

## 24. Working With REST APIs - The Basics

### \* Chapter 354: Module Introduction





#### What's In This Module?

What are "REST APIs"?

Why use/ build REST APIs?

Core REST Concepts & Principles

First REST API!

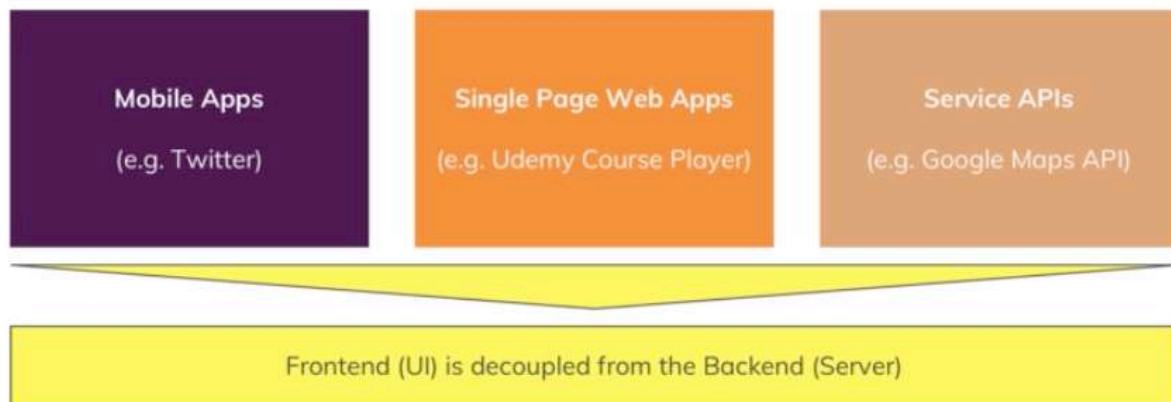
Academind

### \* Chapter 355: What Are REST APIs And Why Do We Use Them?



## What & Why?

Not every Frontend (UI) requires HTML Pages!



*By the way*

- for example, the Twitter app, these apps don't work with server-side rendered HTML code. they don't need a templating language on the server to render the HTML code because you build these apps with Java for Android or with Swift Objective C for iOS and you use rich suite of pre-built UI widgets, you use libraries provided by Apple, by Google to build your user interfaces in the respective IDEs of these programming language language like Android Studio for Android development.
  - not just on Udemy but on many modern web application is that you only fetch one initial HTML page that does not really contain a lot of real HTML content but that does load all these javascript script files and then these javascript script reach out to some backend API, to a RESTFUL API and only fetch the data they need to work with to then re-render the user interface.
  - so such web application are very popular because they give us a mobile app like feeling. we click around and we don't have to wait for a page refresh. we always stay on the same page and only the data that gets rendered changes and therefore only the data is exchanged behind the scenes, all the user interface rendering is done through browser side javascript.
  - maybe you are working on a classic node application as we did thus far but you also have certain service API like Google Maps API. it's not the frontend that requires us to build a REST API on our own but this is another example for a case where you only need the data and no user interface. you don't expect the Google Maps API to give you back the HTML code. you might be interested in some coordinates, something like this.
  - we have a frontend or we have code that is decoupled from the backend or from a certain backend logic like Google Maps and we only need to exchange the data because we don't wanna get any user interface, we don't wanna get HTML Code. we build that on our own. we just have a backend that needs to serve us data and that is a core idea of building REST API.
- 

## A Different Kind of Response is Needed

### Representational State Transfer

Transfer Data instead of User Interfaces

Important: Only the response (and the request data) changes, NOT the general server-side logic!

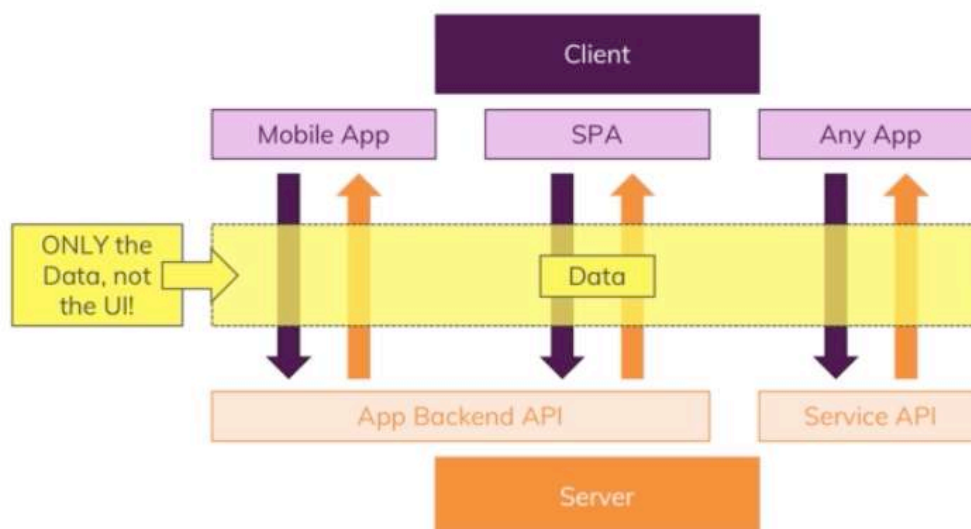
By: [unintelligible]

- we just transfer data and we leave it to the client or to the front-end. be that a mobile app, be that a single page application.
- REST APIs and Traditional web apps where you render the views on the server are seen as 2 totally different things. but they are not, they only differ in the response and in the kind of data you expect but they don't differ in what happens on the server besides that the fact that you don't render the view there and that is really important
- we will reuse 99% of the knowledge, only tune our data usage or our data handling and the response a little bit.

## \* Chapter 356: Accessing Data With REST APIs



### REST API Big Picture



By: [unintelligible]

- one advantage by the way is that we can use one and the same API for multiple clients. because mobile apps

and single page web apps use the same data.





JSON stands for JavaScript Object Notation. Formats			
HTML	Plain Text	XML	JSON
<code>&lt;p&gt;Node.js&lt;/p&gt;</code>	<code>Node.js</code>	<code>&lt;name&gt;Node.js&lt;/name&gt;</code>	<code>{"title": "Node.js"}</code>
Data + Structure	Data	Data	
Contains User Interface	No UI Assumptions	No UI Assumptions	
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	



## Data Formats

HTML	Plain Text	XML	JSON
<code>&lt;p&gt;Node.js&lt;/p&gt;</code>	<code>Node.js</code>	<code>&lt;name&gt;Node.js&lt;/name&gt;</code>	<code>{"title": "Node.js"}</code>
Data + Structure	Data	Data	Data
Contains User Interface	No UI Assumptions	No UI Assumptions	No UI Assumptions
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	Machine-readable and concise; Can easily be converted to JavaScript

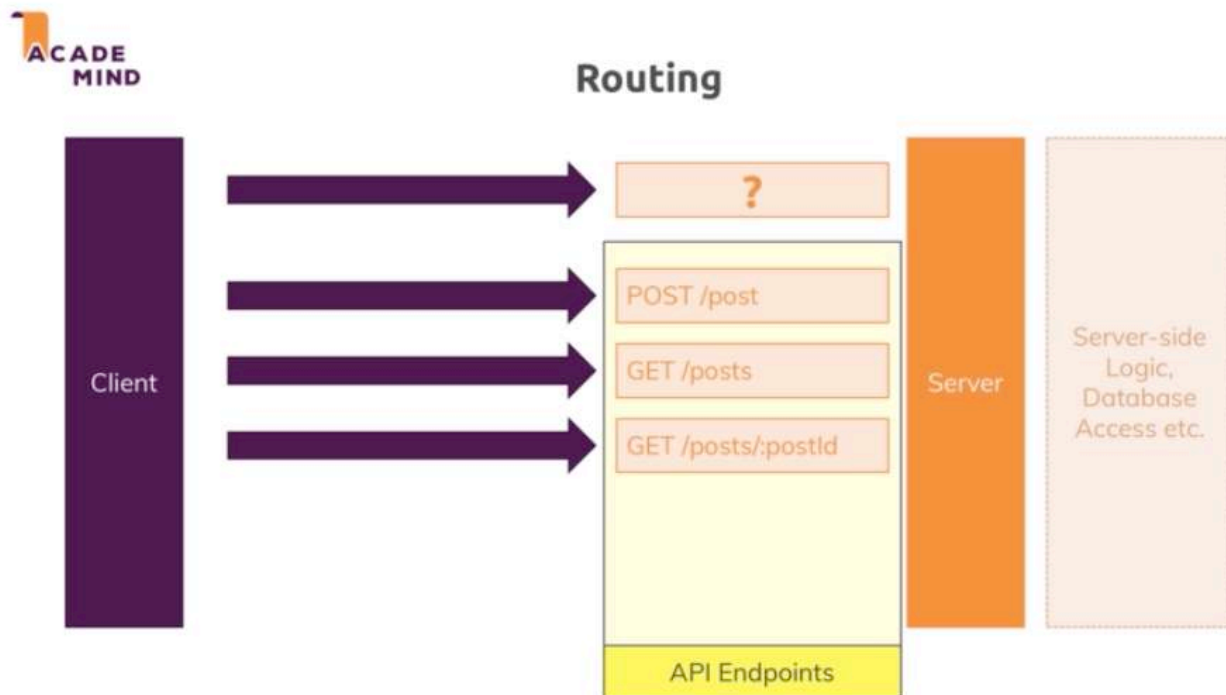
- when we rendered an EJS view, what we did is we sent HTML code to the browser because the view was rendered on the server and the result of that rendering process was HTML page, was HTML code.

- it's unnecessarily difficult to parse in plain text because the text is easy for human but for the computer, it isn't. there is no clear pattern in the text and therefore this is not really a great way of exchanging data.

- JSON is our winner data format.

## \* Chapter 357: Understanding Routing & HTTP Methods





- the core thing is we in the end still send normal requests, these are totally normal requests that don't expect any HTML response and we send a combination of HTTP method and path and this is how we communicate with our server.

- in the rest world or in the API world, we like to call these things 'API endpoints'. so when i'm talking about the combination of a HTTP method like POST and GET and the respective path. these are the endpoints we defined on our REST API and we defined a logic that should execute on the server when a request reaches such an endpoint.





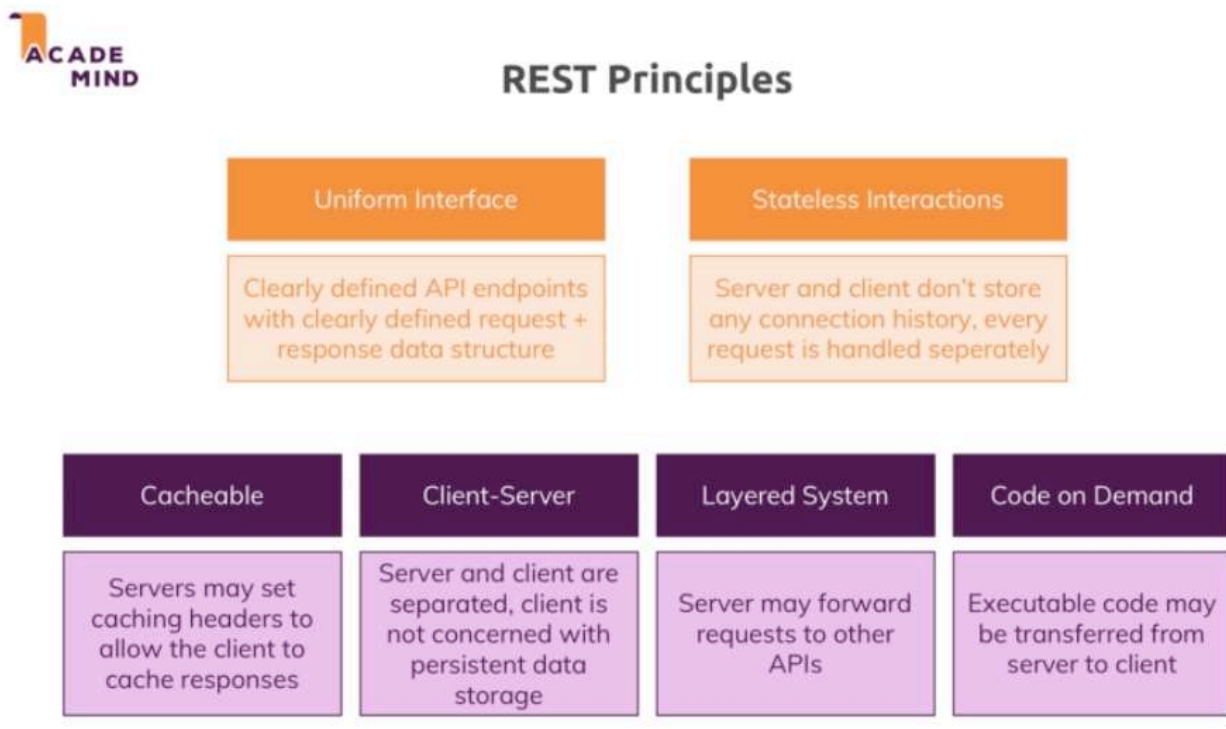
- if we wanna put a resource onto the server, which means we wanna create it or overwrite an existing resource. POST will never overwrite or should never overwrite.

- PATCH also don't overwrite it entirely necessarily but update parts of it.

- for the rest world, we should use a POST request to create or append a resource. no one is stopping you from deleting something on a server because you only define a method path pair on your server side and then you run any code you want. and what happens in that code is not restricted by the method that was used to execute that code. you can restrict it yourself and you wanna implement the REST API that follows these ideas but you don't have to.

## \* Chapter 358: REST APIs - The Core Principles





- when building a REST API, the server and the client are totally separated, they don't share a common history. so no connection history is stored and no sessions will be used because every incoming request is treated as if no prior requests were sent. the server has a look at every request on its own. it doesn't store a session for the client. it doesn't care about the client at all.

- for example, like the Google Maps API and you don't care about the individual client. you just say here are the endpoints i have, here's the data you get back for each endpoint. here's the data i expect from you for my endpoints and then i don't care about you. i don't store a session with you. we have a strong decoupling of the client and the server even if they were to run on the same server because we are building our own API for our own frontend. we still would decouple both so that they work independently and just exchange data. this means that every time we set up a new endpoint, we have to make sure that it works independently from prior requests. and a typical problem is authentication where once we logged in, future requests should be treated as logged in.

- 'Cacheable' means on your REST API, you could send back some headers that tell the client how long the response is valid so that the client may cache the response.

- 'Layered System' means as a client when we send a request to an API, we can't rely on that server we sent it to you immediately handling the request, the server might instead forward the request or distribute it to another server. ultimately we only care about the data we get back which should follow the structure that was defined by the API.

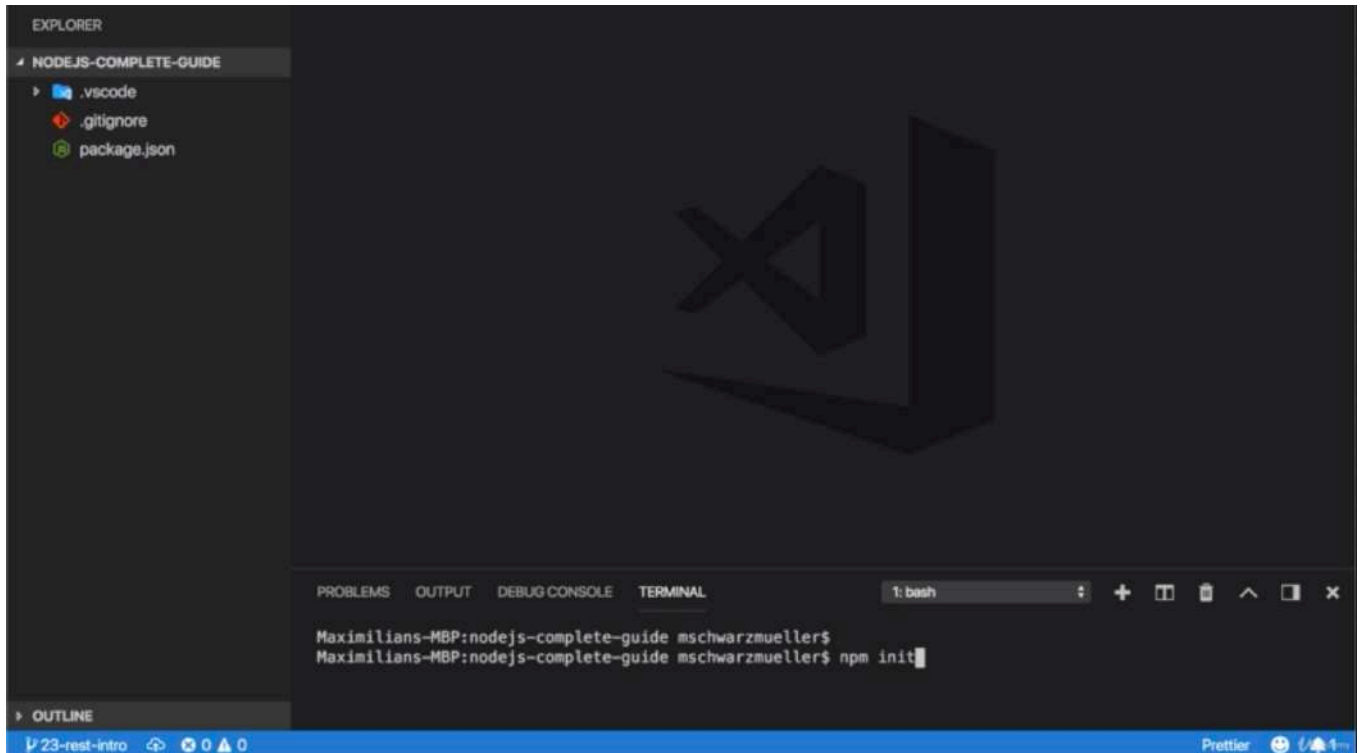
- 'Code on demand' is last optional principle that means API could also for some endpoints transfer executable code to the client. in reality, you don't see that too often. we are mostly talking about normal data. we are using.



