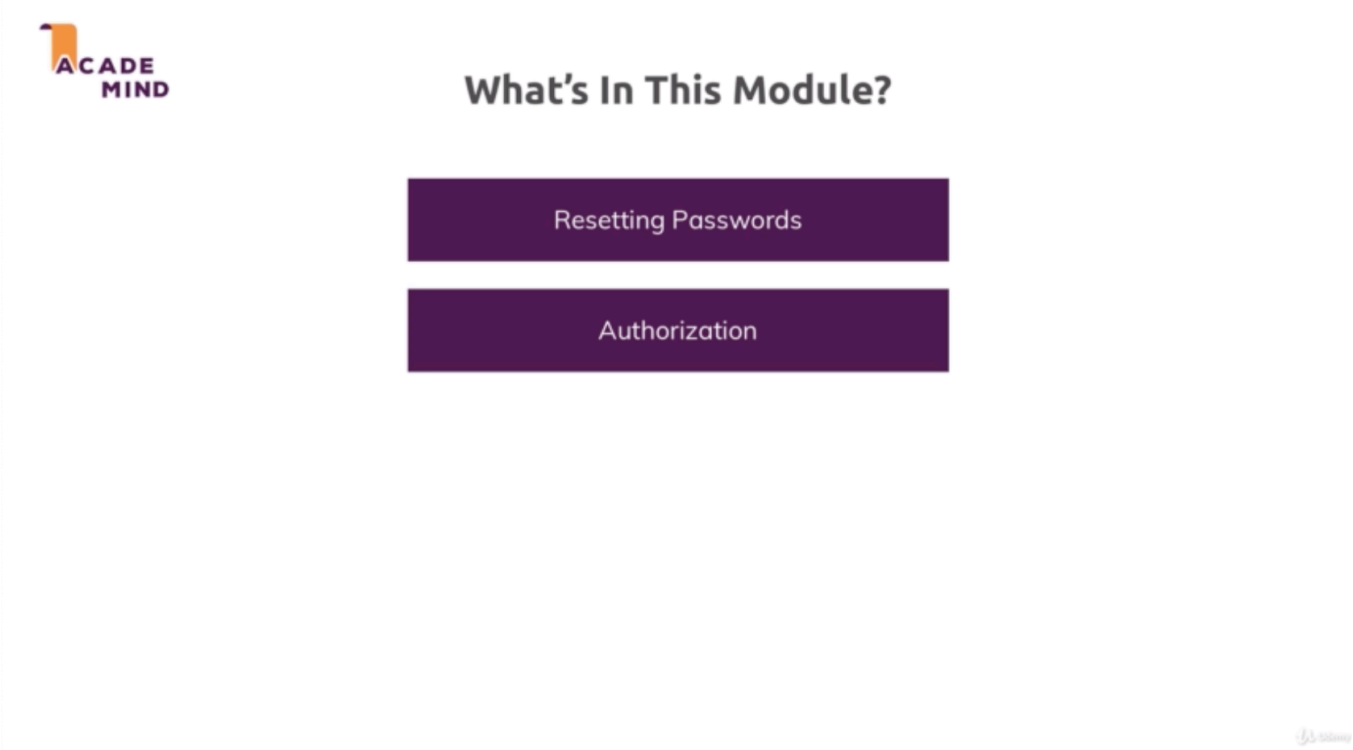
17. Advanced Authentication

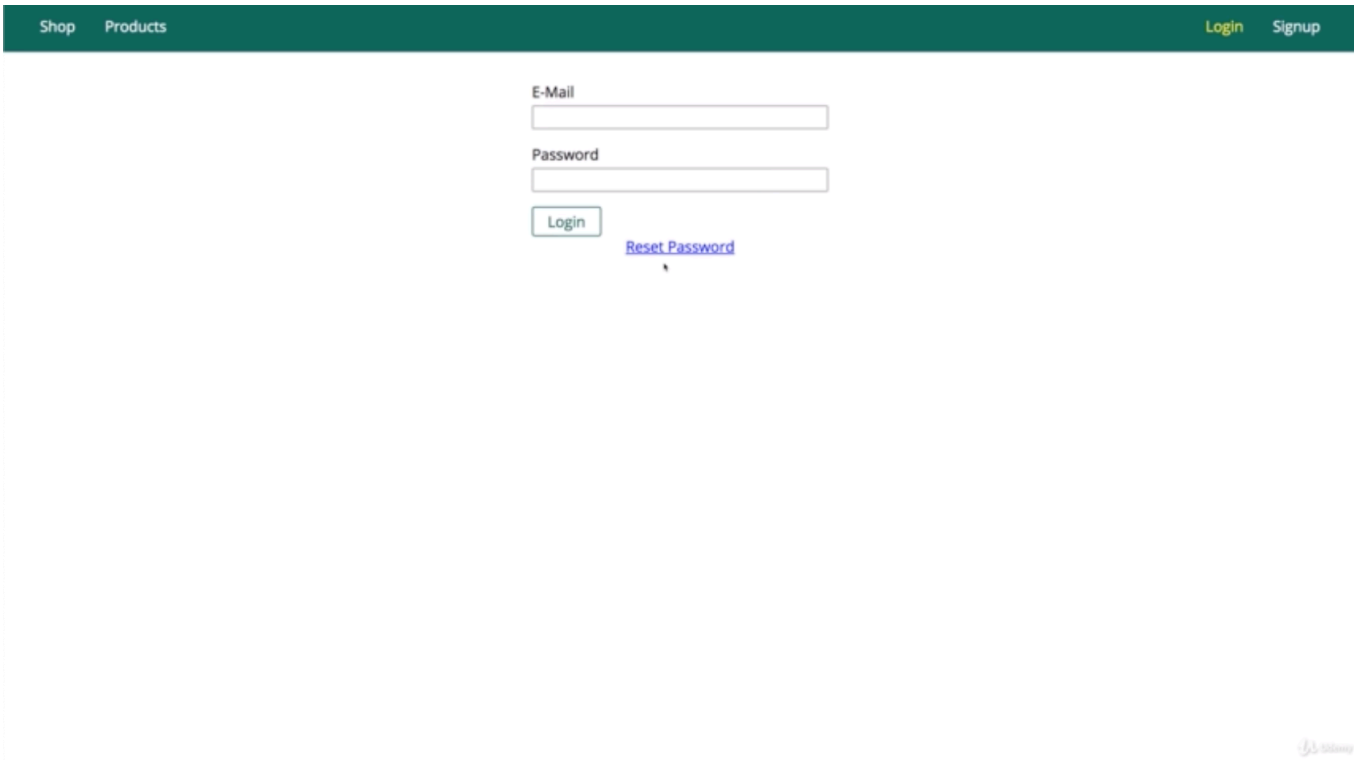
* Chapter 272: Module Introduction

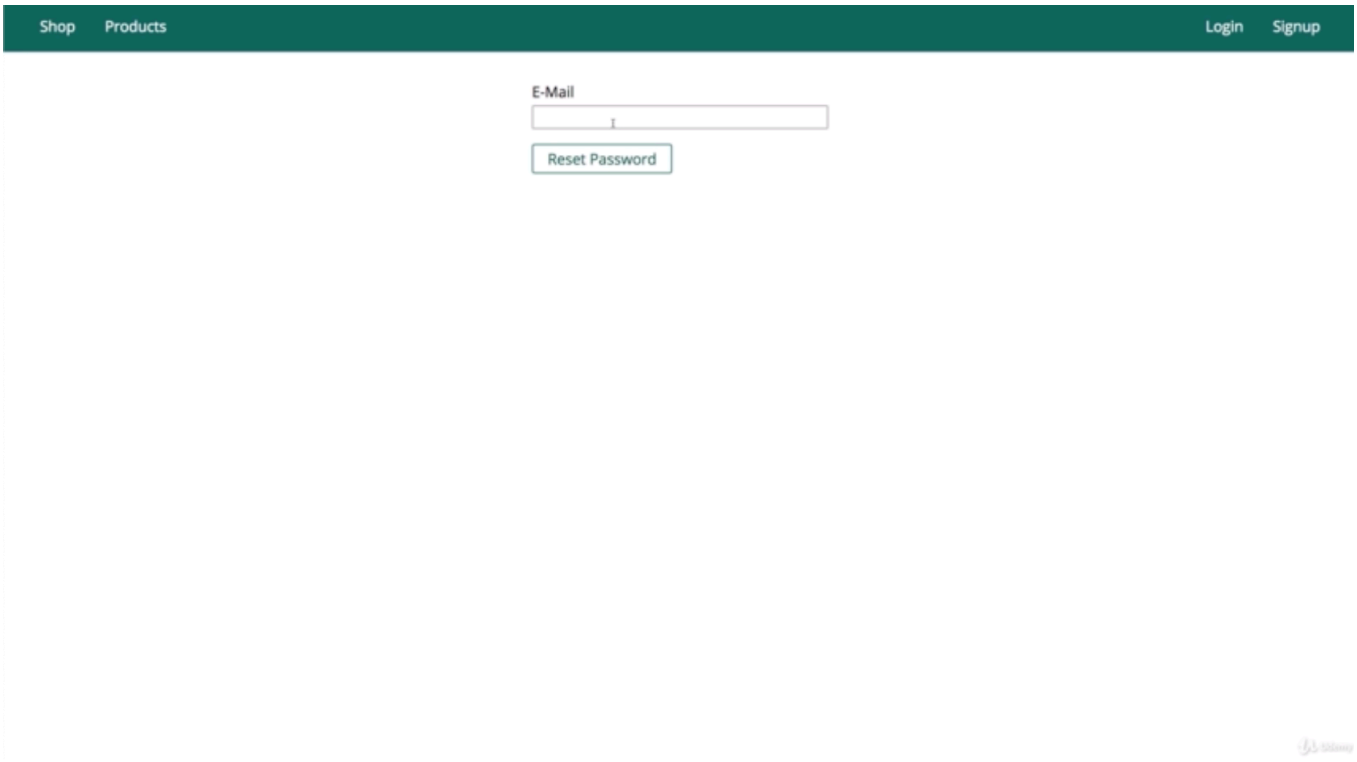


* Chapter 273: Resetting Passwords

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.






```
<!--./views/auth/reset.ejs-->

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

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

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
```

```
        <form class="login-form" action="/login" method="POST">
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input type="email" name="email" id="email">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Reset Password</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

```
//./controllers/auth.js

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer')
const sendgridTransport = require('nodemailer-sendgrid-transport')

const User = require('../models/user');

const transporter = nodemailer.createTransport(sendgridTransport({
  auth: {
    api_key: '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'
  }
}))

exports.getLogin = (req, res, next) => {
  let message = req.flash('error')
  if(message.length > 0){
    message = message[0]
  } else {
    message = null
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error')
  if(message.length > 0){
    message = message[0]
  } else {
    message = null
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
```

```
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.')
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.')
          res.redirect('/login');
        })
        .catch(err => {
          console.log(err);
          res.redirect('/login');
        });
    })
    .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  const confirmPassword = req.body.confirmPassword;
  User.findOne({ email: email })
    .then(userDoc => {
      if (userDoc) {
        req.flash('error', 'E-Mail exists already, please pick a different one. ')
        return res.redirect('/signup');
      }
      return bcrypt
        .hash(password, 12)
        .then(hashedPassword => {
          const user = new User({
            email: email,
            password: hashedPassword,
            cart: { items: [] }
          });
          return user.save();
        })
        .then(result => {
          res.redirect('/login');
          return transporter.sendMail({
            to: email,
            from: 'shop@nodecomplete.com',
            subject: 'Signup Succeeded!',
            html: '<h1>You Successfully Signed Up!</h1>'
          })
        })
```

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

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error')
  if(message.length > 0){
    message = message[0]
  } else {
    message = null
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });

}
```

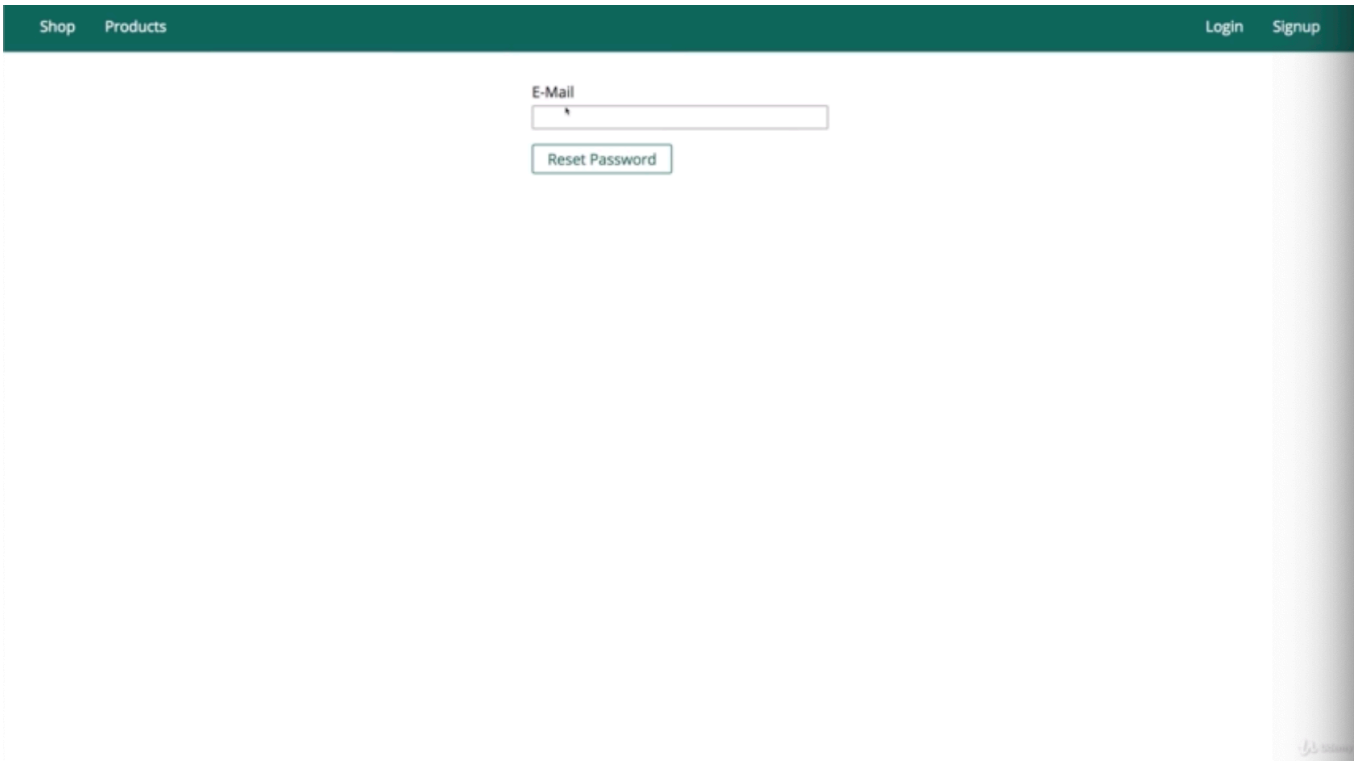
```
// ./routes/auth.js

const express = require('express');

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

router.post('/signup', authController.postSignup);

router.post('/logout', authController.postLogout);

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

module.exports = router;
```

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

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

```
    <link rel="stylesheet" href="/css/auth.css">
</head>

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

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/login" method="POST">
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input type="email" name="email" id="email">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password" id="password">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Login</button>
        </form>
        <div class="centered">
            <a href="/reset">Reset Password</a>
        </div>
    </main>
<%- include('../includes/end.ejs') %>
```

* Chapter 274: Implementing The Token Logic

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



- 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.

```
//./controllers/auth.js

/**node.js has a built-in 'crypto' library
 * which helps us with creating secure random values and other things
 * but we will need that unique secure random value here.
 */

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
        '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
```

```
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
        })
        .catch(err => {
          console.log(err);
          res.redirect('/login');
        });
    })
    .catch(err => console.log(err));
};

```

```
exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  const confirmPassword = req.body.confirmPassword;
  User.findOne({ email: email })
    .then(userDoc => {
      if (userDoc) {
        req.flash(
          'error',
          'E-Mail exists already, please pick a different one.'
        );
        return res.redirect('/signup');
      }
      return bcrypt
        .hash(password, 12)
        .then(hashedPassword => {
          const user = new User({
            email: email,
            password: hashedPassword,
            cart: { items: [] }
          });
          return user.save();
        })
        .then(result => {
          res.redirect('/login');
          return transporter.sendMail({
            to: email,
            from: 'shop@node-complete.com',
            subject: 'Signup succeeded!',
            html: '<h1>You successfully signed up!</h1>'
          });
        })
        .catch(err => {
          console.log(err);
        });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
```

```
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  /**2nd argument will be called once it's done.
   * and error and buffer of bytes.
  */
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    /**we need to pass hex
     * because that buffer will store hexadecimal values
     * this is information toString need to convert hexadecimal values to normal ASCII
Characters.
     *
     * that token should get stored in the database
     * and it should get stored on the user object
     * because it belongs to that user.
     *
     * so first of all, go to our ./models/user.js
     */

     /**we gonna fine one user
      * where the email matches the email we are trying to reset
      * and that email which we are trying to reset
      * can be extracted from the req.body.email field
      * because on our reset view here,
      * we do have that email field in ./views/auth/reset.ejs file
      *
      */
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          /**i'm placing '${token}' in the URL
           * because that token is then what we will later look for in the database
           * to confirm, this link was sent by us
           * because only we know the token.
```

```
        */
        html: `
          <p>You requested a password reset</p>
          <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>
        `
      });
    })
    .catch(err => {
      console.log(err);
    });
  });
};
```

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

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({
  email: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  /**this is not required
   * because not always this token will be there,
   * only if we requested a reset
   */
  resetToken: String,
  resetTokenExpiration: Date,
  cart: {
    items: [
      {
        productId: {
          type: Schema.Types.ObjectId,
          ref: 'Product',
          required: true
        },
        quantity: { type: Number, required: true }
      }
    ]
  }
});

userSchema.methods.addToCart = function(product) {
  const cartProductIndex = this.cart.items.findIndex(cp => {
    return cp.productId.toString() === product._id.toString();
  });
  let newQuantity = 1;
  const updatedCartItems = [...this.cart.items];

  if (cartProductIndex >= 0) {
```

```
    newQuantity = this.cart.items[cartProductIndex].quantity + 1;
    updatedCartItems[cartProductIndex].quantity = newQuantity;
  } else {
    updatedCartItems.push({
      productId: product._id,
      quantity: newQuantity
    });
  }
  const updatedCart = {
    items: updatedCartItems
  };
  this.cart = updatedCart;
  return this.save();
};

