

14. Sessions & Cookies

* Chapter 226: Module Introduction



What's In This Module?

What are Cookies?

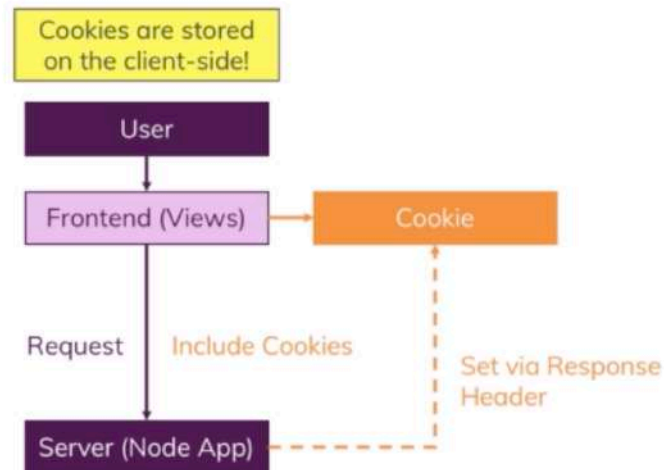
What are Sessions?

Using Session & Cookies?

© 2019 Academind

* Chapter 227: What is a Cookie?

What's a Cookie?



35 slides

- the request requires us to store some kind of data in the browser.
- let's say we have a login page and when the user logs in, we wanna store the information that the user is logged in somewhere, so that when the user reloads the page and therefore technically a new request is sent, we still have that information around that the user is logged in
- cookie is important to telling the user or to storing that information that the user is authenticated. we can store that information in the browser in the environment the user interacts with and we can send this with subsequent requests to include the cookie there to send the data we stored in the cookie like the information that we are logged in to the server. so cookies are stored on the client side.

* Chapter 228: The Current Project Status

1. update
 - ./views/includes/navigation.ejs
 - ./public/css/main.css

- authentication is a great example for data you would wanna store for a specific user and that is one of the typical use cases for using cookies and sessions.

```

1 <!--./views/includes/navigation.ejs-->
2
3 <div class="backdrop"></div>
4 <header class="main-header">
5   <button id="side-menu-toggle">Menu</button>
6   <nav class="main-header__nav">
7     <ul class="main-header__item-list">
8       <li class="main-header__item">
9         <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10      </li>
11      <li class="main-header__item">
12        <a class="<%= path === '/products' ? 'active' : '' %>"
13        href="/products">Products</a>
14      </li>
15      <li class="main-header__item">

```

```

15         <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
16     </li>
17     <li class="main-header__item">
18         <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
19     </li>
20     <!-- <li class="main-header__item">
21         <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
22     </a>
23 </li>
24 <li class="main-header__item">
25     <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
26 </a>
27 </li> -->
28 </ul>
29 <!--i added new list item which is named 'login'-->
30 <ul class="main-header__item-list">
31     <li class="main-header__item">
32         <a href="/login">Login</a>
33     </li>
34 </ul>
35 </nav>
36 </header>
37
38 <nav class="mobile-nav">
39     <ul class="mobile-nav__item-list">
40         <li class="mobile-nav__item">
41             <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
42         </li>
43         <li class="mobile-nav__item">
44             <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
45         </li>
46         <li class="mobile-nav__item">
47             <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
48         </li>
49         <li class="mobile-nav__item">
50             <a class="<%= path === '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
51         </li>
52         <li class="mobile-nav__item">
53             <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
54         </a>
55     </li>
56     <li class="mobile-nav__item">
57         <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
58     </a>
59 </li>
60 </ul>
61 </nav>

```

```

1 /*./public/css/main.css*/

```

```

2

```

```

3 @import url('https://fonts.googleapis.com/css?family=Open+Sans:400,700');

```

```
4
5 * {
6   box-sizing: border-box;
7 }
8
9 body {
10  padding: 0;
11  margin: 0;
12  font-family: 'Open Sans', sans-serif;
13 }
14
15 main {
16   padding: 1rem;
17   margin: auto;
18 }
19
20 form {
21   display: inline;
22 }
23
24 .centered {
25   text-align: center;
26 }
27
28 .image {
29   height: 20rem;
30 }
31
32 .image img {
33   height: 100%;
34 }
35
36 .main-header {
37   width: 100%;
38   height: 3.5rem;
39   background-color: #00695c;
40   padding: 0 1.5rem;
41   display: flex;
42   align-items: center;
43 }
44
45 .main-header__nav {
46   height: 100%;
47   width: 100%;
48   display: flex;
49   align-items: center;
50   justify-content: space-between;
51 }
52
53 .main-header__item-list {
54   list-style: none;
55   margin: 0;
56   padding: 0;
57   display: flex;
58 }
59
```

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

```
116   color: black;
117   font-size: 1.5rem;
118   padding: 0.5rem 2rem;
119 }
120
121 .mobile-nav__item a:active,
122 .mobile-nav__item a:hover,
123 .mobile-nav__item a.active {
124   background: #00695c;
125   color: white;
126   border-radius: 3px;
127 }
128
129 #side-menu-toggle {
130   border: 1px solid white;
131   font: inherit;
132   padding: 0.5rem;
133   display: block;
134   background: transparent;
135   color: white;
136   cursor: pointer;
137 }
138
139 #side-menu-toggle:focus {
140   outline: none;
141 }
142
143 #side-menu-toggle:active,
144 #side-menu-toggle:hover {
145   color: #ffeb3b;
146   border-color: #ffeb3b;
147 }
148
149 .backdrop {
150   position: fixed;
151   top: 0;
152   left: 0;
153   width: 100%;
154   height: 100vh;
155   background: rgba(0, 0, 0, 0.5);
156   z-index: 5;
157   display: none;
158 }
159
160 .grid {
161   display: flex;
162   flex-wrap: wrap;
163   justify-content: space-around;
164   align-items: stretch;
165 }
166
167 .card {
168   box-shadow: 0 2px 8px rgba(0, 0, 0, 0.26);
169 }
170
171 .card__header,
```

```
172 .card__content {
173     padding: 1rem;
174 }
175
176 .card__header h1,
177 .card__content h1,
178 .card__content h2,
179 .card__content p {
180     margin: 0;
181 }
182
183 .card__image {
184     width: 100%;
185 }
186
187 .card__image img {
188     width: 100%;
189 }
190
191 .card__actions {
192     padding: 1rem;
193     text-align: center;
194 }
195
196 .card__actions button,
197 .card__actions a {
198     margin: 0 0.25rem;
199 }
200
201 .btn {
202     display: inline-block;
203     padding: 0.25rem 1rem;
204     text-decoration: none;
205     font: inherit;
206     border: 1px solid #00695c;
207     color: #00695c;
208     background: white;
209     border-radius: 3px;
210     cursor: pointer;
211 }
212
213 .btn:hover,
214 .btn:active {
215     background-color: #00695c;
216     color: white;
217 }
218
219 .btn.danger {
220     color: red;
221     border-color: red;
222 }
223
224 .btn.danger:hover,
225 .btn.danger:active {
226     background: red;
227     color: white;
```

```

228 }
229
230 @media (min-width: 768px) {
231   .main-header__nav {
232     display: flex;
233   }
234
235   #side-menu-toggle {
236     display: none;
237   }
238 }
239

```

* Chapter 229: Optional: Creating The Login Form

1. update

- ./routes/auth.js
- app.js
- ./controllers/auth.js
- ./views/includes/navigation.ejs
- ./views/auth/login.ejs
- ./public/css/auth.css

Shop Products Cart Orders Login

E-Mail

Password

Login



E-Mail

Password

Login

```
1 // ./routes/auth.js
2
3 const express = require('express')
4
5 const authController = require('../controllers/auth')
6
7 const router = express.Router()
8
9 router.get('/login', authController.getLogin)
10
11 module.exports = router
```

```
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
9 const errorController = require('../controllers/error');
10 const User = require('../models/user')
11
12 const app = express();
13
14 app.set('view engine', 'ejs');
15 app.set('views', 'views');
16
17 const adminRoutes = require('../routes/admin');
18 const shopRoutes = require('../routes/shop');
19 const authRoutes = require('../routes/auth');
20
21 app.use(bodyParser.urlencoded({ extended: false }));
22 app.use(express.static(path.join(__dirname, 'public')));
23
```

```

24
25 app.use((req, res, next) => {
26   User.findById('5cbb2b2c80bd7193adb9eeeb')
27     .then(user => {
28       req.user = user
29       next();
30     })
31     .catch(err => console.log(err));
32 });
33
34 app.use('/admin', adminRoutes);
35 app.use(shopRoutes);
36 app.use(authRoutes)
37
38 app.use(errorController.get404);
39
40 mongoose
41   .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
retryWrites=true')
42   .then(result => {
43     User
44       .findOne()
45       .then(user => {
46         if(!user){
47           const user = new User({
48             name: 'Max',
49             email: 'max@test.com',
50             cart: {
51               items: []
52             }
53           })
54           user.save()
55         }
56       })
57     app.listen(3000)
58   })
59   .catch(err => {
60     console.log(err)
61   })

```

```

1 //./controllers/auth.js
2
3 exports.getLogin = (req, res, next) => {
4   res.render('auth/login', {
5     path: '/login',
6     pageTitle: 'Login',
7   });
8 };

```

```

1 <!--./views/includes/navigation.ejs-->
2
3 <div class="backdrop"></div>
4 <header class="main-header">
5   <button id="side-menu-toggle">Menu</button>
6   <nav class="main-header__nav">
7     <ul class="main-header__item-list">
8       <li class="main-header__item">

```

```

9         <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10     </li>
11     <li class="main-header__item">
12         <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
13     </li>
14     <li class="main-header__item">
15         <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
16     </li>
17     <li class="main-header__item">
18         <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
19     </li>
20     <!-- <li class="main-header__item">
21         <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
22     </a>
23     </li>
24     <li class="main-header__item">
25         <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
26     </a>
27     </li> -->
28 </ul>
29 <!--i added new list item which is named 'login'-->
30 <ul class="main-header__item-list">
31     <li class="main-header__item">
32         <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
33     </li>
34 </ul>
35 </nav>
36 </header>
37
38 <nav class="mobile-nav">
39     <ul class="mobile-nav__item-list">
40         <li class="mobile-nav__item">
41             <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
42         </li>
43         <li class="mobile-nav__item">
44             <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
45         </li>
46         <li class="mobile-nav__item">
47             <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
48         </li>
49         <li class="mobile-nav__item">
50             <a class="<%= path === '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
51         </li>
52         <li class="mobile-nav__item">
53             <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
54         </a>
55         </li>
56         <li class="mobile-nav__item">
57             <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products

```

```

58     </a>
59   </li>
60 </ul>
61 </nav>

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

1 /*./public/css/auth.css*/
2
3 .login-form {
4   width: 20rem;
5   max-width: 90%;
6   margin: auto;
7   display: block;
8 }

```

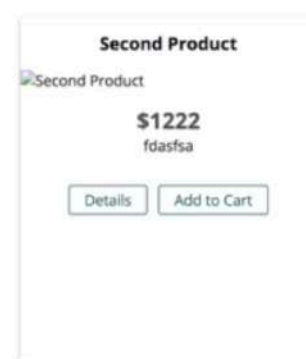
* Chapter 230: Adding The Request Driven Login Solution

1. update
 - ./controllers/auth.js
 - routes/auth.js
 - ./views/includes/navigation.ejs
 - ./controllers/admin.js
 - ./controllers/error.js
 - ./controllers/shop.js

E-Mail

Password

Login



```
1 //./controllers/auth.js
2
3 exports.getLogin = (req, res, next) => {
4   res.render('auth/login', {
5     path: '/login',
6     pageTitle: 'Login',
7     isAuthenticated: req.isLoggedIn
8   });
9 };
10
11 exports.postLogin = (req, res, next) => {
12   /**we wanna store that information that the user is authenticated
13    * how could we store that?
```

```

14  * i store that information in my request object and 'isLoggedIn'
15  * because we are already doing a similar thing in app.js file like 'req.user = user'
16  * so that we can use it for the rest of that request.
17  * so in all the routes and controllers where we handle that request.
18  *
19  * we can set this 'true' by default at the start it will not be set.
20  * so that the value will be undefined which is treated as false
21  * and that is the information i need
22  * i will set to true when we do login.
23  */
24
25  /**even though i'm storing the information that i'm logged in,
26   * in 'isLoggedIn', when we click the 'LogIn' button,
27   * i'm storing it in my request and then i use that information in the request on every
other route i handle
28   * and i pass it into 'isAuthenticated' which is the field which i'm using in my frontend
like ./views/includes/navigation.ejs
29   * there i'm checking for 'isAuthenticated'
30   * and that is what i'm passing to that frontend in my render calls
31   *
32   * the problem is that i update 'isLoggedIn' here in the request
33   * and what happens to the request once i send a response,
34   * and we do send a response by redirecting?
35   * the request is dead, it's done.
36   * with a response, we basically finished a request, we got a 'request("req.isLoggedIn =
true")' and we sent a 'response("res.redirect('/')")' here
37   * we are done. so 'req.isLoggedIn' doesn't stick around
38   * 'req.isLoggedIn' is lost after the request of after we send the response
39   *
40   * so whenever we visit a different page, like where we get redirected,
41   * so we get redirected here, and we reach our 'getIndex' in ./controllers/shop.js
42   * and there we do render the 'shop/index' page
43   *
44   * but this is a brand new request,
45   * the redirection creates a brand new request and this is important to understand
46   * we are working with totally separate requests and that is important
47   *
48   * because your application, your page will have hundreds of users
49   * and the requests of all these users are not related to each other.
50   * otherwise they could maybe look into data that they shouldn't see
51   * and even the requests of a single user.
52   * so requests made from the same IP address are treated as totally independent requests
53   * they are not seen in a bigger context or anything like that
54   * this is a good thing and deliberately designed that way
55   * and therefore any data we store here can be used as long as we are working on the same
request.
56   * that is why when we retrieve the user in app.js file here
57   * and i store it in the 'request("req.user = user")',
58   * that is why we still can use that req.user in all our action controllers
59   * because they can again, at a later point of time,
60   * this middleware in app.js file like below runs on every incoming request before our
routes handle it.
61   *
62   * app.use((req, res, next) => {
63   *   User.findById('5cbb2b2c80bd7193adb9eeeb')
64   *     .then(user => {

```

```

65     *      req.user = user
66     *      next()
67     *  })
68     *      .catch(err => console.log(err))
69     *  })
70     *
71     * so the data we store on the middleware above is used in the same request cycle,
72     * in our route handler in our controllers.
73     *
74     * but if i change the request at the end of its lifetime
75     * like here 'req', 'res' and dead it's done, before i send the response
76     * this data will be not useful to us
77     *
78     */
79     req.isLoggedIn = true
80     res.redirect('/')
81 };

```

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

```

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

```

```

24         <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
25     </a>
26 </li>
27 <li class="main-header__item">
28     <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
29 </a>
30 </li>
31 <% } %>
32 </ul>
33 <ul class="main-header__item-list">
34     <li class="main-header__item">
35         <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
36     </li>
37 </ul>
38 </nav>
39 </header>
40
41 <nav class="mobile-nav">
42     <ul class="mobile-nav__item-list">
43         <li class="mobile-nav__item">
44             <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
45         </li>
46         <li class="mobile-nav__item">
47             <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
48         </li>
49         <li class="mobile-nav__item">
50             <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
51         </li>
52         <li class="mobile-nav__item">
53             <a class="<%= path === '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
54         </li>
55         <li class="mobile-nav__item">
56             <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
57         </a>
58     </li>
59     <li class="mobile-nav__item">
60         <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
61     </a>
62 </li>
63 </ul>
64 </nav>

```

```

1 exports.get404 = (req, res, next) => {
2     res.status(404)
3     .render(
4         '404',
5         {
6             pageTitle: 'Page Not Found',
7             path: '/404',
8             isAuthenticated: req.isLoggedIn
9         }
10    );

```



```

11 };

1  //../controllers/shop.js
2
3  const Product = require('../models/product');
4  const Order = require('../models/order')
5
6  exports.getProducts = (req, res, next) => {
7    Product.find()
8      .then(products => {
9        console.log(products)
10       res.render('shop/product-list', {
11         prods: products,
12         pageTitle: 'All Products',
13         path: '/products',
14         isAuthenticated: req.isAuthenticated()
15       });
16     })
17     .catch(err => {
18       console.log(err);
19     });
20 };
21
22 exports.getProduct = (req, res, next) => {
23   const prodId = req.params.productId;
24   Product.findById(prodId)
25     .then(product => {
26       res.render('shop/product-detail', {
27         product: product,
28         pageTitle: product.title,
29         path: '/products',
30         isAuthenticated: req.isAuthenticated()
31       });
32     })
33     .catch(err => console.log(err));
34 };
35
36 exports.getIndex = (req, res, next) => {
37   Product.find()
38     .then(products => {
39       res.render('shop/index', {
40         prods: products,
41         pageTitle: 'Shop',
42         path: '/',
43         isAuthenticated: req.isAuthenticated()
44       });
45     })
46     .catch(err => {
47       console.log(err);
48     });
49 };
50
51 exports.getCart = (req, res, next) => {
52   req.user
53     .populate('cart.items.productId')
54     .execPopulate()
55     .then(user => {

```

```

56     const products = user.cart.items
57     res.render('shop/cart', {
58       path: '/cart',
59       pageTitle: 'Your Cart',
60       products: products,
61       isAuthenticated: req.isLoggedIn
62     });
63   })
64   .catch(err => console.log(err));
65 };
66
67 exports.postCart = (req, res, next) => {
68   const prodId = req.body.productId;
69   Product.findById(prodId)
70     .then(product => {
71       return req.user.addToCart(product);
72     })
73     .then(result => {
74       console.log(result);
75       res.redirect('/cart');
76     });
77 };
78
79 exports.postCartDeleteProduct = (req, res, next) => {
80   const prodId = req.body.productId;
81   req.user
82     .removeFromCart(prodId)
83     .then(result => {
84       res.redirect('/cart');
85     })
86     .catch(err => console.log(err));
87 };
88
89 exports.postOrder = (req, res, next) => {
90   req.user
91     .populate('cart.items.productId')
92     .execPopulate()
93     .then(user => {
94       console.log(user.cart.items)
95       const products = user.cart.items.map(i => {
96
97         return { quantity: i.quantity, product: { ...i.productId._doc } }
98       })
99       const order = new Order({
100         user: {
101           name: req.user.name,
102           userId: req.user
103         },
104         products: products
105       })
106       order.save()
107     })
108     .then(result => {
109       return req.user.clearCart()
110     })
111     .then(() => {

```

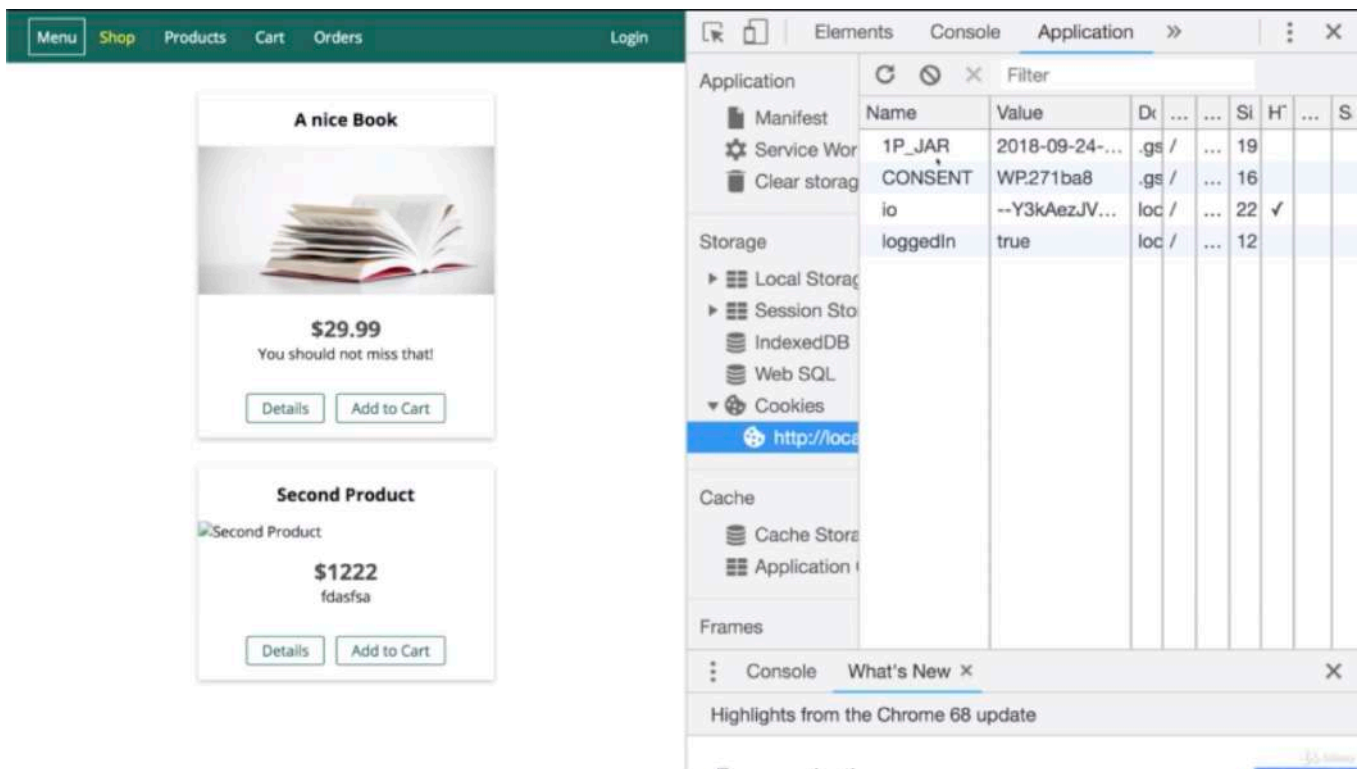
```

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

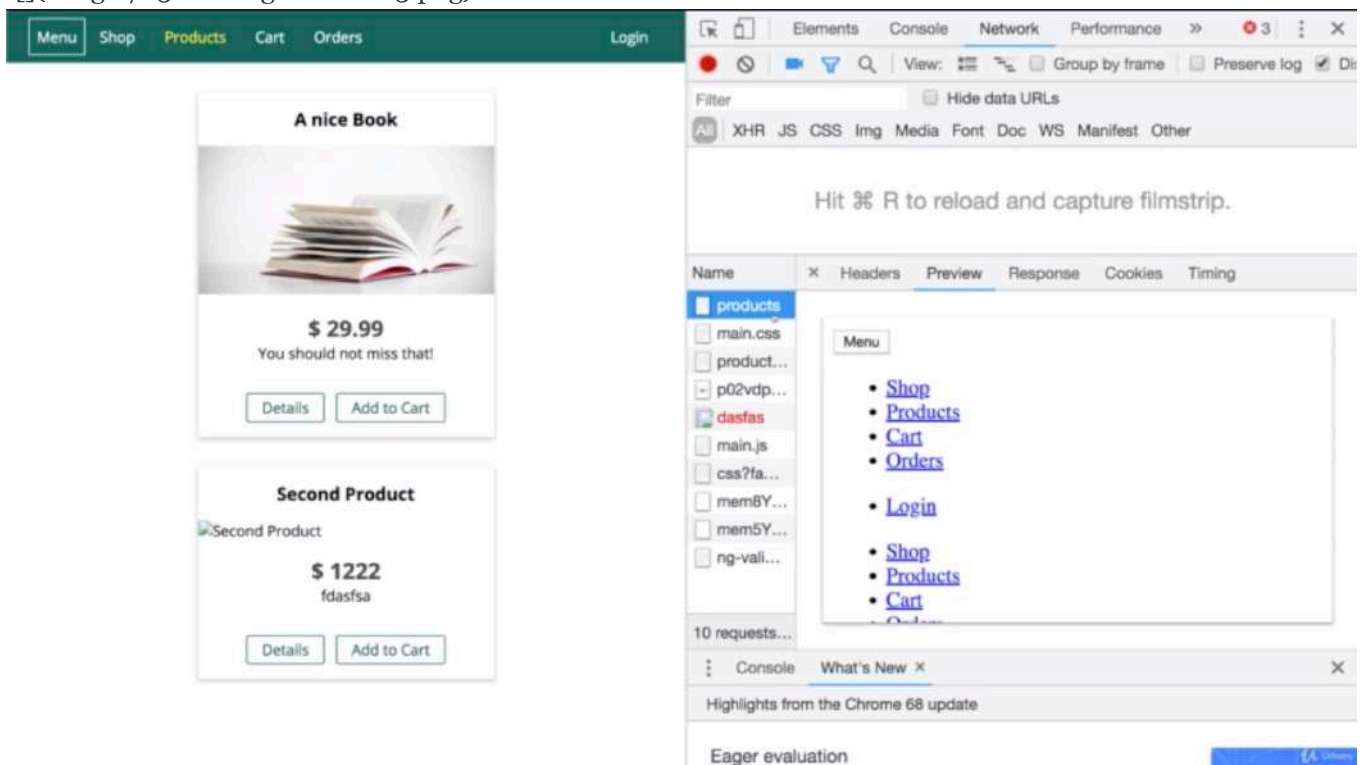
```

