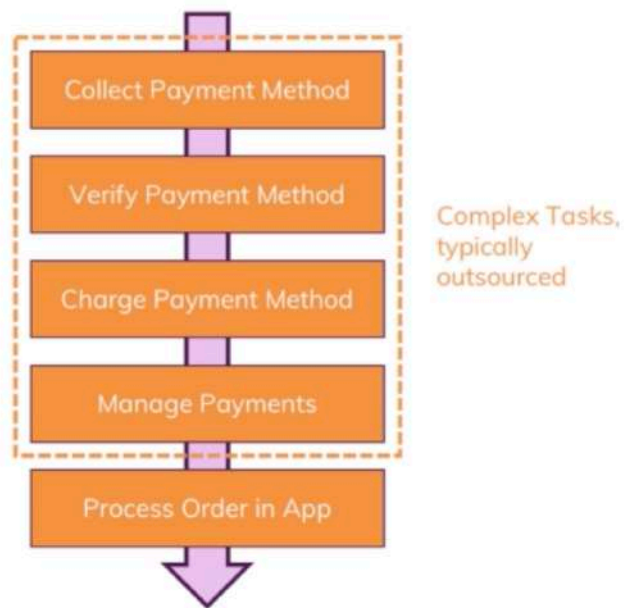


23. Adding Payments

* Chapter 350: How Payments Work

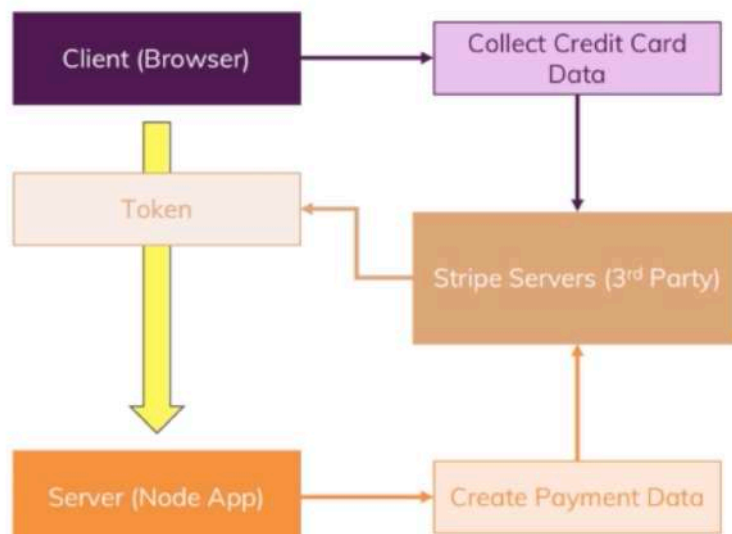


Payment Process



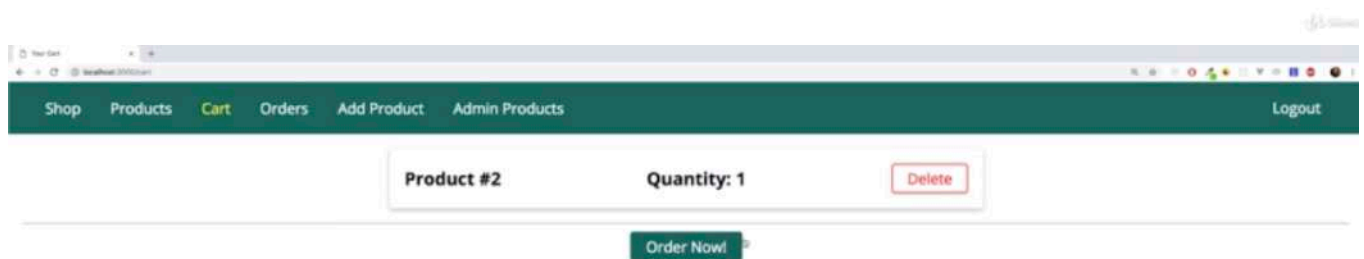
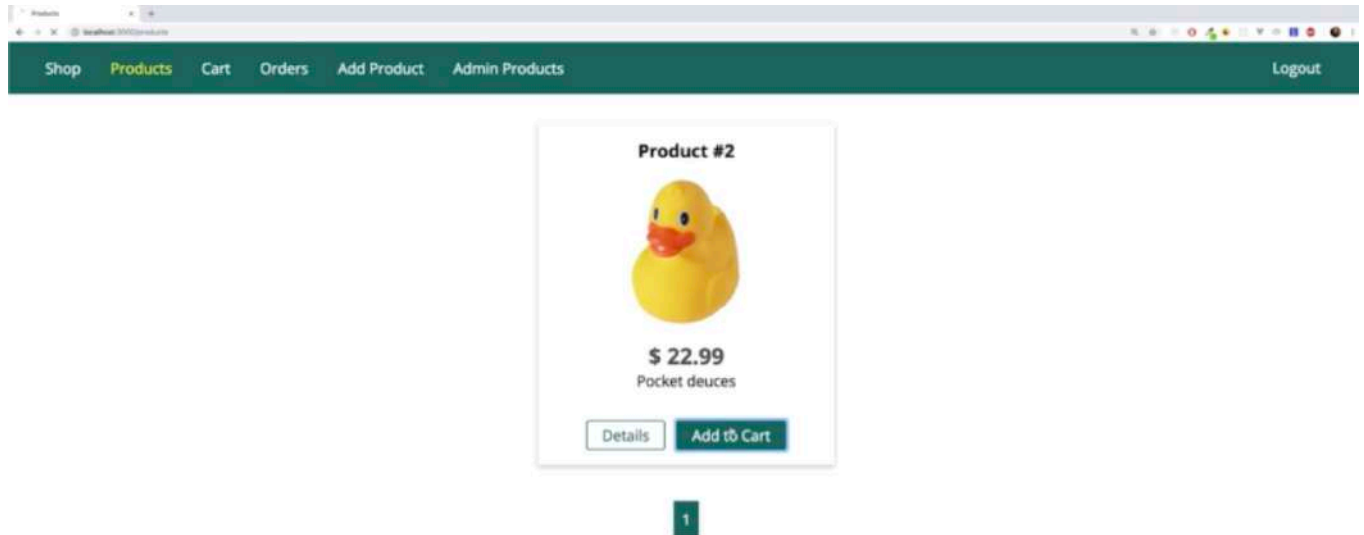


How Stripe Works



* Chapter 351: Adding A Checkout Page

1. update
 - ./views/shop/checkout.ejs
 - ./routes/shop.js
 - ./controllers/shop.js
 - ./views/shop/cart.ejs





```
1 <!--./views/shop/checkout.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/cart.css">
5
6   </head>
7
8   <body>
9     <%- include('../includes/navigation.ejs') %>
10    <main>
11      <ul class="cart__item-list">
12        <% products.forEach(p => { %>
13          <li class="cart__item">
14            <h1><%= p.productId.title %></h1>
15            <h2>Quantity: <%= p.quantity %></h2>
16          </li>
17        <% }) %>
18      </ul>
19      <div class="centered">
20        <h2>Total: <%= totalSum %></h2>
21      </div>
22    </main>
23    <%- include('../includes/end.ejs') %>
```

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

```

12 router.get('/', shopController.getIndex);
13
14 router.get('/products', shopController.getProducts);
15
16 router.get('/products/:productId', shopController.getProduct);
17
18 router.get('/cart', isAuth, shopController.getCart);
19
20 router.post('/cart', isAuth, shopController.postCart);
21
22 router.post('/cart-delete-item', isAuth, shopController.postCartDeleteProduct);
23
24 router.get('/checkout', isAuth, shopController.getCheckout);
25
26 router.post('/create-order', isAuth, shopController.postOrder);
27
28 router.get('/orders', isAuth, shopController.getOrders);
29
30 router.get('/orders/:orderId', isAuth, shopController.getInvoice);
31
32 module.exports = router;

```

```

1  ../controllers/shop.js
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const PDFDocument = require('pdfkit');
7
8  const Product = require('../models/product');
9  const Order = require('../models/order');
10
11 const ITEMS_PER_PAGE = 2;
12
13 exports.getProducts = (req, res, next) => {
14   const page = +req.query.page || 1;
15   let totalItems;
16
17   Product.find()
18     .countDocuments()
19     .then(numProducts => {
20       totalItems = numProducts;
21       return Product.find()
22         .skip((page - 1) * ITEMS_PER_PAGE)
23         .limit(ITEMS_PER_PAGE);
24     })
25     .then(products => {
26       res.render('shop/product-list', {
27         prods: products,
28         pageTitle: 'Products',
29         path: '/products',
30         currentPage: page,
31         hasNextPage: ITEMS_PER_PAGE * page < totalItems,
32         hasPreviousPage: page > 1,
33         nextPage: page + 1,
34         previousPage: page - 1,
35         lastPage: Math.ceil(totalItems / ITEMS_PER_PAGE)

```

```

36     });
37   })
38   .catch(err => {
39     const error = new Error(err);
40     error.httpStatusCode = 500;
41     return next(error);
42   });
43 };
44
45 exports.getProduct = (req, res, next) => {
46   const prodId = req.params.productId;
47   Product.findById(prodId)
48     .then(product => {
49       res.render('shop/product-detail', {
50         product: product,
51         pageTitle: product.title,
52         path: '/products'
53       });
54     })
55     .catch(err => {
56       const error = new Error(err);
57       error.httpStatusCode = 500;
58       return next(error);
59     });
60 };
61
62 exports.getIndex = (req, res, next) => {
63   const page = +req.query.page || 1;
64   let totalItems;
65
66   Product.find()
67     .countDocuments()
68     .then(numProducts => {
69       totalItems = numProducts;
70       return Product.find()
71         .skip((page - 1) * ITEMS_PER_PAGE)
72         .limit(ITEMS_PER_PAGE);
73     })
74     .then(products => {
75       res.render('shop/index', {
76         prods: products,
77         pageTitle: 'Shop',
78         path: '/',
79         currentPage: page,
80         hasNextPage: ITEMS_PER_PAGE * page < totalItems,
81         hasPreviousPage: page > 1,
82         nextPage: page + 1,
83         previousPage: page - 1,
84         lastPage: Math.ceil(totalItems / ITEMS_PER_PAGE)
85       });
86     })
87     .catch(err => {
88       const error = new Error(err);
89       error.httpStatusCode = 500;
90       return next(error);
91     });

```

```

92 };
93
94 exports.getCart = (req, res, next) => {
95   req.user
96     .populate('cart.items.productId')
97     .execPopulate()
98     .then(user => {
99       const products = user.cart.items;
100       res.render('shop/cart', {
101         path: '/cart',
102         pageTitle: 'Your Cart',
103         products: products
104       });
105     })
106     .catch(err => {
107       const error = new Error(err);
108       error.httpStatusCode = 500;
109       return next(error);
110     });
111 };
112
113 exports.postCart = (req, res, next) => {
114   const prodId = req.body.productId;
115   Product.findById(prodId)
116     .then(product => {
117       return req.user.addToCart(product);
118     })
119     .then(result => {
120       console.log(result);
121       res.redirect('/cart');
122     })
123     .catch(err => {
124       const error = new Error(err);
125       error.httpStatusCode = 500;
126       return next(error);
127     });
128 };
129
130 exports.postCartDeleteProduct = (req, res, next) => {
131   const prodId = req.body.productId;
132   req.user
133     .removeFromCart(prodId)
134     .then(result => {
135       res.redirect('/cart');
136     })
137     .catch(err => {
138       const error = new Error(err);
139       error.httpStatusCode = 500;
140       return next(error);
141     });
142 };
143
144 exports.getCheckout = (req, res, next) => {
145   req.user
146     .populate('cart.items.productId')
147     .execPopulate()

```

```

148 .then(user => {
149   const products = user.cart.items;
150   let total = 0
151   products.forEach(p => {
152     total += p.quantity * p.productId.price
153   })
154   res.render('shop/checkout', {
155     path: '/checkout',
156     pageTitle: 'Checkout',
157     products: products,
158     /**'products' is in an array of products
159      * where we have the quantity
160      * and a productId field with detail data about the products
161      * because we populate this productId field with the detailed product data
162      */
163     totalSum: total
164   });
165 })
166 .catch(err => {
167   const error = new Error(err);
168   error.httpStatusCode = 500;
169   return next(error);
170 });
171 }
172
173 exports.postOrder = (req, res, next) => {
174   req.user
175     .populate('cart.items.productId')
176     .execPopulate()
177     .then(user => {
178       const products = user.cart.items.map(i => {
179         return { quantity: i.quantity, product: { ...i.productId._doc } };
180       });
181       const order = new Order({
182         user: {
183           email: req.user.email,
184           userId: req.user
185         },
186         products: products
187       });
188       return order.save();
189     })
190     .then(result => {
191       return req.user.clearCart();
192     })
193     .then(() => {
194       res.redirect('/orders');
195     })
196     .catch(err => {
197       const error = new Error(err);
198       error.httpStatusCode = 500;
199       return next(error);
200     });
201 });
202
203 exports.getOrders = (req, res, next) => {

