

17. Advanced Authentication

* Chapter 272: Module Introduction



What's In This Module?

Resetting Passwords

Authorization

Academind

* Chapter 273: Resetting Passwords

1. update
- ./views/auth/reset.ejs
- ./controllers/auth.js
- ./routes/auth.js
- ./views/auth/login.ejs

- let's set the resetting passwords. that's a common thing you need to do in applications. people forget the password. you wanna offer them a way of resetting them. for that, we will need a new view and some new routes.

E-Mail

Password

[Reset Password](#)

 login

E-Mail

 login

```
1 <!--./views/auth/reset.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          <input type="hidden" name="_csrf" value="<%= csrfToken %>">
21          <button class="btn" type="submit">Reset Password</button>
22      </form>
23  </main>
24 <%-- include('../includes/end.ejs') %>

```

```

1 //./controllers/auth.js
2
3 const bcrypt = require('bcryptjs');
4 const nodemailer = require('nodemailer')
5 const sendgridTransport = require('nodemailer-sendgrid-transport')
6
7 const User = require('../models/user');
8
9 const transporter = nodemailer.createTransport(sendgridTransport({
10   auth: {
11     api_key: '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'
12   }
13 }));
14
15 exports.getLogin = (req, res, next) => {
16   let message = req.flash('error')
17   if(message.length > 0){
18     message = message[0]
19   } else {
20     message = null
21   }
22   res.render('auth/login', {
23     path: '/login',
24     pageTitle: 'Login',
25     errorMessage: message
26   });
27 };
28
29 exports.getSignup = (req, res, next) => {
30   let message = req.flash('error')
31   if(message.length > 0){
32     message = message[0]
33   } else {
34     message = null
35   }
36   res.render('auth/signup', {
37     path: '/signup',
38     pageTitle: 'Signup',
39     errorMessage: message
40   });
41 };
42
43 exports.postLogin = (req, res, next) => {
44   const email = req.body.email;
45   const password = req.body.password;
46   User.findOne({ email: email })

```

```

47 .then(user => {
48   if (!user) {
49     req.flash('error', 'Invalid email or password.')
50     return res.redirect('/login');
51   }
52   bcrypt
53     .compare(password, user.password)
54     .then(doMatch => {
55       if (doMatch) {
56         req.session.isLoggedIn = true;
57         req.session.user = user;
58         return req.session.save(err => {
59           console.log(err);
60           res.redirect('/');
61         });
62       }
63       req.flash('error', 'Invalid email or password.')
64       res.redirect('/login');
65     })
66     .catch(err => {
67       console.log(err);
68       res.redirect('/login');
69     });
70   })
71   .catch(err => console.log(err));
72 };
73
74 exports.postSignup = (req, res, next) => {
75   const email = req.body.email;
76   const password = req.body.password;
77   const confirmPassword = req.body.confirmPassword;
78   User.findOne({ email: email })
79     .then(userDoc => {
80       if (userDoc) {
81         req.flash('error', 'E-Mail exists already, please pick a different one. ')
82         return res.redirect('/signup');
83     }
84     return bcrypt
85       .hash(password, 12)
86       .then(hashedPassword => {
87         const user = new User({
88           email,
89           password: hashedPassword,
90           cart: { items: [] }
91         });
92         return user.save();
93     })
94     .then(result => {
95       res.redirect('/login');
96       return transporter.sendMail({
97         to: email,
98         from: 'shop@nodecomplete.com',
99         subject: 'Signup Succeeded!',
100        html: '<h1>You Successfully Signed Up!</h1>'
101      })
102    })

```

```

103     .catch(err => {
104       console.log(err)
105     })
106   })
107   .catch(err => {
108     console.log(err);
109   });
110 };
111
112 exports.postLogout = (req, res, next) => {
113   req.session.destroy(err => {
114     console.log(err);
115     res.redirect('/');
116   });
117 };
118
119 exports.getReset = (req, res, next) => {
120   let message = req.flash('error')
121   if(message.length > 0){
122     message = message[0]
123   } else {
124     message = null
125   }
126   res.render('auth/reset', {
127     path: '/reset',
128     pageTitle: 'Reset Password',
129     errorMessage: message
130   });
131
132 }

```

```

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 router.get('/reset', authController.getReset)
20
21 module.exports = router;

```

```

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    <div class="centered">
28      <a href="/reset">Reset Password</a>
29    </div>
30  </main>
31 <%- include('../includes/end.ejs') %>

```

* Chapter 274: Implementing The Token Logic

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

E-Mail

Reset Password

- we wanna able to enter an email address here and then receive an email with a link that allows us to reset the password.

- for that, we need to first of all create a unique token that also has some expiry date which we will store in our database. so that the link which we didn't click includes that token and we can verify that the user did get that link from us because if we let the user now change that password, we got no security mechanism in place. so we need that token to put it into the email we are about to send to only let users change the password through the email that contains that token. that's an additional security mechanism.

```
1 //./controllers/auth.js
2
3 /**
4  * node.js has a built-in 'crypto' library
5  * which helps us with creating secure random values and other things
6  * but we will need that unique secure random value here.
7 */
8 const crypto = require('crypto');
9
10 const bcrypt = require('bcryptjs');
11 const nodemailer = require('nodemailer');
12 const sendgridTransport = require('nodemailer-sendgrid-transport');
13
14 const User = require('../models/user');
15
16 const transporter = nodemailer.createTransport(
17   sendgridTransport({
18     auth: {
19       api_key:
20         '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'
21     }
22   })
23 );
24
25 exports.getLogin = (req, res, next) => {
26   let message = req.flash('error');
27   if (message.length > 0) {
```

```
28     message = message[0];
29 } else {
30     message = null;
31 }
32 res.render('auth/login', {
33     path: '/login',
34     pageTitle: 'Login',
35     errorMessage: message
36 });
37 };
38
39 exports.getSignup = (req, res, next) => {
40     let message = req.flash('error');
41     if (message.length > 0) {
42         message = message[0];
43     } else {
44         message = null;
45     }
46     res.render('auth/signup', {
47         path: '/signup',
48         pageTitle: 'Signup',
49         errorMessage: message
50     });
51 };
52
53 exports.postLogin = (req, res, next) => {
54     const email = req.body.email;
55     const password = req.body.password;
56     User.findOne({ email })
57         .then(user => {
58             if (!user) {
59                 req.flash('error', 'Invalid email or password.');
60                 return res.redirect('/login');
61             }
62             bcrypt
63                 .compare(password, user.password)
64                 .then(doMatch => {
65                     if (doMatch) {
66                         req.session.isLoggedIn = true;
67                         req.session.user = user;
68                         return req.session.save(err => {
69                             console.log(err);
70                             res.redirect('/');
71                         });
72                     }
73                     req.flash('error', 'Invalid email or password.');
74                     res.redirect('/login');
75                 })
76                 .catch(err => {
77                     console.log(err);
78                     res.redirect('/login');
79                 });
80         })
81         .catch(err => console.log(err));
82 };
83
```

```
84 exports.postSignup = (req, res, next) => {
85   const email = req.body.email;
86   const password = req.body.password;
87   const confirmPassword = req.body.confirmPassword;
88   User.findOne({ email })
89     .then(userDoc => {
90       if (userDoc) {
91         req.flash(
92           'error',
93           'E-Mail exists already, please pick a different one.'
94         );
95         return res.redirect('/signup');
96       }
97       return bcrypt
98         .hash(password, 12)
99         .then(hashedPassword => {
100       const user = new User({
101         email: email,
102         password: hashedPassword,
103         cart: { items: [] }
104       });
105       return user.save();
106     })
107     .then(result => {
108       res.redirect('/login');
109       return transporter.sendMail({
110         to: email,
111         from: 'shop@node-complete.com',
112         subject: 'Signup succeeded!',
113         html: '<h1>You successfully signed up!</h1>'
114       });
115     })
116     .catch(err => {
117       console.log(err);
118     });
119   })
120   .catch(err => {
121     console.log(err);
122   });
123 };
124
125 exports.postLogout = (req, res, next) => {
126   req.session.destroy(err => {
127     console.log(err);
128     res.redirect('/');
129   });
130 };
131
132 exports.getReset = (req, res, next) => {
133   let message = req.flash('error');
134   if (message.length > 0) {
135     message = message[0];
136   } else {
137     message = null;
138   }
139   res.render('auth/reset', {
```

```

140     path: '/reset',
141     pageTitle: 'Reset Password',
142     errorMessage: message
143   });
144 }
145
146 exports.postReset = (req, res, next) => {
147   /**2nd argument will be called once it's done.
148    * and error and buffer of bytes.
149   */
150   crypto.randomBytes(32, (err, buffer) => {
151     if (err) {
152       console.log(err);
153       return res.redirect('/reset');
154     }
155     /**we need to pass hex
156      * because that buffer will store hexadecimal values
157      * this is information toString need to convert hexadecimal values to normal ASCII
158      Characters.
159      *
160      * that token should get stored in the database
161      * and it should get stored on the user object
162      * because it belongs to that user.
163      *
164      * so first of all, go to our ./models/user.js
165      */
166
167     /**we gonna fine one user
168      * where the email matches the email we are trying to reset
169      * and that email which we are trying to reset
170      * can be extracted from the req.body.email field
171      * because on our reset view here,
172      * we do have that email field in ./views/auth/reset.ejs file
173      */
174     const token = buffer.toString('hex');
175     User.findOne({ email: req.body.email })
176       .then(user => {
177         if (!user) {
178           req.flash('error', 'No account with that email found.');
179           return res.redirect('/reset');
180         }
181         user.resetToken = token;
182         user.resetTokenExpiration = Date.now() + 3600000;
183         return user.save();
184       })
185       .then(result => {
186         res.redirect('/');
187         transporter.sendMail({
188           to: req.body.email,
189           from: 'shop@node-complete.com',
190           subject: 'Password reset',
191           /**i'm placing '${token}' in the URL
192           * because that token is then what we will later look for in the database
193           * to confirm, this link was sent by us
194           * because only we know the token.