# \* Chapter 359: Creating Our REST API Project & Implementing The Route Setup

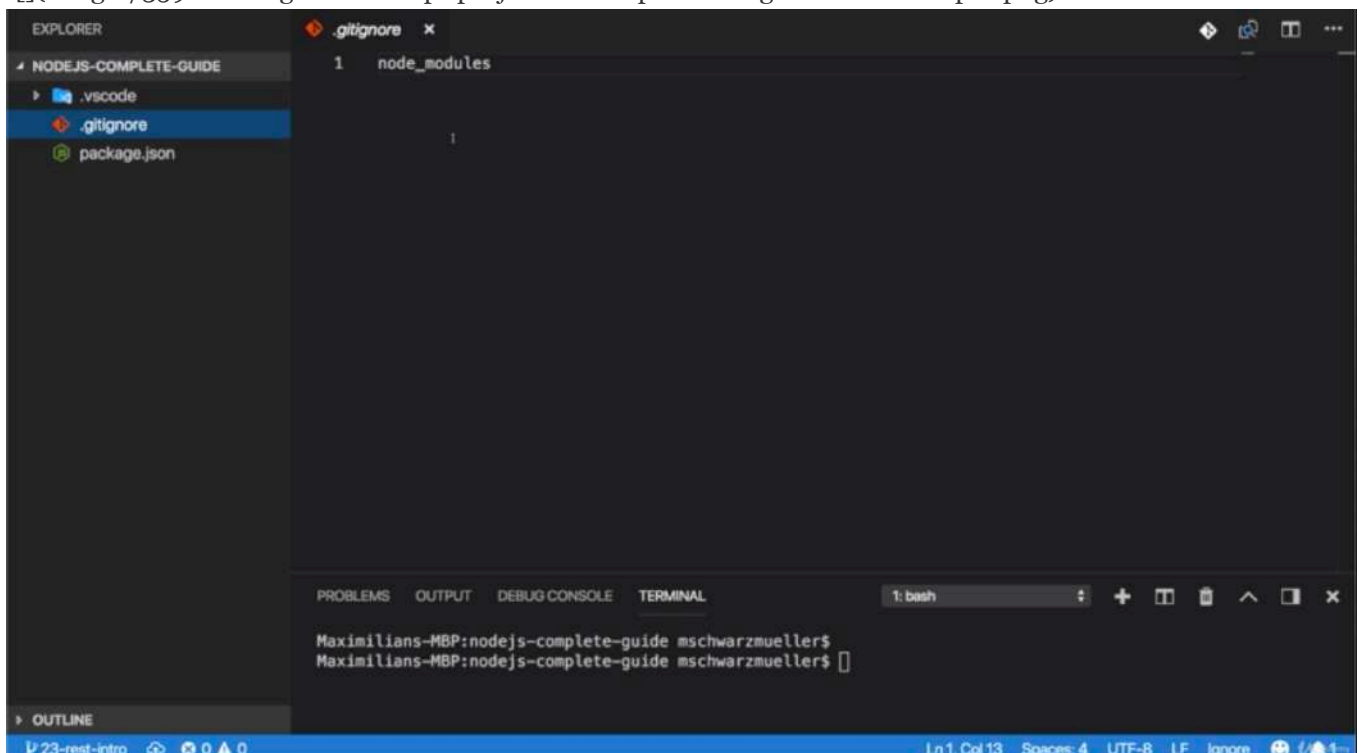
- 1. update
- .gitignore
- package.json
- app.js
- ./routes/feed.js
- ./controllers/feed.js





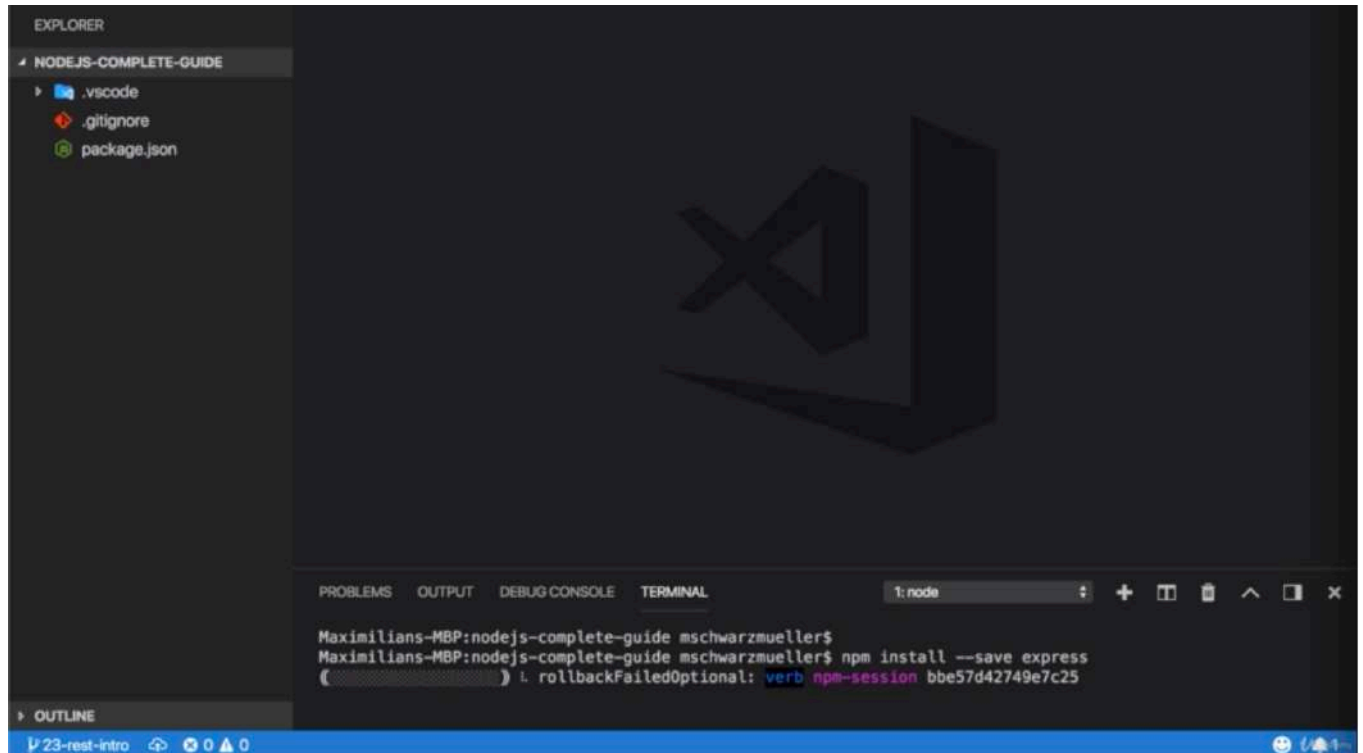
- Let's build our first simple REST API and for that, i'm in a brand new folder.
- i only did one thing in there, i ran 'npm init' and confirmed all the default settings.





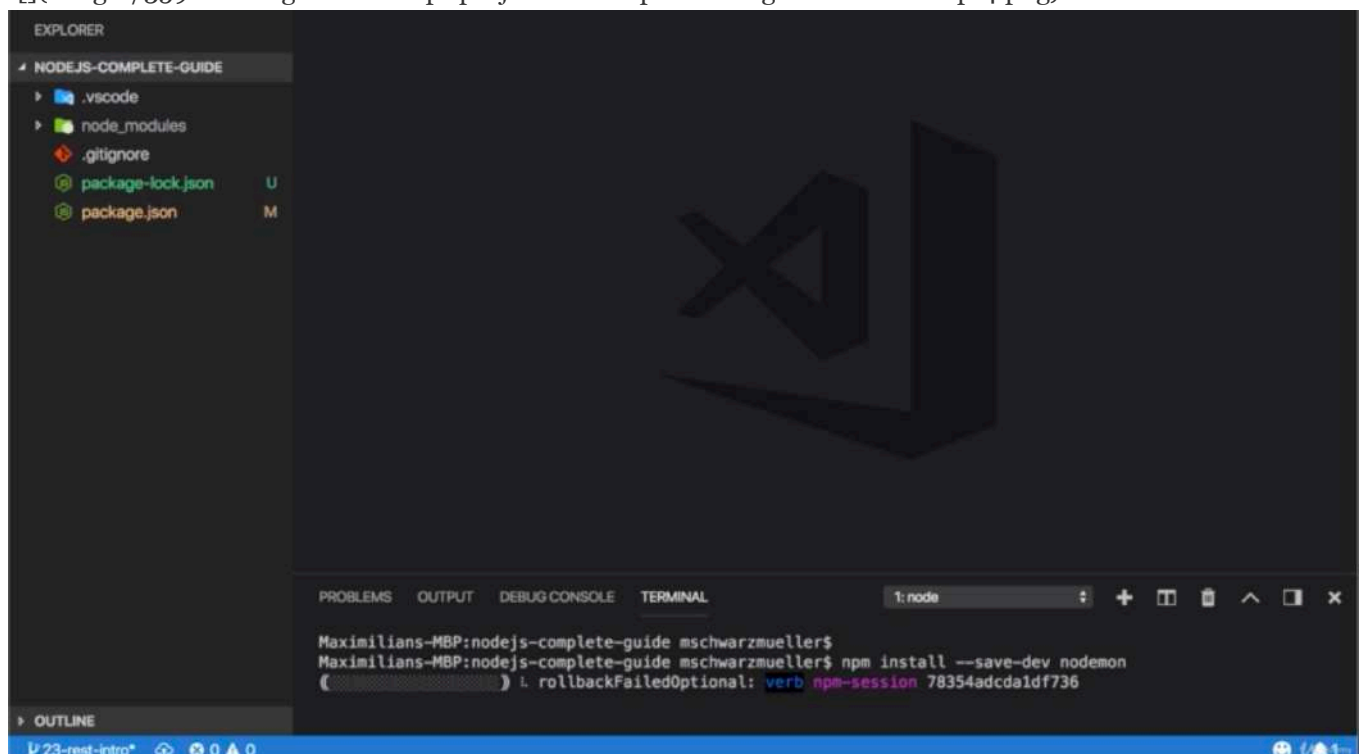
- since i'm using version control, i also added a .gitignore file so that i can ignore an upcoming node\_modules folder and that is it.





- we will use express which is a great all-rounder and therefore i will install as a production dependency with '—save express', we need express to build an API conveniently.

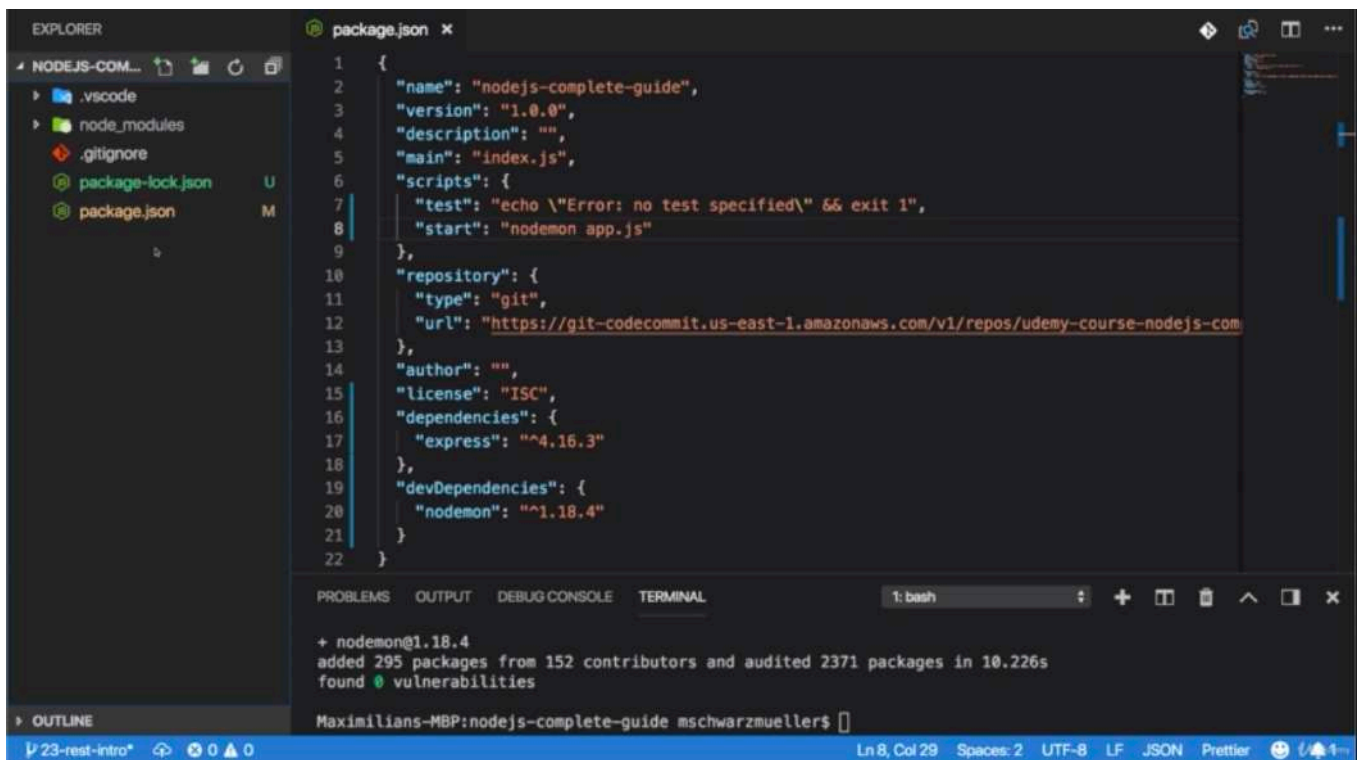




- and i'm gonna install nodemon with development dependency because i still don't wanna restart my server manually after every change.







The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'NODEJS-COM...' with files like '.vscode', 'node\_modules', '.gitignore', 'package-lock.json', and 'package.json'. The 'package.json' file is open in the editor, showing a JSON configuration for a project named 'nodejs-complete-guide'. The terminal shows the output of an npm install command, indicating that 295 packages were added and 2371 packages were audited, with no vulnerabilities found.