```

```

204 Order.find({ 'user.userId': req.user._id })
205   .then(orders => {
206     res.render('shop/orders', {
207       path: '/orders',
208       pageTitle: 'Your Orders',
209       orders: orders
210     });
211   })
212   .catch(err => {
213     const error = new Error(err);
214     error.httpStatusCode = 500;
215     return next(error);
216   });
217 };
218
219 exports.getInvoice = (req, res, next) => {
220   const orderId = req.params.orderId;
221   Order.findById(orderId)
222     .then(order => {
223       if (!order) {
224         return next(new Error('No order found.'));
225       }
226       if (order.user.userId.toString() !== req.user._id.toString()) {
227         return next(new Error('Unauthorized'));
228       }
229       const invoiceName = 'invoice-' + orderId + '.pdf';
230       const invoicePath = path.join('data', 'invoices', invoiceName);
231
232       const pdfDoc = new PDFDocument();
233       res.setHeader('Content-Type', 'application/pdf');
234       res.setHeader(
235         'Content-Disposition',
236         'inline; filename="' + invoiceName + '"'
237       );
238       pdfDoc.pipe(fs.createWriteStream(invoicePath));
239       pdfDoc.pipe(res);
240
241       pdfDoc.fontSize(26).text('Invoice', {
242         underline: true
243       });
244       pdfDoc.text('-----');
245       let totalPrice = 0;
246       order.products.forEach(prod => {
247         totalPrice += prod.quantity * prod.product.price;
248         pdfDoc
249           .fontSize(14)
250           .text(
251             prod.product.title +
252             ' - ' +
253             prod.quantity +
254             ' x ' +
255             '$' +
256             prod.product.price
257           );
258       });
259       pdfDoc.text('---');

```



```

260 pdfDoc.fontSize(20).text('Total Price: $' + totalPrice);
261
262 pdfDoc.end();
263 // fs.readFile(invoicePath, (err, data) => {
264 //   if (err) {
265 //     return next(err);
266 //   }
267 //   res.setHeader('Content-Type', 'application/pdf');
268 //   res.setHeader(
269 //     'Content-Disposition',
270 //     'inline; filename="' + invoiceName + '"'
271 //   );
272 //   res.send(data);
273 // });
274 // const file = fs.createReadStream(invoicePath);
275
276 // file.pipe(res);
277 })
278 .catch(err => next(err));
279 };
280

```

```

1 <!--./views/shop/cart.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4 <link rel="stylesheet" href="/css/cart.css">
5 </head>
6
7 <body>
8   <%- include('../includes/navigation.ejs') %>
9   <main>
10     <% if (products.length > 0) { %>
11       <ul class="cart__item-list">
12         <% products.forEach(p => { %>
13           <li class="cart__item">
14             <h1><%= p.productId.title %></h1>
15             <h2>Quantity: <%= p.quantity %></h2>
16             <form action="/cart-delete-item" method="POST">
17               <input type="hidden" value="<%= p.productId._id %>"
name="productId">
18               <input type="hidden" name="_csrf" value="<%= csrfToken %>">
19               <button class="btn danger" type="submit">Delete</button>
20             </form>
21           </li>
22         <% }) %>
23       </ul>
24       <hr>
25       <div class="centered">
26         <!--
27         <form action="/create-order" method="POST">
28           <input type="hidden" name="_csrf" value="<%= csrfToken %>">
29           <button type="submit" class="btn">Order Now!</button>
30         </form>
31         -->
32         <a class="btn" href="/checkout">Order Now!</a>
33       </div>
34

```

```

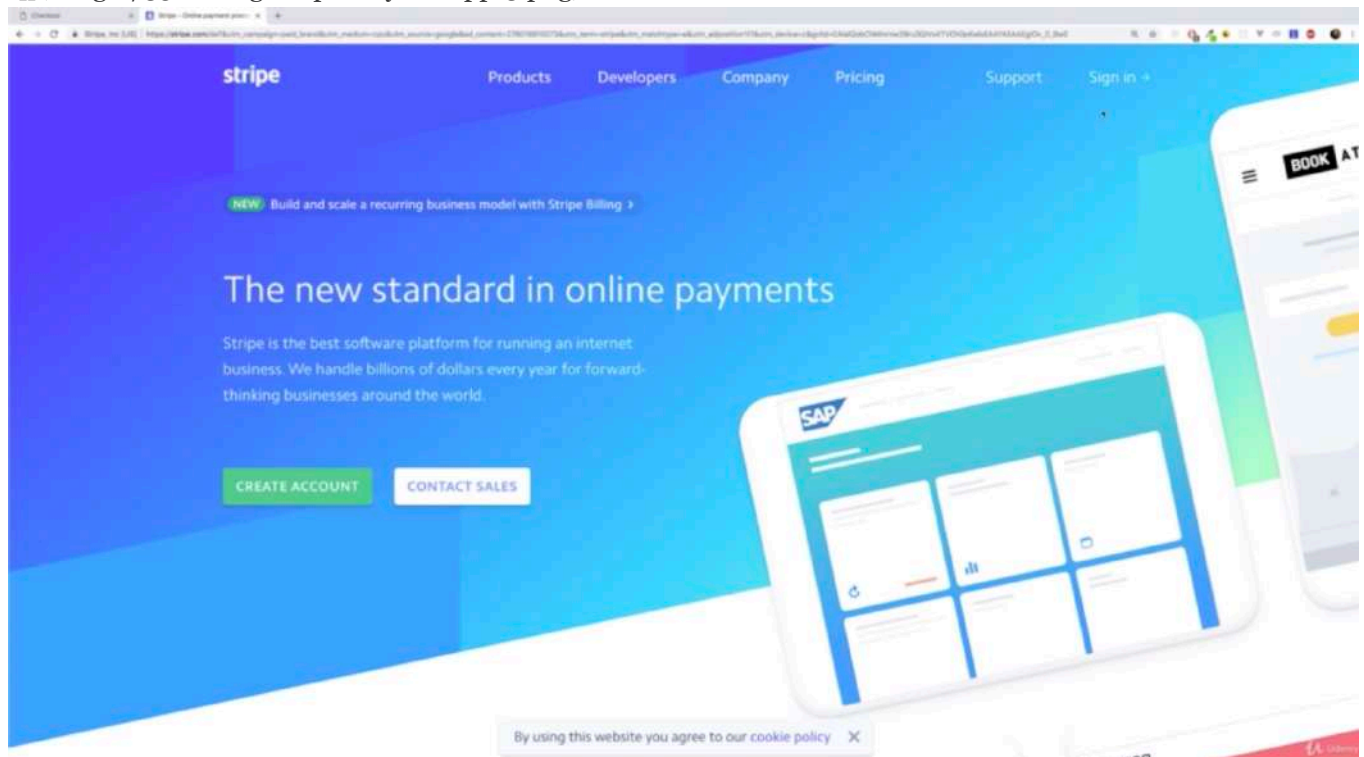
35     <% } else { %>
36         <h1>No Products in Cart!</h1>
37     <% } %>
38 </main>
39 <%- include('../includes/end.ejs') %>

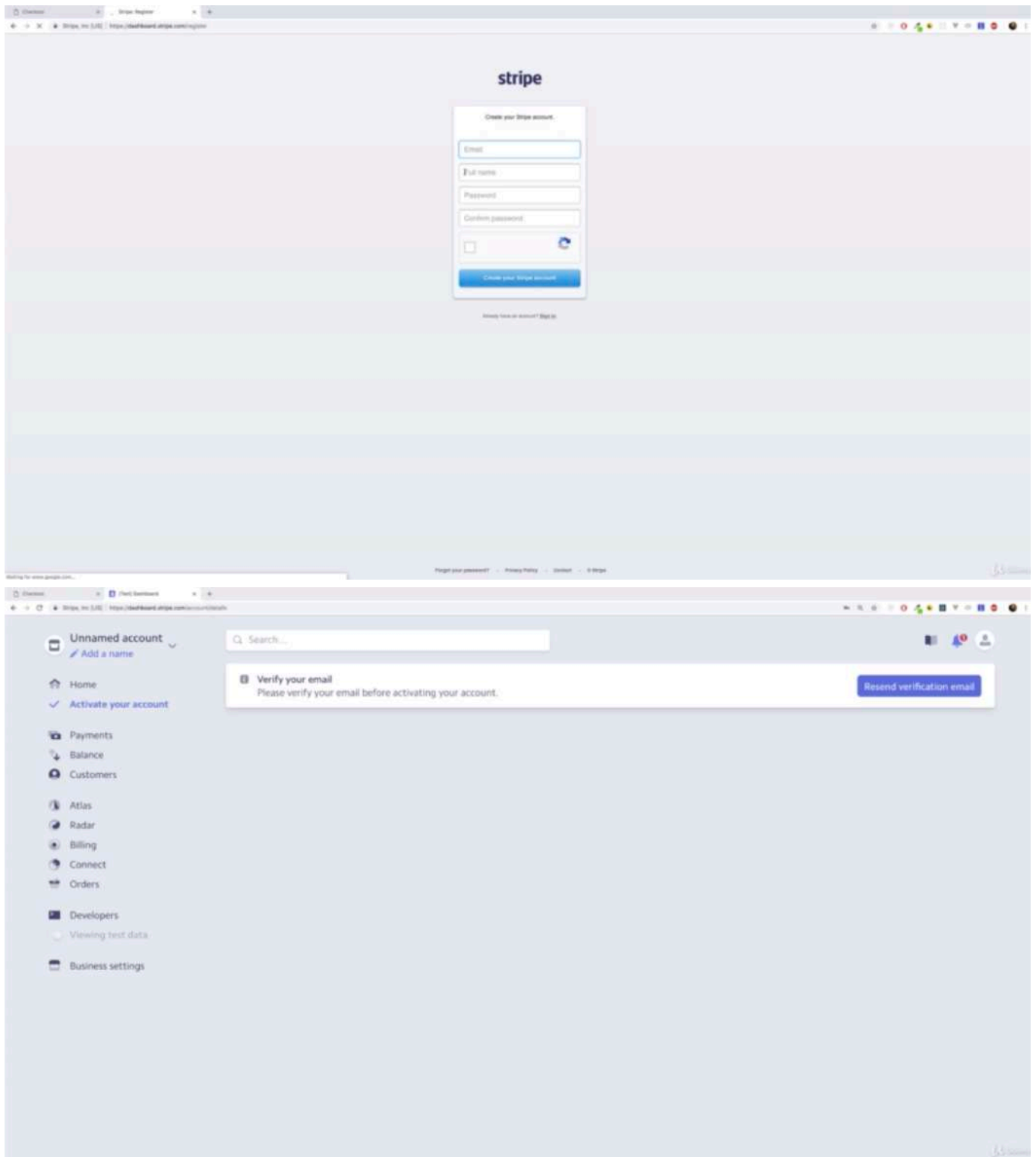
```

