="main-header__item">
41          <form action="/logout" method="post">
42            <!--
43              'csrfToken' in here is in 'getIndex()' in ./controllers/shop.js
44              we have to give this a name and the name has to be '_csrf'
45              because the package which we added will look for this name
46
47              the same should be done in the mobile logout button
48            -->
49            <input type="hidden" name="_csrf" value="<%= csrfToken %>">

```

```

50         <button type="submit">Logout</button>
51     </form>
52 </li>
53 <% } %>
54 </ul>
55 </nav>
56 </header>
57
58 <nav class="mobile-nav">
59     <ul class="mobile-nav__item-list">
60         <li class="mobile-nav__item">
61             <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
62         </li>
63         <li class="mobile-nav__item">
64             <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
65         </li>
66         <% if (isAuthenticated) { %>
67             <li class="mobile-nav__item">
68                 <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
69             </li>
70             <li class="mobile-nav__item">
71                 <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
72             </li>
73             <li class="mobile-nav__item">
74                 <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
75                 </a>
76             </li>
77             <li class="mobile-nav__item">
78                 <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
79                 </a>
80             </li>
81         <% } %>
82         <% if (!isAuthenticated) { %>
83             <li class="mobile-nav__item">
84                 <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
85             </li>
86             <li class="mobile-nav__item">
87                 <a class="<%= path === '/signup' ? 'active' : '' %>"
href="/signup">Signup</a>
88             </li>
89         <% } else { %>
90             <li class="mobile-nav__item">
91                 <form action="/logout" method="post">
92                     <!--
93                     <input type="hidden" name="_csrf" value="<%= csrfToken %>">'
94                     this should be input to every views(ejs file) which have POST
request
95                     -->
96                     <input type="hidden" name="_csrf" value="<%= csrfToken %>">
97                     <button type="submit">Logout</button>
98                 </form>
99             </li>

```

```
100     <% } %>
101 </ul>
102 </nav>
```

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

```
1 <!--./views/admin/edit-product.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4     <link rel="stylesheet" href="/css/forms.css">
5     <link rel="stylesheet" href="/css/product.css">
6 </head>
7
8 <body>
9     <%- include('../includes/navigation.ejs') %>
10
11     <main>
12         <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else
13 { %>add-product<% } %>" method="POST">
14             <div class="form-control">
15                 <label for="title">Title</label>
16                 <input type="text" name="title" id="title" value="<% if (editing) { %><%=
17 product.title %><% } %>">
18             </div>
19             <div class="form-control">
20                 <label for="imageUrl">Image URL</label>
21                 <input type="text" name="imageUrl" id="imageUrl" value="<% if (editing) { %>
22 <%= product.imageUrl %><% } %>">
23             </div>
24             <div class="form-control">
25                 <label for="price">Price</label>
26                 <input type="number" name="price" id="price" step="0.01" value="<% if
27 (editing) { %><%= product.price %><% } %>">
28             </div>
29             <div class="form-control">
30                 <label for="description">Description</label>
31                 <textarea name="description" id="description" rows="5"><% if (editing) { %>
32 <%= product.description %><% } %></textarea>
33             </div>
34             <% if (editing) { %>
35                 <input type="hidden" value="<%= product._id %>" name="productId">
36                 <% } %>
37
38                 <input type="hidden" name="_csrf" value="<%= csrfToken %>">
39                 <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else {
40 %>Add Product<% } %></button>
41             </form>
42         </main>
43     <%- include('../includes/end.ejs') %>
```

```
1 <!--./views/auth/login.ejs-->
```

```

2
3 <%= include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9   <%= include('../includes/navigation.ejs') %>
10
11   <main>
12     <form class="login-form" action="/login" method="POST">
13       <div class="form-control">
14         <label for="email">E-Mail</label>
15         <input type="email" name="email" id="email">
16       </div>
17       <div class="form-control">
18         <label for="password">Password</label>
19         <input type="password" name="password" id="password">
20       </div>
21       <input type="hidden" name="_csrf" value="<%= csrfToken %>">
22       <button class="btn" type="submit">Login</button>
23     </form>
24   </main>
25 <%= include('../includes/end.ejs') %>

```

```

1 <!--./views/auth/signup.ejs-->
2
3 <%= include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9   <%= include('../includes/navigation.ejs') %>
10
11   <main>
12     <form class="login-form" action="/signup" method="POST">
13       <div class="form-control">
14         <label for="email">E-Mail</label>
15         <input type="email" name="email" id="email">
16       </div>
17       <div class="form-control">
18         <label for="password">Password</label>
19         <input type="password" name="password" id="password">
20       </div>
21       <div class="form-control">
22         <label for="confirmPassword">Confirm Password</label>
23         <input type="password" name="confirmPassword" id="confirmPassword">
24       </div>
25       <input type="hidden" name="_csrf" value="<%= csrfToken %>">
26       <button class="btn" type="submit">Signup</button>
27     </form>
28   </main>
29 <%= include('../includes/end.ejs') %>

```

```

1 <!--./views/admin/products.ejs-->
2

```



```

3 <%- include('../includes/head.ejs') %>
4 <link rel="stylesheet" href="/css/product.css">
5 </head>
6
7 <body>
8 <%- include('../includes/navigation.ejs') %>
9
10 <main>
11 <% if (prods.length > 0) { %>
12 <div class="grid">
13 <% for (let product of prods) { %>
14 <article class="card product-item">
15 <header class="card_header">
16 <h1 class="product_title">
17 <%= product.title %>
18 </h1>
19 </header>
20 <div class="card_image">
21 ">
22 </div>
23 <div class="card_content">
24 <h2 class="product_price">$
25 <%= product.price %>
26 </h2>
27 <p class="product_description">
28 <%= product.description %>
29 </p>
30 </div>
31 <div class="card_actions">
32 <a href="/admin/edit-product/<%= product._id %>?
edit=true" class="btn">Edit</a>
33 <form action="/admin/delete-product" method="POST">
34 <input type="hidden" value="<%= product._id %>"
name="productId">
35 <input type="hidden" name="_csrf" value="<%=
csrfToken %>">
36 <button class="btn" type="submit">Delete</button>
37 </form>
38
39 </div>
40 </article>
41 <% } %>
42 </div>
43 <% } else { %>
44 <h1>No Products Found!</h1>
45 <% } %>
46 </main>
47 <%- include('../includes/end.ejs') %>

```

