

20. File Upload & Download

* Chapter 314: Module Introduction

* Chapter 315: Adding A File Picker To The Frontend

1. update
- ./views/admin/edit-product.ejs

- Adding file upload means that we have to do 2 things. first thing is that we need to adjust our form to show a file picker to our users. so a tool which they can use to select a file on their operating system on their computer.
- 2nd part is that we need to be able to accept that file in the place where we handle it, where we handle the incoming requests.

- now we have our file picker here which is a default HTML element

```
1 <!--./views/admin/edit-product.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/forms.css">
5   <link rel="stylesheet" href="/css/product.css">
6 </head>
7
8 <body>
9   <%- include('../includes/navigation.ejs') %>
10
11   <main>
12     <% if (errorMessage) { %>
13       <div class="user-message user-message--error"><%= errorMessage %></div>
14     <% } %>
15     <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else
16 { %>add-product<% } %>" method="POST">
17       <div class="form-control">
18         <label for="title">Title</label>
19         <input
20           class="<%= validationErrors.find(e => e.param === 'title') ? 'invalid' :
21           '' %>"
22           type="text"
23           name="title"
24           id="title"
25           value="<% if (editing || hasError) { %><%= product.title %><% } %>">
26       </div>
```

```

25 <!--
26 <div class="form-control">
27   <label for="imageUrl">Image URL</label>
28   <input
29     class="<%= validationErrors.find(e => e.param === 'imageUrl') ?
'invalid' : '' %>"
30     type="text"
31     name="imageUrl"
32     id="imageUrl"
33     value="<%= if (editing || hasError) { %><%= product.imageUrl %><% } %>"
34   </div>
35   -->
36 <div class="form-control">
37   <label for="image">Image</label>
38   <input
39     type="file"
40     name="image"
41     id="image" >
42 </div>
43 <div class="form-control">
44   <label for="price">Price</label>
45   <input
46     class="<%= validationErrors.find(e => e.param === 'price') ? 'invalid' :
'' %>"
47     type="number"
48     name="price"
49     id="price"
50     step="0.01"
51     value="<%= if (editing || hasError) { %><%= product.price %><% } %>"
52 </div>
53 <div class="form-control">
54   <label for="description">Description</label>
55   <textarea
56     class="<%= validationErrors.find(e => e.param === 'description') ?
'invalid' : '' %>"
57     name="description"
58     id="description"
59     rows="5"><%= if (editing || hasError) { %><%= product.description %><% }
%></textarea>
60 </div>
61 <%= if (editing) { %>
62   <input type="hidden" value="<%= product._id %>" name="productId">
63   <% } %>
64
65   <input type="hidden" name="_csrf" value="<%= csrfToken %>">
66   <button class="btn" type="submit"><%= if (editing) { %>Update Product<% } else {
%>Add Product<% } %></button>
67 </form>
68 </main>
69 <%- include('../includes/end.ejs') %>

```

* Chapter 316: Handling Multipart Form Data

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

- ./views/admin/edit-product.ejs

- the error we are getting is stemming from the fact that we are not able to extract our image correctly.

- if i console log, you don't see anything. because we failed to extract an image from the req.body.

- because for extracting the content of our incoming requests, we set up a middleware in app.js, we are using a special middleware, 'body-parser' middleware and this middleware uses or exposes a couple of different parsers and we are using the 'urlencoded' parser.

- urlencoded data is text data. so if a form is submitted without a file, just with text field, no matter if that text field then stores a number, a url or plain text but it's all encoded in text when it's submitted. this format is then called urlencoded.

- we can see that the content type is application and then x-www-form-urlencoded. this means it tries to put all the data as text into its form body.

- we can see that down there, just form data, title, price and so on and just image with nothing. this is invalid. this is an empty text because it can't extract our file as text because a file is binary data.

- because of that, failing extraction, we need to parse our data differently and the body parser that we are using doesn't give us any parser, it doesn't include any parser that handle file data as well. so we need a new package for that.

- 'npm install --save multer'

- multer is another third party package that parses incoming requests. but this package parses incoming requests for files. so it's able to handle file requests or requests with mixed data, with text and file data.

- we will still keep body-parser because we still have like for example, our sign up form where we only submit urlencoded data but now we will have to use a different encoding and that starts with our form.

```
1 // ./controllers/admin.js
2
3 const mongoose = require('mongoose');
4
5 const { validationResult } = require('express-validator/check');
6
7 const Product = require('../models/product');
8
9 exports.getAddProduct = (req, res, next) => {
10   res.render('admin/edit-product', {
11     pageTitle: 'Add Product',
12     path: '/admin/add-product',
13     editing: false,
14     hasError: false,
15     errorMessage: null,
16     validationErrors: []
17   });
18 };
19
20 exports.postAddProduct = (req, res, next) => {
21   const title = req.body.title;
22   const imageUrl = req.body.image;
```

```

23 const price = req.body.price;
24 const description = req.body.description;
25 const errors = validationResult(req);
26
27 if (!errors.isEmpty()) {
28   console.log(errors.array());
29   return res.status(422).render('admin/edit-product', {
30     pageTitle: 'Add Product',
31     path: '/admin/add-product',
32     editing: false,
33     hasError: true,
34     product: {
35       title: title,
36       imageUrl: imageUrl,
37       price: price,
38       description: description
39     },
40     errorMessage: errors.array()[0].msg,
41     validationErrors: errors.array()
42   });
43 }
44
45 const product = new Product({
46   //_id: new mongoose.Types.ObjectId('5badf72403fd8b5be0366e81'),
47   title: title,
48   price: price,
49   description: description,
50   imageUrl: imageUrl,
51   userId: req.user
52 });
53 product
54   .save()
55   .then(result => {
56     // console.log(result);
57     console.log('Created Product');
58     res.redirect('/admin/products');
59   })
60   .catch(err => {
61     // return res.status(500).render('admin/edit-product', {
62     //   pageTitle: 'Add Product',
63     //   path: '/admin/add-product',
64     //   editing: false,
65     //   hasError: true,
66     //   product: {
67     //     title: title,
68     //     imageUrl: imageUrl,
69     //     price: price,
70     //     description: description
71     //   },
72     //   errorMessage: 'Database operation failed, please try again.',
73     //   validationErrors: []
74     // });
75     //res.redirect('/500');
76     const error = new Error(err)
77     error.httpStatusCode = 500
78     return next(error)

```

```

79     });
80 };
81
82 exports.getEditProduct = (req, res, next) => {
83     const editMode = req.query.edit;
84     if (!editMode) {
85         return res.redirect('/');
86     }
87     const prodId = req.params.productId;
88     Product.findById(prodId)
89         .then(product => {
90             if (!product) {
91                 return res.redirect('/');
92             }
93             res.render('admin/edit-product', {
94                 pageTitle: 'Edit Product',
95                 path: '/admin/edit-product',
96                 editing: editMode,
97                 product: product,
98                 hasError: false,
99                 errorMessage: null,
100                 validationErrors: []
101             });
102         })
103         .catch(err => {
104             const error = new Error(err)
105             error.httpStatusCode = 500
106             return next(error)
107         });
108 };
109
110 exports.postEditProduct = (req, res, next) => {
111     const prodId = req.body.productId;
112     const updatedTitle = req.body.title;
113     const updatedPrice = req.body.price;
114     const updatedImageUrl = req.body.imageUrl;
115     const updatedDesc = req.body.description;
116
117     const errors = validationResult(req);
118
119     if (!errors.isEmpty()) {
120         return res.status(422).render('admin/edit-product', {
121             pageTitle: 'Edit Product',
122             path: '/admin/edit-product',
123             editing: true,
124             hasError: true,
125             product: {
126                 title: updatedTitle,
127                 imageUrl: updatedImageUrl,
128                 price: updatedPrice,
129                 description: updatedDesc,
130                 _id: prodId
131             },
132             errorMessage: errors.array()[0].msg,
133             validationErrors: errors.array()
134         });

```

```

135 }
136
137 Product.findById(prodId)
138   .then(product => {
139     if (product.userId.toString() !== req.user._id.toString()) {
140       return res.redirect('/');
141     }
142     product.title = updatedTitle;
143     product.price = updatedPrice;
144     product.description = updatedDesc;
145     product.imageUrl = updatedImageUrl;
146     return product.save().then(result => {
147       console.log('UPDATED PRODUCT!');
148       res.redirect('/admin/products');
149     });
150   })
151   .catch(err => {
152     const error = new Error(err)
153     error.httpStatusCode = 500
154     return next(error)
155   });
156 };
157
158 exports.getProducts = (req, res, next) => {
159   Product.find({ userId: req.user._id })
160     // .select('title price _id')
161     // .populate('userId', 'name')
162     .then(products => {
163       console.log(products);
164       res.render('admin/products', {
165         prods: products,
166         pageTitle: 'Admin Products',
167         path: '/admin/products'
168       });
169     })
170     .catch(err => {
171       const error = new Error(err)
172       error.httpStatusCode = 500
173       return next(error)
174     });
175 };
176
177 exports.postDeleteProduct = (req, res, next) => {
178   const prodId = req.body.productId;
179   Product.deleteOne({ _id: prodId, userId: req.user._id })
180     .then(() => {
181       console.log('DESTROYED PRODUCT');
182       res.redirect('/admin/products');
183     })
184     .catch(err => {
185       const error = new Error(err)
186       error.httpStatusCode = 500
187       return next(error)
188     });
189 };
190

```

```

1 // ./routes/admin.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const { body } = require('express-validator/check')
7
8 const adminController = require('../controllers/admin');
9 const isAuth = require('../middleware/is-auth');
10
11 const router = express.Router();
12
13 // /admin/add-product => GET
14 router.get('/add-product', isAuth, adminController.getAddProduct);
15
16 // /admin/products => GET
17 router.get('/products', isAuth, adminController.getProducts);
18
19 // /admin/add-product => POST
20 router.post('/add-product', [
21     body('title').isString().isLength({ min: 3 }).trim(),
22     body('price').isFloat(),
23     body('description').isLength({ min: 5, max: 400 }).trim()
24 ],
25     isAuth,
26     adminController.postAddProduct);
27
28 router.get('/edit-product/:productId', isAuth, adminController.getEditProduct);
29
30 router.post('/edit-product', [
31     body('title').isString().isLength({ min: 3 }).trim(),
32     body('imageUrl').isURL(),
33     body('price').isFloat(),
34     body('description').isLength({ min: 5, max: 400 }).trim()
35 ],
36     isAuth,
37     adminController.postEditProduct);
38
39 router.post('/delete-product', isAuth, adminController.postDeleteProduct);
40
41 module.exports = router;

```



```

1 <!--./views/admin/edit-product.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4     <link rel="stylesheet" href="/css/forms.css">
5     <link rel="stylesheet" href="/css/product.css">
6 </head>
7
8 <body>
9     <%- include('../includes/navigation.ejs') %>
10
11     <main>
12         <% if (errorMessage) { %>
13             <div class="user-message user-message--error"><%= errorMessage %></div>
14         <% } %>