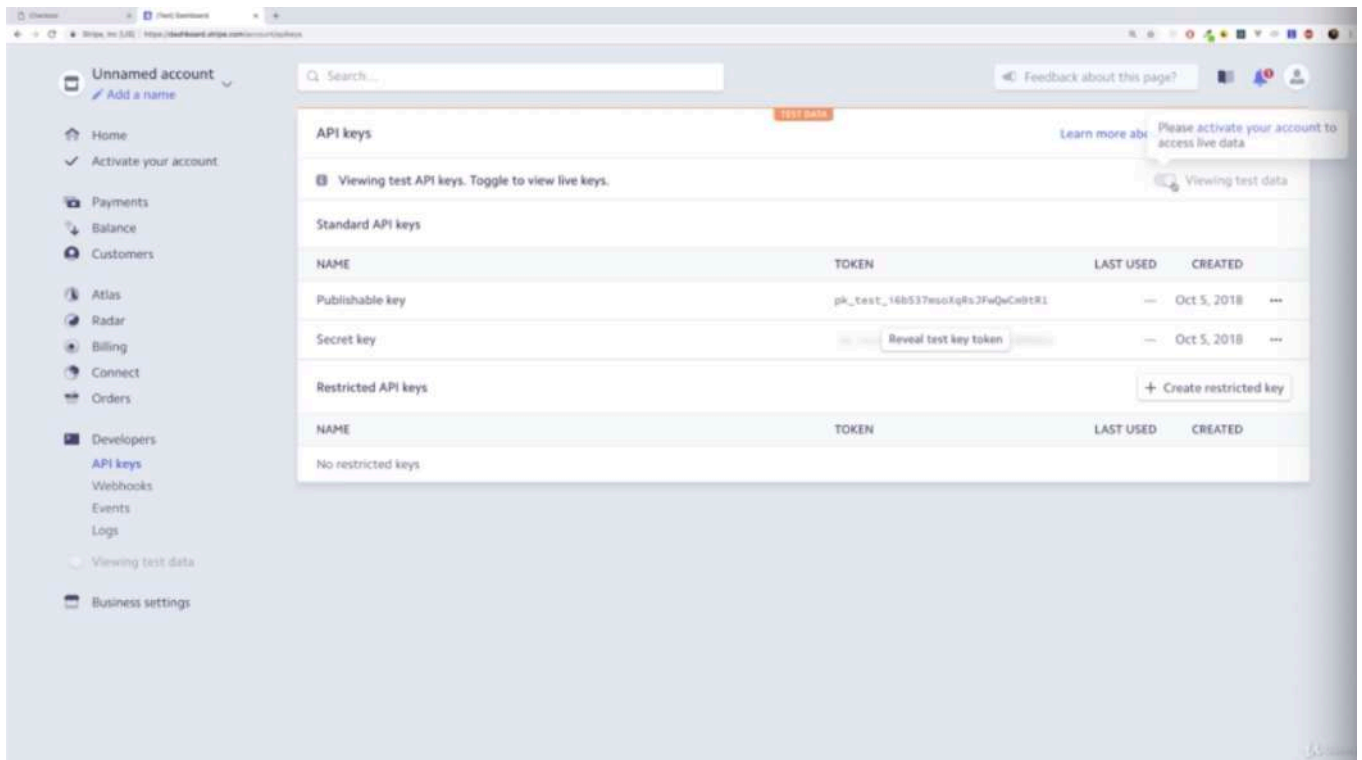
* Chapter 352: Using Stripe In Your App

1. update

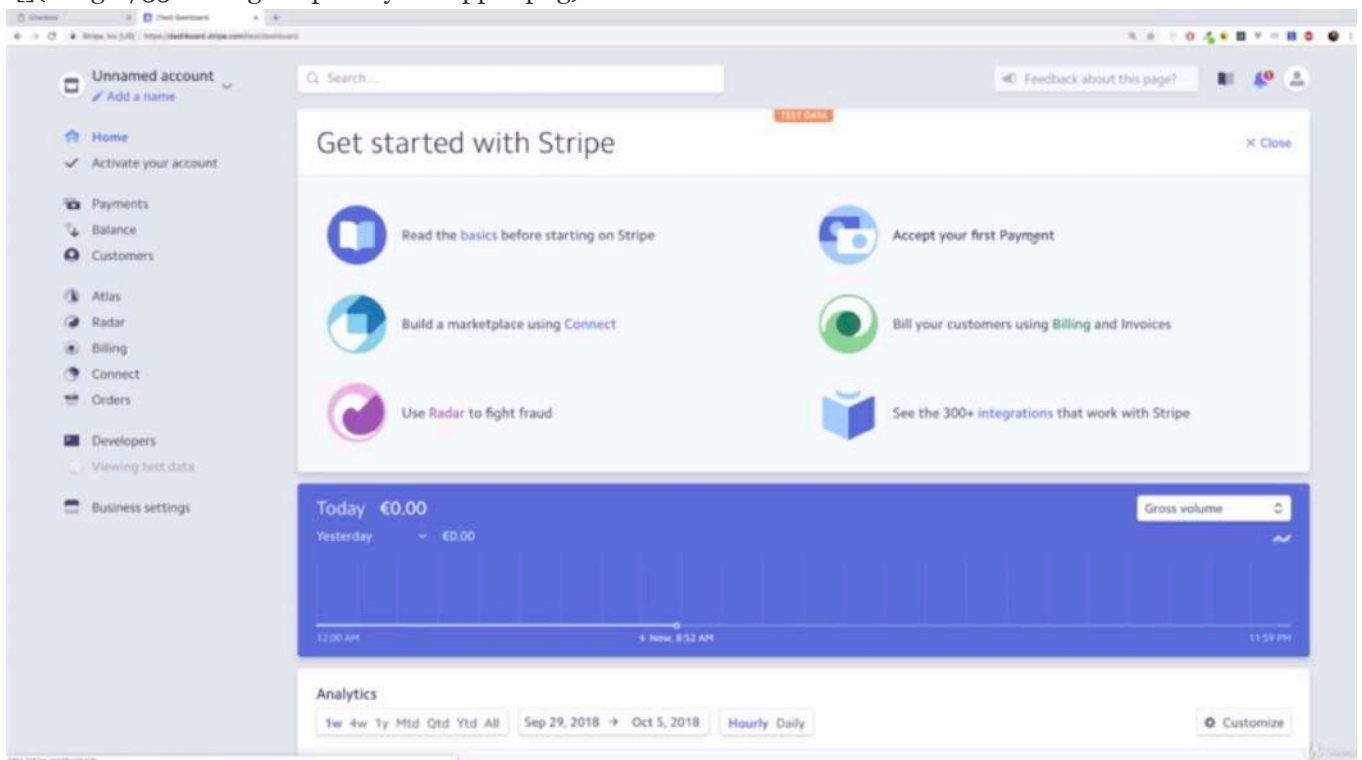
- ./views/shop/checkout.ejs
- ./controllers/shop.js
- app.js
- ./routes/shop.js

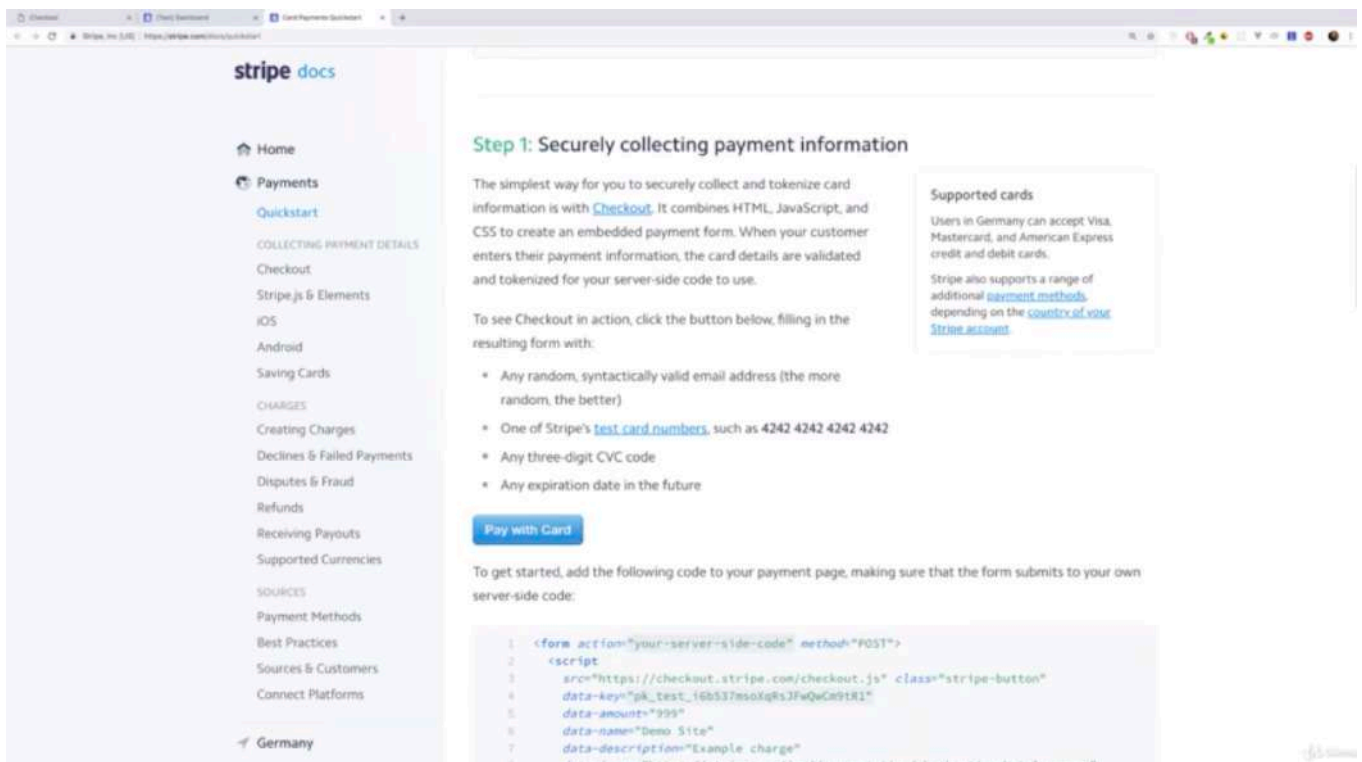




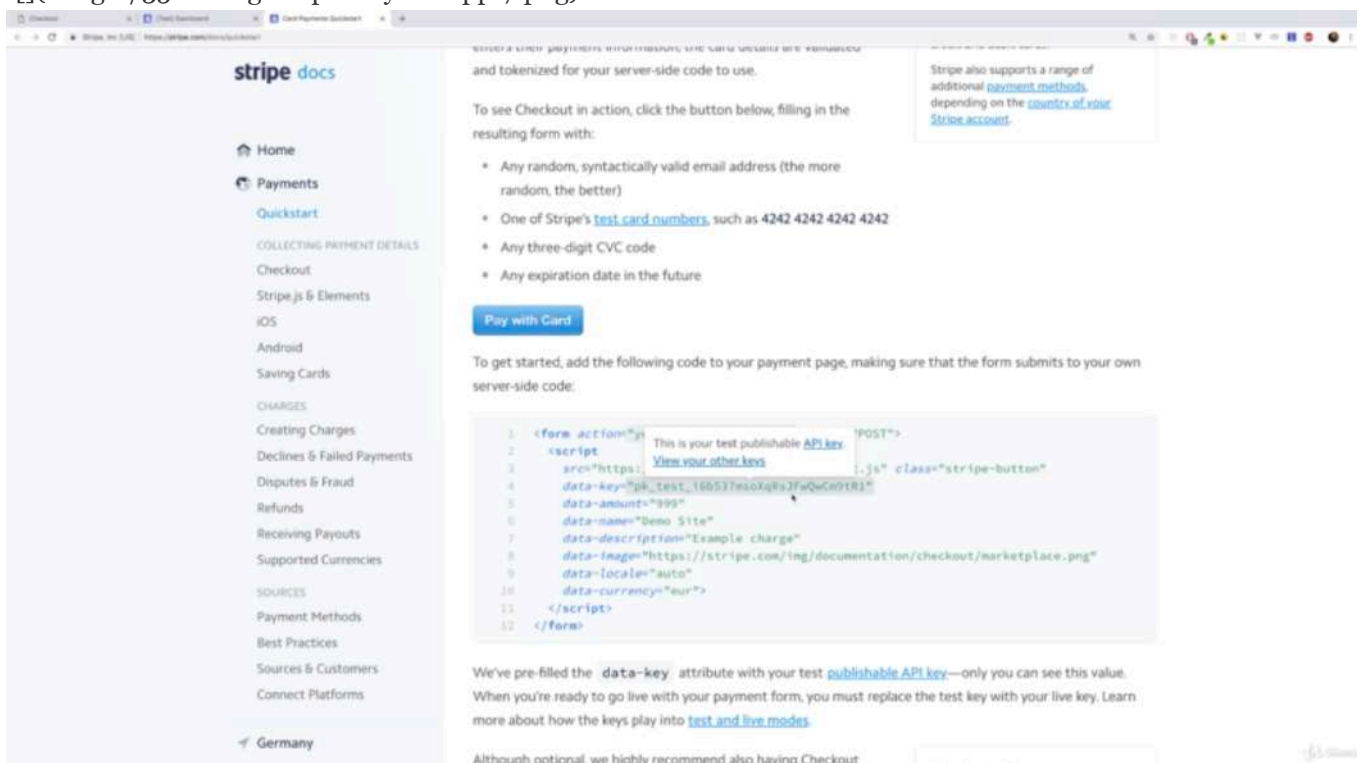


- if you wanna build a real application and you wanna push it to the production, you would switch to your live data here. for this, you have to act with your account. we will not do that here.