```

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

```

```


8      <%- include('../includes/navigation.ejs') %>
9      <main>
10         <% if (products.length > 0) { %>
11             <ul class="cart__item-list">
12                 <% products.forEach(p => { %>
13                     <li class="cart__item">
14                         <h1><%= p.productId.title %></h1>
15                         <h2>Quantity: <%= p.quantity %></h2>
16                         <form action="/cart-delete-item" method="POST">
17                             <input type="hidden" value="<%= p.productId._id %>"
name="productId">
18                             <input type="hidden" name="_csrf" value="<%= csrfToken %>">
19                             <button class="btn danger" type="submit">Delete</button>
20                         </form>
21                     </li>
22                 <% }) %>
23             </ul>
24             <hr>
25             <div class="centered">
26                 <form action="/create-order" method="POST">
27                     <input type="hidden" name="_csrf" value="<%= csrfToken %>">
28                     <button type="submit" class="btn">Order Now!</button>
29                 </form>
30             </div>
31
32         <% } else { %>
33             <h1>No Products in Cart!</h1>
34         <% } %>
35     </main>
36     <%- include('../includes/end.ejs') %>

```

* Chapter 259: Fixing The Order Button

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

A nice Book




\$ 29.99

You should not miss that!

Details

Add to Cart

dasfdas



\$ 12

fadsf

Details

Add to Cart


A nice Book

Quantity: 1

Delete

Order Now!

A nice Book



\$ 29.99

You should not miss that!

Details

Add to Cart

dasfdas



\$ 12

fadsf

Details

Add to Cart

A nice Book

Quantity: 2

Delete

Order Now!

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

Order Now!

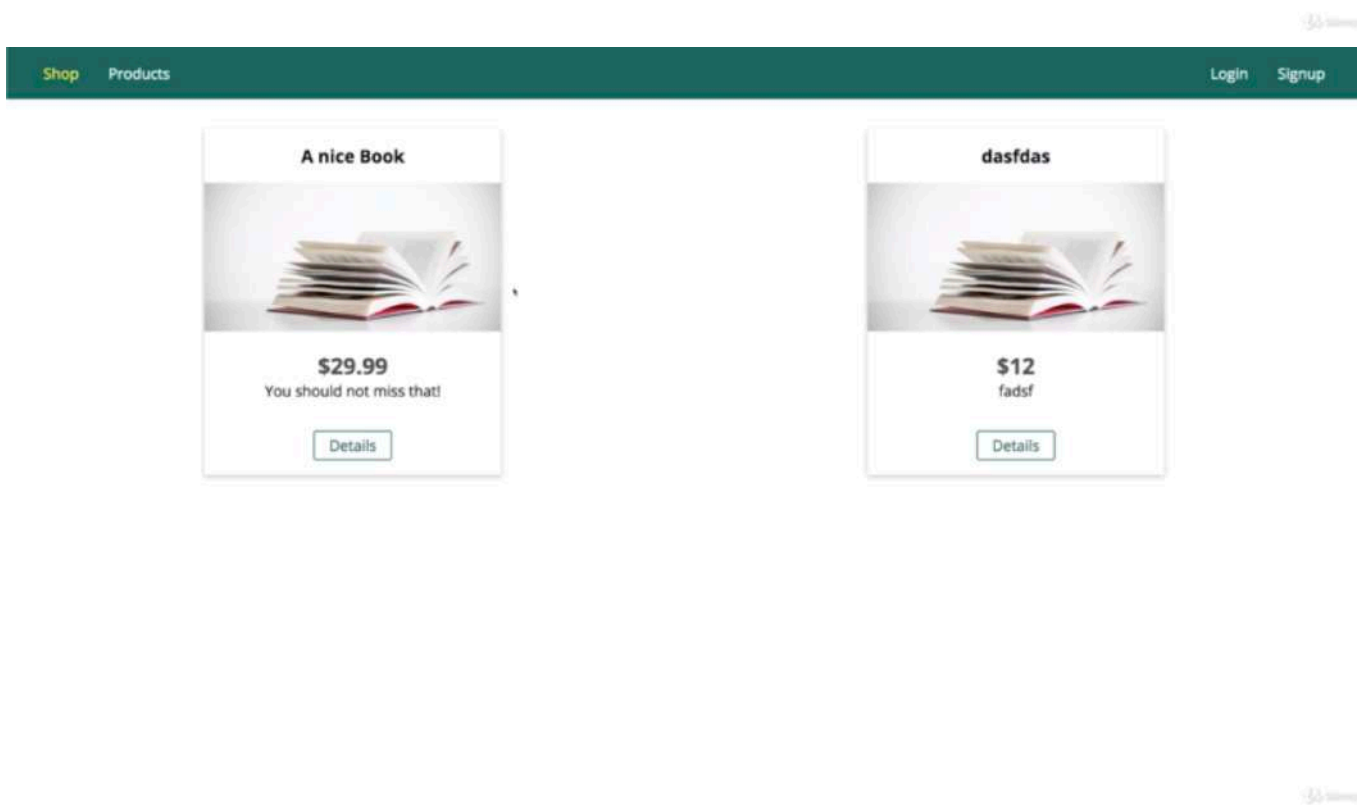
Order - # 5baddef8f37f6350c4a541ca

A nice Book (2)

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout

Order - # 5baddef8f37f6350c4a541ca

A nice Book (2)