```

```

15      <!--
16      'enctype' field which i will set to multipart form data application
17      x-www-form-urlencoded as the default
18      but now we will switch to multipart form data which is the content-type
19      telling the server that this submission and this request will not contain
plaintext
20      but will contain mixed data, text and binary data
21      and multer will be looking for incoming requests,
22      with this type of data 'multipart/form-data'
23      and will then be able to parse both the text and our file
24      -->
25      <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else
{ %>add-product<% } %>" method="POST" enctype="multipart/form-data">
26          <div class="form-control">
27              <label for="title">Title</label>
28              <input
29              class="<%= validationErrors.find(e => e.param === 'title') ? 'invalid' :
'' %>"
30              type="text"
31              name="title"
32              id="title"
33              value="<% if (editing || hasError) { %><%= product.title %><% } %>"
34          </div>
35      <!--
36      <div class="form-control">
37          <label for="imageUrl">Image URL</label>
38          <input
39          class="<%= validationErrors.find(e => e.param === 'imageUrl') ?
'invalid' : '' %>"
40          type="text"
41          name="imageUrl"
42          id="imageUrl"
43          value="<% if (editing || hasError) { %><%= product.imageUrl %><% } %>"
44      </div>
45      -->
46      <div class="form-control">
47          <label for="image">Image</label>
48          <input
49          type="file"
50          name="image"
51          id="image" >
52      </div>
53      <div class="form-control">
54          <label for="price">Price</label>
55          <input
56          class="<%= validationErrors.find(e => e.param === 'price') ? 'invalid' :
'' %>"
57          type="number"
58          name="price"
59          id="price"
60          step="0.01"
61          value="<% if (editing || hasError) { %><%= product.price %><% } %>"
62      </div>
63      <div class="form-control">
64          <label for="description">Description</label>
65          <textarea

```



```

66         class="<%= validationErrors.find(e => e.param === 'description') ?
'invalid' : '' %>"
67         name="description"
68         id="description"
69         rows="5"><% if (editing || hasError) { %><%= product.description %><% }
%></textarea>
70     </div>
71     <% if (editing) { %>
72         <input type="hidden" value="<%= product._id %>" name="productId">
73     <% } %>
74
75     <input type="hidden" name="_csrf" value="<%= csrfToken %>">
76     <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else {
%>Add Product<% } %></button>
77 </form>
78 </main>
79 <%- include('../includes/end.ejs') %>

```

* Chapter 317: Handling File Uploads With Multer

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

- what we see is that multer seems to have done something. it seems to have stored something in that file property on our request object.

- it stored the name of the field where it extracted that, it detected the file name, it detected the mimetype and the buffer which is how node handles the binary data. this is the result of the streamed data, the file was sent to our server as a stream or was handled as a stream to handle it if it was bigger and then this is the collected data in a buffer like a bus stop. it gives you a way of working with the stream data in this case it's the combined stream and data. and we work with the buffer to turn it into a file.

- we don't have the buffer because now multer is able to do something with the buffer, instead of just buffering it all in memory. it can turn that buffer back into binary data

- and it stores it in this path here.

- and now if you have a look at your folder, you should not have an images folder with some file in there.

- now that file has some random hash name, doesn't have a file extension and isn't recognized as an image.

- but if i change that and i add '.png' in the end, this is the image i uploaded. so this works.

```

1 //app.js
2
3 const path = require('path');

```

```
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const session = require('express-session');
9 const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf');
11 const flash = require('connect-flash');
12 const multer = require('multer')
13
14 const errorController = require('./controllers/error');
15 const User = require('./models/user');
16
17 const MONGODB_URI =
18 'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop'
19
20 const app = express();
21 const store = new MongoDBStore({
22   uri: MONGODB_URI,
23   collection: 'sessions'
24 });
25 const csrfProtection = csrf();
26
27 app.set('view engine', 'ejs');
28 app.set('views', 'views');
29
30 const adminRoutes = require('./routes/admin');
31 const shopRoutes = require('./routes/shop');
32 const authRoutes = require('./routes/auth');
33
34 /**'multer' is some middleware
35  * which we execute on every incoming requests
36  * and it then has a look at that request
37  * and see if it's multipart form data
38  * and tries to extract file if that is the case.
39  */
40 app.use(bodyParser.urlencoded({ extended: false }));
41 /**i'm choosing 'image' here
42  * because in ./views/admin/edit-product.ejs file,
43  * this input name in file picker is named image
44  *
45  * 'dest: 'images' or 'dest: '/images' will add a product
46  * and
47  */
48 app.use(multer({dest: 'images'}).single('image'))
49 app.use(express.static(path.join(__dirname, 'public')));
50 app.use(
51   session({
52     secret: 'my secret',
53     resave: false,
54     saveUninitialized: false,
55     store: store
56   })
57 );
58 app.use(csrfProtection);
59 app.use(flash());
```

```

60
61 app.use((req, res, next) => {
62   res.locals.isAuthenticated = req.session.isLoggedIn;
63   res.locals.csrfToken = req.csrfToken();
64   next();
65 });
66
67 app.use((req, res, next) => {
68   //throw new Error('Sync Dummy')
69   if (!req.session.user) {
70     return next();
71   }
72   User.findById(req.session.user._id)
73     .then(user => {
74       if (!user) {
75         return next();
76       }
77       req.user = user;
78       next();
79     })
80     .catch(err => {
81       next(new Error(err))
82     });
83 });
84
85 app.use('/admin', adminRoutes);
86 app.use(shopRoutes);
87 app.use(authRoutes);
88
89 app.get('/500', errorController.get500);
90
91 app.use(errorController.get404);
92
93 app.use((error, req, res, next) => {
94   //res.redirect('/500')
95   res.status(500).render('500', {
96     pageTitle: 'Error!',
97     path: '/500',
98     isAuthenticated: req.session.isLoggedIn
99   });
100 })
101
102 mongoose
103   .connect(MONGODB_URI)
104   .then(result => {
105     app.listen(3000);
106   })
107   .catch(err => {
108     console.log(err);
109   });
110
111
112 // ./controllers/admin.js
113
114 const mongoose = require('mongoose');
115
116 const { validationResult } = require('express-validator/check');

```

```

6
7 const Product = require('../models/product');
8
9 exports.getAddProduct = (req, res, next) => {
10   res.render('admin/edit-product', {
11     pageTitle: 'Add Product',
12     path: '/admin/add-product',
13     editing: false,
14     hasError: false,
15     errorMessage: null,
16     validationErrors: []
17   });
18 };
19
20 exports.postAddProduct = (req, res, next) => {
21   const title = req.body.title;
22   const image = req.file;
23   const price = req.body.price;
24   const description = req.body.description;
25   console.log(imageUrl)
26   const errors = validationResult(req);
27
28   if (!errors.isEmpty()) {
29     console.log(errors.array());
30     return res.status(422).render('admin/edit-product', {
31       pageTitle: 'Add Product',
32       path: '/admin/add-product',
33       editing: false,
34       hasError: true,
35       product: {
36         title: title,
37         imageUrl: imageUrl,
38         price: price,
39         description: description
40       },
41       errorMessage: errors.array()[0].msg,
42       validationErrors: errors.array()
43     });
44   }
45
46   const product = new Product({
47     //_id: new mongoose.Types.ObjectId('5badf72403fd8b5be0366e81'),
48     title: title,
49     price: price,
50     description: description,
51     imageUrl: imageUrl,
52     userId: req.user
53   });
54   product
55     .save()
56     .then(result => {
57       // console.log(result);
58       console.log('Created Product');
59       res.redirect('/admin/products');
60     })
61     .catch(err => {

```

```

62 // return res.status(500).render('admin/edit-product', {
63 //   pageTitle: 'Add Product',
64 //   path: '/admin/add-product',
65 //   editing: false,
66 //   hasError: true,
67 //   product: {
68 //     title: title,
69 //     imageUrl: imageUrl,
70 //     price: price,
71 //     description: description
72 //   },
73 //   errorMessage: 'Database operation failed, please try again.',
74 //   validationErrors: []
75 // });
76 //res.redirect('/500');
77 const error = new Error(err)
78 error.httpStatusCode = 500
79 return next(error)
80 });
81 };
82
83 exports.getEditProduct = (req, res, next) => {
84   const editMode = req.query.edit;
85   if (!editMode) {
86     return res.redirect('/');
87   }
88   const prodId = req.params.productId;
89   Product.findById(prodId)
90     .then(product => {
91       if (!product) {
92         return res.redirect('/');
93       }
94       res.render('admin/edit-product', {
95         pageTitle: 'Edit Product',
96         path: '/admin/edit-product',
97         editing: editMode,
98         product: product,
99         hasError: false,
100         errorMessage: null,
101         validationErrors: []
102       });
103     })
104     .catch(err => {
105       const error = new Error(err)
106       error.httpStatusCode = 500
107       return next(error)
108     });
109 };
110
111 exports.postEditProduct = (req, res, next) => {
112   const prodId = req.body.productId;
113   const updatedTitle = req.body.title;
114   const updatedPrice = req.body.price;
115   const updatedImageUrl = req.body.imageUrl;
116   const updatedDesc = req.body.description;
117

```

```

118 const errors = validationResult(req);
119
120 if (!errors.isEmpty()) {
121   return res.status(422).render('admin/edit-product', {
122     pageTitle: 'Edit Product',
123     path: '/admin/edit-product',
124     editing: true,
125     hasError: true,
126     product: {
127       title: updatedTitle,
128       imageUrl: updatedImageUrl,
129       price: updatedPrice,
130       description: updatedDesc,
131       _id: prodId
132     },
133     errorMessage: errors.array()[0].msg,
134     validationErrors: errors.array()
135   });
136 }
137
138 Product.findById(prodId)
139   .then(product => {
140     if (product.userId.toString() !== req.user._id.toString()) {
141       return res.redirect('/');
142     }
143     product.title = updatedTitle;
144     product.price = updatedPrice;
145     product.description = updatedDesc;
146     product.imageUrl = updatedImageUrl;
147     return product.save().then(result => {
148       console.log('UPDATED PRODUCT!');
149       res.redirect('/admin/products');
150     });
151   })
152   .catch(err => {
153     const error = new Error(err)
154     error.httpStatusCode = 500
155     return next(error)
156   });
157 };
158
159 exports.getProducts = (req, res, next) => {
160   Product.find({ userId: req.user._id })
161     // .select('title price _id')
162     // .populate('userId', 'name')
163     .then(products => {
164       console.log(products);
165       res.render('admin/products', {
166         prods: products,
167         pageTitle: 'Admin Products',
168         path: '/admin/products'
169       });
170     })
171     .catch(err => {
172       const error = new Error(err)
173       error.httpStatusCode = 500

```

```

174     return next(error)
175   });
176 };
177
178 exports.postDeleteProduct = (req, res, next) => {
179   const prodId = req.body.productId;
180   Product.deleteOne({ _id: prodId, userId: req.user._id })
181     .then(() => {
182       console.log('DESTROYED PRODUCT');
183       res.redirect('/admin/products');
184     })
185     .catch(err => {
186       const error = new Error(err)
187       error.httpStatusCode = 500
188       return next(error)
189     });
190 };
191

```

* Chapter 318: Configuring Multer To Adjust Filename & Filepath

1. update
- app.js

- you will see that in those images folder, you have undefined-boat.png instead of that random hash because since we did set our own filename with this function. multer doesn't generate that hash.

