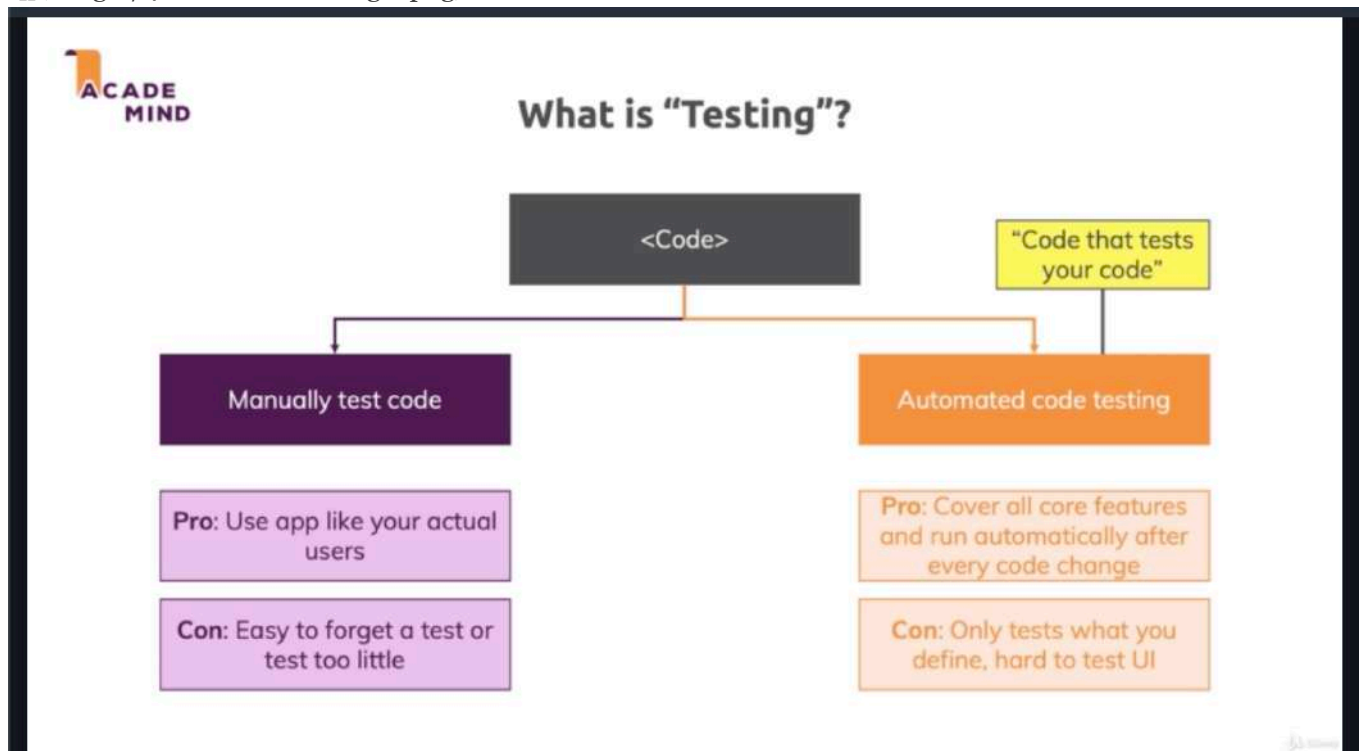


30. Testing Node.js Applications

* Chapter 460: What Is Testing?



- it's a combination of both. manual testing which we did for the course in which you naturally do and automated testing what you will learn about in this module.

* Chapter 461: Why & How?

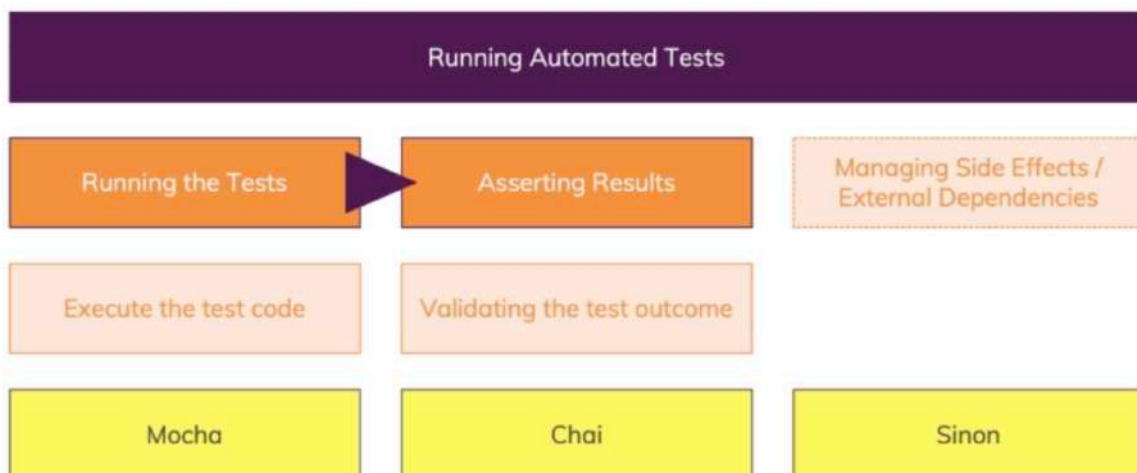
Why "Testing"?

Automatically test everything* after every code adjustment

Easily detect breaking changes (even in the places you didn't expect)

Ensure predictable and clearly defined testing steps

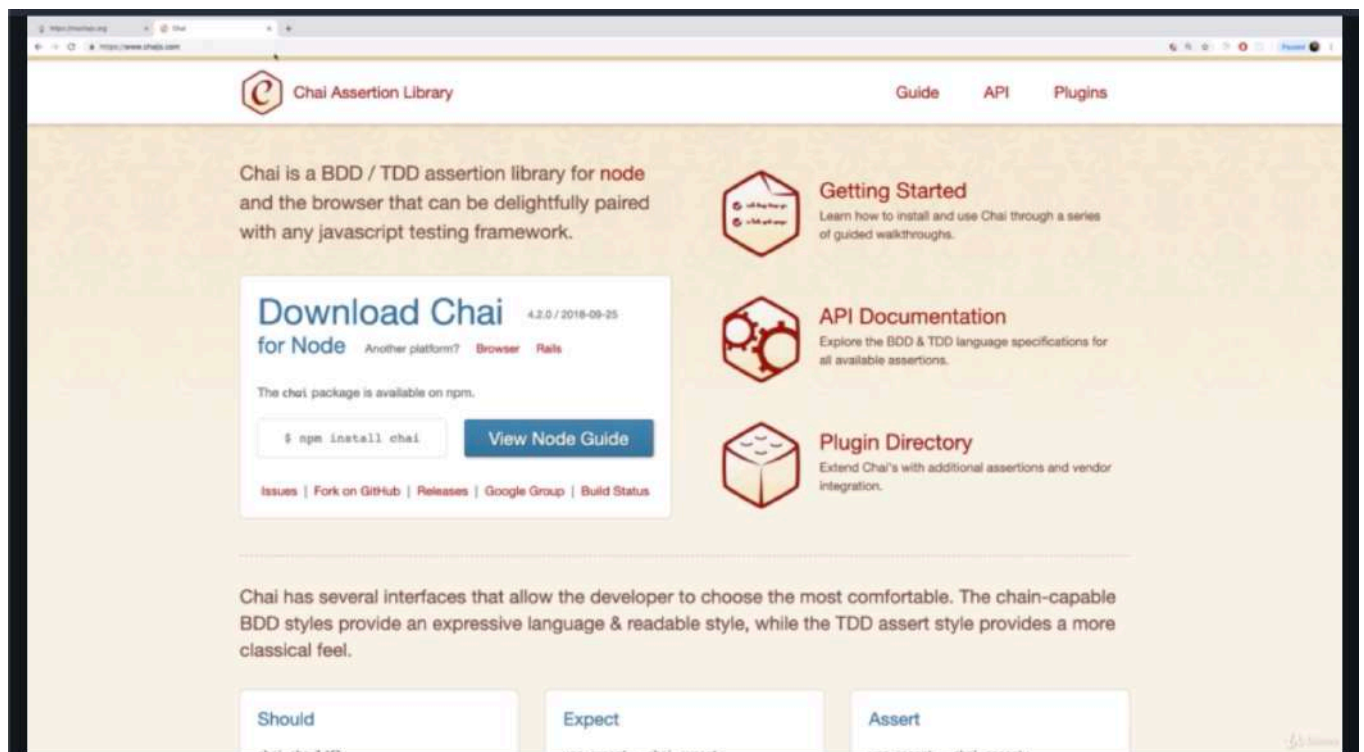
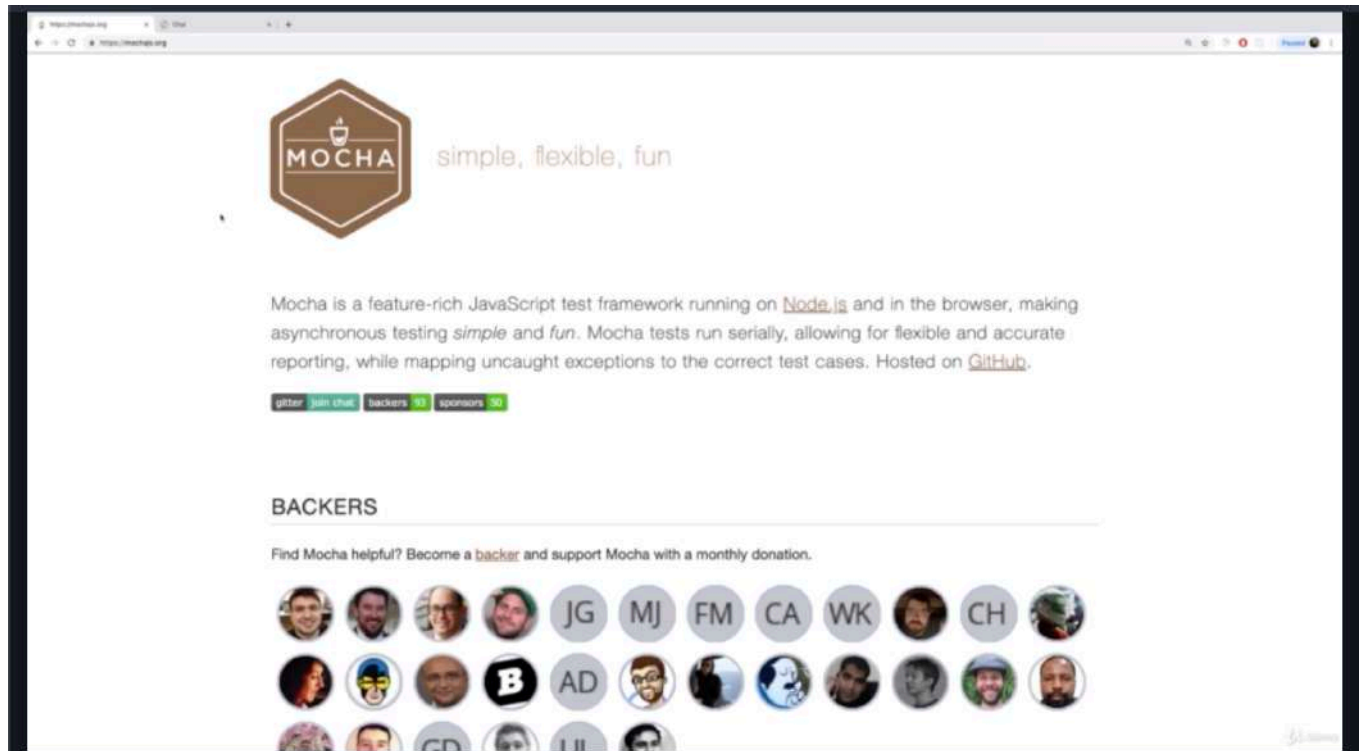
Testing Tools & Setup