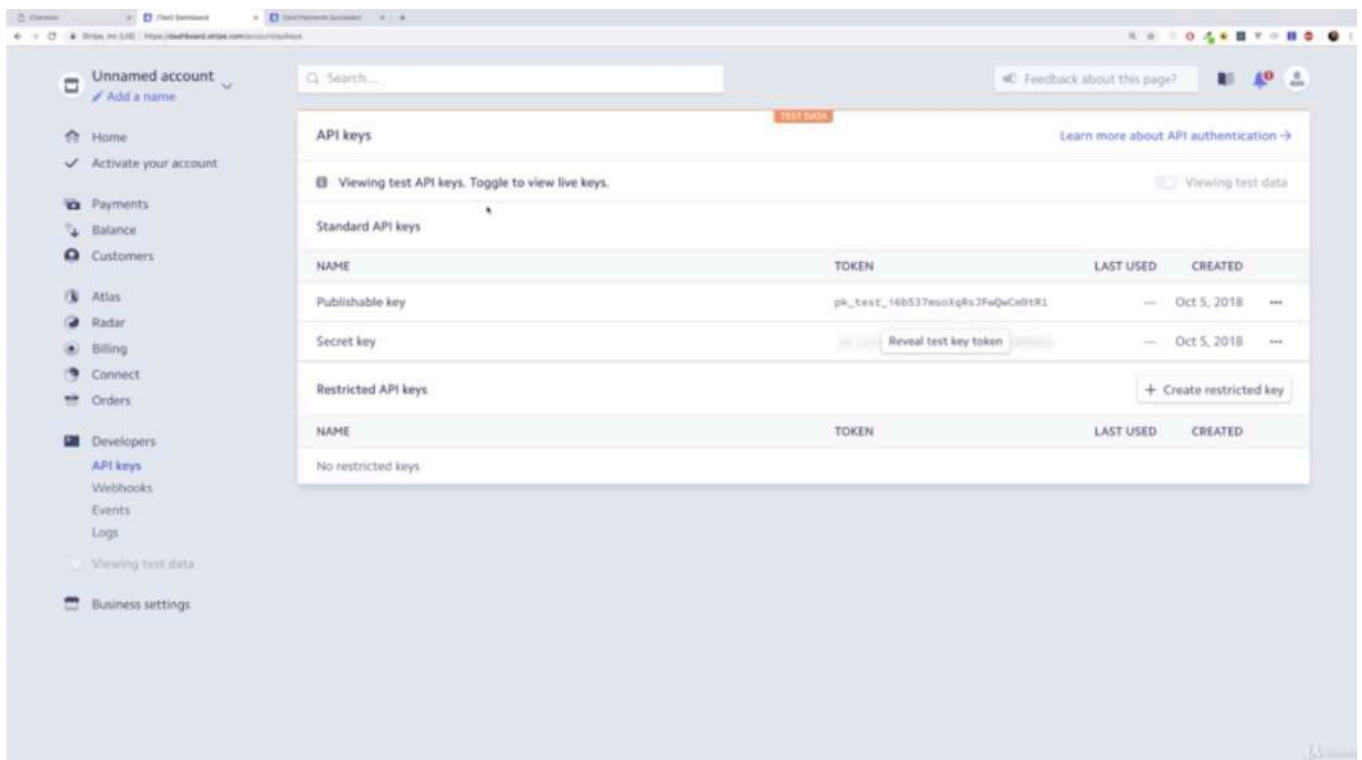




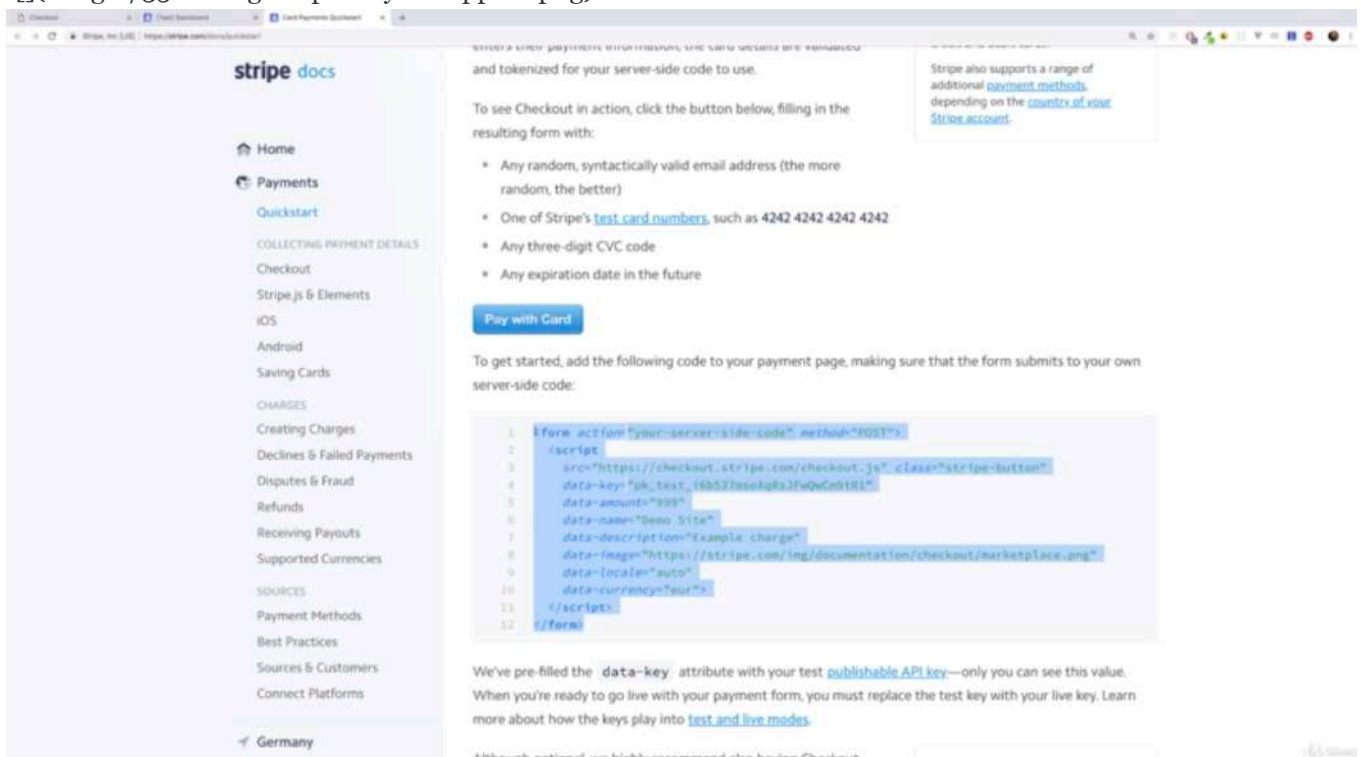
- go to 'Home' and click 'Accept your first Payment'. and then official Docs and learn how we can

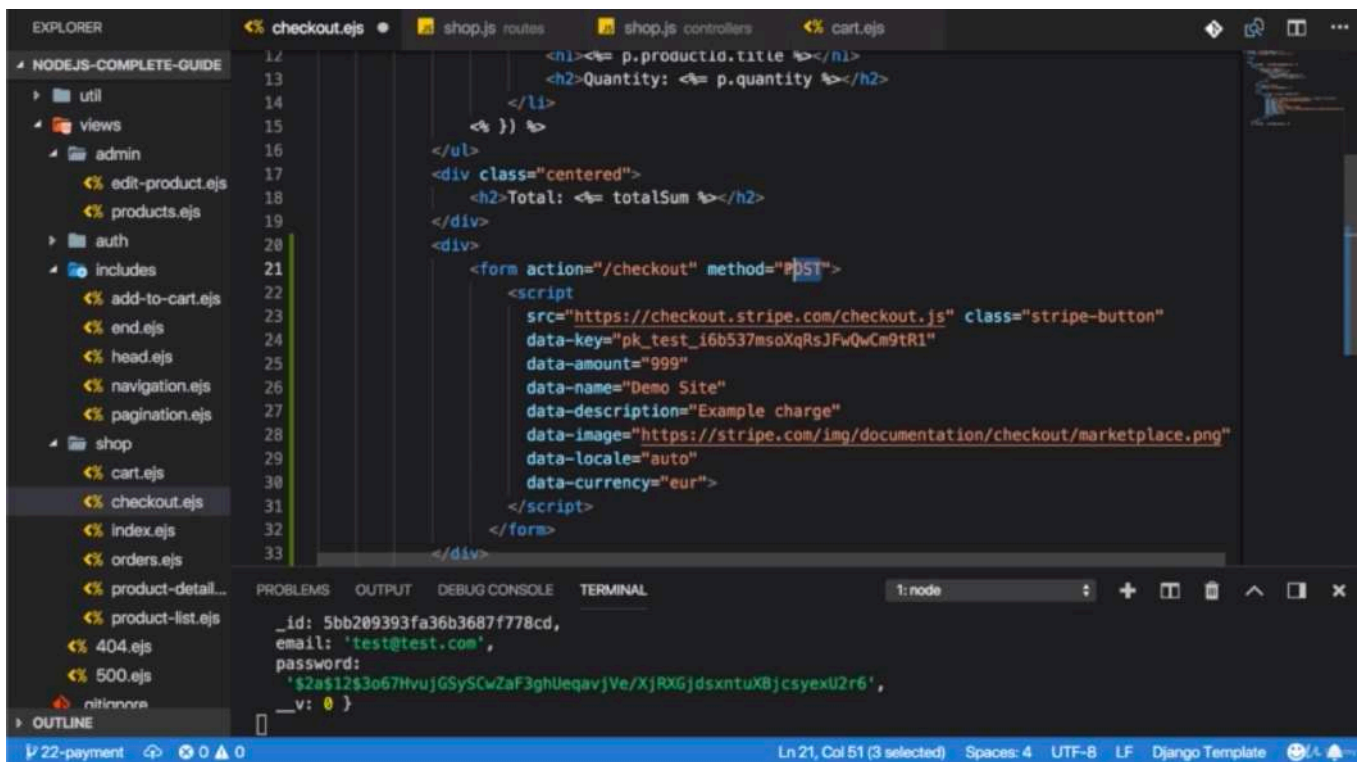


- the coll thing is that this data here is already populated with our data, with our publishable key for example,



- that key you saw under developers API keys, that are keys you need to send with your data to let stripe know to which account this belongs.





```
12 <h1><%= p.productId.title %></h1>
13
14 </li>
15 <%= }) %>
16 </ul>
17 <div class="centered">
18 <h2>Total: <%= totalSum %></h2>
19 </div>
20 <div>
21 <form action="/checkout" method="POST">
22 <script
23   src="https://checkout.stripe.com/checkout.js" class="stripe-button"
24   data-key="pk_test_i6b537msoXqRsJFwQwCm9tR1"
25   data-amount="999"
26   data-name="Demo Site"
27   data-description="Example charge"
28   data-image="https://stripe.com/img/documentation/checkout/marketplace.png"
29   data-locale="auto"
30   data-currency="eur">
31 </script>
32 </form>
33 </div>
```

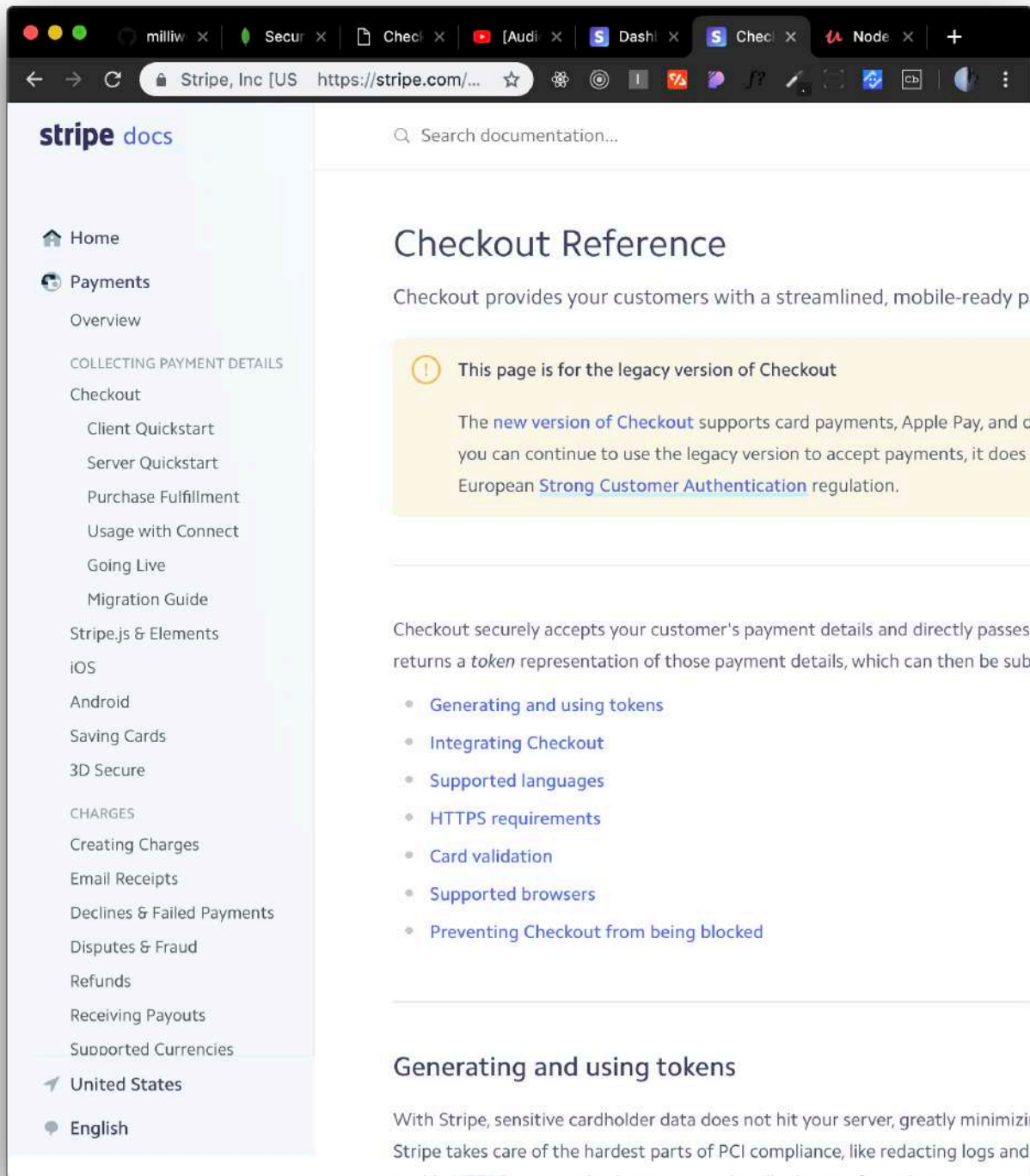
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
_id: 5bb209393fa36b3687f778cd,
email: 'test@test.com',
password:
'$2a$12$3o67HvuJGSySCwZaF3ghUeqavjVe/XjRXGjdsxntuXBjcsyexU2r6',
__v: 0 }
```

Ln 22, Col 15 (3 selected) Spaces: 4 UTF-8 LF Django Template

- a code snippet we can already use and we can add that to our `./views/shop/checkout.ejs` file



Custom. The custom integration lets you create a custom button and passes a Stripe [token \(API\)](#) to a JavaScript callback. Your JavaScript callback will need to send the token to your server for use.


[Simple](#) [Custom](#)

The simple integration uses a `<script>` tag inside your payment form to render the blue Checkout button. Upon completion of the Checkout process, Checkout submits your form to your server, passing along a `stripeToken` and any elements your form contains. When adding the following code to your page, make sure that the form submits to your own [server-side code](#) within the `action` attribute:

```
1 <form action="your-server-side-code" method="POST">
2   <script
3     src="https://checkout.stripe.com/checkout.js" class="stripe-button"
4     data-key="pk_test_ITc6cUJHHk47xmZcp3xRmtL00lh71Vh"
5     data-amount="999"
6     data-name="Demo Site"
7     data-description="Widget"
8     data-image="https://stripe.com/img/documentation/checkout/marketplace.png"
9     data-locale="auto">
10  </script>
11 </form>
```

We've prefilled the example with your [test API key](#). Only you can see this value.

Although optional, we highly recommend also having Checkout collect the user's ZIP code as [address and ZIP code verifications](#) help reduce fraud. Simply add `data-zip-code="true"` to the above and make use of Radar's [built-in rules](#) to decline payments that fail verification.

 Checkout must be loaded directly from <https://checkout.stripe.com/checkout.js>. Using a local copy of Checkout is unsupported, and may result in user-visible errors.

Received parameters

The following parameters are submitted to your form's action endpoint, along with any other elements in your form, once Checkout completes.

PARAMETER	DESCRIPTION
-----------	-------------

- update version is above. UI is changed.

)

)

)

)

)

)

)

)

Product #2

Quantity: 1

Total: 22.99


Pay with Card

Waiting for checkout stripe.com.

Shop Products Cart Orders Add Product Admin Products TEST MODE

Product #2

Quantity: 1



Your Order

All the items you ordered

Email

Card number


MM / YY CVC

☐ Remember me

Pay \$0.22

[Terms](#) | [Privacy](#)

Powered by stripe



EXPLORER checkout.ejs x shop.js routes shop.js controllers cart.ejs

NODEJS-COMPLETE-GUIDE

- util
- views
 - admin
 - edit-product.ejs
 - products.ejs
 - auth
 - includes
 - add-to-cart.ejs
 - end.ejs
 - head.ejs
 - navigation.ejs
 - pagination.ejs
 - shop
 - cart.ejs
 - checkout... M
 - index.ejs
 - orders.ejs
 - product-detail...
 - product-list.ejs
 - 404.ejs
 - 500.ejs
 - nitinnrra

13<h2>Quantity: <%= p.quantity %></h2>

14

15<%= %>

16

17<div class="centered">

18<h2>Total: <%= totalSum %></h2>

19</div>

20<div class="centered">

21<form action="/create-order" method="POST">

22<script

23src="https://checkout.stripe.com/checkout.js" class="stripe-button"

24data-key="pk_test_16b537msoXqRsJFwQwCm9tR1"

25data-amount="<%= totalSum * 100 %>"

26data-name="Your Order"

27data-description="All the items you ordewred"

28data-image="https://stripe.com/img/documentation/checkout/marketplace.png"

29data-locale="auto"

30data-currency="usd">

31</script>

32</form>

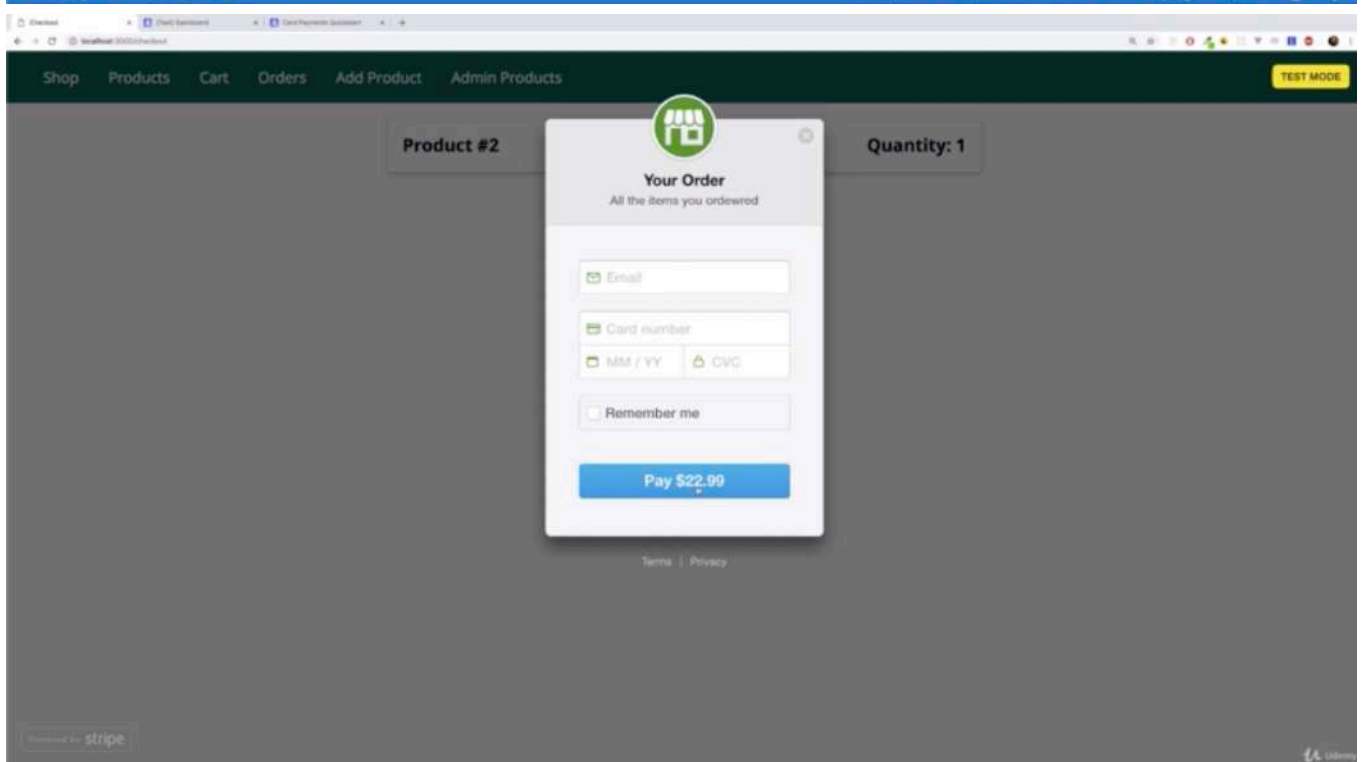
33</div>

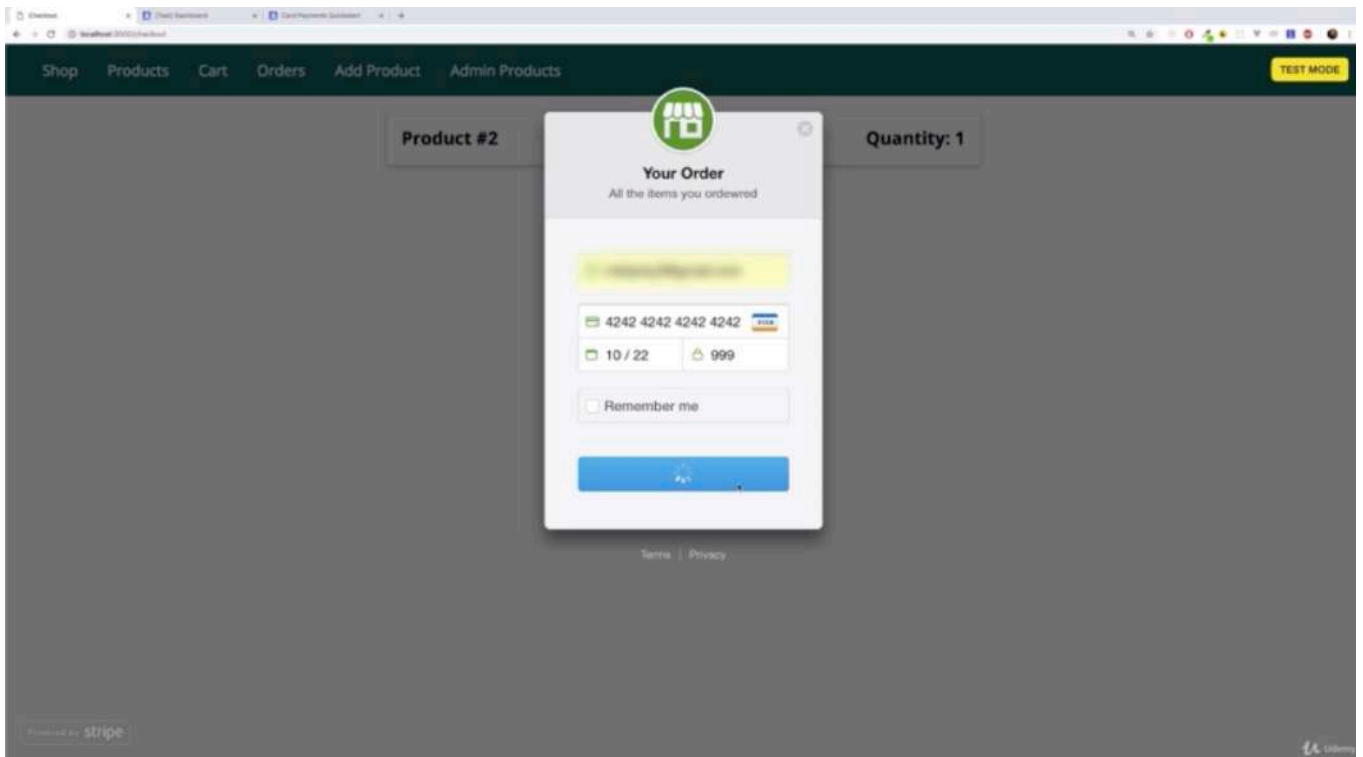
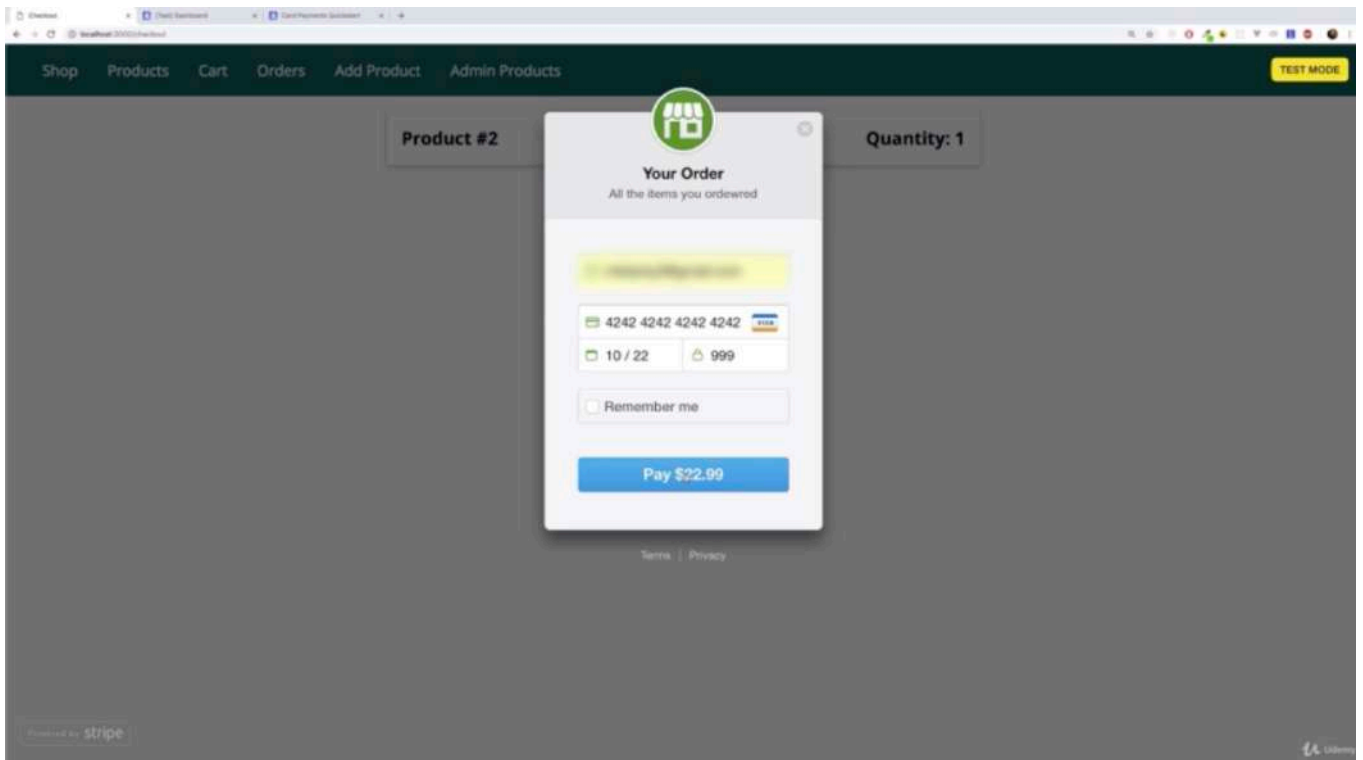
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

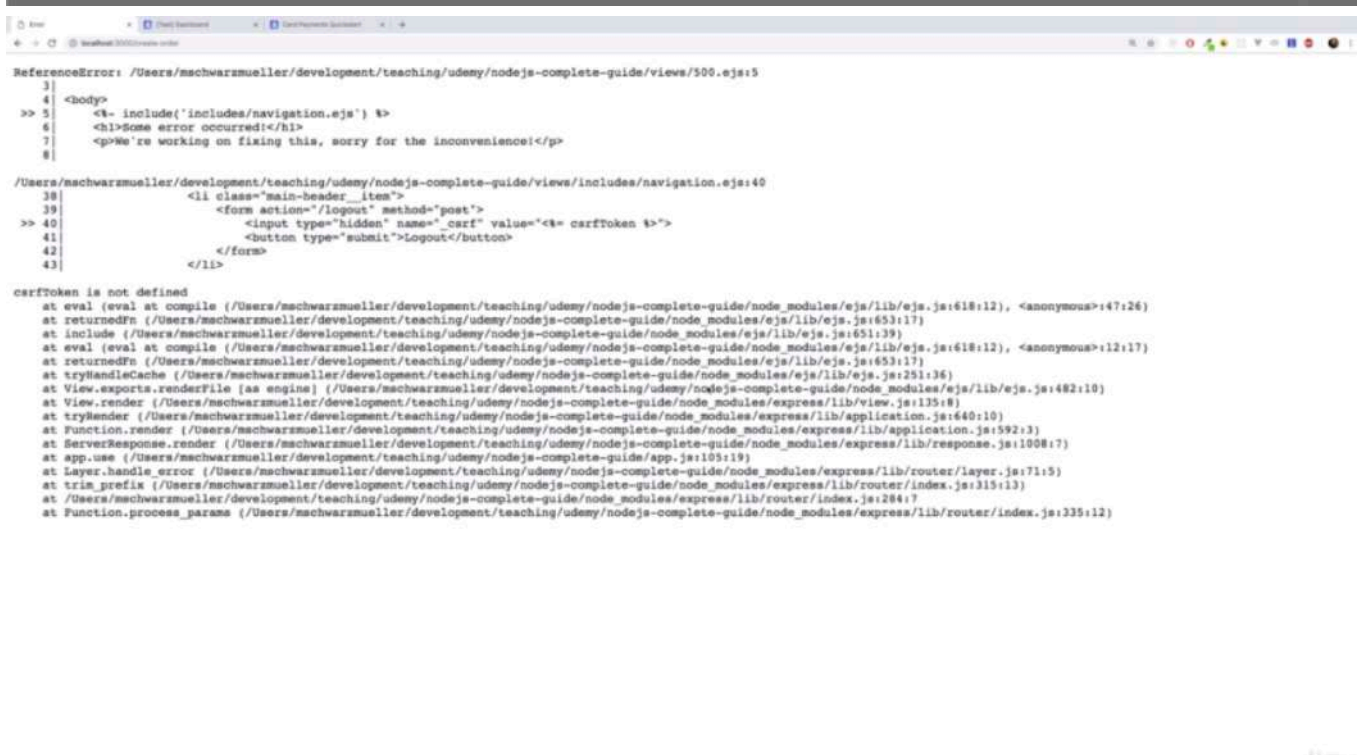
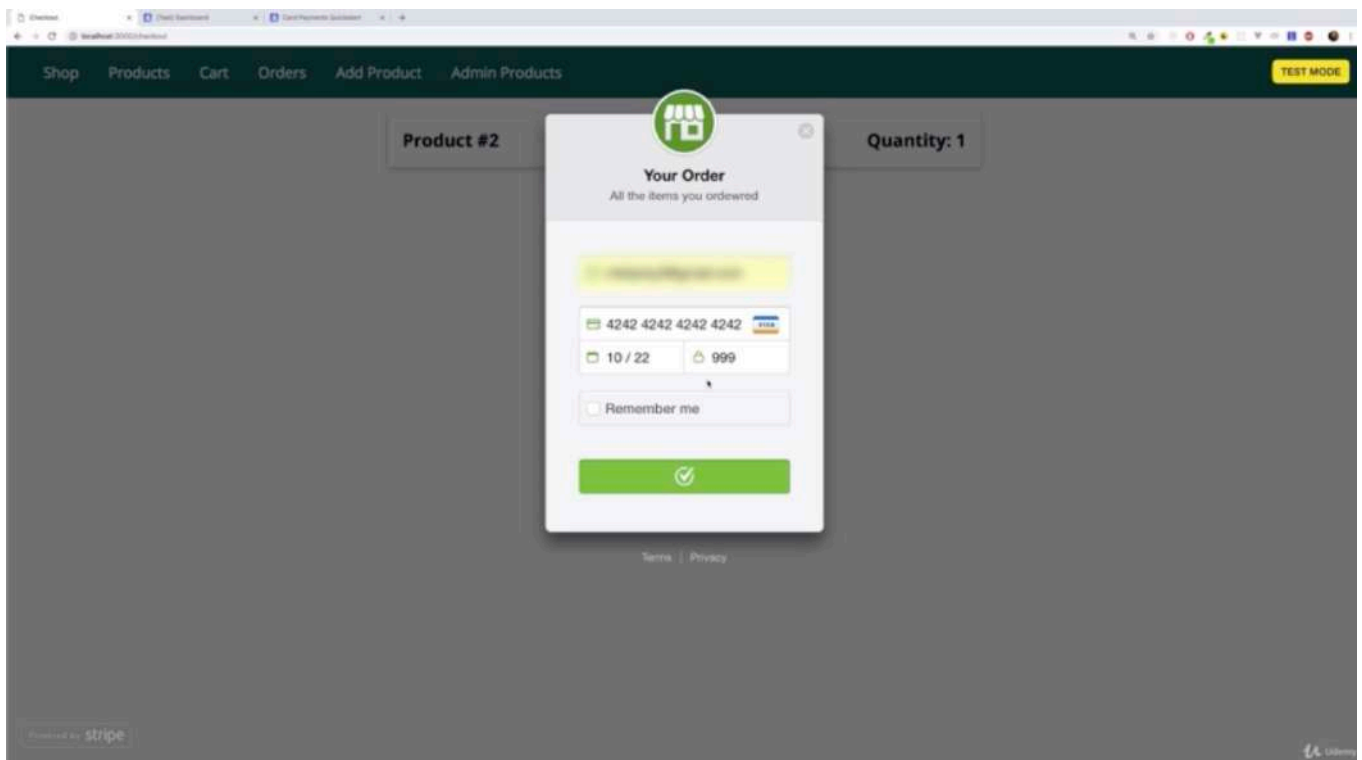
1: node

_id: 5bb209393fa36b3687f778cd,
email: 'test@test.com',
password:
'\$2a\$12\$3o67HvuJGSySCwZaF3ghUeqavjVe/XjRXGjdsxntuXBjcsyexU2r6',
__v: 0 }

Ln 26, Col 45 Spaces: 4 UTF-8 LF Django Template







- now this would send a request to the backend, to the create-order route and there we have to continue because now there we will receive some special data and how to continue is something stripe tells you.

stripe docs

Home

Payments

Quickstart

COLLECTING PAYMENT DETAILS

Checkout

Stripe.js & Elements

iOS

Android

Saving Cards

CHARGES

Creating Charges

Declines & Failed Payments

Disputes & Fraud

Refunds

Receiving Payouts

Supported Currencies

SOURCES

Payment Methods

Best Practices

Sources & Customers

Connect Platforms

Germany

Using our [native mobile libraries](#) for iOS and Android, Stripe can collect your customer's payment information from within your mobile app and create a token for your server-side code to use.

Step 2: Creating a charge to complete the payment

Once a token is created, your server-side code makes an API request to create a one-time charge. This request contains the token, currency, amount to charge, and any additional information you may want to pass (e.g., [metadata](#)).

curl Ruby Python PHP Java Node Go .NET