- 'new Date().toISOString()' gives us a snapshot of the current date and that should also ensure uniqueness

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7  const mongoose = require('mongoose');
8  const session = require('express-session');
9  const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf');
11 const flash = require('connect-flash');
12 const multer = require('multer')
13
14 const errorController = require('./controllers/error');
15 const User = require('./models/user');
16
17 const MONGODB_URI =
18   'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop'
19
20 const app = express();

```

```

21 const store = new MongoDBStore({
22   uri: MONGODB_URI,
23   collection: 'sessions'
24 });
25 const csrfProtection = csrf();
26 /**'idiskStorage()' is a storage engine
27  * which you can use with multer
28  * and pass the javascript object to configure that.
29  *
30  * these are 2 functions
31  * which multer will call for an incoming file
32  * these function then control how that file is handled
33  * regarding the place where you store it and regarding the naming
34  */
35 const fileStorage = multer.diskStorage({
36   destination: (req, file, cb) => {
37     /**the first argument is 'null'
38      * which would be error message you throw to inform mulber
39      * that something is wrong with that incoming file it should not store it
40      * but if that is null, you tell multer that it's OK to store it
41      *
42      * the 2nd argument is the place
43      * where you wanna store.
44      */
45     cb(null, 'images')
46   },
47   /**'filename' is a function
48    *
49    * and we can extract file
50    * and then there is an 'originalname' object
51    * which holds the original file name
52    * there's also is a file name object
53    * which holds this random hash.
54    *
55    * if we have 2 images with the same name,
56    * they don't overwrite each other
57    * so i will use that 'file.filename'
58    * which is auto-generated by multer
59    * and i will concatenate it
60    * with a dash in-between with the originalname
61    * and i also ensure that i end with the extension
62    * and i set my own filename by calling that callback.
63    */
64   filename: (req, file, cb) => {
65     cb(null, new Date().toISOString() + '-' + file.originalname)
66   }
67 })
68
69 app.set('view engine', 'ejs');
70 app.set('views', 'views');
71
72 const adminRoutes = require('./routes/admin');
73 const shopRoutes = require('./routes/shop');
74 const authRoutes = require('./routes/auth');
75
76 app.use(bodyParser.urlencoded({ extended: false }));

```



```

77  /**we can set the 'storage' key
78  * which gives us way more configuration options than just the 'dest' option
79  *
80  * now we need to inform multer
81  * that we wanna use this storage engine
82  * and we do this by setting 'fileStorage' as a value for the storage key in the multer
options
83  */
84  app.use(multer({ storage: fileStorage }).single('image'))
85  app.use(express.static(path.join(__dirname, 'public')));
86  app.use(
87    session({
88      secret: 'my secret',
89      resave: false,
90      saveUninitialized: false,
91      store: store
92    })
93  );
94  app.use(csrfProtection);
95  app.use(flash());
96
97  app.use((req, res, next) => {
98    res.locals.isAuthenticated = req.session.isLoggedIn;
99    res.locals.csrfToken = req.csrfToken();
100    next();
101  });
102
103  app.use((req, res, next) => {
104    //throw new Error('Sync Dummy')
105    if (!req.session.user) {
106      return next();
107    }
108    User.findById(req.session.user._id)
109      .then(user => {
110        if (!user) {
111          return next();
112        }
113        req.user = user;
114        next();
115      })
116      .catch(err => {
117        next(new Error(err))
118      });
119  });
120
121  app.use('/admin', adminRoutes);
122  app.use(shopRoutes);
123  app.use(authRoutes);
124
125  app.get('/500', errorController.get500);
126
127  app.use(errorController.get404);
128
129  app.use((error, req, res, next) => {
130    //res.redirect('/500')
131    res.status(500).render('500', {

```

```

132     pageTitle: 'Error!',
133     path: '/500',
134     isAuthenticated: req.session.isLoggedIn
135   });
136 })
137
138 mongoose
139   .connect(MONGODB_URI)
140   .then(result => {
141     app.listen(3000);
142   })
143   .catch(err => {
144     console.log(err);
145   });
146

```

* Chapter 319: Filtering Files By Mimetype

1. update

- app.js

- if i submit a form with 'boat.png', i succeed, i get an error but i succeed. because i stores the file

- but if i try to submit a different value like pdf file, you can see i only get undefined

- and that undefined is stemming from my ./controllers/admin.js file

```

1  //app.js
2
3  const path = require('path');
4
5  const express = require('express');
6  const bodyParser = require('body-parser');
7  const mongoose = require('mongoose');
8  const session = require('express-session');
9  const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csrf');
11 const flash = require('connect-flash');
12 const multer = require('multer')
13
14 const errorController = require('./controllers/error');
15 const User = require('./models/user');
16
17 const MONGODB_URI =
18   'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop'
19
20 const app = express();
21 const store = new MongoDBStore({

```

```

22   uri: MONGODB_URI,
23   collection: 'sessions'
24 });
25 const csrfProtection = csrf();
26 const fileStorage = multer.diskStorage({
27   destination: (req, file, cb) => {
28     cb(null, 'images')
29   },
30   filename: (req, file, cb) => {
31     cb(null, new Date().toISOString() + '-' + file.originalname)
32   }
33 })
34
35 const fileFilter = (req, file, cb) => {
36   if (
37     file.mimetype === 'image/png' ||
38     file.mimetype === 'image/jpg' ||
39     file.mimetype === 'image/jpeg'
40   ) {
41     /**'null' as an error
42      * and true if we wanna accept that file it should be stored
43      * so false if we don't wanna accept and store that file.
44      */
45     cb(null, true)
46   } else {
47     cb(null, false)
48   }
49 }
50
51 app.set('view engine', 'ejs');
52 app.set('views', 'views');
53
54 const adminRoutes = require('./routes/admin');
55 const shopRoutes = require('./routes/shop');
56 const authRoutes = require('./routes/auth');
57
58 app.use(bodyParser.urlencoded({ extended: false }));
59 app.use(multer({ storage: fileStorage, fileFilter }).single('image'))
60 app.use(express.static(path.join(__dirname, 'public')));
61 app.use(
62   session({
63     secret: 'my secret',
64     resave: false,
65     saveUninitialized: false,
66     store: store
67   })
68 );
69 app.use(csrfProtection);
70 app.use(flash());
71
72 app.use((req, res, next) => {
73   res.locals.isAuthenticated = req.session.isLoggedIn;
74   res.locals.csrfToken = req.csrfToken();
75   next();
76 });
77

```

```

78 app.use((req, res, next) => {
79   //throw new Error('Sync Dummy')
80   if (!req.session.user) {
81     return next();
82   }
83   User.findById(req.session.user._id)
84     .then(user => {
85       if (!user) {
86         return next();
87       }
88       req.user = user;
89       next();
90     })
91     .catch(err => {
92       next(new Error(err))
93     });
94 });
95
96 app.use('/admin', adminRoutes);
97 app.use(shopRoutes);
98 app.use(authRoutes);
99
100 app.get('/500', errorController.get500);
101
102 app.use(errorController.get404);
103
104 app.use((error, req, res, next) => {
105   //res.redirect('/500')
106   res.status(500).render('500', {
107     pageTitle: 'Error!',
108     path: '/500',
109     isAuthenticated: req.session.isLoggedIn
110   });
111 })
112
113 mongoose
114   .connect(MONGODB_URI)
115   .then(result => {
116     app.listen(3000);
117   })
118   .catch(err => {
119     console.log(err);
120   });
121

```

* Chapter 320: Storing File Data In The Database

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

- i'm gonna remove all these products in database.

- first of all, image will be an object of this format with information about the file and where it was stored. so

where the physical file can be found and i wanna save that.

- if i upload an image, i succeed

- but if i upload pdf file, then i get 'attached file is not a image'

- you should not store data like this in the database, file should not be stored in a database. they are too big.

- you need to store the path to the file. that is something you can construct with the information given to you in the file object and you can then pass that data to the database.

- if i save valid image and this works. saving does work.

- if you look into my database for the products, you see this is the part that was stored.

- but if we inspect this, we see the image is not rendered because it can't find this.

- i wanna have a behavior where if we choose 'no file' here, we keep the old one and we only overwrite it with a new file if i choose new one here.

- if i go to my database, that imageUrl didn't change.

- but if i select boat-2.png which is same image but a different file name, different file, i update this, and if we go to database, we see it set boat-2

```
1 // ./controllers/admin.js
2
3 const mongoose = require('mongoose');
4
5 const { validationResult } = require('express-validator/check');
6
7 const Product = require('../models/product');
8
9 exports.getAddProduct = (req, res, next) => {
10   res.render('admin/edit-product', {
11     pageTitle: 'Add Product',
12     path: '/admin/add-product',
13     editing: false,
14     hasError: false,
15     errorMessage: null,
16     validationErrors: []
```

```

17  });
18  };
19
20  exports.postAddProduct = (req, res, next) => {
21    const title = req.body.title;
22    const image = req.file;
23    const price = req.body.price;
24    const description = req.body.description;
25    /**if image is undefined,
26     * then that means that multer declined the incoming file
27     *
28     * if it is undefined,
29     * then i wanna return a response with status 422
30     * because i have an invalid input
31     * so i wanna return that response.
32     */
33    if (!image) {
34      res.status(422).render('admin/edit-product', {
35        pageTitle: 'Add Product',
36        path: '/admin/add-product',
37        editing: false,
38        hasError: true,
39        product: {
40          title: title,
41          price: price,
42          description: description
43        },
44        errorMessage: 'Attached file is not an image',
45        validationErrors: []
46      });
47    }
48    const errors = validationResult(req);
49
50    if (!errors.isEmpty()) {
51      console.log(errors.array());
52      return res.status(422).render('admin/edit-product', {
53        pageTitle: 'Add Product',
54        path: '/admin/add-product',
55        editing: false,
56        hasError: true,
57        product: {
58          title: title,
59          imageUrl: imageUrl,
60          price: price,
61          description: description
62        },
63        errorMessage: errors.array()[0].msg,
64        validationErrors: errors.array()
65      });
66    }
67    /**if we made it after our entire validation,
68     * i have a valid file and valid input data
69     *
70     * i will use my 'image' data which is that file object we get from multer
71     * and there we have information like the file 'path'.
72     * this is the 'path' that a file on my operating system

```