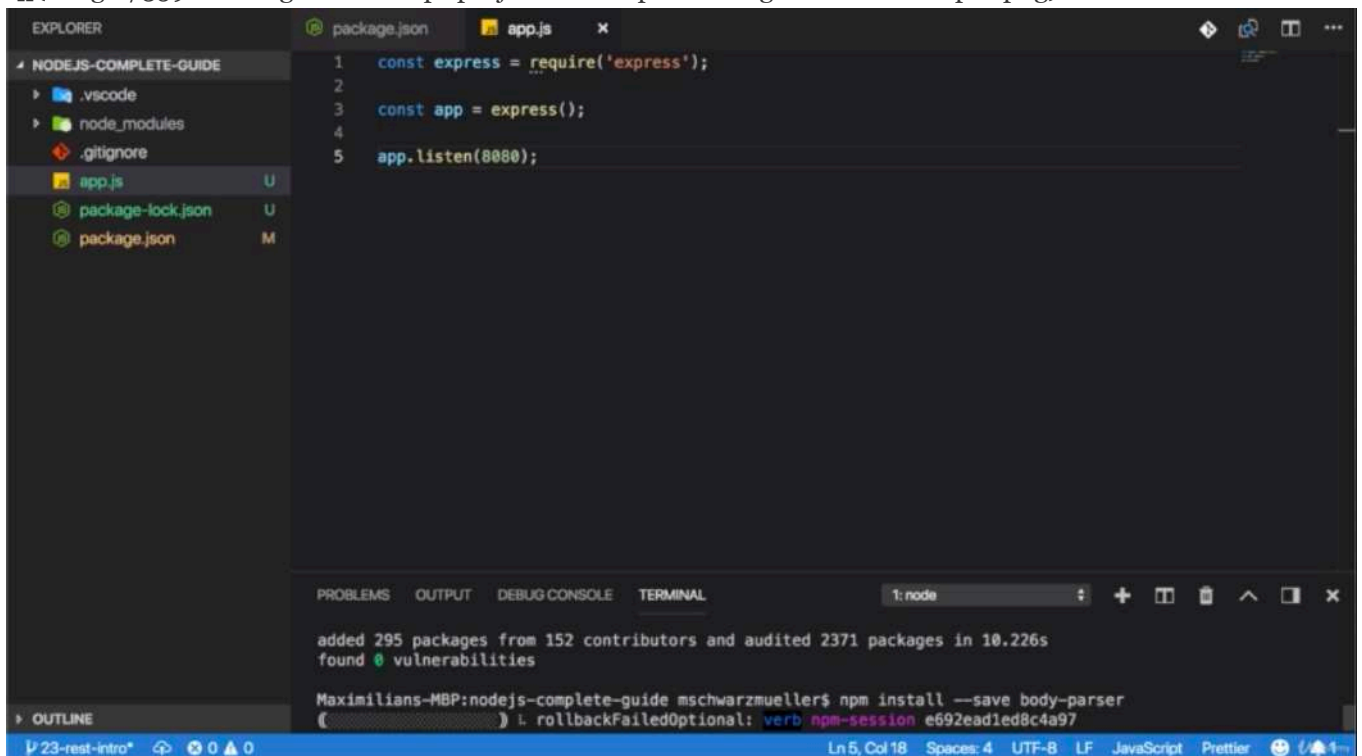
```
1 {
2   "name": "nodejs-complete-guide",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1",
8     "start": "nodemon app.js"
9   },
10  "repository": {
11    "type": "git",
12    "url": "https://git-codecommit.us-east-1.amazonaws.com/v1/repos/udemy-course-nodejs-com"
13  },
14  "author": "",
15  "license": "ISC",
16  "dependencies": {
17    "express": "^4.16.3"
18  },
19  "devDependencies": {
20    "nodemon": "^1.18.4"
21  }
22 }
```

terminal: 1: bash

+ nodemon@1.18.4  
added 295 packages from 152 contributors and audited 2371 packages in 10.226s  
found 0 vulnerabilities

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$





The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'NODEJS-COMplete-GUIDE' with files like '.vscode', 'node\_modules', '.gitignore', 'app.js', 'package-lock.json', and 'package.json'. The 'app.js' file is open in the editor, showing a simple Express.js application. The terminal shows the output of an npm install command, indicating that 295 packages were added and 2371 packages were audited, with no vulnerabilities found.

```
1 const express = require('express');
2
3 const app = express();
4
5 app.listen(8080);
```

terminal: 1: node

added 295 packages from 152 contributors and audited 2371 packages in 10.226s  
found 0 vulnerabilities

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$ npm install --save body-parser

- now to add some routes and to do something with them, i will also install the 'body-parser' as a production dependency. so that i can parse incoming request bodies.

- the views folder will not be recreated because we will not render my views anymore. we will just exchange datas.

```
1 // .gitignore
2
3 node_modules
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
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
```

```
5  "version": "1.0.0",
6  "description": "",
7  "main": "index.js",
8  "scripts": {
9    "test": "echo \\\"Error: no test specified\\\" && exit 1",
10   "start": "nodemon app.js"
11 },
12 "author": "",
13 "license": "ISC",
14 "dependencies": {
15   "express": "^4.16.4"
16 },
17 "devDependencies": {
18   "nodemon": "^1.19.0"
19 }
20 }
```

```
1 //app.js
2
3 const express = require('express')
4
5 const feedRoutes = require('./routes/feed')
6
7 const app = express();
8
9 /**we forward any incoming request to feed route
10 * or we only forward requests
11 * that start with '/feed'
12 * so into ./routes/feed.js file
13 * and we handle one request. '/post'
14 * so in total, /feed/posts would be handled right now
15 * as long as it is a GET request
16 */
17 app.use('/feed', feedRoutes)
18
19 app.listen(8080)
```

```
1 //./routes/feed.js
2
3 const express = require('express')
4
5 const feedController = require('../controllers/feed')
6
7 const router = express.Router()
8
9 // GET /feed/posts
10 router.get('/posts', feedController.getPosts)
11
12 module.exports = router
```

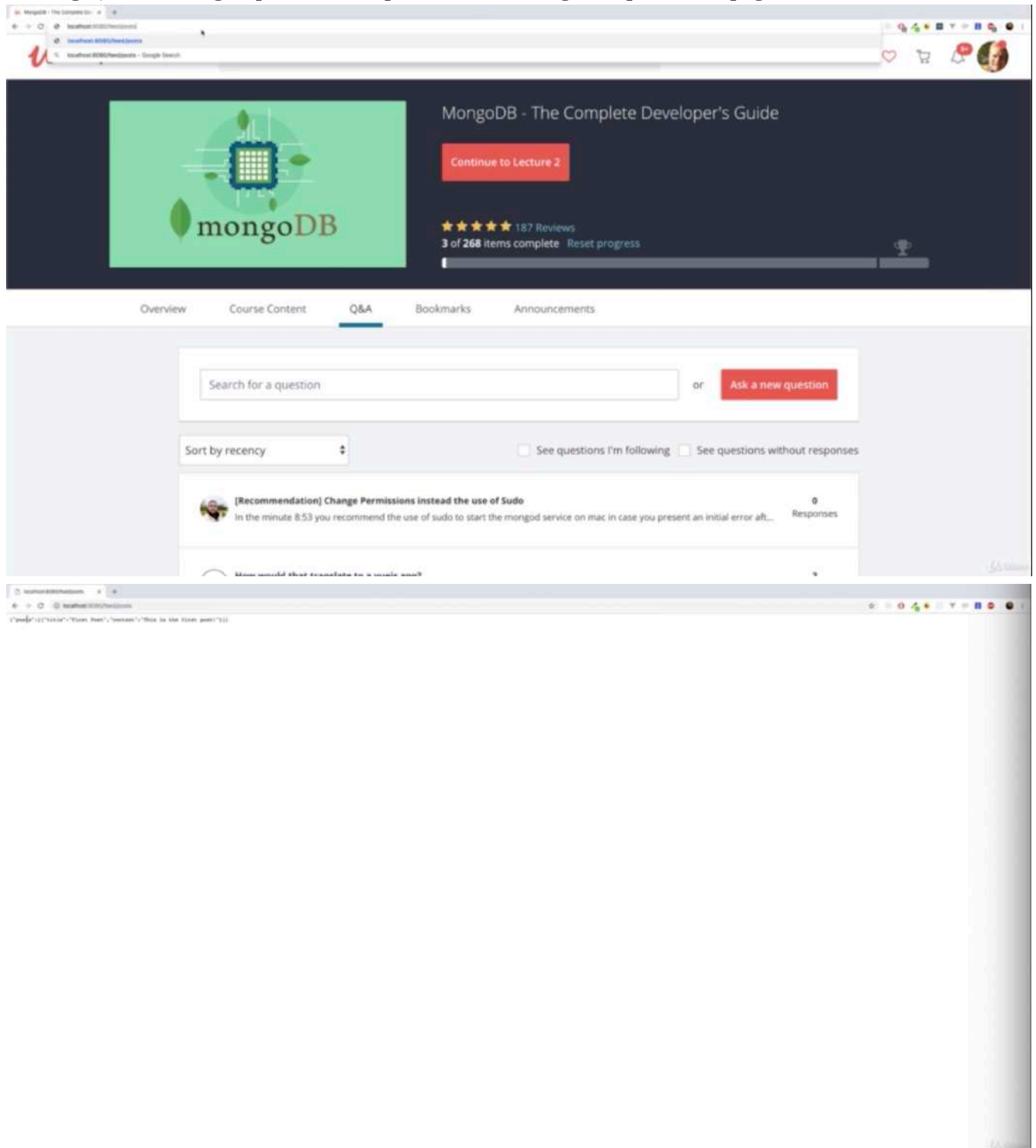
```
1 //./controllers/feed.js
2
3 exports.getPosts = (req, res, next) => {
4
5 }
```

# \* Chapter 360: Sending Requests & Responses And Working With Postman

1. update
  - ./controllers/feed.js
  - app.js
  - ./routes/feed.js

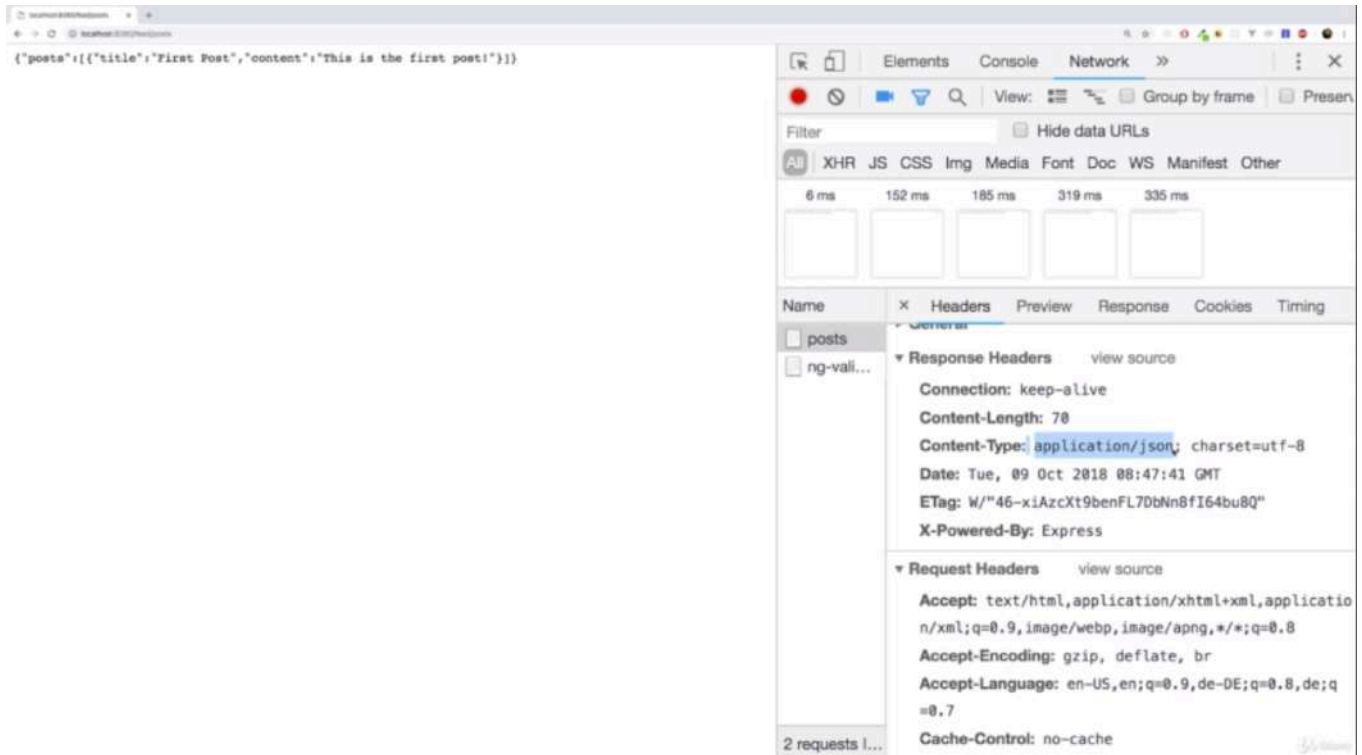






- now we can simply enter 'localhost:8080/feed/posts'. so you should get some JSON data here.

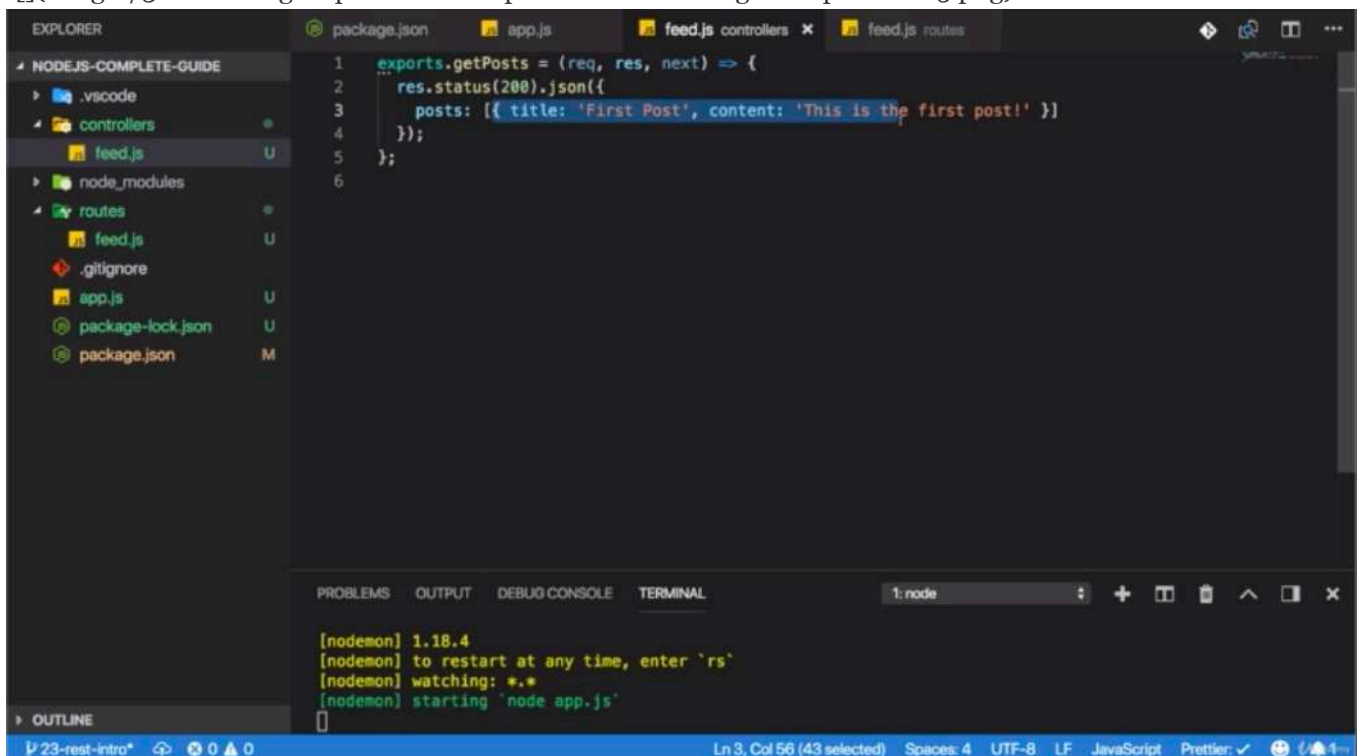




- and we can see 'application/json' was set automatically by our server because we used that json method and we get back the content we defined here.







# Http Methods (Http Verbs)

More than just GET & POST

GET	POST	PUT
Get a Resource from the Server	Post a Resource to the Server (i.e. create or append Resource)	Put a Resource onto the Server (i.e. create or overwrite a Resource)
PATCH	DELETE	OPTIONS
Update parts of an existing Resource on the Server	Delete a Resource on the Server	Determine whether follow-up Request is allowed (sent automatically)

- there might be multiple posts and therefore adding or appending sounds good to me and hence i wanna use the POST method instead of PUT. if we were to manage the user data here, then maybe PUT might be better because we create or overwrite the resource.