* Chapter 231: Setting A Cookie

1. update
- ./controllers/auth.js

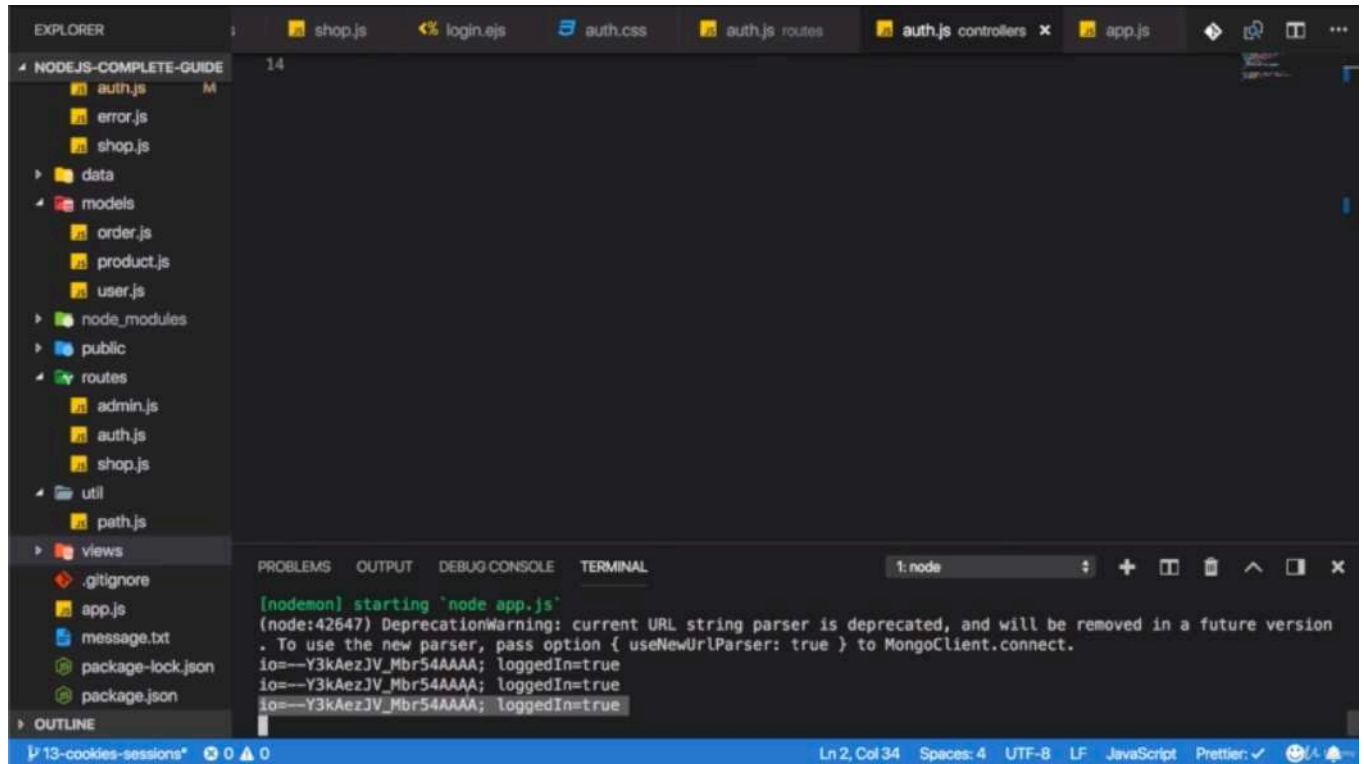


- open your developer tools and go to 'Application' tab and there to cookies, there is the 'Cookies' and if you expand this, you should see your current address here and if you click on that, you will see some cookies
- some cookies will be set by 3rd party plugins. and you will also see 'loggedIn' and the value is true. that is the cookie we just set.
- there is domain to which it belongs, the path, when it will expire and this state is in the past because it's a so-called session cookie which will expire once you close the browser and come back, you see the size.
- cookies is now set. now this cookie is not only set but the browser by default sends it to the server with every request we make

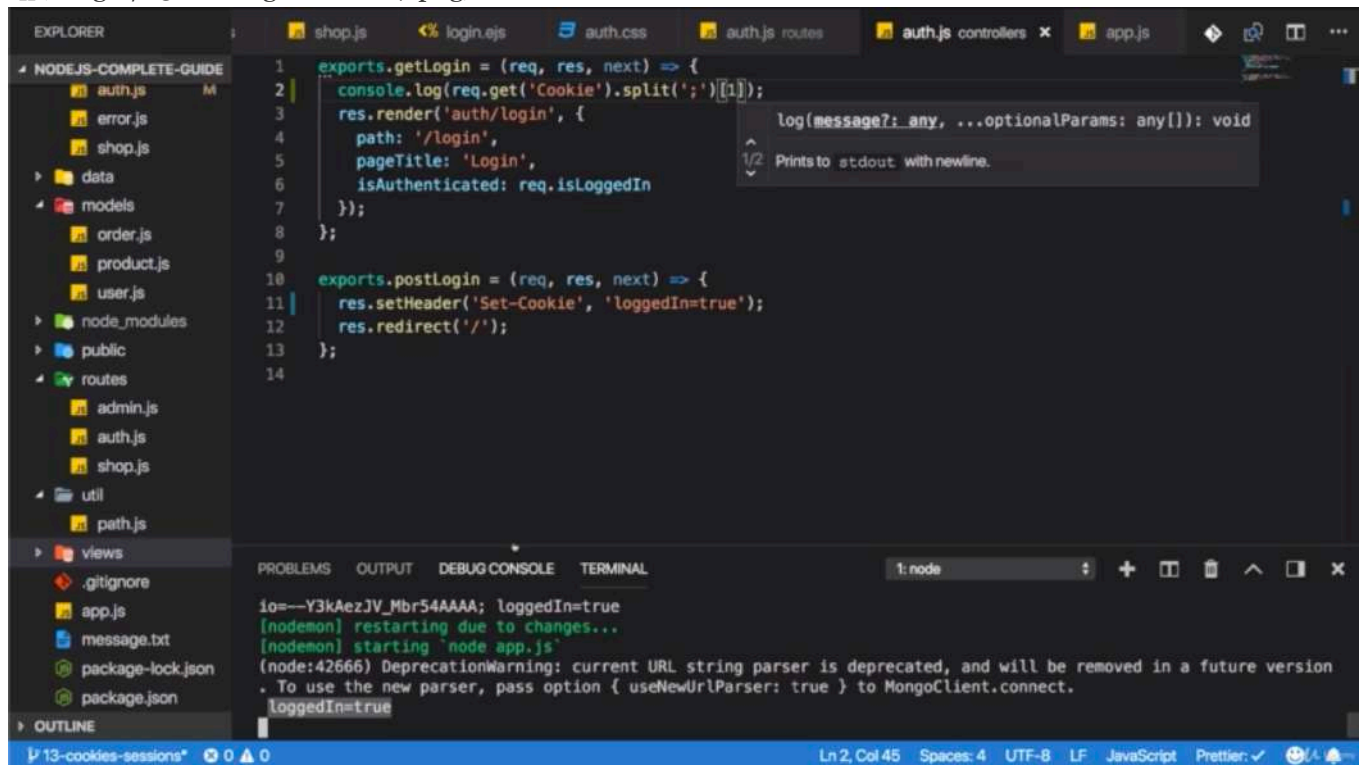


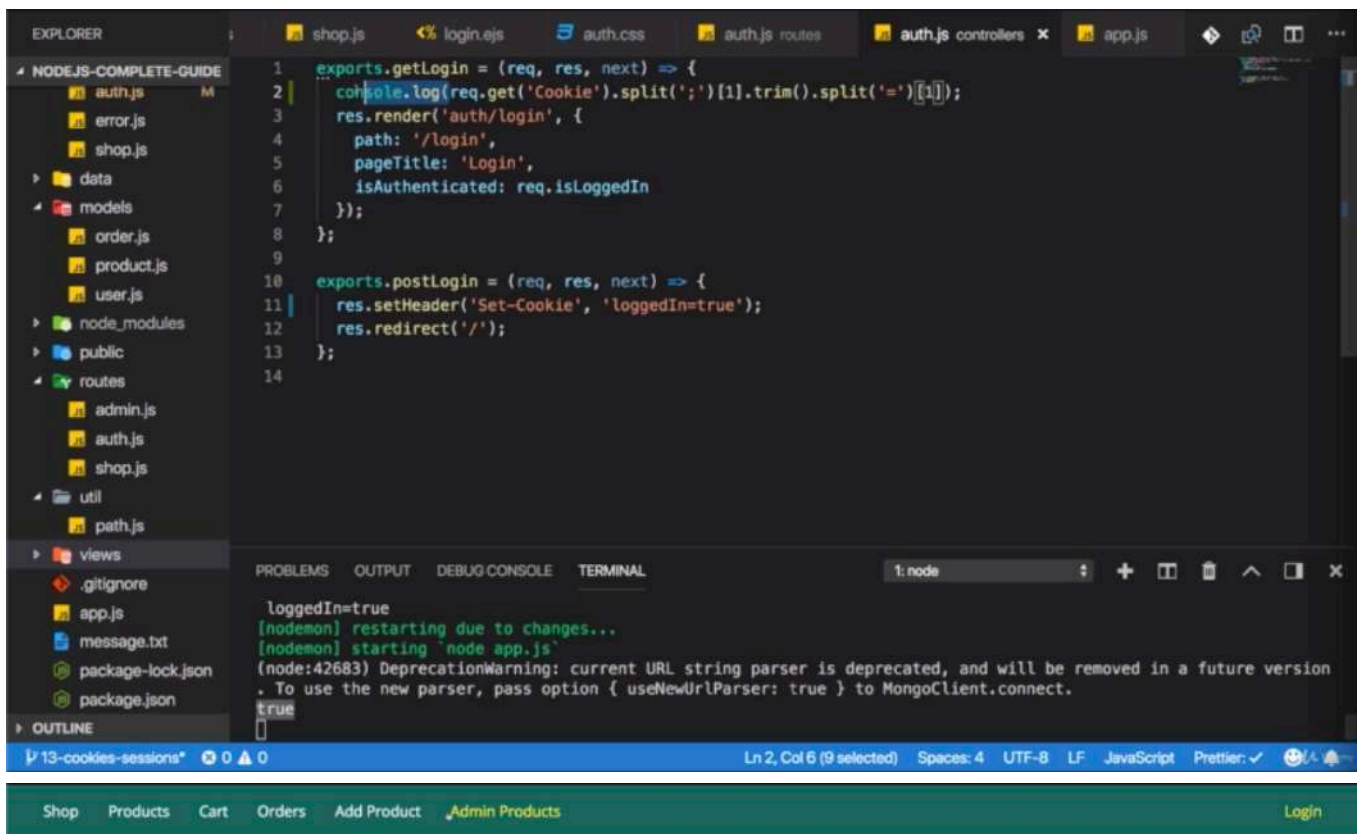
- this is the request which was sent to the product page, and this is the Header and go to 'Request Header', you see that a cookie was sent
- the first one comes from an extension but the 2nd one is our cookie. so it was sent to our server and now since

we have that, every request will have that cookie attached to itself and therefore this data is sent with every request and now we can use that.



- you see this output. we can ignore first cookie but 2nd one 'loggedIn=true' is our 'loggedIn' cookie.





E-Mail

Password

```
1 //./controllers/auth.js
2
3 exports.getLogin = (req, res, next) => {
4   /**i can extract 'isLoggedIn' information from the incoming request header,
5    * i'm getting the cookie header and i'm getting the 2nd cookie that is sent which happens
6    * to be our 'isLoggedIn' in cookie
7    * and then i extract 'true' value.
8    */
9   const isLoggedIn = req.get('Cookie')
10     .split(';')[1]
11     .trim()
12     .split('=')[1]
13   res.render('auth/login', {
```

```

13     path: '/login',
14     pageTitle: 'Login',
15     /**and now i have 'isLoggedIn' information which i can pass to 'isAuthenticated'
16     * with that on that page, if i reload it, there are 'Admin Products' and 'Add
Product' in the menu bar
17     * because we enable that again
18     * because we store that information across request
19     */
20     isAuthenticated: isLoggedIn
21   });
22 };
23
24 exports.postLogin = (req, res, next) => {
25   /** so we found out that using a request for storing this is not ideal
26   * because the request is dead after sending a response.
27   * which alternatives do we have?
28   *
29   * one alternative would be 'global' variable
30   * you could use a global variable which you store in an extra file
31   * and which you export from that file and which you then change
32   * and that variable survive your request cycles
33   * but since that variable would be shared across all request,
34   * it would be also shared across all users
35   *
36   * so that is where cookies can help us
37   * with cookies we can store data in the browser of a single user
38   * and store data in that browser which is customized to that user
39   * which doesn't affect all the other user
40   * but can be sent with requests to tell us
41   * hey i already am authenticated and that is exactly what we will do here
42   *
43   * we can set a cookie by 'setHeader()' on our response
44   * in 'setHeader()', we first of all define the name of the header
45   * and the name is 'Set-Cookie'
46   * then you have the value for cookie in 2nd argument
47   * and its simplest form is key-value pair where you define any name you want and any
value you want
48   * so 'loggedIn=true' would set a cookie
49   *
50   */
51   res.setHeader('Set-Cookie', 'loggedIn=true')
52   res.redirect('/')
53 };
54

```

* Chapter 232: Manipulating Cookies

1. update
- ./controllers/auth.js

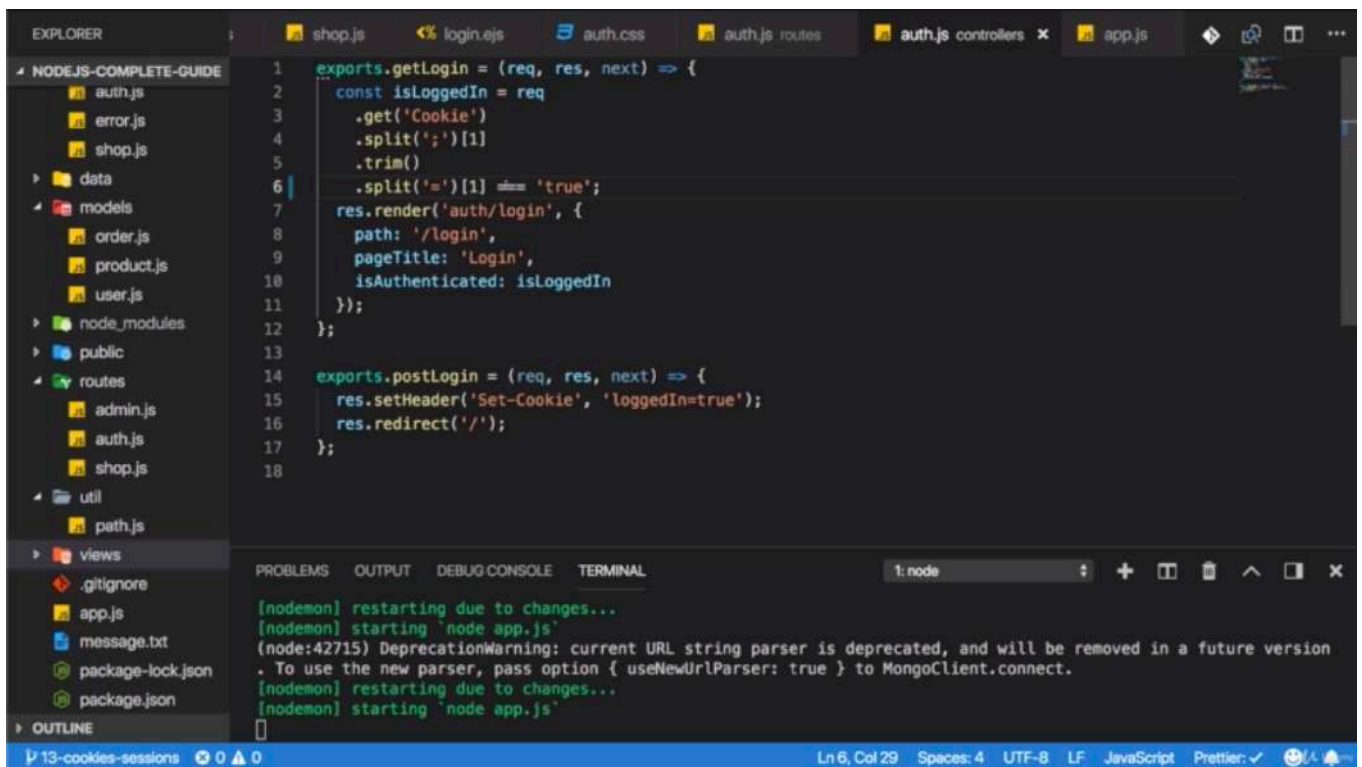
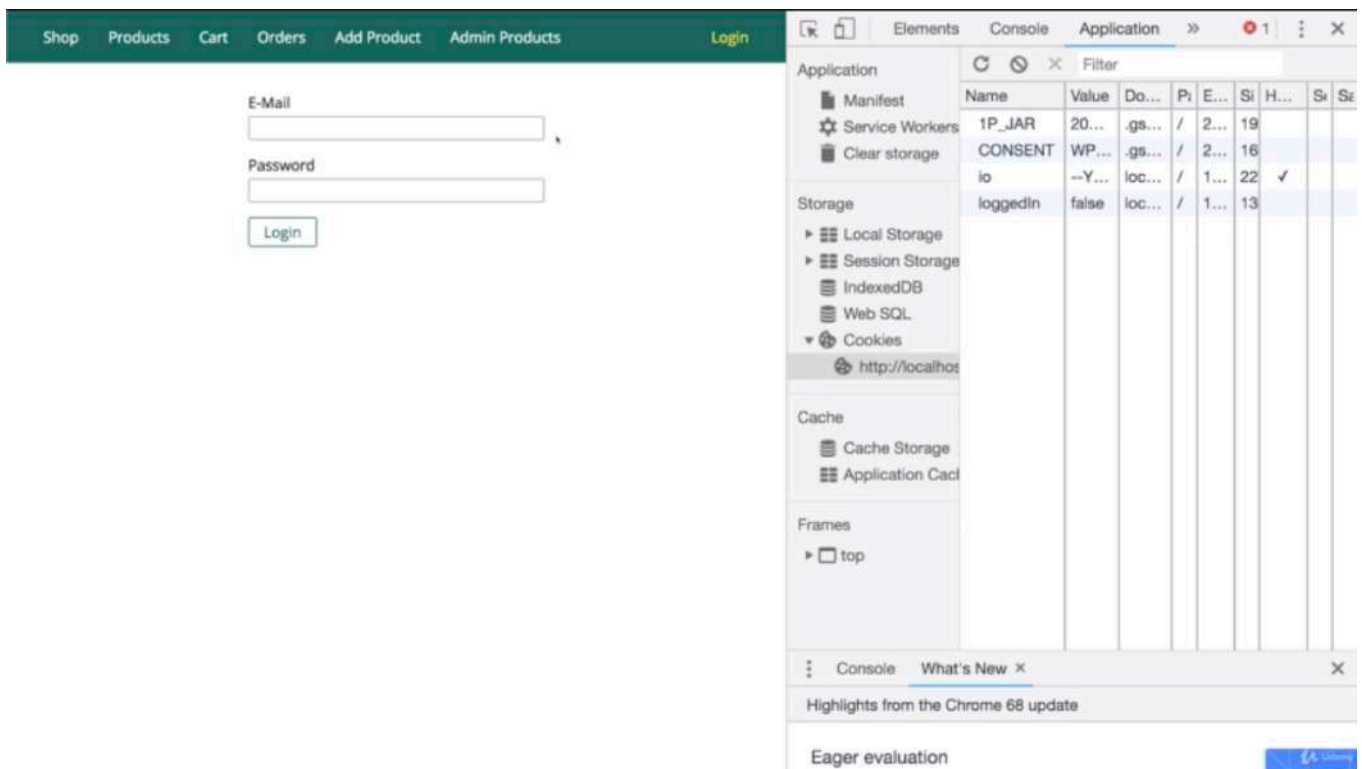
The screenshot shows a web application with a login form and its developer tools. The login form has fields for "E-Mail" and "Password", and a "Login" button. The developer tools, specifically the "Application" tab, show a table of cookies. The "loggedIn" cookie is highlighted, showing a value of "true".