```

```
195     */
196     html: ` 
197       <p>You requested a password reset</p>
198       <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
199       new password.</p>
200     );
201   })
202   .catch(err => {
203     console.log(err);
204   });
205 });
206 };


```

```
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   /**this is not required
17   * because not always this token will be there,
18   * only if we requested a reset
19   */
20   resetToken: String,
21   resetTokenExpiration: Date,
22   cart: {
23     items: [
24       {
25         productId: {
26           type: Schema.Types.ObjectId,
27           ref: 'Product',
28           required: true
29         },
30         quantity: { type: Number, required: true }
31       }
32     ]
33   }
34 });
35
36 userSchema.methods.addToCart = function(product) {
37   const cartProductIndex = this.cart.items.findIndex(cp => {
38     return cp.productId.toString() === product._id.toString();
39   });
40   let newQuantity = 1;
41   const updatedCartItems = [...this.cart.items];
42   if (cartProductIndex >= 0) {
43     updatedCartItems[cartProductIndex].quantity += newQuantity;
44   } else {
45     updatedCartItems.push({ productId: product._id, quantity: newQuantity });
46   }
47   this.cart.items = updatedCartItems;
48   this.save();
49   return this;
50 }


```

```

44     newQuantity = this.cart.items[cartProductIndex].quantity + 1;
45     updatedCartItems[cartProductIndex].quantity = newQuantity;
46 } else {
47     updatedCartItems.push({
48         productId: product._id,
49         quantity: newQuantity
50     });
51 }
52 const updatedCart = {
53     items: updatedCartItems
54 };
55 this.cart = updatedCart;
56 return this.save();
57 };
58
59 userSchema.methods.removeFromCart = function(productId) {
60     const updatedCartItems = this.cart.items.filter(item => {
61         return item.productId.toString() !== productId.toString();
62     });
63     this.cart.items = updatedCartItems;
64     return this.save();
65 };
66
67 userSchema.methods.clearCart = function() {
68     this.cart = { items: [] };
69     return this.save();
70 };
71
72 module.exports = mongoose.model('User', userSchema);
73
74 // const mongodb = require('mongodb');
75 // const getDb = require('../util/database').getDb;
76
77 // const ObjectId = mongodb.ObjectId;
78
79 // class User {
80 //     constructor(username, email, cart, id) {
81 //         this.name = username;
82 //         this.email = email;
83 //         this.cart = cart; // {items: []}
84 //         this._id = id;
85 //     }
86 //
87 //     save() {
88 //         const db = getDb();
89 //         return db.collection('users').insertOne(this);
90 //     }
91 //
92 //     addToCart(product) {
93 //         const cartProductIndex = this.cart.items.findIndex(cp => {
94 //             return cp.productId.toString() === product._id.toString();
95 //         });
96 //         let newQuantity = 1;
97 //         const updatedCartItems = [...this.cart.items];
98 //
99 //         if (cartProductIndex >= 0) {

```

```
100 //         newQuantity = this.cart.items[cartProductIndex].quantity + 1;
101 //         updatedCartItems[cartProductIndex].quantity = newQuantity;
102 //     } else {
103 //         updatedCartItems.push({
104 //             productId: new ObjectId(product._id),
105 //             quantity: newQuantity
106 //         });
107 //     }
108 //     const updatedCart = {
109 //         items: updatedCartItems
110 //     };
111 //     const db = getDb();
112 //     return db
113 //         .collection('users')
114 //         .updateOne(
115 //             { _id: new ObjectId(this._id) },
116 //             { $set: { cart: updatedCart } }
117 //         );
118 //     }
119
120 //     getCart() {
121 //         const db = getDb();
122 //         const productIds = this.cart.items.map(i => {
123 //             return i.productId;
124 //         });
125 //         return db
126 //             .collection('products')
127 //             .find({ _id: { $in: productIds } })
128 //             .toArray()
129 //             .then(products => {
130 //                 return products.map(p => {
131 //                     return {
132 //                         ...p,
133 //                         quantity: this.cart.items.find(i => {
134 //                             return i.productId.toString() === p._id.toString();
135 //                         }).quantity
136 //                     };
137 //                 });
138 //             });
139 //     }
140
141 //     deleteItemFromCart(productId) {
142 //         const updatedCartItems = this.cart.items.filter(item => {
143 //             return item.productId.toString() !== productId.toString();
144 //         });
145 //         const db = getDb();
146 //         return db
147 //             .collection('users')
148 //             .updateOne(
149 //                 { _id: new ObjectId(this._id) },
150 //                 { $set: { cart: { items: updatedCartItems } } }
151 //             );
152 //     }
153
154 //     addOrder() {
155 //         const db = getDb();
```

```

156 //     return this.getCart()
157 //         .then(products => {
158 //             const order = {
159 //                 items: products,
160 //                 user: {
161 //                     _id: new ObjectId(this._id),
162 //                     name: this.name
163 //                 }
164 //             };
165 //             return db.collection('orders').insertOne(order);
166 //         })
167 //         .then(result => {
168 //             this.cart = { items: [] };
169 //             return db
170 //                 .collection('users')
171 //                 .updateOne(
172 //                     { _id: new ObjectId(this._id) },
173 //                     { $set: { cart: { items: [] } } }
174 //                 );
175 //         });
176 //     }
177
178 //     getOrders() {
179 //         const db = getDb();
180 //         return db
181 //             .collection('orders')
182 //             .find({ 'user._id': new ObjectId(this._id) })
183 //             .toArray();
184 //     }
185
186 //     static findById(userId) {
187 //         const db = getDb();
188 //         return db
189 //             .collection('users')
190 //             .findOne({ _id: new ObjectId(userId) })
191 //             .then(user => {
192 //                 console.log(user);
193 //                 return user;
194 //             })
195 //             .catch(err => {
196 //                 console.log(err);
197 //             });
198 //     }
199 // }
200
201 // module.exports = User;
202

```

```

1 <!--./views/auth/reset.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') %>