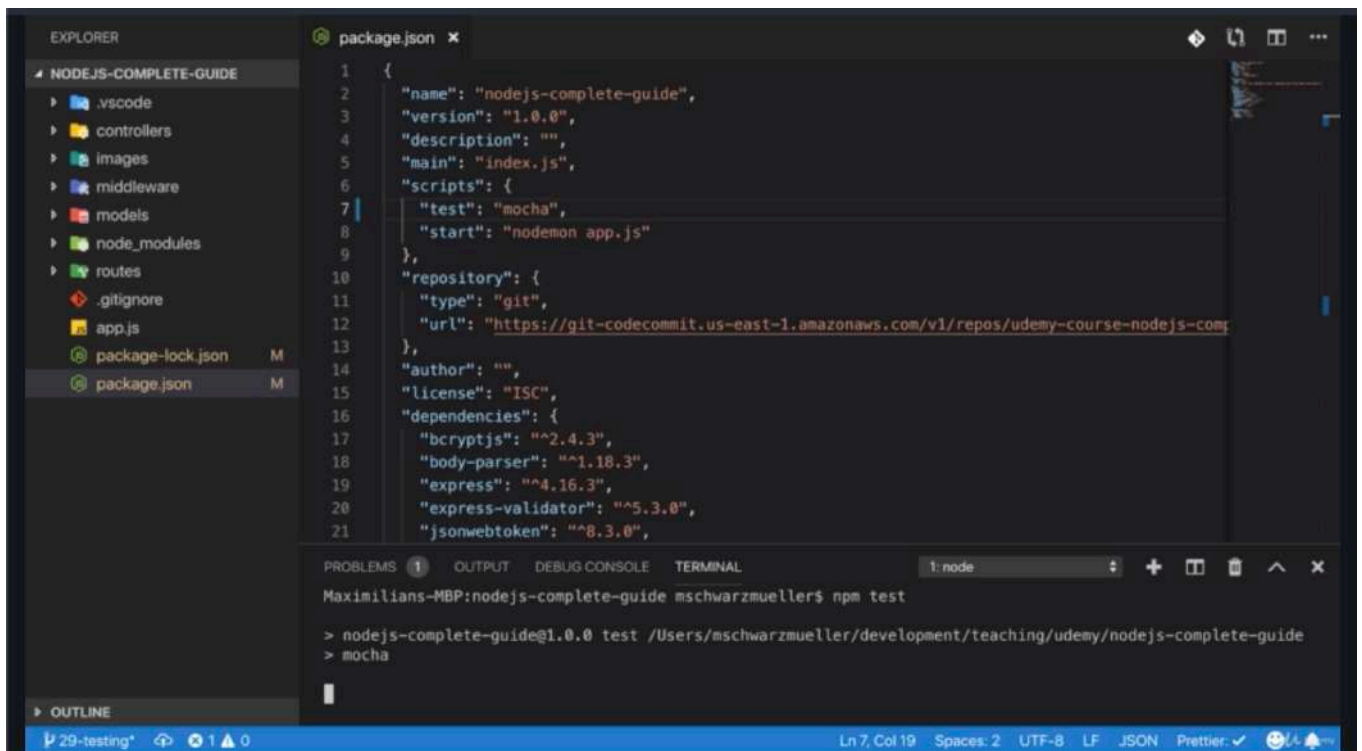
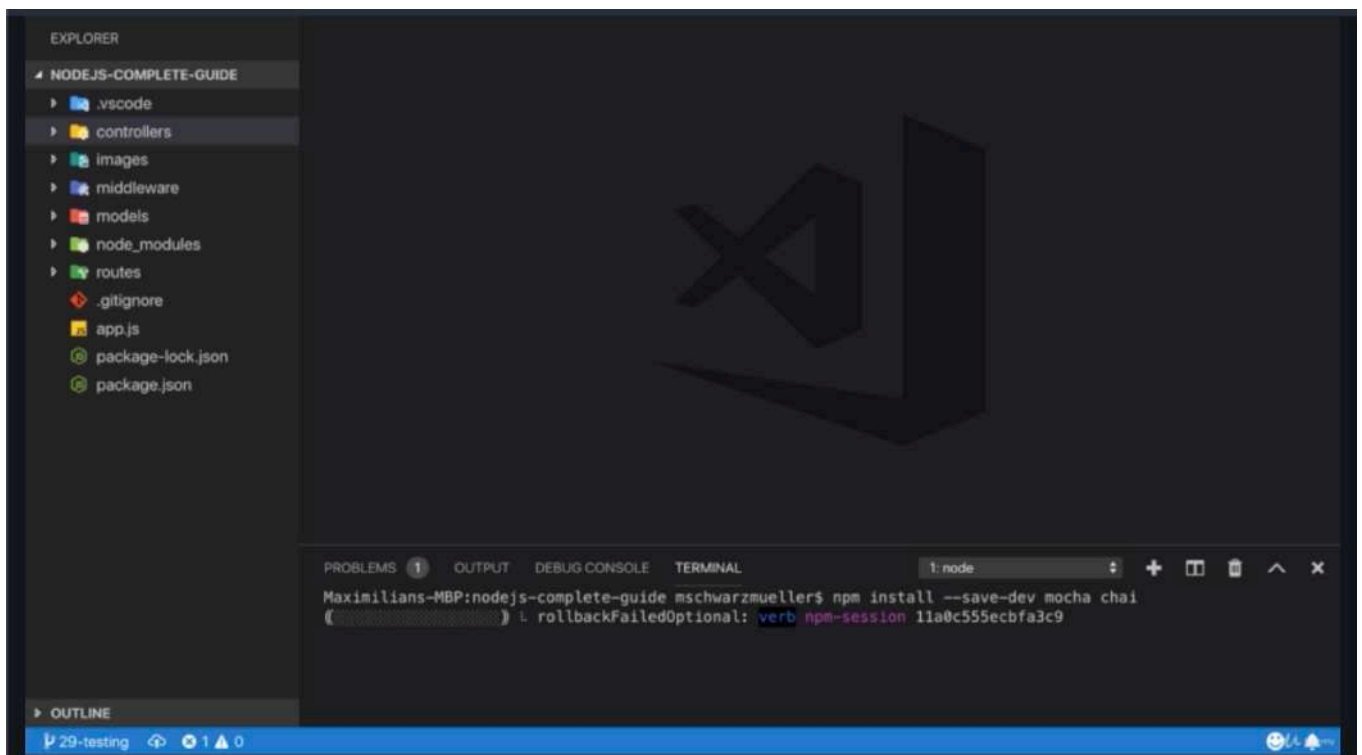