```

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

```

```

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

```

```

70     return req.user.addToCart(product);
71   })
72   .then(result => {
73     console.log(result);
74     res.redirect('/cart');
75   });
76 };
77
78 exports.postCartDeleteProduct = (req, res, next) => {
79   const prodId = req.body.productId;
80   req.user
81     .removeFromCart(prodId)
82     .then(result => {
83       res.redirect('/cart');
84     })
85     .catch(err => console.log(err));
86 };
87
88 exports.postOrder = (req, res, next) => {
89   req.user
90     .populate('cart.items.productId')
91     .execPopulate()
92     .then(user => {
93       const products = user.cart.items.map(i => {
94         return { quantity: i.quantity, product: { ...i.productId._doc } };
95       });
96       const order = new Order({
97         user: {
98           email: req.user.email,
99           userId: req.user
100         },
101         products: products
102       });
103       return order.save();
104     })
105     .then(result => {
106       return req.user.clearCart();
107     })
108     .then(() => {
109       res.redirect('/orders');
110     })
111     .catch(err => console.log(err));
112 };
113
114 exports.getOrders = (req, res, next) => {
115   Order.find({ 'user.userId': req.user._id })
116     .then(orders => {
117       res.render('shop/orders', {
118         path: '/orders',
119         pageTitle: 'Your Orders',
120         orders: orders,
121         isAuthenticated: req.session.isLoggedIn
122       });
123     })
124     .catch(err => console.log(err));
125 };

```

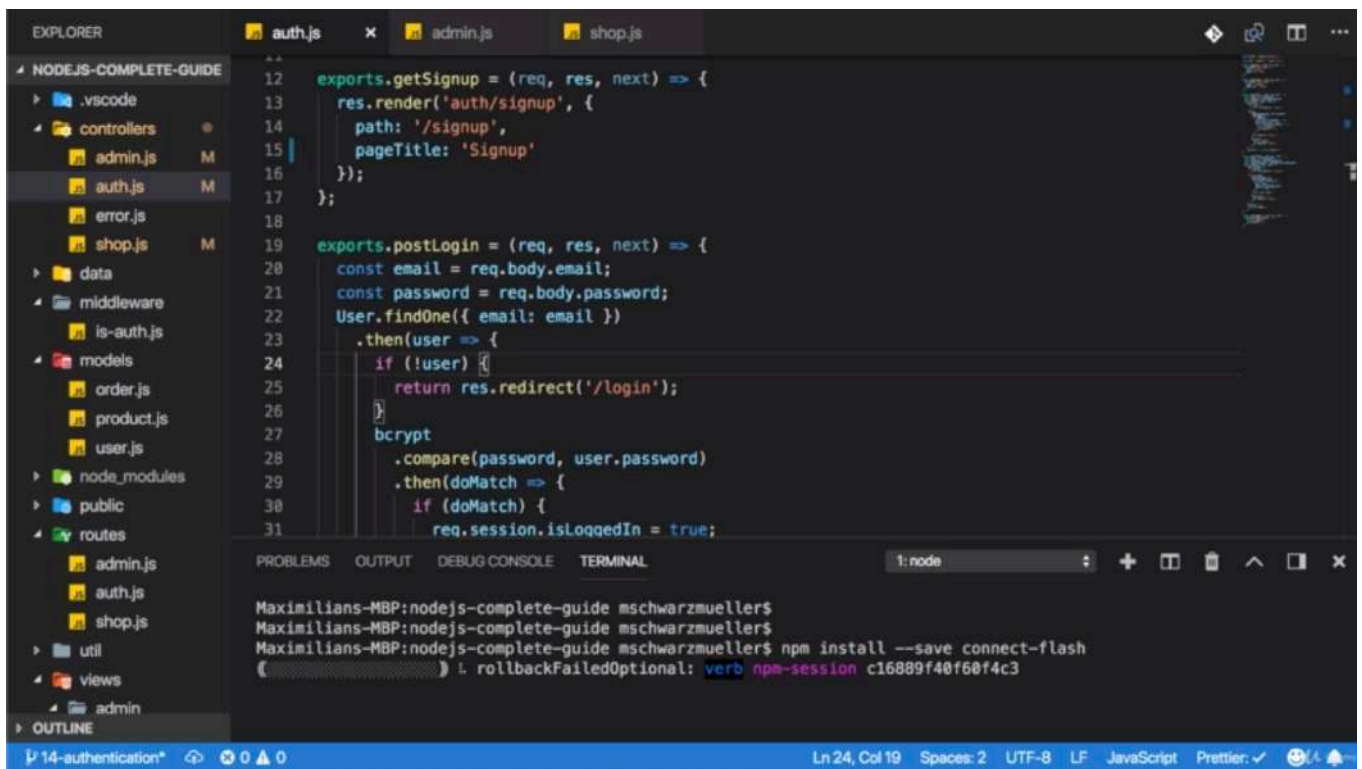

126

127

```
1 // ./models/order.js
2
3 const mongoose = require('mongoose');
4
5 const Schema = mongoose.Schema;
6
7 const orderSchema = new Schema({
8   products: [
9     {
10      product: { type: Object, required: true },
11      quantity: { type: Number, required: true }
12    }
13  ],
14   user: {
15     email: {
16       type: String,
17       required: true
18     },
19     userId: {
20       type: Schema.Types.ObjectId,
21       required: true,
22       ref: 'User'
23     }
24   }
25 });
26
27 module.exports = mongoose.model('Order', orderSchema);
28
```

* Chapter 260: Providing User Feedback

- 1. update
- ./controllers/auth.js
- ./controllers/admin.js
- ./controllers/shop.js
- app.js
- ./views/auth/login.ejs



```
12 exports.getSignup = (req, res, next) => {
13   res.render('auth/signup', {
14     path: '/signup',
15     pageTitle: 'Signup'
16   });
17 };
18
19 exports.postLogin = (req, res, next) => {
20   const email = req.body.email;
21   const password = req.body.password;
22   User.findOne({ email: email })
23     .then(user => {
24       if (!user) {
25         return res.redirect('/login');
26       }
27       bcrypt
28         .compare(password, user.password)
29         .then(doMatch => {
30           if (doMatch) {
31             req.session.isLoggedIn = true;
```

```
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save connect-flash
( ) 1 rollbackFailedOptional: verb npm-session c16889f40f60f4c3
```

- if we wanna pass some data into the rendered view when we are redirecting as we are doing it here because upon a redirect, technically a new request is started, a new request to '/login' in this case. we don't know that we got here because the user entered an invalid email or anything like that when we triggered this request, this is treated in the same way as a request that was triggered by clicking on the login button in our menu
 - so we have no way of finding out if we wanna display an error message or not. to solve that problem and store some data before we redirect which we then use in the brand new request that is triggered by the redirect how could we do that?
 - if you wanna store data across requests, you need a session. so we can use a session here but i don't wanna store the error message in the session permanently. i wanna add something to the error message, kind of flash it onto the session. and once the error message was then used, so once we pull it out of the session and did something with it. i wanna remove it from the session. so that for the subsequent requests, this error message is not part of the session anymore for that we can use another package called 'connect-flash'
 - 'npm install --save connect-flash'
-
-

E-Mail
test@test.com

Password

Login

Invalid email or password.

E-Mail

Password

Login

- let me enter an invalid email, i see invalid e-mail or password.

[Shop](#) [Products](#) [Login](#) [Signup](#)

Invalid email or password.

E-Mail

test2@test.com

Password

••

Login

[Shop](#) [Products](#) [Login](#) [Signup](#)

E-Mail

Password

Login

- if i enter a valid email but an invalid password, i don't see any error message because i don't flash anything onto my session yet.

```
1 //./controllers/auth.js
2
3 const bcrypt = require('bcryptjs');
4
5 const User = require('../models/user');
6
7 exports.getLogin = (req, res, next) => {
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    /**we now access the key for which we wanna to get the message
```

```

12     * whatever i stored in error will be retrieved and stored in error message
13     * and thereafter, this information is removed from the session.
14     *
15     * so 'errorMessage' will be set and will hold a value
16     * only if we have an error flashed into our session
17     */
18     errorMessage: req.flash('error')
19   });
20 };
21
22 exports.getSignup = (req, res, next) => {
23   res.render('auth/signup', {
24     path: '/signup',
25     pageTitle: 'Signup'
26   });
27 };
28
29 exports.postLogin = (req, res, next) => {
30   const email = req.body.email;
31   const password = req.body.password;
32   User.findOne({ email: email })
33     .then(user => {
34       if (!user) {
35         /**'flash()' takes a key under which this message will be stored
36         * we could name this 'error' and the message
37         * and in this case, this would be invalid e-mail or password
38         */
39         req.flash('error', 'Invalid email or password.')
40         return res.redirect('/login');
41       }
42       bcrypt
43         .compare(password, user.password)
44         .then(doMatch => {
45           if (doMatch) {
46             req.session.isLoggedIn = true;
47             req.session.user = user;
48             return req.session.save(err => {
49               console.log(err);
50               res.redirect('/');
51             });
52           }
53           res.redirect('/login');
54         })
55         .catch(err => {
56           console.log(err);
57           res.redirect('/login');
58         });
59     })
60     .catch(err => console.log(err));
61 };
62
63 exports.postSignup = (req, res, next) => {
64   const email = req.body.email;
65   const password = req.body.password;
66   const confirmPassword = req.body.confirmPassword;
67   User.findOne({ email: email })

```