```

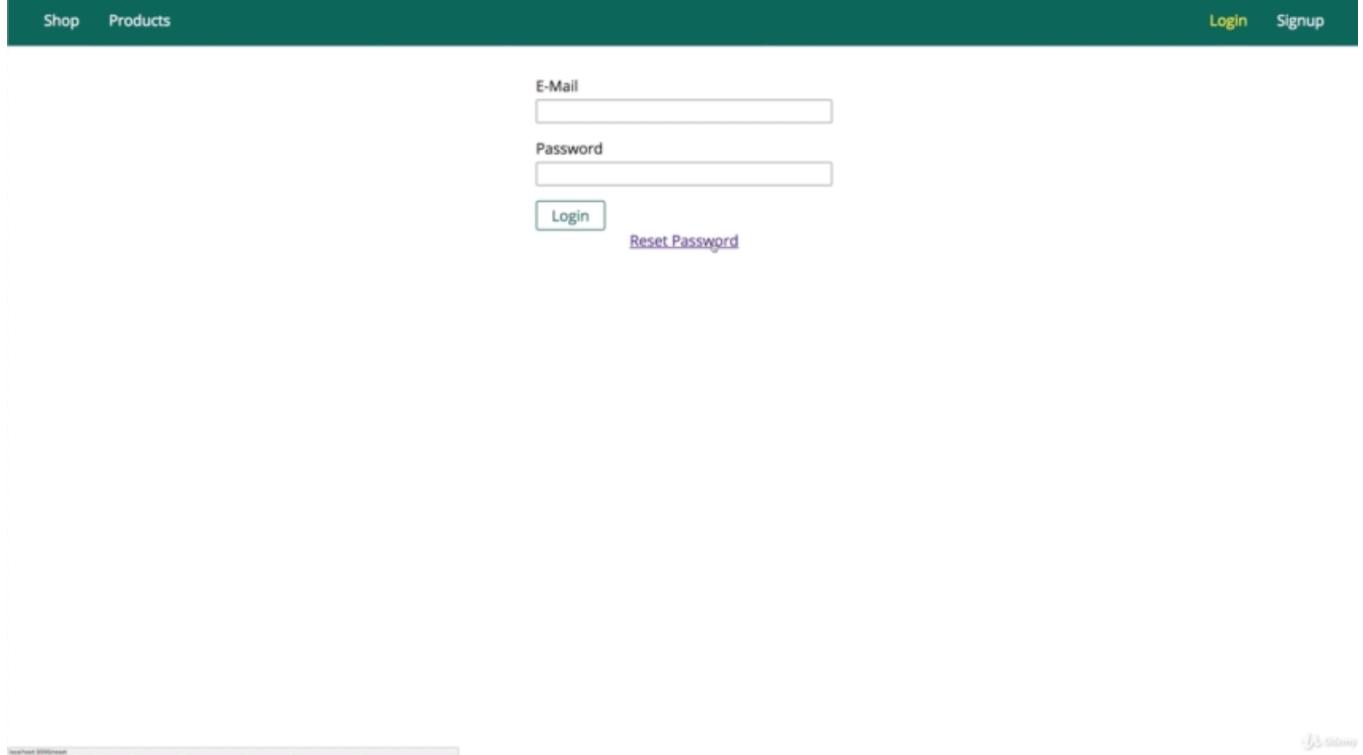
```

10
11 <main>
12   <% if (errorMessage) { %>
13     <div class="user-message user-message--error"><%= errorMessage %></div>
14   <% } %>
15   <form class="login-form" action="/reset" 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     <input type="hidden" name="_csrf" value="<%= csrfToken %>">
21     <button class="btn" type="submit">Reset Password</button>
22   </form>
23 </main>
24 <%- include('../includes/end.ejs') %>

```

* Chapter 275: Creating The Token

1. update
- routes/auth.js



E-Mail
test44@test.com

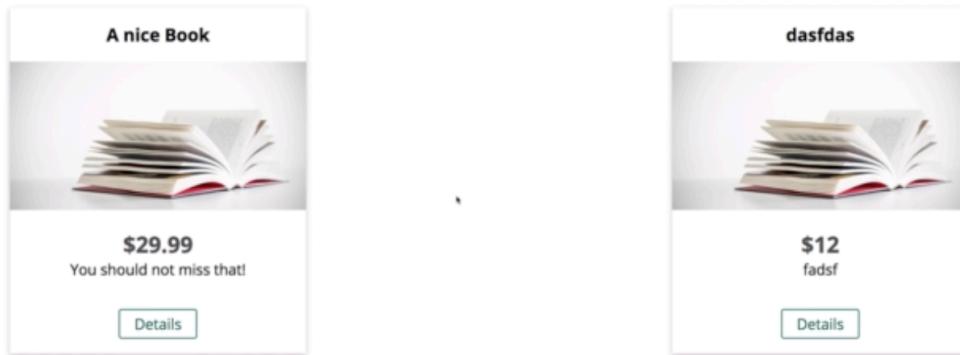
Reset Password

- click ‘Reset Password’ and try out the incorrect email which doesn’t exist in the database and we get this message which is great.

No account with that email found.

E-Mail
[REDACTED]

Reset Password



- now i use real email which you used for signing up and click 'Rest Password' and you should be redirected back.
- if you check your email there, you should have a password reset email with a link.



Page Not Found!

The screenshot shows the MongoDB Compass application. In the left sidebar, under the 'My Cluster' section, there are 3 DBs and 11 Collections. The 'shop' collection is selected. The main area displays the 'shop.users' collection with 3 documents. The first document has the following fields:

```

_id: ObjectId("5badd5712917ac4c1c827f46")
> cart: Object
email: "test2@test.com"
password: "$2a$12$pX2kME/tXuPTKQ025YjxuzuPay.FqIWePVPS2dX32pMR4c2CjeG"
__v: 1

```

The second document has the following fields:

```

_id: ObjectId("5bade42501bea653ba3a74e5")
> cart: Object
email:
password: "$2a$12$SMJbe9a.ispdevprNmicw.AnP5/Nv/EVsrdFsUR7j80w3C2hRb0vM"
__v: 0

```

The third document has the following fields:

```

_id: ObjectId("5bade9fd407f9955e3d1ba12")
> cart: Object
email:
password: "$2a$12$H1fH3fYL/1g3dQ0Kd3rPb0T3L1vCp.Xs3I9/JVdgk.3B7UMRpXWyy"
__v: 0
resetToken: "012b2414a4dd498a71efc0354165f14fcbb0416ffebcd5891251e6694ca8a13e"
resetTokenExpiration: 2018-09-28 12:13:14.178