Name	Value	Domain	Ex...	...	HTTP
1P_JAR	2018...	.gsta...	/	20...	19		
CONSENT	WP2...	.gsta...	/	20...	16		
io	--Y3k...	local...	/	19...	22	✓	
loggedIn	true	local...	/	19...	12		

The screenshot shows the same web application login page and developer tools. In this instance, the "loggedIn" cookie value has been changed to "false".

Name	Value	Domain	Ex...	...	HTTP
1P_JAR	2018...	.gsta...	/	20...	19		
CONSENT	WP2...	.gsta...	/	20...	16		
io	--Y3k...	local...	/	19...	22	✓	
loggedIn	false	localhos	/	19...	13		

- in the last lecture, i showed you how you can set a cookie and some bit too complex way of extracting that cookie. if you wanna extract cookies, there also are 3rd party packages which can help you with that.
- but our approach has another flaw. since i can access my cookies that easily in the developer tool, i can easily change them, i can go here and manipulate the value.



- for example if i set it to false and i reload, i'm still logged in because false is sent as text and text is always treated as true. but we can simply add a comparison here and see if that value is equal to true, so to the text true here.

Shop
Products
Cart
Orders
Login

E-Mail

Password

Login

Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://localhost:3000
Cache
Cache Storage
Application Cache
Frames
top

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	Session
1P_JAR	20...	.gs...	/	2...	19			
CONSENT	WP...	.gs...	/	2...	16			
io	--Y...	loc...	/	1...	22	✓		
loggedIn	false	loc...	/	1...	13			

Console
What's New
Highlights from the Chrome 68 update
Eager evaluation

- if i reload here, i'm not logged in anymore.

Shop
Products
Cart
Orders
Login

E-Mail

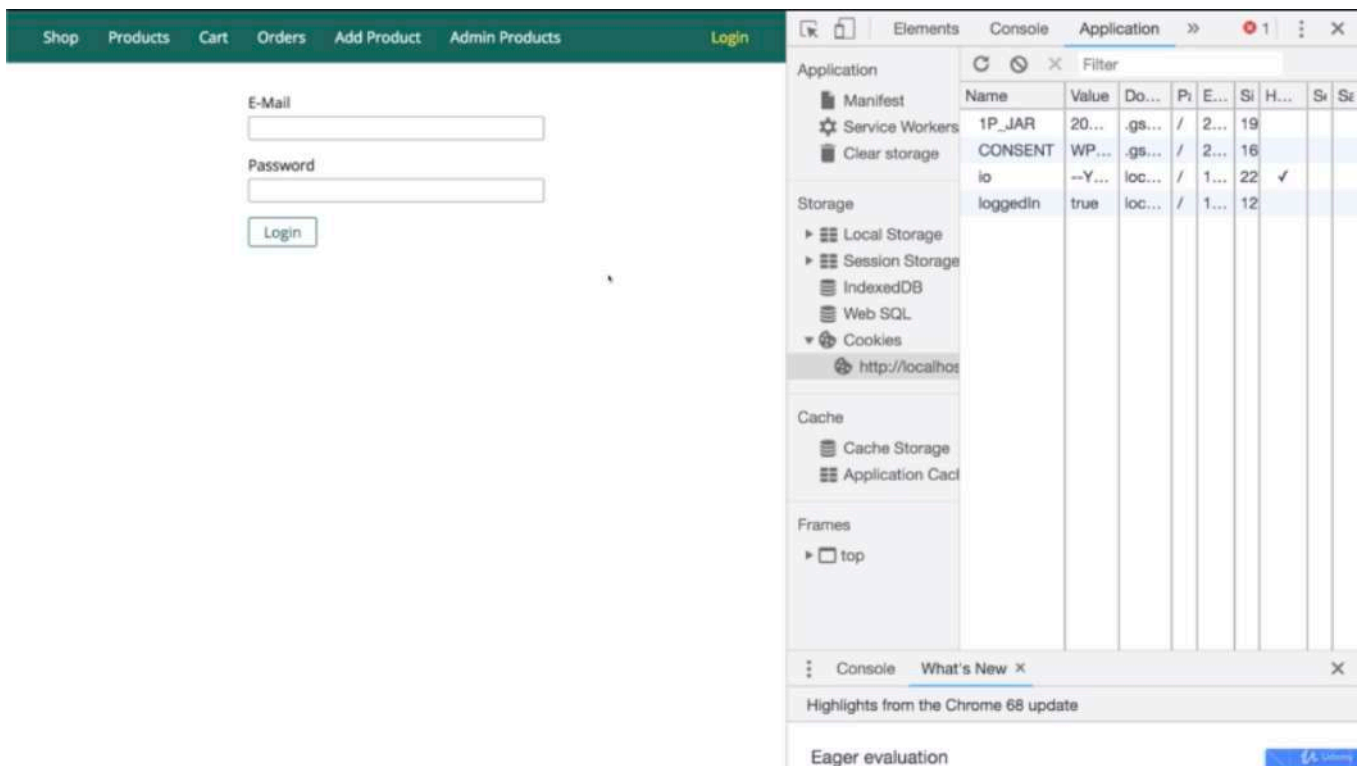
Password

Login

Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
http://localhost:3000
Cache
Cache Storage
Application Cache
Frames
top

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	Session
1P_JAR	20...	.gs...	/	2...	19			
CONSENT	WP...	.gs...	/	2...	16			
io	--Y...	loc...	/	1...	22	✓		
loggedIn	true	localho	/	1...	12			

Console
What's New
Highlights from the Chrome 68 update
Eager evaluation



- and if i change it back to true, and i reload, i am.
 - so the issue here is we can manipulate that from inside the browser and obviously you don't wanna allow the user of your website to login by simply manipulating some cookie value.
 - whilst it's interesting to store some data in the client side, especially things that are related to tracking users, advertisements tracking and so on. whilst this is interesting, sensitive data should not be stored in the browser because users can edit then as you see. we can edit our logged in cookie. so whilst cookies are generally good thing for storing data across requests, but it might not be the best approach in all scenarios and that's exactly something where sessions can help us with. however before we dive into sessions, let me explain you some other fields you can configure about a cookie which will also highlight when a cookie does make sense to be used.
- before we then dive into the scenario where it's not the best tool.

```

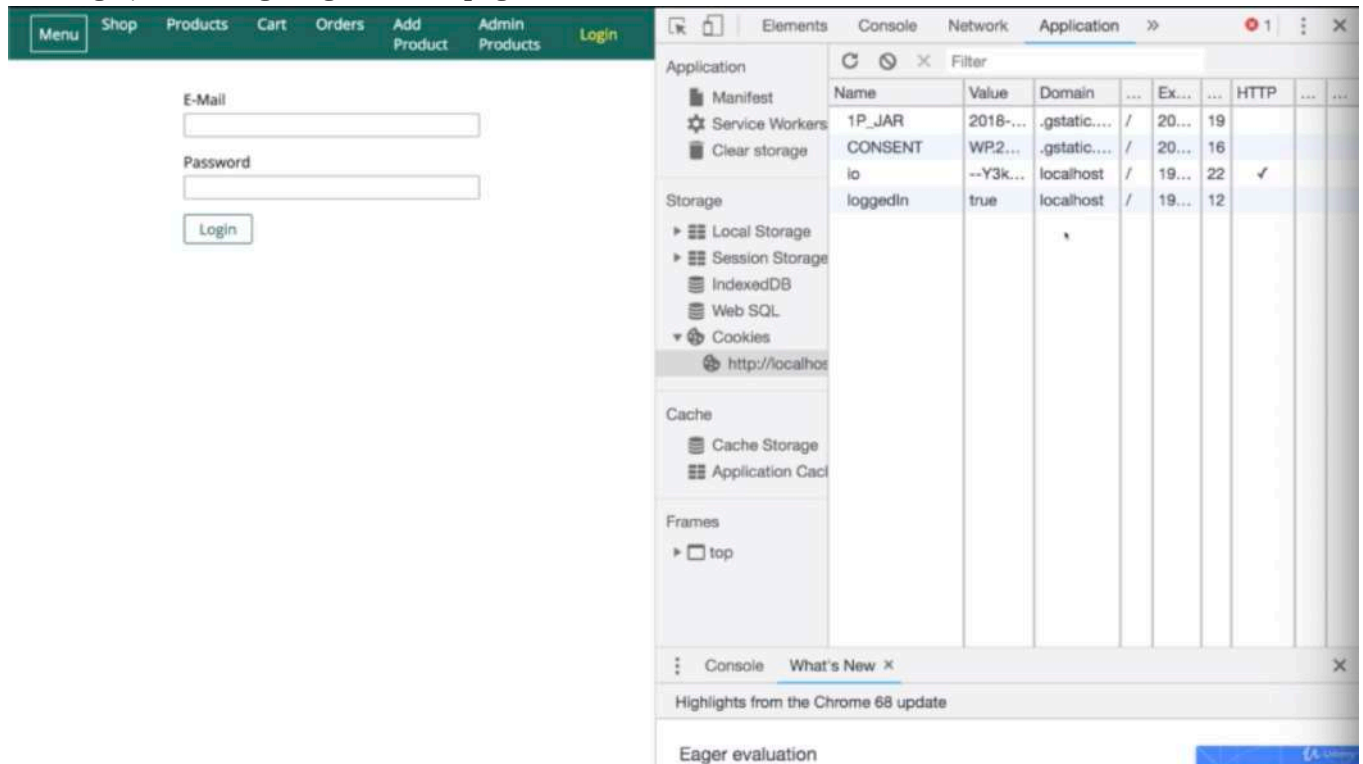
1 //./controllers/auth.js
2
3 exports.getLogin = (req, res, next) => {
4   /**if that value is equal to true,
5    * so to the text true here.
6   */
7   const isLoggedIn = req.get('Cookie')
8     .split(';')[1]
9     .trim()
10    .split('=')[1] === 'true'
11   res.render('auth/login', {
12     path: '/login',
13     pageTitle: 'Login',
14     isAuthenticated: isLoggedIn
15   });
16 };
17
18 exports.postLogin = (req, res, next) => {
19   res.setHeader('Set-Cookie', 'loggedIn=true')
20   res.redirect('/')
21 };

```

* Chapter 233: Configuring Cookies

1. update

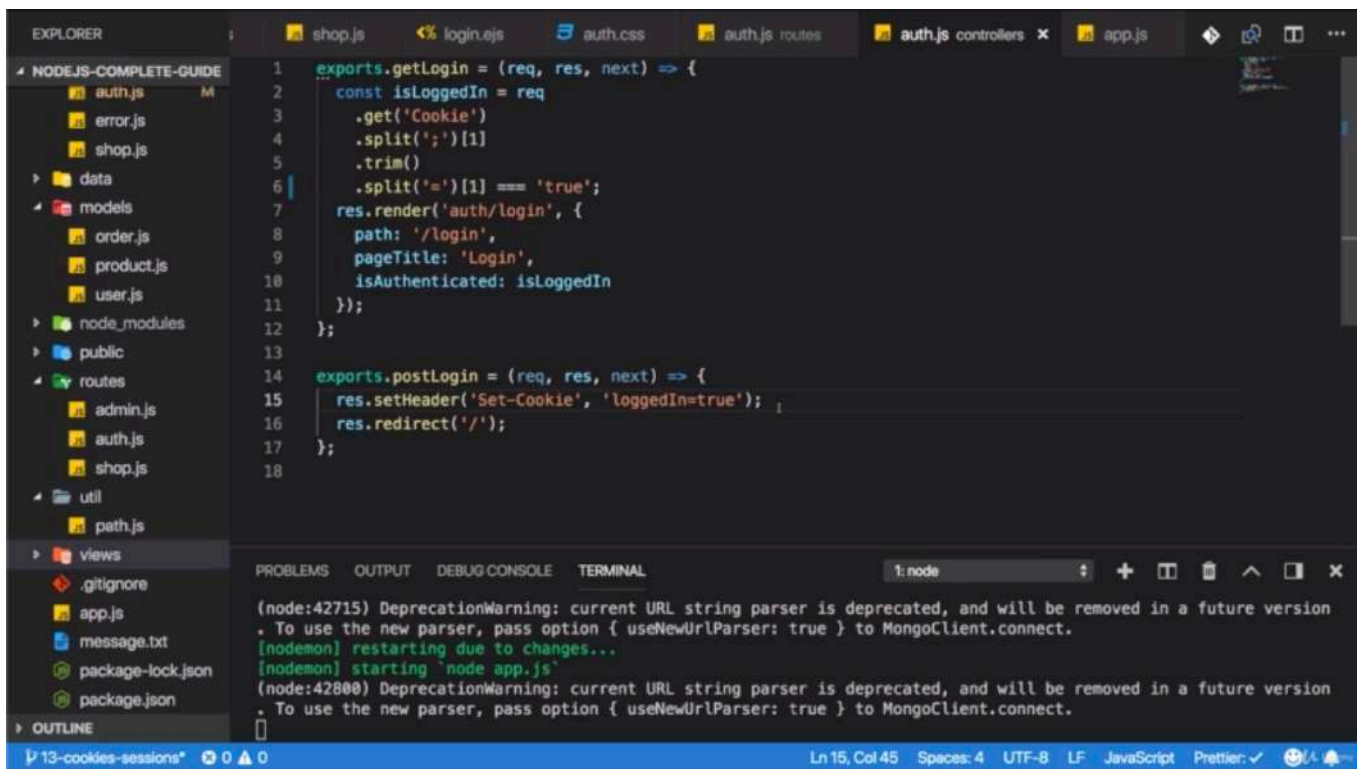
- ./controllers/auth.js



- so we manipulate cookies. so storing sensitive data is not ideal but for example for tracking users, it's a popular instrument and why is that? because as you can see with the cookies i have here, the cookies don't only have to relate to your page. a cookie can also be sent to another page and that is a common instrument in tracking where you have that so-called 'tracking pixel on pages' which is an imageUrl with no real image.

- but that image can be located on, for example, google's server and you have a cookie on that page which is also sent along with that and therefore google can track on which page you are and how you are moving through the web even if you are not on their website because some data is stored in your client and you could delete it which is why you can block such mechanisms too.

- but it's stored there and it's sent with every request to google. so they can track you without you being on that server, so storing that information on their servers would not work but storing it on your computer will work because that can be sent on every page you visit. so that is something where this could be interesting if you wanna track your users, that is common things to do.

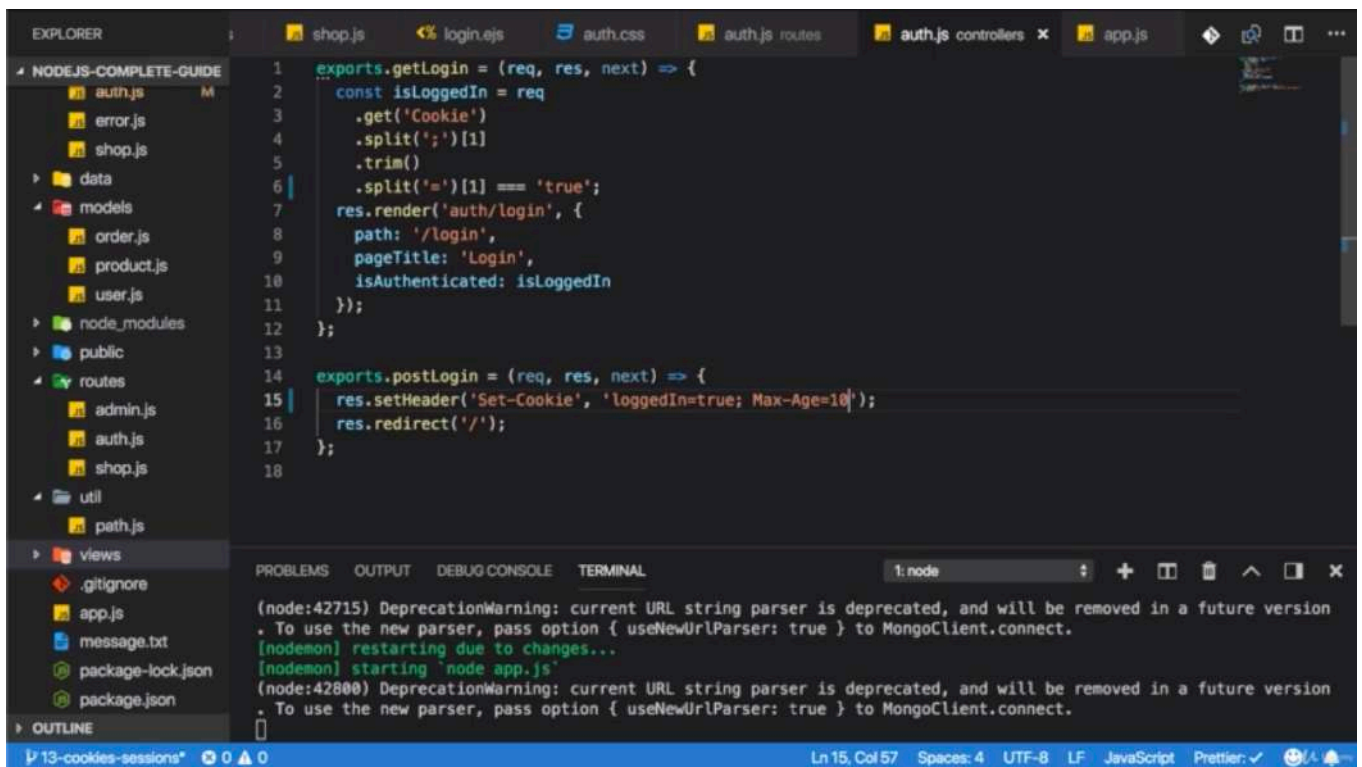


```
1 exports.getLogin = (req, res, next) => {
2   const isLoggedIn = req
3     .get('Cookie')
4     .split(';')[1]
5     .trim()
6     .split('=')[1] === 'true';
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    isAuthenticated: isLoggedIn
11  });
12 };
13
14 exports.postLogin = (req, res, next) => {
15   res.setHeader('Set-Cookie', 'loggedIn=true');
16   res.redirect('/');
17 };
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

(node:42715) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:42800) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.



```
1 exports.getLogin = (req, res, next) => {
2   const isLoggedIn = req
3     .get('Cookie')
4     .split(';')[1]
5     .trim()
6     .split('=')[1] === 'true';
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    isAuthenticated: isLoggedIn
11  });
12 };
13
14 exports.postLogin = (req, res, next) => {
15   res.setHeader('Set-Cookie', 'loggedIn=true; Max-Age=10');
16   res.redirect('/');
17 };
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

(node:42715) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:42800) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

- we could set 'Max-Age=10' for example. now if i click that login button here, i got 'loggedIn' and you see the expire date also changed, the expiry date. the expiry date is today and now it already is expired

The screenshot shows a web application with a navigation bar (Menu, Shop, Products, Cart, Orders, Login) and two product cards. The first card, 'A nice Book', features an image of an open book, a price of \$29.99, and a message 'You should not miss that!'. The second card, 'Second Product', shows a price of \$1222 and the text 'fdasfsa'. Both cards have 'Details' and 'Add to Cart' buttons.

The Chrome DevTools Application tab is open, displaying the 'Cookies' section for the URL 'http://localhost'. The cookies table is as follows:

Name	Value	Domain	Expires / Max-Age	Size	HTTP	Secure	SameSite
1P_JAR	2018...	.gsta...	/ 2018-10-24T12:06:03.022Z	19			
CONSENT	WP2...	.gsta...	/ 2038-01-01T00:00:01.022Z	16			
io	--Y3...	local...	/ 1989-12-31T23:59:59.000Z	22	✓		
loggedIn	true	local...	/ 2018-09-27T13:23:16.395Z	12			

and if i reload that page, 'loggedIn' is gone. so this is something we can do.

This screenshot shows the same web application after a page reload. The product cards remain the same. However, the Chrome DevTools Application tab shows that the 'loggedIn' cookie has been removed from the cookies table. The remaining cookies are:

Name	Value	Domain	Expires / Max-Age	Size	HTTP	Secure	SameSite
1P_JAR	201...	.gst...	/ 2018-10-24T12:06:03.022Z	19			
CONSENT	WP...	.gst...	/ 2038-01-01T00:00:01.022Z	16			
io	--Y3...	local...	/ 1989-12-31T23:59:59.000Z	22	✓		

The screenshot shows a web application with a login form and the Chrome DevTools Application tab. The login form has fields for "E-Mail" and "Password" and a "Login" button. The Application tab shows the "Consent" cookie set for the "io" domain.

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19			
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16			
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓		

The screenshot shows a web application with a product page and the Chrome DevTools Application tab. The product page displays two products: "A nice Book" for \$29.99 and "Second Product" for \$1222. The Application tab shows the "Consent" cookie set for the "io" domain.

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19			
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16			
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓		

- let's comment this out and set 'isAuthenticated' always to false temporarily because of showing you how this cookie is now set.

- if i reload and i click here, you don't see the cookie here because i added 'Secure' and it would only be set if we are serving the page via https.