```

68     .then(userDoc => {
69         if (userDoc) {
70             return res.redirect('/signup');
71         }
72         return bcrypt
73             .hash(password, 12)
74             .then(hashedPassword => {
75                 const user = new User({
76                     email: email,
77                     password: hashedPassword,
78                     cart: { items: [] }
79                 });
80                 return user.save();
81             })
82             .then(result => {
83                 res.redirect('/login');
84             });
85     })
86     .catch(err => {
87         console.log(err);
88     });
89 };
90
91 exports.postLogout = (req, res, next) => {
92     req.session.destroy(err => {
93         console.log(err);
94         res.redirect('/');
95     });
96 };

```

```

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

```

```

28     // console.log(result);
29     console.log('Created Product');
30     res.redirect('/admin/products');
31 }
32 .catch(err => {
33     console.log(err);
34 });
35 };
36
37 exports.getEditProduct = (req, res, next) => {
38     const editMode = req.query.edit;
39     if (!editMode) {
40         return res.redirect('/');
41     }
42     const prodId = req.params.productId;
43     Product.findById(prodId)
44         .then(product => {
45             if (!product) {
46                 return res.redirect('/');
47             }
48             res.render('admin/edit-product', {
49                 pageTitle: 'Edit Product',
50                 path: '/admin/edit-product',
51                 editing: editMode,
52                 product: product
53             });
54         })
55         .catch(err => console.log(err));
56 };
57
58 exports.postEditProduct = (req, res, next) => {
59     const prodId = req.body.productId;
60     const updatedTitle = req.body.title;
61     const updatedPrice = req.body.price;
62     const updatedImageUrl = req.body.imageUrl;
63     const updatedDesc = req.body.description;
64
65     Product.findById(prodId)
66         .then(product => {
67             product.title = updatedTitle;
68             product.price = updatedPrice;
69             product.description = updatedDesc;
70             product.imageUrl = updatedImageUrl;
71             return product.save();
72         })
73         .then(result => {
74             console.log('UPDATED PRODUCT!');
75             res.redirect('/admin/products');
76         })
77         .catch(err => console.log(err));
78 };
79
80 exports.getProducts = (req, res, next) => {
81     Product.find()
82         // .select('title price -_id')
83         // .populate('userId', 'name')

```

```

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

```

```

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

```



```

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

```

```

92     });
93     const order = new Order({
94       user: {
95         email: req.user.email,
96         userId: req.user
97       },
98       products: products
99     });
100    return order.save();
101  })
102  .then(result => {
103    return req.user.clearCart();
104  })
105  .then(() => {
106    res.redirect('/orders');
107  })
108  .catch(err => console.log(err));
109 };
110
111 exports.getOrders = (req, res, next) => {
112   Order.find({ 'user.userId': req.user._id })
113     .then(orders => {
114       res.render('shop/orders', {
115         path: '/orders',
116         pageTitle: 'Your Orders',
117         orders: orders
118       });
119     })
120     .catch(err => console.log(err));
121 };
122
123

```

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7  const mongoose = require('mongoose');
8  const session = require('express-session');
9  const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf')
11 const flash = require('connect-flash')
12
13 const errorController = require('./controllers/error');
14 const User = require('./models/user');
15
16 const MONGODB_URI =
17   'mongodb+srv://maximilian:rldnjs12@cluster0-z3vbk.mongodb.net/shop?retryWrites=true';
18
19 const app = express();
20 const store = new MongoDBStore({
21   uri: MONGODB_URI,
22   collection: 'sessions'
23 });
24 const csrfProtection = csrf()

```

```

25
26 app.set('view engine', 'ejs');
27 app.set('views', 'views');
28
29 const adminRoutes = require('./routes/admin');
30 const shopRoutes = require('./routes/shop');
31 const authRoutes = require('./routes/auth');
32
33 app.use(bodyParser.urlencoded({ extended: false }));
34 app.use(express.static(path.join(__dirname, 'public')));
35 app.use(
36   session({
37     secret: 'my secret',
38     resave: false,
39     saveUninitialized: false,
40     store: store
41   })
42 );
43 app.use(csrfProtection)
44 /**now we can use that flash middleware anywhere in our application on the request object */
45 app.use(flash())
46
47 app.use((req, res, next) => {
48   if (!req.session.user) {
49     return next();
50   }
51   User.findById(req.session.user._id)
52     .then(user => {
53       req.user = user;
54       next();
55     })
56     .catch(err => console.log(err));
57 });
58
59 app.use((req, res, next) => {
60   res.locals.isAuthenticated = req.session.isLoggedIn
61   res.locals.csrfToken = req.csrfToken()
62   next()
63 })
64
65 app.use('/admin', adminRoutes);
66 app.use(shopRoutes);
67 app.use(authRoutes);
68
69 app.use(errorController.get404);
70
71 mongoose
72   .connect(MONGODB_URI)
73   .then(result => {
74     app.listen(3000);
75   })
76   .catch(err => {
77     console.log(err);
78   });
79

```

```

1 <!--./views/auth/login.ejs-->

```

```

2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9   <%- include('../includes/navigation.ejs') %>
10
11   <main>
12     <% if (errorMessage) { %>
13       <div><%= errorMessage %></div>
14     <% } %>
15     <form class="login-form" action="/login" method="POST">
16       <div class="form-control">
17         <label for="email">E-Mail</label>
18         <input type="email" name="email" id="email">
19       </div>
20       <div class="form-control">
21         <label for="password">Password</label>
22         <input type="password" name="password" id="password">
23       </div>
24       <input type="hidden" name="_csrf" value="<%= csrfToken %>">
25       <button class="btn" type="submit">Login</button>
26     </form>
27   </main>
28 <%- include('../includes/end.ejs') %>

```

* Chapter 261: Optional: Styling Error Messages

The screenshot shows a web application interface. At the top, there is a dark green navigation bar with the text "Shop" and "Products" on the left, and "Login" and "Signup" on the right. Below the navigation bar, a red error message "Invalid email or password." is displayed. The main content area contains a login form. The form has two input fields: "E-Mail" with the value "test@test.com" and "Password" with a single asterisk "*" visible. Below the password field is a blue "Login" button.

Invalid email or password.

E-Mail

*

Password

Login

- let me try entering an invalid email again.

Invalid email or password.

E-Mail

test@test.com

Password

Login

Invalid email or password.