```
1 // Set your secret key: remember to change this to your live secret key in production
2 // See your keys here: https://dashboard.stripe.com/account/apikeys
3 var stripe = require("stripe")("sk_test_BMD9aaviqJzK0h1R0g2KMRbD");
4
5 // Token is created using Checkout or Elements!
6 // Get the payment token ID submitted by the form:
7 const token = request.body.stripeToken; // Using Express
8
9 const charge = stripe.charges.create({
10   amount: 999,
11   currency: 'usd',
12   description: 'Example charge',
13   source: token,
14 });
```

These requests expect the ID of the [Token](#) (e.g., tok_KPte7942xySKBKyrBu1fyEpf) to be provided as the value of the `source` parameter.

Tokens can only be used once, and within a few minutes of creation. Using this approach, your customers need to re-enter their payment details each time they make a purchase. You can also [save card details](#) with Stripe for later use. [Using this method](#), returning customers can re-enter a payment without re-authenticating their card.

EXPLORER

NODE.JS-COMPLETE-GUIDE

.vscode

controllers

admin.js

auth.js

error.js

shop.js

data

images

middleware

models

node_modules

public

routes

admin.js

auth.js

shop.js

util

views

admin

edit-product.ejs

products.ejs

auth

OUTLINE

checkout.ejs

app.js

shop.js routes

shop.js controllers

cart.ejs

```
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

[nodemon] restarting due to changes...

[nodemon] restarting due to changes...

[nodemon] starting 'node app.js'

(node:62175) 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.

Ln 178, Col 4 Spaces: 2 UTF-8 LF JavaScript Prettier: ✓

- you can see how to handle the data on the incoming request and you will see that the incoming request will have a stripe token in its body.