```

- if you click that link, you should end up on a ‘page not found’ error but you will see that were on reset and then some token and that token can also be seen in your user collection here. this is the password reset token which restored
- so now we just need to add some logic to add this route, extract that token valid date wherever we have a user for that token and then offer a form that allows the user to set a new password.

```

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 router.get('/reset', authController.getReset)
20
21 router.post('/reset', authController.postReset)
22
23 module.exports = router;

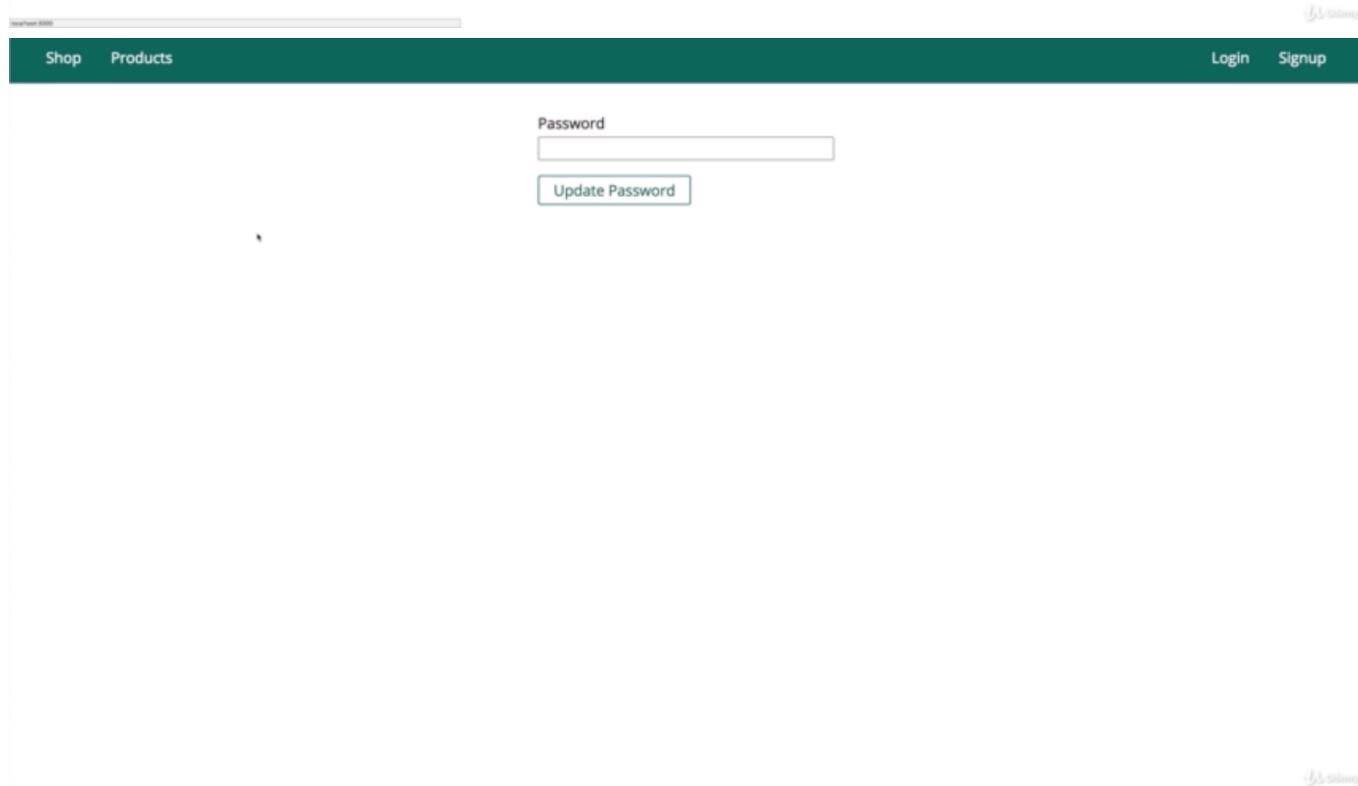
```

* Chapter 276: Creating The Reset Password Form

1. update
- ./views/auth/new-password.ejs
- ./controllers/auth.js
- ./routes/auth.js



Page Not Found!



```
1 <!--./views/auth/new-password.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="/new-password" method="POST">
16       <div class="form-control">
17         <label for="password">Password</label>
18         <input type="password" name="password" id="password">
19       </div>
20       <!--
21           i'm passing into the view
22           so does input is new with that hidden userId
23           and it will output that userId
24           that userId will need that for the POST request we wanna save our new
25           password.
26           -->
27         <input type="hidden" name="userId" value="<%= userId %>">
28         <input type="hidden" name="_csrf" value="<%= csrfToken %>">
29         <button class="btn" type="submit">Update Password</button>
30     </form>
31   </main>
32 <%- include('../includes/end.ejs') %>
```

```

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

```

```
32 };
33
34 exports.getSignup = (req, res, next) => {
35   let message = req.flash('error');
36   if (message.length > 0) {
37     message = message[0];
38   } else {
39     message = null;
40   }
41   res.render('auth/signup', {
42     path: '/signup',
43     pageTitle: 'Signup',
44     errorMessage: message
45   });
46 };
47
48 exports.postLogin = (req, res, next) => {
49   const email = req.body.email;
50   const password = req.body.password;
51   User.findOne({ email: email })
52     .then(user => {
53       if (!user) {
54         req.flash('error', 'Invalid email or password.');
55         return res.redirect('/login');
56       }
57       bcrypt
58         .compare(password, user.password)
59         .then(doMatch => {
60           if (doMatch) {
61             req.session.isLoggedIn = true;
62             req.session.user = user;
63             return req.session.save(err => {
64               console.log(err);
65               res.redirect('/');
66             });
67           }
68           req.flash('error', 'Invalid email or password.');
69           res.redirect('/login');
70         })
71         .catch(err => {
72           console.log(err);
73           res.redirect('/login');
74         });
75       })
76       .catch(err => console.log(err));
77 };
78
79 exports.postSignup = (req, res, next) => {
80   const email = req.body.email;
81   const password = req.body.password;
82   const confirmPassword = req.body.confirmPassword;
83   User.findOne({ email: email })
84     .then(userDoc => {
85       if (userDoc) {
86         req.flash(
87           'error',
```

```
88     'E-Mail exists already, please pick a different one.'
89   );
90   return res.redirect('/signup');
91 }
92 return bcrypt
93   .hash(password, 12)
94   .then(hashedPassword => {
95     const user = new User({
96       email: email,
97       password: hashedPassword,
98       cart: { items: [] }
99     });
100    return user.save();
101  })
102  .then(result => {
103    res.redirect('/login');
104    return transporter.sendMail({
105      to: email,
106      from: 'shop@node-complete.com',
107      subject: 'Signup succeeded!',
108      html: '<h1>You successfully signed up!</h1>'
109    });
110  })
111  .catch(err => {
112    console.log(err);
113  });
114 }
115 .catch(err => {
116   console.log(err);
117 });
118 };
119
120 exports.postLogout = (req, res, next) => {
121   req.session.destroy(err => {
122     console.log(err);
123     res.redirect('/');
124   });
125 };
126
127 exports.getReset = (req, res, next) => {
128   let message = req.flash('error');
129   if (message.length > 0) {
130     message = message[0];
131   } else {
132     message = null;
133   }
134   res.render('auth/reset', {
135     path: '/reset',
136     pageTitle: 'Reset Password',
137     errorMessage: message
138   });
139 };
140
141 exports.postReset = (req, res, next) => {
142   crypto.randomBytes(32, (err, buffer) => {
143     if (err) {
```

```

144     console.log(err);
145     return res.redirect('/reset');
146   }
147   const token = buffer.toString('hex');
148   User.findOne({ email: req.body.email })
149     .then(user => {
150       if (!user) {
151         req.flash('error', 'No account with that email found.');
152         return res.redirect('/reset');
153       }
154       user.resetToken = token;
155       user.resetTokenExpiration = Date.now() + 3600000;
156       return user.save();
157     })
158     .then(result => {
159       res.redirect('/');
160       transporter.sendMail({
161         to: req.body.email,
162         from: 'shop@node-complete.com',
163         subject: 'Password reset',
164         html: `
165           <p>You requested a password reset</p>
166           <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
167           `
168       });
169     })
170     .catch(err => {
171       console.log(err);
172     });
173   });
174 };
175
176 exports.getNewPassword = (req, res, next) => {
177   /**
178    * i wanna check whether i find a user for data token
179    * which i receive here
180    * because this will be the page we load for this page
181    * where we do have a token in the URL
182    *
183    * we will need to add a route later
184    * that i code the token in a token field in our parameters
185    * i'm looking for token in my params
186    *
187    * and we wanna make sure that it's still valid from a date perspective
188    * so that the documents find does not just fulfil this criteria
189    * but another one to add more criteria with comma
190    * that the token expiration is still higher than the current date
191    * for data, i can use a special operator wrapped in curly braces.
192    * and '$gt' stands for a greater than
193    * and i can compare if it's greater than now for the current date.
194    */
195   const token = req.params.token;
196   User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } })
197     .then(user => {
198       let message = req.flash('error');

```

```

199     if (message.length > 0) {
200         message = message[0];
201     } else {
202         message = null;
203     }
204     res.render('auth/new-password', {
205         path: '/new-password',
206         pageTitle: 'New Password',
207         errorMessage: message,
208         /**we will pass my userId to that view
209         * so that i can include it in the POST request we will update the password
210         */
211         userId: user._id.toString()
212     });
213 }
214 .catch(err => {
215     console.log(err);
216 });
217 };
218

```

```

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 router.get('/reset', authController.getReset);
20
21 router.post('/reset', authController.postReset);
22
23 /**it has to be the name called 'token'
24 * because in that 'getNewPassword' in authController,
25 * i'm looking for token in my params
26 * so token here which i'm looking for in the request params means
27 * i have to name a token here as well
28 */
29 router.get('/reset/:token', authController.getNewPassword);
30
31 module.exports = router;

```

* Chapter 277: Adding Logic To Update The Password

1. update

- ./controllers/auth.js
- ./views/auth/new-password.ejs
- ./routes/auth.js

A screenshot of a web application's header bar with "Shop" and "Products" buttons on the left and "Login" and "Signup" buttons on the right. Below the header is a password update form. It has a "Password" label above a text input field containing four asterisks. Below the input is a green "Update Password" button.