E-Mail

Password

Login

```

1 <!--./views/auth/login.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9   <%- include('../includes/navigation.ejs') %>
10
11   <main>
12     <% if (errorMessage) { %>
13       <div class="user-message user-message--error"><%= errorMessage %></div>
14     <% } %>
15     <form class="login-form" action="/login" method="POST">
16       <div class="form-control">
17         <label for="email">E-Mail</label>
18         <input type="email" name="email" id="email">
19       </div>
20       <div class="form-control">
21         <label for="password">Password</label>
22         <input type="password" name="password" id="password">
23       </div>
24       <input type="hidden" name="_csrf" value="<%= csrfToken %>">
25       <button class="btn" type="submit">Login</button>
26     </form>
27   </main>
28 <%- include('../includes/end.ejs') %>

```

```

1 /*./public/css/main.css*/
2
3 @import url('https://fonts.googleapis.com/css?family=Open+Sans:400,700');
4
5 * {
6   box-sizing: border-box;

```

```
7 }
8
9 body {
10   padding: 0;
11   margin: 0;
12   font-family: 'Open Sans', sans-serif;
13 }
14
15 main {
16   padding: 1rem;
17   margin: auto;
18 }
19
20 form {
21   display: inline;
22 }
23
24 .centered {
25   text-align: center;
26 }
27
28 .image {
29   height: 20rem;
30 }
31
32 .image img {
33   height: 100%;
34 }
35
36 .main-header {
37   width: 100%;
38   height: 3.5rem;
39   background-color: #00695c;
40   padding: 0 1.5rem;
41   display: flex;
42   align-items: center;
43 }
44
45 .main-header__nav {
46   height: 100%;
47   width: 100%;
48   display: flex;
49   align-items: center;
50   justify-content: space-between;
51 }
52
53 .main-header__item-list {
54   list-style: none;
55   margin: 0;
56   padding: 0;
57   display: flex;
58 }
59
60 .main-header__item {
61   margin: 0 1rem;
62   padding: 0;
```

```
63 }
64
65 .main-header__item a,
66 .main-header__item button {
67     font: inherit;
68     background: transparent;
69     border: none;
70     text-decoration: none;
71     color: white;
72     cursor: pointer;
73 }
74
75 .main-header__item a:hover,
76 .main-header__item a:active,
77 .main-header__item a.active,
78 .main-header__item button:hover,
79 .main-header__item button:active {
80     color: #ffeb3b;
81 }
82
83 .mobile-nav {
84     width: 30rem;
85     height: 100vh;
86     max-width: 90%;
87     position: fixed;
88     left: 0;
89     top: 0;
90     background: white;
91     z-index: 10;
92     padding: 2rem 1rem 1rem 2rem;
93     transform: translateX(-100%);
94     transition: transform 0.3s ease-out;
95 }
96
97 .mobile-nav.open {
98     transform: translateX(0);
99 }
100
101 .mobile-nav__item-list {
102     list-style: none;
103     display: flex;
104     flex-direction: column;
105     margin: 0;
106     padding: 0;
107 }
108
109 .mobile-nav__item {
110     margin: 1rem;
111     padding: 0;
112 }
113
114 .mobile-nav__item a {
115     text-decoration: none;
116     color: black;
117     font-size: 1.5rem;
118     padding: 0.5rem 2rem;
```



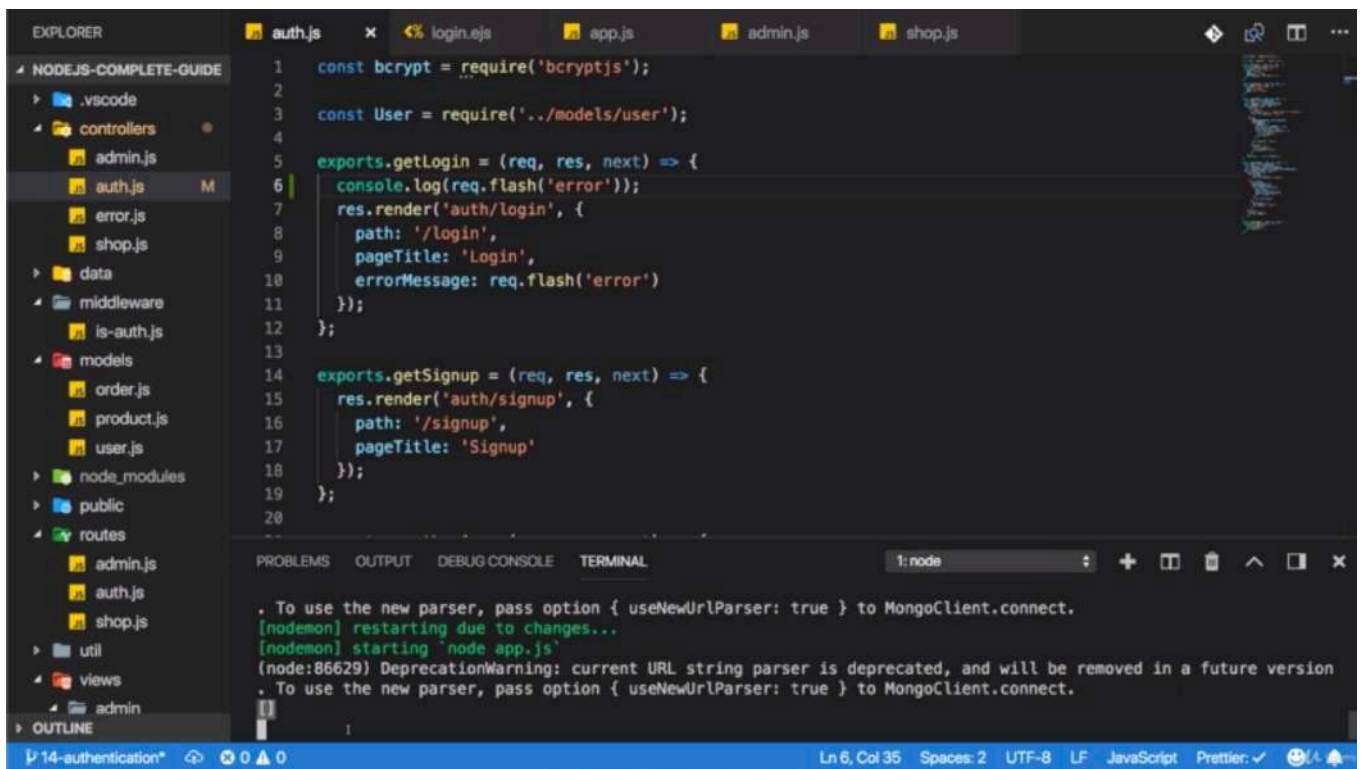
```
119 }
120
121 .mobile-nav__item a:active,
122 .mobile-nav__item a:hover,
123 .mobile-nav__item a.active {
124     background: #00695c;
125     color: white;
126     border-radius: 3px;
127 }
128
129 #side-menu-toggle {
130     border: 1px solid white;
131     font: inherit;
132     padding: 0.5rem;
133     display: block;
134     background: transparent;
135     color: white;
136     cursor: pointer;
137 }
138
139 #side-menu-toggle:focus {
140     outline: none;
141 }
142
143 #side-menu-toggle:active,
144 #side-menu-toggle:hover {
145     color: #ffeb3b;
146     border-color: #ffeb3b;
147 }
148
149 .backdrop {
150     position: fixed;
151     top: 0;
152     left: 0;
153     width: 100%;
154     height: 100vh;
155     background: rgba(0, 0, 0, 0.5);
156     z-index: 5;
157     display: none;
158 }
159
160 .grid {
161     display: flex;
162     flex-wrap: wrap;
163     justify-content: space-around;
164     align-items: stretch;
165 }
166
167 .card {
168     box-shadow: 0 2px 8px rgba(0, 0, 0, 0.26);
169 }
170
171 .card__header,
172 .card__content {
173     padding: 1rem;
174 }
```

```
175
176 .card__header h1,
177 .card__content h1,
178 .card__content h2,
179 .card__content p {
180     margin: 0;
181 }
182
183 .card__image {
184     width: 100%;
185 }
186
187 .card__image img {
188     width: 100%;
189 }
190
191 .card__actions {
192     padding: 1rem;
193     text-align: center;
194 }
195
196 .card__actions button,
197 .card__actions a {
198     margin: 0 0.25rem;
199 }
200
201 .btn {
202     display: inline-block;
203     padding: 0.25rem 1rem;
204     text-decoration: none;
205     font: inherit;
206     border: 1px solid #00695c;
207     color: #00695c;
208     background: white;
209     border-radius: 3px;
210     cursor: pointer;
211 }
212
213 .btn:hover,
214 .btn:active {
215     background-color: #00695c;
216     color: white;
217 }
218
219 .btn.danger {
220     color: red;
221     border-color: red;
222 }
223
224 .btn.danger:hover,
225 .btn.danger:active {
226     background: red;
227     color: white;
228 }
229
230 .user-message {
```

```
231 margin: auto;
232 width: 90%;
233 border: 1px solid #4771fa;
234 padding: 0.5rem;
235 border-radius: 3px;
236 background: #8ea8fd;
237 text-align: center;
238 }
239
240 .user-message--error {
241   border-color: red;
242   background: rgb(255, 176, 176);
243   color: red;
244 }
245
246 @media (min-width: 768px) {
247   .main-header__nav {
248     display: flex;
249   }
250
251   #side-menu-toggle {
252     display: none;
253   }
254
255   .user-message {
256     width: 30rem;
257   }
258 }
```