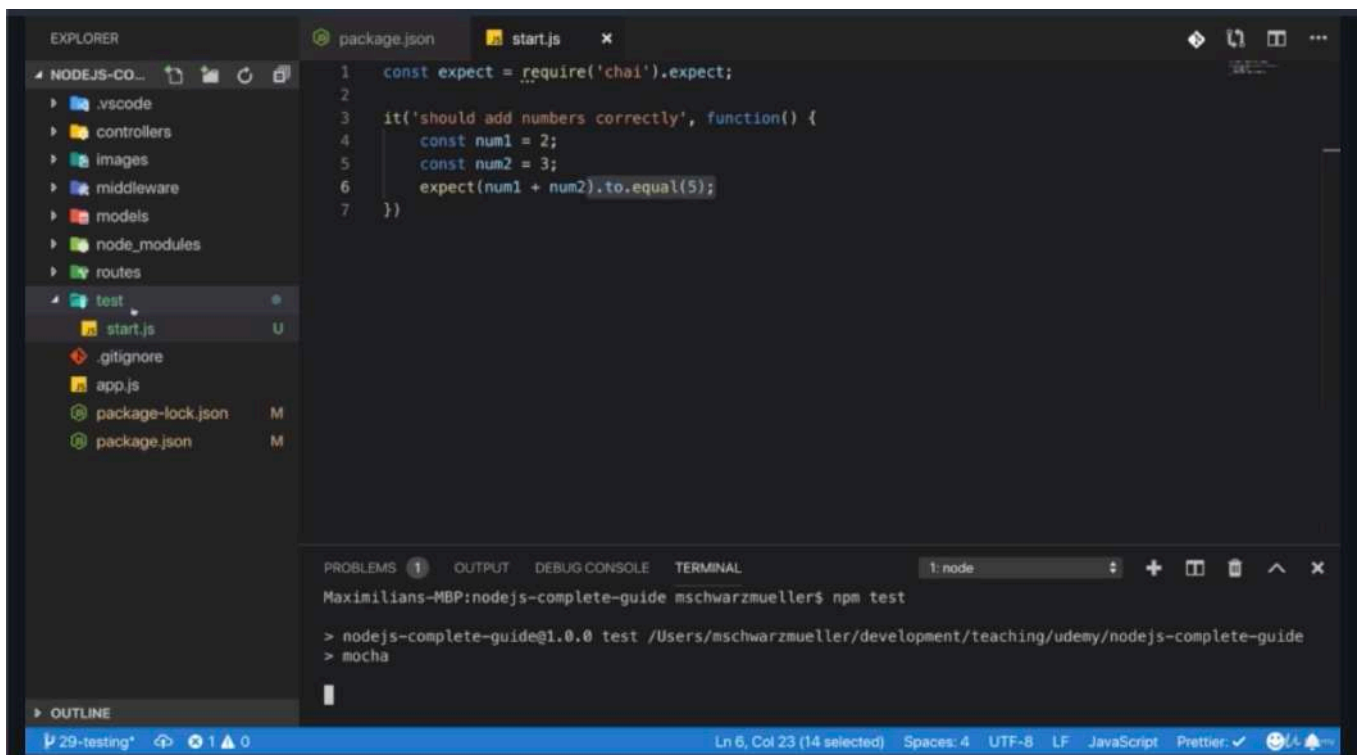
- we will use mocha and Chai in this module here too. there are alternatives like 'Jest' but mocha and Chai are really popular exist for a long time.

* Chapter 462: Setup And Writing A First Test

1. update
- package.json
- ./test/start.js

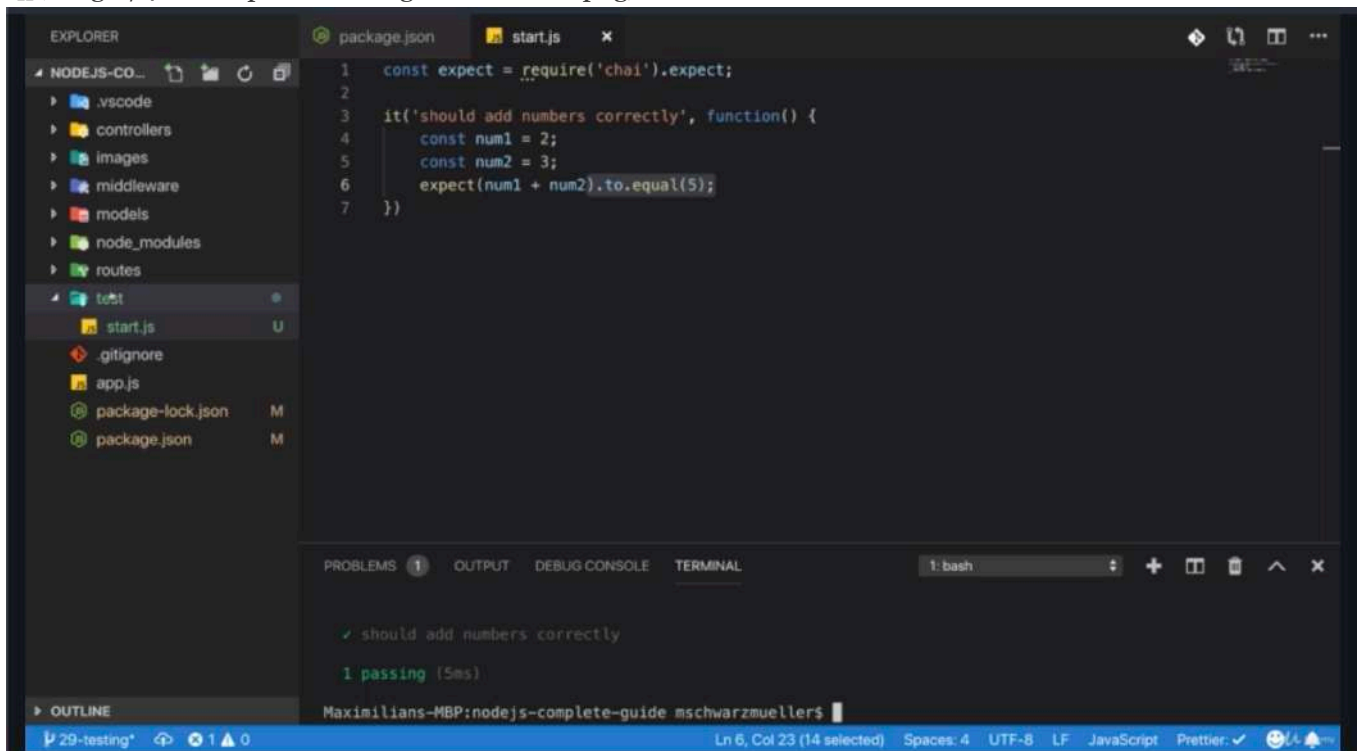






- 'npm test' will now run all your tests you defined in this project. and mocha by default looks for a folder named 'test'. it should be named as 'test' folder.

- so if you run npm test, it will automatically look for test folder and execute all tests and all files defined in that test folder.



- in the output, we see it a checkmark which is now executed correctly as you can see and that is what you wannna see when writing tests.

VS Code editor interface showing a file explorer on the left with a project named 'NODEJS-COMPLETE-GUIDE'. The file explorer lists directories like .vscode, controllers, images, middleware, models, node_modules, routes, and test. The 'test' directory is expanded, showing 'start.js' and 'package.json'. The 'start.js' file is open in the editor, showing a JavaScript test suite using Chai and Mocha. The test suite has two tests: 'should add numbers correctly' and 'should not give a result of 6'. The second test is failing, indicated by a yellow lightbulb icon. The terminal at the bottom shows the command 'npm test' and the output '1 passing (5ms)'.

```
1 const expect = require('chai').expect;
2
3 it('should add numbers correctly', function() {
4   const num1 = 2;
5   const num2 = 3;
6   expect(num1 + num2).to.equal(5);
7 })
8
9 it('should not give a result of 6', function() {
10  const num1 = 2;
11  const num2 = 3;
12  expect(num1 + num2).not.to.equal(6);
13 })
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: node

1 passing (5ms)

Maximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm test

> nodejs-complete-guide@1.0.0 test /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide

> mocha

VS Code editor interface showing the same file explorer and 'start.js' file as the first image. The test suite is still open, but the second test is now passing, indicated by a green checkmark icon. The terminal at the bottom shows the command 'npm test' and the output '2 passing (6ms)'.