```

73  * so this is the path i wanna use later on when fetching that image.
74  * so this is the path i will store in the imageUrl
75  * and i can store imageUrl in the database again.
76  */
77  const imageUrl = image.path
78
79  const product = new Product({
80    //_id: new mongoose.Types.ObjectId('5badf72403fd8b5be0366e81'),
81    title: title,
82    price: price,
83    description: description,
84    imageUrl: imageUrl,
85    userId: req.user
86  });
87  product
88    .save()
89    .then(result => {
90      // console.log(result);
91      console.log('Created Product');
92      res.redirect('/admin/products');
93    })
94    .catch(err => {
95      // return res.status(500).render('admin/edit-product', {
96      //   pageTitle: 'Add Product',
97      //   path: '/admin/add-product',
98      //   editing: false,
99      //   hasError: true,
100     // product: {
101     //   title: title,
102     //   imageUrl: imageUrl,
103     //   price: price,
104     //   description: description
105     // },
106     // errorMessage: 'Database operation failed, please try again.',
107     // validationErrors: []
108     // });
109     //res.redirect('/500');
110     const error = new Error(err)
111     error.httpStatusCode = 500
112     return next(error)
113   });
114 };
115
116 exports.getEditProduct = (req, res, next) => {
117   const editMode = req.query.edit;
118   if (!editMode) {
119     return res.redirect('/');
120   }
121   const prodId = req.params.productId;
122   Product.findById(prodId)
123     .then(product => {
124       if (!product) {
125         return res.redirect('/');
126       }
127       res.render('admin/edit-product', {
128         pageTitle: 'Edit Product',

```

```

129     path: '/admin/edit-product',
130     editing: editMode,
131     product: product,
132     hasError: false,
133     errorMessage: null,
134     validationErrors: []
135   });
136 })
137 .catch(err => {
138   const error = new Error(err)
139   error.httpStatusCode = 500
140   return next(error)
141 });
142 };
143
144 exports.postEditProduct = (req, res, next) => {
145   const prodId = req.body.productId;
146   const updatedTitle = req.body.title;
147   const updatedPrice = req.body.price;
148   /**if 'image' is undefined,
149    * i know that no file was saved.
150    * and this means i wanna keep the old file,
151    * by the way same will be true
152    * if i upload a pdf.
153    * so we don't need to throw an error message or error
154    *
155    */
156   const image = req.body.imageUrl;
157   const updatedDesc = req.body.description;
158
159   const errors = validationResult(req);
160
161   if (!errors.isEmpty()) {
162     return res.status(422).render('admin/edit-product', {
163       pageTitle: 'Edit Product',
164       path: '/admin/edit-product',
165       editing: true,
166       hasError: true,
167       product: {
168         /**we don't need 'imageUrl: updatedImageUrl'
169          * because we don't render image in page anymore.
170          */
171         title: updatedTitle,
172         price: updatedPrice,
173         description: updatedDesc,
174         _id: prodId
175       },
176       errorMessage: errors.array()[0].msg,
177       validationErrors: errors.array()
178     });
179   }
180
181   Product.findById(prodId)
182     .then(product => {
183       if (product.userId.toString() !== req.user._id.toString()) {
184         return res.redirect('/');

```



```

185     }
186     product.title = updatedTitle;
187     product.price = updatedPrice;
188     product.description = updatedDesc;
189     if (image) {
190         /**this is the same logic as i have it in postAddProduct */
191         product.imageUrl = updatedImageUrl;
192     }
193     return product.save().then(result => {
194         console.log('UPDATED PRODUCT!');
195         res.redirect('/admin/products');
196     });
197 })
198 .catch(err => {
199     const error = new Error(err)
200     error.httpStatusCode = 500
201     return next(error)
202 });
203 };
204
205 exports.getProducts = (req, res, next) => {
206     Product.find({ userId: req.user._id })
207         .select('title price _id')
208         .populate('userId', 'name')
209         .then(products => {
210             console.log(products);
211             res.render('admin/products', {
212                 prods: products,
213                 pageTitle: 'Admin Products',
214                 path: '/admin/products'
215             });
216         })
217         .catch(err => {
218             const error = new Error(err)
219             error.httpStatusCode = 500
220             return next(error)
221         });
222 };
223
224 exports.postDeleteProduct = (req, res, next) => {
225     const prodId = req.body.productId;
226     Product.deleteOne({ _id: prodId, userId: req.user._id })
227         .then(() => {
228             console.log('DESTROYED PRODUCT');
229             res.redirect('/admin/products');
230         })
231         .catch(err => {
232             const error = new Error(err)
233             error.httpStatusCode = 500
234             return next(error)
235         });
236 };
237

```

```

1 // ./routes/admin.js
2
3 const path = require('path');

```

```

4
5 const express = require('express');
6 const { body } = require('express-validator/check')
7
8 const adminController = require('../controllers/admin');
9 const isAuth = require('../middleware/is-auth');
10
11 const router = express.Router();
12
13 // /admin/add-product => GET
14 router.get('/add-product', isAuth, adminController.getAddProduct);
15
16 // /admin/products => GET
17 router.get('/products', isAuth, adminController.getProducts);
18
19 // /admin/add-product => POST
20 router.post('/add-product', [
21     body('title').isString().isLength({ min: 3 }).trim(),
22     body('price').isFloat(),
23     body('description').isLength({ min: 5, max: 400 }).trim()
24 ],
25     isAuth,
26     adminController.postAddProduct);
27
28 router.get('/edit-product/:productId', isAuth, adminController.getEditProduct);
29
30 router.post('/edit-product', [
31     body('title').isString().isLength({ min: 3 }).trim(),
32     body('price').isFloat(),
33     body('description').isLength({ min: 5, max: 400 }).trim()
34 ],
35     isAuth,
36     adminController.postEditProduct);
37
38 router.post('/delete-product', isAuth, adminController.postDeleteProduct);
39
40 module.exports = router;

```

* Chapter 322: Serving Images Statically

1. update
 - app.js
 - ./views/admin/products.ejs
 - ./views/shop/index.ejs
 - ./views/shop/product-list.ejs
 - ./views/shop/product-detail.ejs

- we got multiple options for serving files. option 1 is we serve our images folder in a static way which means that we are serving a public folder with the express.static middleware. and we can serve more than one folder statically.

- 'statically serving a folder' means that requests to files in that folder will be handled automatically and the files will be returned so all the heavy lifting is done behind the scenes by express.

- and now also serve the images folder just like this.

- we point to a folder like 'public' and 'images' and we tell express serve the files from inside that folder as if they were on the root folder.

- so we would copy that URL and open new tab and paste it on the URL.

- and if we remove 'images' in URL, the reason for that is that express assumes that the files in the images folder are served as if they are in the root folder. so '/' slash nothing.

- so if we save and we reload, we see our image here.

- now i wanna do something differently. i have an order and i wanna download an invoice for that order. and that will now work differently because invoices will not be public files that everyone should be able to access. i wanna be able to access my invoice. but no one else should be.

```
1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const session = require('express-session');
9 const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csrf');
11 const flash = require('connect-flash');
12 const multer = require('multer')
13
14 const errorController = require('./controllers/error');
15 const User = require('./models/user');
16
17 const MONGODB_URI =
18 'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop'
19
20 const app = express();
21 const store = new MongoDBStore({
22   uri: MONGODB_URI,
23   collection: 'sessions'
24 });
25 const csrfProtection = csrf();
26 const fileStorage = multer.diskStorage({
27   destination: (req, file, cb) => {
28     cb(null, 'images')
29   },
30   filename: (req, file, cb) => {
31     cb(null, new Date().toISOString() + '-' + file.originalname)
32   }
33 })
34
```

```

35 const fileFilter = (req, file, cb) => {
36   if (
37     file.mimetype === 'image/png' ||
38     file.mimetype === 'image/jpg' ||
39     file.mimetype === 'image/jpeg'
40   ) {
41     cb(null, true)
42   } else {
43     cb(null, false)
44   }
45 }
46
47 app.set('view engine', 'ejs');
48 app.set('views', 'views');
49
50 const adminRoutes = require('./routes/admin');
51 const shopRoutes = require('./routes/shop');
52 const authRoutes = require('./routes/auth');
53
54 app.use(bodyParser.urlencoded({ extended: false }));
55 app.use(multer({ storage: fileStorage, fileFilter }).single('image'))
56 /**we point to a folder like 'public' and 'images'
57  * and we tell express serve the files from inside that folder
58  * as if they were on the root folder.
59  *
60  * and if we remove 'images' in URL,
61  * the reason for that is that express assumes that
62  * the files in the images folder are served
63  * as if they are in the root folder. so '/' slash nothing.
64  *
65  * so we can adjust our middleware
66  * and if we have a request that goes to '/images'
67  * that start with '/images'
68  * then serve these files statically
69  * and now '/images' is the folder we assume for this static serving
70  */
71 app.use(express.static(path.join(__dirname, 'public')));
72 app.use('/images', express.static(path.join(__dirname, 'images')));
73 app.use(
74   session({
75     secret: 'my secret',
76     resave: false,
77     saveUninitialized: false,
78     store: store
79   })
80 );
81 app.use(csrfProtection);
82 app.use(flash());
83
84 app.use((req, res, next) => {
85   res.locals.isAuthenticated = req.session.isLoggedIn;
86   res.locals.csrfToken = req.csrfToken();
87   next();
88 });
89
90 app.use((req, res, next) => {

```

```

91 //throw new Error('Sync Dummy')
92 if (!req.session.user) {
93   return next();
94 }
95 User.findById(req.session.user._id)
96   .then(user => {
97     if (!user) {
98       return next();
99     }
100     req.user = user;
101     next();
102   })
103   .catch(err => {
104     next(new Error(err))
105   });
106 });
107
108 app.use('/admin', adminRoutes);
109 app.use(shopRoutes);
110 app.use(authRoutes);
111
112 app.get('/500', errorController.get500);
113
114 app.use(errorController.get404);
115
116 app.use((error, req, res, next) => {
117   //res.redirect('/500')
118   res.status(500).render('500', {
119     pageTitle: 'Error!',
120     path: '/500',
121     isAuthenticated: req.session.isLoggedIn
122   });
123 })
124
125 mongoose
126   .connect(MONGODB_URI)
127   .then(result => {
128     app.listen(3000);
129   })
130   .catch(err => {
131     console.log(err);
132   });
133

```

```

1 <!--./views/shop/index.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/product.css">
5 </head>
6
7 <body>
8   <%- include('../includes/navigation.ejs') %>
9
10   <main>
11     <% if (prods.length > 0) { %>
12       <div class="grid">
13         <% for (let product of prods) { %>

```

```

14 <article class="card product-item">
15   <header class="card_header">
16     <h1 class="product__title"><%= product.title %></h1>
17   </header>
18   <div class="card__image">
19     "
21   </div>
22   <div class="card__content">
23     <h2 class="product__price">$<%= product.price %></h2>
24     <p class="product__description"><%= product.description %></p>
25   </div>
26   <div class="card__actions">
27     <a href="/products/<%= product._id %>" class="btn">Details</a>
28     <% if (isAuthenticated) { %>
29       <%= include('../includes/add-to-cart.ejs', {product:
product}}) %>
30     <% } %>
31   </div>
32 </article>
33 <% } %>
34 </div>
35 <% } else { %>
36   <h1>No Products Found!</h1>
37 <% } %>
38 </main>
39 <%= include('../includes/end.ejs') %>

```