* Chapter 262: Finishing The Flash Messages

1. update
- ./controllers/auth.js



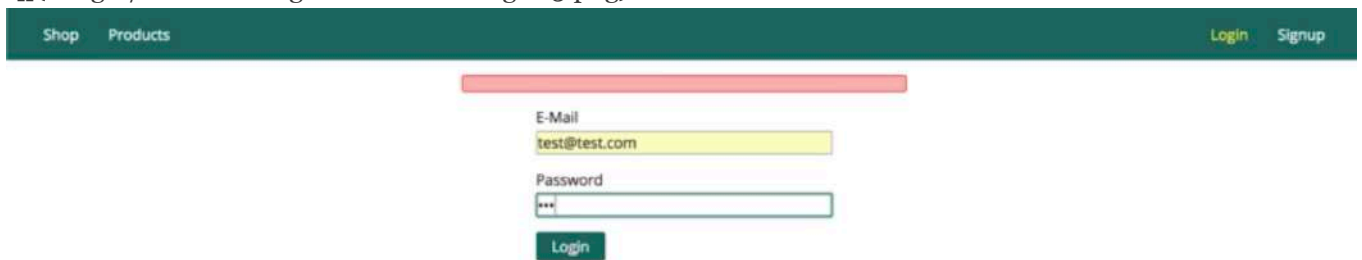
The screenshot shows a VS Code editor with the file explorer on the left displaying a project structure for '14-authentication'. The main editor window shows the `auth.js` file with the following code:

```
1 const bcrypt = require('bcryptjs');
2
3 const User = require('../models/user');
4
5 exports.getLogin = (req, res, next) => {
6   console.log(req.flash('error'));
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    errorMessage: req.flash('error')
11  });
12 };
13
14 exports.getSignup = (req, res, next) => {
15   res.render('auth/signup', {
16     path: '/signup',
17     pageTitle: 'Signup'
18   });
19 };
20
```

The terminal window at the bottom shows the following output:

```
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:86629) 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.
```

- if i console log `req.flash('error')`, i see it's an empty array.



The screenshot shows a web application with a dark green header containing the links 'Shop' and 'Products' on the left, and 'Login' and 'Signup' on the right. Below the header is a white login form with a red border. The form contains the following fields and buttons:

- E-Mail:
- Password:
- Login:

The screenshot shows a VS Code editor with the file explorer on the left displaying a project structure for '14-authentication'. The main editor window shows the `auth.js` file with the following code:

```
1 const bcrypt = require('bcryptjs');
2
3 const User = require('../models/user');
4
5 exports.getLogin = (req, res, next) => {
6   console.log(req.flash('error'));
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    errorMessage: req.flash('error')
11  });
12 };
13
14 exports.getSignup = (req, res, next) => {
15   res.render('auth/signup', {
16     path: '/signup',
17     pageTitle: 'Signup'
18   });
19 };
20
```

The terminal window at the bottom shows the following output:

```
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
(node:86683) 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.
[ 'Invalid email or password.' ]
```


The screenshot shows a web application interface with a dark green header bar containing the links 'Shop' and 'Products' on the left, and 'Login' and 'Signup' on the right. Below the header, there is a login form with two input fields: 'E-Mail' containing the text 'test@test.com' and 'Password' containing a single asterisk '*'. A blue 'Login' button is positioned below the password field.

Invalid email or password.