EXPLORER

NODEJS-COMPLETE-GUIDE

.vscode

controllers

admin.js

auth.js

error.js

shop.js

data

images

middleware

models

node_modules

public

routes

admin.js

auth.js

shop.js

util

views

admin

edit-product.ejs

products.ejs

auth

OUTLINE

checkout.ejs

app.js

shop.js routes

shop.js controllers

cart.ejs

```
160
161   const error = new Error(err);
162   error.httpStatusCode = 500;
163   return next(error);
164   });
165
166   exports.postOrder = (req, res, next) => {
167     const stripe = require('stripe')('sk_test_BMD9aaviqJzK0hLR0g2KMRbD');
168
169     // Token is created using Checkout or Elements!
170     // Get the payment token ID submitted by the form:
171     const token = request.body.stripeToken; // Using Express
172
173     const charge = stripe.charges.create({
174       amount: 999,
175       currency: 'usd',
176       description: 'Example charge',
177       source: token
178     });
179     req.user
180       .populate('cart.items.productId')
181       .execPopulate()
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

1: node

[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:62175) 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.
^CMaximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm install --save stripe
() loadIdealTree:loadAllDepsIntoIdealTree: sill install loadIdealTree

Ln 167, Col 34 (8 selected)

Spaces: 2

UTF-8

LF

JavaScript

Prettier: ✓

Shop

Products

Cart

Orders

Add Product

Admin Products

TEST MODE

Product #2

Quantity: 1

Your Order

All the items you ordered

4242 4242 4242 4242

10 / 22

999

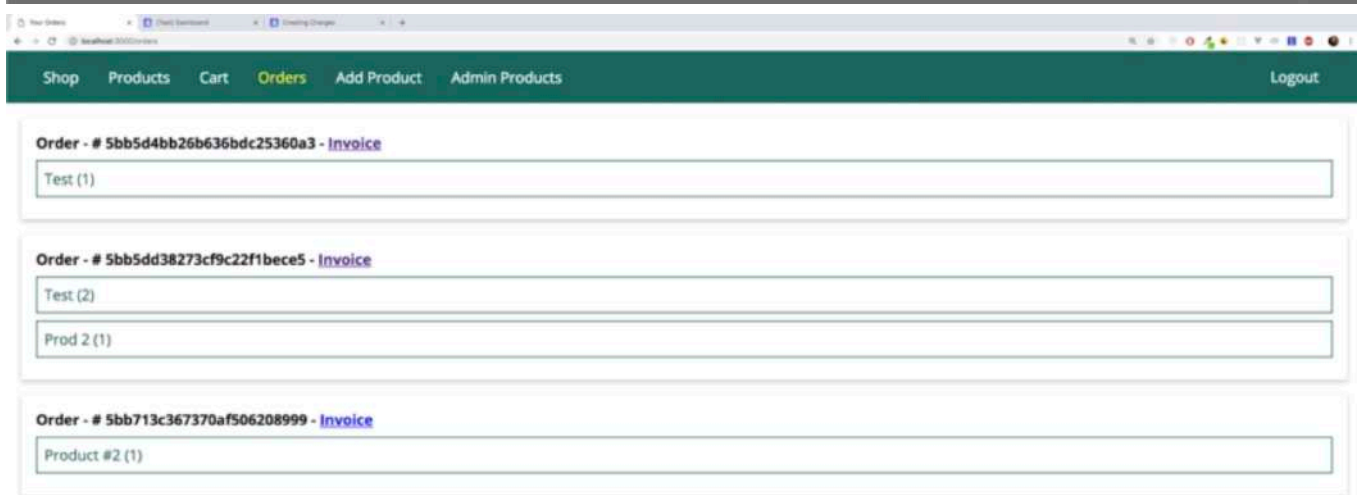
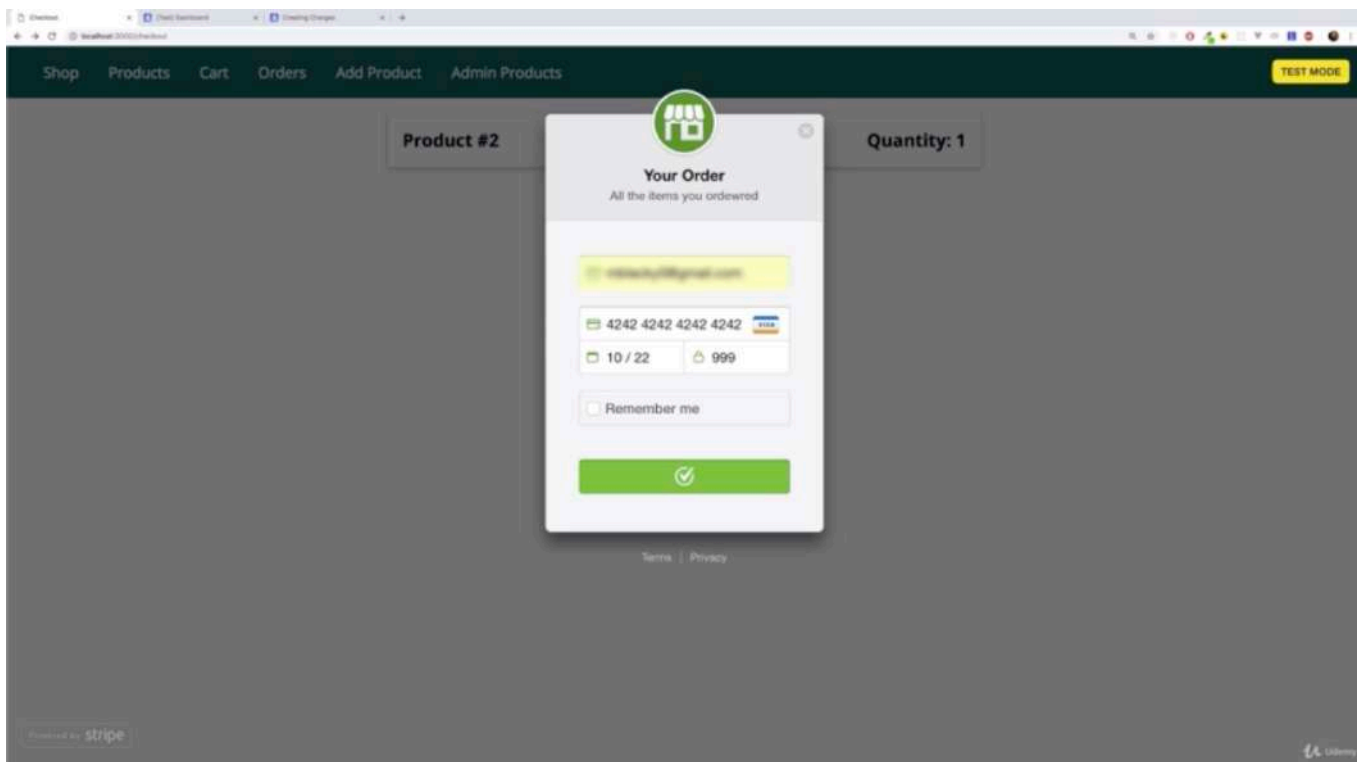
Remember me

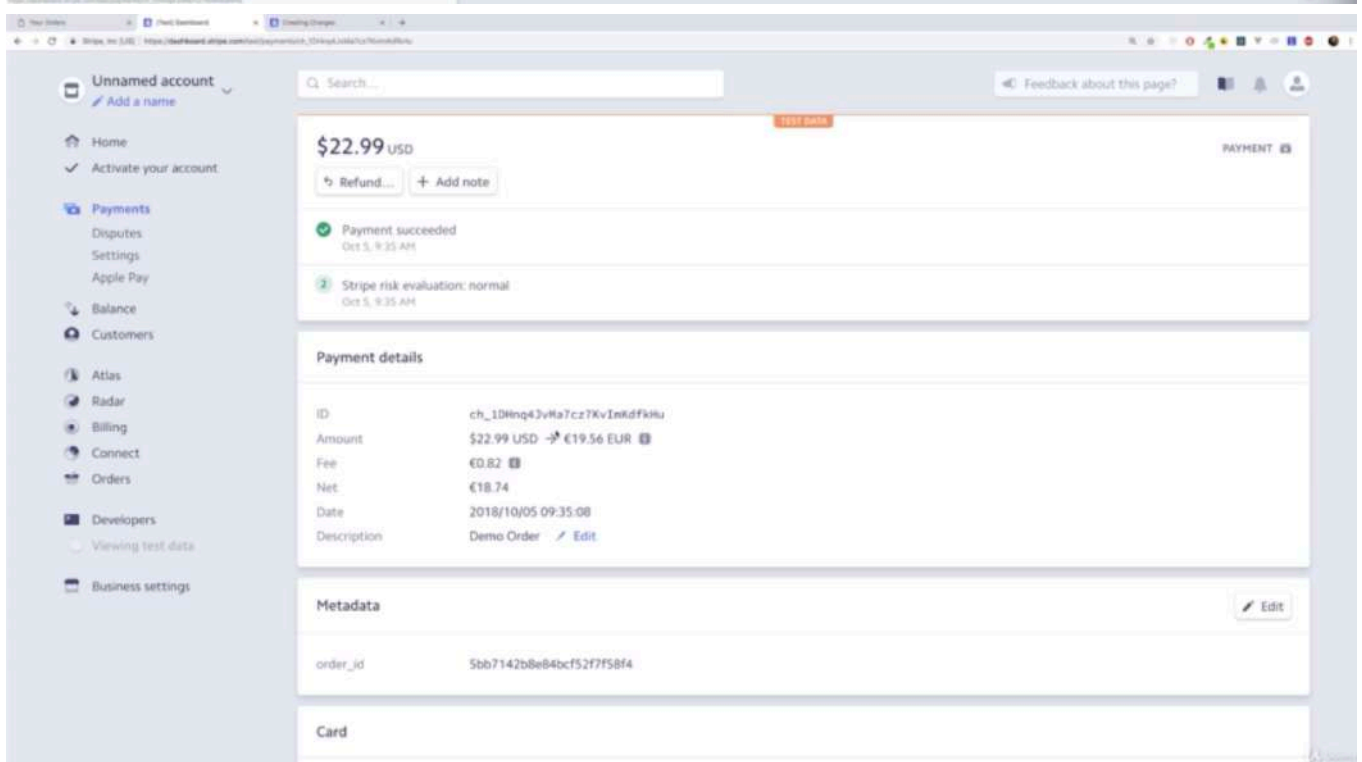
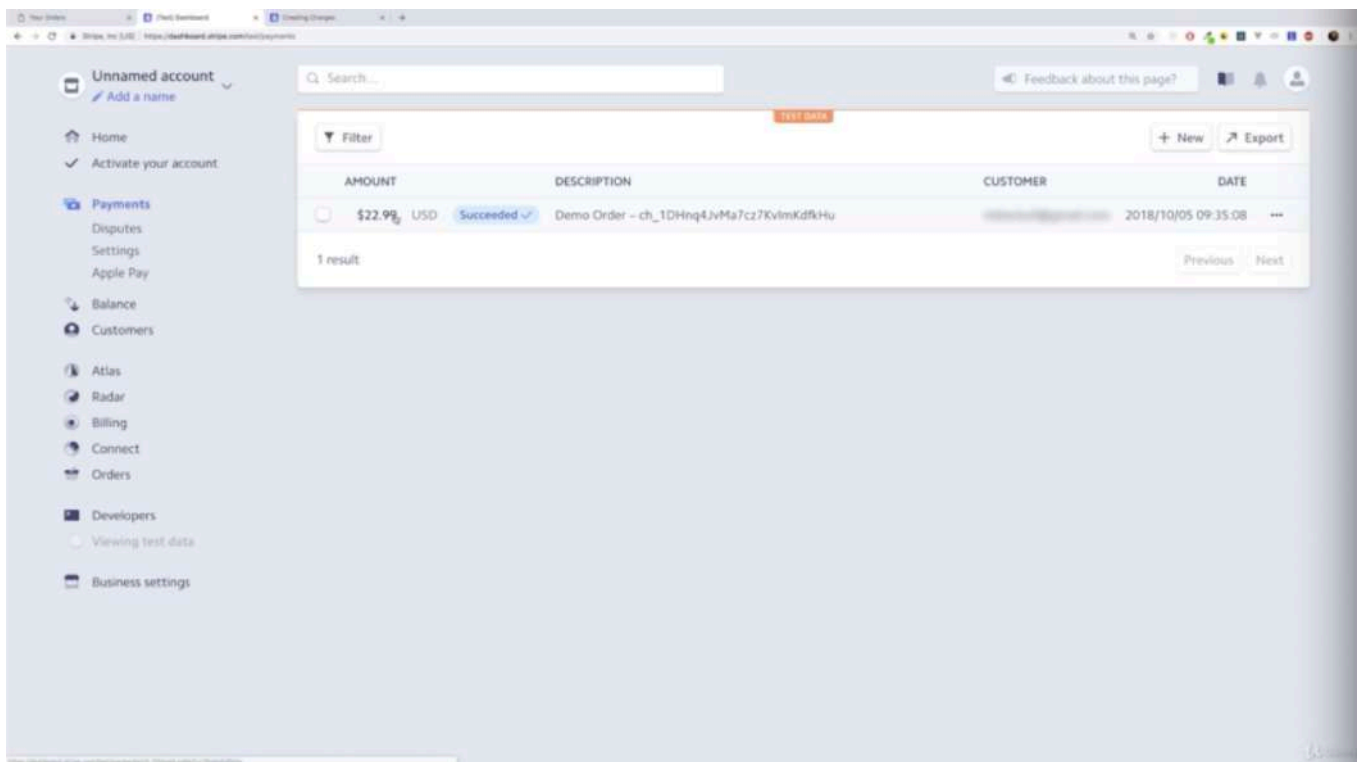
Terms

Privacy

Powered by stripe

Udemy





```

1 <!--./views/shop/checkout.ejs-->
2
3 <%- include('../includes/head.ejs') %>
4   <link rel="stylesheet" href="/css/cart.css">
5
6   </head>
7
8   <body>
9     <%- include('../includes/navigation.ejs') %>
10    <main>
11      <ul class="cart__item-list">
12        <% products.forEach(p => { %>
13          <li class="cart__item">
14            <h1><%= p.productId.title %></h1>

```

```

15         <h2>Quantity: <%= p.quantity %></h2>
16     </li>
17 <% }) %>
18 </ul>
19 <div class="centered">
20     <h2>Total: <%= totalSum %></h2>
21 </div>
22 <div class="centered">
23     <!--
24         from app.js file,
25         for the request sent by the stripe modal,
26         by that modal we entered our credit card data in,
27         this request doesn't include our CSRF token.
28         so '/checkout' form doesn't include it.
29
30         you could include your hidden input field
31         but that wouldn't do anything
32         because stripe creates its own form in the end in an ifram which it
loads
33         but that also means that stripe takes care about securing that form
34         so this is not this form
35         which you embed into your document there,
36         stripe renders and loads and secures its own form
37         so you don't need to pass the CSRF token.
38
39         so this just means we have to disable CSRF token protection for this
POST create-order route
40         to do that, i will cut the route from my ./routes/shop.js
41         and i will move it directly into app.js
42     -->
43     <form action="/create-order" method="POST">
44         <script
45             src="https://checkout.stripe.com/checkout.js" class="stripe-button"
46             data-key="pk_test_ITc6cUJHHk47xmZcpx3xRmtL00lhse7lVh"
47             data-amount="<%= totalSum %>"
48             data-name="Your Order"
49             data-description="All the items you ordered"
50             data-
image="https://stripe.com/img/documentation/checkout/marketplace.png"
51             data-locale="auto"
52             data-currency="usd">
53         </script>
54     </form>
55 </div>
56 </main>
<%- include('../includes/end.ejs') %>

```

```

1 //./controllers/shop.js
2
3 const fs = require('fs');
4 const path = require('path');
5
6 const PDFDocument = require('pdfkit');
7 const stripe = require("stripe")("sk_test_dAPYh6CKcipsTX13kbj0Fk1T00NICMWhhg");
8
9 const Product = require('../models/product');
10 const Order = require('../models/order');
11

```