```

1 <!--./views/shop/product-list.ejs-->
2
3 <%= include('../includes/head.ejs') %>
4 <link rel="stylesheet" href="/css/product.css">
5 </head>
6
7 <body>
8   <%= include('../includes/navigation.ejs') %>
9
10  <main>
11    <% if (prods.length > 0) { %>
12      <div class="grid">
13        <% for (let product of prods) { %>
14          <article class="card product-item">
15            <header class="card_header">
16              <h1 class="product__title">
17                <%= product.title %>
18              </h1>
19            </header>
20            <div class="card__image">
21              "
22            </div>
23            <div class="card__content">
24              <h2 class="product__price">$
25                <%= product.price %>
26              </h2>
27              <p class="product__description">
28                <%= product.description %>

```

```

29         </p>
30     </div>
31     <div class="card_actions">
32         <a href="/products/<%= product._id %>"
class="btn">Details</a>
33         <% if (isAuthenticated) { %>
34             <%- include('../includes/add-to-cart.ejs', {product:
product}) %>
35         <% } %>
36     </div>
37 </article>
38 <% } %>
39 </div>
40 <% } else { %>
41     <h1>No Products Found!</h1>
42     <% } %>
43 </main>
44 <%- include('../includes/end.ejs') %>

```

```

1 <!--./views/shop/product-detail.ejs-->
2
3 <%= include('../includes/head.ejs') %>
4   </head>
5
6   <body>
7     <%= include('../includes/navigation.ejs') %>
8     <main class="centered">
9       <h1><%= product.title %></h1>
10      <hr>
11      <div class="image">
12        ">
13      </div>
14      <h2><%= product.price %></h2>
15      <p><%= product.description %></p>
16      <% if (isAuthenticated) { %>
17        <%= include('../includes/add-to-cart.ejs', {product: product}) %>
18      <% } %>
19    </main>
20    <%= include('../includes/end.ejs') %>

```

```

1 <!--./views/admin/products.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/product.css">
5   </head>
6
7   <body>
8     <%- include('../includes/navigation.ejs') %>
9
10    <main>
11      <% if (prods.length > 0) { %>
12        <div class="grid">
13          <% for (let product of prods) { %>
14            <article class="card product-item">
15              <header class="card_header">
16                <h1 class="product_title">
17                  <%= product.title %>

```

```

18         </h1>
19     </header>
20     <div class="card_image">
21         <!--
22         we need to add a slash at the beginning
23         which will turn this into an absolute path
24         it will not append it to the current path
25         but rather create a new path with only our domain
26         and then the path which get rendered here
27         and i do this in products.
28         -->
29         "
30     </div>
31     <div class="card_content">
32         <h2 class="product_price">$
33         <%= product.price %>
34     </h2>
35     <p class="product_description">
36         <%= product.description %>
37     </p>
38 </div>
39 <div class="card_actions">
40     <a href="/admin/edit-product/<%= product._id %>?
edit=true" class="btn">Edit</a>
41     <form action="/admin/delete-product" method="POST">
42         <input type="hidden" value="<%= product._id %>"
name="productId">
43         <input type="hidden" name="_csrf" value="<%=
csrfToken %>">
44         <button class="btn" type="submit">Delete</button>
45     </form>
46
47 </div>
48 </article>
49 <% } %>
50 </div>
51 <% } else { %>
52     <h1>No Products Found!</h1>
53     <% } %>
54 </main>
55 <%- include('../includes/end.ejs') %>

```

* Chapter 323: Downloading Files With Authentication

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

- grab that latest object's `_id` in orders collection, and paste it to the invoice.pdf title name like 'invoice-5cce2e256bcaa07fd3631fo.pdf'

- if i click 'invoice' anchor tag, and i get my download option here, let me save that and try to open that file. it should open as a PDF or you should be able to open it as a PDF but it was not the most convenient way of downloading this.

```
1 <!--./views/shop/orders.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/orders.css">
5   </head>
6
7   <body>
8     <%- include('../includes/navigation.ejs') %>
9     <main>
10      <% if (orders.length <= 0) { %>
11        <h1>Nothing there!</h1>
12      <% } else { %>
13        <ul class="orders">
14          <% orders.forEach(order => { %>
15            <li class="orders__item">
16              <h1>Order - # <%= order._id %></h1>
17              <ul class="orders__products">
18                <% order.products.forEach(p => { %>
19                  <li class="orders__products-item"><%= p.product.title %>
20                    (<%= p.quantity %>) - <a href="/orders/<%= order._id %>">Invoice</a></li>
21                <% }); %>
22              </ul>
23            </li>
24          <% }); %>
25        </ul>
26      <% } %>
27    </main>
28    <%- include('../includes/end.ejs') %>
```

```
1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8 const isAuth = require('../middleware/is-auth');
9
10 const router = express.Router();
11
12 router.get('/', shopController.getIndex);
13
14 router.get('/products', shopController.getProducts);
15
16 router.get('/products/:productId', shopController.getProduct);
17
18 router.get('/cart', isAuth, shopController.getCart);
```

```

19
20 router.post('/cart', isAuth, shopController.postCart);
21
22 router.post('/cart-delete-item', isAuth, shopController.postCartDeleteProduct);
23
24 router.post('/create-order', isAuth, shopController.postOrder);
25
26 router.get('/orders', isAuth, shopController.getOrders);
27
28 router.get('/orders/:orderId', isAuth, shopController.getInvoice)
29
30 module.exports = router;

```

```

1 //./controllers/shop.js
2
3 const fs = require('fs')
4 const path = require('path')
5
6 const Product = require('../models/product');
7 const Order = require('../models/order');
8
9 exports.getProducts = (req, res, next) => {
10   Product.find()
11     .then(products => {
12       console.log(products);
13       res.render('shop/product-list', {
14         prods: products,
15         pageTitle: 'All Products',
16         path: '/products'
17       });
18     })
19     .catch(err => {
20       const error = new Error(err)
21       error.httpStatusCode = 500
22       return next(error)
23     });
24 };
25
26 exports.getProduct = (req, res, next) => {
27   const prodId = req.params.productId;
28   Product.findById(prodId)
29     .then(product => {
30       res.render('shop/product-detail', {
31         product: product,
32         pageTitle: product.title,
33         path: '/products'
34       });
35     })
36     .catch(err => {
37       const error = new Error(err)
38       error.httpStatusCode = 500
39       return next(error)
40     });
41 };
42
43 exports.getIndex = (req, res, next) => {
44   Product.find()

```

```

45     .then(products => {
46         res.render('shop/index', {
47             prods: products,
48             pageTitle: 'Shop',
49             path: '/'
50         });
51     })
52     .catch(err => {
53         const error = new Error(err)
54         error.httpStatusCode = 500
55         return next(error)
56     });
57 };
58
59 exports.getCart = (req, res, next) => {
60     req.user
61         .populate('cart.items.productId')
62         .execPopulate()
63         .then(user => {
64             const products = user.cart.items;
65             res.render('shop/cart', {
66                 path: '/cart',
67                 pageTitle: 'Your Cart',
68                 products: products
69             });
70         })
71         .catch(err => {
72             const error = new Error(err)
73             error.httpStatusCode = 500
74             return next(error)
75         });
76 };
77
78 exports.postCart = (req, res, next) => {
79     const prodId = req.body.productId;
80     Product.findById(prodId)
81         .then(product => {
82             return req.user.addToCart(product);
83         })
84         .then(result => {
85             console.log(result);
86             res.redirect('/cart');
87         })
88         .catch(err => {
89             const error = new Error(err)
90             error.httpStatusCode = 500
91             return next(error)
92         })
93 };
94
95 exports.postCartDeleteProduct = (req, res, next) => {
96     const prodId = req.body.productId;
97     req.user
98         .removeFromCart(prodId)
99         .then(result => {
100             res.redirect('/cart');

```

```

101     })
102     .catch(err => {
103         const error = new Error(err)
104         error.httpStatusCode = 500
105         return next(error)
106     });
107 };
108
109 exports.postOrder = (req, res, next) => {
110     req.user
111         .populate('cart.items.productId')
112         .execPopulate()
113         .then(user => {
114             const products = user.cart.items.map(i => {
115                 return { quantity: i.quantity, product: { ...i.productId._doc } };
116             });
117             const order = new Order({
118                 user: {
119                     email: req.user.email,
120                     userId: req.user
121                 },
122                 products: products
123             });
124             return order.save();
125         })
126         .then(result => {
127             return req.user.clearCart();
128         })
129         .then(() => {
130             res.redirect('/orders');
131         })
132         .catch(err => {
133             const error = new Error(err)
134             error.httpStatusCode = 500
135             return next(error)
136         });
137 };
138
139 exports.getOrders = (req, res, next) => {
140     Order.find({ 'user.userId': req.user._id })
141         .then(orders => {
142             res.render('shop/orders', {
143                 path: '/orders',
144                 pageTitle: 'Your Orders',
145                 orders: orders
146             });
147         })
148         .catch(err => {
149             const error = new Error(err)
150             error.httpStatusCode = 500
151             return next(error)
152         });
153 };
154
155 exports.getInvoice = (req, res, next) => {
156     /**request and encoded in the url

```

```

157 * so it's params and then orderId
158 *
159 * that is specified in ./routes/shop.js
160 *
161 * and now we need to retrieve that file
162 * and we can retrieve files with nodes file system.
163 * now the path should be constructed with the path core module
164 * so that it works on all operating system
165 */
166 const orderId = req.params.orderId
167 const invoiceName = 'invoice-' + orderId + '.pdf'
168 const invoicePath = path.join('data', 'invoices', invoiceName)
169 fs.readFile(invoicePath, (err, data) => {
170   if (err) {
171     return next(err)
172   }
173   res.send(data)
174 })
175 }

```

* Chapter 324: Setting File Type Headers

1. update
- ./controllers/shop.js

- if i click this 'invoice' button, now i open it in the browser. so this already changes the behavior. it already gives the browser some information which allows the browser to handle this in a better way.

- and if i save it and click 'invoice' button, the download menu opens again. and we have the proper file name with the proper extension. this is how you can control how the browser should handle the incoming data.

- now i have a set up where only authenticated users can requests this invoice. we can still improve that.