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

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

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

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

// const ObjectId = mongodb.ObjectId;

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

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

//   addToCart(product) {
//     const cartProductIndex = this.cart.items.findIndex(cp => {
//       return cp.productId.toString() === product._id.toString();
//     });
//     let newQuantity = 1;
//     const updatedCartItems = [...this.cart.items];

//     if (cartProductIndex >= 0) {
```

```
//       newQuantity = this.cart.items[cartProductIndex].quantity + 1;
//       updatedCartItems[cartProductIndex].quantity = newQuantity;
//     } else {
//       updatedCartItems.push({
//         productId: new ObjectId(product._id),
//         quantity: newQuantity
//       });
//     }
//     const updatedCart = {
//       items: updatedCartItems
//     };
//     const db = getDb();
//     return db
//       .collection('users')
//       .updateOne(
//         { _id: new ObjectId(this._id) },
//         { $set: { cart: updatedCart } }
//       );
//   }

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

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

//   addOrder() {
//     const db = getDb();
```

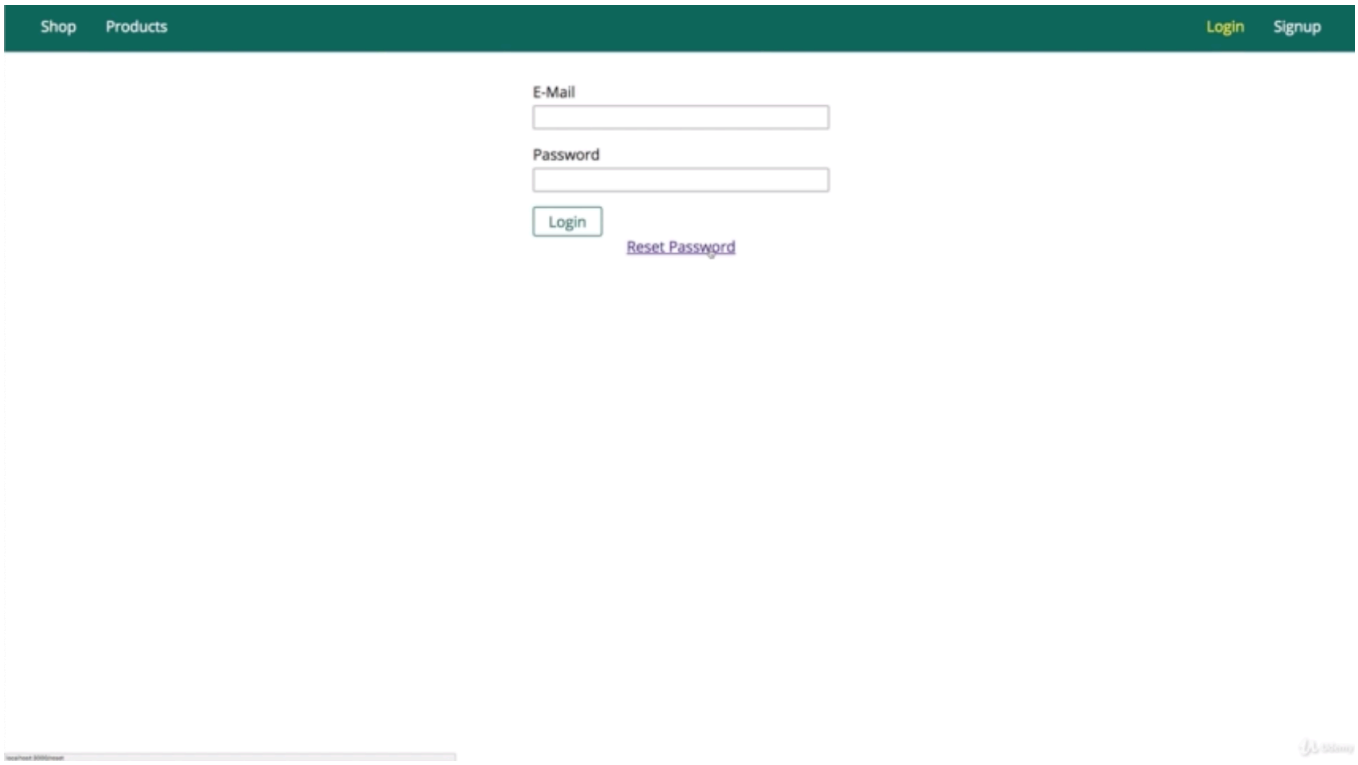
```
//     return this.getCart()
//       .then(products => {
//         const order = {
//           items: products,
//           user: {
//             _id: new ObjectId(this._id),
//             name: this.name
//           }
//         };
//         return db.collection('orders').insertOne(order);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

//   getOrders() {
//     const db = getDb();
//     return db
//       .collection('orders')
//       .find({ 'user._id': new ObjectId(this._id) })
//       .toArray();
//   }

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

// module.exports = User;

```

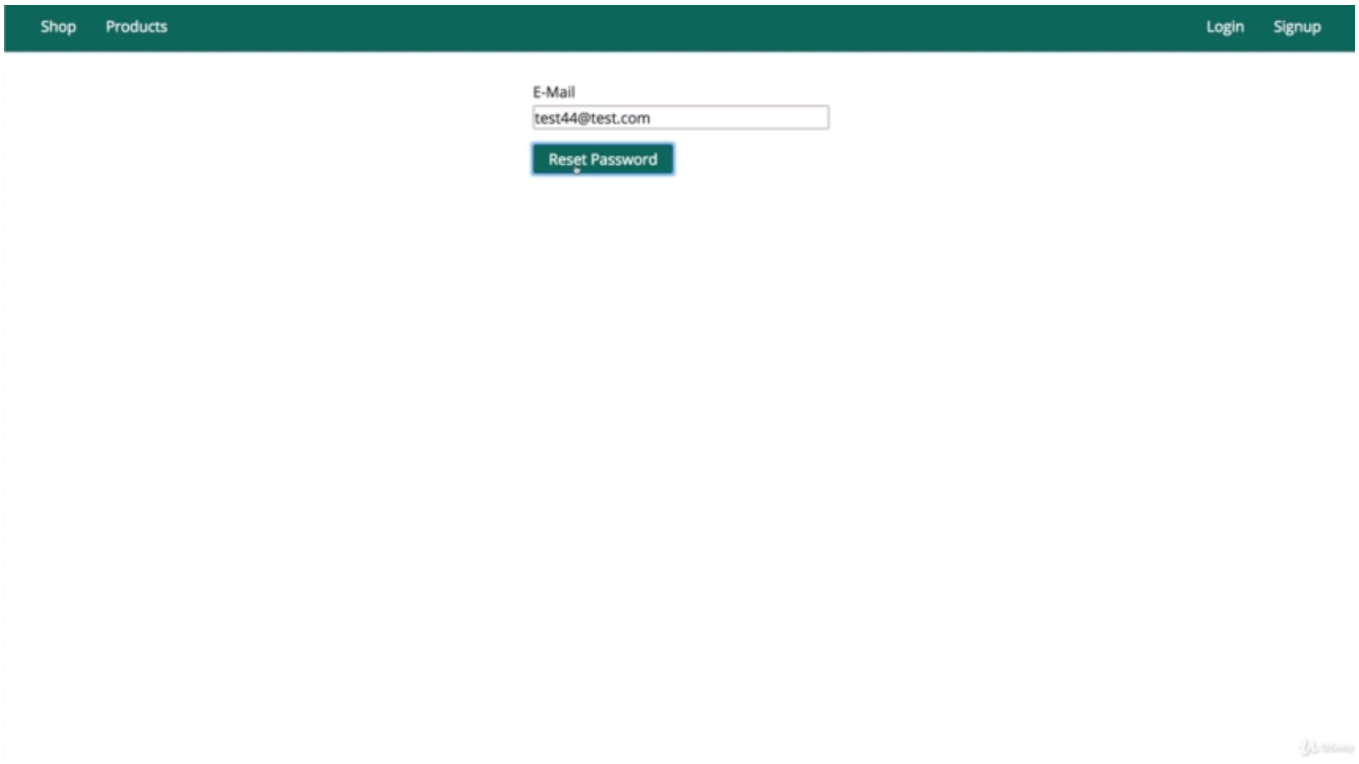
```
<!--./views/auth/reset.ejs-->

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

<body>
   <%- 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') %>
```

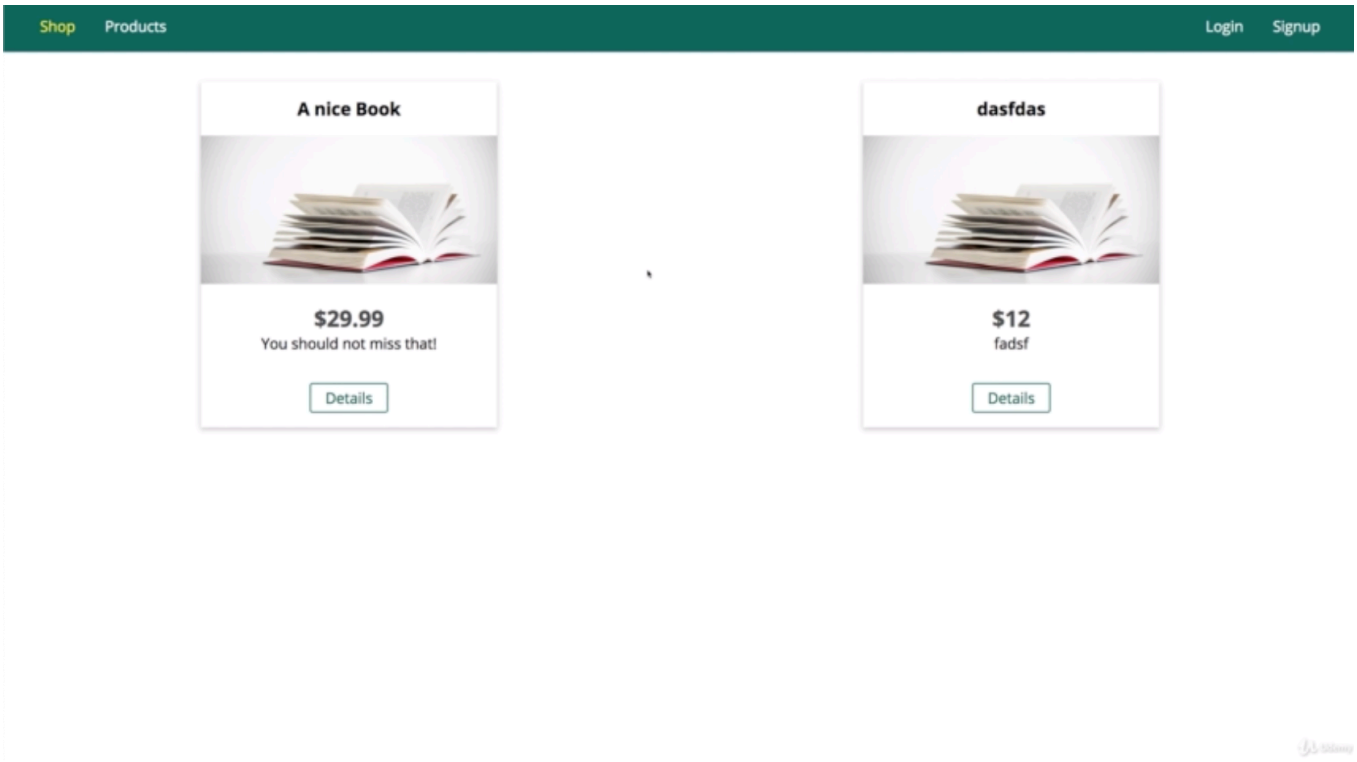
* Chapter 275: Creating The Token

1. update
- routes/auth.js

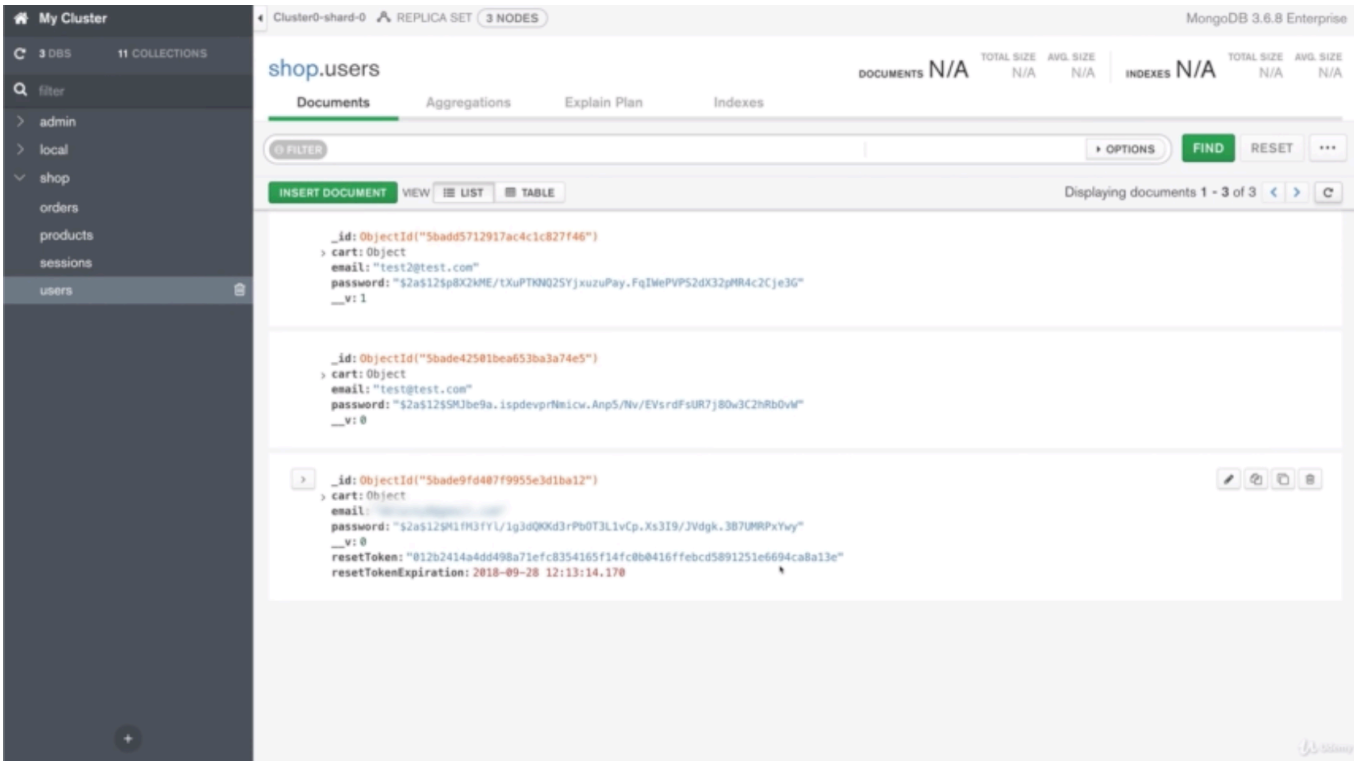




- click ‘Reset Password’ and try out the incorrect email which doesn’t exist in the database and we get this message which is great.



- 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.



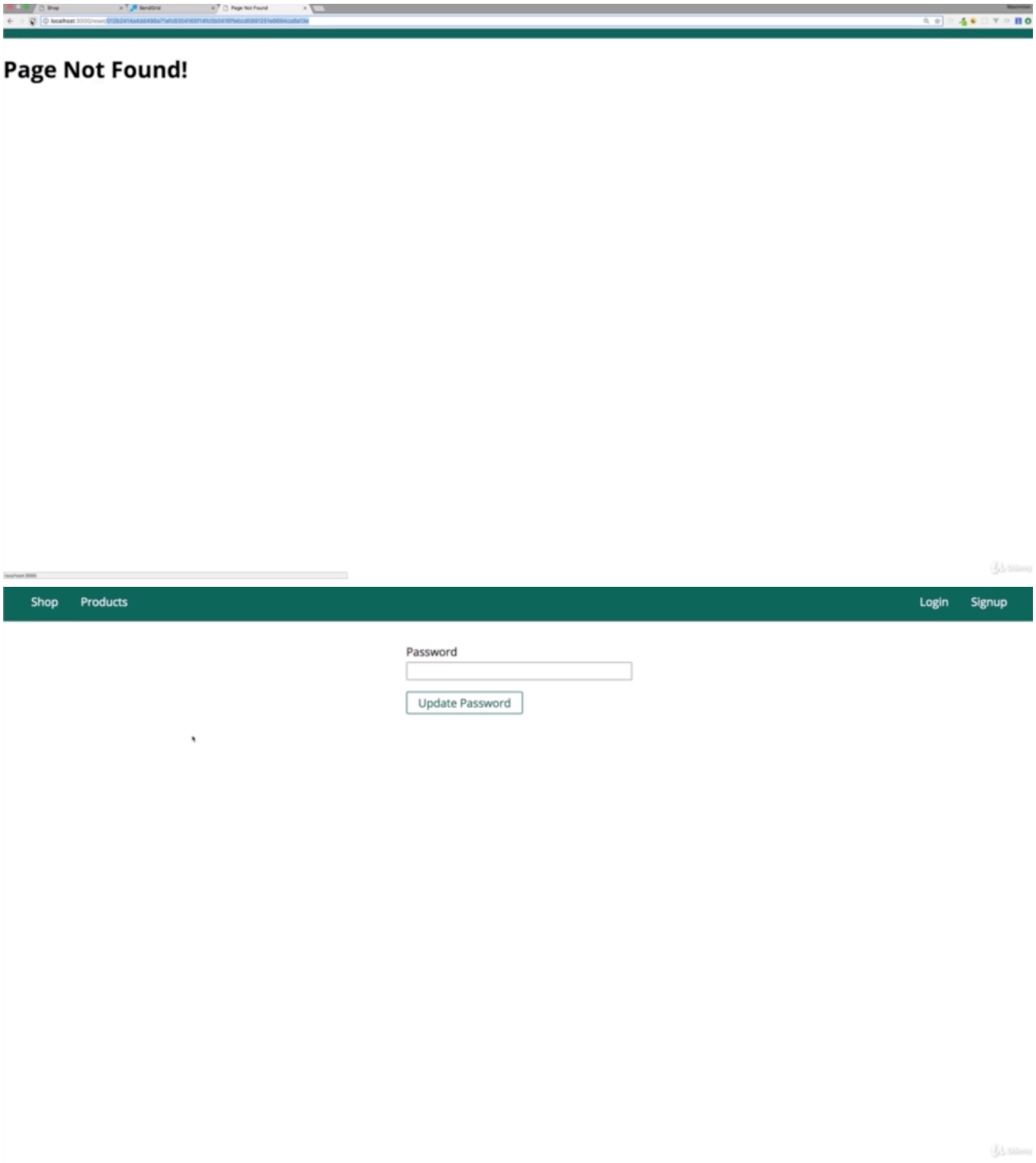
- if you click that link, you should end up on a 'page not found' error but you will see that were on reset and then some token and that token can also be seen in your user collection here. this is the password reset token which restored

- 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.