EXPLORER

NODEJS-COMPLETE-GUIDE

auth.js

error.js

shop.js

data

models

order.js

product.js

user.js

node_modules

public

routes

admin.js

auth.js

shop.js

util

path.js

views

gitignore

app.js

message.txt

package-lock.json

package.json

OUTLINE

shop.js

login.js

auth.css

auth.js routes

auth.js controllers

app.js

```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    isAuthenticated: false
11  });
12 };
13
14 exports.postLogin = (req, res, next) => {
15   res.setHeader('Set-Cookie', 'loggedIn=true; HttpOnly');
16   res.redirect('/');
17 };
18
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

1: node

at /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/express/lib/router/index.js:635:15
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:42978) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

13-cookies-sessions*

0

0

Ln 15, Col 55

Spaces: 4

UTF-8

LF

JavaScript

Prettier: ✓

Menu

Shop

Products

Cart

Orders

Login

E-Mail

Password

Login

Application

Manifest

Service Workers

Clear storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

http://localhost

Cache

Cache Storage

Application Cache

Frames

top

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓	

Console

What's New

Highlights from the Chrome 68 update

Eager evaluation

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1969-12-31T23:59:59.000Z	22	✓	
loggedIn	true	local...	/	1969-12-31T23:59:59.000Z	12	✓	

- and you can also set this to 'httpOnly' and if i reload and login, it's set but now it has this checkmark in the http column. and that means that now we can't access the cookie value through client side javascript. so in the scripts running in the browser, this can be an important security mechanism because it protects us against cross-site scripting attacks now. because your client side javascript where someone could have injected malicious code can't read your cookie values and that will be important later with authentication where a cookie will not store the sensitive information but an important part of authenticating the user. so this can be an extra security layer because the cookie will still be attached to every request that is sent to the server, but you can't read the cookie value from inside the browser javascript code.

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1969-12-31T23:59:59.000Z	22	✓	
loggedIn	true	local...	/	1969-12-31T23:59:59.000Z	12	✓	

- as a user in the development tools, you can still read it, but it's your own cookie and you will not store information like hey i'm logged in there because that would be easy to manipulate and you can't protect against that.

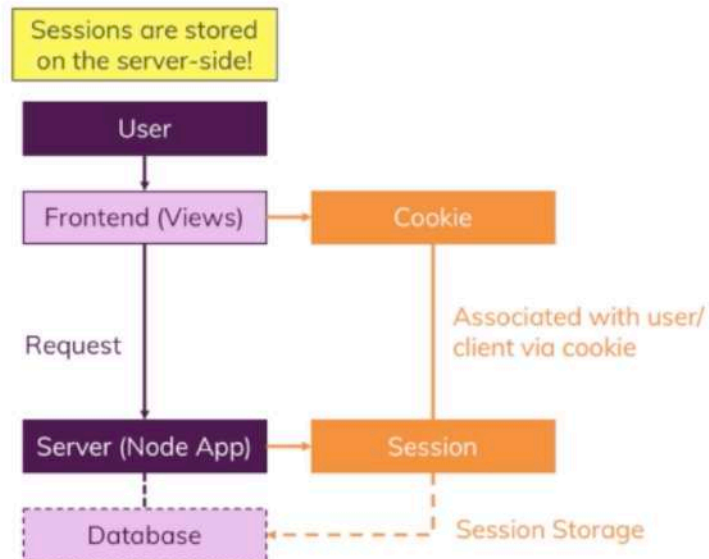
- often you will not directly set your cookies because you rather use some package like, for example, for authentication that will manage setting the cookie for you.

```
1  //./controllers/auth.js
2
3  exports.getLogin = (req, res, next) => {
4      const isLoggedIn = req.get('Cookie')
5          .split(';')[1]
6          .trim()
7          .split('=')[1] === 'true'
8      res.render('auth/login', {
9          path: '/login',
10         pageTitle: 'Login',
11         isAuthenticated: isLoggedIn
12     });
13 };
14
15 exports.postLogin = (req, res, next) => {
16     /**you can also configure cookies.
17      * we set a value but you can set more things than just the value.
18      * here i set my cookie by adding the key-value pair 'loggedIn=true'. we could add
19      * multiple cookies,
20      *
21      * we can also add a semi-colon ; after the key-value pair
22      * and for example set expire to some expiration date,
23      * this date would have to follow a certain format,
24      * the http date format.
25      * you could set a certain data when this cookie will 'Expire'
26      * because remember if you don't set this,
27      * it will expire once you close your browser,
28      *
29      * alternatively to 'Expires'
30      * you can set 'Max-Age' which is a number in seconds,
31      * how long that cookie should stay around
32      * we could set this to '10' for example
33      *
34      * and if you wanna control, for example, how long you wanna track a user
35      * or we will use that together with authentication later,
36      * you could use this to control how long an authenticated session stays active for a
37      * user.
38      * you might know that from your online back where you timeout after a certain duration
39      *
40      * you can add a domain to which the cookie should be sent.
41      * and we again are on that tracking thing again.
42      *
43      * you can add 'Secure' without an equal sign, just 'Secure'
44      * this means this cookie will only be set if the page is served via https
45      * i can't demonstrate this because our local development server is not using https
46      * but we will eventually https later in the course.
47      *
48      * */
49     res.setHeader('Set-Cookie', 'loggedIn=true; HttpOnly')
50     res.redirect('/')
51 };
52
```

* Chapter 234: What Is A Session?



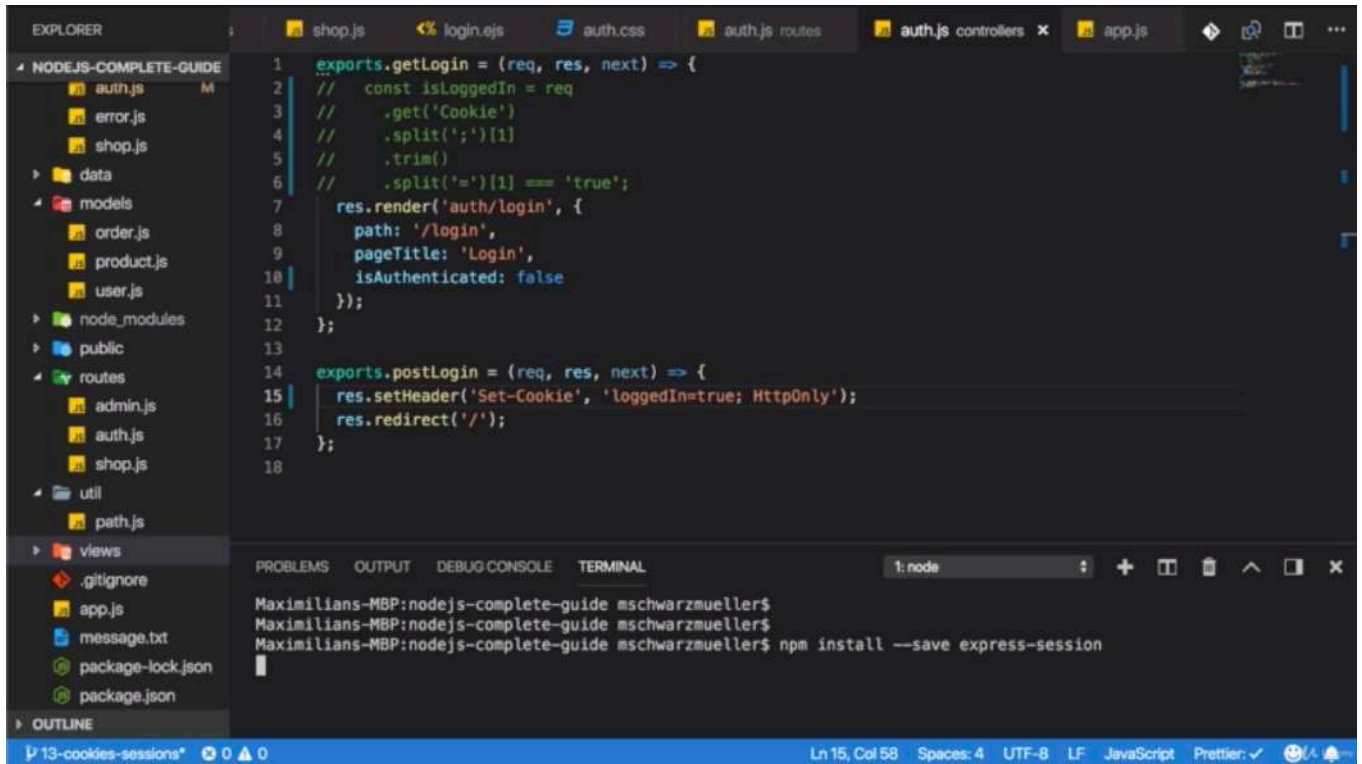
What's a Session?



- instead of storing the information that the user is authenticated in the frontend which was a bad place, we will store it in the backend with a so-called 'session'
- we only wanna share the information across all requests of the same user and that's really important. the same user so that other users can't see your data. for that we need to store it on the server. we will start by storing it in memory which is then similar to storing in that variable. but eventually we will move to a different session storage. the database
- and we need one important piece of information. a client needs to tell the server to which session he belongs because the session will in the end be an entry stored in memory or stored in a database.
- we are not matching this by IP address. because that is a bit hard to maintain and can be faked. instead we will use a cookie where we will store the ID of the session. now you can still change that and assume a different ID if you wanna but that will not work because the value we store will not be the ID but the hashed ID, hashed with a certain algorithm where only the can confirm that it hasn't been fiddled with so that you didn't play around with it.
- so this will be secure way because you store the ID in an encrypted way where only the server is able to confirm that stored cookie value relate to a certain ID in the database and therefore we got a safe value stored in the cookie which you can't. you can change it but you will not assume a different session.
- a session can be matched and that session can then contain the confidential data which you can't change from inside the browser.

* Chapter 235: Initializing The Session Middleware

1. update
- app.js



The screenshot shows a VS Code editor with the 'login.ejs' file open. The file contains two Express.js route handlers. The first, 'getLogin', checks if a user is logged in by looking at a cookie and renders the 'auth/login' view. The second, 'postLogin', sets a cookie 'loggedIn=true' and redirects the user to the root path. The left sidebar shows the project structure, and the bottom panel shows the terminal with the command 'npm install --save express-session' being executed.

```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   res.render('auth/login', {
8     path: '/login',
9     pageTitle: 'Login',
10    isAuthenticated: false
11  });
12 };
13
14 exports.postLogin = (req, res, next) => {
15   res.setHeader('Set-Cookie', 'loggedIn=true; HttpOnly');
16   res.redirect('/');
17 };
18
```

Terminal output:

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

- 'npm install --save express-session' is a package which is part of the express.js suite but not baked into express.js itself.

```
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 /**we wanna initialize that session early on.
9  * when we start up our server then we wanna initialize the session middleware at least
10  * and the session will then be used for every incoming request.
11  */
12 const session = require('express-session')
13
14 const errorController = require('./controllers/error');
15 const User = require('./models/user')
16
17 const app = express();
18
19 app.set('view engine', 'ejs');
20 app.set('views', 'views');
21
22 const adminRoutes = require('./routes/admin');
23 const shopRoutes = require('./routes/shop');
24 const authRoutes = require('./routes/auth');
25
26 app.use(bodyParser.urlencoded({ extended: false }));
27 app.use(express.static(path.join(__dirname, 'public')));
28 /**we pass a javascript object where we configure the session setup
29  * 'secret' will be used for signing the hash which secretly stores our ID in the cookie
30  * in production, this should be a long string value
31  *
```

```

32 * 'resave: false' means that the session will not be saved on every request that is done.
33 * so on every response that is sent, but only if something changed in the session.
34 * this will obviously improve performance and so on.
35 *
36 * 'saveUninitialized' value which you should set to false
37 * because this will also basically ensure that no session gets saved for a request where it
  doesn't need to be saved
38 * because nothing was changed about it.
39 *
40 * this is the core things you need to set.
41 * you could, for example, also configure the session cookie,
42 * you could give it a 'maxAge' by setting a date
43 * or add the 'expires' key
44 *
45 * anyway middleware is initialized and we are now ready to use the session.
46 */
47 app.use(session({secret: 'my secret', resave: false, saveUninitialized: false}))
48
49 app.use((req, res, next) => {
50   User.findById('5cbb2b2c80bd7193adb9eeeb')
51     .then(user => {
52       req.user = user
53       next();
54     })
55     .catch(err => console.log(err));
56 });
57
58 app.use('/admin', adminRoutes);
59 app.use(shopRoutes);
60 app.use(authRoutes)
61
62 app.use(errorController.get404);
63
64 mongoose
65   .connect('mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-z3v1k.mongodb.net/shop?
  retryWrites=true')
66   .then(result => {
67     User
68       .findOne()
69       .then(user => {
70         if(!user){
71           const user = new User({
72             name: 'Max',
73             email: 'max@test.com',
74             cart: {
75               items: []
76             }
77           })
78           user.save()
79         }
80       })
81     app.listen(3000)
82   })
83   .catch(err => {
84     console.log(err)
85   })

```

* Chapter 236: Using The Session Middleware

1. update
- ./controllers/auth.js

The screenshot shows a web application with a navigation bar (Menu, Shop, Products, Cart, Orders, Login) and a login form with fields for E-Mail and Password, and a Login button. The Chrome DevTools Application tab is open, showing the 'loggedIn' cookie set to 'true'.

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓	
loggedIn	true	local...	/	1989-12-31T23:59:59.000Z	12	✓	

The screenshot shows the same web application after reloading. The 'loggedIn' cookie has been removed from the Application tab.

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓	

- if i reload and get rid of 'loggedIn' cookie.

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓	

Name	Value	Domain	Path	Expires / Max-Age	Secure	HTTP	SameSite
1P_JAR	201...	.gst...	/	2018-10-24T12:06:03.022Z	19		
CONSENT	WP...	.gst...	/	2038-01-01T00:00:01.022Z	16		
connect...	s%3...	local...	/	1989-12-31T23:59:59.000Z	91	✓	
io	--Y3...	local...	/	1989-12-31T23:59:59.000Z	22	✓	

- now if i click 'login' and what you should see is that a new cookie 'connect.sid' was added for a session id cookie.
 - and you will see some strange string in 'Value' and that is what i meant. this is encrypted value. this is now the cookie. by default this is a session cookie so it will expire when you close the browser.
 - it's session cookie that will identify your user here, your running instance of this website where you are browsing around, this will identify you to the server and to the session.
-
-
-

EXPLORER

shop.js login.ejs auth.css auth.js routes auth.js controllers app.js

NODEJS-COMPLETE-GUIDE

.vscode

controllers

admin.js

auth.js

error.js

shop.js

data

models

order.js

product.js

user.js

node_modules

public

routes

admin.js

auth.js

shop.js

util

path.js

views

.gitignore

app.js

OUTLINE

```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 };
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

(node:43296) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:43387) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

Ln7, Col 28 Spaces: 4 UTF-8 LF JavaScript Prettier: ✓

Menu Shop Products Cart Orders Login

E-Mail

Password

Login

Application

Manifest

Service Workers

Clear storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

http://localhost

Cache

Cache Storage

Application Cache

Frames

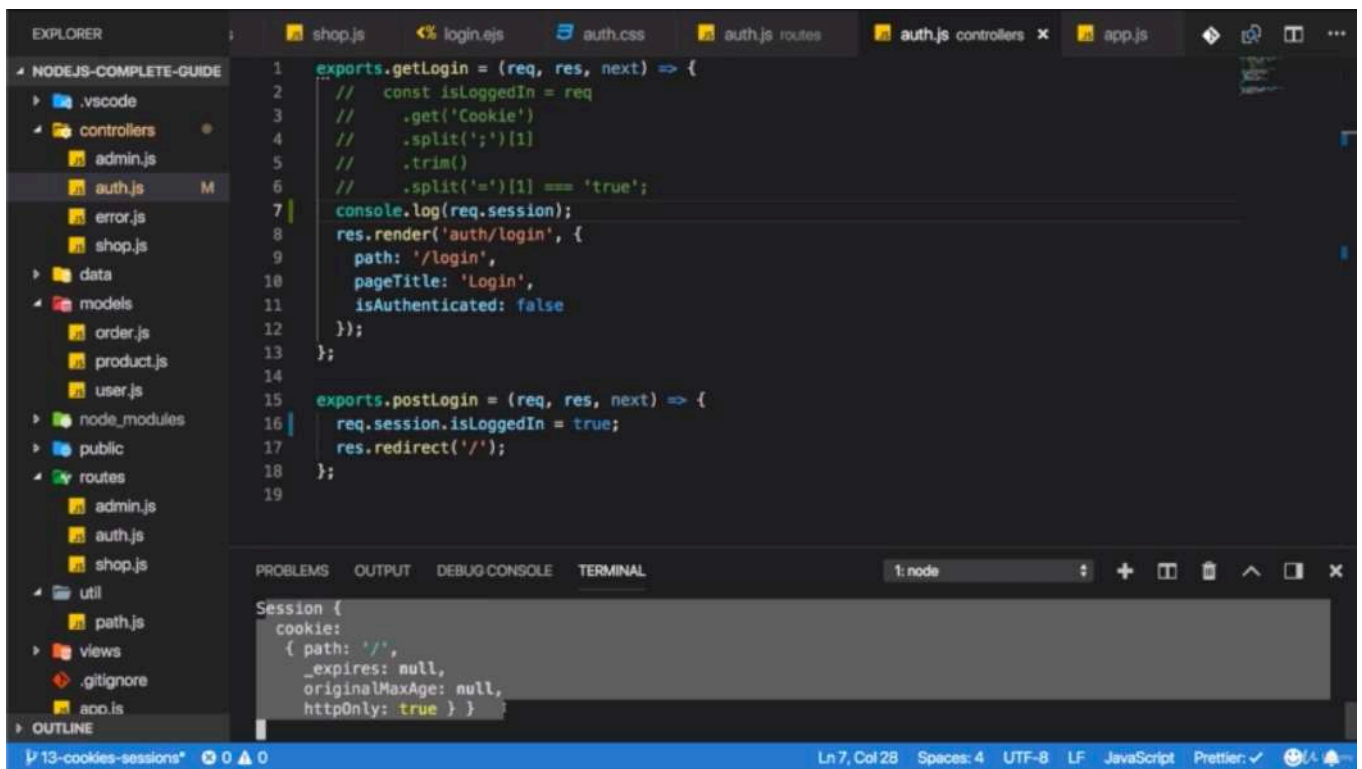
top

Console What's New

Highlights from the Chrome 68 update

Eager evaluation

Name	Value	Domain	Path	Expires / ...	S...	HTTP	Se...	Sa...
1P_JAR	2018-09-24-12	.gstati...	/	2018-10-2...	19			
CONS...	WP.271ba8	.gstati...	/	2038-01-0...	16			
connec...	s%3AU4O52x9zs_aG...	localh...	/	1969-12-3...	91	✓		
io	--Y3kAezJV_Mbr54AA...	localh...	/	1969-12-3...	22	✓		



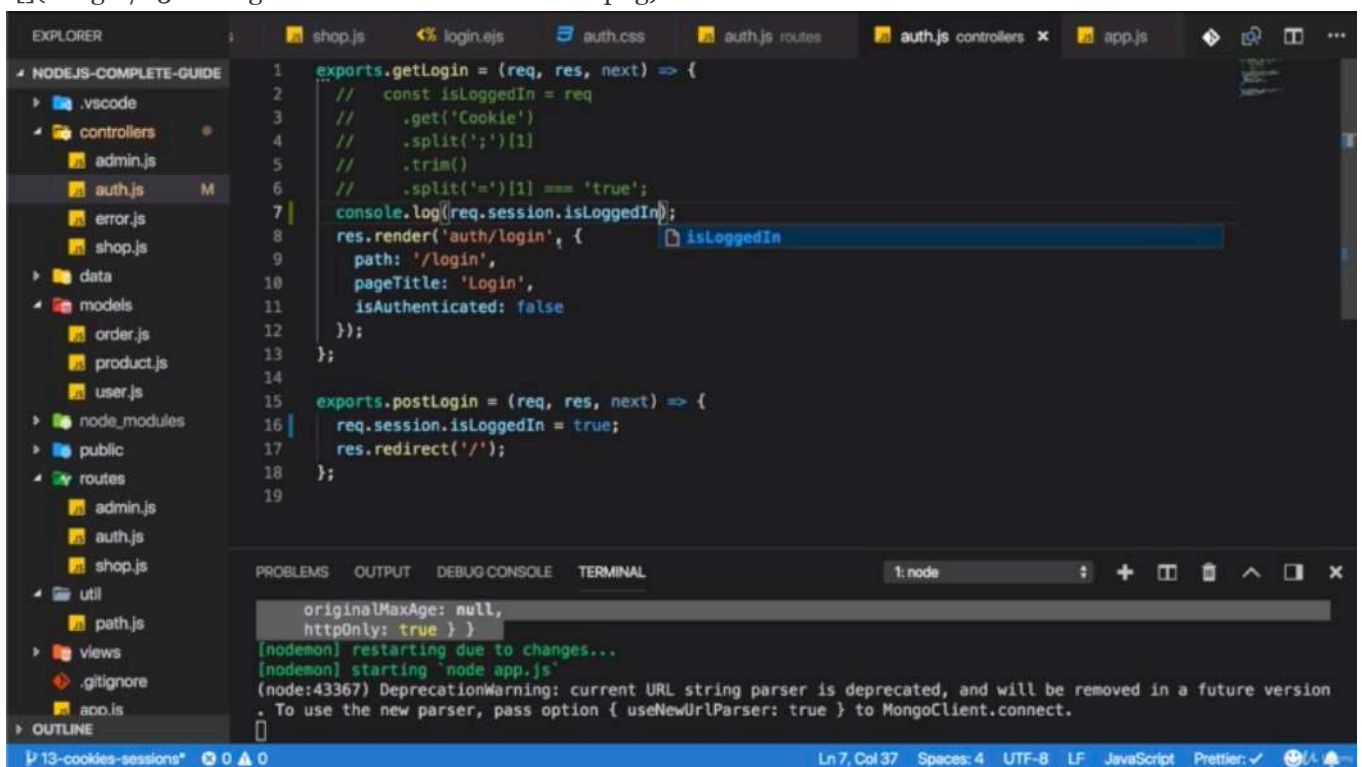
```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 };
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
Session {
  cookie:
    { path: '/',
      _expires: null,
      originalMaxAge: null,
      httpOnly: true } }
```

- go to login page and go back to our server and there you see the session object is logged.