```

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

```

```

14     prods: products,
15     pageTitle: 'All Products',
16     path: '/products'
17   });
18 }
19 .catch(err => {
20   const error = new Error(err)
21   error.httpStatusCode = 500
22   return next(error)
23 });
24 };
25
26 exports.getProduct = (req, res, next) => {
27   const prodId = req.params.productId;
28   Product.findById(prodId)
29     .then(product => {
30     res.render('shop/product-detail', {
31       product: product,
32       pageTitle: product.title,
33       path: '/products'
34     });
35   })
36   .catch(err => {
37     const error = new Error(err)
38     error.httpStatusCode = 500
39     return next(error)
40   });
41 };
42
43 exports.getIndex = (req, res, next) => {
44   Product.find()
45     .then(products => {
46     res.render('shop/index', {
47       prods: products,
48       pageTitle: 'Shop',
49       path: '/'
50     });
51   })
52   .catch(err => {
53     const error = new Error(err)
54     error.httpStatusCode = 500
55     return next(error)
56   });
57 };
58
59 exports.getCart = (req, res, next) => {
60   req.user
61     .populate('cart.items.productId')
62     .execPopulate()
63     .then(user => {
64     const products = user.cart.items;
65     res.render('shop/cart', {
66       path: '/cart',
67       pageTitle: 'Your Cart',
68       products: products
69     });

```

```

70     })
71     .catch(err => {
72         const error = new Error(err)
73         error.httpStatusCode = 500
74         return next(error)
75     });
76 };
77
78 exports.postCart = (req, res, next) => {
79     const prodId = req.body.productId;
80     Product.findById(prodId)
81     .then(product => {
82         return req.user.addToCart(product);
83     })
84     .then(result => {
85         console.log(result);
86         res.redirect('/cart');
87     })
88     .catch(err => {
89         const error = new Error(err)
90         error.httpStatusCode = 500
91         return next(error)
92     })
93 };
94
95 exports.postCartDeleteProduct = (req, res, next) => {
96     const prodId = req.body.productId;
97     req.user
98     .removeFromCart(prodId)
99     .then(result => {
100         res.redirect('/cart');
101     })
102     .catch(err => {
103         const error = new Error(err)
104         error.httpStatusCode = 500
105         return next(error)
106     });
107 };
108
109 exports.postOrder = (req, res, next) => {
110     req.user
111     .populate('cart.items.productId')
112     .execPopulate()
113     .then(user => {
114         const products = user.cart.items.map(i => {
115             return { quantity: i.quantity, product: { ...i.productId._doc } };
116         });
117         const order = new Order({
118             user: {
119                 email: req.user.email,
120                 userId: req.user
121             },
122             products: products
123         });
124         return order.save();
125     })

```

```

126     .then(result => {
127         return req.user.clearCart();
128     })
129     .then(() => {
130         res.redirect('/orders');
131     })
132     .catch(err => {
133         const error = new Error(err)
134         error.httpStatusCode = 500
135         return next(error)
136     });
137 };
138
139 exports.getOrders = (req, res, next) => {
140     Order.find({ 'user.userId': req.user._id })
141         .then(orders => {
142             res.render('shop/orders', {
143                 path: '/orders',
144                 pageTitle: 'Your Orders',
145                 orders: orders
146             });
147         })
148         .catch(err => {
149             const error = new Error(err)
150             error.httpStatusCode = 500
151             return next(error)
152         });
153 };
154
155 exports.getInvoice = (req, res, next) => {
156     const orderId = req.params.orderId
157     const invoiceName = 'invoice-' + orderId + '.pdf'
158     const invoicePath = path.join('data', 'invoices', invoiceName)
159     fs.readFile(invoicePath, (err, data) => {
160         if (err) {
161             return next(err)
162         }
163         res.setHeader('Content-Type', 'application/pdf')
164         /*'Content-Disposition' allows us to define how this content should be served to the
165         * and we can set this to 'inline' to tell the browser to open it inline.
166         *
167         * and we can also add filename
168         */
169         res.setHeader('Content-Disposition', 'inline; filename="' + invoiceName + '"')
170         res.send(data)
171     })
172 }

```

* Chapter 325: Restricting File Access

1. update
- ./controllers/shop.js

- it works

- and if i change 3 here to 2 in the URL, then i get my error because it's an invalid URL

- and if i logout and try to access this and access the original URL, i can't see that.

- if i log in with the different user, and i try to access the original order before, i can't access that. i get error.

```
1  ../controllers/shop.js
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const Product = require('../models/product');
7  const Order = require('../models/order');
8
9  exports.getProducts = (req, res, next) => {
10   Product.find()
11     .then(products => {
12       console.log(products);
13       res.render('shop/product-list', {
14         prods: products,
15         pageTitle: 'All Products',
16         path: '/products'
17       });
18     })
19     .catch(err => {
20       const error = new Error(err);
21       error.httpStatusCode = 500;
22       return next(error);
23     });
24 };
25
26 exports.getProduct = (req, res, next) => {
27   const prodId = req.params.productId;
28   Product.findById(prodId)
29     .then(product => {
30       res.render('shop/product-detail', {
31         product: product,
32         pageTitle: product.title,
33         path: '/products'
34       });
35     })
36     .catch(err => {
37       const error = new Error(err);
38       error.httpStatusCode = 500;
39       return next(error);
40     });
41 };
42
43 exports.getIndex = (req, res, next) => {
```

```

44 Product.find()
45   .then(products => {
46     res.render('shop/index', {
47       prods: products,
48       pageTitle: 'Shop',
49       path: '/'
50     });
51   })
52   .catch(err => {
53     const error = new Error(err);
54     error.httpStatusCode = 500;
55     return next(error);
56   });
57 };
58
59 exports.getCart = (req, res, next) => {
60   req.user
61     .populate('cart.items.productId')
62     .execPopulate()
63     .then(user => {
64       const products = user.cart.items;
65       res.render('shop/cart', {
66         path: '/cart',
67         pageTitle: 'Your Cart',
68         products: products
69       });
70     })
71     .catch(err => {
72       const error = new Error(err);
73       error.httpStatusCode = 500;
74       return next(error);
75     });
76 };
77
78 exports.postCart = (req, res, next) => {
79   const prodId = req.body.productId;
80   Product.findById(prodId)
81     .then(product => {
82       return req.user.addToCart(product);
83     })
84     .then(result => {
85       console.log(result);
86       res.redirect('/cart');
87     })
88     .catch(err => {
89       const error = new Error(err);
90       error.httpStatusCode = 500;
91       return next(error);
92     });
93 };
94
95 exports.postCartDeleteProduct = (req, res, next) => {
96   const prodId = req.body.productId;
97   req.user
98     .removeFromCart(prodId)
99     .then(result => {

```

```

100     res.redirect('/cart');
101   })
102   .catch(err => {
103     const error = new Error(err);
104     error.httpStatusCode = 500;
105     return next(error);
106   });
107 };
108
109 exports.postOrder = (req, res, next) => {
110   req.user
111     .populate('cart.items.productId')
112     .execPopulate()
113     .then(user => {
114       const products = user.cart.items.map(i => {
115         return { quantity: i.quantity, product: { ...i.productId._doc } };
116       });
117       const order = new Order({
118         user: {
119           email: req.user.email,
120           userId: req.user
121         },
122         products: products
123       });
124       return order.save();
125     })
126     .then(result => {
127       return req.user.clearCart();
128     })
129     .then(() => {
130       res.redirect('/orders');
131     })
132     .catch(err => {
133       const error = new Error(err);
134       error.httpStatusCode = 500;
135       return next(error);
136     });
137 };
138
139 exports.getOrders = (req, res, next) => {
140   Order.find({ 'user.userId': req.user._id })
141     .then(orders => {
142       res.render('shop/orders', {
143         path: '/orders',
144         pageTitle: 'Your Orders',
145         orders: orders
146       });
147     })
148     .catch(err => {
149       const error = new Error(err);
150       error.httpStatusCode = 500;
151       return next(error);
152     });
153 };
154
155 exports.getInvoice = (req, res, next) => {

```

```

156   const orderId = req.params.orderId;
157   Order.findById(orderId)
158     .then(order => {
159       if (!order) {
160         return next(new Error('No order found.));
161       }
162       if (order.user.userId.toString() !== req.user._id.toString()) {
163         return next(new Error('Unauthorized'));
164       }
165       const invoiceName = 'invoice-' + orderId + '.pdf';
166       const invoicePath = path.join('data', 'invoices', invoiceName);
167       fs.readFile(invoicePath, (err, data) => {
168         if (err) {
169           return next(err);
170         }
171         res.setHeader('Content-Type', 'application/pdf');
172         res.setHeader(
173           'Content-Disposition',
174           'inline; filename="' + invoiceName + '"'
175         );
176         res.send(data);
177       });
178     })
179     .catch(err => next(err));
180   });
181

```

* Chapter 326: Streaming Data Vs Preloading Data

1. update
- ./controllers/shop.js

- this is streamend data created with that 'createReadStream()' which is the recommended way of getting your file data for bigger files.

```

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

```

```

17     });
18   })
19   .catch(err => {
20     const error = new Error(err);
21     error.httpStatusCode = 500;
22     return next(error);
23   });
24 };
25
26 exports.getProduct = (req, res, next) => {
27   const prodId = req.params.productId;
28   Product.findById(prodId)
29     .then(product => {
30       res.render('shop/product-detail', {
31         product: product,
32         pageTitle: product.title,
33         path: '/products'
34       });
35     })
36     .catch(err => {
37       const error = new Error(err);
38       error.httpStatusCode = 500;
39       return next(error);
40     });
41 };
42
43 exports.getIndex = (req, res, next) => {
44   Product.find()
45     .then(products => {
46       res.render('shop/index', {
47         prods: products,
48         pageTitle: 'Shop',
49         path: '/'
50       });
51     })
52     .catch(err => {
53       const error = new Error(err);
54       error.httpStatusCode = 500;
55       return next(error);
56     });
57 };
58
59 exports.getCart = (req, res, next) => {
60   req.user
61     .populate('cart.items.productId')
62     .execPopulate()
63     .then(user => {
64       const products = user.cart.items;
65       res.render('shop/cart', {
66         path: '/cart',
67         pageTitle: 'Your Cart',
68         products: products
69       });
70     })
71     .catch(err => {
72       const error = new Error(err);

```

```

73     error.httpStatusCode = 500;
74     return next(error);
75 });
76 };
77
78 exports.postCart = (req, res, next) => {
79     const prodId = req.body.productId;
80     Product.findById(prodId)
81         .then(product => {
82             return req.user.addToCart(product);
83         })
84         .then(result => {
85             console.log(result);
86             res.redirect('/cart');
87         })
88         .catch(err => {
89             const error = new Error(err);
90             error.httpStatusCode = 500;
91             return next(error);
92         });
93 };
94
95 exports.postCartDeleteProduct = (req, res, next) => {
96     const prodId = req.body.productId;
97     req.user
98         .removeFromCart(prodId)
99         .then(result => {
100             res.redirect('/cart');
101         })
102         .catch(err => {
103             const error = new Error(err);
104             error.httpStatusCode = 500;
105             return next(error);
106         });
107 };
108
109 exports.postOrder = (req, res, next) => {
110     req.user
111         .populate('cart.items.productId')
112         .execPopulate()
113         .then(user => {
114             const products = user.cart.items.map(i => {
115                 return { quantity: i.quantity, product: { ...i.productId._doc } };
116             });
117             const order = new Order({
118                 user: {
119                     email: req.user.email,
120                     userId: req.user
121                 },
122                 products: products
123             });
124             return order.save();
125         })
126         .then(result => {
127             return req.user.clearCart();
128         })

```