```
1 const expect = require('chai').expect;
2
3 it('should add numbers correctly', function() {
4   const num1 = 2;
5   const num2 = 3;
6   expect(num1 + num2).to.equal(5);
7 })
8
9 it('should not give a result of 6', function() {
10  const num1 = 2;
11  const num2 = 3;
12  expect(num1 + num2).not.to.equal(6);
13 })
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

✓ should add numbers correctly

✓ should not give a result of 6

2 passing (6ms)

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

This screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The main editor shows `start.js` with two test cases using Chai and Mocha. The bottom panel shows the Terminal with the output of `npm test`, indicating that both tests passed.

```
1  const expect = require('chai').expect;
2
3  it('should add numbers correctly', function() {
4    const num1 = 2;
5    const num2 = 3;
6    expect(num1 + num2).to.equal(5);
7  })
8
9  it('should not give a result of 6', function() {
10   const num1 = 3;
11   const num2 = 3;
12   expect(num1 + num2).not.to.equal(6);
13 })
```

```
2 passing (6ms)

Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm test
> nodejs-complete-guide@1.0.0 test /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide
> mocha
```

This screenshot shows the same VS Code interface, but the test has failed. The terminal output shows a failing test case with an `AssertionError`. The error message indicates that the expected result (6) did not match the actual result (5).

```
1 failing

1) should not give a result of 6:

AssertionError: expected 6 to not equal 5
+ expected - actual

at Context.<anonymous> (test/start.js:12:32)
```

- if i change to get failed result, then get failed test result.

```
1 {
2   "name": "nodejs-complete-guide",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "mocha",
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-complete"
```

```

13 },
14 "author": "",
15 "license": "ISC",
16 "dependencies": {
17   "bcryptjs": "^2.4.3",
18   "body-parser": "^1.18.3",
19   "express": "^4.16.3",
20   "express-validator": "^5.3.0",
21   "jsonwebtoken": "^8.3.0",
22   "mongoose": "^5.3.2",
23   "multer": "^1.4.0"
24 },
25 "devDependencies": {
26   "chai": "^4.2.0",
27   "mocha": "^6.1.4",
28   "nodemon": "^1.18.4"
29 }
30 }
31

```

```

1  //./test/start.js
2
3  /**they are all different way of defining our conditions
4   * like 'should, or 'assert'
5   */
6  const expect = require('chai').expect;
7
8  /**'it()' function is provided by mocha
9   * that gives your test name and read like plain english sentences
10  * because indeed it takes 2 arguments
11  * and the 1st one is a string which describes your test.
12  * so this should be a title
13  * that describes what is happening in the test.
14  *
15  * 2nd argument is the function
16  * which defines your actual test code.
17  */
18  it('should add numbers correctly', function() {
19    const num1 = 2;
20    const num2 = 3;
21    /**mocha is responsible for running our tests
22     * and for giving us this 'it()' function
23     * that defines where we define our test code
24     * chai is responsible for defining our success conditions
25     * and for this we just need to import something Chai
26     *
27     * inside 'expect()',
28     * you pass your code or result that you wanna test
29     * and there you have properties like 'to'
30     * and then on 'to' property,
31     * you have another object which gives you things like 'equal()'
32     * which is a function where you can define the value you expect as 'to.equal()'
33     */
34    expect(num1 + num2).to.equal(5)
35  })
36
37  it('should not give a result of 6', function(){

```



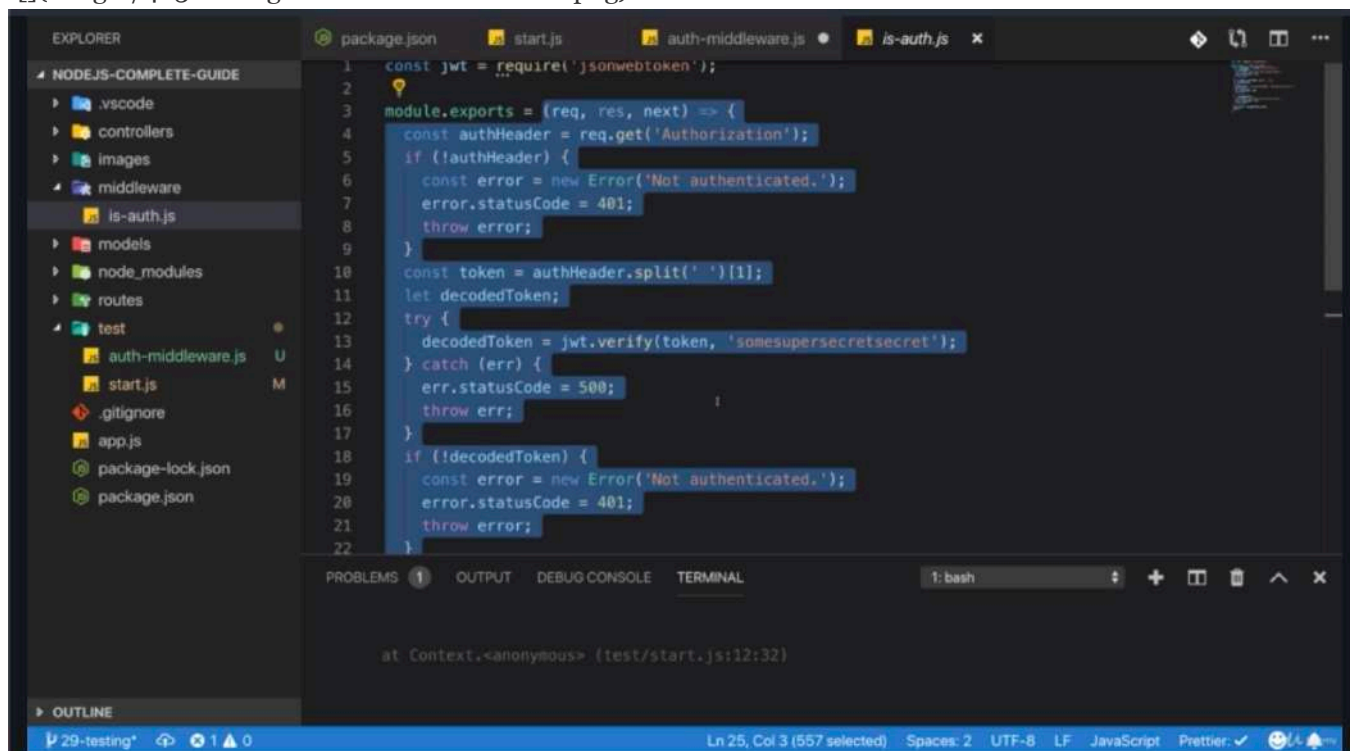
```

38     const num1 = 2;
39     const num2 = 3;
40     /**we are checking whether this is not only 5 but also not 6*/
41     expect(num1 + num2).not.to.equal(6);
42 })

```

* Chapter 463: Testing The Auth Middleware

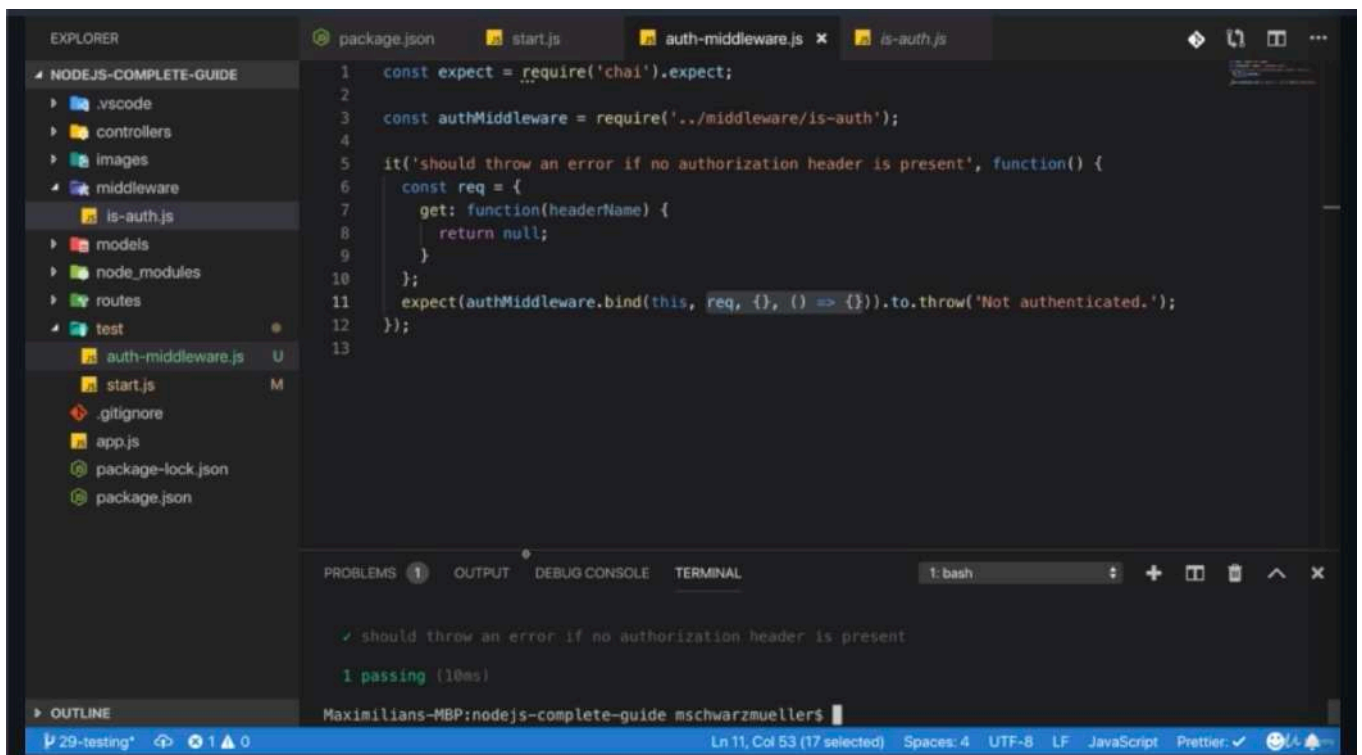
1. update
- ./test/auth-middleware.js



- we are not simulating a click of the user which then sends a request which then triggers the middleware. we just wanna test our middleware function. this is called 'unit test' we test one unit of our application. in this case, these code is a unit the function.

- 'intergration test' would be where we test a more complete flow. whether your request is routed correctly and then all of the middleware and then also the controller function. but you don't test that too often because it's very complex to test such long chains. there are multiple things to fail. by unit test, it's way easier to test different scenarios for each unit. and if all your unit tests succeed, you have a great chance of your overall application working correctly. and therefore unit test is very helpful. if a unit test fails, it's easy to find out why it fails.