A screenshot of a web application's header bar with "Shop" and "Products" buttons on the left and "Login" and "Signup" buttons on the right. Below the header is a login form. It has an "E-Mail" label above an empty text input field, a "Password" label above an empty text input field, a green "Login" button, and a "Reset Password" link below it.



My Cluster

Cluster0-shard-0 REPLICA SET 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						
<input type="button" value="FILTER"/>				<input type="button" value="OPTIONS"/>	<input type="button" value="FIND"/>	<input type="button" value="RESET"/>
<input type="button" value="INSERT DOCUMENT"/>	<input type="button" value="VIEW LIST"/>	<input type="button" value="TABLE"/>	Displaying documents 1 - 3 of 3 < > C			
<pre>_id: ObjectId("5badd5712917ac4c1c827f46") > cart: Object email: "test@test.com" password: "\$2a\$12\$pX2kME/tXuPTKQ025YjxuzuPay.FqIWePVPS2dX32pMR4c2Cje3G" __v: 1</pre> <pre>_id: ObjectId("5bade42501bea653ba3a74e5") > cart: Object email: "test@test.com" password: "\$2a\$12\$SMJbe9a.ispdevprNmicw.Anp5/Nv/EVsrdFsUR7j80v3C2hRb0vh" __v: 0</pre> <pre>_id: ObjectId("5bade9fd407f995e3d1ba12") > cart: Object email: password: "\$2a\$12\$H1fH3TYL1g3dQKKd3rPb0T3L1vCp.Xs319/JVdgk.3B7U9RPxYw" __v: 0 resetToken: "012b2414a4dd98a1efcb354165f14fcbb0416ffebcd589125" resetTokenExpiration: 2018-09-28 12:13:14.178</pre>						

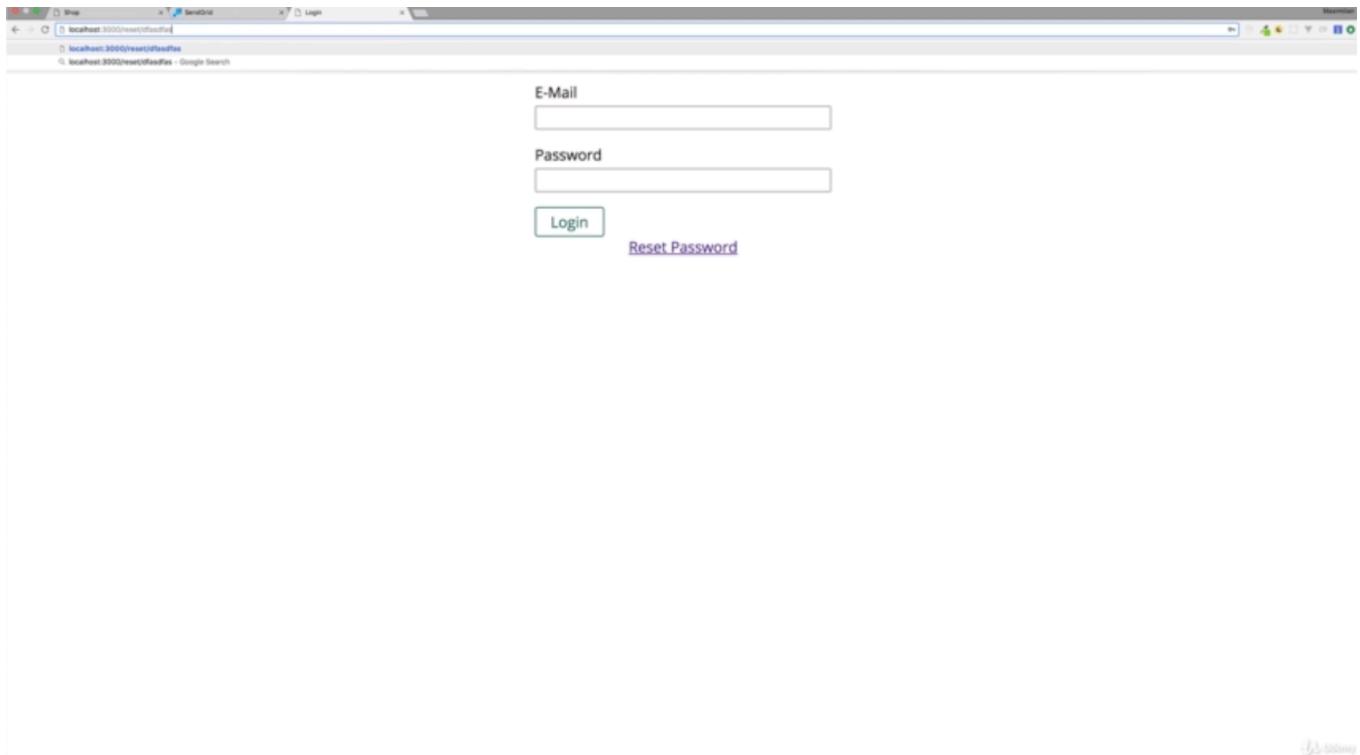
My Cluster

Cluster0-shard-0 REPLICA SET 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						
<input type="button" value="FILTER"/>				<input type="button" value="OPTIONS"/>	<input type="button" value="FIND"/>	<input type="button" value="RESET"/>
<input type="button" value="INSERT DOCUMENT"/>	<input type="button" value="VIEW LIST"/>	<input type="button" value="TABLE"/>	Displaying documents 1 - 3 of 3 < > C			
<pre>_id: ObjectId("5badd5712917ac4c1c827f46") > cart: Object email: "test@test.com" password: "\$2a\$12\$pX2kME/tXuPTKQ025YjxuzuPay.FqIWePVPS2dX32pMR4c2Cje3G" __v: 1</pre> <pre>_id: ObjectId("5bade42501bea653ba3a74e5") > cart: Object email: "test@test.com" password: "\$2a\$12\$SMJbe9a.ispdevprNmicw.Anp5/Nv/EVsrdFsUR7j80v3C2hRb0vh" __v: 0</pre> <pre>_id: ObjectId("5bade9fd407f995e3d1ba12") > cart: Object email: password: "\$2a\$12\$H1fH3TYL1g3dQKKd3rPb0T3L1vCp.Xs319/JVdgk.3B7U9RPxYw" __v: 0 resetToken: "012b2414a4dd98a1efcb354165f14fcbb0416ffebcd589125" resetTokenExpiration: 2018-09-28 12:13:14.178</pre>						

- if i update password, then resetToken is changed and past resetToken and resetTokenExpiration is gone.



EXPLORER auth.js controllers user.js auth.js routes login.ejs reset.ejs new-password.ejs

```

NODEJS-COMPLETE-GUIDE
├── .vscode
├── controllers
│   ├── admin.js
│   ├── auth.js
│   ├── error.js
│   └── shop.js
├── data
├── middleware
│   └── is-auth.js
└── models
    ├── order.js
    ├── product.js
    └── user.js

```

```

auth.js
207     })
208     .then(user => {
209         resetUser = user;
210         return bcrypt.hash(newPassword, 12);
211     })
212     .then(hashedPassword => {
213         resetUser.password = hashedPassword;
214         resetUser.resetToken = undefined;
215         resetUser.resetTokenExpiration = undefined;
216         return resetUser.save();
217     })
218     .then(result => {
219         res.redirect('/login');
220     })
221     .catch(err => {
222         console.log(err);
223     });
224 };
225 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

[nodemon] starting `node app.js`
(node:88714) 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.
TypeError: Cannot read property '_id' of null
    at User.findOne.then.user (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/controller
s/auth.js:188:22)