```
// ./routes/auth.js

const express = require('express');

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

router.post('/signup', authController.postSignup);

router.post('/logout', authController.postLogout);

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

router.post('/reset', authController.postReset)

module.exports = router;
```

* Chapter 276: Creating The Reset Password Form

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




```ejs
<!--./views/auth/new-password.ejs-->

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

```

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

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/new-password" method="POST">
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password" id="password">
            </div>
            <!--
                i'm passing into the view
                so does input is new with that hidden userId
                and it will output that userId
                that userId will need that for the POST request we wanna save our new
password.
            -->
            <input type="hidden" name="userId" value="<%= userId %>">
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Update Password</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
        '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
```

```
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
        })
        .catch(err => {
          console.log(err);
          res.redirect('/login');
        });
    })
    .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  const confirmPassword = req.body.confirmPassword;
  User.findOne({ email: email })
    .then(userDoc => {
      if (userDoc) {
        req.flash(
          'error',
```

```
        'E-Mail exists already, please pick a different one.'
      );
      return res.redirect('/signup');
    }
    return bcrypt
      .hash(password, 12)
      .then(hashedPassword => {
        const user = new User({
          email: email,
          password: hashedPassword,
          cart: { items: [] }
        });
        return user.save();
      })
      .then(result => {
        res.redirect('/login');
        return transporter.sendMail({
          to: email,
          from: 'shop@node-complete.com',
          subject: 'Signup succeeded!',
          html: '<h1>You successfully signed up!</h1>'
        });
      })
      .catch(err => {
        console.log(err);
      });
  })
  .catch(err => {
    console.log(err);
  });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
```

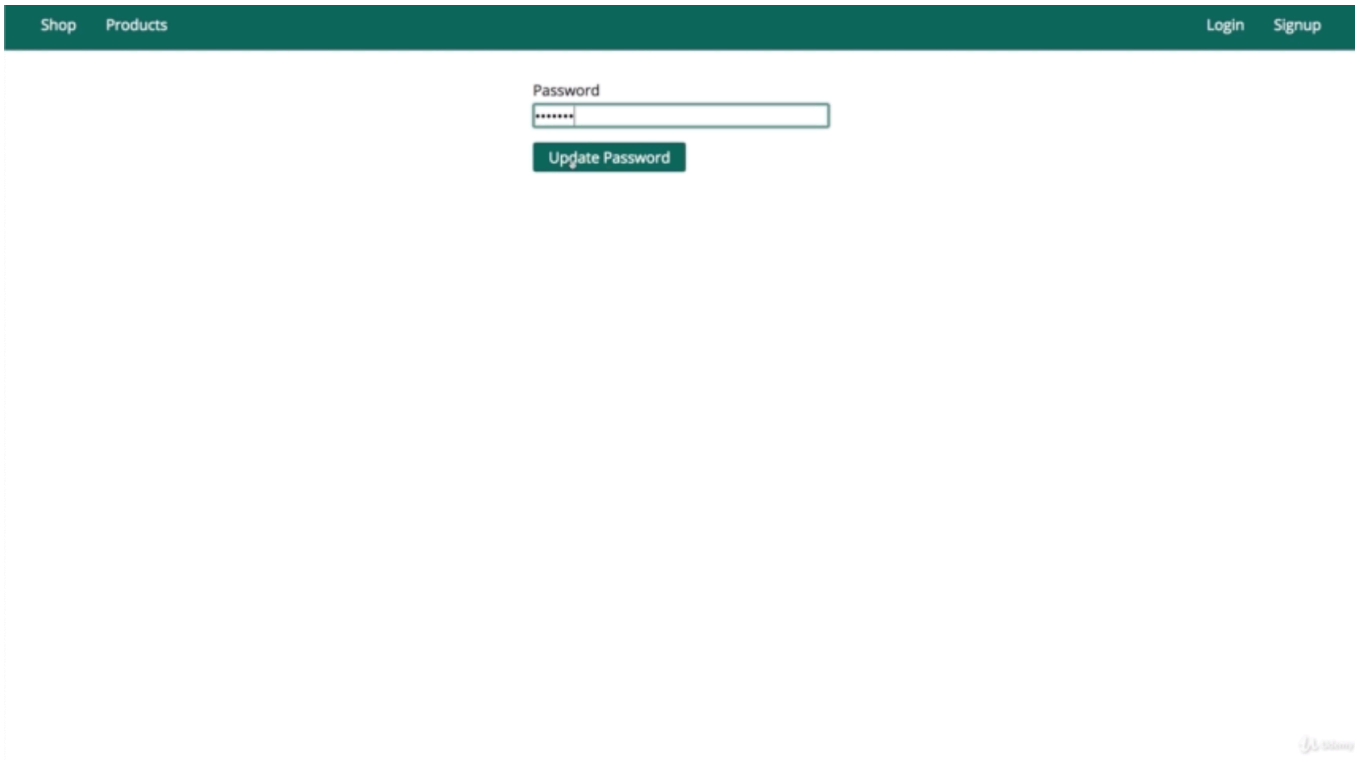
```
    console.log(err);
    return res.redirect('/reset');
   }
   const token = buffer.toString('hex');
   User.findOne({ email: req.body.email })
    .then(user => {
     if (!user) {
      req.flash('error', 'No account with that email found.');
      return res.redirect('/reset');
     }
     user.resetToken = token;
     user.resetTokenExpiration = Date.now() + 3600000;
     return user.save();
    })
    .then(result => {
     res.redirect('/');
     transporter.sendMail({
      to: req.body.email,
      from: 'shop@node-complete.com',
      subject: 'Password reset',
      html: `
       <p>You requested a password reset</p>
       <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
      `
     });
    })
    .catch(err => {
     console.log(err);
    });
  });
};

exports.getNewPassword = (req, res, next) => {
  /**i wanna check whether i find a user for data token
   * which i receive here
   * because this will be the page we load for this page
   * where we do have a token in the URL
   *
   * we will need to add a route later
   * that i code the token in a token field in our parameters
   * i'm looking for token in my params
   *
   *
   * and we wanna make sure that it's still valid from a date perspective
   * so that the documents find does not jus fulfil this criteria
   * but another one to add more criteria with comma
   * that the token expiration is still higher that the current date
   * for data, i can use a special operator wrapped in curly braces.
   * and '$gt' stands for a greater than
   * and i can compare if it's greater than now for the current date.
   */
  const token = req.params.token;
  User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } })
    .then(user => {
     let message = req.flash('error');
```

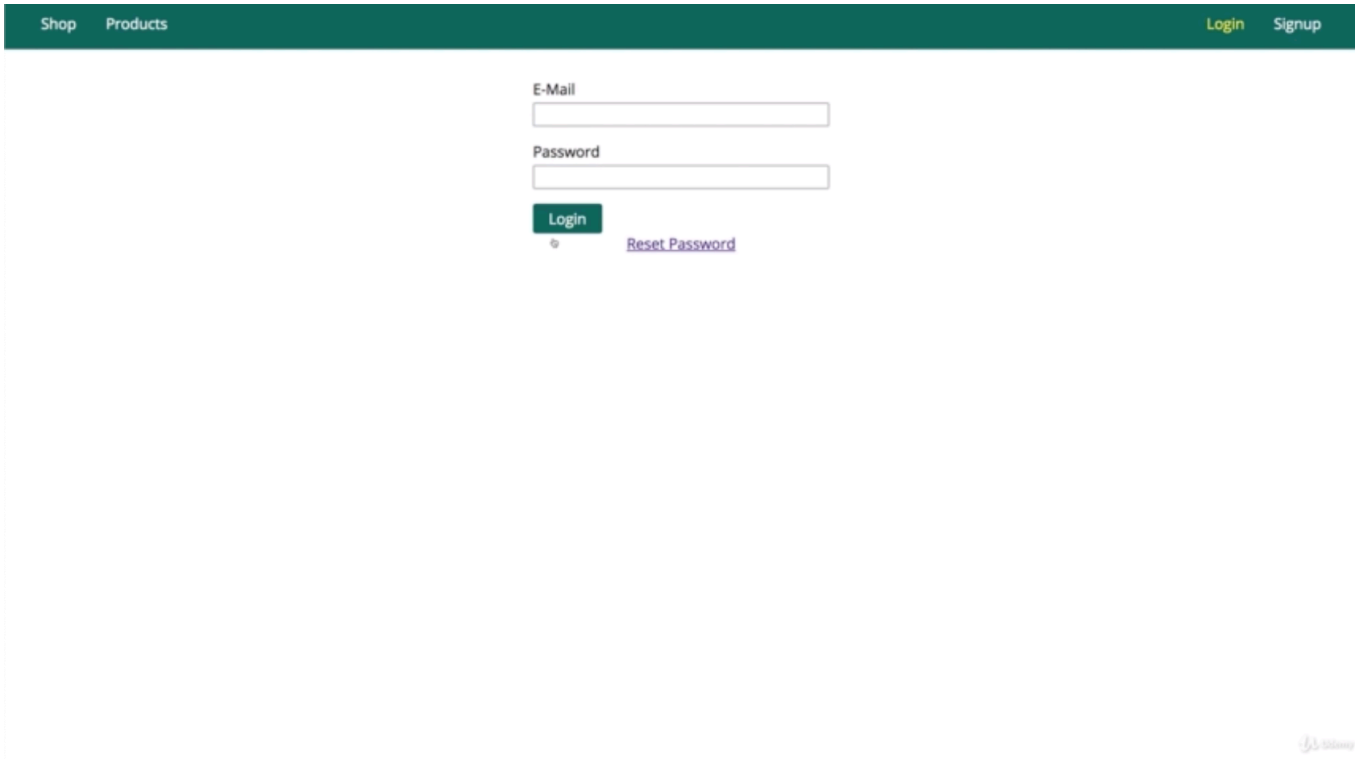
```
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        /**we will pass my userId to that view
         * so that i can include it in the POST request we will update the password
         */
        userId: user._id.toString()
      });
    })
    .catch(err => {
      console.log(err);
    });
};

```

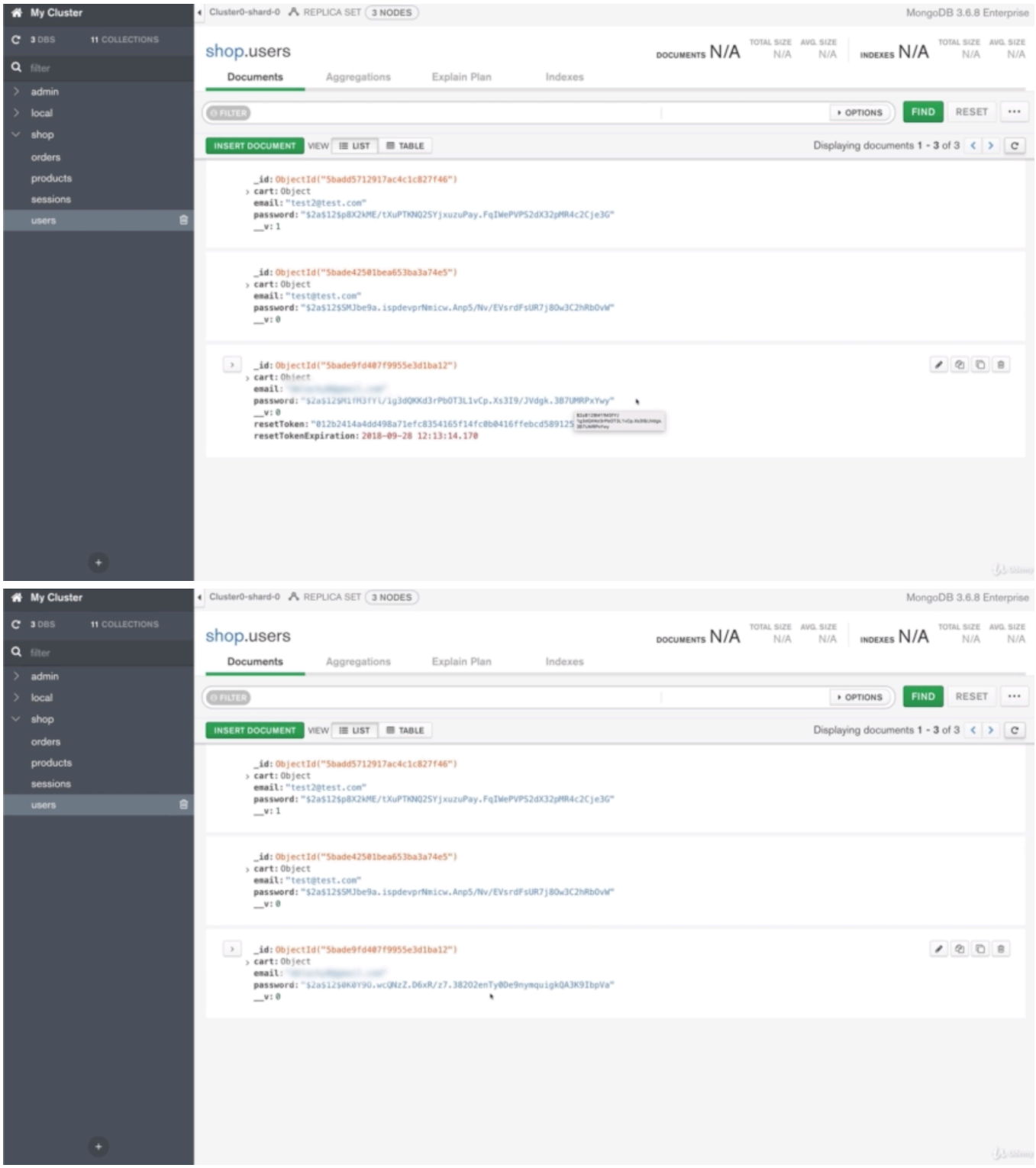
```
// ./routes/auth.js

const express = require('express');

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

router.post('/signup', authController.postSignup);

router.post('/logout', authController.postLogout);

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

router.post('/reset', authController.postReset);

/**it has to be the name called 'token'
 * because in that 'getNewPassword' in authController,
 * i'm looking for token in my params
 * so token here which i'm looking for in the request params means
 * i have to name a token here as well
 */
router.get('/reset/:token', authController.getNewPassword);

module.exports = router;
```

* Chapter 277: Adding Logic To Update The Password

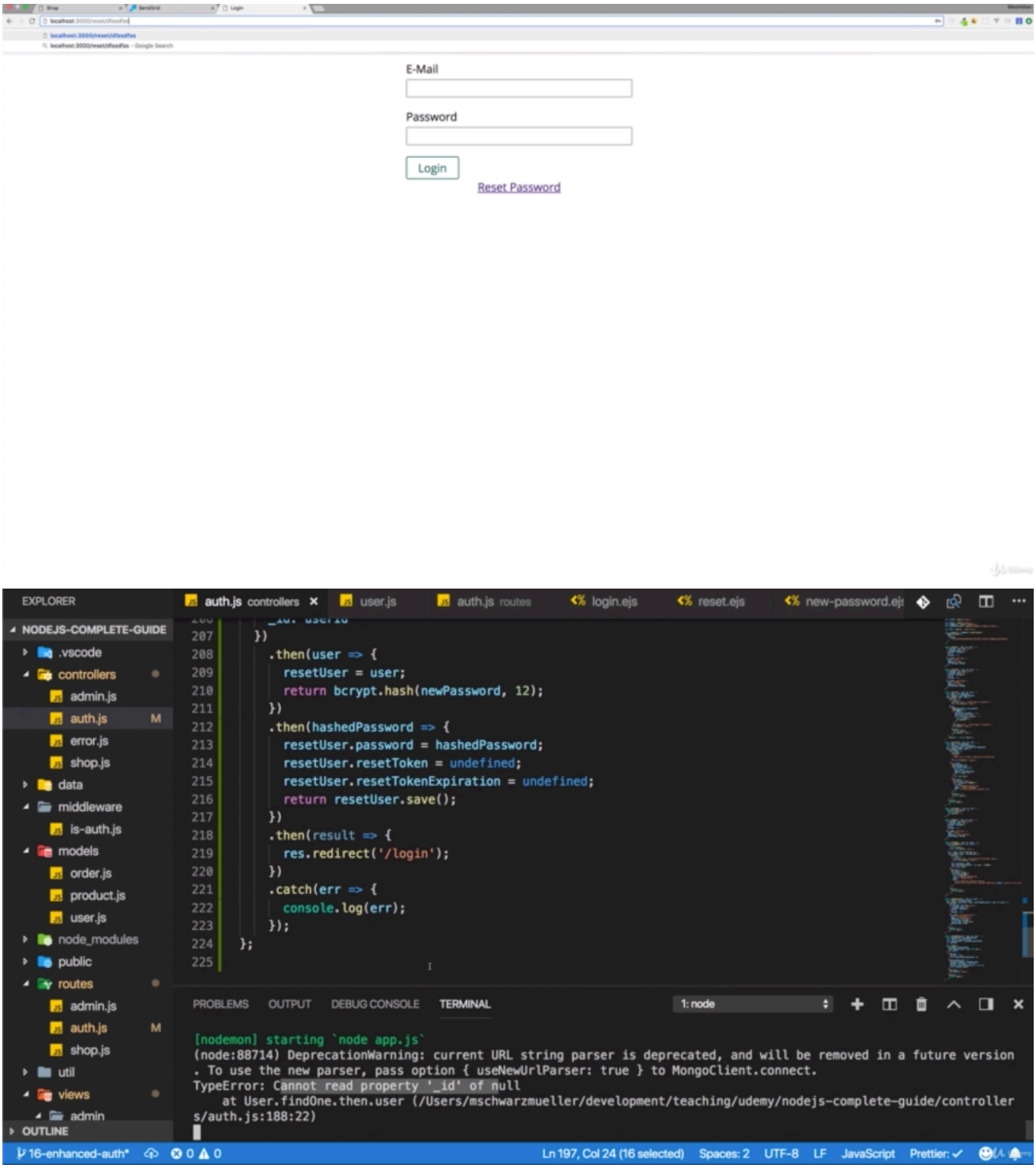
1. update
- ./controllers/auth.js
- ./views/auth/new-password.ejs
- ./routes/auth.js