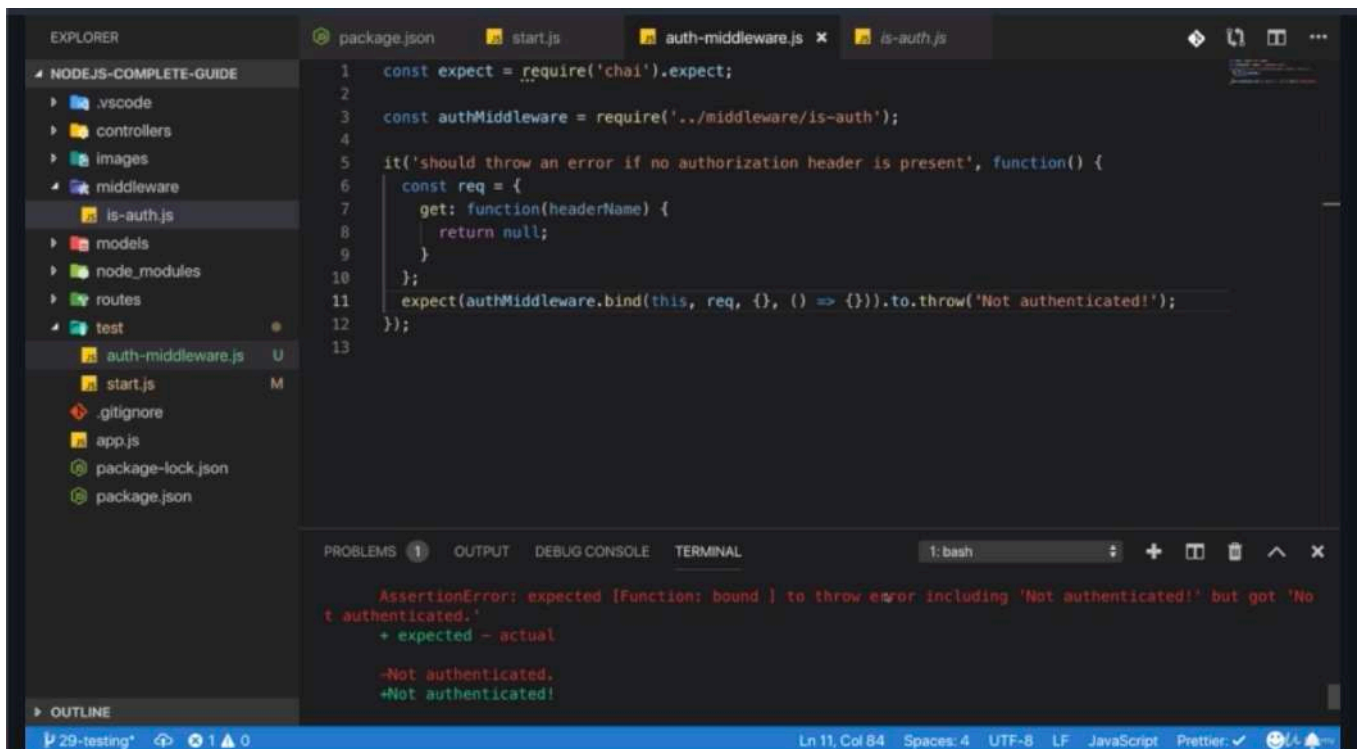
- i know that the code i wanna test calls to get method and i wanna return null to simulate that there is no authorization header because i know the code and testing and i wanna test different scenarios and this is how you have to think about testing your testing different scenarios. you are not trying to rebuild the framework. you wanna force your code into certain scenarios you wanna test them under certain scenarios. and here the scenario is that the get method on the request object returns null.



```
1 const expect = require('chai').expect;
2
3 const authMiddleware = require('../middleware/is-auth');
4
5 it('should throw an error if no authorization header is present', function() {
6   const req = {
7     get: function(headerName) {
8       return null;
9     }
10  };
11  expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated.');
```

✓ should throw an error if no authorization header is present

1 passing (10ms)



```
1 const expect = require('chai').expect;
2
3 const authMiddleware = require('../middleware/is-auth');
4
5 it('should throw an error if no authorization header is present', function() {
6   const req = {
7     get: function(headerName) {
8       return null;
9     }
10  };
11  expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated!');
```

AssertionError: expected [Function: bound] to throw error including 'Not authenticated!' but got 'Not authenticated.'

+ expected - actual

-Not authenticated.

+Not authenticated!

- if you are expecting a slightly different error message from 'Not authenticated.' to 'Not authenticated!' which is different from ./middleware/is-auth.js file, then you get a failure because we expect it to throw an error including 'Not authenticated!' not 'Not authenticated.' so we have to be clear because you can test for all kinds of different things.

- therefore this test only succeeds if we have an error.


```
2 module.exports = (req, res, next) => {
3   const authHeader = req.get('Authorization');
4   // If (!authHeader) {
5   //   const error = new Error('Not authenticated.');
```

```
1: bash
nnot read property '\split\' of null'
+ expected - actual

-Cannot read property 'split' of null
+Not authenticated.

at Context.<anonymous> (test/auth-middleware.js:11:64)
```

```
1 const expect = require('chai').expect;
2
3 const authMiddleware = require('../middleware/is-auth');
```

```
1: bash
nnot read property '\split\' of null'
+ expected - actual

-Cannot read property 'split' of null
+Not authenticated.
```

- the cool thing is if we ever change something about this code for example, we decide to remove this check where we check for the header and i run my npm test, i get an error and that tells me i need to do something because my test failed. this test where i wanna make sure that not being able to retrieve the header throw the error. that doesn't pass anymore and therefore i should have a look at my middleware and i can find out i committed that out or maybe i never added this functionality in the first place.

```
1 //./test/auth-middleware.js
2
3 const expect = require('chai').expect;
4
5 const authMiddleware = require('../middleware/is-auth');
```

```
1 //./test/auth-middleware.js
2
3 const expect = require('chai').expect;
4
5 const authMiddleware = require('../middleware/is-auth');
```

```

9      * which then triggers the middleware
10     * we just wanna test our middleware function
11     * this is called 'unit test'
12     * by the way we test one unit of our application
13     *
14     * we wanna define our own request object
15     * and that is great
16     * because that allows us to define different scenarios.
17     */
18     const req = {
19       /**'get' is from
20        * 'const authHeader = req.get('Authorization')'
21        * in ./middleware/is-auth.js
22        *
23        * in reality,
24        * it returns the value of authorization header
25        * in reality,
26        * it doesn't only scan headers,
27        * it scans different parts of the incoming request
28        * but our goal now is not to replicate the express.js
29        * but to test one specific scenario
30        * so 'get('Authorization')' doesn't return an authorization header
31        * because that's what we wanna test here
32        * that in this case we throw an error.
33        * */
34       get: function(headerName){
35         /**this means it doesn't return a value for our authorization call here
36          * i know that the code i wanna test calls to get method
37          * and i wanna return null to simulate
38          * that there is no authorization header
39          * because i know the code and testing
40          * and i wanna test different scenarios
41          * and this is how you have to think about testing different scenarios.
42          * you are not trying to rebuild the framework.
43          * you wanna force your code into certain scenarios
44          * you wanna test them under certain scenarios.
45          * and here the scenario is that the get method on the request object returns
46          null.
47          */
48         return null;
49       };
50       /**we can call authMiddleware
51        * and pass in our own request object
52        * which has nothing else
53        * the request object normally has
54        * but it has everything we need for this test there.
55        * we only need get method
56        * and it has that now for the response object we can pass on an empty object
57        * because we are not testing anything related to this response object
58        * and the code we are testing also doesn't rely on it
59        * it doesn't use the response object in this entire function
60        * and therefore we don't need to spend any time on adding any logic to this response
61        object
62        *
63        * the next function which is called at the end