```

129     .then(() => {
130         res.redirect('/orders');
131     })
132     .catch(err => {
133         const error = new Error(err);
134         error.httpStatusCode = 500;
135         return next(error);
136     });
137 };
138
139 exports.getOrders = (req, res, next) => {
140     Order.find({ 'user.userId': req.user._id })
141         .then(orders => {
142             res.render('shop/orders', {
143                 path: '/orders',
144                 pageTitle: 'Your Orders',
145                 orders: orders
146             });
147         })
148         .catch(err => {
149             const error = new Error(err);
150             error.httpStatusCode = 500;
151             return next(error);
152         });
153 };
154
155 exports.getInvoice = (req, res, next) => {
156     const orderId = req.params.orderId;
157     Order.findById(orderId)
158         .then(order => {
159             if (!order) {
160                 return next(new Error('No order found.'));
161             }
162             if (order.user.userId.toString() !== req.user._id.toString()) {
163                 return next(new Error('Unauthorized'));
164             }
165             const invoiceName = 'invoice-' + orderId + '.pdf';
166             const invoicePath = path.join('data', 'invoices', invoiceName);
167             /**if you read a file like this,
168              * node will first of all access that file,
169              * read the entire content into memory
170              * and then return it with the response.
171              *
172              * this means that for bigger files,
173              * this will take very long before a response is sent
174              * and your memory on the server might overflow at some point for many incoming
requests
175              * because it has to read all the data into memory which is limited.
176              *
177              * instead you should be streaming your response data
178              */
179
180             //fs.readFile(invoicePath, (err, data) => {
181             //    if (err) {
182             //        return next(err);
183             //    }

```

```

184 // res.setHeader('Content-Type', 'application/pdf');
185 // res.setHeader(
186 //   'Content-Disposition',
187 //   'inline; filename="' + invoiceName + '"'
188 // );
189 // res.send(data);
190 //});
191
192 const file = fs.createReadStream(invoicePath)
193   res.setHeader('Content-Type', 'application/pdf');
194   res.setHeader(
195     'Content-Disposition',
196     'inline; filename="' + invoiceName + '"'
197   )
198   /**and call the 'pipe()' method to forward the data
199    * that is read in with that stream to my response
200    * because the response object is a writable stream.
201    * and you can use readable streams to pipe their output into a writable stream.
202    *
203    * we can pipe our readable stream
204    * the file stream into the response
205    * and that means that the response will be streamed to the browser
206    * and will contain the data
207    * and the data will be downloaded by the browser step by step.
208    *
209    * this is huge advantage for big file
210    * because node never has to pre-load all the data into memory
211    * but just streams it to the the client on the fly.
212    * and the most it has to store is one chunk of data.
213    *
214    * the chunks are what we work with
215    * the buffers gives us access to these chunks.
216    *
217    * we don't wait for all the chunks to come together
218    * and concatenate them into one object,
219    * instead we forward them to the browser
220    * which is able to concatenate the incoming data pieces into the final file.
221    */
222   file.pipe(res)
223 }
224 .catch(err => next(err));
225 };
226

```

* Chapter 327: Using PDFKit For .pdf Generation

1. update
- ./controllers/shop.js

- delete my invoice.pdf file in ./invoice folder. but instead when we get the invoice for an order, i don't wanna serve a file that already exists. but i wanna generate that file based on the real order data.

- 'PDFKit' is the 3rd party package which we can use and is very prominent or popular package for creating pdfs on a node.js server.


```
1  //./controllers/shop.js
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const PDFDocument = require('pdfkit')
7
8  const Product = require('../models/product');
9  const Order = require('../models/order');
10
11 exports.getProducts = (req, res, next) => {
12   Product.find()
13     .then(products => {
14       console.log(products);
15       res.render('shop/product-list', {
16         prods: products,
17         pageTitle: 'All Products',
18         path: '/products'
19       });
20     })
21     .catch(err => {
22       const error = new Error(err);
23       error.httpStatusCode = 500;
24       return next(error);
25     });
26 };
27
28 exports.getProduct = (req, res, next) => {
29   const prodId = req.params.productId;
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
38     .catch(err => {
39       const error = new Error(err);
40       error.httpStatusCode = 500;
41       return next(error);
42     });
43 };
44
45 exports.getIndex = (req, res, next) => {
46   Product.find()
47     .then(products => {
48       res.render('shop/index', {
```

```

49     prods: products,
50     pageTitle: 'Shop',
51     path: '/'
52   });
53 }
54 .catch(err => {
55   const error = new Error(err);
56   error.httpStatusCode = 500;
57   return next(error);
58 });
59 };
60
61 exports.getCart = (req, res, next) => {
62   req.user
63     .populate('cart.items.productId')
64     .execPopulate()
65     .then(user => {
66       const products = user.cart.items;
67       res.render('shop/cart', {
68         path: '/cart',
69         pageTitle: 'Your Cart',
70         products: products
71       });
72     })
73     .catch(err => {
74       const error = new Error(err);
75       error.httpStatusCode = 500;
76       return next(error);
77     });
78 };
79
80 exports.postCart = (req, res, next) => {
81   const prodId = req.body.productId;
82   Product.findById(prodId)
83     .then(product => {
84       return req.user.addToCart(product);
85     })
86     .then(result => {
87       console.log(result);
88       res.redirect('/cart');
89     })
90     .catch(err => {
91       const error = new Error(err);
92       error.httpStatusCode = 500;
93       return next(error);
94     });
95 };
96
97 exports.postCartDeleteProduct = (req, res, next) => {
98   const prodId = req.body.productId;
99   req.user
100     .removeFromCart(prodId)
101     .then(result => {
102       res.redirect('/cart');
103     })
104     .catch(err => {

```

```

105     const error = new Error(err);
106     error.httpStatusCode = 500;
107     return next(error);
108   });
109 };
110
111 exports.postOrder = (req, res, next) => {
112   req.user
113     .populate('cart.items.productId')
114     .execPopulate()
115     .then(user => {
116       const products = user.cart.items.map(i => {
117         return { quantity: i.quantity, product: { ...i.productId._doc } };
118       });
119       const order = new Order({
120         user: {
121           email: req.user.email,
122           userId: req.user
123         },
124         products: products
125       });
126       return order.save();
127     })
128     .then(result => {
129       return req.user.clearCart();
130     })
131     .then(() => {
132       res.redirect('/orders');
133     })
134     .catch(err => {
135       const error = new Error(err);
136       error.httpStatusCode = 500;
137       return next(error);
138     });
139 };
140
141 exports.getOrders = (req, res, next) => {
142   Order.find({ 'user.userId': req.user._id })
143     .then(orders => {
144       res.render('shop/orders', {
145         path: '/orders',
146         pageTitle: 'Your Orders',
147         orders: orders
148       });
149     })
150     .catch(err => {
151       const error = new Error(err);
152       error.httpStatusCode = 500;
153       return next(error);
154     });
155 };
156
157 exports.getInvoice = (req, res, next) => {
158   const orderId = req.params.orderId;
159   Order.findById(orderId)
160     .then(order => {

```

```

161 if (!order) {
162   return next(new Error('No order found.'));
163 }
164 if (order.user.userId.toString() !== req.user._id.toString()) {
165   return next(new Error('Unauthorized'));
166 }
167 const invoiceName = 'invoice-' + orderId + '.pdf';
168 const invoicePath = path.join('data', 'invoices', invoiceName);
169
170 const pdfDoc = new PDFDocument()
171 res.setHeader('Content-Type', 'application/pdf');
172 res.setHeader(
173   'Content-Disposition',
174   'inline; filename="' + invoiceName + '"'
175 )
176 /**this ensure that the pdf we generate also gets stored on the server
177  * and not just serve to the client.
178  *
179  * and when you call 'end()',
180  * these writable streams for creating the file and for sending the response
181  * will be closed. so file will be saved
182  * and the response will be sent.
183  */
184 pdfDoc.pipe(fs.createWriteStream(invoicePath))
185 pdfDoc.pipe(res)
186
187 /**'text()' allows us to ad a single line of text into the PDFDocument
188  * and you have to call pdfDoc to tell node
189  * when you are done writing to that stream
190  * because you have to be done at some point.
191  */
192 pdfDoc.text('Hello World!')
193
194 pdfDoc.end()
195 //fs.readFile(invoicePath, (err, data) => {
196 //  if (err) {
197 //    return next(err);
198 //  }
199 //  res.setHeader('Content-Type', 'application/pdf');
200 //  res.setHeader(
201 //    'Content-Disposition',
202 //    'inline; filename="' + invoiceName + '"'
203 //  );
204 //  res.send(data);
205 //});
206
207 //const file = fs.createReadStream(invoicePath)
208 //file.pipe(res)
209 })
210 .catch(err => next(err));
211 };
212

```

* Chapter 328: Generating .pdf Files With Order

Data

1. update
- ./controllers/shop.js


```
1  //./controllers/shop.js
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const PDFDocument= require('pdfkit')
7
8  const Product = require('../models/product');
9  const Order = require('../models/order');
10
11 exports.getProducts = (req, res, next) => {
12   Product.find()
13     .then(products => {
14       console.log(products);
15       res.render('shop/product-list', {
16         prods: products,
17         pageTitle: 'All Products',
18         path: '/products'
19       });
20     })
21     .catch(err => {
22       const error = new Error(err);
23       error.httpStatusCode = 500;
24       return next(error);
25     });
26 };
27
28 exports.getProduct = (req, res, next) => {
29   const prodId = req.params.productId;
30   Product.findById(prodId)
31     .then(product => {
32       res.render('shop/product-detail', {
33         product: product,
34         pageTitle: product.title,
35         path: '/products'
36       });
37     })
```

```

38     .catch(err => {
39         const error = new Error(err);
40         error.httpStatusCode = 500;
41         return next(error);
42     });
43 };
44
45 exports.getIndex = (req, res, next) => {
46     Product.find()
47         .then(products => {
48             res.render('shop/index', {
49                 prods: products,
50                 pageTitle: 'Shop',
51                 path: '/'
52             });
53         })
54         .catch(err => {
55             const error = new Error(err);
56             error.httpStatusCode = 500;
57             return next(error);
58         });
59 };
60
61 exports.getCart = (req, res, next) => {
62     req.user
63         .populate('cart.items.productId')
64         .execPopulate()
65         .then(user => {
66             const products = user.cart.items;
67             res.render('shop/cart', {
68                 path: '/cart',
69                 pageTitle: 'Your Cart',
70                 products: products
71             });
72         })
73         .catch(err => {
74             const error = new Error(err);
75             error.httpStatusCode = 500;
76             return next(error);
77         });
78 };
79
80 exports.postCart = (req, res, next) => {
81     const prodId = req.body.productId;
82     Product.findById(prodId)
83         .then(product => {
84             return req.user.addToCart(product);
85         })
86         .then(result => {
87             console.log(result);
88             res.redirect('/cart');
89         })
90         .catch(err => {
91             const error = new Error(err);
92             error.httpStatusCode = 500;
93             return next(error);

```