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



- 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.

```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');

const User = require('../models/user');

```

```
const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
        '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
```

```
      }
      req.flash('error', 'Invalid email or password.');
      res.redirect('/login');
    })
    .catch(err => {
      console.log(err);
      res.redirect('/login');
    });
  })
  .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  const confirmPassword = req.body.confirmPassword;
  User.findOne({ email: email })
    .then(userDoc => {
      if (userDoc) {
        req.flash(
          'error',
          'E-Mail exists already, please pick a different one.'
        );
        return res.redirect('/signup');
      }
      return bcrypt
        .hash(password, 12)
        .then(hashedPassword => {
          const user = new User({
            email: email,
            password: hashedPassword,
            cart: { items: [] }
          });
          return user.save();
        })
        .then(result => {
          res.redirect('/login');
          return transporter.sendMail({
            to: email,
            from: 'shop@node-complete.com',
            subject: 'Signup succeeded!',
            html: '<h1>You successfully signed up!</h1>'
          });
        })
        .catch(err => {
          console.log(err);
        });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
```

```
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>
          `
        });
      })
      .catch(err => {
        console.log(err);
      });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
```

```
  User.findOne({
    resetToken: token,
    resetTokenExpiration: { $gt: Date.now()}
  })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        userId: user._id.toString(),
        passwordToken: token
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postNewPassword = (req, res, next) => {
  /**i still wanna have that token
   * because otherwise people could start entering random tokens
   * and still reach that page and then maybe change users on the back
   * and by entering random userIds and that hidden input field as well.
   * so i wanna have that token.
   */
  const newPassword = req.body.password
  const userId = req.body.userId
  const passwordToken = req.body.passwordToken
  let resetUser

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: {$gt: Date.now()},
    _id: userId
  })
    .then(user => {
      resetUser = user
      return bcrypt.hash(newPassword, 12)
    })
    .then(hashedPassword => {
      resetUser.password = hashedPassword
      /**so these fields are not required anymore
       * they don't need store any values anymore
       */
      resetUser.resetToken = undefined
      resetUser.resetTokenExpiration = undefined
      return resetUser.save()
    })
    .then(result => {
      /**once you did 'save()',
```

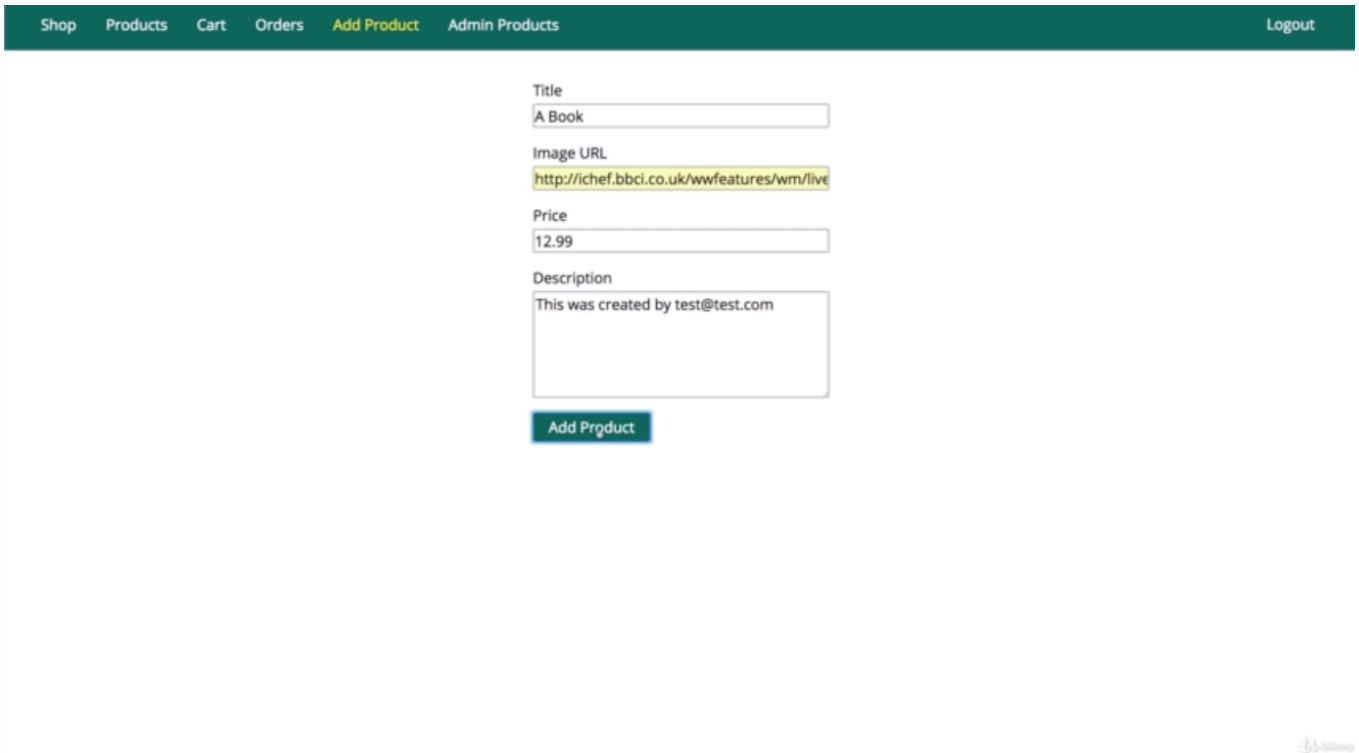
```
      * i can redirect the user back to the log-in page with that new password
      */
     res.redirect('/login')
    })
    .catch(err => {
     console.log(err)
    })

}

```

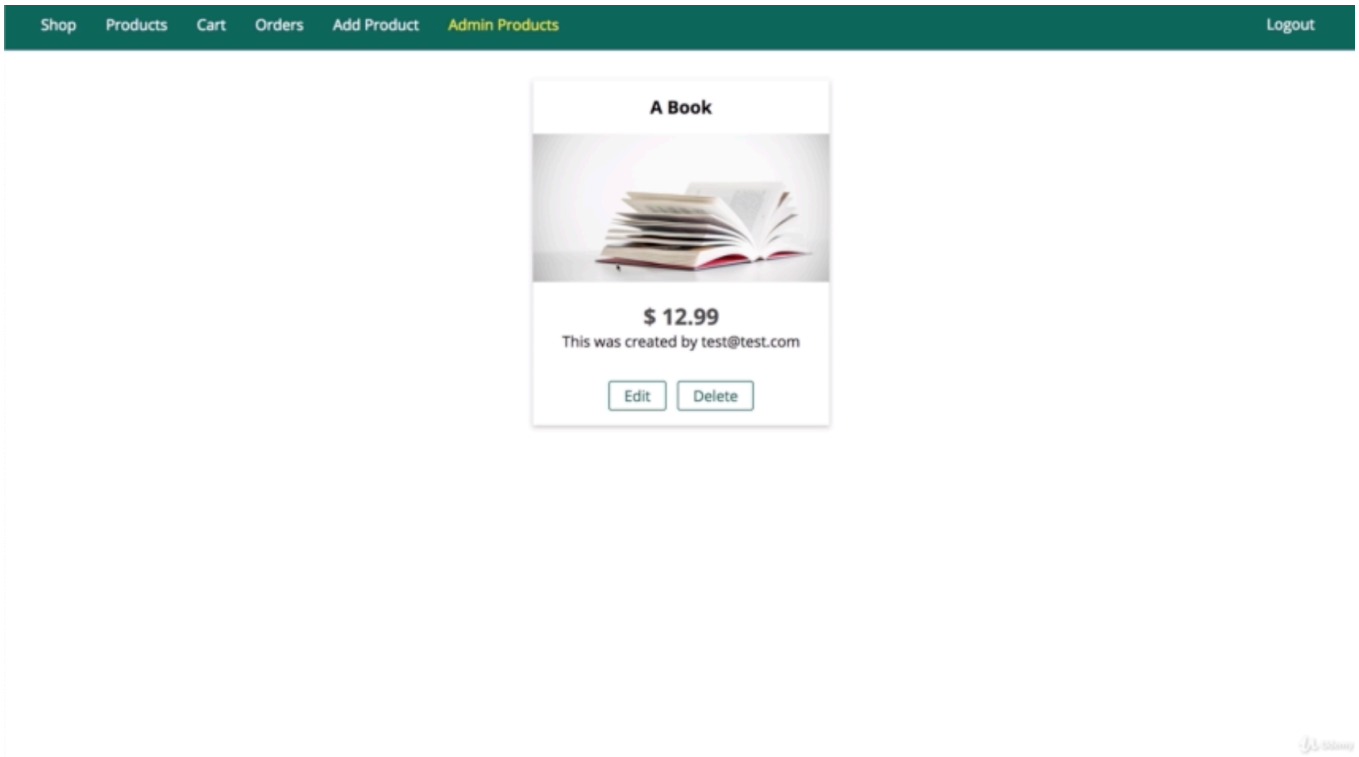
```
<!--./views/auth/new-password.ejs-->

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

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

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/new-password" method="POST">
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password" id="password">
            </div>
            <input type="hidden" name="userId" value="<%= userId %>">
            <input type="hidden" name="passwordToken" value="<%= passwordToken %>">
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Update Password</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

```
// ./routes/auth.js

const express = require('express');

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

router.post('/signup', authController.postSignup);

router.post('/logout', authController.postLogout);

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

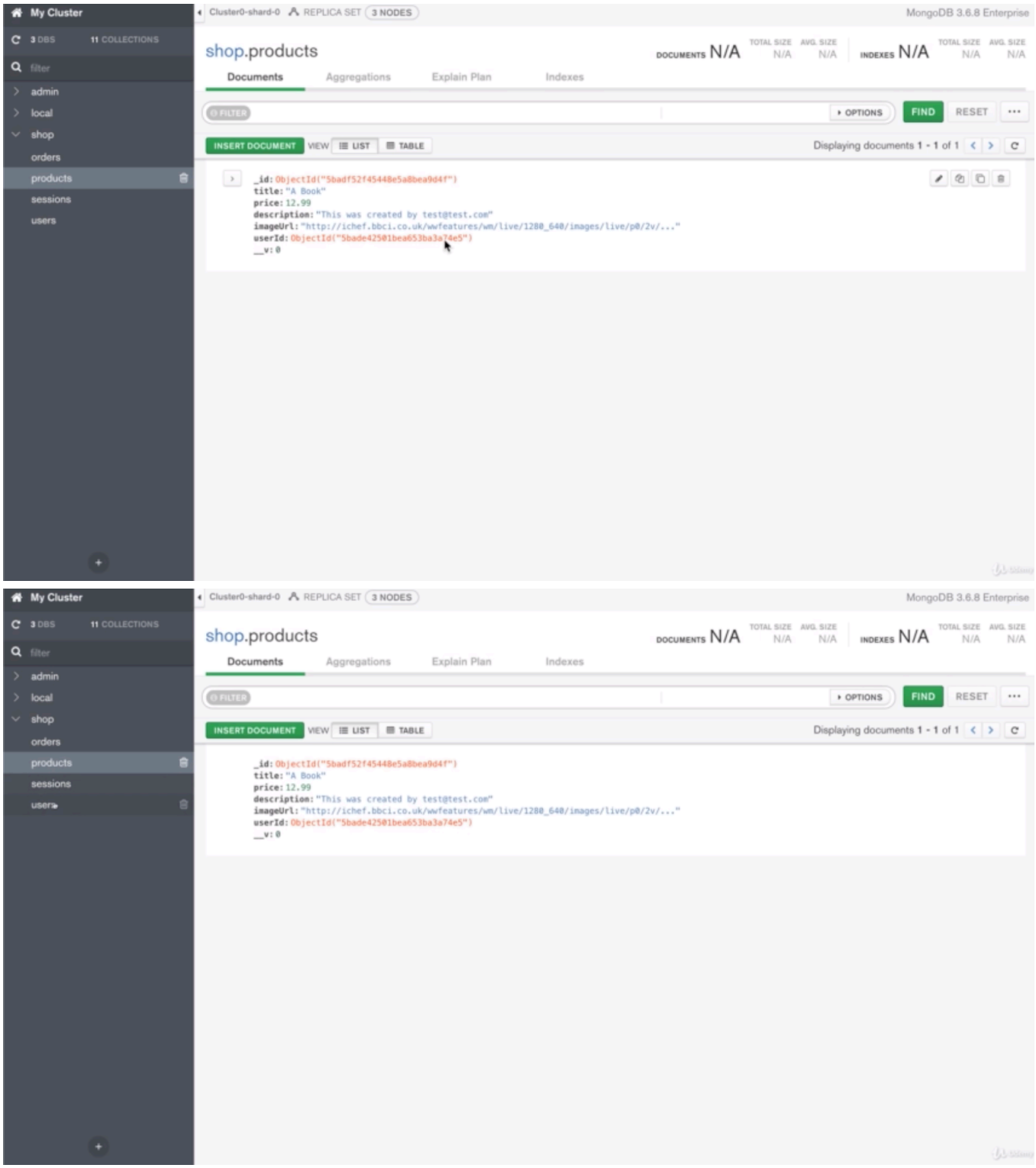
```
20
21 router.post('/reset', authController.postReset);
22
23 router.get('/reset/:token', authController.getNewPassword);
24
25 router.post('/new-password', authController.postNewPassword);
26
27 module.exports = router;
```

* Chapter 278: Why We Need Authorization

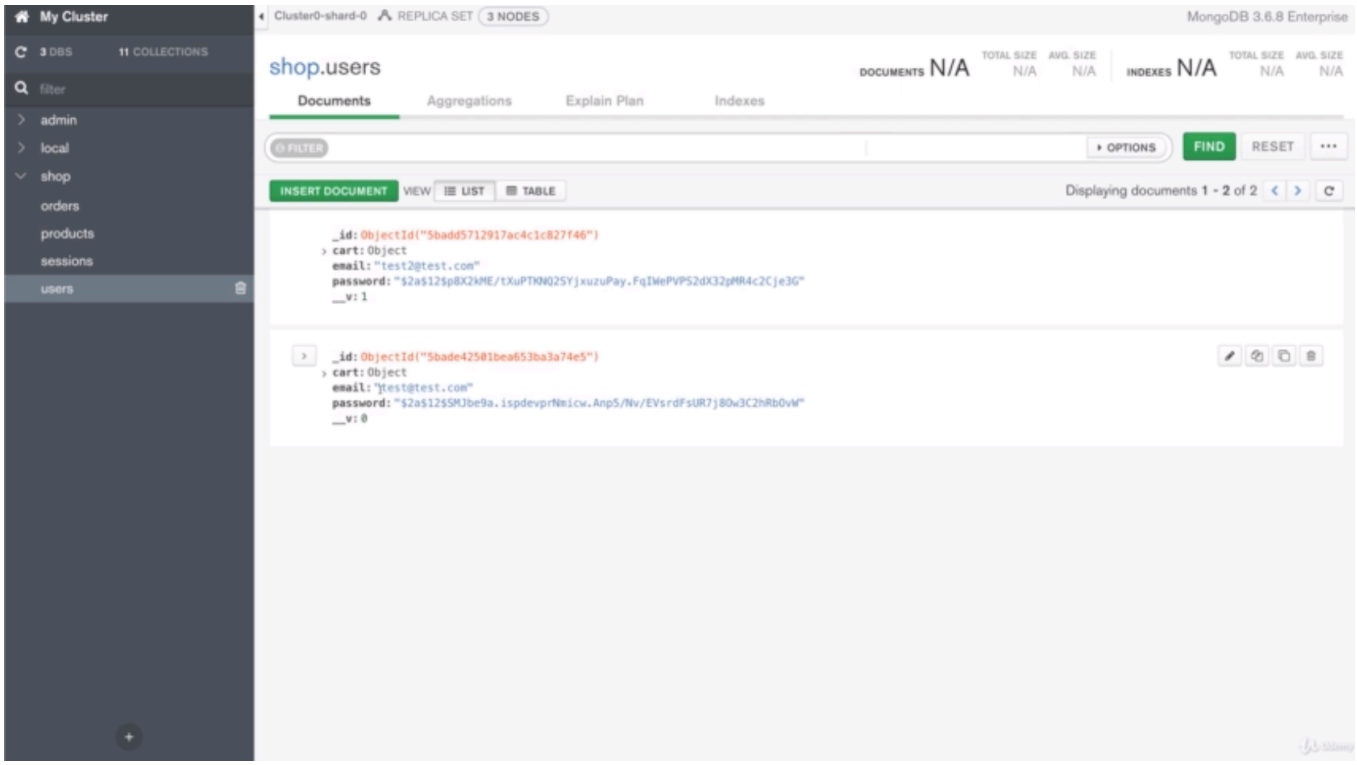




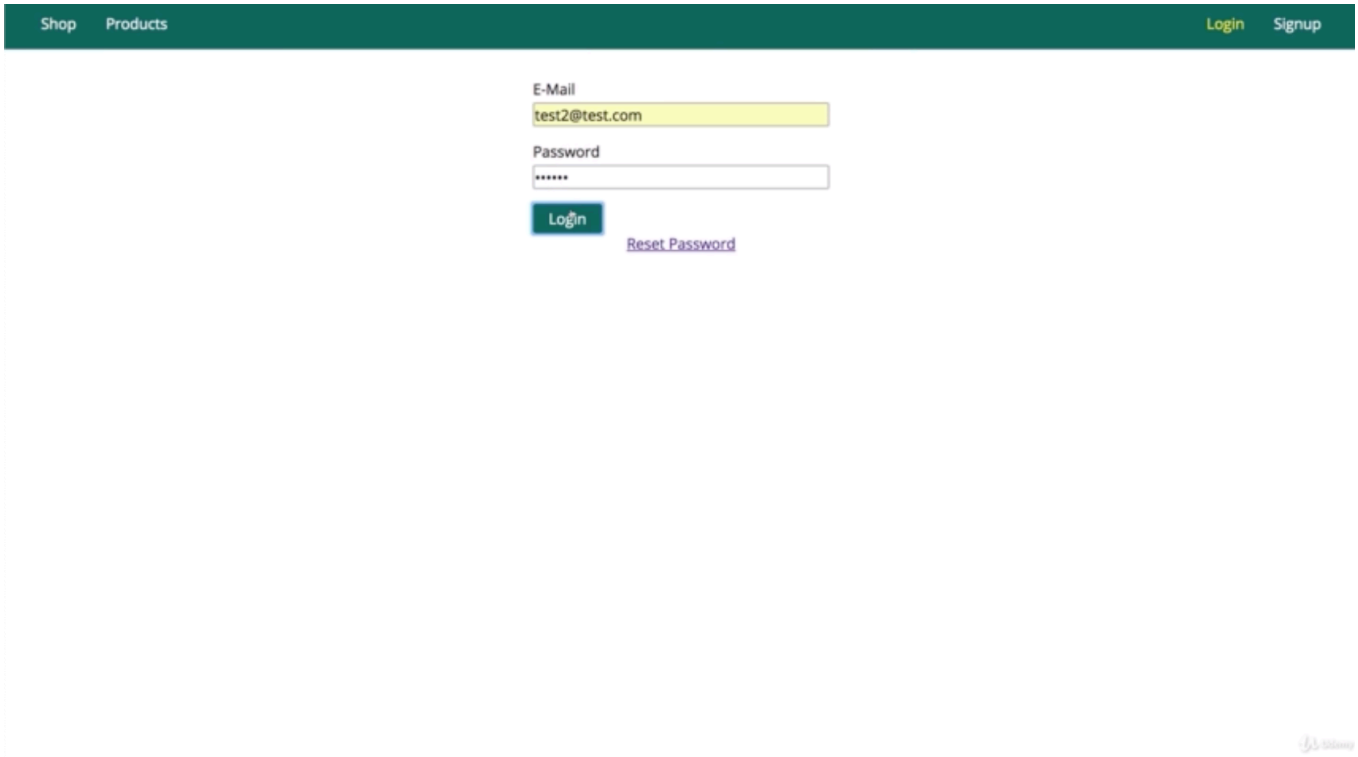
- authorization means that i restrict permissions of a logged in user. so every user might be able to add anything to the car including products created by the user but you might not be able to delete an edit product which were not created by you.