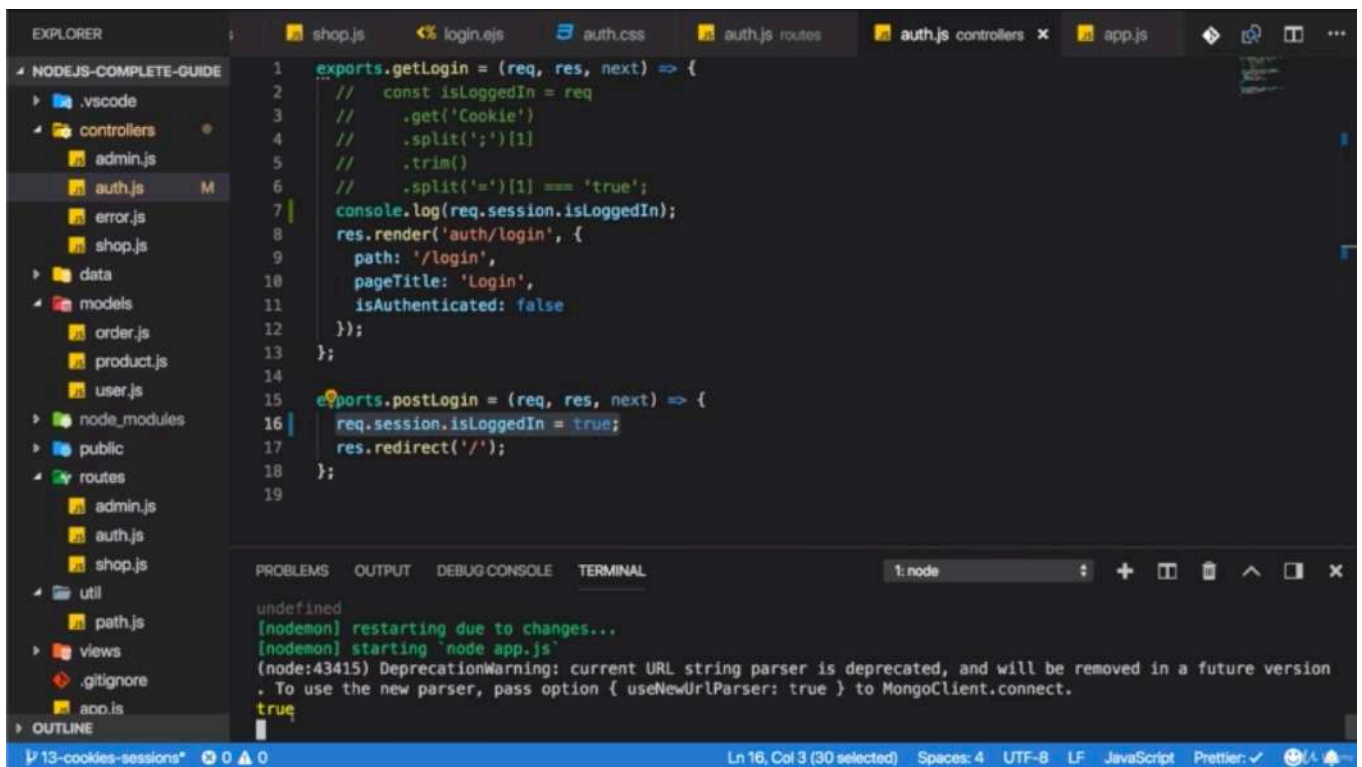
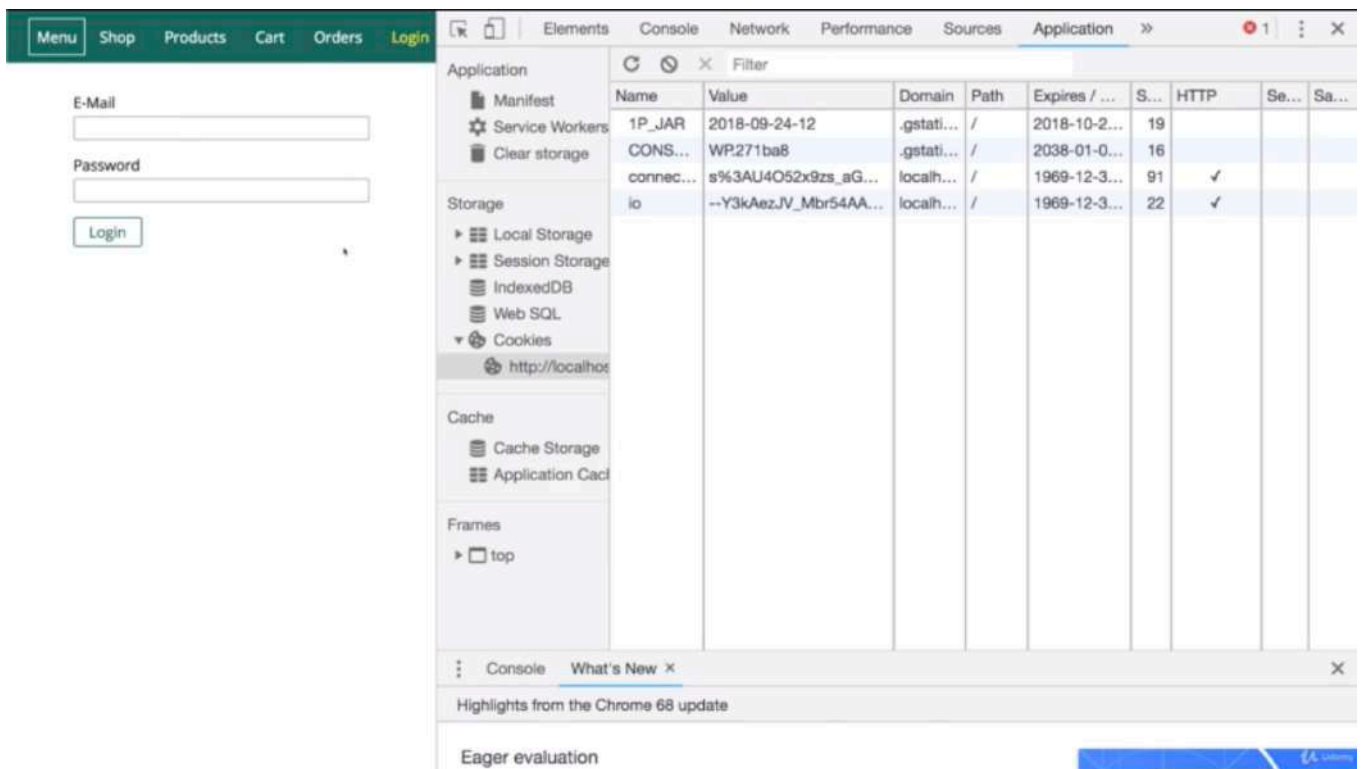


```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session.isLoggedIn);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 };
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
originalMaxAge: null,
httpOnly: true } }
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
(node:43367) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
```



- but if i edit this and go to login page and go back to our server, you see true here because in the session 'isLoggedIn' is stored.

The top part of the image shows a web application with a navigation bar (Menu, Shop, Products, Cart, Orders, Login) and two product cards. The first card, 'A nice Book', features an image of an open book, a price of \$29.99, and a 'You should not miss that!' message. The second card, 'Second Product', has a placeholder image, a price of \$1222, and the text 'fdasfsa'. Both cards have 'Details' and 'Add to Cart' buttons.

The bottom part of the image shows the Chrome DevTools Application tab. The left sidebar lists various storage areas: Manifest, Service Workers, Clear storage, Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Application Cache), and Frames. The main pane displays a table of cookies:

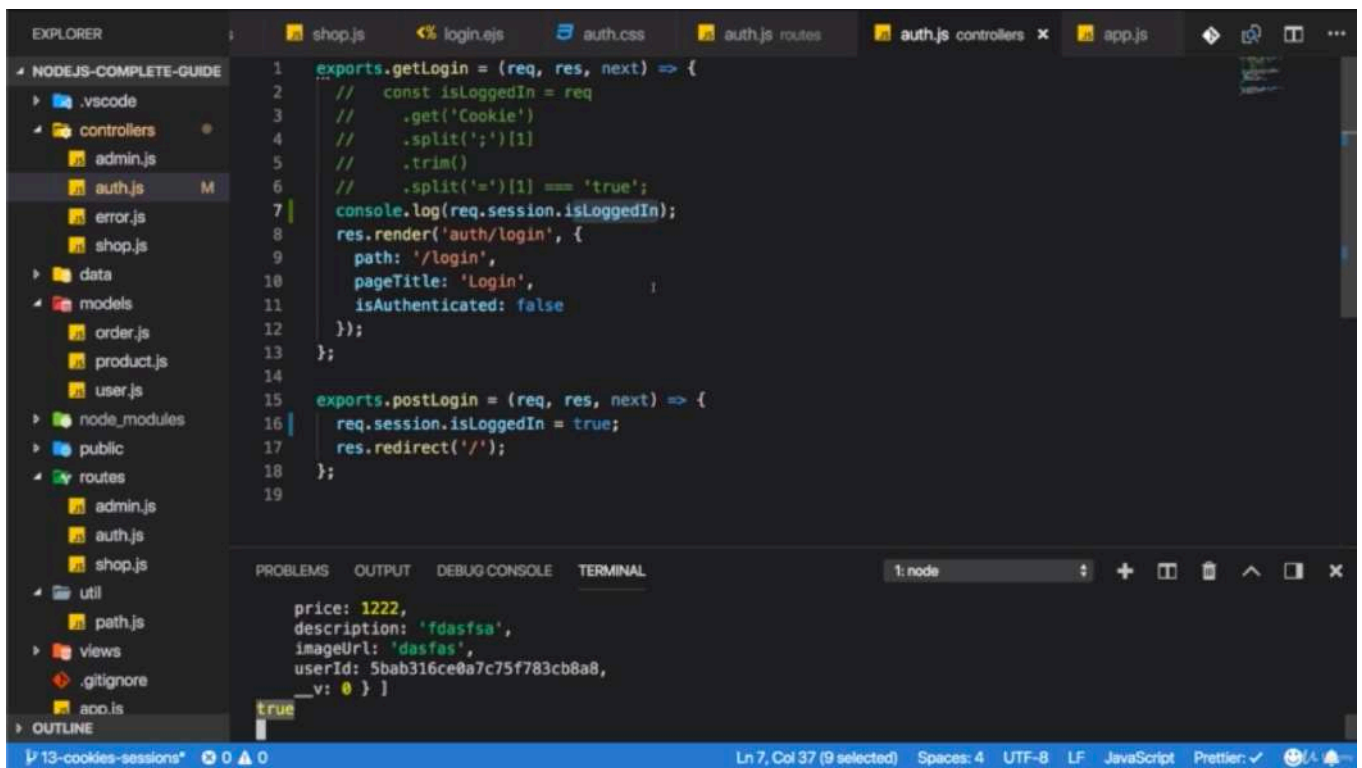
Name	Value	Domain	Path	Expires / ...	S...	HTTP	Se...	Sa...
1P_JAR	2018-09-24-12	.gstati...	/	2018-10-2...	19			
CONS...	WP.271ba8	.gstati...	/	2038-01-0...	16			
connec...	s%3A4ddS8HcThFOK...	localh...	/	1969-12-3...	93	✓		
io	--Y3kAez.JV_Mbr54AA...	localh...	/	1969-12-3...	22	✓		

Below the table, there are sections for 'Console' (What's New), 'Highlights from the Chrome 68 update', and 'Eager evaluation'.

The top part of the image shows a web application with a navigation bar (Menu, Shop, Products, Cart, Orders, Login) and a login form. The form has fields for 'E-Mail' and 'Password', and a 'Login' button.

The bottom part of the image shows the Chrome DevTools Application tab, which is identical to the one in the first screenshot, displaying the same storage areas and cookie table.

- and we can go to different page and com back to login and these are all individual requests which technically are totally individual from each other, totally separated

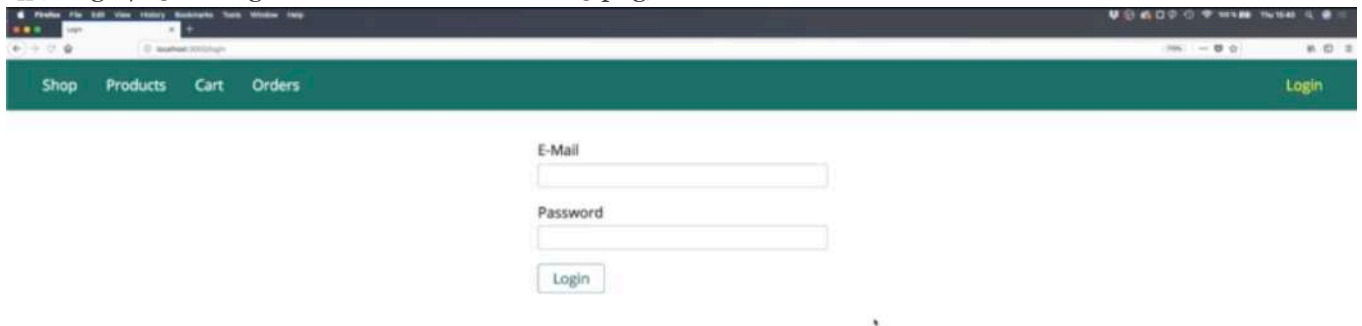


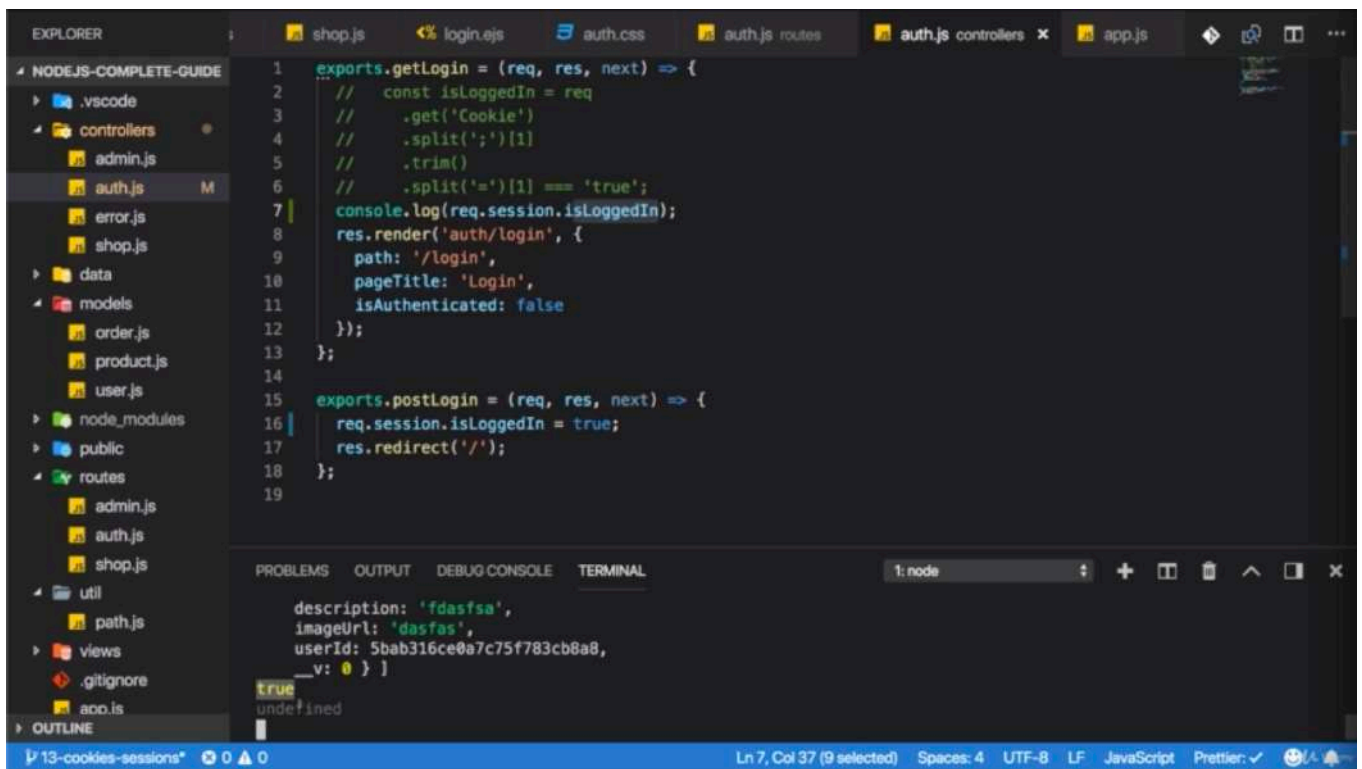
```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session.isLoggedIn);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 }
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 }
19
```

price: 1222,
description: 'fdasfsa',
imageUrl: 'dasfas',
userId: 5bab316ce0a7c75f783cb8a8,
_v: 0 }]

- and still we see true here because we still store this in the session on the server side by default, just in the memory not in the database yet.

- and session is identified for this browser because we have that cookie. and i can prove that to you by starting another browser and this will technically be treated as a totally different session and environment, could be a totally different machine.



A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project structure with folders like 'controllers', 'data', 'models', 'routes', and 'util'. The 'auth.js' file in the 'controllers' folder is selected and open in the main editor. The code in 'auth.js' defines two Express.js routes: 'getLogin' and 'postLogin'. The 'getLogin' route checks for a 'Cookie' in the request, splits it by semicolon, trims the result, and checks if it equals 'true'. If so, it logs the session's 'isLoggedIn' status and renders the 'auth/login' page with 'path: '/login'', 'pageTitle: 'Login'', and 'isAuthenticated: false'. The 'postLogin' route sets 'req.session.isLoggedIn = true' and redirects to '/'. Below the editor, the 'TERMINAL' tab shows the output of a Node.js process, displaying a JSON object: {description: 'fdasfsa', imageUrl: 'dasfas', userId: '5bab316ce0a7c75f783cb8a8', __v: 0}. The status bar at the bottom indicates the file is at line 7, column 37, and is using UTF-8 encoding and the Prettier formatter.

- so if i click login here in different browser firefox, you see undefined and undefined is coming from the login request i sent. because this browser, this user, technically this is a totally different user even though i'm the same but it's a different browser, different user, this user doesn't have this cookie set for him, doesn't have an active session on the server.
- and this is how we can store data that persists across requests
- it still needs a cookie to identify the user but the sensitive information is stored on the server. we can't modify it.
- what we see already is the core mechanism behind authenticating users in the web.
- there's another examples

```
1 //./controllers/auth.js
2
3 exports.getLogin = (req, res, next) => {
4   /*
5     const isLoggedIn = req.get('Cookie')
6     .split(';')[1]
7     .trim()
8     .split('=')[1] === 'true'
9   */
10   console.log(req.session.isLoggedIn)
11   res.render('auth/login', {
12     path: '/login',
13     pageTitle: 'Login',
14     isAuthenticated: false
15   });
16 };
17
18 exports.postLogin = (req, res, next) => {
19   User.findById('5cbb2b2c80bd7193adb9eeeb')
20     .then(user => {
21       /**instead of cookies,
22        * request and 'session' object which is added by session middleware,
23        * in here we can add any key we want. for example 'isLoggedIn'
24        * but you can name this however you want
25        * and set this 'true'
```



```

26 */
27     req.session.isLoggedIn = true;
28     req.session.user = user;
29     res.redirect('/');
30 }
31 .catch(err => console.log(err));
32 };
33

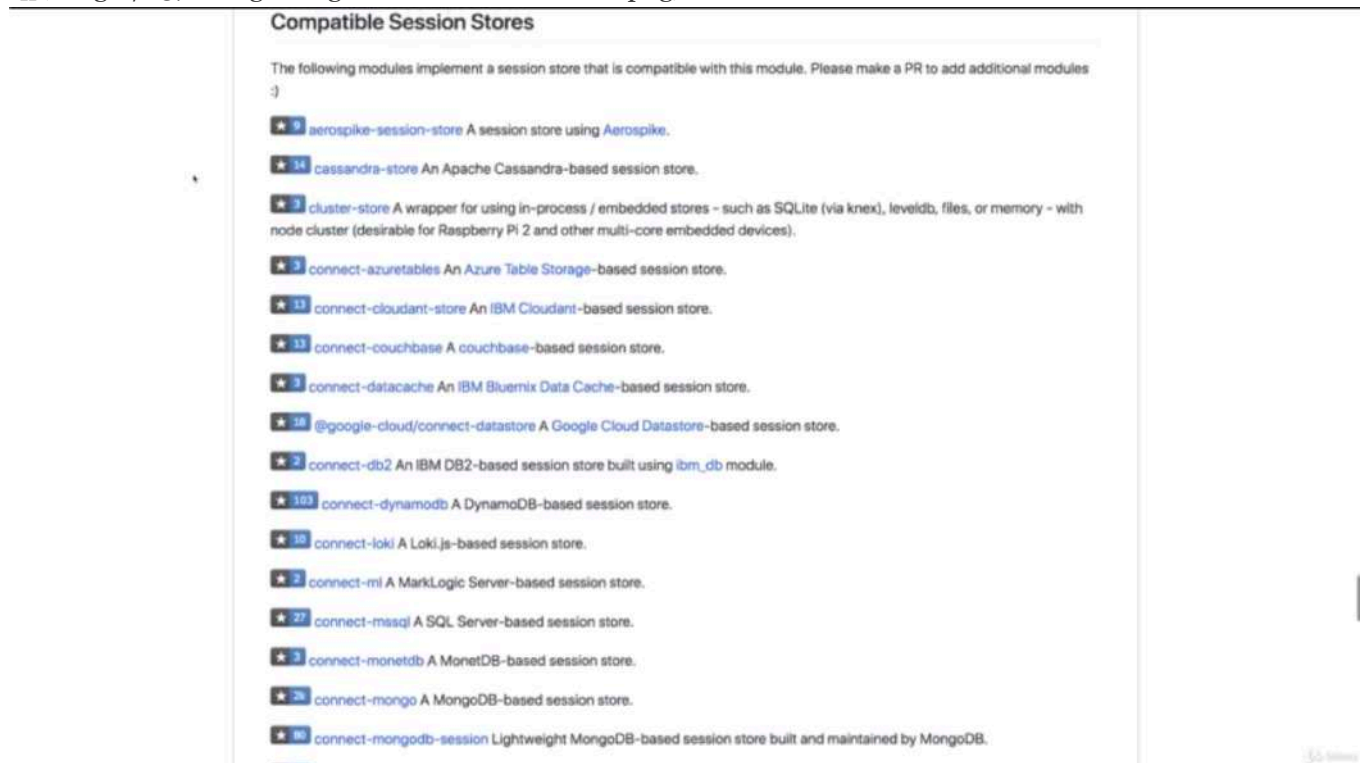
```

* Chapter 237: Using MongoDB To Store Sessions

1. update
- app.js

- The problem is this session is stored in memory and memory is not an infinite resource. so for development, this is fine but for a production server, this would be horrible because if you have thousands of users, your memory will quickly overflow if you store all that information in memory.

- so we wanna store sessions differently.



- in express github directory, you will find a list of session stores you can use and all kinds of database are supported. you could store it in files though that might not give you the best performance.