```

12 const ITEMS_PER_PAGE = 2;
13
14 exports.getProducts = (req, res, next) => {
15   const page = +req.query.page || 1;
16   let totalItems;
17
18   Product.find()
19     .countDocuments()
20     .then(numProducts => {
21       totalItems = numProducts;
22       return Product.find()
23         .skip((page - 1) * ITEMS_PER_PAGE)
24         .limit(ITEMS_PER_PAGE);
25     })
26     .then(products => {
27       res.render('shop/product-list', {
28         prods: products,
29         pageTitle: 'Products',
30         path: '/products',
31         currentPage: page,
32         hasNextPage: ITEMS_PER_PAGE * page < totalItems,
33         hasPreviousPage: page > 1,
34         nextPage: page + 1,
35         previousPage: page - 1,
36         lastPage: Math.ceil(totalItems / ITEMS_PER_PAGE)
37       });
38     })
39     .catch(err => {
40       const error = new Error(err);
41       error.httpStatusCode = 500;
42       return next(error);
43     });
44 };
45
46 exports.getProduct = (req, res, next) => {
47   const prodId = req.params.productId;
48   Product.findById(prodId)
49     .then(product => {
50       res.render('shop/product-detail', {
51         product: product,
52         pageTitle: product.title,
53         path: '/products'
54       });
55     })
56     .catch(err => {
57       const error = new Error(err);
58       error.httpStatusCode = 500;
59       return next(error);
60     });
61 };
62
63 exports.getIndex = (req, res, next) => {
64   const page = +req.query.page || 1;
65   let totalItems;
66
67   Product.find()

```

```

68     .countDocuments()
69     .then(numProducts => {
70         totalItems = numProducts;
71         return Product.find()
72             .skip((page - 1) * ITEMS_PER_PAGE)
73             .limit(ITEMS_PER_PAGE);
74     })
75     .then(products => {
76         res.render('shop/index', {
77             prods: products,
78             pageTitle: 'Shop',
79             path: '/',
80             currentPage: page,
81             hasNextPage: ITEMS_PER_PAGE * page < totalItems,
82             hasPreviousPage: page > 1,
83             nextPage: page + 1,
84             previousPage: page - 1,
85             lastPage: Math.ceil(totalItems / ITEMS_PER_PAGE)
86         });
87     })
88     .catch(err => {
89         const error = new Error(err);
90         error.httpStatusCode = 500;
91         return next(error);
92     });
93 };
94
95 exports.getCart = (req, res, next) => {
96     req.user
97         .populate('cart.items.productId')
98         .execPopulate()
99         .then(user => {
100             const products = user.cart.items;
101             res.render('shop/cart', {
102                 path: '/cart',
103                 pageTitle: 'Your Cart',
104                 products: products
105             });
106         })
107         .catch(err => {
108             const error = new Error(err);
109             error.httpStatusCode = 500;
110             return next(error);
111         });
112 };
113
114 exports.postCart = (req, res, next) => {
115     const prodId = req.body.productId;
116     Product.findById(prodId)
117         .then(product => {
118             return req.user.addToCart(product);
119         })
120         .then(result => {
121             console.log(result);
122             res.redirect('/cart');
123         })

```

```

124     .catch(err => {
125         const error = new Error(err);
126         error.httpStatusCode = 500;
127         return next(error);
128     });
129 };
130
131 exports.postCartDeleteProduct = (req, res, next) => {
132     const prodId = req.body.productId;
133     req.user
134         .removeFromCart(prodId)
135         .then(result => {
136             res.redirect('/cart');
137         })
138         .catch(err => {
139             const error = new Error(err);
140             error.httpStatusCode = 500;
141             return next(error);
142         });
143 };
144
145 exports.getCheckout = (req, res, next) => {
146     req.user
147         .populate('cart.items.productId')
148         .execPopulate()
149         .then(user => {
150             const products = user.cart.items;
151             let total = 0
152             products.forEach(p => {
153                 total += p.quantity * p.productId.price
154             })
155             res.render('shop/checkout', {
156                 path: '/checkout',
157                 pageTitle: 'Checkout',
158                 products: products,
159                 totalSum: total
160             });
161         })
162         .catch(err => {
163             const error = new Error(err);
164             error.httpStatusCode = 500;
165             return next(error);
166         });
167 }
168
169 exports.postOrder = (req, res, next) => {
170     // Token is created using Checkout or Elements!
171     // Get the payment token ID submitted by the form:
172     const token = req.body.stripeToken; // Using Express
173     let totalSum = 0
174
175     req.user
176         .populate('cart.items.productId')
177         .execPopulate()
178         .then(user => {
179             user.cart.items.forEach(p => {

```

```

180     totalSum += p.quantity * p.productId.price
181   })
182   const products = user.cart.items.map(i => {
183     return { quantity: i.quantity, product: { ...i.productId._doc } };
184   });
185   const order = new Order({
186     user: {
187       email: req.user.email,
188       userId: req.user
189     },
190     products: products
191   });
192   return order.save();
193 })
194 .then(result => {
195   const charge = stripe.charges.create({
196     amount: totalSum * 100,
197     currency: 'usd',
198     description: 'Demo Order',
199     /**The source is the token
200      * which we extract up
201      * so that token which includes the validated credit card data.
202      */
203     source: token,
204     /**some metadata which is a javascript object
205      * where you can pass any arbitrary data you wanna match this order with the charge
206      * that will be stored in your stripe account.
207      *
208      * 'result_id' should be Id of the created order.
209      *
210      * itry to send some extra data in my ./controllers/shop.js
211      * to stripe, the resultId should convert to string
212      * otherwise this can't be added
213      * and therefore my charge failed.
214      */
215     metadata: {order_id: result._id.toString()}
216   })
217   return req.user.clearCart();
218 })
219 .then(() => {
220   res.redirect('/orders');
221 })
222 .catch(err => {
223   const error = new Error(err);
224   error.statusCode = 500;
225   return next(error);
226 });
227 };
228
229 exports.getOrders = (req, res, next) => {
230   Order.find({ 'user.userId': req.user._id })
231     .then(orders => {
232       res.render('shop/orders', {
233         path: '/orders',
234         pageTitle: 'Your Orders',
235         orders: orders

```

```

236     });
237   })
238   .catch(err => {
239     const error = new Error(err);
240     error.httpStatusCode = 500;
241     return next(error);
242   });
243 };
244
245 exports.getInvoice = (req, res, next) => {
246   const orderId = req.params.orderId;
247   Order.findById(orderId)
248     .then(order => {
249       if (!order) {
250         return next(new Error('No order found.'));
251       }
252       if (order.user.userId.toString() !== req.user._id.toString()) {
253         return next(new Error('Unauthorized'));
254       }
255       const invoiceName = 'invoice-' + orderId + '.pdf';
256       const invoicePath = path.join('data', 'invoices', invoiceName);
257
258       const pdfDoc = new PDFDocument();
259       res.setHeader('Content-Type', 'application/pdf');
260       res.setHeader(
261         'Content-Disposition',
262         'inline; filename="' + invoiceName + '"'
263       );
264       pdfDoc.pipe(fs.createWriteStream(invoicePath));
265       pdfDoc.pipe(res);
266
267       pdfDoc.fontSize(26).text('Invoice', {
268         underline: true
269       });
270       pdfDoc.text('-----');
271       let totalPrice = 0;
272       order.products.forEach(prod => {
273         totalPrice += prod.quantity * prod.product.price;
274         pdfDoc
275           .fontSize(14)
276           .text(
277             prod.product.title +
278             ' - ' +
279             prod.quantity +
280             ' x ' +
281             '$' +
282             prod.product.price
283           );
284       });
285       pdfDoc.text('---');
286       pdfDoc.fontSize(20).text('Total Price: $' + totalPrice);
287
288       pdfDoc.end();
289       // fs.readFile(invoicePath, (err, data) => {
290       //   if (err) {
291       //     return next(err);

```

```

292 // }
293 // res.setHeader('Content-Type', 'application/pdf');
294 // res.setHeader(
295 //   'Content-Disposition',
296 //   'inline; filename="' + invoiceName + '"'
297 // );
298 // res.send(data);
299 // });
300 // const file = fs.createReadStream(invoicePath);
301
302 // file.pipe(res);
303 })
304 .catch(err => next(err));
305 };
306

```

```

1 //app.js
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const session = require('express-session');
9 const MongoDBStore = require('connect-mongodb-session')(session);
10 const csrf = require('csurf');
11 const flash = require('connect-flash');
12 const multer = require('multer')
13
14 const errorController = require('./controllers/error');
15 const shopController = require('./controllers/shop');
16 const isAuth = require('./middleware/is-auth');
17 const User = require('./models/user');
18
19 const MONGODB_URI =
20 'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/shop'
21
22 const app = express();
23 const store = new MongoDBStore({
24   uri: MONGODB_URI,
25   collection: 'sessions'
26 });
27 const csrfProtection = csrf();
28 const fileStorage = multer.diskStorage({
29   destination: (req, file, cb) => {
30     cb(null, 'images')
31   },
32   filename: (req, file, cb) => {
33     cb(null, new Date().toISOString() + '-' + file.originalname)
34   }
35 })
36
37 const fileFilter = (req, file, cb) => {
38   if (
39     file.mimetype === 'image/png' ||
40     file.mimetype === 'image/jpg' ||
41     file.mimetype === 'image/jpeg'

```



```

42     ) {
43         cb(null, true)
44     } else {
45         cb(null, false)
46     }
47 }
48
49 app.set('view engine', 'ejs');
50 app.set('views', 'views');
51
52 const adminRoutes = require('./routes/admin');
53 const shopRoutes = require('./routes/shop');
54 const authRoutes = require('./routes/auth');
55
56 app.use(bodyParser.urlencoded({ extended: false }));
57 app.use(multer({ storage: fileStorage, fileFilter }).single('image'))
58 app.use(express.static(path.join(__dirname, 'public')));
59 app.use('/images', express.static(path.join(__dirname, 'images')));
60 app.use(
61     session({
62         secret: 'my secret',
63         resave: false,
64         saveUninitialized: false,
65         store: store
66     })
67 );
68 app.use(flash());
69
70 app.use((req, res, next) => {
71     res.locals.isAuthenticated = req.session.isLoggedIn;
72     next();
73 });
74
75 app.use((req, res, next) => {
76     //throw new Error('Sync Dummy')
77     if (!req.session.user) {
78         return next();
79     }
80     User.findById(req.session.user._id)
81         .then(user => {
82             if (!user) {
83                 return next();
84             }
85             req.user = user;
86             next();
87         })
88         .catch(err => {
89             next(new Error(err))
90         });
91 });
92
93 /** from app.js file,
94     for the request sent by the stripe modal,
95     by that modal we entered our credit card data in,
96     this request doesn't include our CSRF token.
97     so '/checkout' form doesn't include it.

```

```
98
99  you could include your hidden input field
100 but that wouldn't do anything
101 because stripe creates its own form in the end in an ifram which it loads
102 but that also means that stripe takes care about securing that form
103 so this is not this form
104 which you embed into your document there,
105 stripe renders and loads and secures its own form
106 so you don't need to pass the CSRF token.
107
108  so this just means we have to disable CSRF token protection for this POST create-order
route
109  to do that, i will cut the route from my ./routes/shop.js
110  and i will move it directly into app.js
111
112  i will replace 'route.post' with 'app.post'
113  because we only have the app available in this file
114  then we also have a POST method to only trigger this middleware upon POST request
115
116  */
117
118 app.post('/create-order', isAuth, shopController.postOrder);
119
120 /**the idea is that
121  * i initialize this middleware
122  * after this route which should not be checked
123  * so that only the routes after this will be affected
124  * because remember the requests travels through the middleware from top to bottom
125  * so CSRF protection only gets added after this routes
126  *   'app.post('/create-order', isAuth, shopController.postOrder);'
127  *
128  * so if you have exception of which you typically don't have too many,
129  * place it before you enable this middleware
130  */
131 app.use(csrfProtection);
132 app.use((req, res, next) => {
133   res.locals.csrfToken = req.csrfToken();
134   next();
135 });
136
137 app.use('/admin', adminRoutes);
138 app.use(shopRoutes);
139 app.use(authRoutes);
140
141 app.get('/500', errorController.get500);
142
143 app.use(errorController.get404);
144
145 app.use((error, req, res, next) => {
146   /**for the request sent by the stripe modal,
147    * by that modal we entered our credit card data in,
148    * this request doesn't include our CSRF token.
149    * so '/checkout' form doesn't include it.
150    */
151   //res.redirect('/500')
152   res.status(500).render('500', {
```

```
153     pageTitle: 'Error!',
154     path: '/500',
155     isAuthenticated: req.session.isLoggedIn
156   });
157 })
158
159 mongoose
160   .connect(MONGODB_URI)
161   .then(result => {
162     app.listen(3000);
163   })
164   .catch(err => {
165     console.log(err);
166   });
167
```

```
1 // ./routes/shop.js
2
3 const path = require('path');
4
5 const express = require('express');
6
7 const shopController = require('../controllers/shop');
8 const isAuth = require('../middleware/is-auth');
9
10 const router = express.Router();
11
12 router.get('/', shopController.getIndex);
13
14 router.get('/products', shopController.getProducts);
15
16 router.get('/products/:productId', shopController.getProduct);
17
18 router.get('/cart', isAuth, shopController.getCart);
19
20 router.post('/cart', isAuth, shopController.postCart);
21
22 router.post('/cart-delete-item', isAuth, shopController.postCartDeleteProduct);
23
24 router.get('/checkout', isAuth, shopController.getCheckout);
25
26 router.get('/orders', isAuth, shopController.getOrders);
27
28 router.get('/orders/:orderId', isAuth, shopController.getInvoice);
29
30 module.exports = router;
```