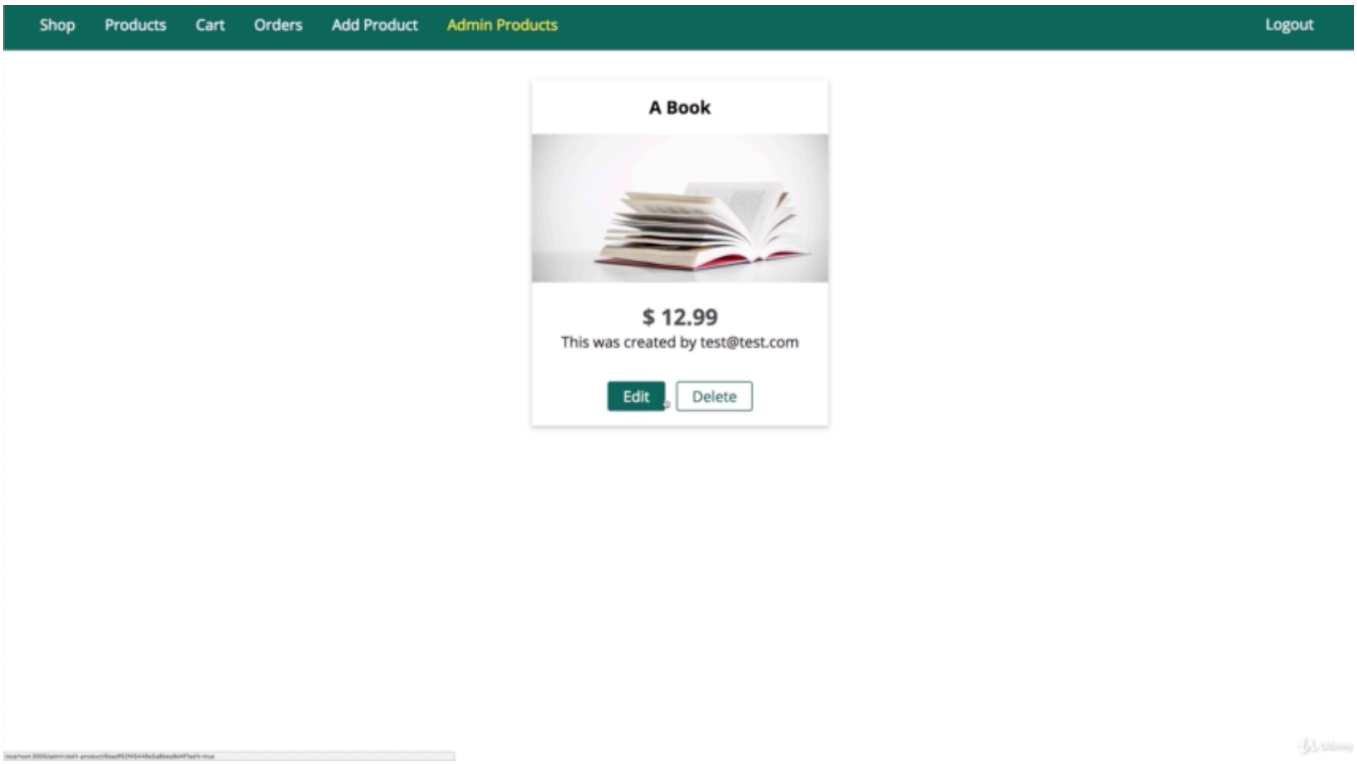


- when i create this product here, we have it here and we can see it in the product collection obviously. it's linked to the user with an ID which ends with 'e5'



- and this is the ‘e5’ and this is ’test@test.com’

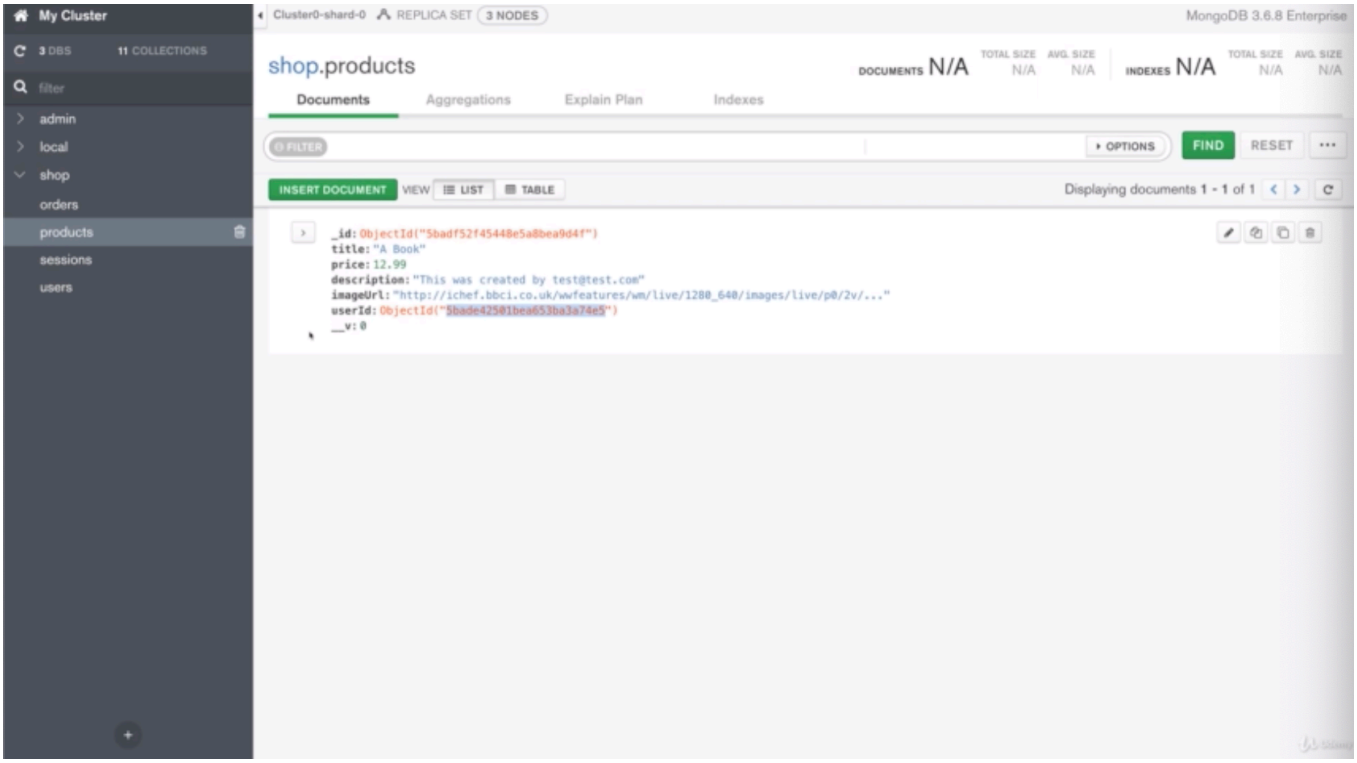




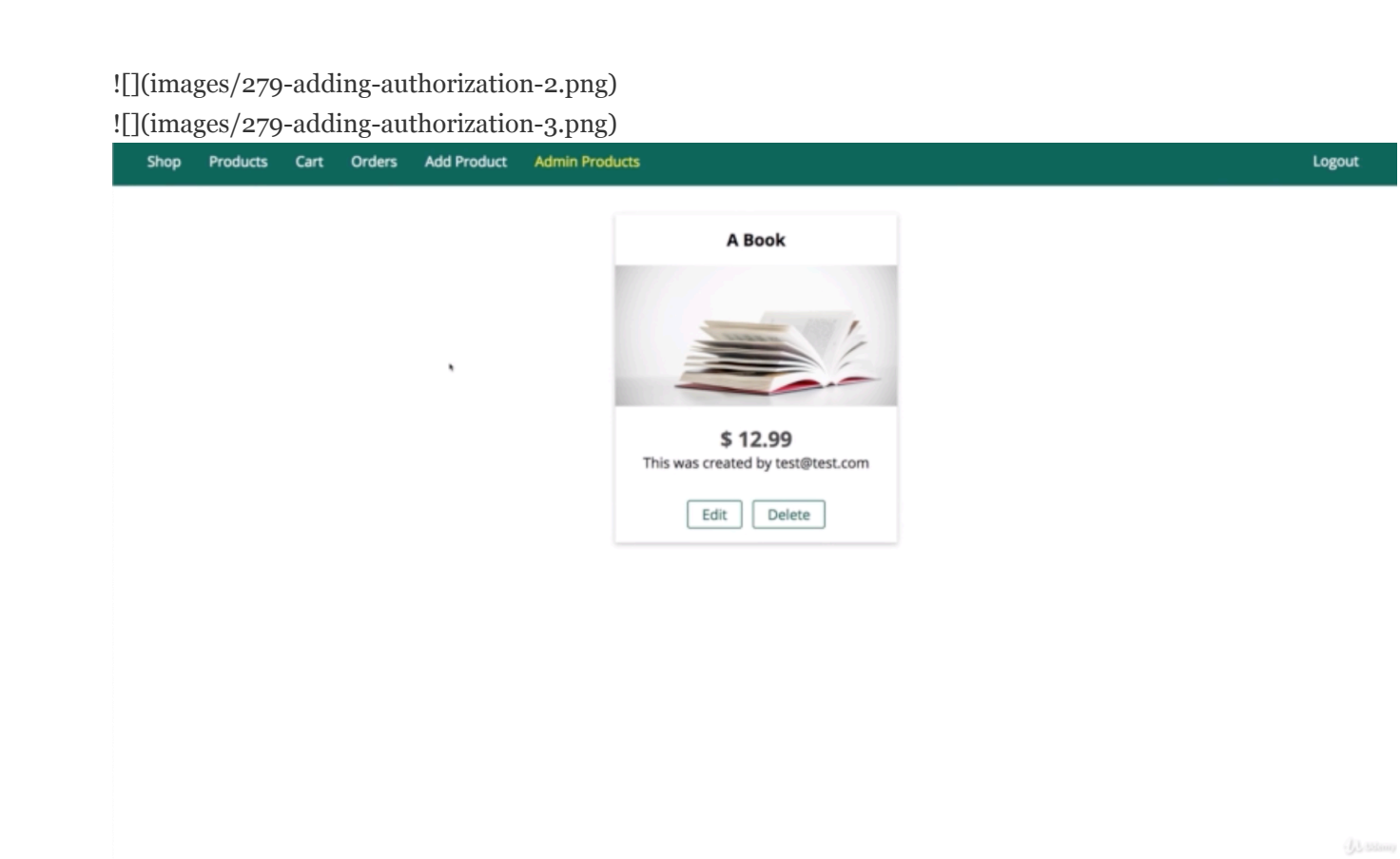
- now we login 'test2@test.com' then if i go to Admin Products, i can still edit and delete it. this is what we wanna prevent from which means we wanna let only me edit or delete and don't let others edit or delete it.

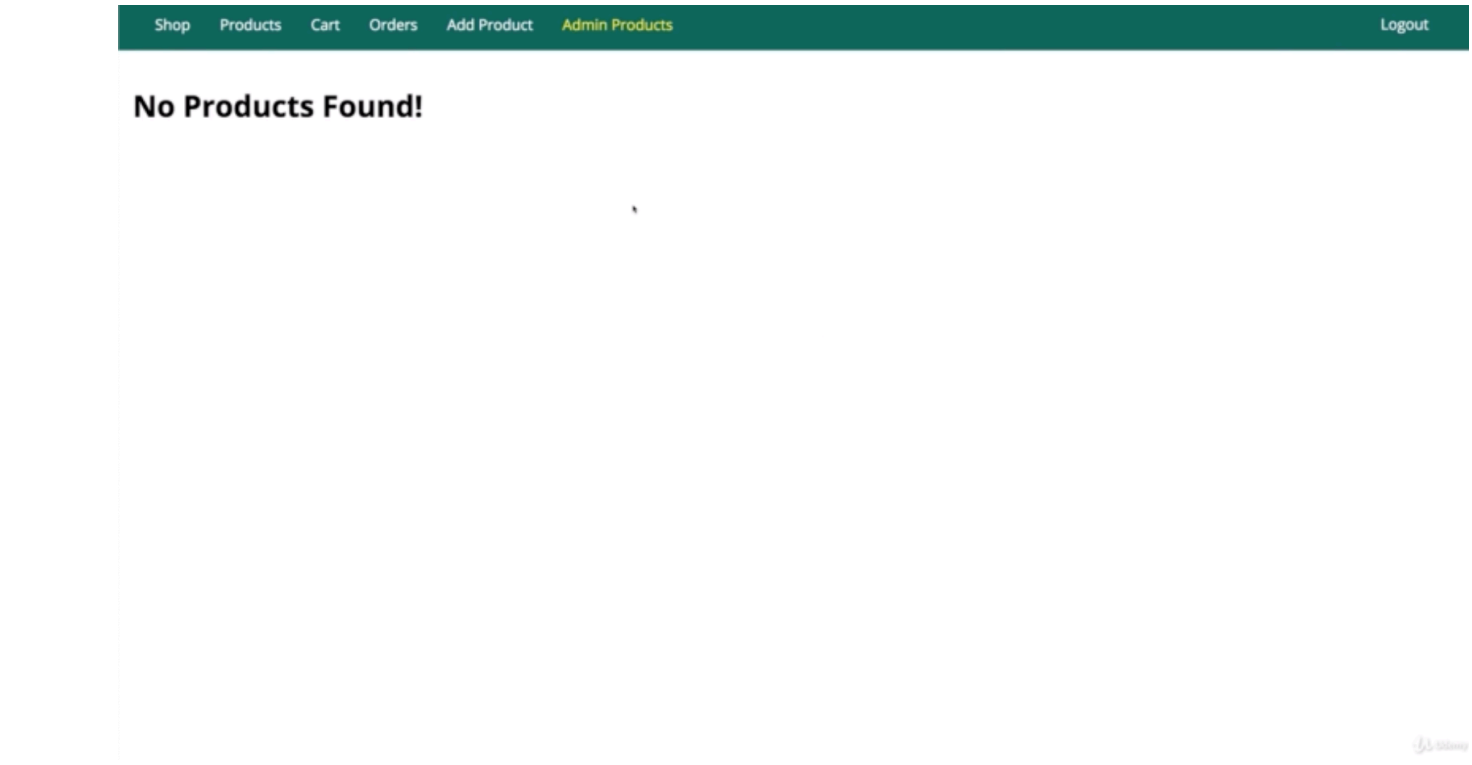
* Chapter 279: Adding Authorization

1. update
- ./controllers/admin.js

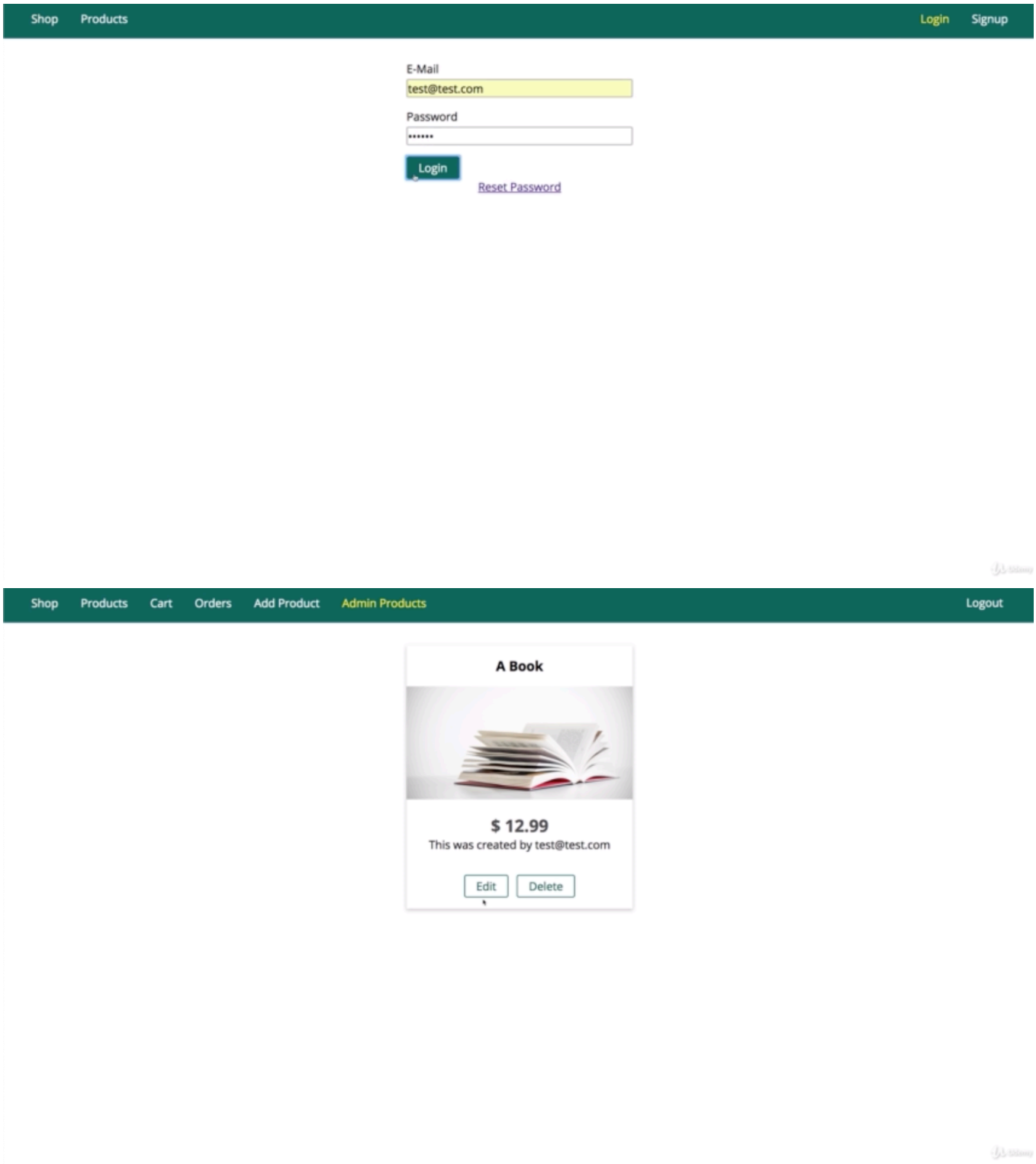


- we know which user did create it. so in the end, we wanna check if the currently logged in user is the user who created that before allowing any edits to that item.