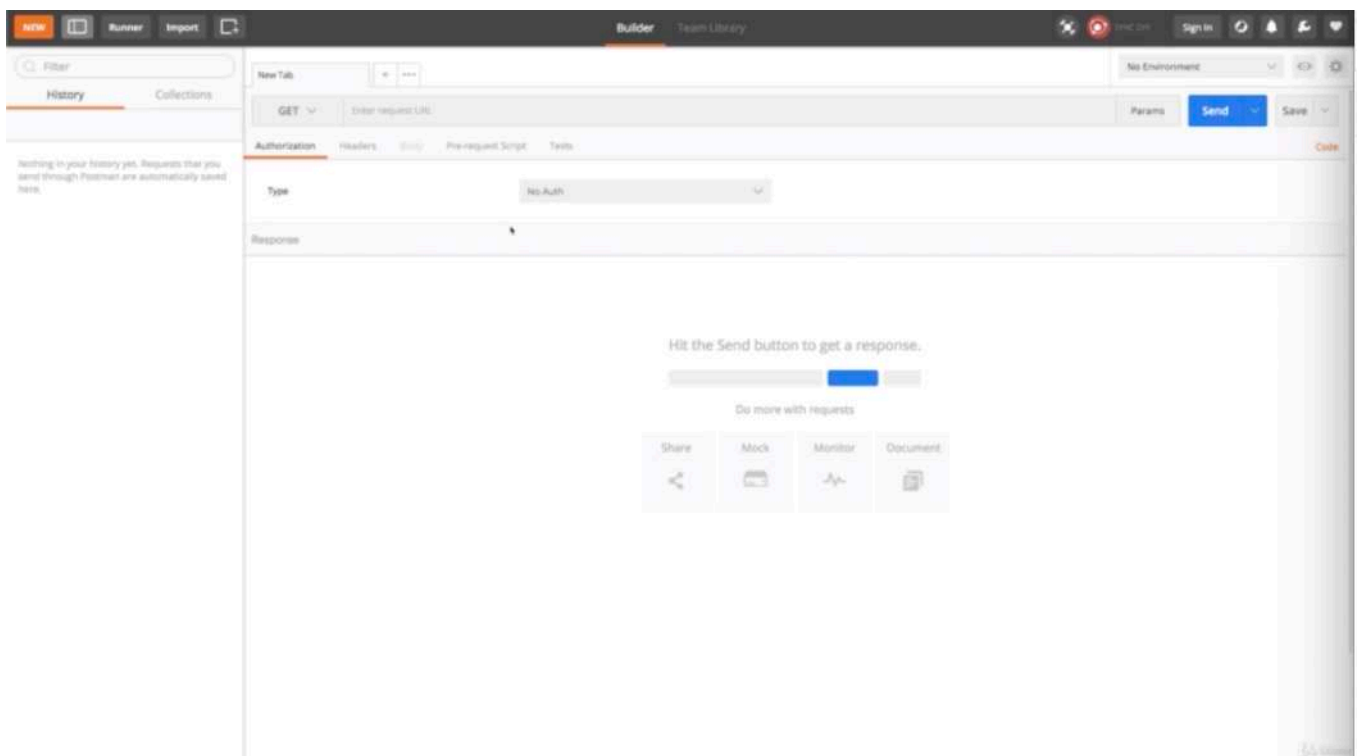
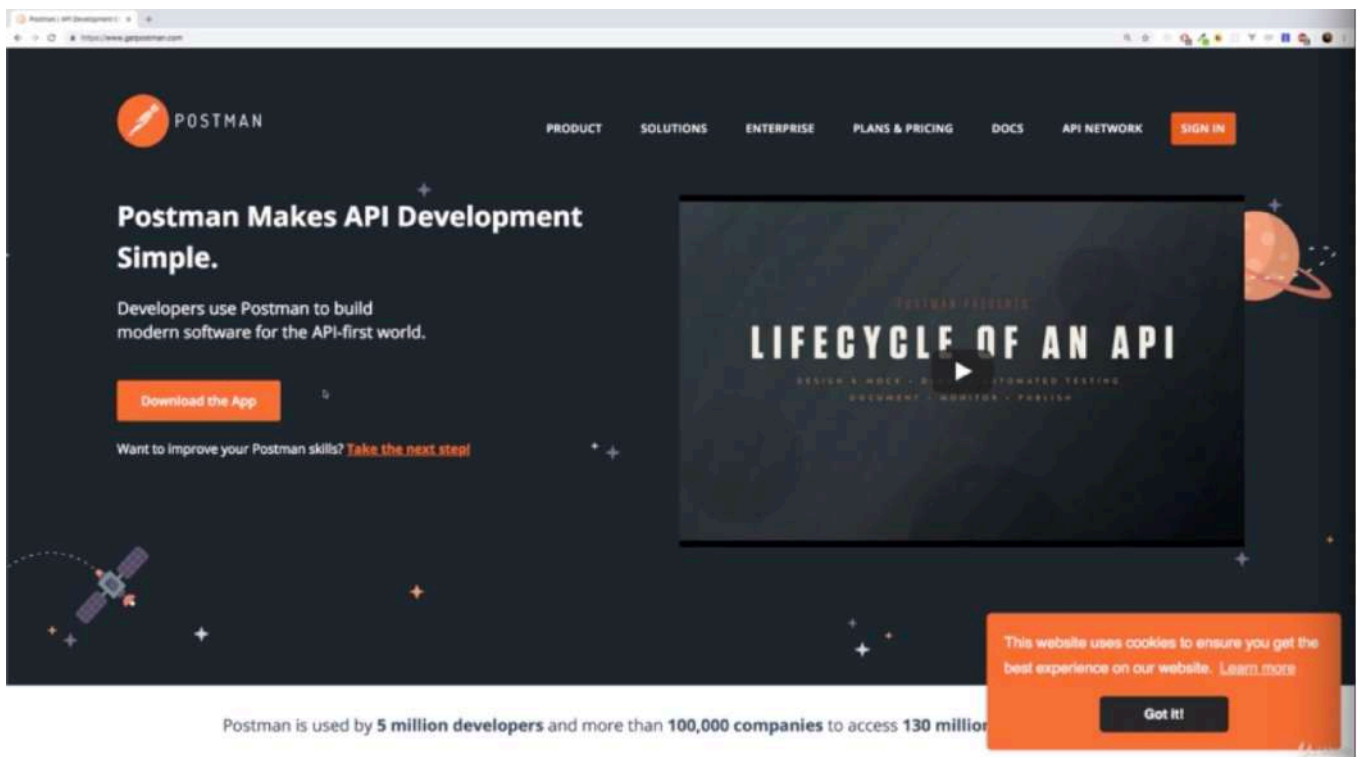






The screenshot shows a Google search for 'postman'. The search results include:

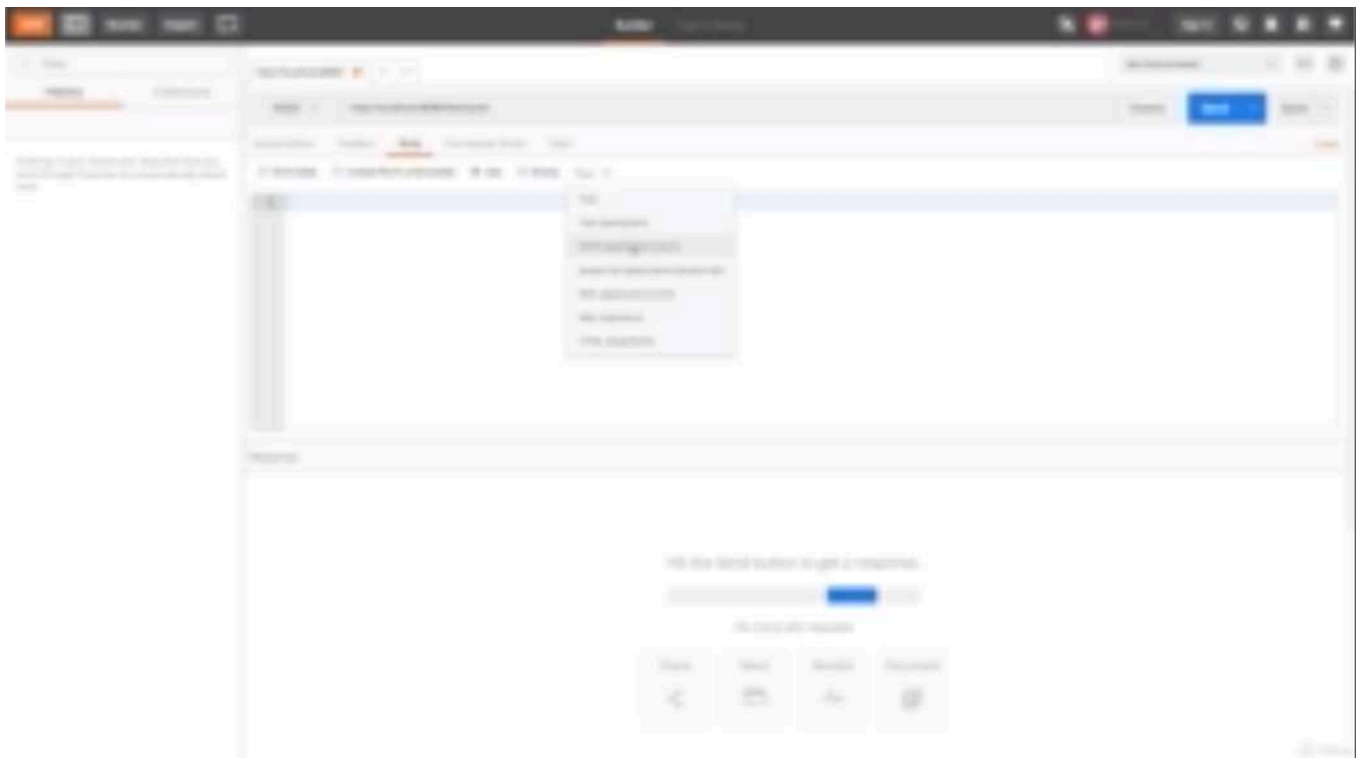
- Postman | API Development Environment**: A link to <https://www.getpostman.com/>. The description states: 'Postman is the only complete API development environment, for API developers, used by more than 5 million developers and 100000 companies worldwide.' It includes links for 'Download the App', 'Products', 'Learn More', 'Postman Learning Center', 'Take the next step!', and 'Jobs'.
- Postdot Technologies Pvt. Ltd.**: A company profile for 'getpostman.com' founded in 2014. It includes a disclaimer to 'Claim this knowledge panel' and a 'Feedback' link.
- Postman - Chrome Web Store**: A link to the Chrome Web Store page for Postman. It shows a rating of 4.7 stars from 9,129 votes, is free, and is developed by Postman. The description mentions: 'Jun 26, 2018 - Postman makes API development faster, easier, and better. The free app is used by more than 3.5 million developers and 30,000... Chrome Apps & Extensions Developer Tool.'
- Videos**: A section showing three video thumbnails: 'Intro to Testing APIs with Postman', 'POSTMAN Beginner Tutorials', and 'How to use APIs with Postman'.



- you can choose your method and then send a request.

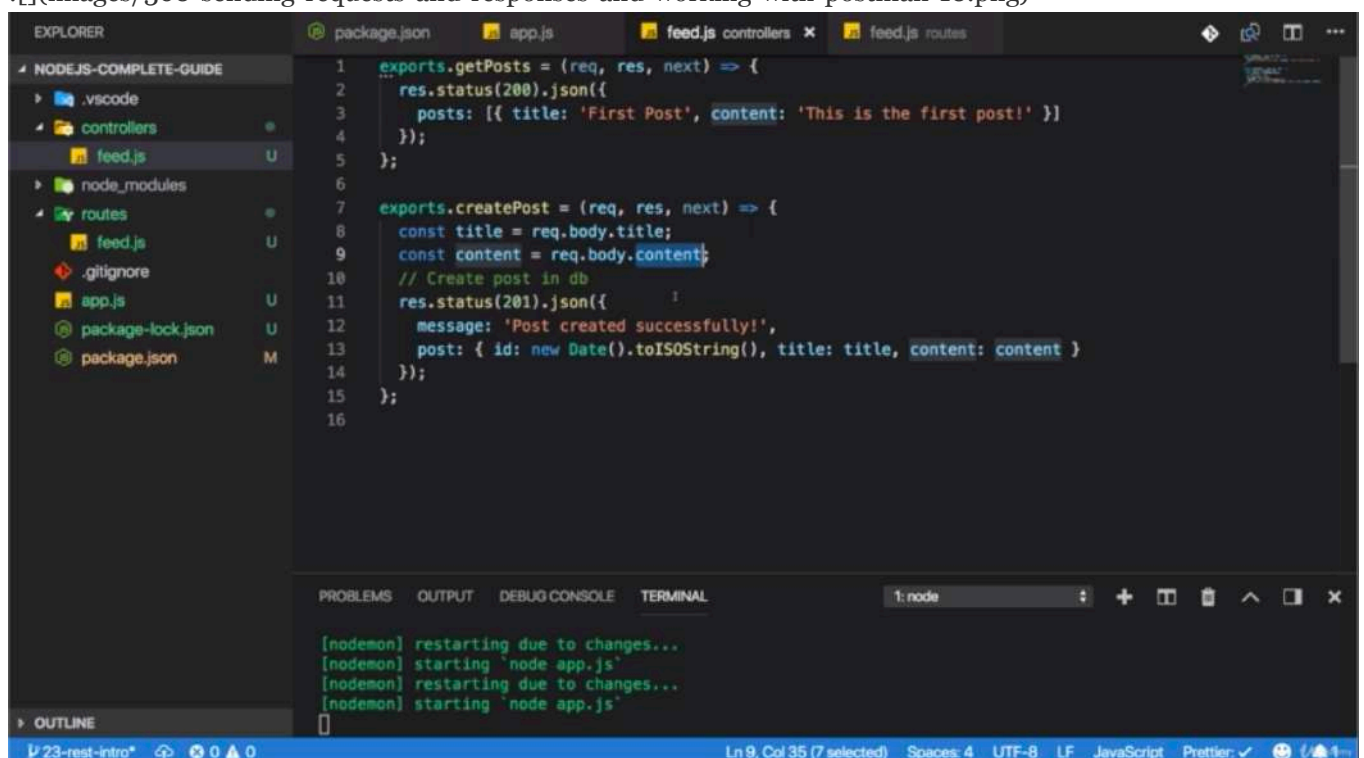






- you see that the 'body' tab now got enabled. with GET it was disabled because GET requests can't hold a body. POST requests can. here. you can choose your format and we don't need any of these. instead i will choose 'raw'. and then there in the dropdown, JSON(application/json). so now here we can write some JSON data.

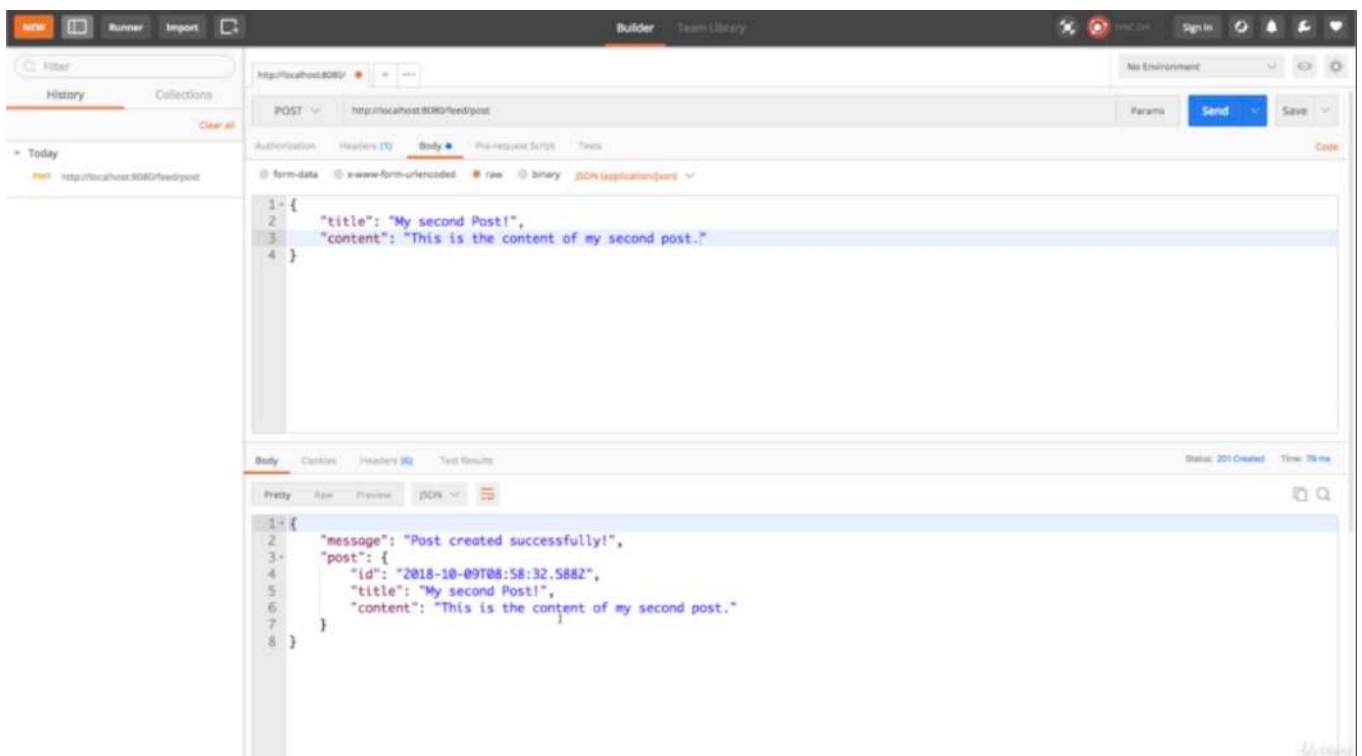
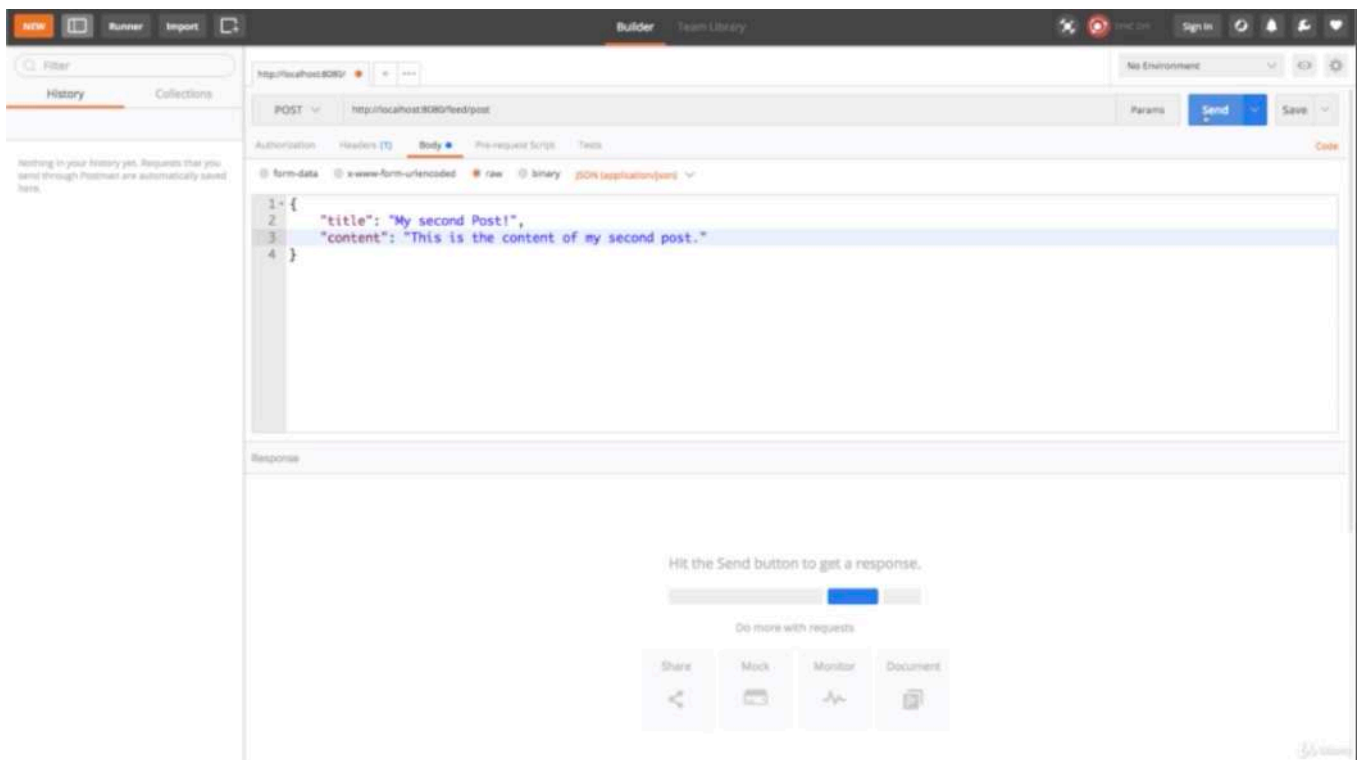




- so we wanna pass these 2 fields to our server 'req.body.title', 'req.body.content'.