```

- if we try gonna reset with some random token then this will not work and we are not doing proper error handling there. because we essentially failed to find a user for that token. so this all are works. we can start changing values of random tokens of random users. so this will not work here.

```

1 //./controllers/auth.js
2
3 const crypto = require('crypto');
4
5 const bcrypt = require('bcryptjs');
6 const nodemailer = require('nodemailer');
7 const sendgridTransport = require('nodemailer-sendgrid-transport');
8
9 const User = require('../models/user');
10 
```

```
11 const transporter = nodemailer.createTransport(  
12   sendgridTransport({  
13     auth: {  
14       api_key:  
15         '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'  
16     }  
17   })  
18 );  
19  
20 exports.getLogin = (req, res, next) => {  
21   let message = req.flash('error');  
22   if (message.length > 0) {  
23     message = message[0];  
24   } else {  
25     message = null;  
26   }  
27   res.render('auth/login', {  
28     path: '/login',  
29     pageTitle: 'Login',  
30     errorMessage: message  
31   });  
32 };  
33  
34 exports.getSignup = (req, res, next) => {  
35   let message = req.flash('error');  
36   if (message.length > 0) {  
37     message = message[0];  
38   } else {  
39     message = null;  
40   }  
41   res.render('auth/signup', {  
42     path: '/signup',  
43     pageTitle: 'Signup',  
44     errorMessage: message  
45   });  
46 };  
47  
48 exports.postLogin = (req, res, next) => {  
49   const email = req.body.email;  
50   const password = req.body.password;  
51   User.findOne({ email: email })  
52     .then(user => {  
53       if (!user) {  
54         req.flash('error', 'Invalid email or password.');//  
55         return res.redirect('/login');//  
56       }  
57       bcrypt  
58         .compare(password, user.password)  
59         .then(doMatch => {  
60           if (doMatch) {  
61             req.session.isLoggedIn = true;  
62             req.session.user = user;  
63             return req.session.save(err => {  
64               if (err) {  
65                 console.log(err);  
66                 res.redirect('/');//  
67               }  
68             });  
69           }  
70         });  
71       });  
72     });  
73   });  
74   res.redirect('/');//  
75 };
```

```
67     }
68     .then(() => {
69       req.flash('error', 'Invalid email or password.');
70       res.redirect('/login');
71     })
72     .catch(err => {
73       console.log(err);
74       res.redirect('/login');
75     });
76   })
77   .catch(err => console.log(err));
78 }
79
80 exports.postSignup = (req, res, next) => {
81   const email = req.body.email;
82   const password = req.body.password;
83   const confirmPassword = req.body.confirmPassword;
84   User.findOne({ email: email })
85     .then(userDoc => {
86       if (userDoc) {
87         req.flash(
88           'error',
89           'E-Mail exists already, please pick a different one.'
90         );
91         return res.redirect('/signup');
92       }
93       return bcrypt
94         .hash(password, 12)
95         .then(hashedPassword => {
96           const user = new User({
97             email: email,
98             password: hashedPassword,
99             cart: { items: [] }
100           });
101           return user.save();
102         })
103         .then(result => {
104           res.redirect('/login');
105           return transporter.sendMail({
106             to: email,
107             from: 'shop@node-complete.com',
108             subject: 'Signup succeeded!',
109             html: '<h1>You successfully signed up!</h1>'
110           });
111         })
112         .catch(err => {
113           console.log(err);
114         });
115       .catch(err => {
116         console.log(err);
117       });
118   };
119
120 exports.postLogout = (req, res, next) => {
121   req.session.destroy(err => {
122     console.log(err);
123   });
124 }
```

```
123     res.redirect('/');
124   });
125 };
126
127 exports.getReset = (req, res, next) => {
128   let message = req.flash('error');
129   if (message.length > 0) {
130     message = message[0];
131   } else {
132     message = null;
133   }
134   res.render('auth/reset', {
135     path: '/reset',
136     pageTitle: 'Reset Password',
137     errorMessage: message
138   });
139 };
140
141 exports.postReset = (req, res, next) => {
142   crypto.randomBytes(32, (err, buffer) => {
143     if (err) {
144       console.log(err);
145       return res.redirect('/reset');
146     }
147     const token = buffer.toString('hex');
148     User.findOne({ email: req.body.email })
149       .then(user => {
150         if (!user) {
151           req.flash('error', 'No account with that email found.');
152           return res.redirect('/reset');
153         }
154         user.resetToken = token;
155         user.resetTokenExpiration = Date.now() + 3600000;
156         return user.save();
157       })
158       .then(result => {
159         res.redirect('/');
160         transporter.sendMail({
161           to: req.body.email,
162           from: 'shop@node-complete.com',
163           subject: 'Password reset',
164           html: `
165             <p>You requested a password reset</p>
166             <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
167             `
168         });
169       })
170       .catch(err => {
171         console.log(err);
172       });
173   });
174 };
175
176 exports.getNewPassword = (req, res, next) => {
177   const token = req.params.token;
```

```

178 User.findOne({
179   resetToken: token,
180   resetTokenExpiration: { $gt: Date.now() }
181 })
182   .then(user => {
183     let message = req.flash('error');
184     if (message.length > 0) {
185       message = message[0];
186     } else {
187       message = null;
188     }
189     res.render('auth/new-password', {
190       path: '/new-password',
191       pageTitle: 'New Password',
192       errorMessage: message,
193       userId: user._id.toString(),
194       passwordToken: token
195     });
196   })
197   .catch(err => {
198     console.log(err);
199   });
200 };
201
202 exports.postNewPassword = (req, res, next) => {
203   /**
204    * because otherwise people could start entering random tokens
205    * and still reach that page and then maybe change users on the back
206    * and by entering random userIds and that hidden input field as well.
207    * so i wanna have that token.
208   */
209   const newPassword = req.body.password
210   const userId = req.body.userId
211   const passwordToken = req.body.passwordToken
212   let resetUser
213
214   User.findOne({
215     resetToken: passwordToken,
216     resetTokenExpiration: {$gt: Date.now()},
217     _id: userId
218   })
219     .then(user => {
220       resetUser = user
221       return bcrypt.hash(newPassword, 12)
222     })
223     .then(hashedPassword => {
224       resetUser.password = hashedPassword
225       /**
226        * so these fields are not required anymore
227        * they don't need store any values anymore
228       */
229       resetUser.resetToken = undefined
230       resetUser.resetTokenExpiration = undefined
231       return resetUser.save()
232     })
233     .then(result => {
234       /**
235        * once you did 'save()',
236       */
237     })

```

```

234     * i can redirect the user back to the log-in page with that new password
235     */
236     res.redirect('/login')
237   })
238   .catch(err => {
239     console.log(err)
240   })
241
242 }
243

```

```

1 <!--./views/auth/new-password.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="/new-password" method="POST">
16      <div class="form-control">
17        <label for="password">Password</label>
18        <input type="password" name="password" id="password">
19      </div>
20      <input type="hidden" name="userId" value="<%= userId %>">
21      <input type="hidden" name="passwordToken" value="<%= passwordToken %>">
22      <input type="hidden" name="_csrf" value="<%= csrfToken %>">
23      <button class="btn" type="submit">Update Password</button>
24    </form>
25  </main>
26 <%-- include('../includes/end.ejs') %>