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



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

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

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

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

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

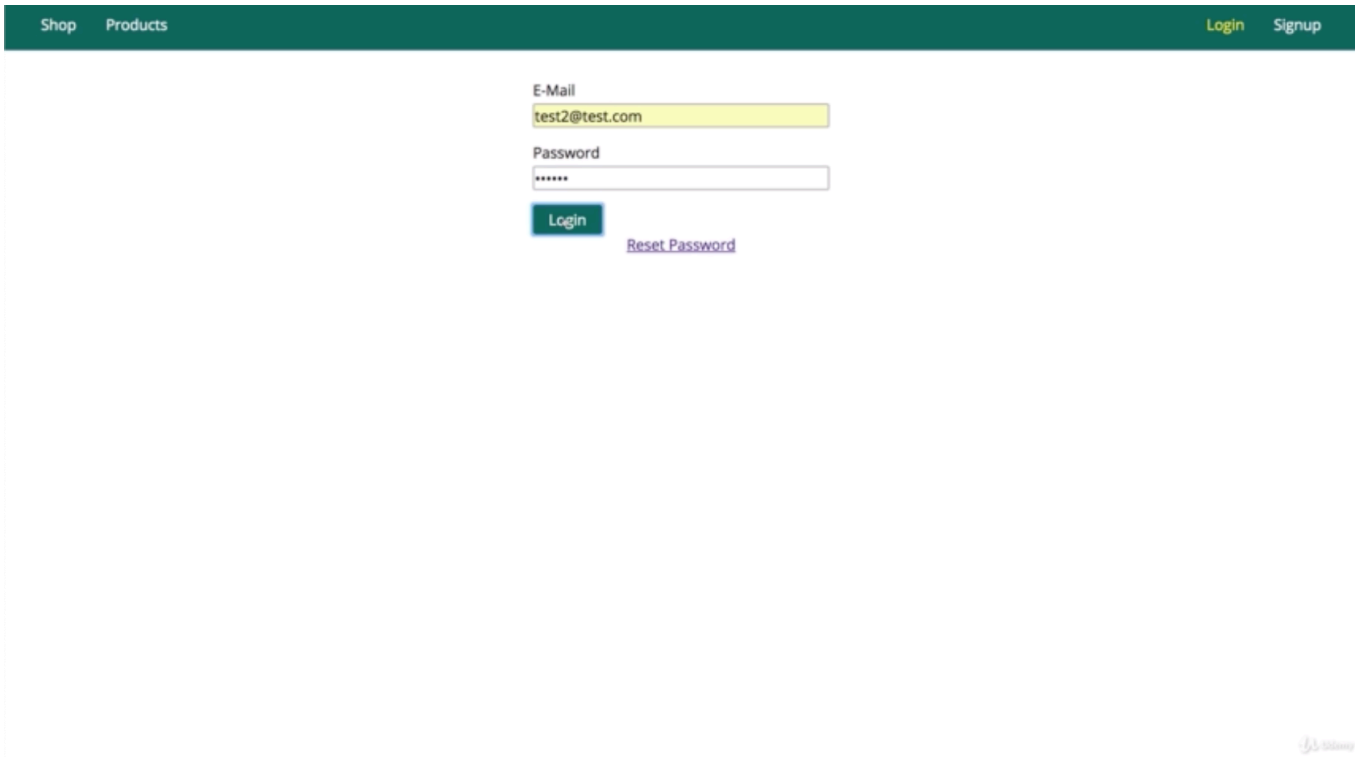
```
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const product = new Product({
    title: title,
    price: price,
    description: description,
    imageUrl: imageUrl,
    userId: req.user
  });
 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

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

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

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

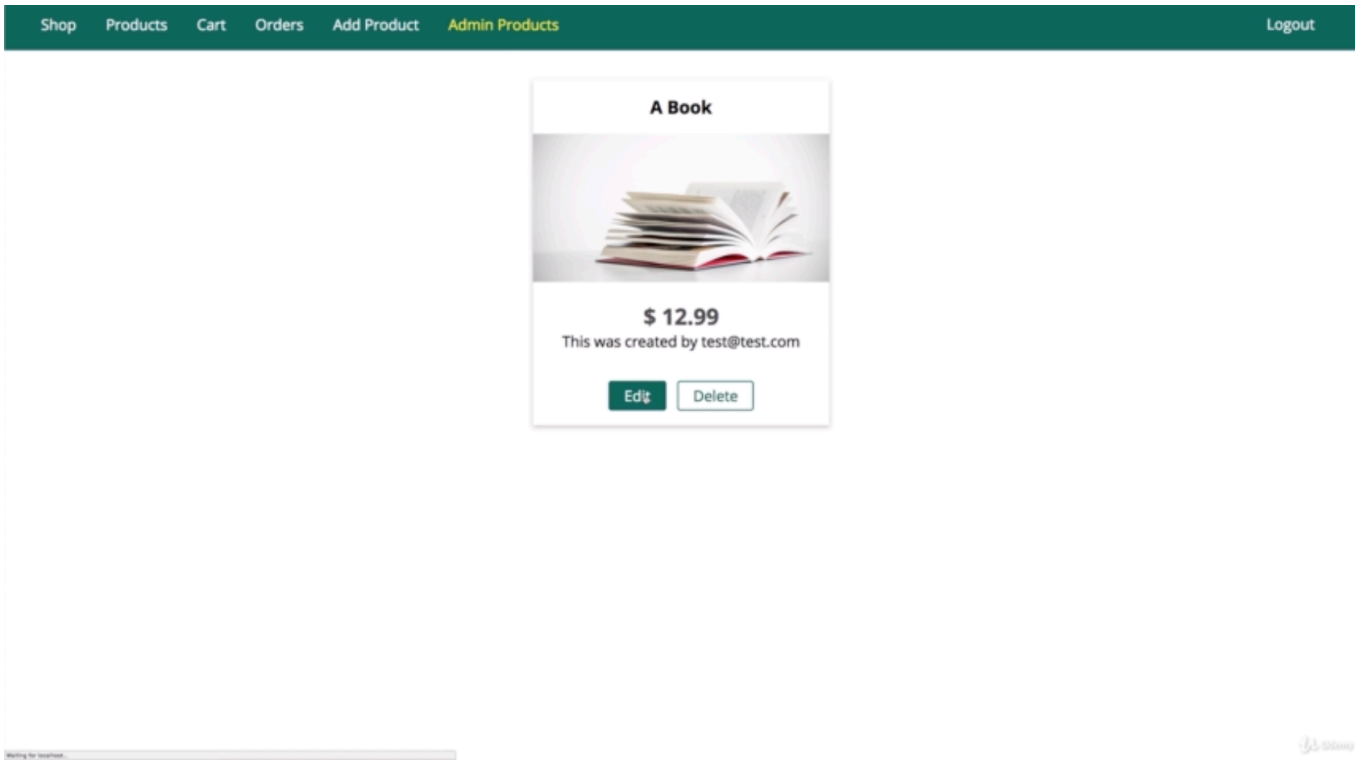
```
      product.imageUrl = updatedImageUrl;
      return product.save();
    })
    .then(result => {
      console.log('UPDATED PRODUCT!');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  /**authorization means we restrict the permissions
   * and we can do that by restricting the data we return.
   * so here when i find product,
   * i don't find all
   * but i will add a filter
   * and i will filter for products where the the userId is equal to userId of the currently logged in user,
   * so userId is equal to req.user._id
   */
  Product.find({userId: req.user._id})
    // .select('title price -_id')
    // .populate('userId', 'name')
    .then(products => {
      console.log(products);
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