```

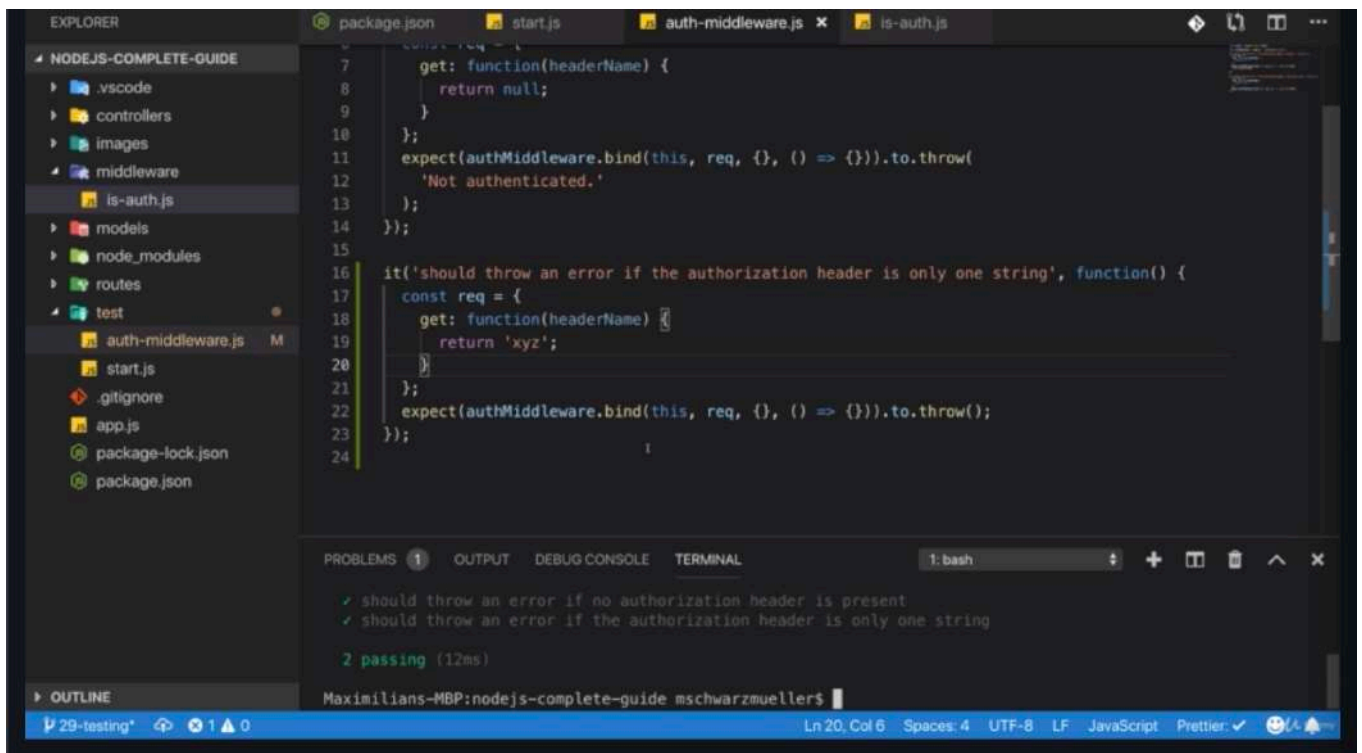
```

63 * we wanna pass it in
64 * but we don't care about what it does
65 * because we are not really executing on next step anyways.
66 * so i will just pass in an empty arrow function
67 * so that it's able to call that without throwing it error
68 * but it doesn't do anything
69 * because it's not when i want a test
70 * i only wanna test behavior when get returns null
71 */
72
73 /**so i expect this function call for a request
74 * that returns null
75 * when we try to get something for given header name.
76 */
77
78 /**i'm calling my middleware function
79 * and this middleware function happens to throw an error
80 * i don't wanna call it myself,
81 * i wanna let my testing framework mocha and chai
82 * so that they can handle the flow and error
83 * instead of calling authMiddleware function directly ourselves,
84 * we instead only pass a reference to this function
85 *
86 * we only wanna bind the arguments we wanna pass in
87 * when our testing set up calls this function
88 * 'bind' first of all requires input for the 'this' keyword
89 * it has 3 arguments
90 * that will be passed into the authMiddleware function once it gets called
91 * we are instead passing a prepared reference to our function
92 * you could say prepared in the sense of where defining
93 * which argument get passed in
94 * so we are passing the prepared reference to expect.
95 *
96 * this test therefore only succeeds if we have an error
97 * but if we are not throwing an error
98 * or if we are instead sending our own error response
99 * and we are not throwing an error,
100 * then this test wouldn't succeed
101 */
102 expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated.');
```

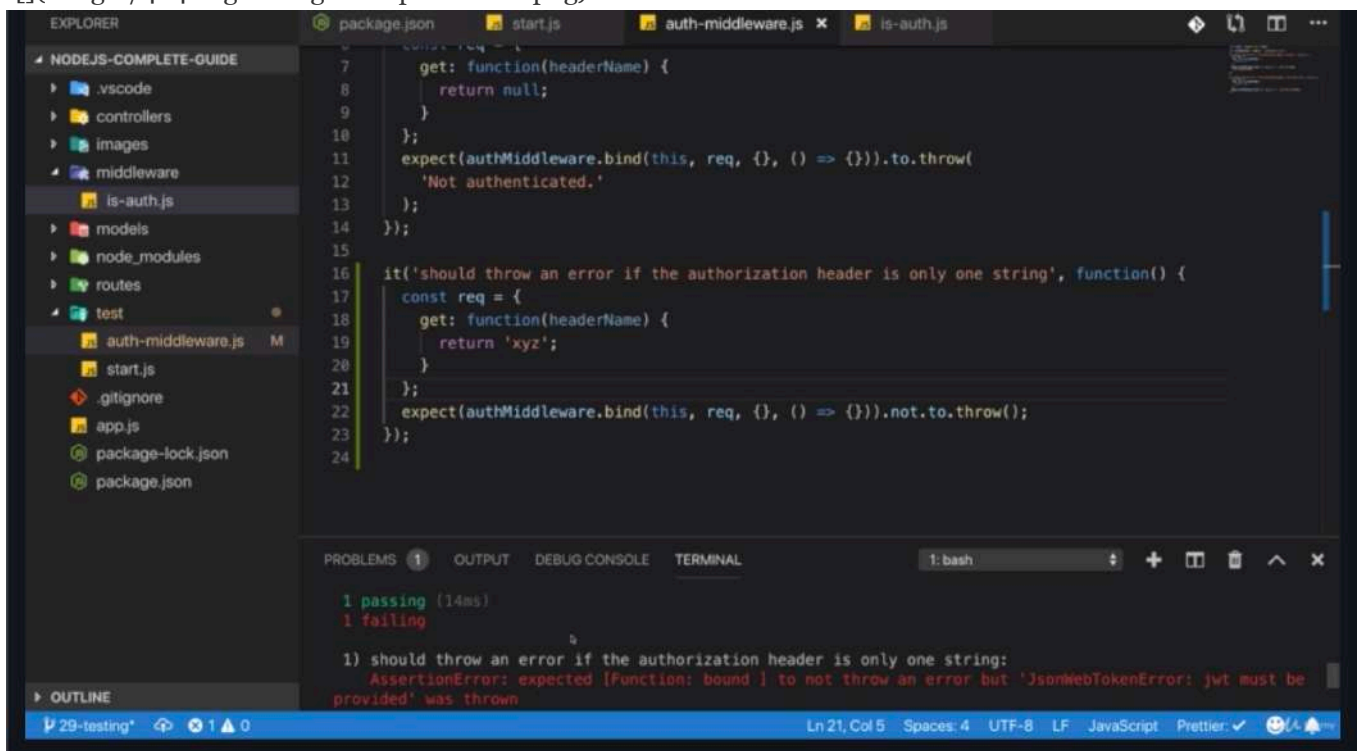
```
103 })
```

* Chapter 464: Organizing Multiple Tests

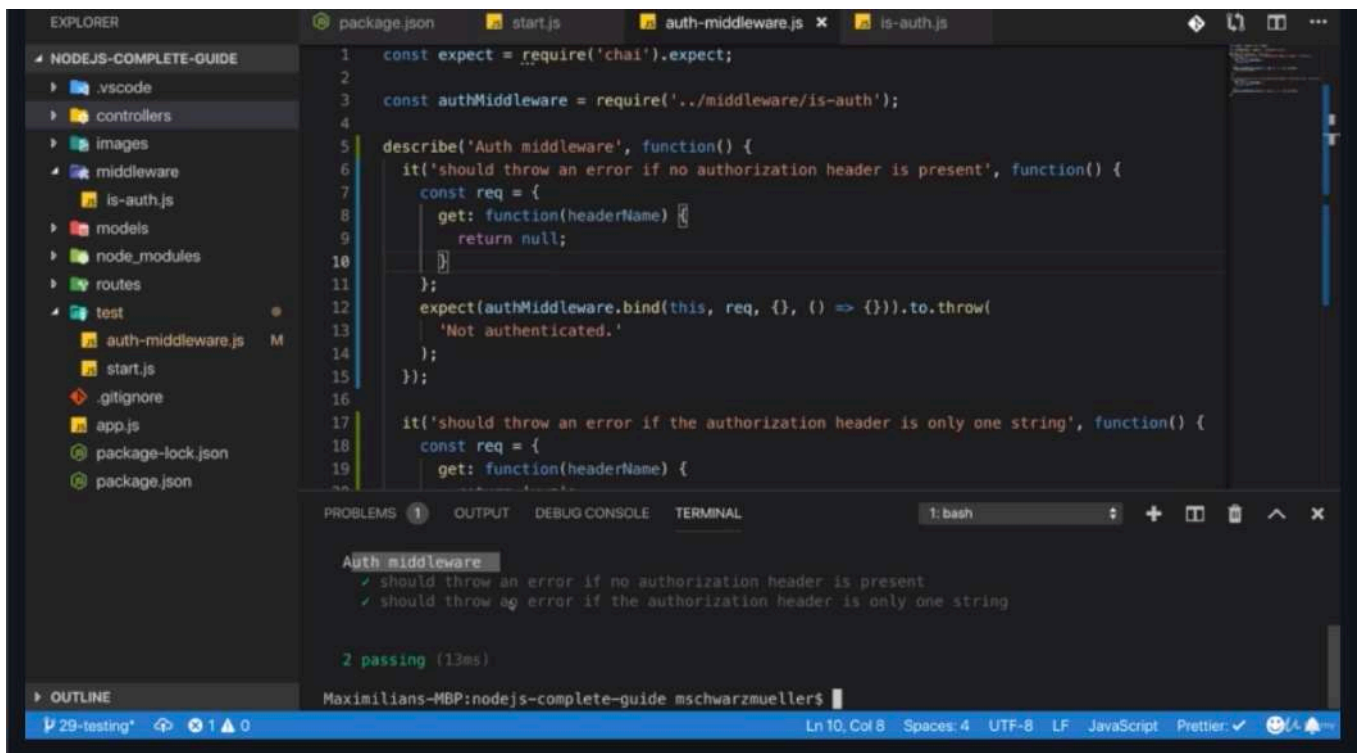
1. update
- ./test/auth-middleware.js



- it succeed because we get an error if we passed this 'xyz' in



- if i check for '.not.to.throw()', then we get error here because it expected that not to throw an error but we got an error here.