```

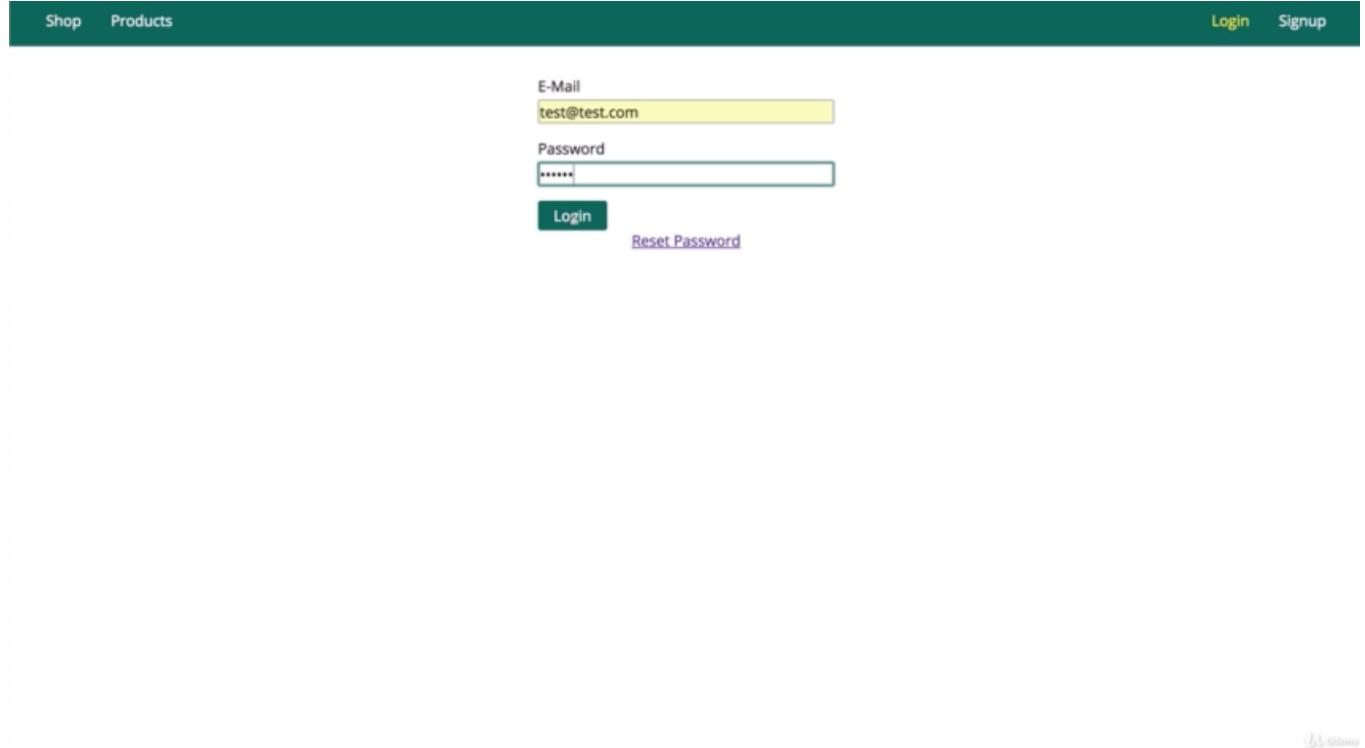
```

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 router.get('/reset', authController.getReset);

```

```
20
21 router.post('/reset', authController.postReset);
22
23 router.get('/reset/:token', authController.getNewPassword);
24
25 router.post('/new-password', authController.postNewPassword);
26
27 module.exports = router;
```

* Chapter 278: Why We Need Authorization



Title

Image URL

Price

Description

Add Product

A Book



\$ 12.99

This was created by test@test.com

[Edit](#) [Delete](#)



- authorization means that i restrict permissions of a logged in user. so every user might be able to add anything to the car including products created by the user but you might not be able to delete an edit product which were not created by you.

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases (My Cluster, 3 DBs) and collections (11 Collections). Under 'shop' are 'products', 'sessions', and 'users'. The main area shows the 'shop.products' collection with 1 document. The document details are:

```

_id: ObjectId("5badf52f45448e5a8bea9d4f")
title: "A Book"
price: 12.99
description: "This was created by test@test.com"
imageUrl: "http://ichef.bbci.co.uk/wfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bade42501bea653ba3a74e5")
__v: 0

```

This screenshot is identical to the one above, showing the same document in the 'shop.products' collection. The document details are the same:

```

_id: ObjectId("5badf52f45448e5a8bea9d4f")
title: "A Book"
price: 12.99
description: "This was created by test@test.com"
imageUrl: "http://ichef.bbci.co.uk/wfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bade42501bea653ba3a74e5")
__v: 0

```

- when I create this product here, we have it here and we can see it in the product collection obviously. It's linked to the user with an ID which ends with 'e5'

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.8 Enterprise

3 DBS 11 COLLECTIONS

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				FILTER			
Aggregations				OPTIONS			
Explain Plan				FIND			
Indexes				RESET			
INSERT DOCUMENT	VIEW LIST TABLE	Displaying documents 1 - 2 of 2					
<pre>_id: ObjectId("5badd5712917ac4c1c827f46") > cart: Object email: "test2@test.com" password: "\$2a\$12\$p8X2kME/tXuPTkN025YjxuzuPay.FqDMePVPS2dX32pMR4c2Cje3G" __v: 1</pre>							
<pre>_id: ObjectId("5bade42501bea653ba3a74e5") > cart: Object email: "test@test.com" password: "\$2a\$12\$SHJbe9a.ispdevprNmicw.Anp5/Nv/EVsrdFsUR7j80v3C2hRb0vM" __v: 0</pre>							

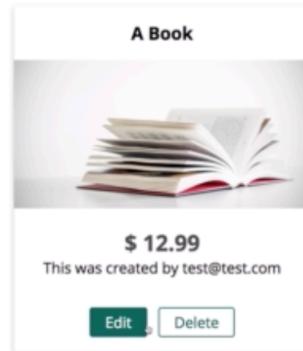
- and this is the 'e5' and this is 'test@test.com'

Shop Products Login Signup

E-Mail

Password

Login [Reset Password](#)



[https://mern-tutorial-production-01.netlify.app/admin/products](#)

by Simey

- now we login 'test2@test.com' then if i go to Admin Products, i can still edit and delete it. this is what we wanna prevent from which means we wanna let only me edit or delete and don't let others edit or delete it.

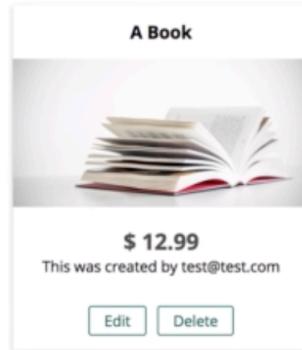
* Chapter 279: Adding Authorization

1. update

- ./controllers/admin.js

Fields	Values
_id	ObjectId("5badf52f45448e5a8bea9d4f")
title	"A Book"
price	12.99
description	"This was created by test@test.com"
imageUrl	"http://ichef.bbci.co.uk/wfeatures/wm/live/1288_640/images/live/p0/2v/..."
userId	ObjectId("5baade42501bea653ba3a74e5")
__v	0

- we know which user did create it. so in the end, we wanna check if the currently logged in user is the user who created that before allowing any edits to that item.



Logout

No Products Found!

Logout

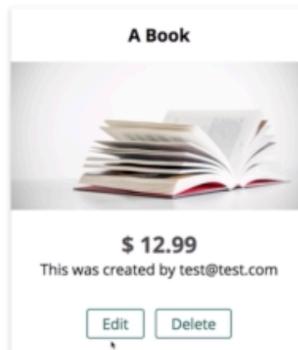
Logout

- with that change, if i reload this page, i find no products for test2@test.com.

E-Mail

Password

[Reset Password](#)



- but i login with my other account with test@test.com, i go to Admin Products, i find that book.

```
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,
20   price,
21   description,
22   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   /**authorization means we restrict the permissions
82    * and we can do that by restricting the data we return.
83    * so here when i find product,
84    * i don't find all
85    * but i will add a filter
86    * and i will filter for products where the the userId is equal to userId of the currently
87    * logged in user,
88    * so userId is equal to req.user._id
89    */
90   Product.find({userId: req.user._id})
91     // .select('title price _id')
92     // .populate('userId', 'name')
93     .then(products => {
94       console.log(products);
95       res.render('admin/products', {
96         prods: products,
97         pageTitle: 'Admin Products',
98         path: '/admin/products'
99       });
100      })
101      .catch(err => console.log(err));
102    };
103
104  exports.postDeleteProduct = (req, res, next) => {
105    const prodId = req.body.productId;
106    Product.findByIdAndRemove(prodId)
107      .then(() => {
108        console.log('DESTROYED PRODUCT');
109        res.redirect('/admin/products');
110      })
111      .catch(err => console.log(err));
112  };

```

* Chapter 280: Adding Protection To Post Actions

1. update
- ./controllers/admin.js

E-Mail
test2@test.com

Password
.....