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

```

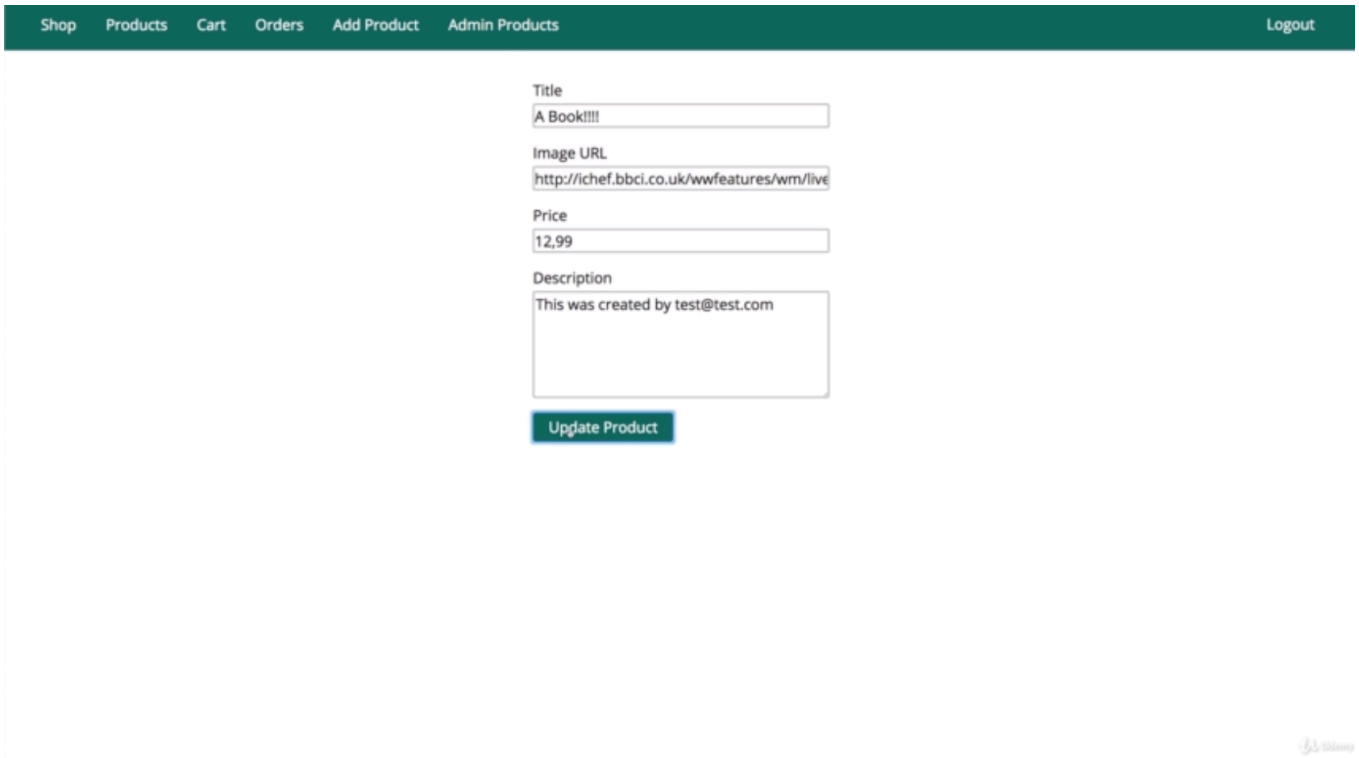
* Chapter 280: Adding Protection To Post Actions

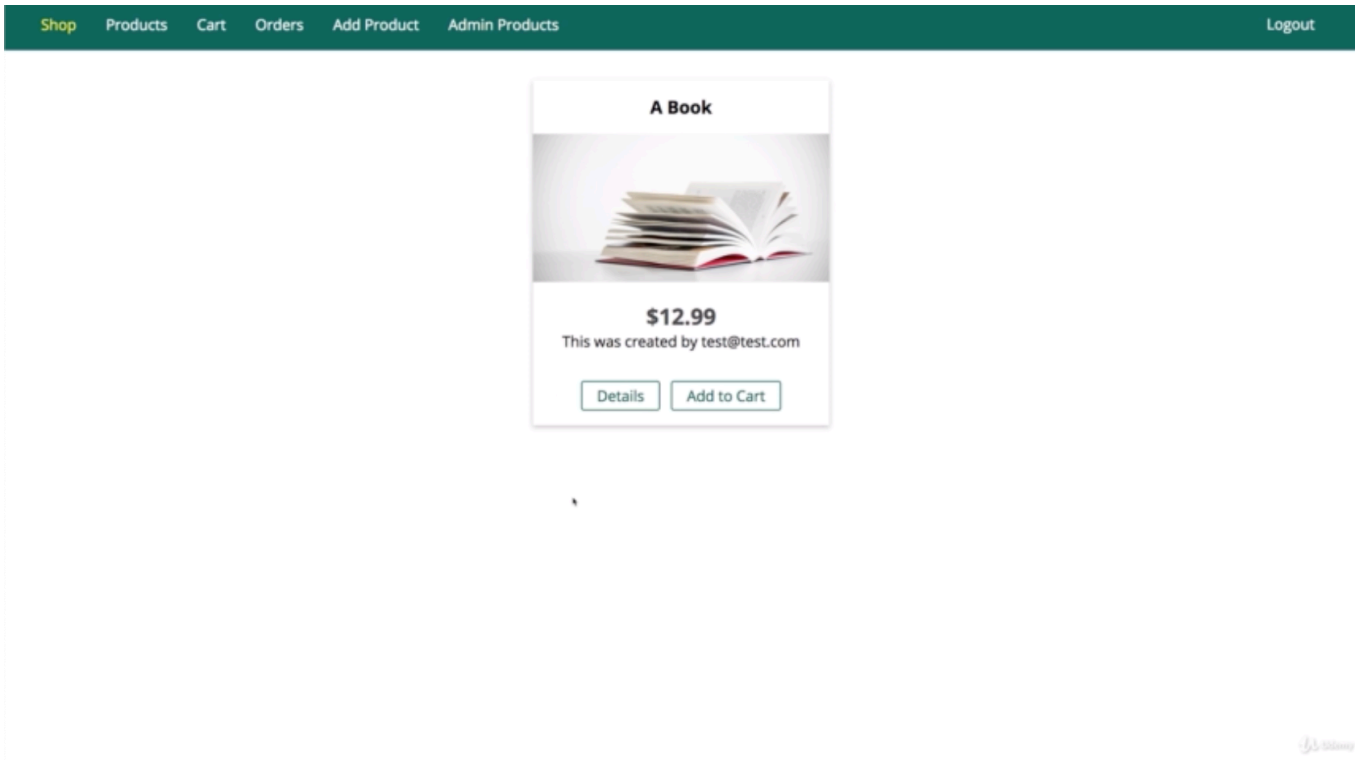
1. update
- ./controllers/admin.js

!(images/280-adding-protection-to-post-actions-4.png)

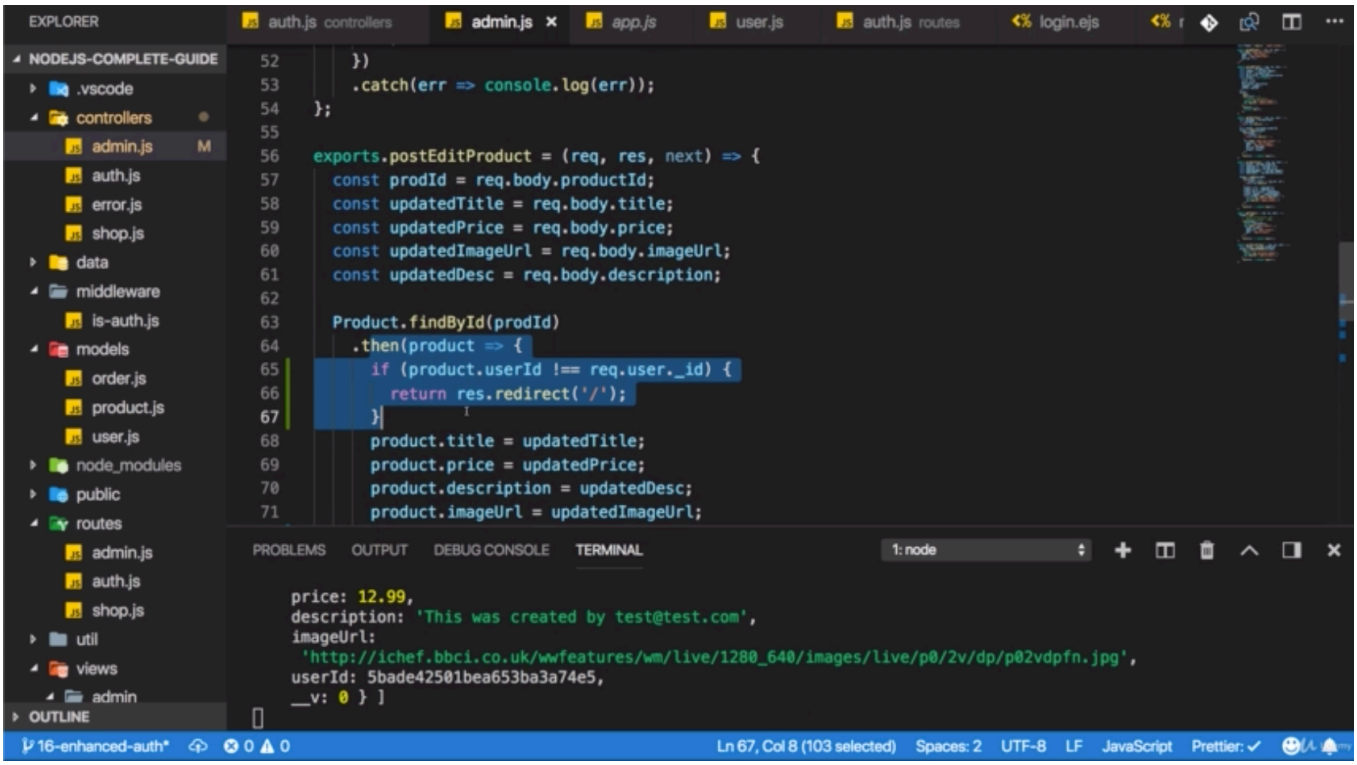




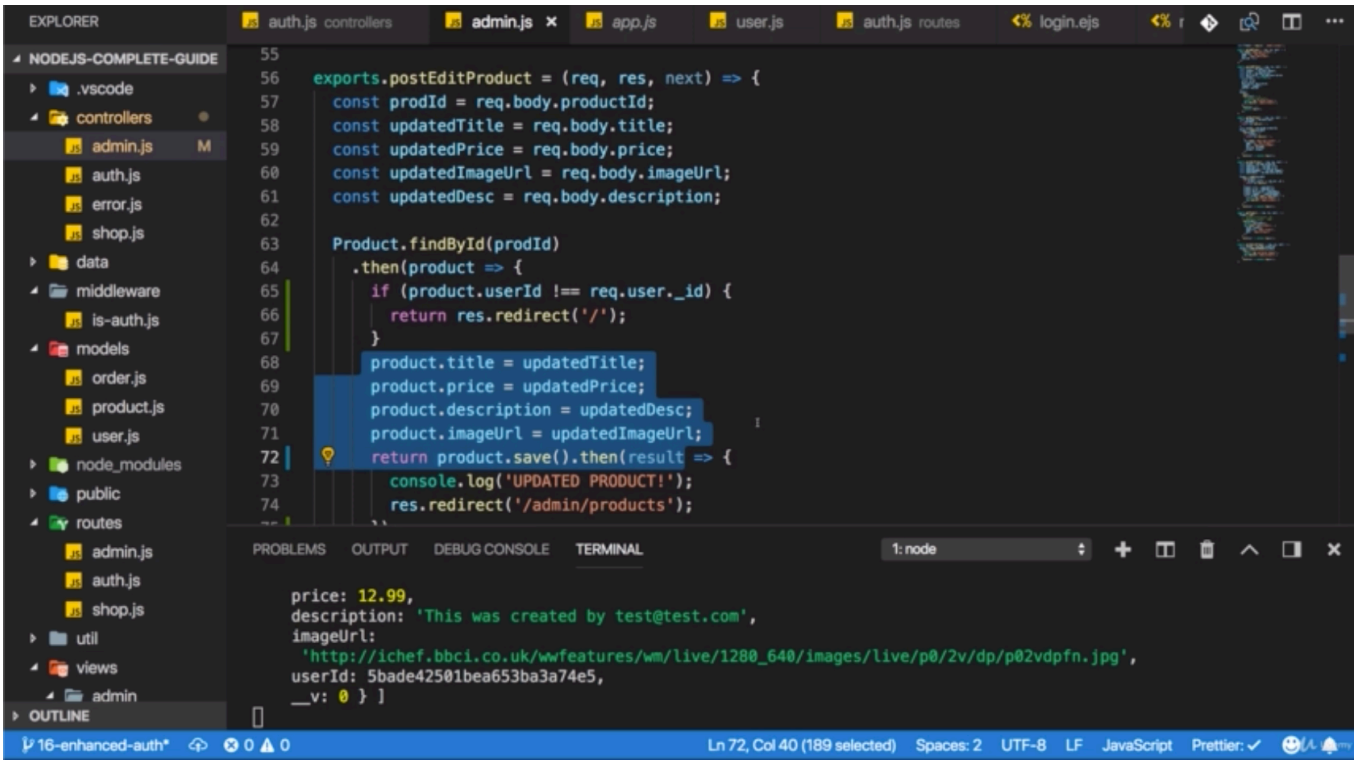




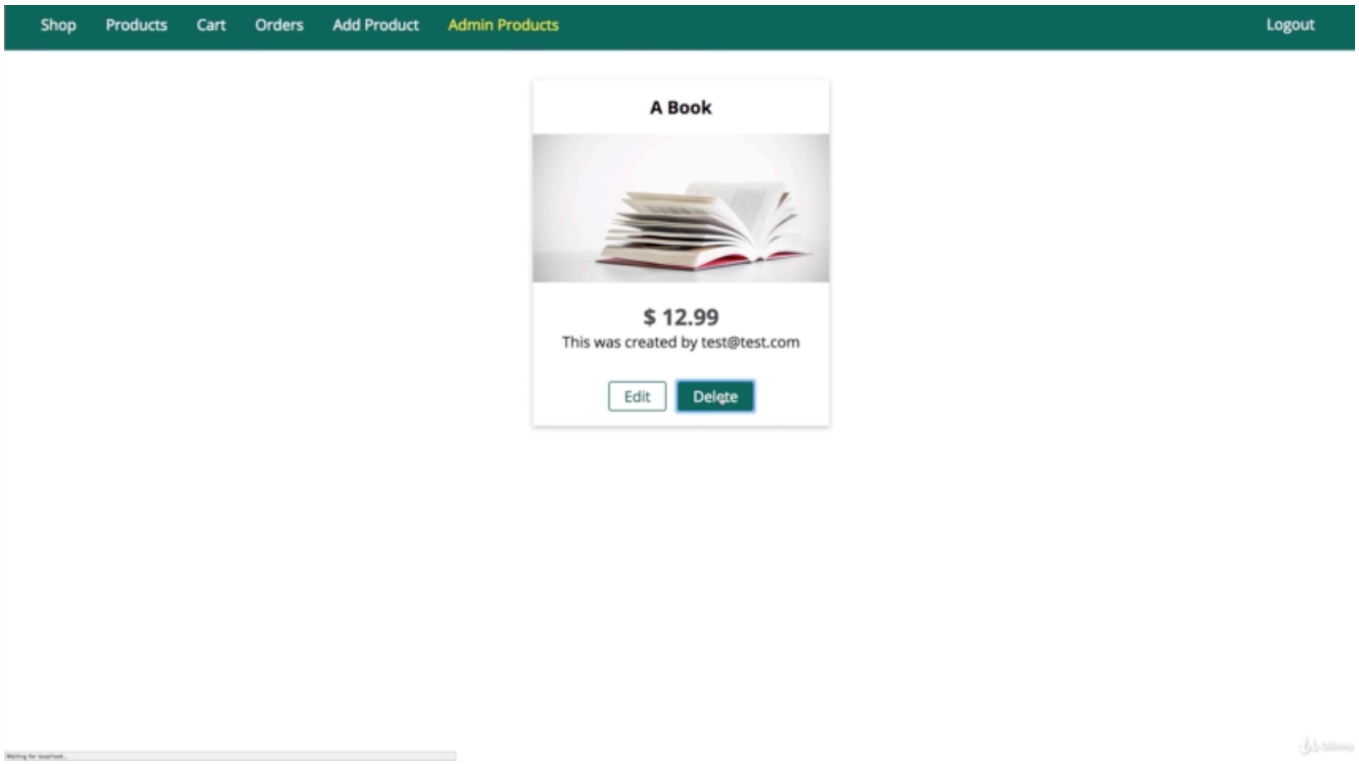
- i’m redirected but the edit wasn’t saved

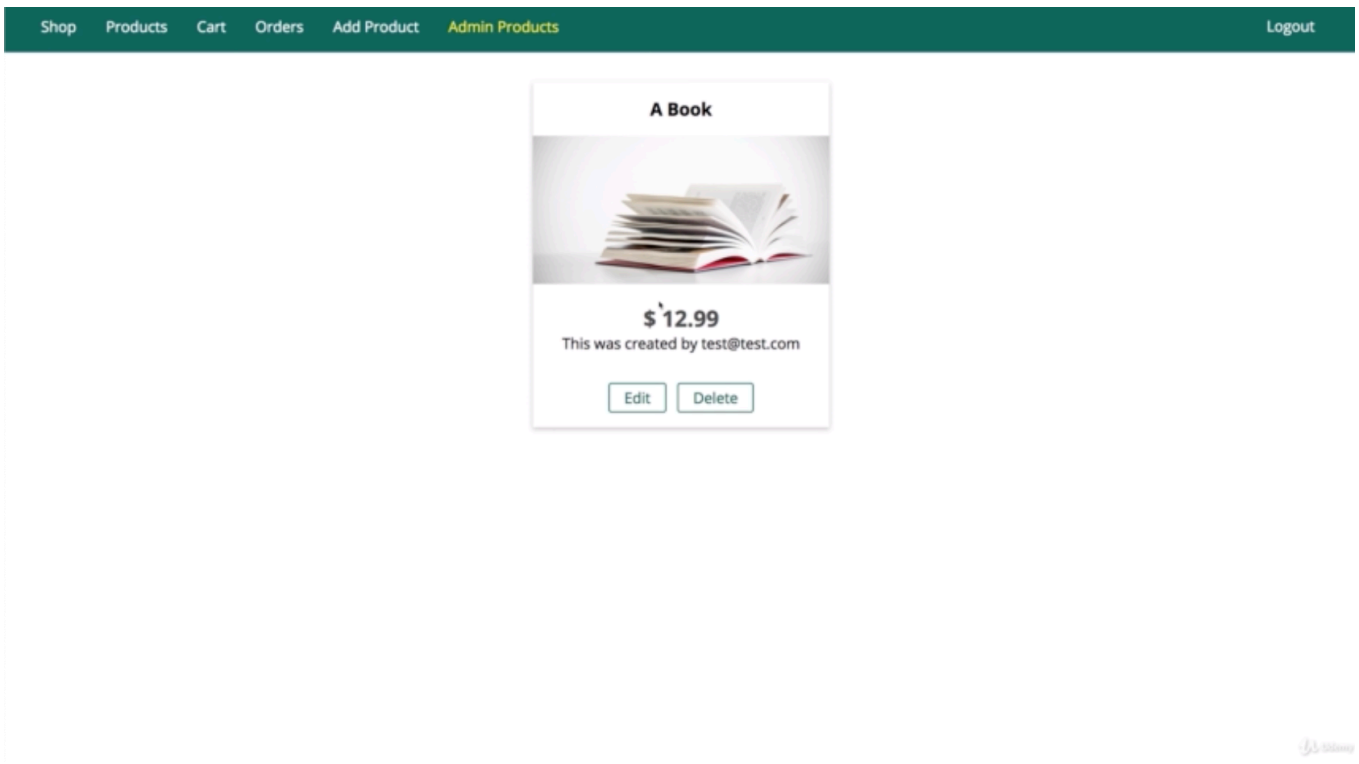


because i essentially ended up in my extra check here and i was redirected back but without saving the changes



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.

Shop Products Login Signup

E-Mail

test@test.com

Password

Login

Reset Password

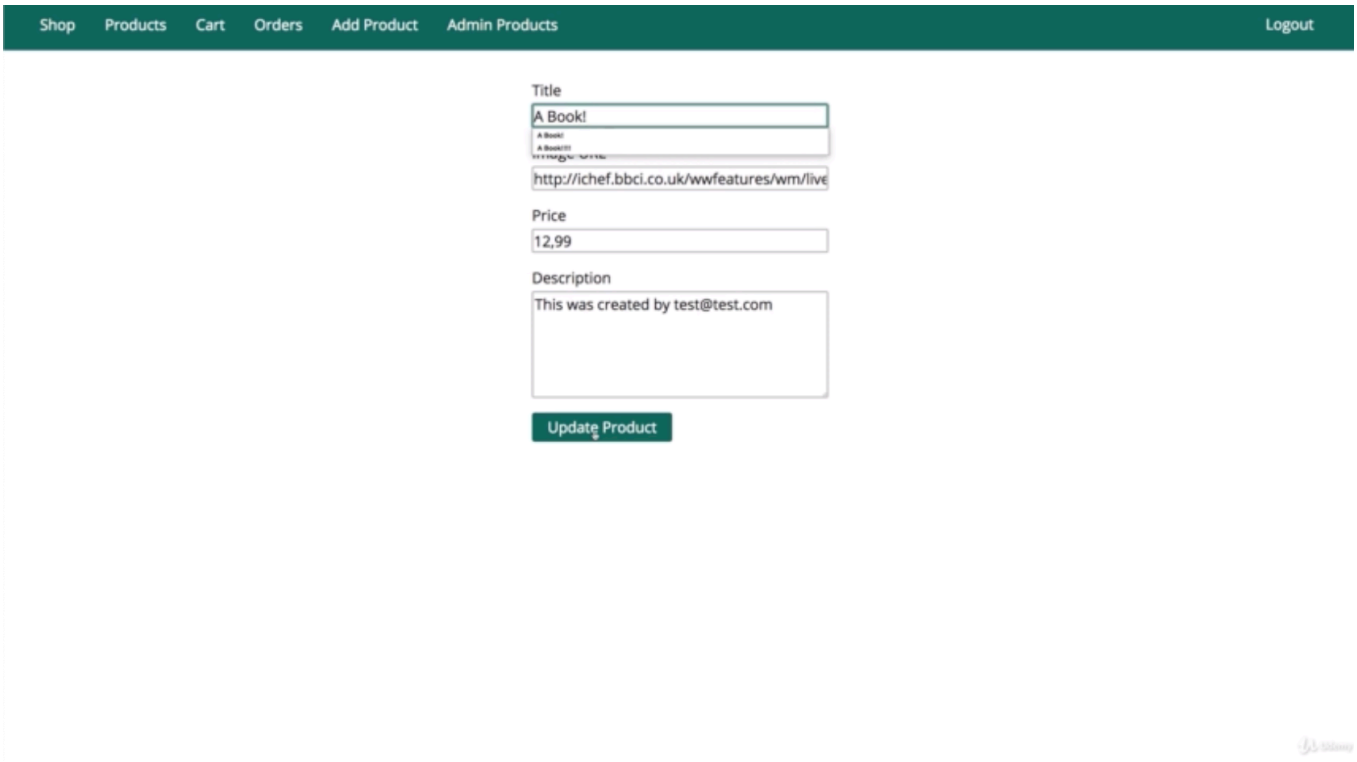
Shop Products Cart Orders Add Product Admin Products Logout

A Book

$ 12.99

This was created by test@test.com

Edit Delete



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

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

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

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

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

```
};

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

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

  Product.findById(prodId)
    .then(product => {
  /**we wanna check that the product i try to delete is really created by the user
   * who is currently logged in user */
      if(product.userId !== req.user._id){
        return res.redirect('/')
      }
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save()
        .then(result => {
          console.log('UPDATED PRODUCT!');
          res.redirect('/admin/products');
        })
      })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.find({userId: req.user._id})
    // .select('title price -_id')
    // .populate('userId', 'name')
    .then(products => {
      console.log(products);
```