- and now we can click 'Send' button and we get back a response which looks good because it's the response i defined.

```

1 //./controllers/feed.js
2
3 exports.getPosts = (req, res, next) => {
4   /**you will not call 'res.render()'
5    * because REST APIs don't render views
6    * because they don't return HTML code
7    *
8    * we can pass a normal javascript object to json()
9    * and it will be converted to the JSON format
10   * and sent back as a response to the client who sent the request.
11   *
12   * 'status(200)' is a default

```

```

13  * but we will work with different status codes throughout this module
14  * and we wanna be clear about the status code our response has.
15  *
16  * the client now has to render the user interface based on your response
17  * and therefore error codes are super important to pass back to the client
18  * so that the client can just have a look at the status code
19  * should i render my normal user interface
20  * because the requests succeeded or did i get an error
21  * and i wanna render an appropriate error interface.
22  */
23  res.status(200).json({
24    posts: [{ title: 'First Post', content: 'This is the first post!' }]
25  });
26  };
27
28  exports.createPost = (req, res, next) => {
29    const title = req.body.title;
30    const content = req.body.content;
31    //Create post in db
32    /**201 is the better status code to use
33     * if you wanna tell the client success a resource was created
34     * just 200 is just success.
35     * 201 indicate that we created a resource.
36     */
37    res.status(201).json({
38      message: 'Post created successfully!',
39      post: { id: new Date().toISOString(), title: title, content: content }
40    });
41  };

```

```

1  //app.js
2
3  const express = require('express')
4  const bodyParser = require('body-parser')
5
6  const feedRoutes = require('./routes/feed')
7
8  const app = express();
9
10 /**'app.use(bodyParser.urlencoded())'
11  * this is great for data format
12  * or for requests that hold data in the format of x-www-form-urlencoded
13  * this is the default data
14  * that data has if submitted through a form post request.
15  * but we don't need form data
16  * we have no form data
17  *
18  * instead we wanna use body parser with the json method
19  * which is able to parse json data from incoming requests
20  * this is good for 'application/json' as is the official name
21  * that you will find in the header.
22  * this is how the data will be appended to the request that reaches our server.
23  * so we need this middleware to parse incoming json data
24  * so that we are able to extract it on the body.
25  * because 'req.body.title' or 'req.body.content' will be added by body parser
26  * this body field on the incoming request.
27  *

```

```
28 * with that
29 * we can extract all that data
30 * but how can we test this?
31 * we can create a form
32 * in which we submit
33 * because we would be back to x-www-form-urlencoded data
34 * it would not be a realistic test
35 * because you don't use forms like this
36 * when working with REST APIs
37 *
38 * but you can instead use 'postman'
39 */
40 app.use(bodyParser.json())
41
42 app.use('/feed', feedRoutes)
43
44 app.listen(8080)
```

```
1 //./routes/feed.js
2
3 const express = require('express');
4
5 const feedController = require('../controllers/feed');
6
7 const router = express.Router();
8
9 // GET /feed/posts
10 router.get('/posts', feedController.getPosts);
11
12 // POST /feed/post
13 router.post('/post', feedController.createPost);
14
15 module.exports = router;
```

## \* Chapter 361: REST APIs, Clients & CORS Errors

1. update

- app.js







Untitled  
A PEN BY CAPTAIN ANONYMOUS

Save Settings Change View

HTML

```
<button id="get">Get Posts</button>
<button id="post">Create a Post</button>
```

CSS

JS

```
const getButton = document.getElementById('get');
const postButton = document.getElementById('post');

getButton.addEventListener('click', () => {
  fetch('http://localhost:8080/feed/posts')
    .then(res => res.json())
    .then(resData => console.log(resData))
    .catch(err => console.log(err));
});
```

Get Posts Create a Post

Elements Console Network

Failed to load http://localhost:8080/feed/posts index.html:1 is: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'https://s.codepen.io' is therefore not allowed access. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

TypeError: Failed console\_runner-ce303\_a462c826d54a73.js:1 to fetch

Cross-Origin Read Blocking (CORB) blocked cross-origin response http://localhost:8080/feed/posts with MIME type application/json. See https://www.chromestatus.com/feature/5629709824032768 for more details.

Untitled  
A PEN BY CAPTAIN ANONYMOUS

Save Settings Change View

HTML

```
<button id="get">Get Posts</button>
<button id="post">Create a Post</button>
```

CSS

JS

```
const getButton = document.getElementById('get');
const postButton = document.getElementById('post');

getButton.addEventListener('click', () => {
  fetch('http://localhost:8080/feed/posts')
    .then(res => res.json())
    .then(resData => console.log(resData))
    .catch(err => console.log(err));
});
```

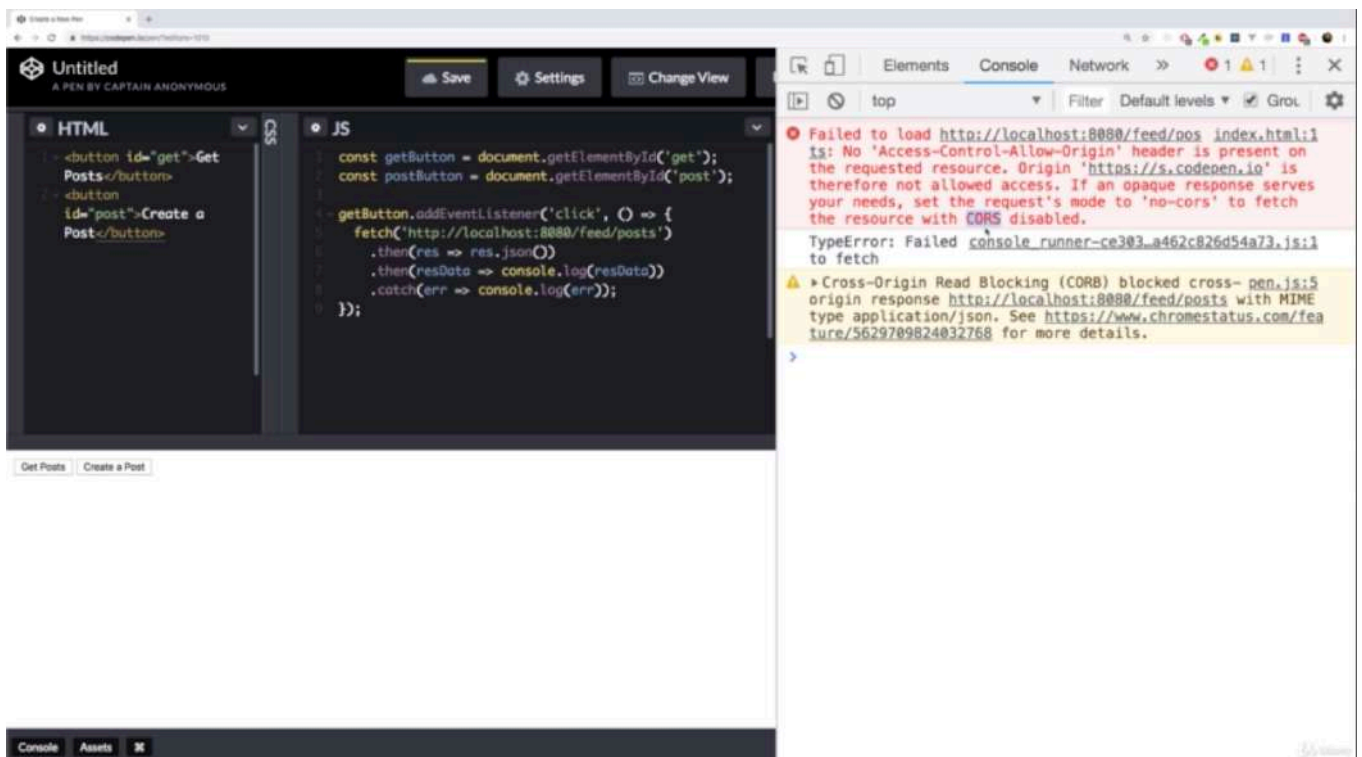
Get Posts Create a Post

Elements Console Network

Failed to load http://localhost:8080/feed/posts index.html:1 is: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'https://s.codepen.io' is therefore not allowed access. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

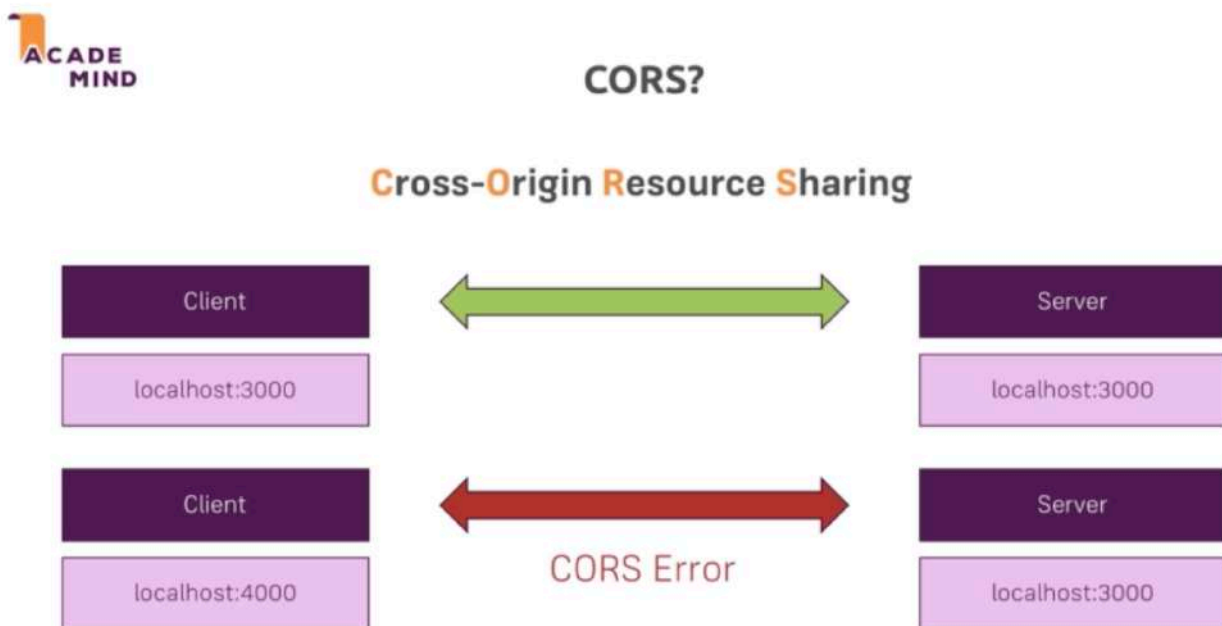
TypeError: Failed console\_runner-ce303\_a462c826d54a73.js:1 to fetch

Cross-Origin Read Blocking (CORB) blocked cross-origin response http://localhost:8080/feed/posts with MIME type application/json. See https://www.chromestatus.com/feature/5629709824032768 for more details.



- when you click 'Get Posts', and you should get a no access control allow origin headers present error. this is the error you see a lot when building modern web applications and it often leads to a lot of confusion. this error is also called a CORS error and you see the word CORS down there.





- CORS error is not allowed by browsers. before here, we render our HTML files on the server and therefore they were served by the same server as you send your requests too. so we never had issues

- however here, if client server run on different domains like the client on 'localhost:4000' which is a different domain than 3000 and in production, you would run on my [app.com](#) and let's say [myAPI.com](#). if you send requests and responses here, you get problems, you get a CORS error because it's a security mechanism provided by the browser that you can't share resources across domains, across servers, across origins. as it's called here.

- however we can overwrite this because this mechanism makes sense for some applications, for REST APIs, it typically doesn't. we wanna allow our server to share its data. we wanna offer data from our server to different clients and these clients will often not be served by the same server as our API runs on. take Google Maps, you

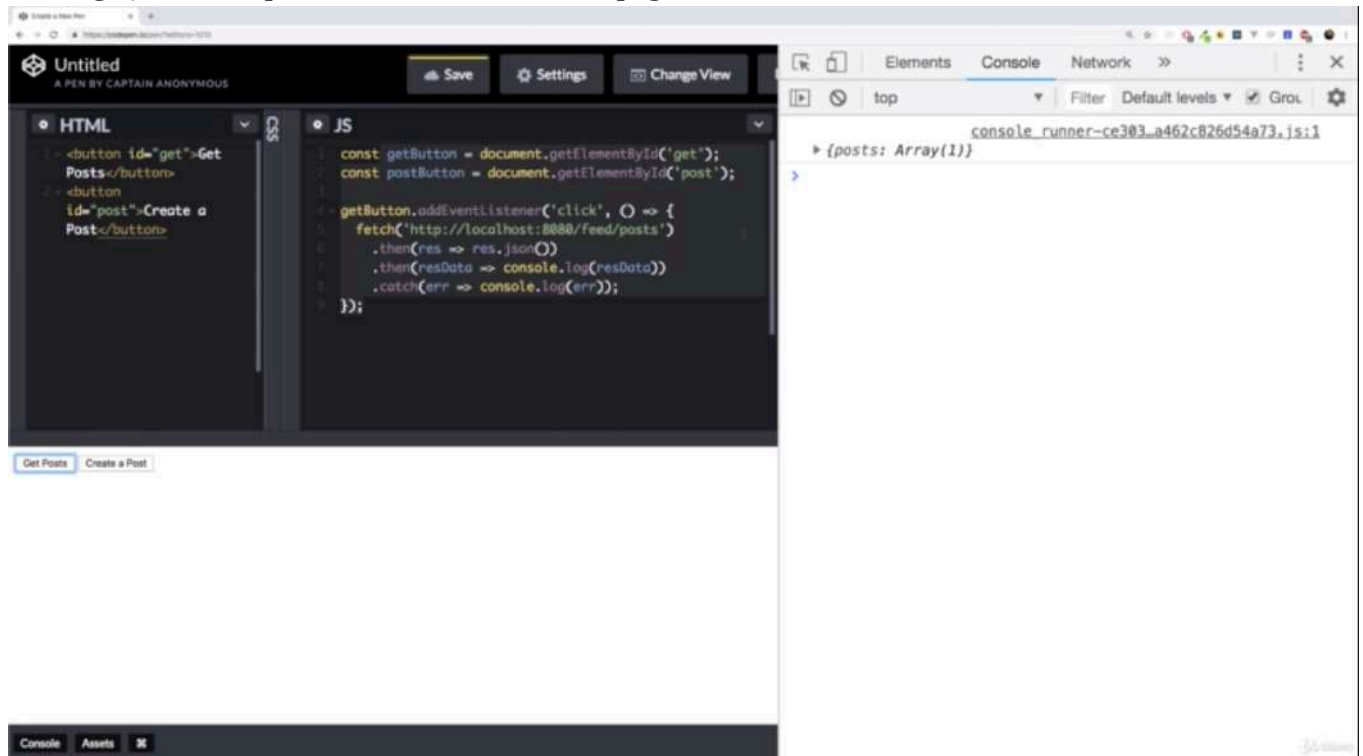


are not running your app on Google servers. but you still can access it and the same is true for your own API and even if you build both the frontend and backend, you will often serve the 2 ends from different servers because you can choose a server. because you can choose server for the frontend that is optimized for frontend code that really serves that well and you serve your server side code, your node code from a different server, so you will have different domains, different address there too.

- therefore we need to solve such a CORS error or tell the browser that it may accept the response sent by our server. and to tell the browser, we have to change something on the server

- you can't solve CORS error in frontend javascript code. you can solve in serverside code. we wanna set these headers on any responses that leaves our server. so the app.js file and general middleware is a great place.