[Login](#) [Reset Password](#)



A Book



\$ 12.99
This was created by test@test.com

[Edit](#) [Delete](#)



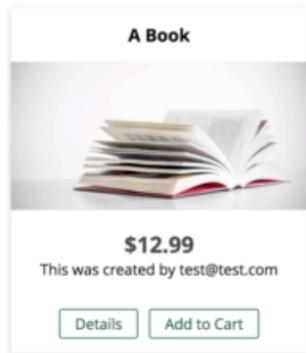
Title

Image URL

Price

Description

[Update Product](#)



- i'm redirected but the edit wasn't saved

The screenshot shows a code editor in VS Code with several tabs open: auth.js controllers, admin.js, app.js, user.js, auth.js routes, login.ejs, and others. The admin.js file contains a function for updating a product. A cursor is at line 67, which contains a check for the user's ID. If it doesn't match, the user is redirected to the root path ('/'). The code then updates the product object with new values from req.body. After the update, it calls product.save() and logs 'UPDATED PRODUCT!' to the console. Finally, it redirects the user back to the admin products page. The terminal shows the product data being passed to the save method.

```
52     })
53     .catch(err => console.log(err));
54   };
55 
56 exports.postEditProduct = (req, res, next) => {
57   const prodId = req.body.productId;
58   const updatedTitle = req.body.title;
59   const updatedPrice = req.body.price;
60   const updatedImageUrl = req.body.imageUrl;
61   const updatedDesc = req.body.description;
62 
63   Product.findById(prodId)
64     .then(product => {
65       if (product.userId !== req.user._id) {
66         return res.redirect('/');
67       }
68       product.title = updatedTitle;
69       product.price = updatedPrice;
70       product.description = updatedDesc;
71       product.imageUrl = updatedImageUrl;
72     })
73     .then(result => {
74       console.log('UPDATED PRODUCT!');
75       res.redirect('/admin/products');
76     })
77     .catch(err => console.log(err));
78   };
79 }

price: 12.99,
description: 'This was created by test@test.com',
imageUrl:
  'http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',
userId: 5bade42501bea653ba3a74e5,
__v: 0 } ]
```

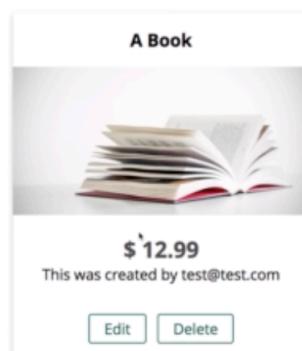
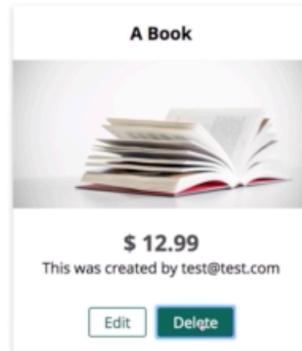
because i essentially ended up in my extra check here and i was redirected back but without saving the changes

This screenshot is similar to the previous one, but the code has been modified. The cursor is now at line 72, where the product is saved. The log message 'UPDATED PRODUCT!' is visible in the terminal, indicating that the update was successful. The rest of the code and terminal output are identical to the previous screenshot.

```
55 
56 exports.postEditProduct = (req, res, next) => {
57   const prodId = req.body.productId;
58   const updatedTitle = req.body.title;
59   const updatedPrice = req.body.price;
60   const updatedImageUrl = req.body.imageUrl;
61   const updatedDesc = req.body.description;
62 
63   Product.findById(prodId)
64     .then(product => {
65       if (product.userId !== req.user._id) {
66         return res.redirect('/');
67       }
68       product.title = updatedTitle;
69       product.price = updatedPrice;
70       product.description = updatedDesc;
71       product.imageUrl = updatedImageUrl;
72       return product.save().then(result => {
73         console.log('UPDATED PRODUCT!');
74         res.redirect('/admin/products');
75       })
76     })
77     .catch(err => console.log(err));
78   };
79 }

price: 12.99,
description: 'This was created by test@test.com',
imageUrl:
  'http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/dp/p02vdpfn.jpg',
userId: 5bade42501bea653ba3a74e5,
__v: 0 } ]
```

because this code was never executed.



- i go to Admin Products with the wrong account still and i click delete, the product doesn't go anywhere. still there.

E-Mail

Password

[Reset Password](#)



A Book



\$ 12.99
This was created by test@test.com



Title

Price

Description

- if i login with valid 'test@test.com' who add product, it should be able to edit it.

```

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       /**we wanna check that the product i try to delete is really created by the user
68       * who is currently logged in user */
69       if(product.userId !== req.user._id){
70         return res.redirect('/')
71       }
72       product.title = updatedTitle;
73       product.price = updatedPrice;
74       product.description = updatedDesc;
75       product.imageUrl = updatedImageUrl;
76       return product.save()
77         .then(result => {
78           console.log('UPDATED PRODUCT!');
79           res.redirect('/admin/products');
80         })
81       }
82     )
83     .catch(err => console.log(err));
84 };
85
86 exports.getProducts = (req, res, next) => {
87   Product.find({userId: req.user._id})
88     // .select('title price _id')
89     // .populate('userId', 'name')
90     .then(products => {
91       console.log(products);

```

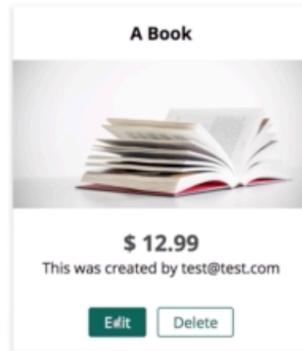
```

91     res.render('admin/products', {
92       prods: products,
93       pageTitle: 'Admin Products',
94       path: '/admin/products'
95     });
96   })
97   .catch(err => console.log(err));
98 };
99
100 /**
101 * i wanna check if i'm really allowed to delete that product
102 * and we can implement this by changing our deletion method
103 * and use 'deleteOne()'
104 */
105 exports.postDeleteProduct = (req, res, next) => {
106   const prodId = req.body.productId;
107   /**
108    * both has to be true now.
109    * prodId is not enough,
110    * req.user._id also has to be matched
111    *
112    * even if a valid userId is there,
113    * it will not delete this product if the userId doesn't match
114   */
115   Product.deleteOne({_id: prodId, userId: req.user._id})
116     .then(() => {
117       console.log('DESTROYED PRODUCT');
118       res.redirect('/admin/products');
119     })
120     .catch(err => console.log(err));

```

* Chapter 281: Why Editing Fails?

1. update
- ./controllers/admin.js

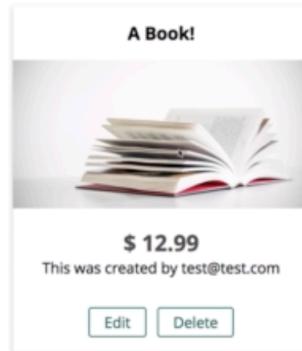


Title

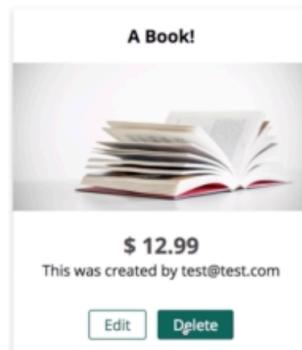
Image URL

Price

Description



- now i can update this



No Products Found!



- i also can delete this.

Title

Image URL

Price

Description



A Book



\$ 22
This works!

[Edit](#) [Delete](#)

A Book



\$ 22
This works!

[Edit](#) [Delete](#)

E-Mail

Password

[Reset Password](#)

No Products Found!

- but it not work with the wrong account because if i log in there, i don't see it in the Admin Products. even if i would see it, i could not interact with it.

```
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       /**i should convert both to a string
68        * because i'm also checking for type equality.
```

```

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

```

* Chapter 282: Wrap Up

Module Summary

Password Resetting	Authorization
<ul style="list-style-type: none">• Password resetting has to be implemented in a way that prevents users from resetting random user accounts• Reset tokens have to be random, unguessable and unique	<ul style="list-style-type: none">• Authorization is an important part of pretty much every app• Not every authenticated user should be able to do everything• Instead, you want to lock down access by restricting the permissions of your users