```
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

/**i wanna check if i'm really allowed to delete that product
 * and we can implement this by changing our deletion method
 * and use 'deleteOne()'
 */
exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  /**both has to be true now.
   * prodId is not enough,
   * req.user._id also has to be matched
   *
   * even if a valid userId is there,
   * it will not delete this product if the userId doesn't match
  */
  Product.deleteOne({_id: prodId, userId: req.user._id})
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

```

* Chapter 281: Why Editing Fails?

1. update
- ./controllers/admin.js

Shop Products Cart Orders Add Product Admin Products Logout

A Book

$ 12.99

This was created by test@test.com

Edit Delete

Shop Products Cart Orders Add Product Admin Products Logout

Title

A Book!

Image URL

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

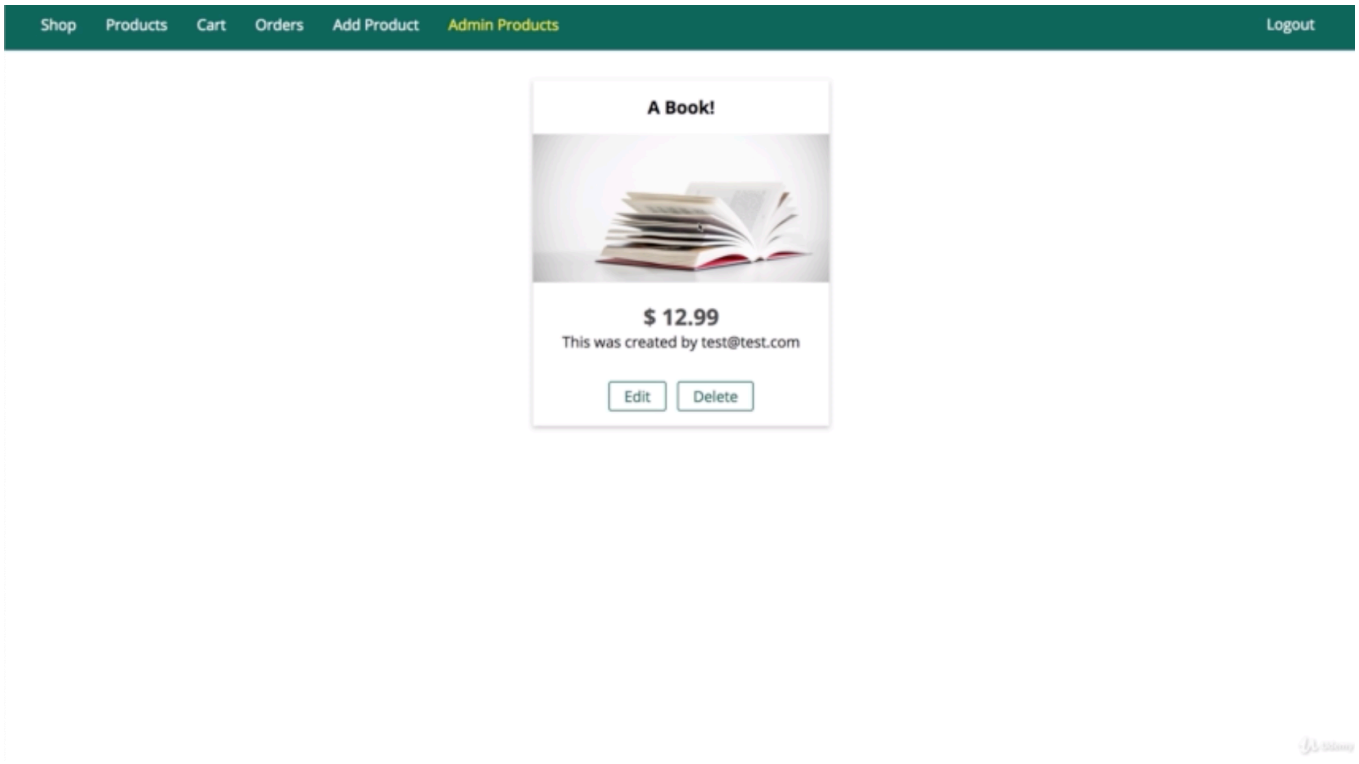
Price

12,99

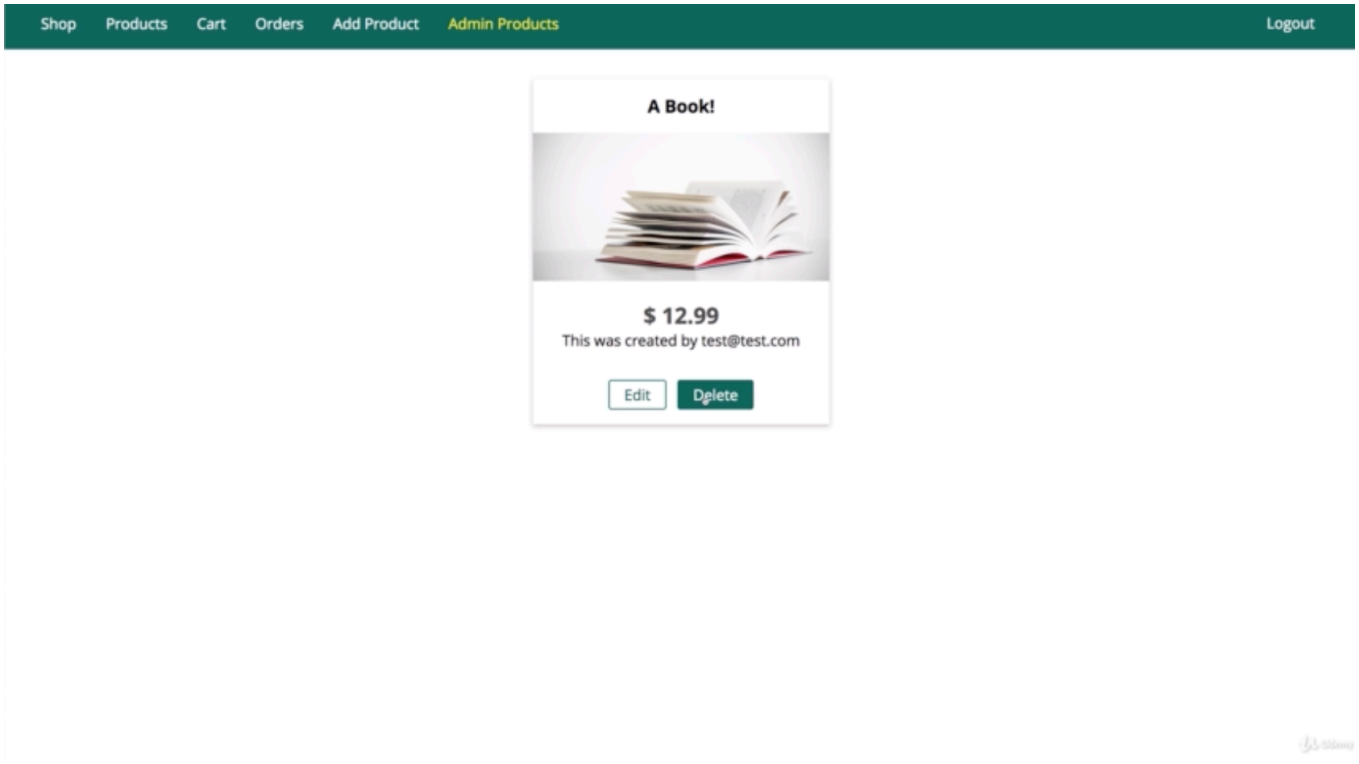
Description

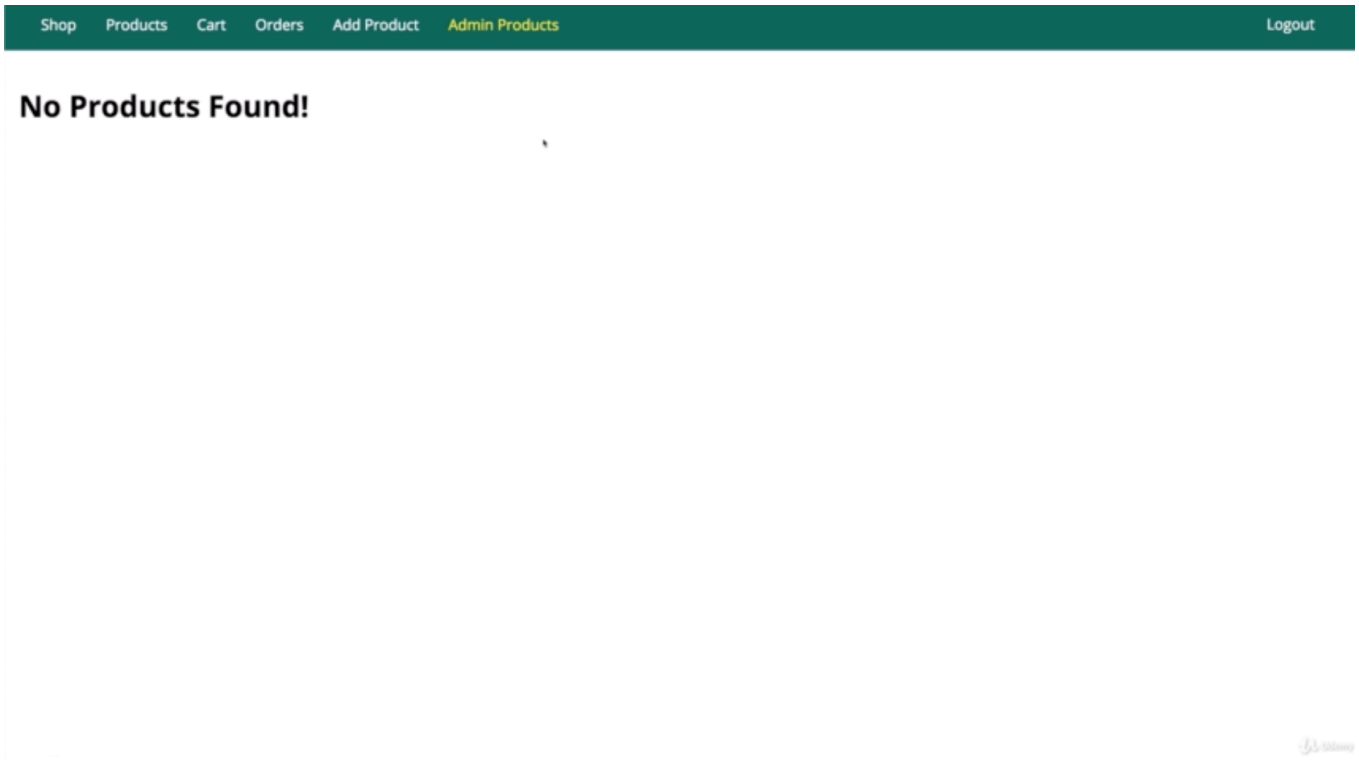
This was created by test@test.com

Update Product

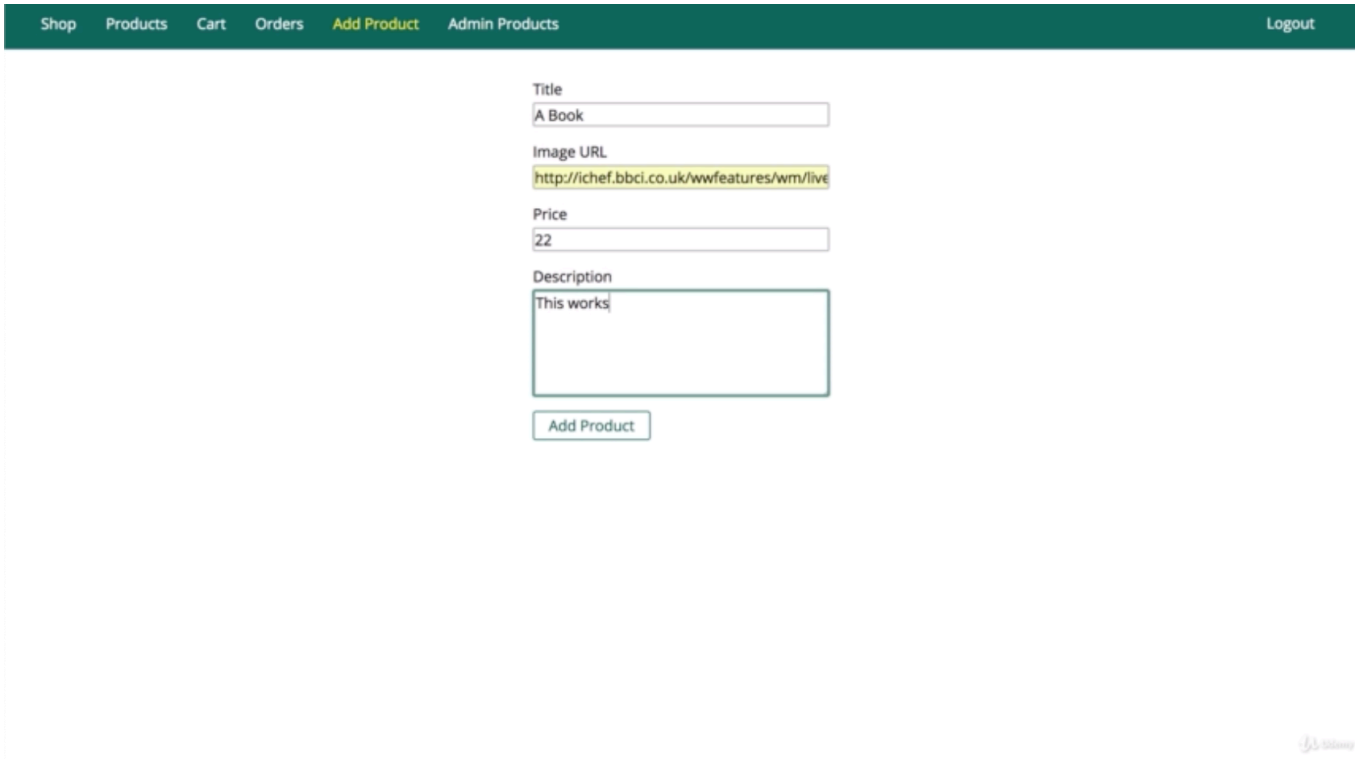


- now i can update this





- i also can delete this.



Shop Products Cart Orders Add Product Admin Products Logout

A Book

$ 22

This works!

Edit Delete

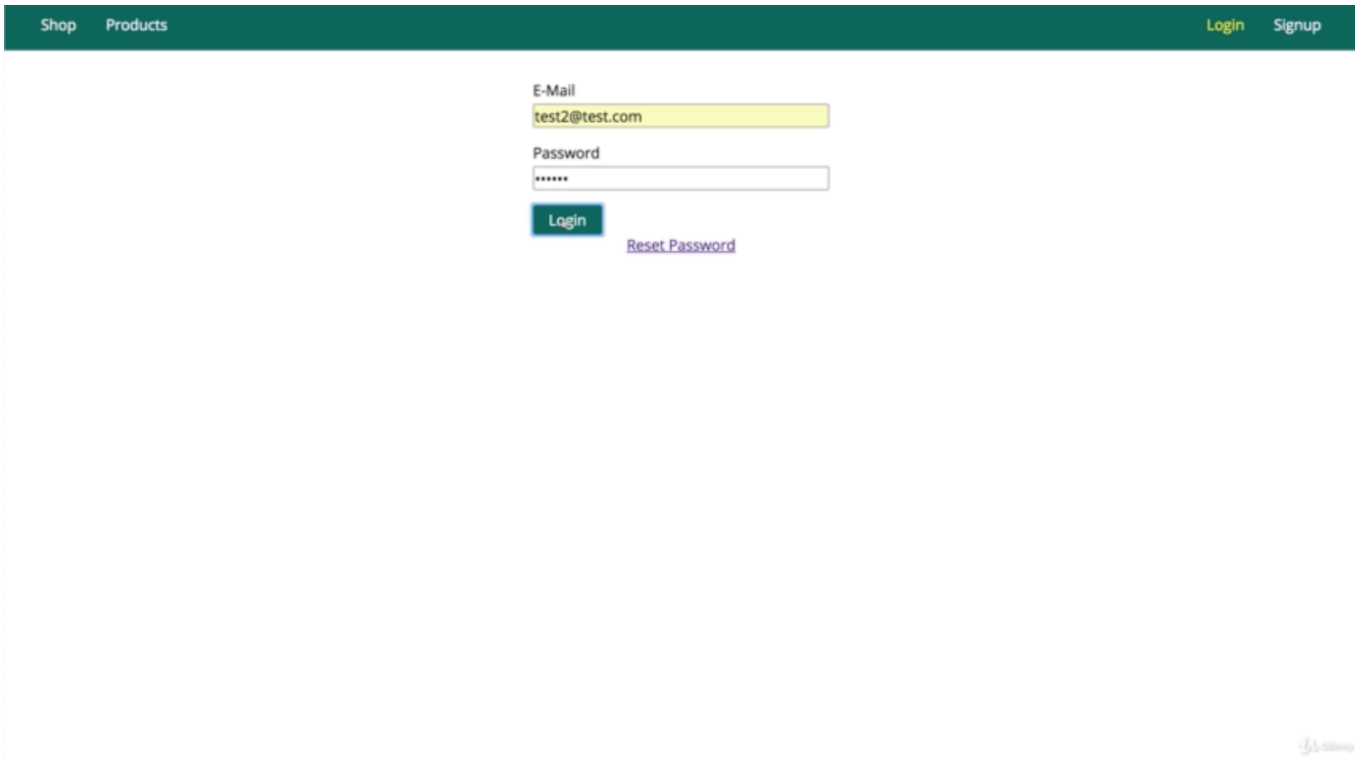
Shop Products Cart Orders Add Product Admin Products Logout

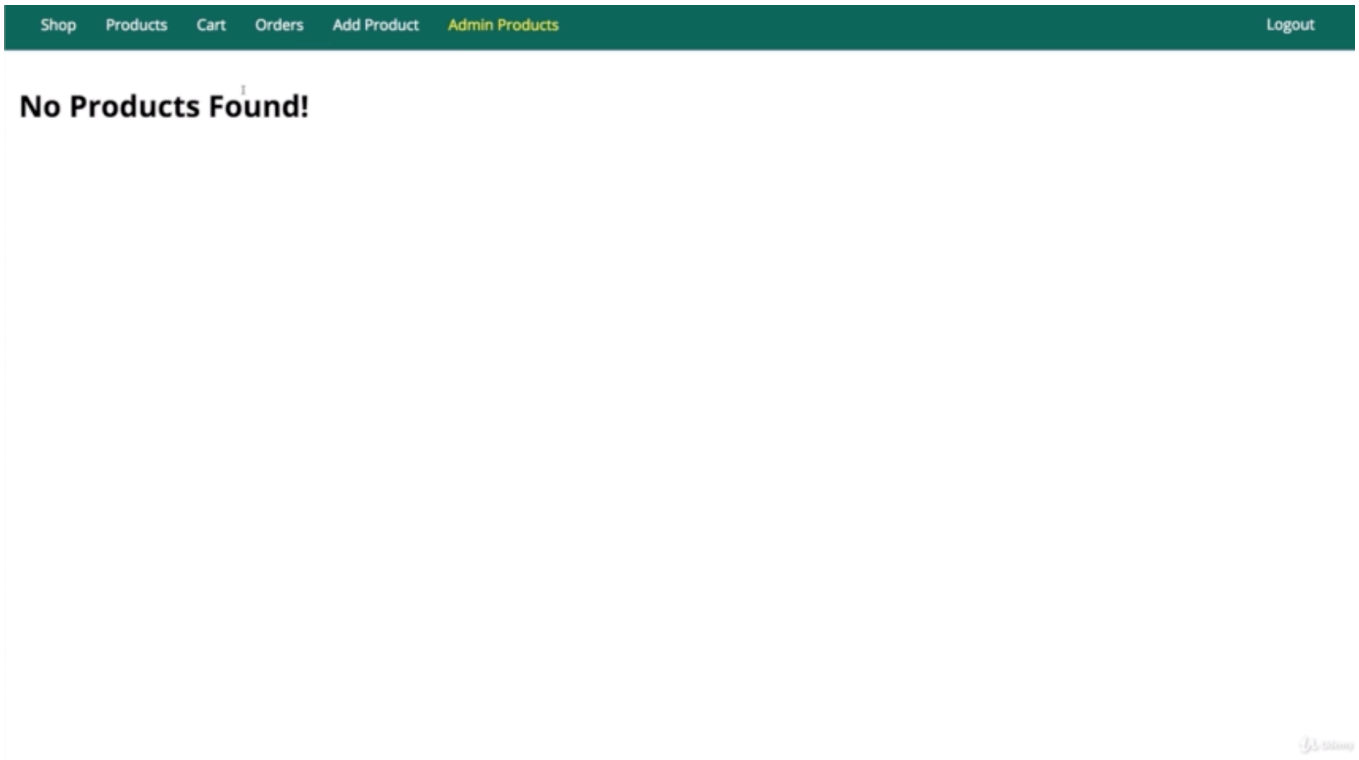
A Book

$ 22

This works!

Edit Delete





- 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.

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

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

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

```

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

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

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

  Product.findById(prodId)
    .then(product => {
      /**i should convert both to a string
       * because i'm also checking for type equality.
```

```
     */
    if(product.userId.toString() !== req.user._id.toString()){
      return res.redirect('/')
    }
    product.title = updatedTitle;
    product.price = updatedPrice;
    product.description = updatedDesc;
    product.imageUrl = updatedImageUrl;
    return product.save()
      .then(result => {
        console.log('UPDATED PRODUCT!');
        res.redirect('/admin/products');
      })
    })
   .catch(err => console.log(err));
};

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

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

```

* Chapter 282: Wrap Up

Module Summary

Password Resetting	Authorization
• 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	• 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