E-Mail

Password

Login

50 lines

```
1  ../controllers/auth.js
2
3  const bcrypt = require('bcryptjs');
4
5  const User = require('../models/user');
6
7  exports.getLogin = (req, res, next) => {
8    let message = req.flash('error')
9    if(message.length > 0){
10      message = message[0]
11    } else {
12      message = null
13    }
14    res.render('auth/login', {
15      path: '/login',
16      pageTitle: 'Login',
17      errorMessage: req.flash('error')
18    });
19  };
20
21  exports.getSignup = (req, res, next) => {
22    res.render('auth/signup', {
23      path: '/signup',
24      pageTitle: 'Signup',
25      errorMessage: message
26    });
27  };
28
29  exports.postLogin = (req, res, next) => {
30    const email = req.body.email;
31    const password = req.body.password;
32    User.findOne({ email: email })
33      .then(user => {
34        if (!user) {
35          req.flash('error', 'Invalid email or password.')
```

```

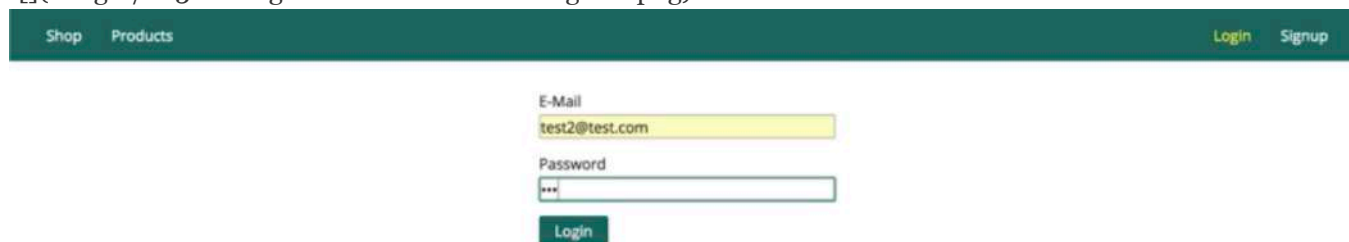
36     return res.redirect('/login');
37 }
38 bcrypt
39     .compare(password, user.password)
40     .then(doMatch => {
41         if (doMatch) {
42             req.session.isLoggedIn = true;
43             req.session.user = user;
44             return req.session.save(err => {
45                 console.log(err);
46                 res.redirect('/');
47             });
48         }
49         res.redirect('/login');
50     })
51     .catch(err => {
52         console.log(err);
53         res.redirect('/login');
54     });
55 })
56 .catch(err => console.log(err));
57 };
58
59 exports.postSignup = (req, res, next) => {
60     const email = req.body.email;
61     const password = req.body.password;
62     const confirmPassword = req.body.confirmPassword;
63     User.findOne({ email: email })
64     .then(userDoc => {
65         if (userDoc) {
66             return res.redirect('/signup');
67         }
68         return bcrypt
69             .hash(password, 12)
70             .then(hashedPassword => {
71                 const user = new User({
72                     email: email,
73                     password: hashedPassword,
74                     cart: { items: [] }
75                 });
76                 return user.save();
77             })
78             .then(result => {
79                 res.redirect('/login');
80             });
81     })
82     .catch(err => {
83         console.log(err);
84     });
85 };
86
87 exports.postLogout = (req, res, next) => {
88     req.session.destroy(err => {
89         console.log(err);
90         res.redirect('/');
91     });

```

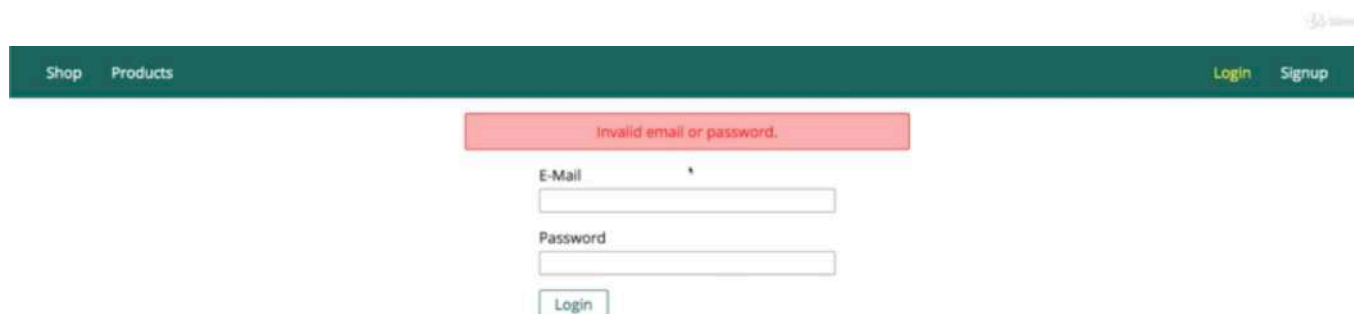
* Chapter 263: Adding Additional Flash Messages

1. update

- ./controllers/auth.js
- ./views/auth/signup.ejs



The screenshot shows a web application interface with a dark green header bar. On the left, there are links for 'Shop' and 'Products'. On the right, there are links for 'Login' and 'Signup'. Below the header, there is a login form. The form has two input fields: 'E-Mail' and 'Password'. The 'E-Mail' field contains the text 'test2@test.com'. The 'Password' field contains three asterisks '***'. Below the password field is a 'Login' button. A yellow flash message box is displayed above the 'E-Mail' field, containing the text 'test2@test.com'.



The screenshot shows the same web application interface as the previous one. However, the 'E-Mail' field is now empty. A red flash message box is displayed above the 'E-Mail' field, containing the text 'Invalid email or password.'. The 'Password' field still contains three asterisks '***'. The 'Login' button is still present.

- i might have a valid email but an invalid password and i get this error message

[Shop](#) [Products](#) [Login](#) [Signup](#)

Invalid email or password.

E-Mail


test2@test.com

Password

Login

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

A nice Book




\$29.99

You should not miss that!

Details

Add to Cart

dasfdas



\$12

fadsf

Details

Add to Cart

- and with valid credentials, it works fine though.



 (images/263-adding-additional-flash-messages-8.png)

[Shop](#)[Products](#)[Login](#)[Signup](#)

E-Mail

test2@test.com

Password

Confirm Password

Signup »

[Shop](#)[Products](#)[Login](#)[Signup](#)

E-Mail

Password

Confirm Password

Signup

- if i enter the E-mail address which does already exist, i'm redirected without having any effect.

E-Mail

test2@test.com

Password

Confirm Password

Signup



E-Mail exists already, please pick a different one.

E-Mail

Password

Confirm Password

Signup



[Shop](#) [Products](#) [Login](#) [Signup](#)

E-Mail exists already, please pick a different one.