- you can group test using 'describe()' function which can be nested inside itself.

```

1  //./test/auth-middleware.js
2
3  const expect = require('chai').expect;
4
5  /**'describe()' function group your tests
6   * and you can nest as many describe function calls
7   * as you want inside of each other
8   * 'describe' also takes a title
9   * and that is not english sentence
10  * but instead like a header for the group you are describing like for example
    authMiddleware
11  *
12  * in 2nd argument,
13  * you pass all your test cases as this function calls are called.
14  */
15  describe('Auth middleware', function(){
16      const authMiddleware = require('../middleware/is-auth');
17      it('should throw an error if no authorization header is present', function(){
18          const req = {
19              get: function(headerName){
20                  return null;
21              }
22          };
23          expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated.');
```

```

24      })
25
26      it('should throw an error if the authorization header is only one string', function(){
27          const req = {
28              get: function(headerName){
29                  return xyz;
30              }
31          };
32          /**you could check for an exact error message
33          * but if you are not sure or

```



```

34      * if you only care about whether an error is frozen at all,
35      * you can also just check for just 'throw()'
36      * without passing any argument to it.
37      */
38      expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
39    })
40  })

```

* Chapter 465: What Not To Test

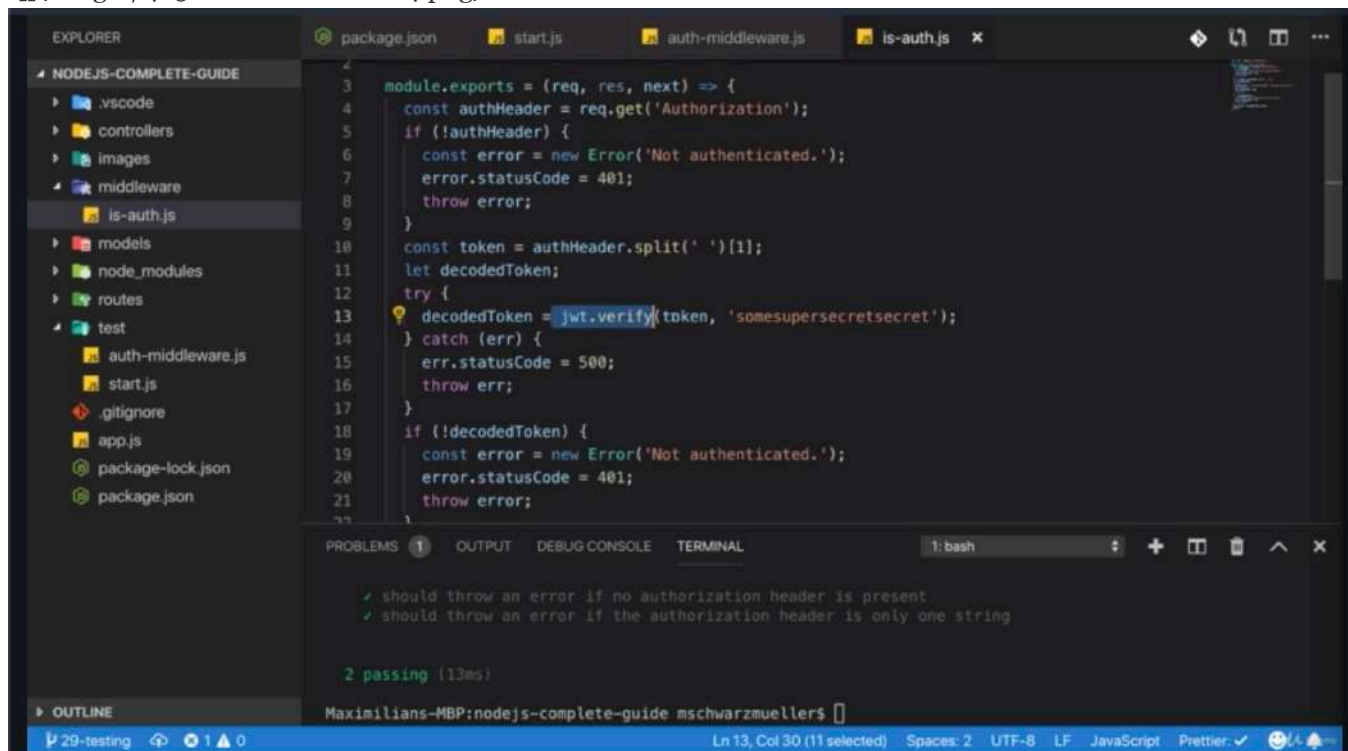
1. update

- ./test/auth-middleware.js

- you should not test whether the verify function works correctly. you should not test whether this verifies a token correctly. because this is not a function or a method owned by you. this is coming from a 3rd party package from the jsonwebtoken package and it's the job of the package offeres to test their own code and to make sure that works correctly. it's not your job.

- we only wanna test if our code behaves correctly when verifications fails for example or when it succeeds. we wanna test if our code then behaves correctly depending what verify does

- this introduce one new problem. verify comes from a 3rd party package and therefore it does its job. if you wanna test if our code works correctly, it's easy to test for failure. we can pass on a token that is not verified by this package because we don't know which tokens it creates for us. tokens are these super long strings. we can't guess them. so whatever we pass into function will probably fail here in the verification step.



EXPLORER

- NODEJS-COMPLETE-GUIDE
 - .vscode
 - controllers
 - images
 - middleware
 - is-auth.js
 - models
 - node_modules
 - routes
 - test
 - auth-middleware.js
 - start.js
 - .gitignore
 - app.js
 - package-lock.json
 - package.json

```
16
17 it('should throw an error if the authorization header is only one string', function() {
18   const req = {
19     get: function(headerName) {
20       return 'xyz';
21     }
22   };
23   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
24 });
25
26 it('should throw an error if the token cannot be verified', function() {
27   const req = {
28     get: function(headerName) {
29       return 'Bearer xyz';
30     }
31   };
32   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
33 });
34
35
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

T: bash

✓ should throw an error if the authorization header is only one string
✓ should throw an error if the token cannot be verified

3 passing (12ms)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

Ln 32, Col 57 Spaces: 4 UTF-8 LF JavaScript Prettier: ✓

EXPLORER

- NODEJS-COMPLETE-GUIDE
 - .vscode
 - controllers
 - images
 - middleware
 - is-auth.js
 - models
 - node_modules
 - routes
 - test
 - auth-middleware.js
 - start.js
 - .gitignore
 - app.js
 - package-lock.json
 - package.json

```
7   error.statusCode = 401;
8   throw error;
9 }
10 const token = authHeader.split(' ')[1];
11 let decodedToken;
12 try {
13   decodedToken = jwt.verify(token, 'somesupersecretsecret');
14 } catch (err) {
15   err.statusCode = 500;
16   throw err;
17 }
18 if (!decodedToken) {
19   const error = new Error('Not authenticated.');
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

T: bash

✓ should throw an error if the authorization header is only one string
✓ should throw an error if the token cannot be verified

3 passing (12ms)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

Ln 22, Col 4 (120 selected) Spaces: 2 UTF-8 LF JavaScript Prettier: ✓

```
26 it('should throw an error if the token cannot be verified', function() {
27   const req = {
28     get: function(headerName) {
29       return 'Bearer xyz';
30     }
31   };
32   expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
33 });
34
35 it('should yield a userId after decoding the token', function() {
36   const req = {
37     get: function(headerName) {
38       return 'Bearer fjdkslaf;d';
39     }
40   };
41   authMiddleware(req, {}, () => {});
42   expect(req).to.have.property('userId');
43 });
44 });
```

3 passing (12ms)
1 failing

1) Auth middleware
should yield a userId after decoding the token:
JsonWebTokenError: jwt malformed
at Object.module.exports [as verify] (node_modules/jsonwebtoken/verify.js:49:17)

- this test failed. it should yield a userId after decoding the token. but what we get is an error. we get that error because our token is malformed because it's too short 'Bearer fjdkslaf;d'. it's not fulfilling the criteria of the JWT verify method and therefore it's filling an error. it's throwing an error that the token is malformed.


```
7   error.statusCode = 401;
8   throw error;
9 }
10 const token = authHeader.split(' ')[1];
11 let decodedToken;
12 try {
13   decodedToken = jwt.verify(token, 'somesupersecretsecret');
14 } catch (err) {
15   err.statusCode = 500;
16   throw err;
17 }
18 if (!decodedToken) {
19   const error = new Error('Not authenticated.');
```

1) Auth middleware
should yield a userId after decoding the token:
JsonWebTokenError: jwt malformed
at Object.module.exports [as verify] (node_modules/jsonwebtoken/verify.js:49:17)
at module.exports (middleware/is-auth.js:13:24)
at Context.<anonymous> (test/auth-middleware.js:41:5)