- now it just works as you can see because now we set the appropriate CORS headers on the server.

```
1 //app.js
2
3 const express = require('express')
4 const bodyParser = require('body-parser')
5
6 const feedRoutes = require('./routes/feed')
7
8 const app = express();
9
10 app.use(bodyParser.json())
11
12 /**before i forward the requests to my routes where i will send the response,
13  * i wanna add headers to any response.
14  * so i will set up a general middleware in app.js
15  * which gets my request response next function.
16  */
17 app.use((req, res, next) => {
18   /**'setHeader' only modify it and add a new header.
19   * the first header is the 'Access-Control-Allow-Origin'
20   * and we wanna set it to all the URLs or domain
21   * that should be able to access our server
22   * often you will just set this to '*'(start) to access from any client.
```

```

23  */
24  res.setHeader('Access-Control-Allow-Origin', 'start');
25  /**here we allow that origins to use specific HTTP methods
26  * because by just unlocking the origins,
27  * it would still not work.
28  * we also need to tell the clients, the origins
29  * which methods are allowed.
30  * there you can allow 'GET, POST, PUT, PATCH, DELETE'
31  * you can allow what you wanna be usable from outside.
32  */
33  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
34  /**'Headers' is that our clients might set on their requests
35  * now there are some default headers
36  * which are always allowed
37  * but you wanna add the 'Content-Type' header and 'Authorization' header
38  * so that your clients can send requests that hold extra authorization data in the
header
39  * and also define the content-type of the request
40  */
41  res.setHeader('Access-Control-Allow-Headers')
42  next()
43  })
44
45  app.use('/feed', feedRoutes)
46
47  app.listen(8080)

```

```

1  //HTML
2
3  <button id="get">Get Posts</button>
4  <button id="post">Create a Post</button>

```

```

1  //JS
2
3  const getButton = document.getElementById('get');
4  const postButton = document.getElementById('post');
5
6  getButton.addEventListener('click', () => {
7    fetch('http://localhost:8080/feed/posts')
8      .then(res => res.json())
9      .then(resData => console.log(resData))
10     .catch(err => console.log(err));
11  });

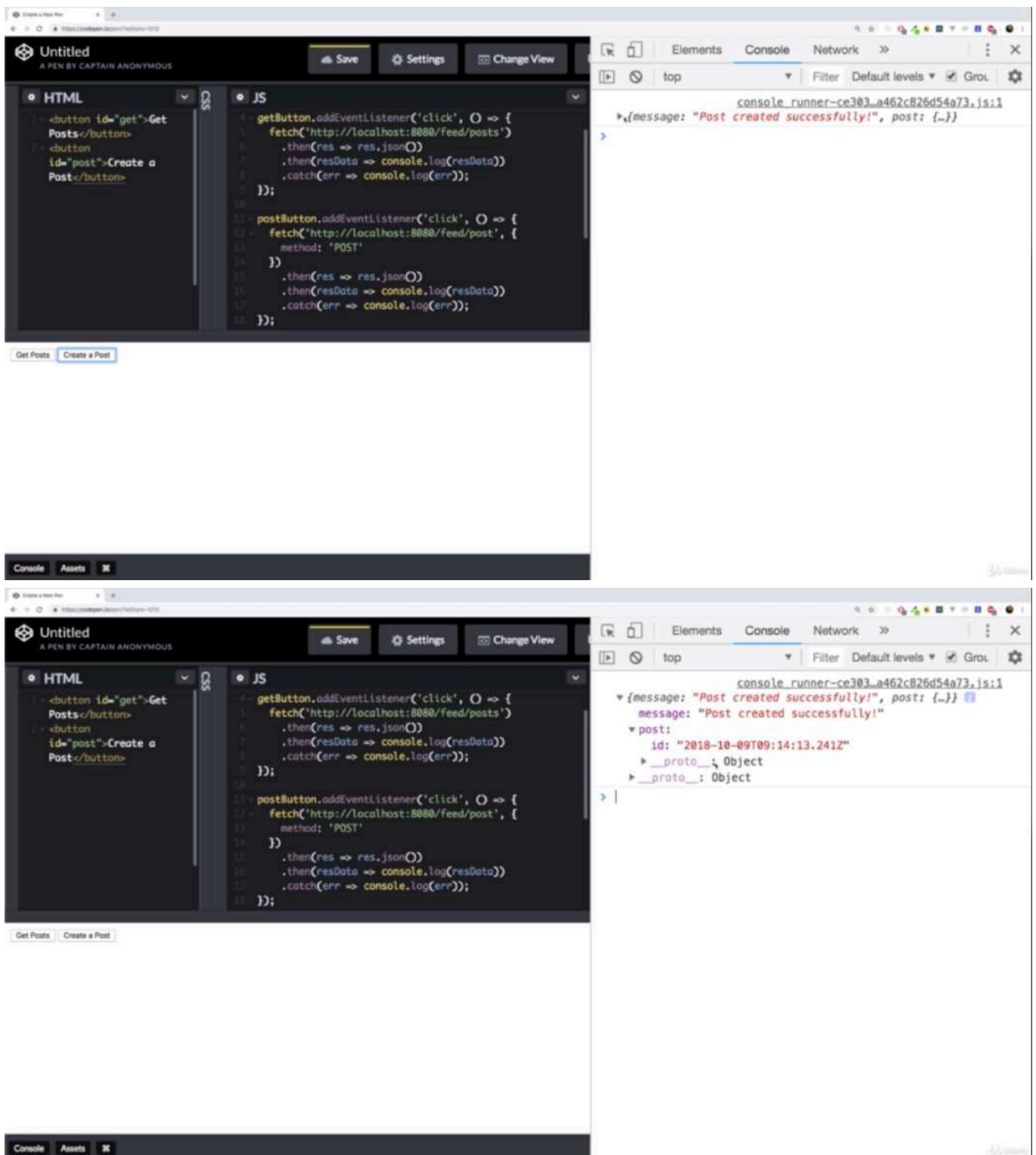
```

## \* Chapter 362: Sending POST Requests

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







- i get post created successfully but if we inspect the post object, we see that the title and content are missing because we didn't send that data.



```

1 exports.getPosts = (req, res, next) => {
2   res.status(200).json({
3     posts: [{ title: 'First Post', content: 'This is the first post!' }]
4   });
5 };
6
7 exports.createPost = (req, res, next) => {
8   const title = req.body.title;
9   const content = req.body.content;
10  console.log(title, content);
11  // Create post in db
12  res.status(201).json({
13    message: 'Post created successfully!',
14    post: { id: new Date().toISOString(), title: title, content: content }
15  });
16 };
17

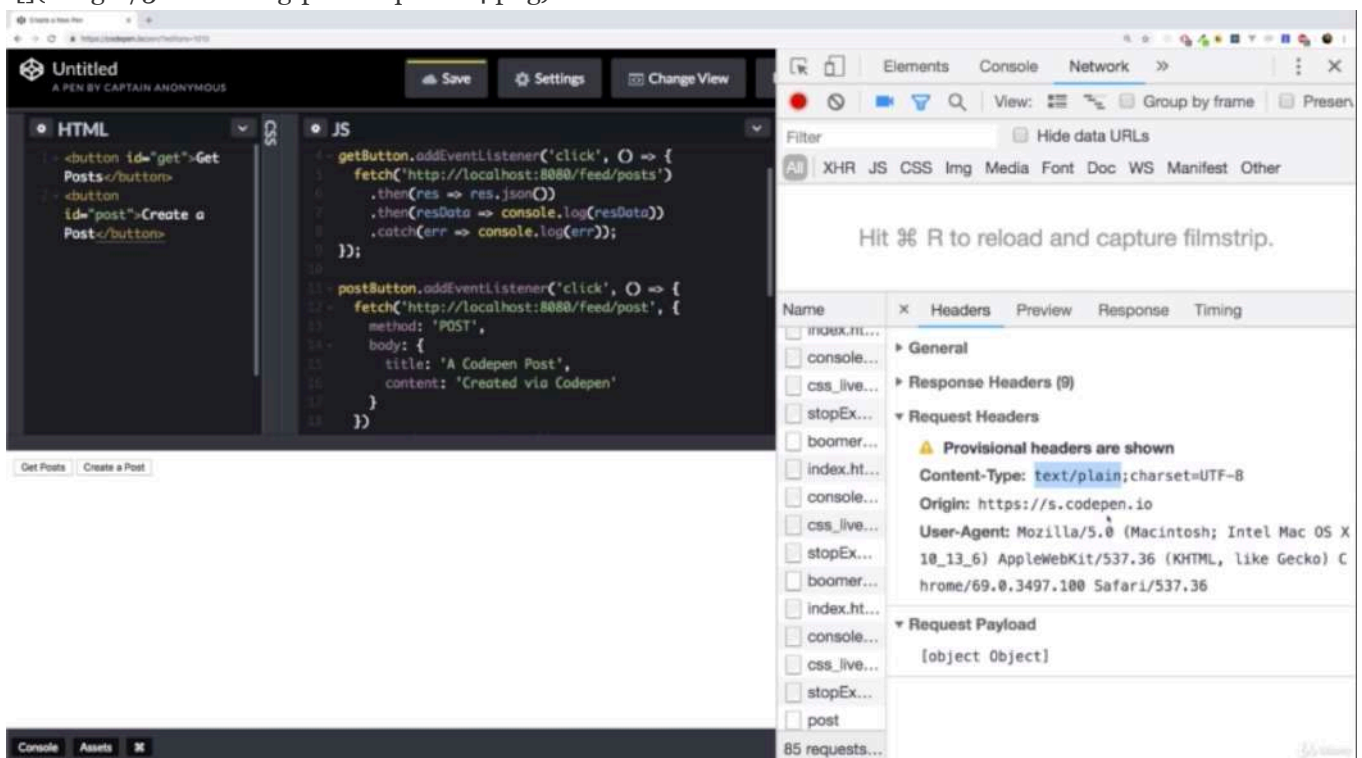
```

```

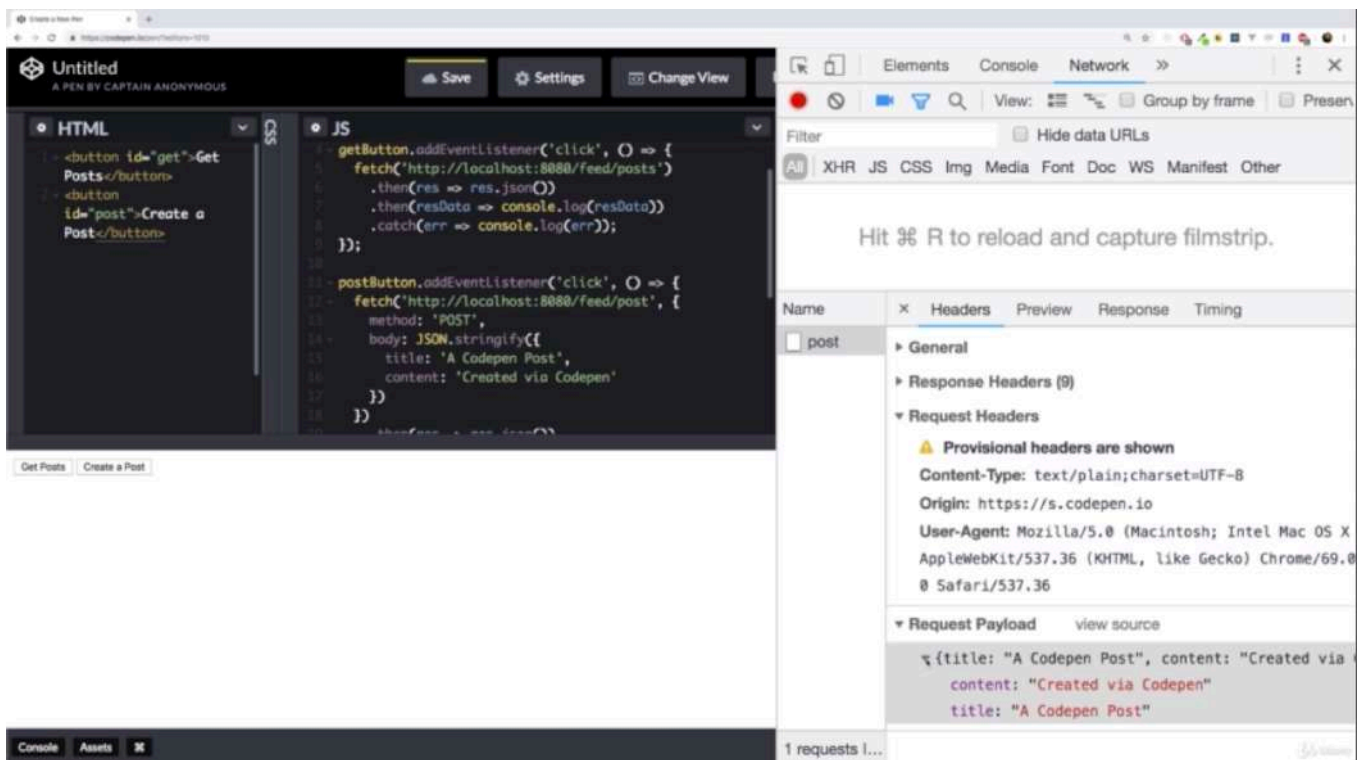
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
undefined undefined

```

- we can see in the console 'undefined undefined' so we are not able to extract that data  
 



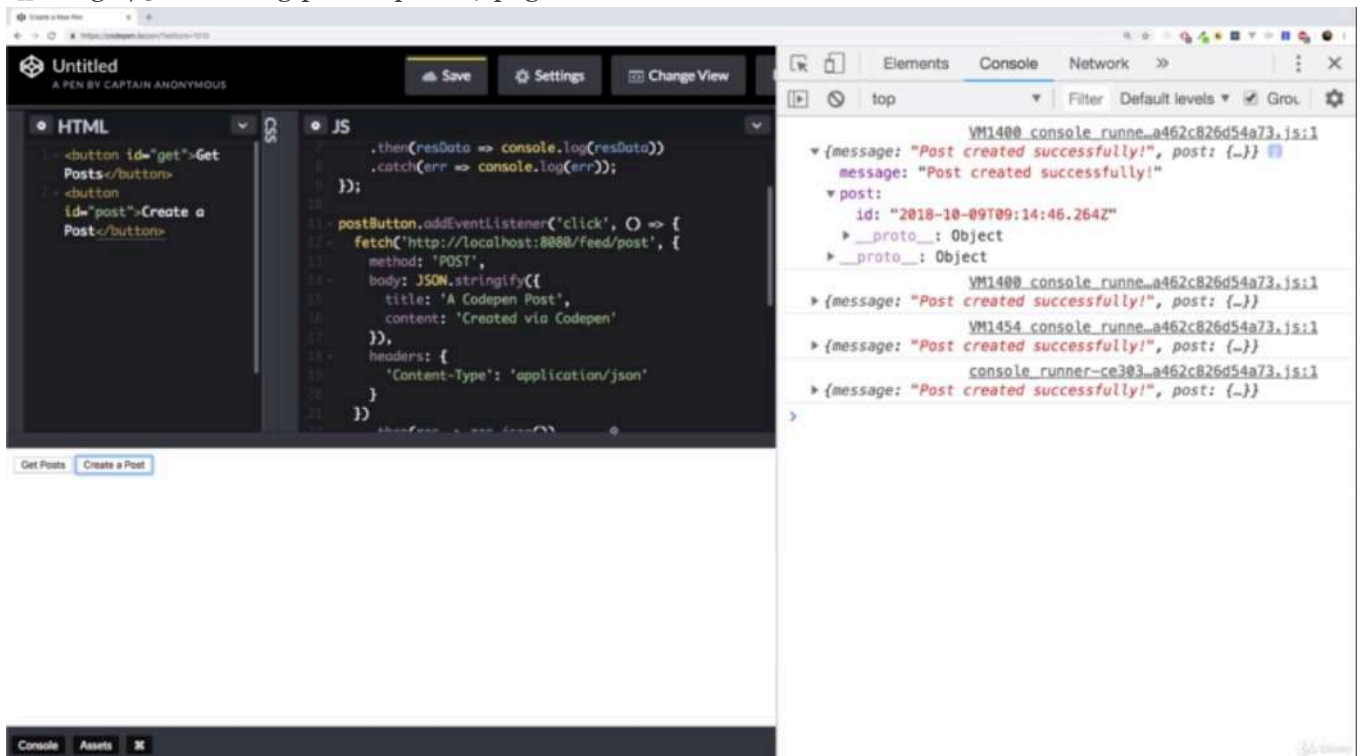
- because we see the content-type was text/plain and that is the problem. it should be application/json  
 - but we also see that the request payload was not json data which in the end is just text. but that it was a javascript object which can't be sent or which can't be handled.  
 - first of all on the body, i will call JSON.stringify() which is method provided by default by javascript. it will take a javascript object and convert it to JSON. we can see that immediately if i click 'Create a Post' again.  
 