E-Mail

test@test.com

Password

Confirm Password

Signup

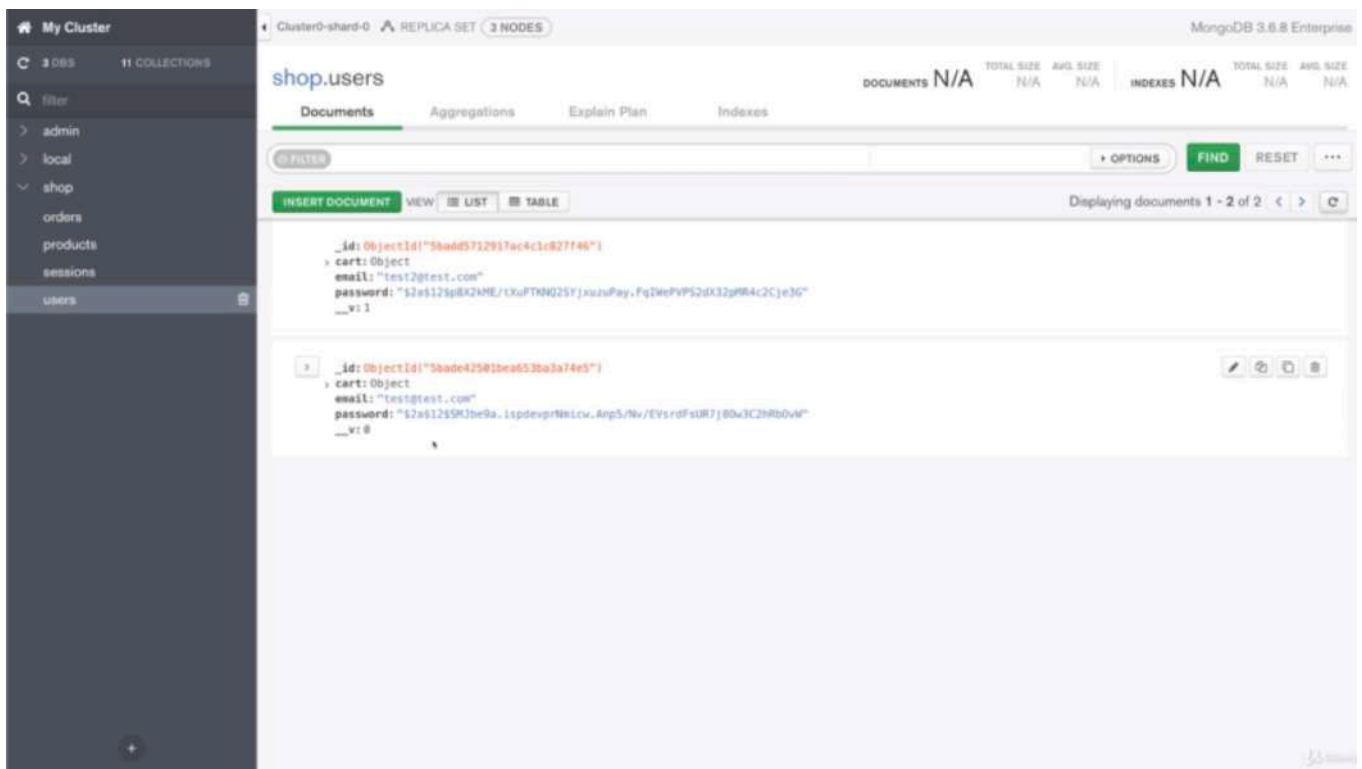
[Shop](#) [Products](#) [Login](#) [Signup](#)

E-Mail

Password

Login

- if i create a valid user with a valid email, this works.



- we can see that was added.

```

1  ../controllers/auth.js
2
3  const bcrypt = require('bcryptjs');
4
5  const User = require('../models/user');
6
7  exports.getLogin = (req, res, next) => {
8    let message = req.flash('error')
9    if(message.length > 0){
10      message = message[0]
11    } else {
12      message = null
13    }
14    res.render('auth/login', {
15      path: '/login',
16      pageTitle: 'Login',
17      errorMessage: message
18    });
19  };
20
21 exports.getSignup = (req, res, next) => {
22   let message = req.flash('error')
23   if(message.length > 0){
24     message = message[0]
25   } else {
26     message = null
27   }
28   res.render('auth/signup', {
29     path: '/signup',
30     pageTitle: 'Signup',
31     errorMessage: message
32   });
33 };

```

```

34
35 exports.postLogin = (req, res, next) => {
36   const email = req.body.email;
37   const password = req.body.password;
38   User.findOne({ email: email })
39     .then(user => {
40       if (!user) {
41         req.flash('error', 'Invalid email or password.')
42         return res.redirect('/login');
43       }
44       bcrypt
45         .compare(password, user.password)
46         .then(doMatch => {
47           if (doMatch) {
48             req.session.isLoggedIn = true;
49             req.session.user = user;
50             return req.session.save(err => {
51               console.log(err);
52               res.redirect('/');
53             });
54           }
55           /**if we have a password that doesn't match,
56            * here is where we also redirect to login,
57            * i also want to flash my invalid email or password message onto the session
58            * this is another great place
59            */
60           req.flash('error', 'Invalid email or password.')
61           res.redirect('/login');
62         })
63         .catch(err => {
64           console.log(err);
65           res.redirect('/login');
66         });
67     })
68     .catch(err => console.log(err));
69 };
70
71 exports.postSignup = (req, res, next) => {
72   const email = req.body.email;
73   const password = req.body.password;
74   const confirmPassword = req.body.confirmPassword;
75   User.findOne({ email: email })
76     .then(userDoc => {
77       if (userDoc) {
78         req.flash('error', 'E-Mail exists already, please pick a different one. ')
79         return res.redirect('/signup');
80       }
81       return bcrypt
82         .hash(password, 12)
83         .then(hashPassword => {
84           const user = new User({
85             email: email,
86             password: hashPassword,
87             cart: { items: [] }
88           });
89           return user.save();

```

```

90     })
91     .then(result => {
92       res.redirect('/login');
93     });
94   })
95   .catch(err => {
96     console.log(err);
97   });
98 };
99
100 exports.postLogout = (req, res, next) => {
101   req.session.destroy(err => {
102     console.log(err);
103     res.redirect('/');
104   });
105 };

```

```

1 <!--./views/auth/signup.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/auth.css">
6 </head>
7
8 <body>
9   <%- include('../includes/navigation.ejs') %>
10
11   <main>
12     <% if (errorMessage) { %>
13       <div class="user-message user-message--error"><%= errorMessage %></div>
14     <% } %>
15     <form class="login-form" action="/signup" method="POST">
16       <div class="form-control">
17         <label for="email">E-Mail</label>
18         <input type="email" name="email" id="email">
19       </div>
20       <div class="form-control">
21         <label for="password">Password</label>
22         <input type="password" name="password" id="password">
23       </div>
24       <div class="form-control">
25         <label for="confirmPassword">Confirm Password</label>
26         <input type="password" name="confirmPassword" id="confirmPassword">
27       </div>
28       <input type="hidden" name="_csrf" value="<%= csrfToken %>">
29       <button class="btn" type="submit">Signup</button>
30     </form>
31   </main>
32 <%- include('../includes/end.ejs') %>

```

* Chapter 264: Wrap Up

Module Summary

Authentication

- Authentication means that not every visitor of the page can view and interact with everything
- Authentication has to happen on the server-side and builds up on sessions
- You can protect routes by checking the (session-controlled) login status right before you access a controller action

Security & UX

- Passwords should be stored in a hashed form
- CSRF attacks are a real issue and you should therefore include CSRF protection in ANY application you build!
- For a better user experience, you can flash data/ messages into the session which you then can display in your views