Need Private Modules

[npm Enterprise](#)
[Features](#)
[Pricing](#)
[Docs](#)
[Support](#)

Search packages

log in or sign up

Share your code. npm Orgs help your team discover, share, and reuse code. [Create a free org >](#)

connect-mongodb-session

2.0.3 • Public • Published 4 months ago

Readme

1 Dependencies

43 Dependents

22 Versions

connect-mongodb-session

MongoDB-backed session storage for [connect](#) and [Express](#). Meant to be a well-maintained and fully-featured replacement for modules like [connect-mongo](#)

build

passing

coverage

100%

MongoDBStore

This module exports a single function which takes an instance of [connect](#) (or [Express](#)) and returns a `MongoDBStore` class that can be used to store sessions in MongoDB.

It can store sessions for Express 4

If you pass in an instance of the [express-session](#) module the `MongoDBStore` class will enable you to store your Express sessions in MongoDB.

enable you to store your Express sessions in MongoDB.

The `MongoDBStore` class has 3 required options:

1. `uri`: a [MongoDB connection string](#).

2. `databaseName`: the MongoDB database to store sessions in

3. `collection`: the MongoDB collection to store sessions in

Note: You can pass a callback to the `MongoDBStore` constructor, but this is entirely optional. The [Express 3.x](#) example demonstrates that you can use the `MongoDBStore` class in a synchronous-like style: the module will manage the internal connection state for you.

```

var express = require('express');
var session = require('express-session');
var MongoDBStore = require('connect-mongodb-session')(session);

var app = express();
var store = new MongoDBStore({
  uri: 'mongodb://localhost:27017/connect_mongodb_session_test',
  collection: 'mySessions'
});

store.on('connected', function() {
  store.client; // The underlying MongoClient object from the MongoDB driver
});

// Catch errors
store.on('error', function(error) {
  assert.ifError(error);
  assert.ok(false);
});

```

install

> npm i connect-mongodb-session

weekly downloads

2,101

version

2.0.3

license

none

open issues

6

pull requests

1

homepage

github.com

repository

github

last publish

4 months ago

last publish

4 months ago

collaborators

Test with RunKit

Report a vulnerability

- and we will use MongoDB because we are already using that and for that, we will use the connect MongoDB session package here.


```

1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session.isLoggedIn);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 };
19

```

```

Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save connect-mongodb-session

```

- we will install this package now and register this as a store with which we can work.

The screenshot shows a web application with a login form on the left and the Chrome DevTools Application tab on the right. The login form has fields for "E-Mail" and "Password" and a "Login" button. The Application tab shows the "Cookies" section expanded, displaying a table of cookies.

Name	Value	Domain	Path	Expires / ...	S...	HTTP	Se...	Sa...
1P_JAR	2018-09-24-12	.gstati...	/	2018-10-2...	19			
CONS...	WP271ba8	.gstati...	/	2038-01-0...	16			
connec...	s%3A4ddS8HcThFOK...	localh...	/	1969-12-3...	93	✓		
io	--Y3kAezJV_Mbr54AA...	localh...	/	1969-12-3...	22	✓		

- if i click login, and i got a new session, a new session cookie and that session will now be stored in MongoDB

The top part of the image shows a web application interface with a navigation bar (Menu, Shop, Products, Cart, Orders, Login) and two product cards. The first card, 'A nice Book', features an image of an open book, a price of \$29.99, and a promotional message 'You should not miss that!'. The second card, 'Second Product', has a placeholder image, a price of \$1222, and the text 'fdasfsa'. Both cards have 'Details' and 'Add to Cart' buttons.

Below the web application is a screenshot of the Chrome DevTools Application tab. The left sidebar shows the storage hierarchy: Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Application Cache), and Frames (top). The main pane displays a table of cookies for the 'http://localhost' domain.

Name	Value	Domain	Path	Expires / ...	S...	HTTP	Se...	Sa...
1P_JAR	2018-09-24-12	.gstat...	/	2018-10-2...	19			
CONS...	WP271ba8	.gstat...	/	2038-01-0...	16			
connec...	s%3Axpt3lCqU3chAe...	localh...	/	1969-12-3...	95	✓		
io	--Y3kAezJV_Mbr54AA...	localh...	/	1969-12-3...	22	✓		

The bottom part of the image shows the MongoDB Compass interface. The left sidebar displays the database structure: My Cluster, Cluster0-shard-0, REPLICA SET, 3 NODES, 3 DBS, 10 COLLECTIONS, and a list of collections including admin, local, shop, orders, products, sessions (selected), and users.

The main pane shows the 'shop.sessions' collection. The 'Documents' tab is active, displaying a single document in a JSON format:

```
{
  "_id": "xp13lCqU3chAeYTG0xjPlgu5K2rYxK",
  "session": {
    "cookie": {
      "originalMaxAge": null,
      "expires": null,
      "secure": null,
      "httpOnly": true,
      "domain": null,
      "path": "/",
      "sameSite": null
    },
    "isLoggedIn": true,
    "expires": "2018-10-11 15:54:34.832"
  }
}
```

- in the sessions collection, you will find a session with an ID and in that session, you will find that information like 'isLoggedIn' and some information about the cookie which belongs to that session. and also you find the expiry data that was set by default.
- so this is how sessions are now stored and this is how you should store them for production. use a real session store, don't use the memory store which is less secure and which also is less unlimited or which will reach limits when more users use your app.
- with that sessions are a powerful tool for storing data across requests while still scoping them to a single user and not sharing the data across users. because different users have different sessions. this is great way mostly for managing authentication but you could also store something like the shopping cart in a session.
- in general, use a session for any data that belongs to a user that you don't wanna lose after every response you send.

```
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 /**this will gives you a function which should execute to which you pass your session
10 * this 'session' object you are importing from express-session is passed to a function
11 * which is yielded by required mongodb-session
12 * and the result of that function call is stored in 'MongoDBStore'
13 */
14 const MongoDBStore = require('connect-mongodb-session')(session)
15
16 const errorController = require('./controllers/error');
17 const User = require('./models/user')
18
19 const MONGODB_URI = 'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-
z3v1k.mongodb.net/shop'
20
21 const app = express();
22 /**'require("connect-mongodb-session")(session)' happens to yield a constructor function
which we store in MongoDBStore
23 *
24 * you pass some options and now which options could that database store require?
25 * it will require a connection string
26 * because it needs to know in which database server to store your data
27 *
28 * and i will define the collection
29 * and you need to define the collection where your sessions will be stored.
30 */
31 const store = new MongoDBStore({
32   uri: MONGODB_URI,
33   collection: 'sessions'
34 })
35
36 app.set('view engine', 'ejs');
37 app.set('views', 'views');
38
39 const adminRoutes = require('./routes/admin');
40 const shopRoutes = require('./routes/shop');
41 const authRoutes = require('./routes/auth');
42
43 /**your session data will be stored in there */
44 app.use(bodyParser.urlencoded({ extended: false }));
45 app.use(express.static(path.join(__dirname, 'public')));
46 app.use(
47   session({
48     secret: 'my secret',
49     resave: false,
50     saveUninitialized: false,
51     store: store
52   })
53 )
54
55 app.use((req, res, next) => {

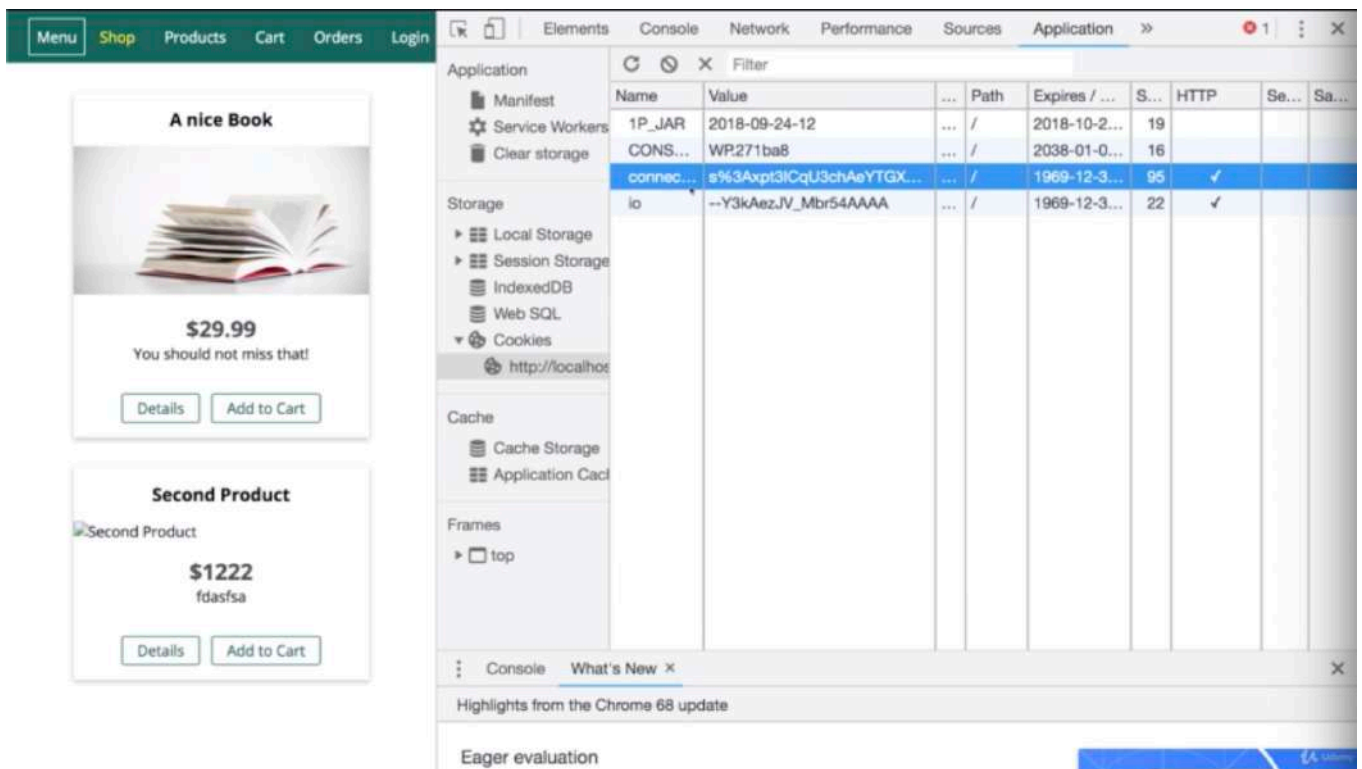
```

```

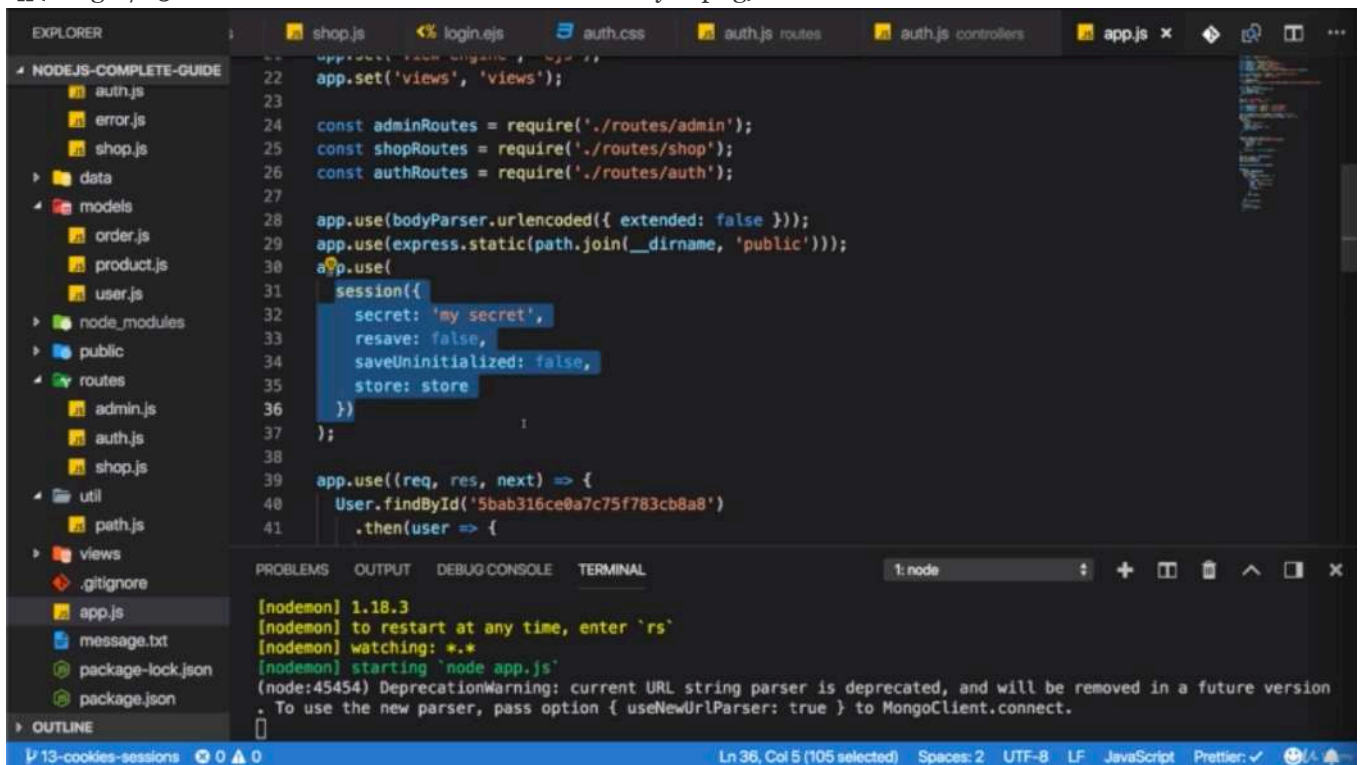
56 User.findById('5cbb2b2c80bd7193adb9eeeb')
57   .then(user => {
58     req.user = user
59     next();
60   })
61   .catch(err => console.log(err));
62 });
63
64 app.use('/admin', adminRoutes);
65 app.use(shopRoutes);
66 app.use(authRoutes)
67
68 app.use(errorController.get404);
69
70 mongoose
71   .connect(
72     MONGODB_URI
73   )
74   .then(result => {
75     User
76       .findOne()
77       .then(user => {
78         if(!user){
79           const user = new User({
80             name: 'Max',
81             email: 'max@test.com',
82             cart: {
83               items: []
84             }
85           })
86           user.save()
87         }
88       })
89     app.listen(3000)
90   })
91   .catch(err => {
92     console.log(err)
93   })

```

* Chapter 238: Sessions & Cookies - A Short Summary



- if you are wondering how the session cookie, this cookie here, how this is set, this is done automatically by express-session.



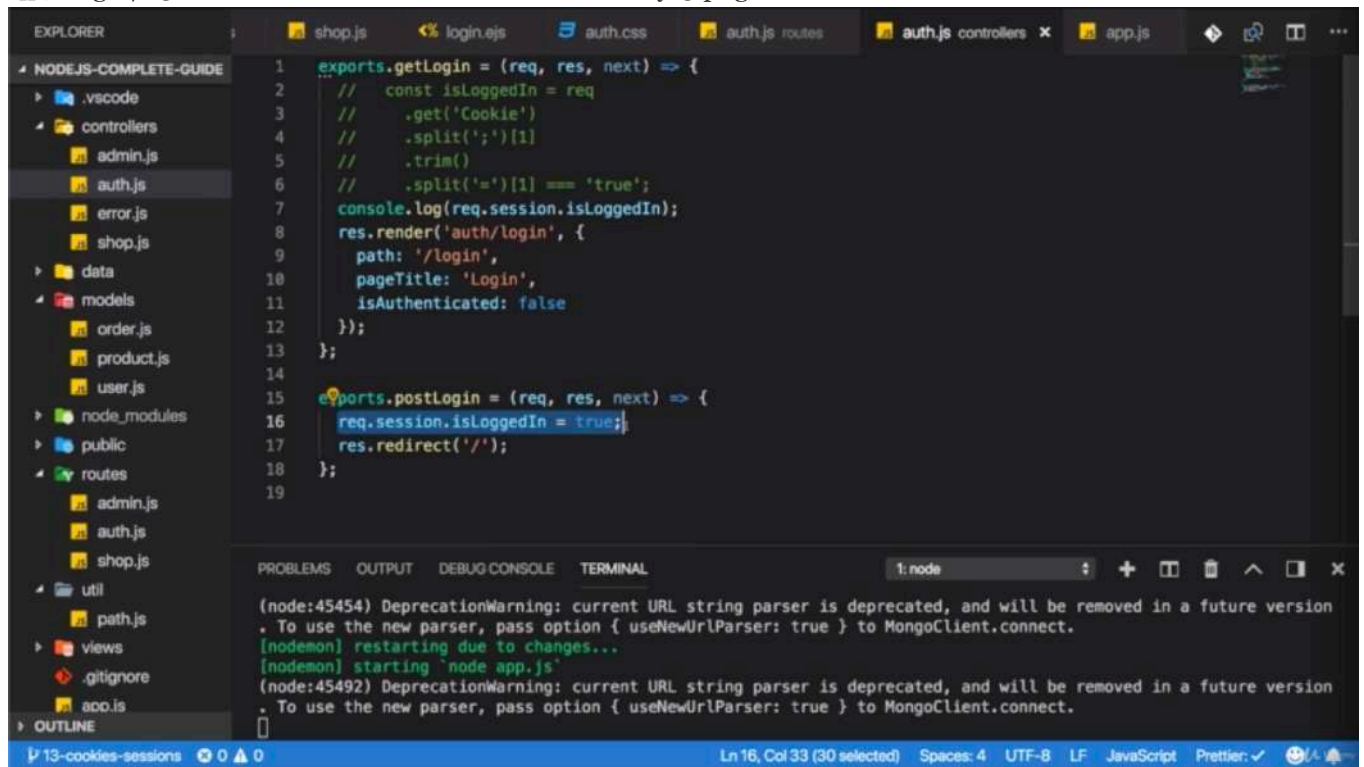
- so by that middleware we are configuring

- and that's also why you can add cookie related configurations because this middleware automatically sets a cookie for you and it automatically reads the cookie value for you too. so it does all the cookie parsing and setting for you.

- with that, you rarely need to manage cookies on your own. because that session cookie and with that i don't mean a cookie which gets lost after you close browser but that cookie that identifies a server side session, that is the most prominent, the most common use case for cookies besides advertisement, tracking which you typically don't implement on your own but you use 3rd party tools like google for that.

- that session cookie, so that session identifying cookie is an important thing and sessions on the server are often

used for authentication but you could use them for any kind of data you wanna store.



```
1 exports.getLogin = (req, res, next) => {
2   // const isLoggedIn = req
3   //   .get('Cookie')
4   //   .split(';')[1]
5   //   .trim()
6   //   .split('=')[1] === 'true';
7   console.log(req.session.isLoggedIn);
8   res.render('auth/login', {
9     path: '/login',
10    pageTitle: 'Login',
11    isAuthenticated: false
12  });
13 };
14
15 exports.postLogin = (req, res, next) => {
16   req.session.isLoggedIn = true;
17   res.redirect('/');
18 };
19
```

1: node

(node:45454) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

[nodemon] restarting due to changes...

[nodemon] starting 'node app.js'

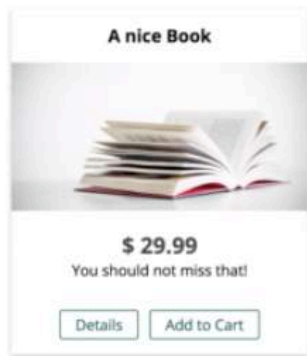
(node:45492) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

- we happen to store the information whether the user is logged in. but you could be storing the carts, the shopping cart of the user or anything which belongs to a user which should be shared across requests as i highlighted

* Chapter 239: Deleting A Cookie

1. update

- ./views/includes/navigation.ejs
- ./routes/auth.js
- ./controllers/auth.js



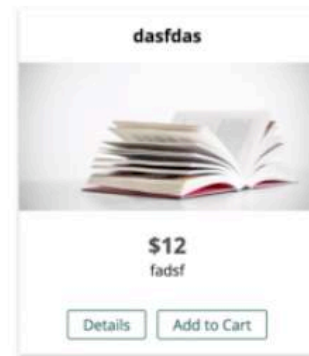
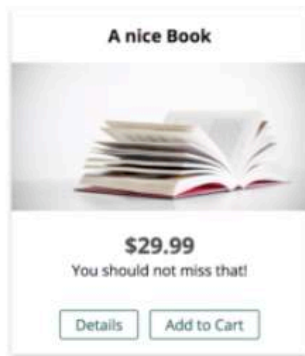
- if we reload our page, we see logout next to login.
 - and when i click this, i wanna clear any session i might have.
-

E-Mail

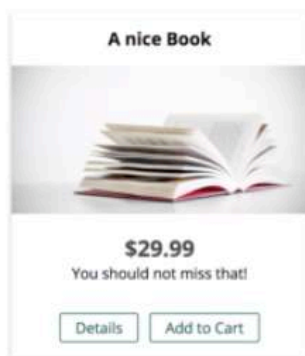
Password

Login

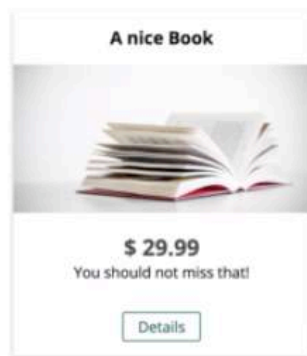
- if i reload, i have got no session cookie here
-



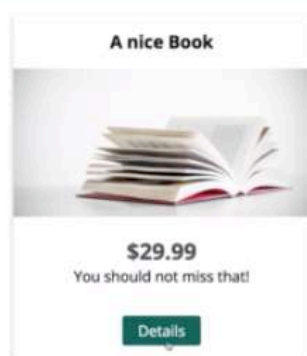
- and if i click 'Login' button, we can now use that session.



- if we go to MongoDB Compass, we see we have 4 objects which make sense,



- and if i click Logout, we are redirected, the session cookie still exists.



- but you see the session was deleted over there and the session cookie still exists but that is no problem. because no matching session will be found. so that is fine. it's not doing anything. and it will be renewed once we login again. then this will be overwritten.