```

94     });
95 };
96
97 exports.postCartDeleteProduct = (req, res, next) => {
98     const prodId = req.body.productId;
99     req.user
100         .removeFromCart(prodId)
101         .then(result => {
102             res.redirect('/cart');
103         })
104         .catch(err => {
105             const error = new Error(err);
106             error.httpStatusCode = 500;
107             return next(error);
108         });
109 };
110
111 exports.postOrder = (req, res, next) => {
112     req.user
113         .populate('cart.items.productId')
114         .execPopulate()
115         .then(user => {
116             const products = user.cart.items.map(i => {
117                 return { quantity: i.quantity, product: { ...i.productId._doc } };
118             });
119             const order = new Order({
120                 user: {
121                     email: req.user.email,
122                     userId: req.user
123                 },
124                 products: products
125             });
126             return order.save();
127         })
128         .then(result => {
129             return req.user.clearCart();
130         })
131         .then(() => {
132             res.redirect('/orders');
133         })
134         .catch(err => {
135             const error = new Error(err);
136             error.httpStatusCode = 500;
137             return next(error);
138         });
139 };
140
141 exports.getOrders = (req, res, next) => {
142     Order.find({ 'user.userId': req.user._id })
143         .then(orders => {
144             res.render('shop/orders', {
145                 path: '/orders',
146                 pageTitle: 'Your Orders',
147                 orders: orders
148             });
149         })

```

```

150     .catch(err => {
151         const error = new Error(err);
152         error.httpStatusCode = 500;
153         return next(error);
154     });
155 };
156
157 exports.getInvoice = (req, res, next) => {
158     const orderId = req.params.orderId;
159     Order.findById(orderId)
160         .then(order => {
161             if (!order) {
162                 return next(new Error('No order found.'));
163             }
164             if (order.user.userId.toString() !== req.user._id.toString()) {
165                 return next(new Error('Unauthorized'));
166             }
167             const invoiceName = 'invoice-' + orderId + '.pdf';
168             const invoicePath = path.join('data', 'invoices', invoiceName);
169
170             const pdfDoc = new PDFDocument()
171             res.setHeader('Content-Type', 'application/pdf');
172             res.setHeader(
173                 'Content-Disposition',
174                 'inline; filename="' + invoiceName + '"'
175             )
176             pdfDoc.pipe(fs.createWriteStream(invoicePath))
177             pdfDoc.pipe(res)
178
179             pdfDoc.fontSize(26).text('Invoice', {
180                 underline: true
181             })
182
183             pdfDoc.text('-----')
184             let totalPrice = 0
185             /**'products' is an array
186              * because we store this products in a database as an array.
187              */
188             order.products.forEach(prod => {
189                 totalPrice += prod.quantity * prod.product.price
190                 pdfDoc
191                     .fontSize(14)
192                     .text(
193                         prod.product.title +
194                         ' - ' +
195                         prod.quantity +
196                         ' x ' +
197                         '$' +
198                         prod.product.price)
199             })
200             pdfDoc.text('---')
201             pdfDoc.fontSize(20).text('Total Price: $' + totalPrice)
202
203             pdfDoc.end()
204             //fs.readFile(invoicePath, (err, data) => {
205             //    if (err) {

```



```

206 // return next(err);
207 // }
208 // res.setHeader('Content-Type', 'application/pdf');
209 // res.setHeader(
210 //   'Content-Disposition',
211 //   'inline; filename="' + invoiceName + '"'
212 // );
213 // res.send(data);
214 //});
215
216 //const file = fs.createReadStream(invoicePath)
217 //file.pipe(res)
218 })
219 .catch(err => next(err));
220 };
221

```

* Chapter 329: Deleting Files

1. update
 - ./util/file.js
 - ./controllers/admin.js

- i delete prod1, and in the images folder, one image connected to prod1 is removed.

- fetching my invoice or generating it on the fly works. thanks to the fact that we store a snapshot of that in the database, it even works after the product was deleted.

```

1  ../util/file.js
2
3  const fs = require('fs')
4
5  const deleteFile = (filePath) => {
6    /**'unlink()' method deletes the name and file
7     * that is connected to the name,
8     * so it deletes a file at this path.
9     */
10   fs.unlink(filePath, (err) => {
11     if (err) {
12       throw (err)
13     }
14   })
15 }
16
17 exports.deleteFile = deleteFile

```

```

1 // ./controllers/admin.js
2
3 const mongoose = require('mongoose');
4
5 const fileHelper = require('../util/file')
6
7 const { validationResult } = require('express-validator/check');
8
9 const Product = require('../models/product');
10
11 exports.getAddProduct = (req, res, next) => {
12   res.render('admin/edit-product', {
13     pageTitle: 'Add Product',
14     path: '/admin/add-product',
15     editing: false,
16     hasError: false,
17     errorMessage: null,
18     validationErrors: []
19   });
20 };
21
22 exports.postAddProduct = (req, res, next) => {
23   const title = req.body.title;
24   const image = req.file;
25   const price = req.body.price;
26   const description = req.body.description;
27   if (!image) {
28     res.status(422).render('admin/edit-product', {
29       pageTitle: 'Add Product',
30       path: '/admin/add-product',
31       editing: false,
32       hasError: true,
33       product: {
34         title: title,
35         price: price,
36         description: description
37       },
38       errorMessage: 'Attached file is not an image',
39       validationErrors: []
40     });
41   }
42   const errors = validationResult(req);
43
44   if (!errors.isEmpty()) {
45     console.log(errors.array());
46     return res.status(422).render('admin/edit-product', {
47       pageTitle: 'Add Product',
48       path: '/admin/add-product',
49       editing: false,
50       hasError: true,
51       product: {
52         title: title,
53         imageUrl: imageUrl,
54         price: price,
55         description: description
56       },

```

```

57     errorMessage: errors.array()[0].msg,
58     validationErrors: errors.array()
59   });
60 }
61 const imageUrl = image.path
62
63 const product = new Product({
64   //_id: new mongoose.Types.ObjectId('5badf72403fd8b5be0366e81'),
65   title: title,
66   price: price,
67   description: description,
68   imageUrl: imageUrl,
69   userId: req.user
70 });
71 product
72   .save()
73   .then(result => {
74     // console.log(result);
75     console.log('Created Product');
76     res.redirect('/admin/products');
77   })
78   .catch(err => {
79     // return res.status(500).render('admin/edit-product', {
80     //   pageTitle: 'Add Product',
81     //   path: '/admin/add-product',
82     //   editing: false,
83     //   hasError: true,
84     //   product: {
85     //     title: title,
86     //     imageUrl: imageUrl,
87     //     price: price,
88     //     description: description
89     //   },
90     //   errorMessage: 'Database operation failed, please try again.',
91     //   validationErrors: []
92     // });
93     //res.redirect('/500');
94     const error = new Error(err)
95     error.httpStatusCode = 500
96     return next(error)
97   });
98 };
99
100 exports.getEditProduct = (req, res, next) => {
101   const editMode = req.query.edit;
102   if (!editMode) {
103     return res.redirect('/');
104   }
105   const prodId = req.params.productId;
106   Product.findById(prodId)
107     .then(product => {
108       if (!product) {
109         return res.redirect('/');
110       }
111       res.render('admin/edit-product', {
112         pageTitle: 'Edit Product',

```

```

113     path: '/admin/edit-product',
114     editing: editMode,
115     product: product,
116     hasError: false,
117     errorMessage: null,
118     validationErrors: []
119   });
120 })
121 .catch(err => {
122   const error = new Error(err)
123   error.httpStatusCode = 500
124   return next(error)
125 });
126 };
127
128 exports.postEditProduct = (req, res, next) => {
129   const prodId = req.body.productId;
130   const updatedTitle = req.body.title;
131   const updatedPrice = req.body.price;
132   const image = req.body.imageUrl;
133   const updatedDesc = req.body.description;
134
135   const errors = validationResult(req);
136
137   if (!errors.isEmpty()) {
138     return res.status(422).render('admin/edit-product', {
139       pageTitle: 'Edit Product',
140       path: '/admin/edit-product',
141       editing: true,
142       hasError: true,
143       product: {
144         title: updatedTitle,
145         price: updatedPrice,
146         description: updatedDesc,
147         _id: prodId
148       },
149       errorMessage: errors.array()[0].msg,
150       validationErrors: errors.array()
151     });
152   }
153
154   Product.findById(prodId)
155     .then(product => {
156       if (product.userId.toString() !== req.user._id.toString()) {
157         return res.redirect('/');
158       }
159       product.title = updatedTitle;
160       product.price = updatedPrice;
161       product.description = updatedDesc;
162       if (image) {
163         fileHelper.deleteFile(product.imageUrl)
164         product.imageUrl = updatedImageUrl;
165       }
166       return product.save().then(result => {
167         console.log('UPDATED PRODUCT!');
168         res.redirect('/admin/products');

```

```

169     });
170   })
171   .catch(err => {
172     const error = new Error(err)
173     error.httpStatusCode = 500
174     return next(error)
175   });
176 };
177
178 exports.getProducts = (req, res, next) => {
179   Product.find({ userId: req.user._id })
180     // .select('title price _id')
181     // .populate('userId', 'name')
182     .then(products => {
183       console.log(products);
184       res.render('admin/products', {
185         prods: products,
186         pageTitle: 'Admin Products',
187         path: '/admin/products'
188       });
189     })
190     .catch(err => {
191       const error = new Error(err)
192       error.httpStatusCode = 500
193       return next(error)
194     });
195 };
196
197 exports.postDeleteProduct = (req, res, next) => {
198   const prodId = req.body.productId;
199   Product.findById(prodId)
200     .then(product => {
201       if (!product) {
202         return next(new Error('Product not found'))
203       }
204       fileHelper.deleteFile(product.imageUrl)
205       /**i should trigger 'deleteOne'
206        * after i found 'product' above
207        * otherwise we have a race condition
208        * where deleting could finish before finding is finished
209        * and that would be bad.
210        */
211       return Product.deleteOne({ _id: prodId, userId: req.user._id })
212     })
213     .then(() => {
214       console.log('DESTROYED PRODUCT');
215       res.redirect('/admin/products');
216     })
217     .catch(err => {
218       const error = new Error(err)
219       error.httpStatusCode = 500
220       return next(error)
221     });
222 };
223

```

* Chapter 330: Fixing Invoice Links

1. update
- ./views/shop/orders.ejs


```
1 <!--./views/shop/orders.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/orders.css">
5   </head>
6
7   <body>
8     <%- include('../includes/navigation.ejs') %>
9     <main>
10      <% if (orders.length <= 0) { %>
11        <h1>Nothing there!</h1>
12      <% } else { %>
13        <ul class="orders">
14          <% orders.forEach(order => { %>
15            <li class="orders__item">
16              <h1>Order - # <%= order._id %> - <a href="/orders/<%= order._id
17              %>">Invoice</a></h1>
18              <ul class="orders__products">
19                <% order.products.forEach(p => { %>
20                  <li class="orders__products-item"><%= p.product.title %>
21                  (<%= p.quantity %>)</li>
22                <% }); %>
23              </ul>
24            </li>
25          <% }); %>
26        </ul>
27      <% } %>
28    </main>
29    <%- include('../includes/end.ejs') %>
```