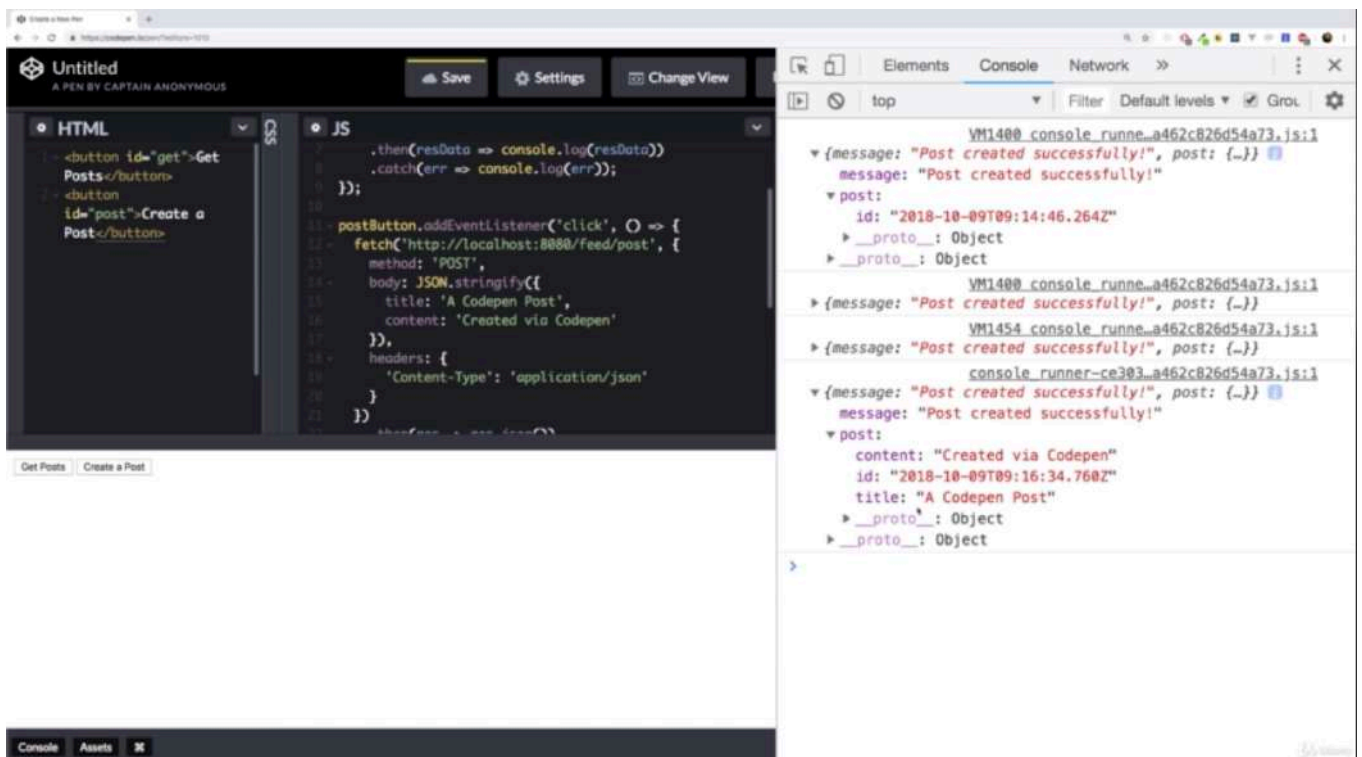
- the payload is text in the JSON format but we need to tell the server that our content







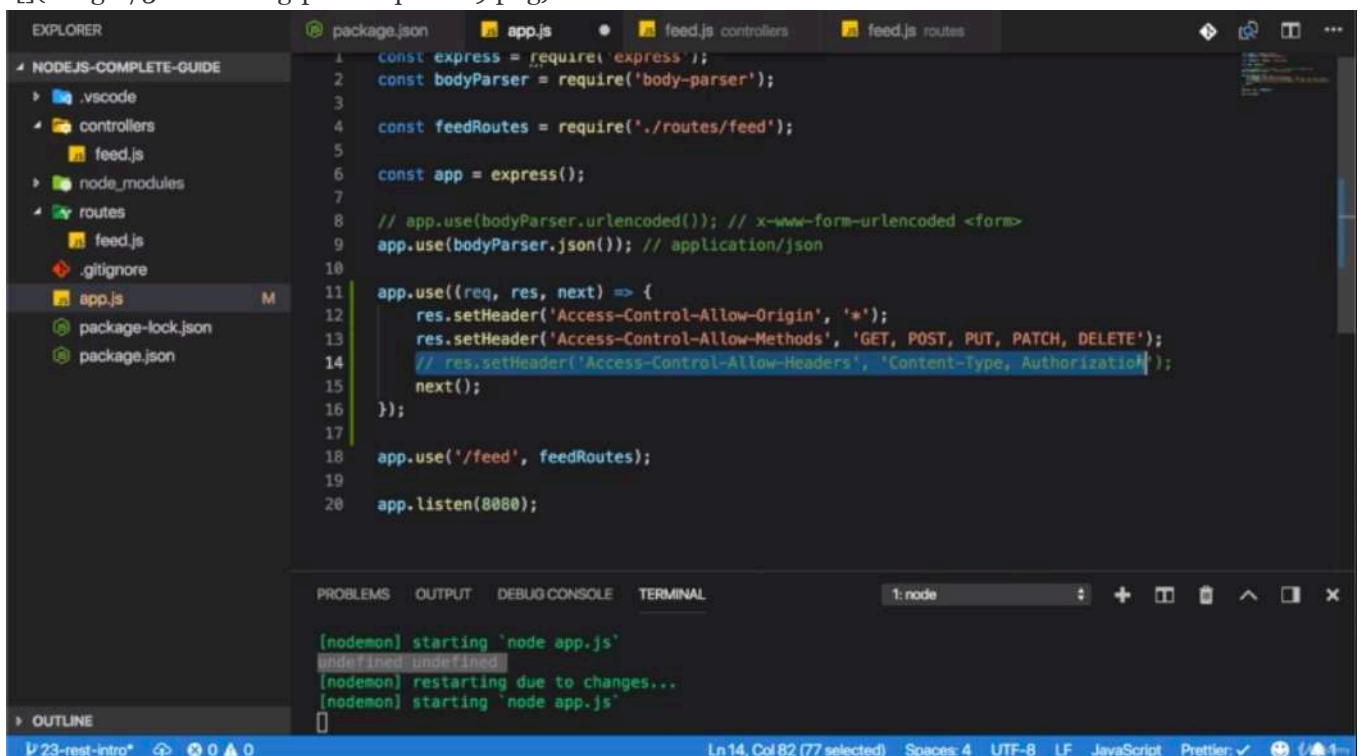




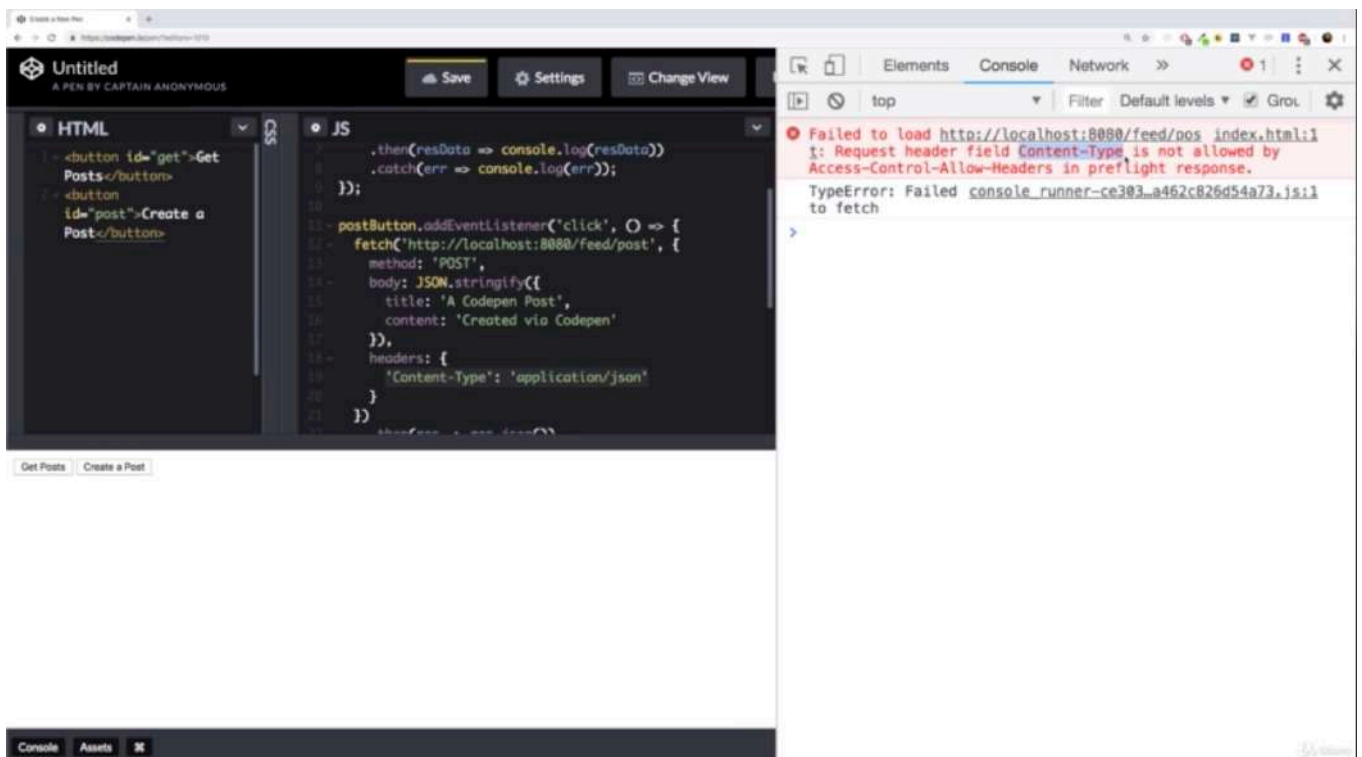
- we can see title or content because now that data is sent and extracted correctly because we send it in the right format and we inform the server about the content type.







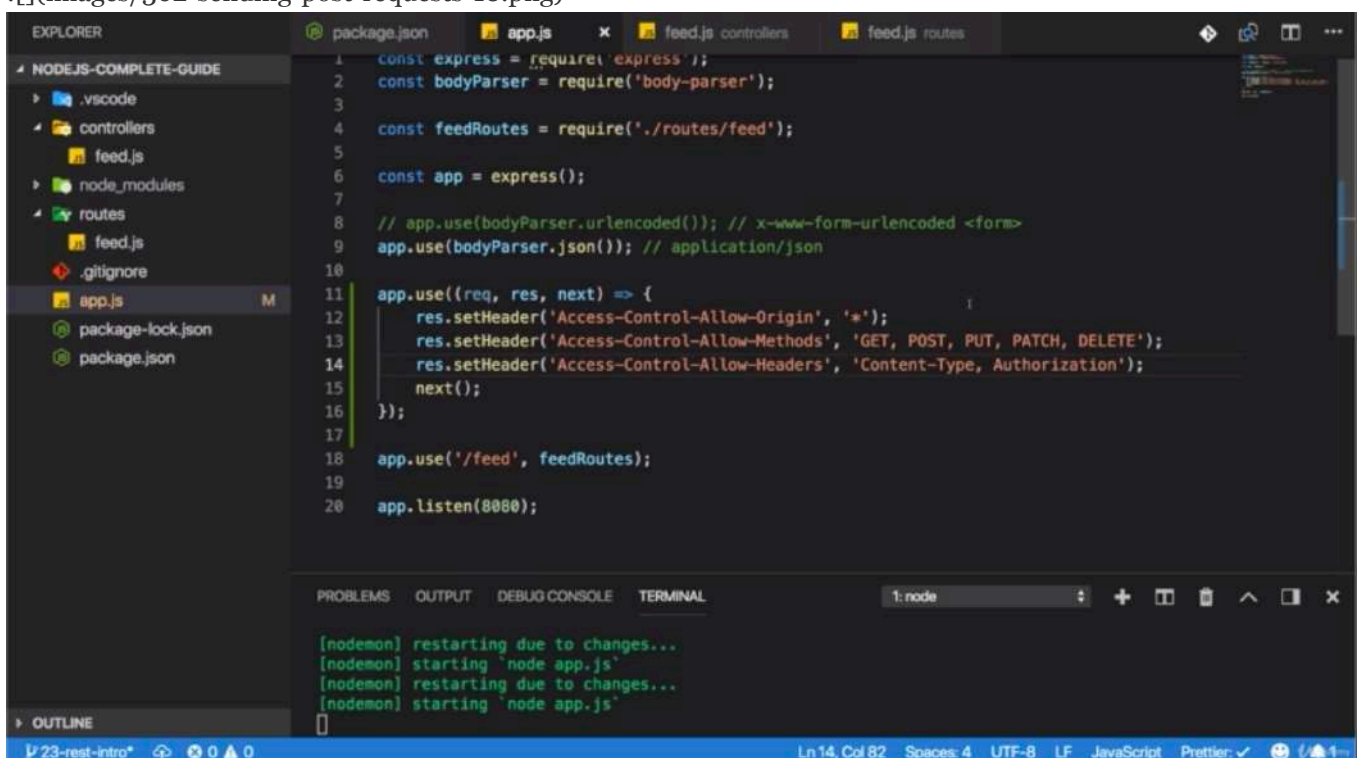




- this also allows me to demonstrate what happens if i would comment out this header here, the access-control-allow-headers on the server side.

- after clicking 'Create a Post', i fail because i'm not allowed to set content-type.