- and when we close the browser, it would also be deleted because it's not a permanent cookie, it's a session cookie which means it's a cookie that doesn't have an expiry date in the future, it doesn't have 'maxAge', it will get deleted when we close the browser and it's worthless in this state here.

```

1 <!--./views/includes/navigation.ejs-->
2
3 <div class="backdrop"></div>
4 <header class="main-header">

```



```

5     <button id="side-menu-toggle">Menu</button>
6     <nav class="main-header__nav">
7         <ul class="main-header__item-list">
8             <li class="main-header__item">
9                 <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10            </li>
11            <li class="main-header__item">
12                <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
13            </li>
14            <li class="main-header__item">
15                <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
16            </li>
17            <li class="main-header__item">
18                <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
19            </li>
20            <% if (isAuthenticated) { %>
21            <li class="main-header__item">
22                <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
23            </a>
24            </li>
25            <li class="main-header__item">
26                <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
27            </a>
28            </li>
29            <% } %>
30        </ul>
31        <ul class="main-header__item-list">
32            <li class="main-header__item">
33                <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
34            </li>
35            <li class="main-header__item">
36                <form action="/logout" method="post">
37                    <button type="submit">Logout</button>
38                </form>
39            </li>
40        </ul>
41    </nav>
42</header>
43
44<nav class="mobile-nav">
45    <ul class="mobile-nav__item-list">
46        <li class="mobile-nav__item">
47            <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
48        </li>
49        <li class="mobile-nav__item">
50            <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
51        </li>
52        <li class="mobile-nav__item">
53            <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
54        </li>
55        <li class="mobile-nav__item">

```

```

56         <a class="<%= path === '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
57     </li>
58     <li class="mobile-nav__item">
59         <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
60     </a>
61 </li>
62 <li class="mobile-nav__item">
63     <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
64 </a>
65 </li>
66 </ul>
67 </nav>

```

```

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

```

```

1 //./controllers/auth.js
2
3 const User = require('../models/user');
4
5 exports.getLogin = (req, res, next) => {
6     res.render('auth/login', {
7         path: '/login',
8         pageTitle: 'Login',
9         isAuthenticated: false
10    });
11 };
12
13 exports.postLogin = (req, res, next) => {
14     User.findById('5cbb2b2c80bd7193adb9eeeb')
15         .then(user => {
16             req.session.isLoggedIn = true;
17             req.session.user = user;
18             res.redirect('/');
19         })
20         .catch(err => console.log(err));
21 };
22
23 exports.postLogout = (req, res, next) => {
24     /**we can do this by reaching out to our session object
25     * and then we can call 'destroy()' there
26     * this is a method provided by the session package we are using.

```

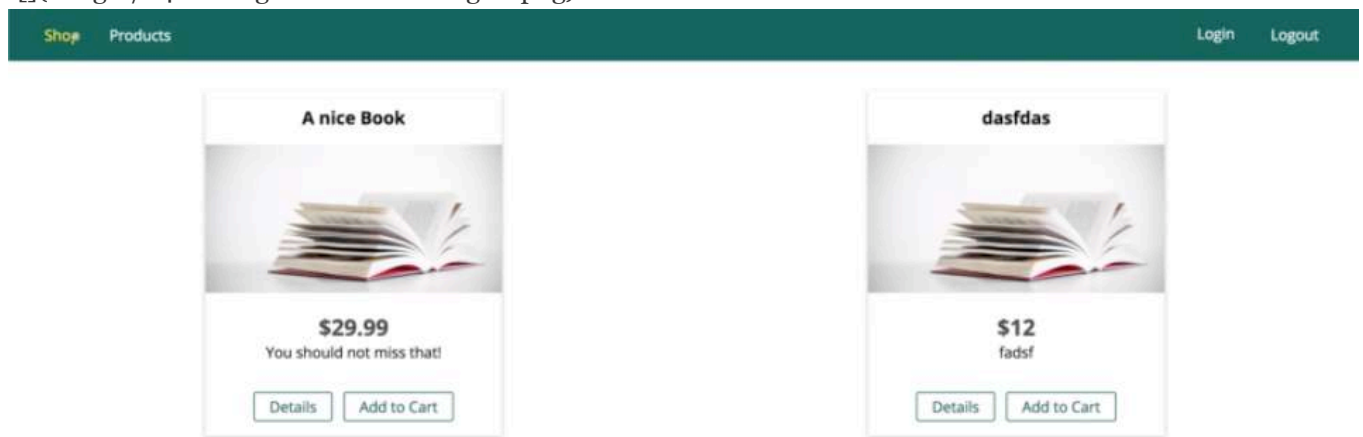
```

27 * 'destroy()' takes a function which we pass to it
28 * which will be called once it's done destroying the session
29 * and in there, request session will then not be available anymore
30 * because we got rid of that session
31 */
32 req.session.destroy(err => {
33   console.log(err);
34   res.redirect('/');
35 });
36 };
37
38

```

* Chapter 240: Fixing Some Minor Bugs

1. update
 - ./controllers/shop.js
 - ./views/includes/navigation.ejs
 - ./views/shop/product-list.ejs
 - ./views/includes/add-to-cart.ejs
 - ./views/shop/index.ejs




- if i reload this page, we only see shop and products by default

E-Mail

Password

Login

A nice Book




\$29.99

You should not miss that!

Details

Add to Cart

dasfdas

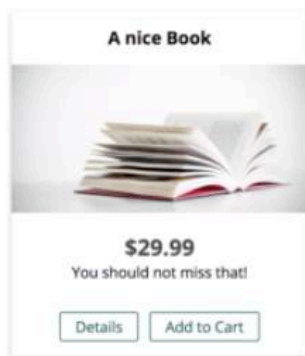


\$12

fadsf

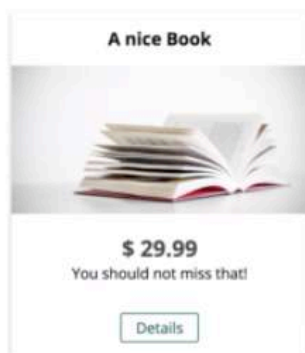
Details

Add to Cart



- once i login, we see the rest.

- and if i click logout, then we see login button



A nice Book



\$29.99

You should not miss that!

[Details](#)

dasfdas



\$12

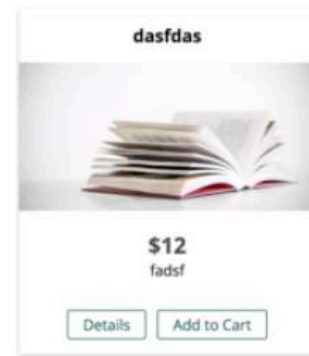
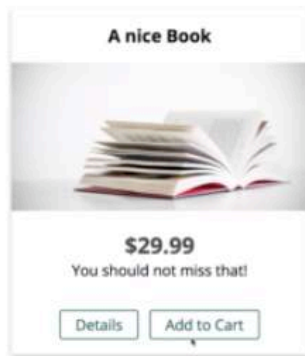
fadsf

[Details](#)

E-Mail

Password

[Login](#)



```

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

```

```

36 exports.getIndex = (req, res, next) => {
37   Product.find()
38     .then(products => {
39     res.render('shop/index', {
40       prods: products,
41       pageTitle: 'Shop',
42       path: '/',
43       isAuthenticated: req.session.isLoggedIn
44     });
45   })
46     .catch(err => {
47     console.log(err);
48   });
49 };
50
51 exports.getCart = (req, res, next) => {
52   req.user
53     .populate('cart.items.productId')
54     .execPopulate()
55     .then(user => {
56     const products = user.cart.items;
57     res.render('shop/cart', {
58       path: '/cart',
59       pageTitle: 'Your Cart',
60       products: products,
61       isAuthenticated: req.session.isLoggedIn
62     });
63   })
64     .catch(err => console.log(err));
65 };
66
67 exports.postCart = (req, res, next) => {
68   const prodId = req.body.productId;
69   Product.findById(prodId)
70     .then(product => {
71     return req.user.addToCart(product);
72   })
73     .then(result => {
74     console.log(result);
75     res.redirect('/cart');
76   });
77 };
78
79 /**without a valid session and we don't have a valid session now after destroying it
80 * all these methods, actions
81 * where i do reach out to my user 'req.session.user' to fetch the current orders,
82 * these will all fail
83 * and this makes sense because we need a session
84 * because we need a user for that.
85 * so what would work are shop and products and there details
86 * 'Add to Cart' will not work though.
87 *
88 * so first step to improving that besides checking for the existence of a user on the
89 server
89 * which we will all add in the authentication module
90 * but first thing we can do is that

```



```

91  * we adjust our frontend to only display things we can interact with based on our current
    authentication status
92  *
93  */
94  exports.postCartDeleteProduct = (req, res, next) => {
95    const prodId = req.body.productId;
96    req.user
97      .removeFromCart(prodId)
98      .then(result => {
99        res.redirect('/cart');
100      })
101      .catch(err => console.log(err));
102  };
103
104  exports.postOrder = (req, res, next) => {
105    req.user
106      .populate('cart.items.productId')
107      .execPopulate()
108      .then(user => {
109        const products = user.cart.items.map(i => {
110          return { quantity: i.quantity, product: { ...i.productId._doc } };
111        });
112        const order = new Order({
113          user: {
114            name: req.user.name,
115            userId: req.user
116          },
117          products: products
118        });
119        return order.save();
120      })
121      .then(result => {
122        return req.user.clearCart();
123      })
124      .then(() => {
125        res.redirect('/orders');
126      })
127      .catch(err => console.log(err));
128  };
129
130  exports.getOrders = (req, res, next) => {
131    Order.find({ 'user.userId': req.user._id })
132      .then(orders => {
133        res.render('shop/orders', {
134          path: '/orders',
135          pageTitle: 'Your Orders',
136          orders: orders,
137          isAuthenticated: req.session.isLoggedIn
138        });
139      })
140      .catch(err => console.log(err));
141  };
142
143  <!--./views/includes/navigation.ejs-->
144
145  <div class="backdrop"></div>

```

```

4 <header class="main-header">
5   <button id="side-menu-toggle">Menu</button>
6   <nav class="main-header__nav">
7     <ul class="main-header__item-list">
8       <li class="main-header__item">
9         <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
10      </li>
11      <li class="main-header__item">
12        <a class="<%= path === '/products' ? 'active' : '' %>"
href="/products">Products</a>
13      </li>
14      <!--first of all, 'if (isAuthenticated)' should be moved up
15          and also include cart and orders
16          because these only make sense if we have a user
17          because otherwise as we saw, we will get an error. if we visit these pages
18      -->
19      <% if (isAuthenticated) { %>
20        <li class="main-header__item">
21          <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
22        </li>
23        <li class="main-header__item">
24          <a class="<%= path === '/orders' ? 'active' : '' %>"
href="/orders">Orders</a>
25        </li>
26        <li class="main-header__item">
27          <a class="<%= path === '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
28          </a>
29        </li>
30        <li class="main-header__item">
31          <a class="<%= path === '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
32          </a>
33        </li>
34      <% } %>
35    </ul>
36    <ul class="main-header__item-list">
37      <!--login and logout should only be rendered when we are not logged in.
38          so i'm checking if we are authenticated to show 'Add Product' and 'Admin
Products',
39
40          and then we can repeat the logic down there
41          and only show this entire unordered if we are not authenticated
42          so this is not true then i wanna show these items
43      -->
44      <% if (!isAuthenticated) { %>
45        <li class="main-header__item">
46          <a class="<%= path === '/login' ? 'active' : '' %>" href="/login">Login</a>
47        </li>
48      <% } else { %>
49        <li class="main-header__item">
50          <form action="/logout" method="post">
51            <button type="submit">Logout</button>
52          </form>
53        </li>
54      <% } %>

```

```

55     </ul>
56   </nav>
57 </header>
58
59 <nav class="mobile-nav">
60   <ul class="mobile-nav__item-list">
61     <li class="mobile-nav__item">
62       <a class="<%= path == '/' ? 'active' : '' %>" href="/">Shop</a>
63     </li>
64     <li class="mobile-nav__item">
65       <a class="<%= path == '/products' ? 'active' : '' %>"
href="/products">Products</a>
66     </li>
67     <li class="mobile-nav__item">
68       <a class="<%= path == '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
69     </li>
70     <li class="mobile-nav__item">
71       <a class="<%= path == '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
72     </li>
73     <li class="mobile-nav__item">
74       <a class="<%= path == '/admin/add-product' ? 'active' : '' %>"
href="/admin/add-product">Add Product
75     </a>
76     </li>
77     <li class="mobile-nav__item">
78       <a class="<%= path == '/admin/products' ? 'active' : '' %>"
href="/admin/products">Admin Products
79     </a>
80     </li>
81   </ul>
82 </nav>

```

```

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
34     <% if (!isAuthenticated) { %>
35         <%= include('../includes/add-to-cart.ejs', {product:
product}) %>
36     <% } %>
37 </div>
38 </article>
39 <% } %>
40 </div>
41 <% } else { %>
42     <h1>No Products Found!</h1>
43 <% } %>
44 </main>
45 <%= include('../includes/end.ejs') %>

```

```

1 <!--../views/includes/add-to-cart.ejs-->
2
3
4 <form action="/cart" method="post">
5     <button class="btn" type="submit">Add to Cart</button>
6     <input type="hidden" name="productId" value="<%= product._id %>">
7 </form>
8

```

```

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>

```

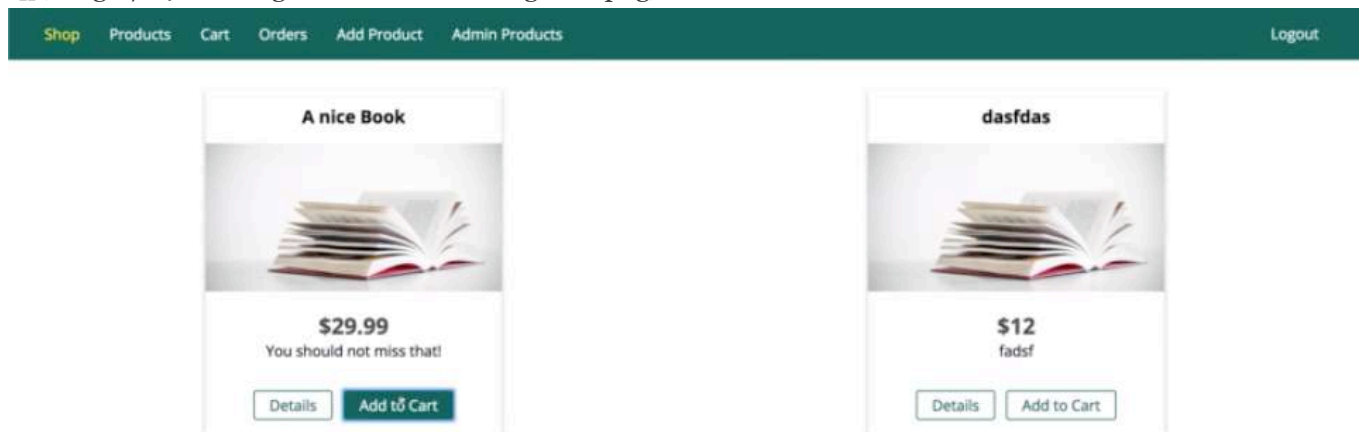
```

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') %>

```

* Chapter 241: Making "Add To Cart" Work Again

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



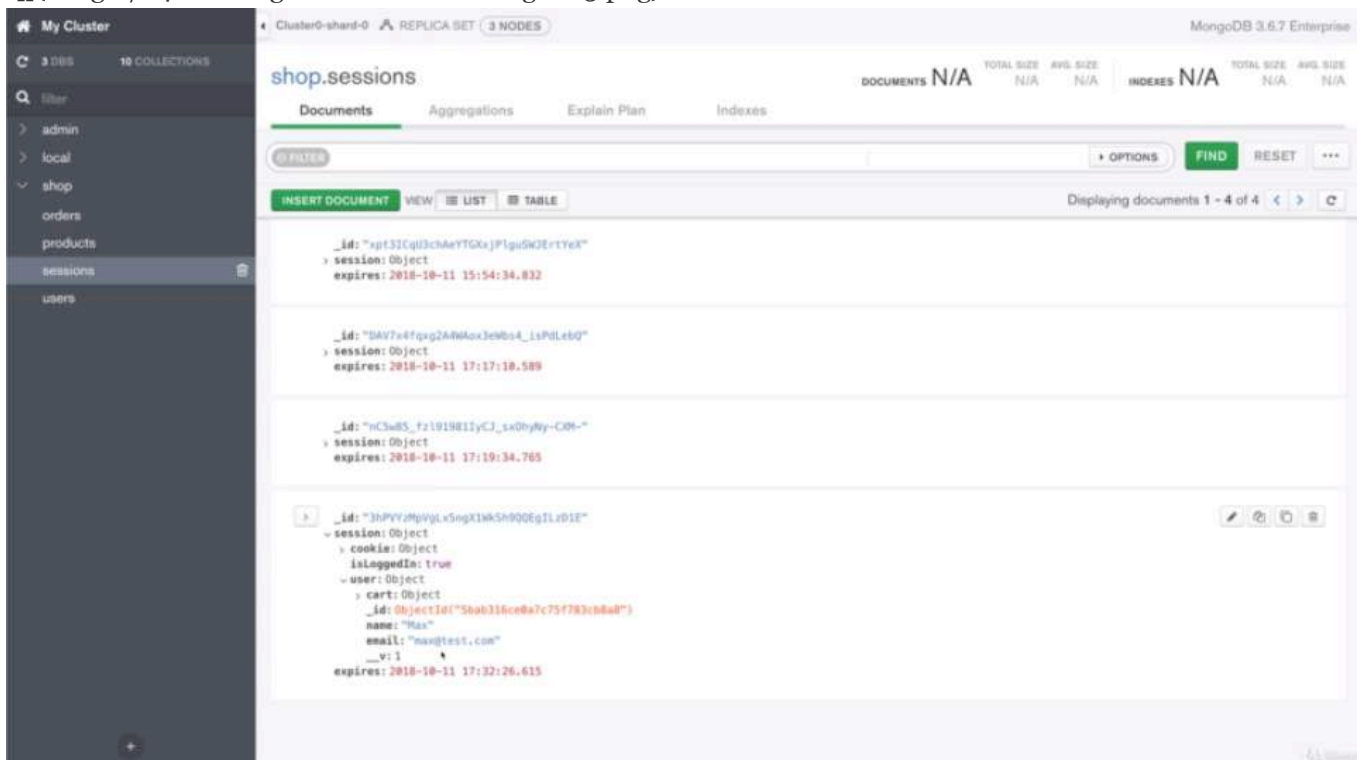
Warning: For production...

100% Success

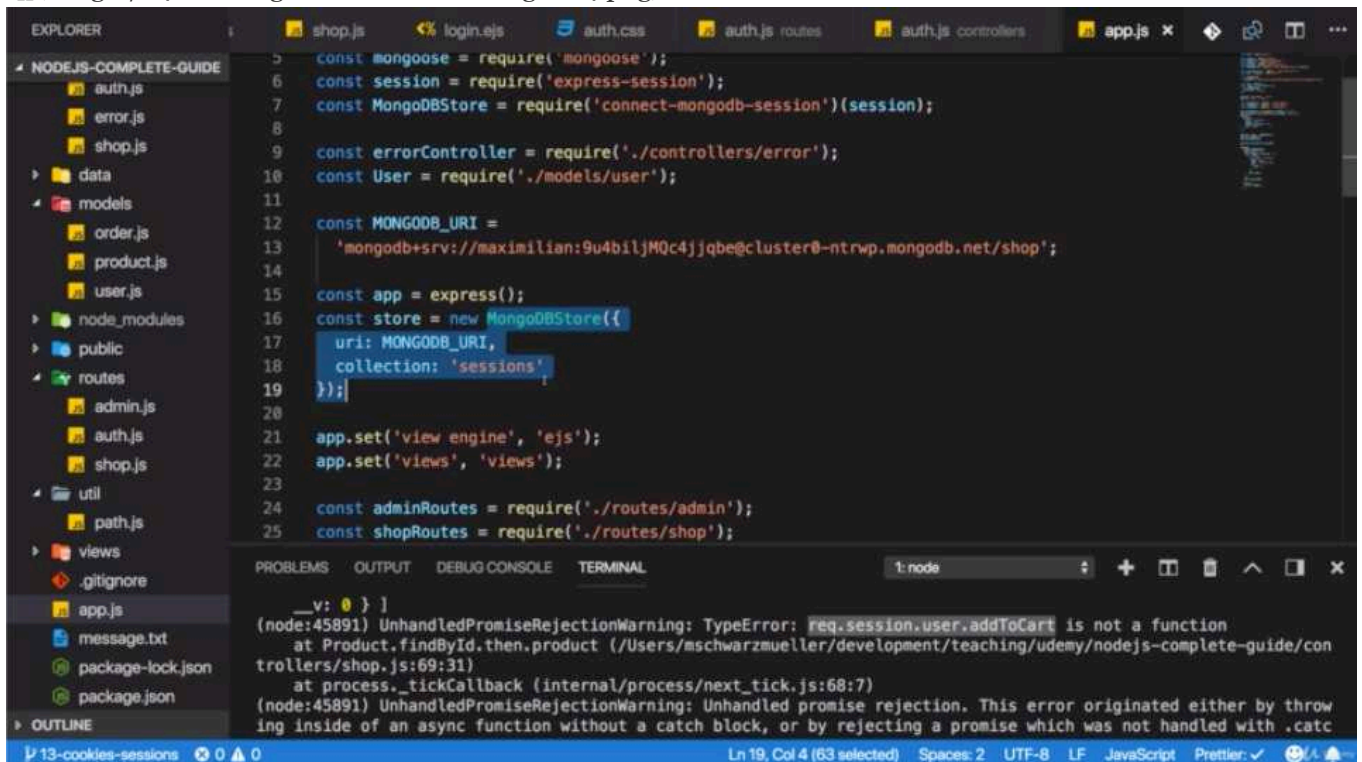
- i clicked 'Add To Cart' then we are now failing to do this.

The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project structure with files like `admin.js`, `auth.js`, `error.js`, `shop.js`, `data`, `models`, `order.js`, `product.js`, `user.js`, `node_modules`, `public`, `routes`, `admin.js`, `auth.js`, `shop.js`, `util`, `path.js`, `views`, `.gitignore`, `app.js`, and `OUTLINE`. The code editor shows a snippet of HTML code for a product card. The terminal shows an error: `(node:45891) UnhandledPromiseRejectionWarning: TypeError: req.session.user.addToCart is not a function`. The error message also includes the file path `/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/controllers/shop.js:69:31` and the function `process._tickCallback`.