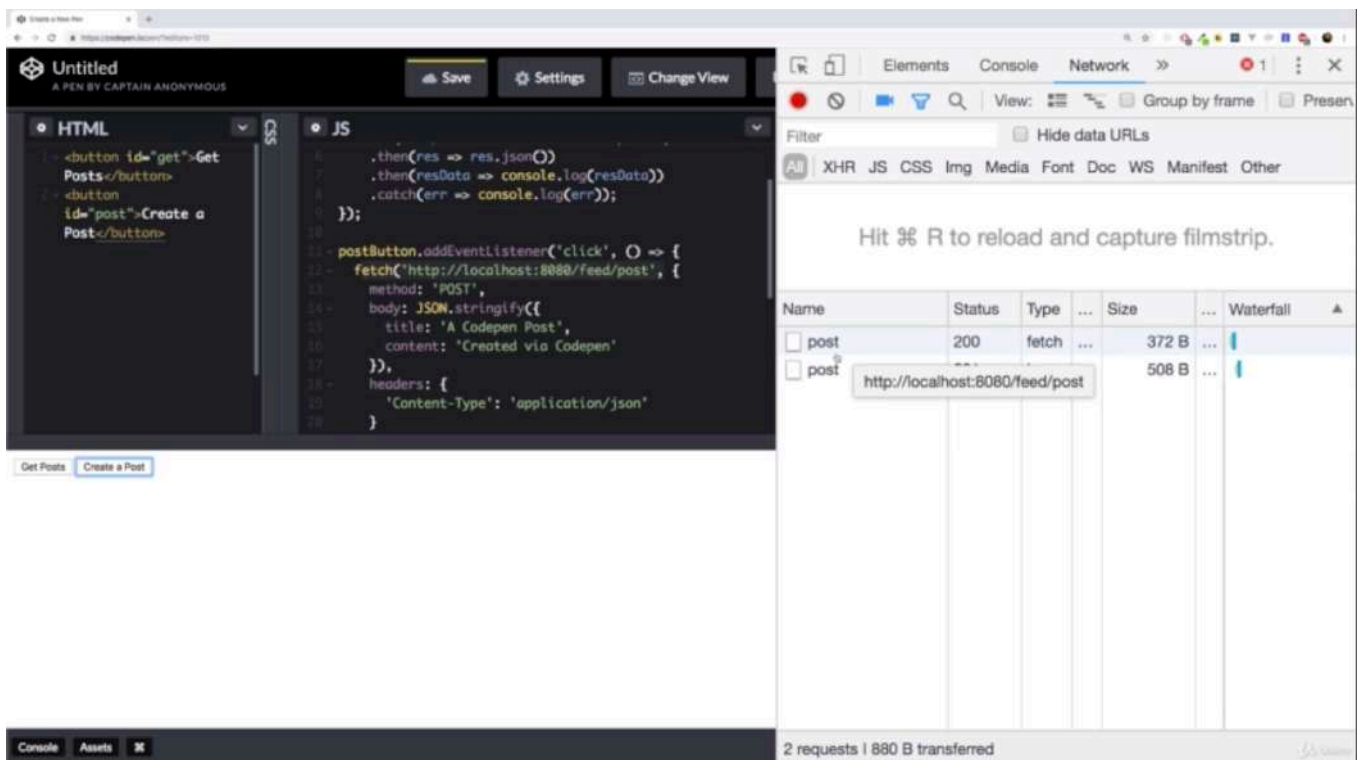




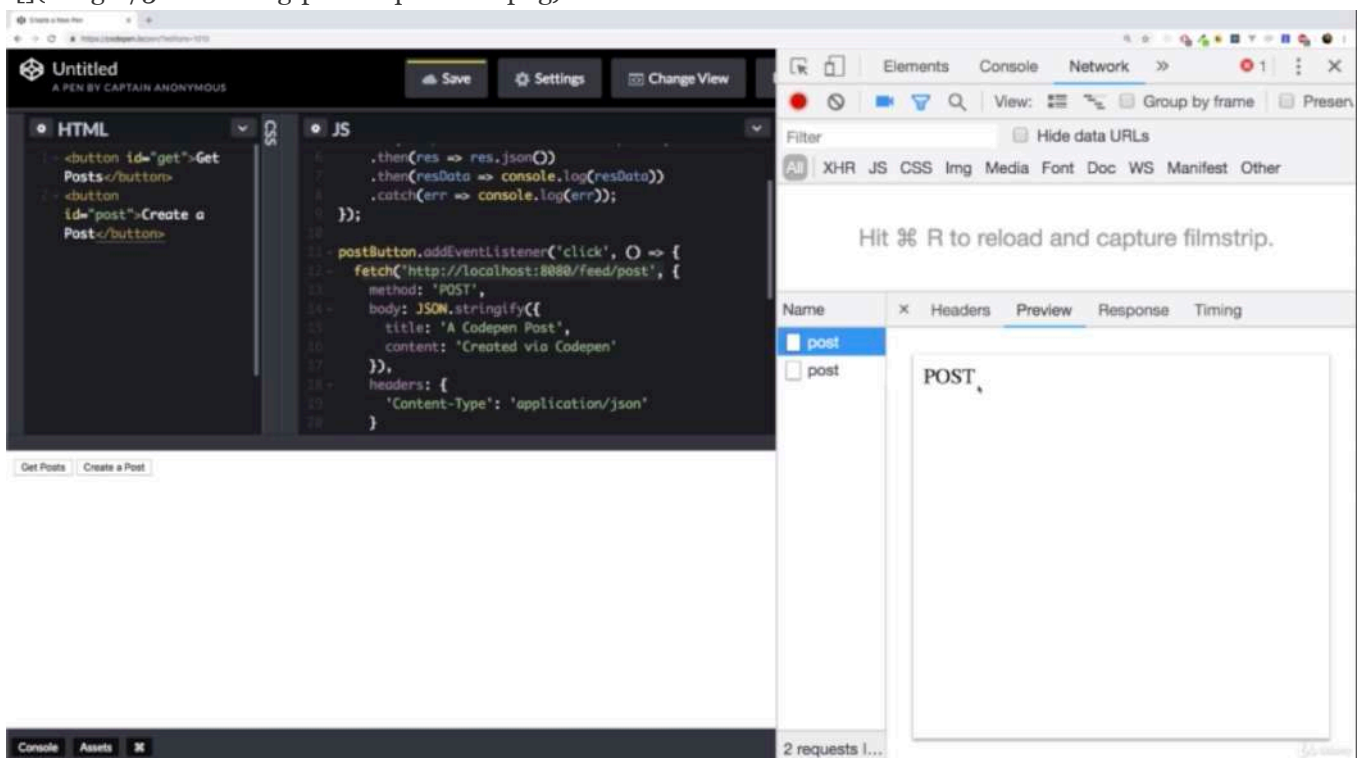
- i allow this by adding this header on the server side. this is how you communicate between client and server.

- the client code differs depending on the client you are using.





- you might see that i have 2 requests being sent.  
 



- the second one is our POST requests  
   
   
 

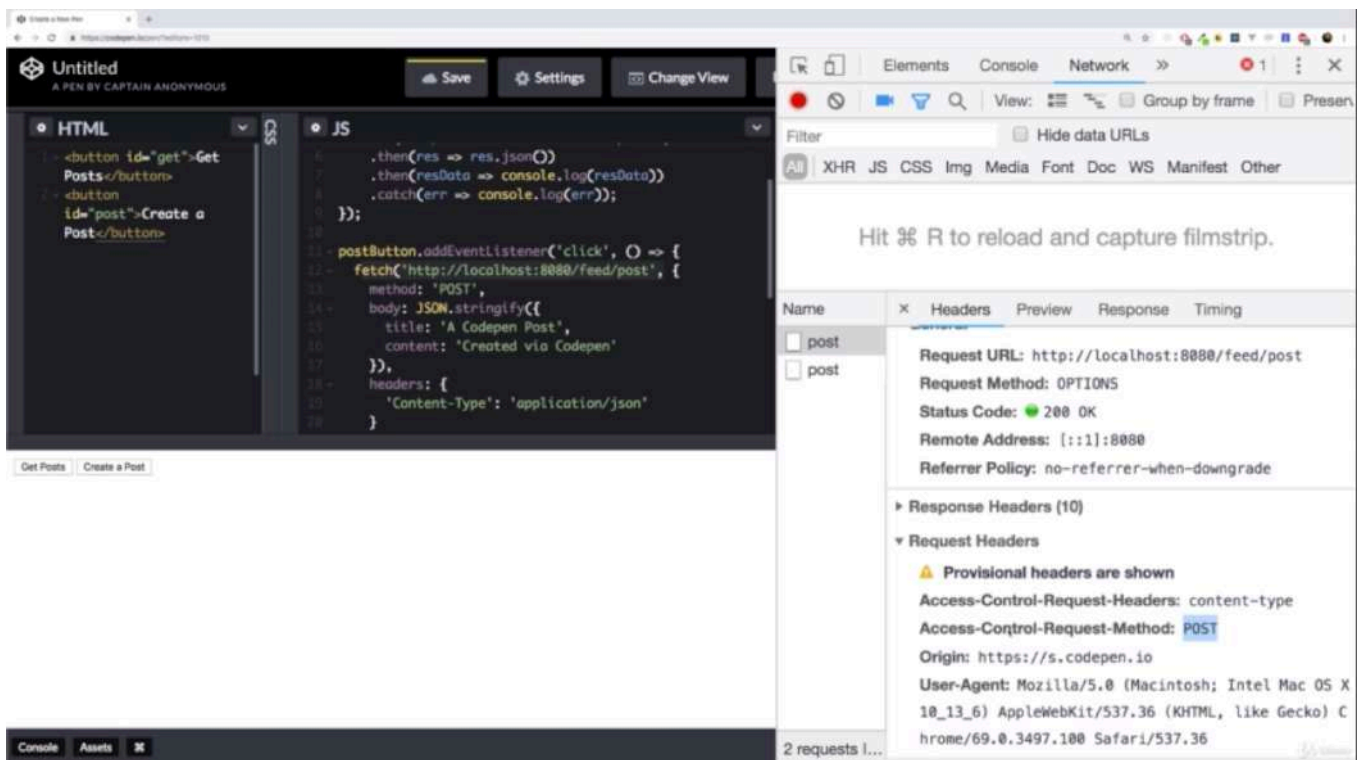
The screenshot shows a web browser with a REST client interface on the left and a network tab on the right. The REST client shows a POST request to `http://localhost:8080/feed/post` with a JSON body: `{ "title": "A Codepen Post", "content": "Created via Codepen" }`. The network tab shows the response with status `200 OK` and headers: `Access-Control-Request-Headers: content-type`, `Access-Control-Request-Method: POST`, `Origin: https://s.codepen.io`, `User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36`.



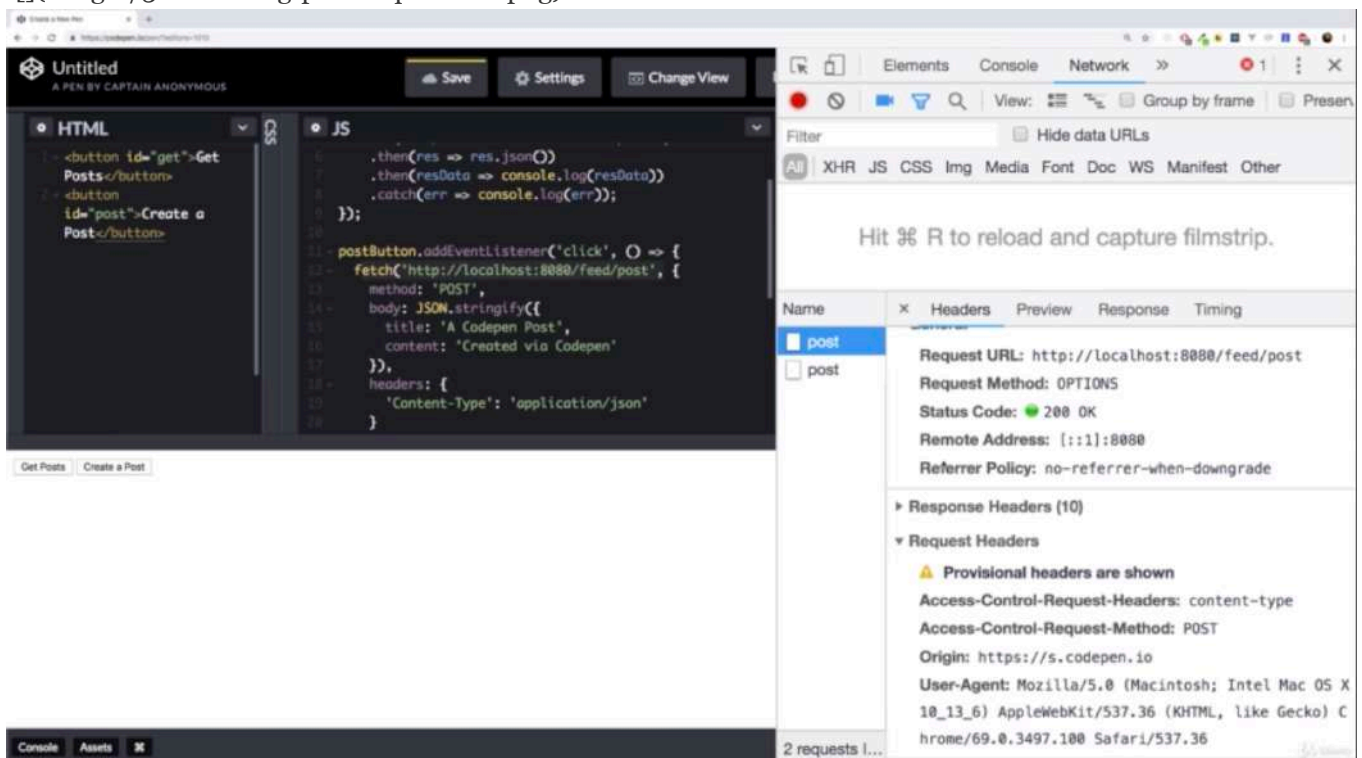
## Http Methods (Http Verbs)

More than just GET & POST

GET	POST	PUT
Get a Resource from the Server	Post a Resource to the Server (i.e. create or append Resource)	Put a Resource onto the Server (i.e. create or overwrite a Resource)
PATCH	DELETE	OPTIONS
Update parts of an existing Resource on the Server	Delete a Resource on the Server	Determine whether follow-up Request is allowed (sent automatically)

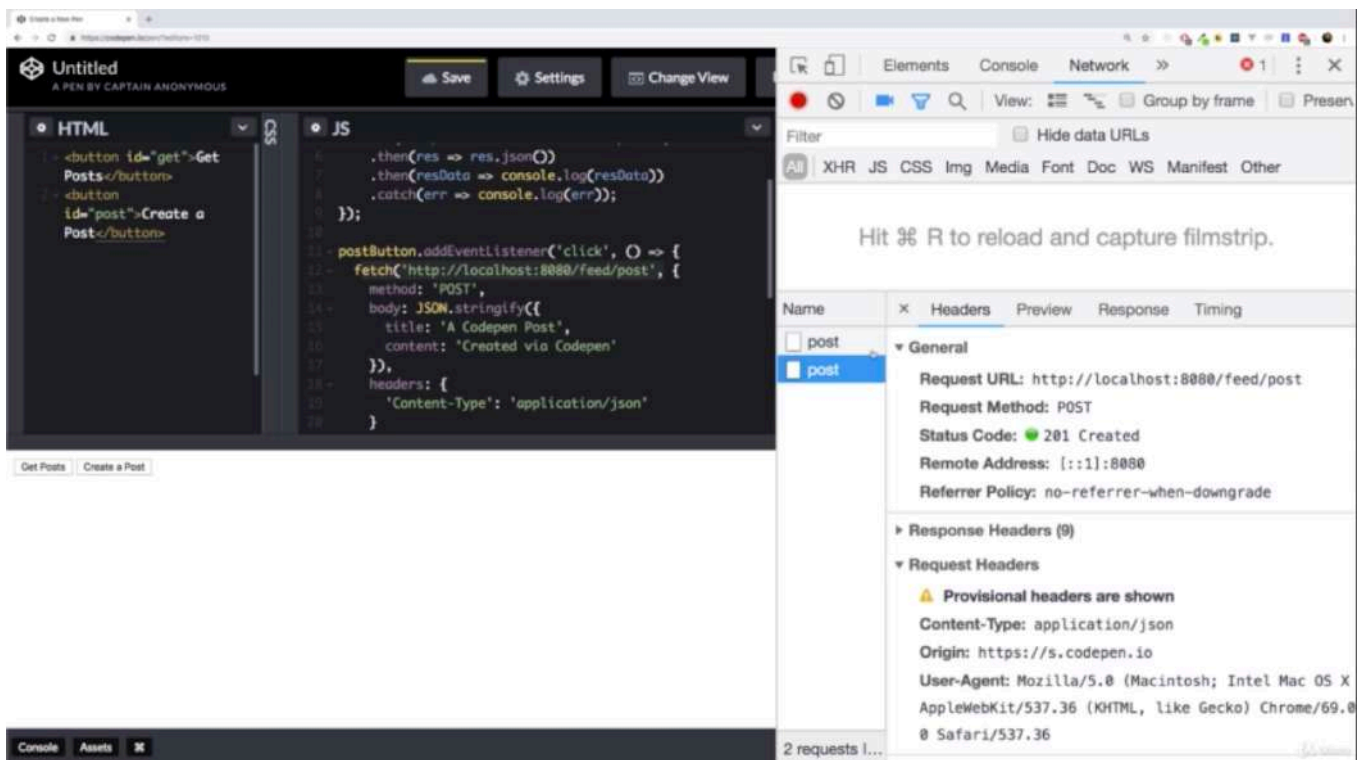


- but first request is just 'POST', and headers are interesting. Requesting Methods is OPTIONS which would be sent automatically by the browser and also by many mobile app clients. what is the idea behind the options?  
 



- the browser goes ahead and checks whether the requests you plan to send which is a POST request, that is why, in the request headers, which are generated automatically by the browser, it checks for the POST requests, it checks if that will be allowed. otherwise it will throw an error.  
 





- this is a mechanism the browser and many other clients use and there is not too much you need to do to make this work. you wanna make sure that you set the right CORS headers here. you can add OPTIONS here to the allowed methods
- but as you see, it was able to make this request before. this is not really something you need to do. but you can do it
- the important thing is that you are not confused by that extra request, it is a mechanism the browser uses to see if the next request, which it wanna view, the post request will succeed if it is allowed.

```

1 //HTML
2
3 <button id="get">Get Posts</button>
4 <button id="post">Create a Post</button>

```

```

1 //JS
2
3 const getButton = document.getElementById('get');
4 const postButton = document.getElementById('post');
5
6 getButton.addEventListener('click', () => {
7   fetch('http://localhost:8080/feed/posts')
8     .then(res => res.json())
9     .then(resData => console.log(resData))
10    .catch(err => console.log(err));
11 });
12
13 postButton.addEventListener('click' () => {
14   fetch('http://localhost:8080/feed/post', {
15     method: 'POST',
16     body: JSON.stringify({
17       title: 'A Codepen Post',
18       content: 'Created via Codepen'
19     }),
20     headers: {
21       'Content-Type': 'application/json'
22     }

```

```

23   })
24   .then(res => res.json())
25   .then(resData => console.log(resData))
26   .catch(err => console.log(err));
27 });

```

```

1  //app.js
2
3  const express = require('express')
4  const bodyParser = require('body-parser')
5
6  const feedRoutes = require('./routes/feed')
7
8  const app = express();
9
10 app.use(bodyParser.json())
11
12 app.use((req, res, next) => {
13   res.setHeader('Access-Control-Allow-Origin', 'start');
14   res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
15   res.setHeader('Access-Control-Allow-Headers')
16   next()
17 })
18
19 app.use('/feed', feedRoutes)
20
21 app.listen(8080)

```

```

1  //./controllers/feed.js
2
3  exports.getPosts = (req, res, next) => {
4   res.status(200).json({
5     posts: [{ title: 'First Post', content: 'This is the first post!' }]
6   });
7 };
8
9  exports.createPost = (req, res, next) => {
10   const title = req.body.title;
11   const content = req.body.content;
12   console.log(title, content);
13   //Create post in db
14   res.status(201).json({
15     message: 'Post created successfully!',
16     post: { id: new Date().toISOString(), title: title, content: content }
17   });
18 };

```

## \* Chapter 363: Wrap Up





## Module Summary

### REST Concepts & Ideas

- REST APIs are all about data, no UI logic is exchanged
- REST APIs are normal Node servers which expose different endpoints (Http method + path) for clients to send requests to
- JSON is the common data format that is used both for requests and responses
- REST APIs are decoupled from the clients that use them

### Requests & Responses

- Attach data in JSON format and let the other end know by setting the "Content-Type" header
- CORS errors occur when using an API that does not set CORS headers