- and if we go back, we see an error that "'req.session.user.addToCart' is not a function" so we somehow fail to execute our 'Add To Cart' function
 - our functions that normally are available on the user object. why is that?
 - the reason for that is previous setup, we always store the user in the request 'req.user = user'. this was a per request action anyways and we fetched that user for every request in the middleware in `app.js` file
 - so we fetch the user from the database and mongoose automatically gave us a full user object not just the data in the database. but the full user model with all the methods and we stored that `./models/user.js` file, the user model in the request,
 - with the session, this works a bit different. with the session, we are not fetching this for every request. instead we store the user in our session upon logging-in in `./controllers/auth.js` file like 'req.session.user = user'
 - what happens there? it gets stored to the database
-



- if we go to MongoDB Compass and refresh there, we can see is our user object. and this is just the data.



```
const mongoose = require('mongoose');
const session = require('express-session');
const MongoDBStore = require('connect-mongodb-session')(session);

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

const MONGODB_URI =
  'mongodb+srv://maximilian:9u4biljM0c4jjqbe@cluster0-ntrwp.mongodb.net/shop?';

const app = express();
const store = new MongoDBStore({
  uri: MONGODB_URI,
  collection: 'sessions'
});

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

const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');
```

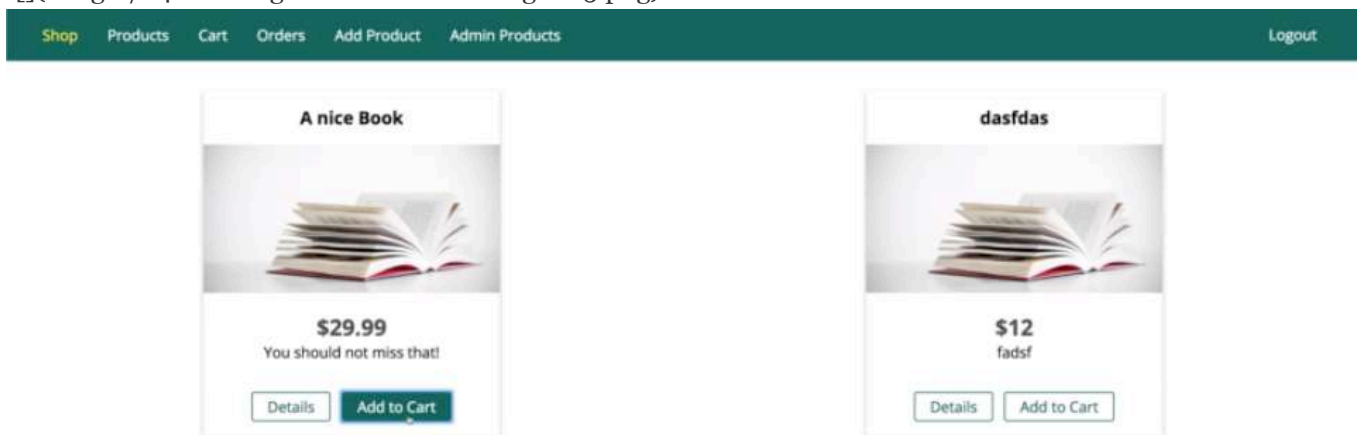
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

__v: 0 }]
(node:45891) UnhandledPromiseRejectionWarning: TypeError: req.session.user.addToCart is not a function
 at Product.findById.then.product (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/controllers/shop.js:69:31)
 at process._tickCallback (internal/process/next_tick.js:68:7)
(node:45891) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch

- now for every new request, the session middleware doesn't go ahead and fetch the user with the help of mongoose, it fetches the session data from MongoDB. it uses the MongoDBStore and the MongoDBStore doesn't know about our mongoose models.

- so when it fetches the data from session database, when it fetches this data, it only fetches the data. it doesn't fetch an object with all the methods provided by mongoose.



- now what can we do regarding that? one thing is that we are reverting back a bit, we can re-add that middle ware which we had earlier after we initialized our session.

ShopProductsCartOrdersAdd ProductAdmin ProductsLogout

A nice Book

Quantity: 1

Delete

Order Now!

ShopProductsCartOrdersAdd ProductAdmin ProductsLogout

A nice Book

Quantity: 1

Delete

Order Now!

E-Mail

Password

[Login](#)


No Products in Cart!

E-Mail

Password

Login

A nice Book




\$29.99

You should not miss that!

DetailsAdd to Cart

dasfdas



\$12

fadsf

DetailsAdd to Cart

A nice Book



29.99

You should not miss that!

Add to Cart

A nice Book

Quantity: 2

Delete

Order Now!

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose')
8 const session = require('express-session')
9 const MongoDBStore = require('connect-mongodb-session')(session)
10
11 const errorController = require('./controllers/error');
12 const User = require('./models/user')
13
14 const MONGODB_URI = 'mongodb+srv://maximilian:DD5EbADjazBuTqk@cluster0-
```

```

15 z3v1k.mongodb.net/shop'
16 const app = express();
17 const store = new MongoDBStore({
18   uri: MONGODB_URI,
19   collection: 'sessions'
20 })
21
22 app.set('view engine', 'ejs');
23 app.set('views', 'views');
24
25 const adminRoutes = require('./routes/admin');
26 const shopRoutes = require('./routes/shop');
27 const authRoutes = require('./routes/auth');
28
29 /**your session data will be stored in there */
30 app.use(bodyParser.urlencoded({ extended: false }));
31 app.use(express.static(path.join(__dirname, 'public')));
32 app.use(
33   session({
34     secret: 'my secret',
35     resave: false,
36     saveUninitialized: false,
37     store: store
38   })
39 )
40
41 /**by the time we reach this middleware,
42  * we will have our session data loaded
43  * this means that now we wanna use that session data to load our real user
44  * to create our mongoose user model and how do we do that?
45  *
46  * we don't need to reach out to the database again
47  * instead we can again set the user for this request only
48  * and here i wanna create a user based on data stored in the session
49  * so data that persists across requests
50  * and i will create that user and store it in the req.user which only live for that request
51  * but it's fueled by data from the session
52  * and therefore it also survives cross request
53  *
54  * and i need to do that
55  * because i need mongoose model to work with
56  * because the data we store in the session storage in MongoDB,
57  * we retrieve it as just plain data,
58  * not as a mongoose model with all the cool methods mongoose gives us
59  * and that is why we get this error regarding 'Add To Cart' not being found and so on.
60  *
61  * so here i will initialize req.user
62  * and i don't wanna find a user by Id like this
63  * i will find a user by reaching out to 'req.session.user._id'
64  * this makes sense because that is the data,
65  * the user is what i store in my session
66  * and i wanna get the ID and then find that user in the database with the help of the user
  model which is provided by mongoose
67  */
68 app.use((req, res, next) => {

```

```

69  if(!req.session.user){
70      /**we gonna call 'return next()'
71      * so that the code below will not be executed.
72      * this code below only run if we don have a session user.
73      */
74      return next()
75  }
76  User.findById(req.session.user._id)
77  .then(user => {
78      /**we don't wanna store anything in the session
79      * because the session is already something which will be managed automatically
80      * and for the incoming requests, we register the middleware,
81      * the middleware will then look for a session cookie,
82      * if it finds one, it will look for a fitting session in the database and load the data
83      */
84      //req.session.isLoggedIn = true;
85      //req.session.user = user;
86      /**here i get back mongoose model user which i store in req.user
87      * and this is what i need now to make sure that i have a mongoose model to work with
88      * so that all these cool mongoose methods work again.
89      *
90      * and we need to adjust ./controllers/admin.js and ./controllers/shop.js again
91      * and replace 'req.session.user' with 'req.user' again
92      * because that is where i'm storing my mongoose model user
93      */
94      req.user = user
95      /** we will call 'next()' so taht the incoming requests can continue with the next
96      middleware. */
97      next()
98  })
99  .catch(err => console.log(err));
100 })
101 app.use((req, res, next) => {
102     User.findById('5cbb2b2c80bd7193adb9eeeb')
103     .then(user => {
104         req.user = user
105         next();
106     })
107     .catch(err => console.log(err));
108 });
109
110 app.use('/admin', adminRoutes);
111 app.use(shopRoutes);
112 app.use(authRoutes)
113
114 app.use(errorController.get404);
115
116 mongoose
117     .connect(
118         MONGODB_URI
119     )
120     .then(result => {
121         User
122         .findOne()

```

```

123     .then(user => {
124         if(!user){
125             const user = new User({
126                 name: 'Max',
127                 email: 'max@test.com',
128                 cart: {
129                     items: []
130                 }
131             })
132             user.save()
133         }
134     })
135     app.listen(3000)
136 })
137 .catch(err => {
138     console.log(err)
139 })

```

```

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

```

```

40     prods: products,
41     pageTitle: 'Shop',
42     path: '/',
43     isAuthenticated: req.session.isLoggedIn
44   });
45 })
46 .catch(err => {
47   console.log(err);
48 });
49 };
50
51 /**i wanna replace 'req.session.user' with 'req.user'
52 * because we store our mongoose model in that request only object again.
53 * do the same in ./controllers/admin.js file
54 */
55 exports.getCart = (req, res, next) => {
56   req.user
57     .populate('cart.items.productId')
58     .execPopulate()
59     .then(user => {
60       const products = user.cart.items;
61       res.render('shop/cart', {
62         path: '/cart',
63         pageTitle: 'Your Cart',
64         products: products,
65         isAuthenticated: req.session.isLoggedIn
66       });
67     })
68     .catch(err => console.log(err));
69 };
70
71 exports.postCart = (req, res, next) => {
72   const prodId = req.body.productId;
73   Product.findById(prodId)
74     .then(product => {
75       return req.user.addToCart(product);
76     })
77     .then(result => {
78       console.log(result);
79       res.redirect('/cart');
80     });
81 };
82
83 exports.postCartDeleteProduct = (req, res, next) => {
84   const prodId = req.body.productId;
85   req.user
86     .removeFromCart(prodId)
87     .then(result => {
88       res.redirect('/cart');
89     })
90     .catch(err => console.log(err));
91 };
92
93 exports.postOrder = (req, res, next) => {
94   req.user
95     .populate('cart.items.productId')

```

```

96     .execPopulate()
97     .then(user => {
98         const products = user.cart.items.map(i => {
99             return { quantity: i.quantity, product: { ...i.productId._doc } };
100         });
101         const order = new Order({
102             user: {
103                 name: req.user.name,
104                 userId: req.user
105             },
106             products: products
107         });
108         return order.save();
109     })
110     .then(result => {
111         return req.user.clearCart();
112     })
113     .then(() => {
114         res.redirect('/orders');
115     })
116     .catch(err => console.log(err));
117 };
118
119 exports.getOrders = (req, res, next) => {
120     Order.find({ 'user.userId': req.user._id })
121     .then(orders => {
122         res.render('shop/orders', {
123             path: '/orders',
124             pageTitle: 'Your Orders',
125             orders: orders,
126             isAuthenticated: req.session.isLoggedIn
127         });
128     })
129     .catch(err => console.log(err));
130 };
131

```

```

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

```



```

21     price: price,
22     description: description,
23     imageUrl: imageUrl,
24     /**this will not mean that it's now only existent for this request
25      * the mongoose model object is but it's fueled by data that is stored in the session
26      * and therefore data that persist across requests
27      * and now that we saved all of that
28      */
29     userId: req.user
30 })
31 product
32   .save()
33   .then(result => {
34     // console.log(result);
35     console.log('Created Product');
36     res.redirect('/admin/products');
37   })
38   .catch(err => {
39     console.log(err);
40   });
41 };
42
43 exports.getEditProduct = (req, res, next) => {
44   const editMode = req.query.edit;
45   if (!editMode) {
46     return res.redirect('/');
47   }
48   const prodId = req.params.productId;
49   Product.findById(prodId)
50     .then(product => {
51       if (!product) {
52         return res.redirect('/');
53       }
54       res.render('admin/edit-product', {
55         pageTitle: 'Edit Product',
56         path: '/admin/edit-product',
57         editing: editMode,
58         product: product,
59         isAuthenticated: req.isAuthenticated()
60       });
61     })
62     .catch(err => console.log(err));
63 };
64
65 exports.postEditProduct = (req, res, next) => {
66   const prodId = req.body.productId;
67   const updatedTitle = req.body.title;
68   const updatedPrice = req.body.price;
69   const updatedImageUrl = req.body.imageUrl;
70   const updatedDesc = req.body.description;
71
72   Product
73     .findById(prodId)
74     .then(product => {
75       product.title = updatedTitle
76       product.price = updatedPrice

```

```

77     product.description = updatedDesc
78     product.imageUrl = updatedImageUrl
79     return product
80     .save()
81   })
82   .then(result => {
83     console.log('UPDATED PRODUCT!');
84     res.redirect('/admin/products');
85   })
86   .catch(err => console.log(err));
87   });
88
89 exports.getProducts = (req, res, next) => {
90   Product.find()
91   // .select('title price -_id')
92   // .populate('userId', 'name')
93   .then(products => {
94     console.log(products)
95     res.render('admin/products', {
96       prods: products,
97       pageTitle: 'Admin Products',
98       path: '/admin/products',
99       isAuthenticated: req.isLoggedIn
100    });
101  })
102  .catch(err => console.log(err));
103  });
104
105 exports.postDeleteProduct = (req, res, next) => {
106   const prodId = req.body.productId;
107   Product.findByIdAndRemove(prodId)
108   .then(() => {
109     console.log('DESTROYED PRODUCT');
110     res.redirect('/admin/products');
111   })
112   .catch(err => console.log(err));
113  });

```

* Chapter 242: Two Tiny Improvements

1. update
 - ./views/shop/product-detail.ejs
 - ./controllers/auth.js

A nice Book



\$29.99

You should not miss that!

Details

dasfdas



\$12

fadsf

Details

Warning: file_exists():

Warning: file_exists():

A nice Book



29.99

You should not miss that!

Warning: file_exists():

E-Mail

Password

Login

A nice Book



\$29.99

You should not miss that!

Details

dasfdas



\$12

fadsf

Details

A nice Book



29.99

You should not miss that!

Add to Cart

A nice Book



29.99

You should not miss that!

Add to Cart

A nice Book



\$29.99

You should not miss that!

Details

dasfdas



\$12

fadsf

Details

Placeholder for a small image or text.

Placeholder for a small image or text.

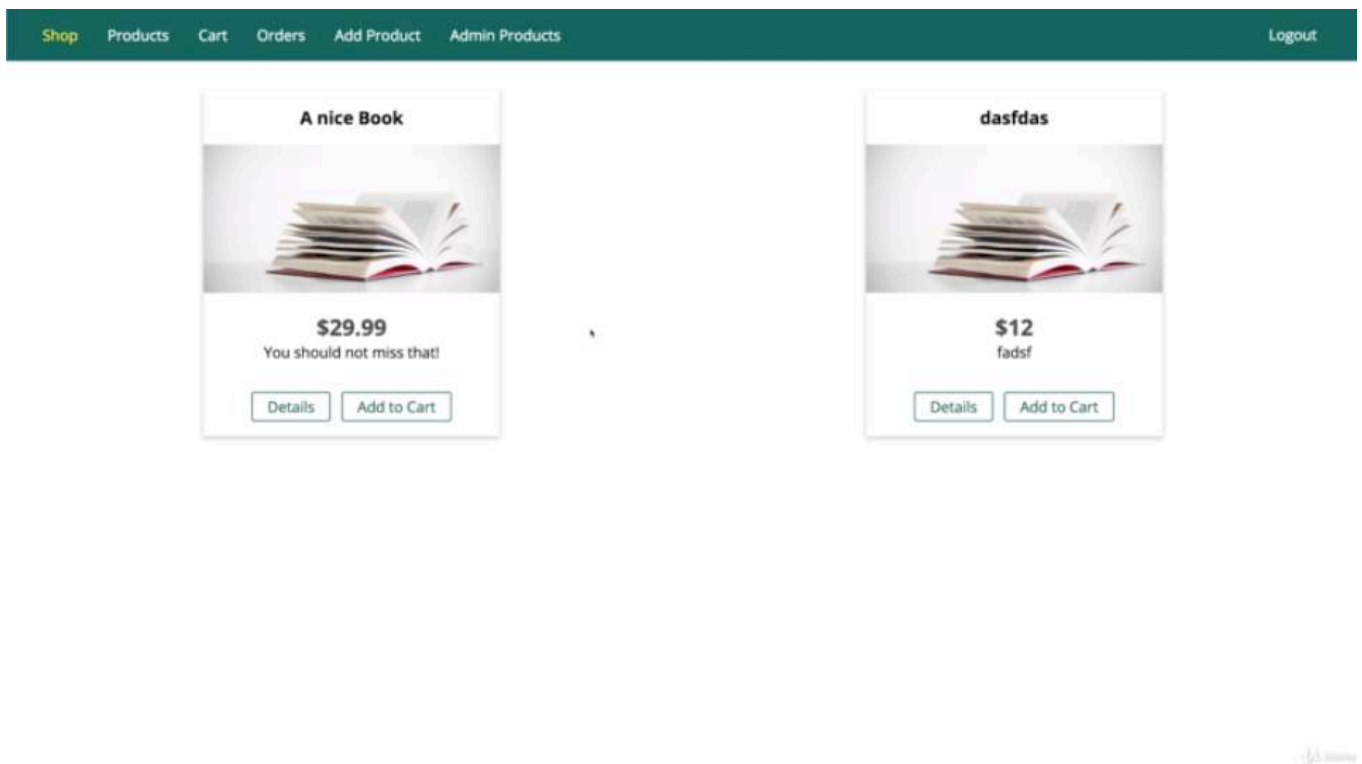
A nice Book



29.99

You should not miss that!

Placeholder for a small image or text.



- after logging in, the view didn't update accordingly. you need to reload the page to get there.


```

EXPLORER    login.ejs    auth.css    auth.js routes    auth.js controllers    app.js    product-detail.ejs
NODEJS-COMLETE-GUIDE
  controllers
    admin.js
    auth.js
    error.js
    shop.js
  data
  models
    order.js
    product.js
    user.js
  node_modules
  public
  routes
    admin.js
    auth.js
    shop.js
  util
    path.js
  views
    admin
      edit-product.ejs
  OUTLINE
  13-cookies-sessions* 0 0 0
Ln 18, Col 27 Spaces: 4 UTF-8 LF JavaScript Prettier: ✓

```

```

exports.getLogin = (req, res, next) => {
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    isAuthenticated: false
  });
};

exports.postLogin = (req, res, next) => {
  User.findById('5bab316ce8a7c75f783cb8a8')
    .then(user => {
      req.session.isLoggedIn = true;
      req.session.user = user;
      req.session.save(err => {
        console.log(err);
        res.redirect('/');
      });
    })
    .catch(err => console.log(err));
};

```

```

undefined
undefined
[nodeemon] restarting due to changes...
[nodeemon] starting 'node app.js'
(node:46801) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.


```

- the reason for this is that in ./controllers/auth.js when i have 'postLogIn' here, i set my session and when i redirect, when i send a response the session middleware goes ahead and create that session and that means it writes it to MongoDB because we use the MongoDB sessions Store and it sets the cookie.

- the problem is writing that data to a database like MongoDB can take a couple of milliseconds or depending on your speed even a bit more. the redirect is fired independent from that, so you might redirect too early.

- to be sure that your session has been set, you can use 'req.session.save()'

A nice Book




\$29.99

You should not miss that!

Details

Add to Cart

dasfdas




\$12

fadsf

Details

Add to Cart

A nice Book



\$29.99

You should not miss that!

Details

dasfdas



\$12

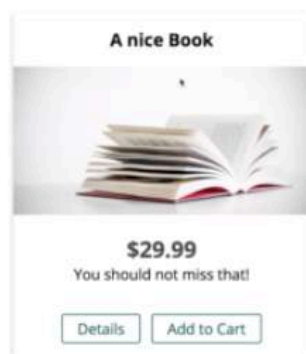
fadsf

Details

E-Mail

Password

Login



- if i logout, and i again login, now this will only continue once that session has really been created

```

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') %>
18     <% } %>
19 </main>
20 <%- include('../includes/end.ejs') %>

```

```

1 //./controllers/auth.js
2
3 const User = require('../models/user');
4
5 exports.getLogin = (req, res, next) => {
6     res.render('auth/login', {
7         path: '/login',
8         pageTitle: 'Login',
9         isAuthenticated: false
10    });
11 };
12
13 exports.postLogin = (req, res, next) => {
14     User.findById('5cbb2b2c80bd7193adb9eeeb')
15         .then(user => {
16             req.session.isLoggedIn = true;
17             req.session.user = user;
18             /** i set my session and when i redirect,
19              * when i send a response the session middleware goes ahead and create that session
20              * and that means it writes it to MongoDB
21              * because we use the MongoDB sessions Store and it sets the cookie.
22              *
23              * the problem is writing that data to a database like MongoDB can take a couple of
24              milliseconds or depending on your speed even a bit more.
25              * the redirect is fired independent from that,
26              * so you might redirect too early.
27              *
28              * to be sure that your session has been set,
29              * you can use 'req.session.save()'
30              * you normally don't need to do that
31              * but you need to do it in scenarios
32              * where you need to be sure that your session was created before you continue
33              * because here you can pass in a function
34              * that will be called once you are done saving the session
35              *
36              * you will get an error here if an error exists
37              * and in here you can safely redirect
38              * and you can be sure that your session has been created here.
39              *
40              * normally you don't need to call 'req.session.save()'
41              * if you need that guarantee which typically is the case when you redirect for
42              example
43              * because in such scenarios, the redirect will be fired independent from the session
44              being saved
45              * and therefore the redirect might be finished
46              * and the new page might be rendered before your session was updated on the server
47              and in the database
48              * */

```

```

45     req.session.save((err) => {
46         console.log(err)
47         res.redirect('/')
48     })
49 })
50 .catch(err => console.log(err));
51 };
52
53 exports.postLogout = (req, res, next) => {
54     req.session.destroy(err => {
55         console.log(err);
56         res.redirect('/');
57     });
58 };
59
60

```

* Chapter 243: Wrap Up



Module Summary

Cookies	Sessions
<ul style="list-style-type: none"> • Great for storing data on the client (browser) • Do NOT store sensitive data here! It can be viewed + manipulated • Cookies can be configured to expire when the browser is closed (=> "Session Cookie") or when a certain age/ expiry date is reached ("Permanent Cookie") • Works well together with Sessions... 	<ul style="list-style-type: none"> • Stored on the server, NOT on the client • Great for storing sensitive data that should survive across requests • You can store ANYTHING in sessions • Often used for storing user data/ authentication status • Identified via Cookie (don't mistake this with the term "Session Cookie") • You can use different storages for saving your sessions on the server