- we can't always test if it fails but we also wanna test the success case because maybe we have scenarios in our application where it's not failing with an error and still the userId is not getting stored in the request because we don't have that code here. then we would have a bug in our application which we wanna detect with tests and this would not throw an error at any point. and still we wanna detect this error already


```
it('should throw an error if the token cannot be verified', function() {
  const req = {
    get: function(headerName) {
      return 'Bearer xyz';
    }
  };
  expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
});

it('should yield a userId after decoding the token', function() {
  const req = {
    get: function(headerName) {
      return 'Bearer djfkalsdjfaslfjdias';
    }
  };
  authMiddleware(req, {}, () => {});
  expect(req).toHaveProperty('userId');
});
```

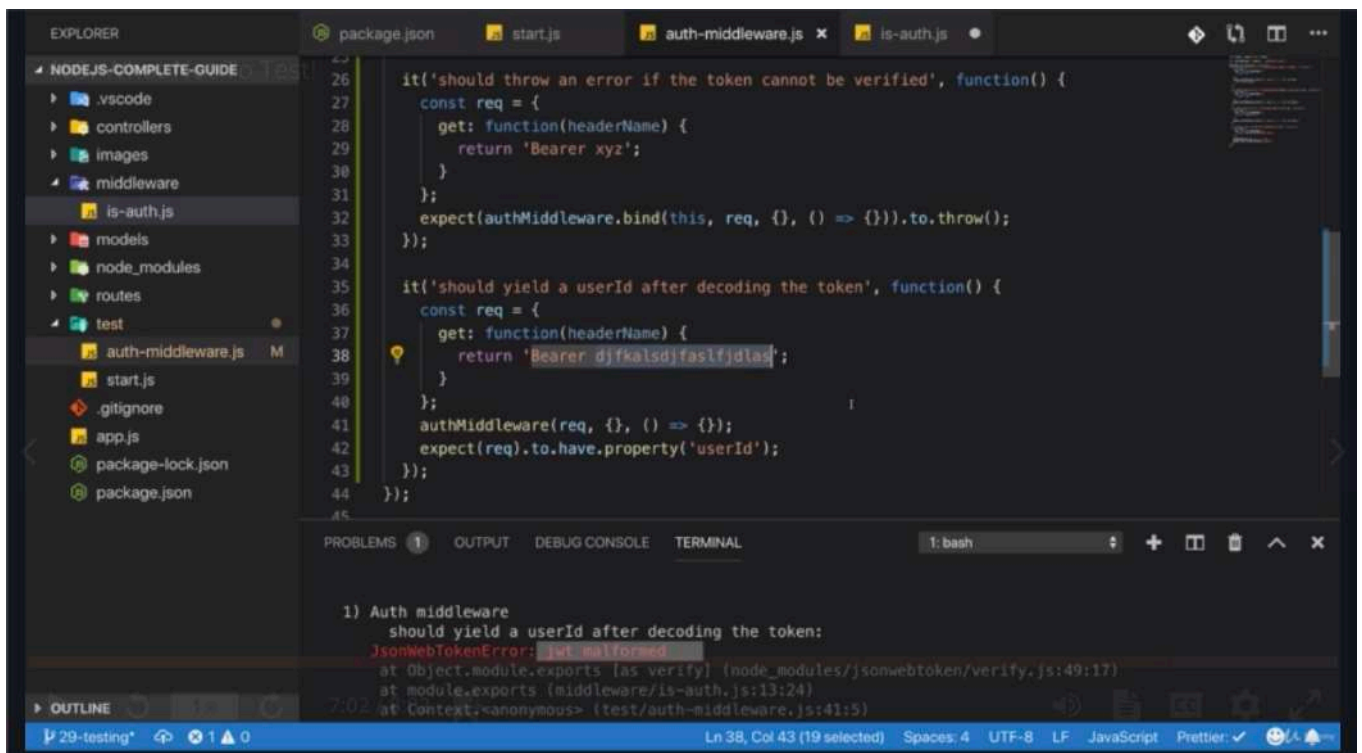
1) Auth middleware
should yield a userId after decoding the token:
JsonWebTokenError: jwt malformed
at Object.module.exports [as verify] (node_modules/jsonwebtoken/verify.js:49:17)
at module.exports (middleware/is-auth.js:13:24)
at Context.<anonymous> (test/auth-middleware.js:41:5)

- therefore test like this one would be super important because we test whether we have a `userId` stored in the request object after running `authMiddleware`


```
error.statusCode = 401;
throw error;
}
const token = authHeader.split(' ')[1];
let decodedToken;
try {
  decodedToken = jwt.verify(token, 'somesupersecretsecret');
} catch (err) {
  err.statusCode = 500;
  throw err;
}
if (!decodedToken) {
  const error = new Error('Not authenticated.');
```

1) Auth middleware
should yield a userId after decoding the token:
JsonWebTokenError: jwt malformed
at Object.module.exports [as verify] (node_modules/jsonwebtoken/verify.js:49:17)
at module.exports (middleware/is-auth.js:13:24)
at Context.<anonymous> (test/auth-middleware.js:41:5)

- so to find that, we need some way of shutting down that verify method.



- we know that this is not a valid token but for this test we don't care. we wanna test scenario where we have a valid token. we know that this would not be a valid token but this is not important to us here. we don't need to test whether it is really fails for a random token here.

- we wanna succeed for a random token because we then wanna test something different. we wanna test if our app works correctly for valid tokens no matter if this really is a valid token here or not.

- how can we shut down verify and make sure it simply gives us an object with a userId so that we can pull that userId from that object?

```

1  //./test/auth-middleware.js
2
3  const expect = require('chai').expect;
4
5  describe('Auth middleware', function(){
6      const authMiddleware = require('../middleware/is-auth');
7      it('should throw an error if no authorization header is present', function(){
8          const req = {
9              get: function(headerName){
10                  return null;
11              }
12          };
13          expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated.');
```

```

14      })
15
16      it('should throw an error if the authorization header is only one string', function(){
17          const req = {
18              get: function(headerName){
19                  return xyz;
20              }
21          };
22          expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
23      })
24
25      it('should throw an error if the token cannot be verified', function(){
26          const req = {
27              get: function(headerName){
```

```

28     /**'xyz' will be incorrect token, not a token by the JWT package
29     */
30     return 'Bearer xyz';
31 }
32 };
33 expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
34 })
35 /**if we add another test where we wanna check
36  * if this is a valid token,
37  * then decodedToken should have a user id
38  */
39 it('should yield a userId after decoding the token', function(){
40     const req = {
41         get: function(headerName){
42             return 'Bearer fjdkslaf;d';
43         }
44     };
45     authMiddleware(req, {}, () => {});
46     /**we now expect our request object to have a new property
47     * because we add a new property in the middleware userId property
48     *
49     * this is something we could expect for a valid token
50     * because if the token is valid
51     * if it is verified,
52     * then we we make it pass 'try and catch' block
53     * then it checks if it is defined
54     * then it checks again if it really is defined
55     * and we expect that to be defined
56     * and then we get the userId from the decodedToken
57     * and we store it in request
58     * so expecting that userId property to exist on the request seems vaild
59     *
60     */
61     expect(req).to.have.property('userId')
62 })
63 })

```

* Chapter 466: Using Stubs


```
7   error.statusCode = 401;
8   throw error;
9 }
10 const token = authHeader.split(' ')[1];
11 let decodedToken;
12 try {
13   decodedToken = jwt.verify(token, 'somesupersecretsecret');
14 } catch (err) {
15   err.statusCode = 500;
16   throw err;
17 }
18 if (!decodedToken) {
19   const error = new Error('Not authenticated.');
```

1) Auth middleware should yield a userId after decoding the token:
JsonWebTokenError: jwt malformed
at Object.module.exports [as verify] (node_modules/jsonwebtoken/verify.js:49:17)
at module.exports (middleware/is-auth.js:13:24)
at Context.<anonymous> (test/auth-middleware.js:41:5)

```
32   };
33   expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
34 });
35
36 it('should yield a userId after decoding the token', function() {
37   const req = {
38     get: function(headerName) {
39       return 'Bearer djfkalsdjfaslfdlas';
40     }
41   };
42   jwt.verify = function() {
43     return { userId: 'abc' };
44   };
45   authMiddleware(req, {}, () => {});
46   expect(req).to.have.property('userId');
47 });
48 });
49
```

npm ERR! Test failed. See above for more details.
Maximilians-MBP:nodejs-complete-guide mschwarzmueller\$ npm test

> nodejs-complete-guide@1.0.0 test /Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide
> mocha


```
32   };
33   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
34   });
35
36   it('should yield a userId after decoding the token', function() {
37     const req = {
38       get: function(headerName) {
39         return 'Bearer djfkalsdjfaslfjdls';
40       }
41     };
42     jwt.verify = function() {
43       return { userId: 'abc' };
44     };
45     authMiddleware(req, {}, () => {});
46     expect(req).toHaveProperty('userId');
47   });
48   });
49
```

1) Auth middleware
should yield a userId after decoding the token:
AssertionError: expected { get: [Function: get] } to have property 'userId'
at Context.<anonymous> (test/auth-middleware.js:46:25)

- to solve that, we can use marks or stubs which means we replace this verify method with a simpler method.
 - 'decodedToken = jwt.verify(token, 'somesupersecretsecret')' we are overwriting the actual verify method that is package has. the way module imports work in node.js. if we overwrite it here, this will be the case in the middleware when it runs to. because we have one global package.
 - so we overwrite with our own function. so if i run npm test, our own function gets executed and that will return a userId in that object that gives us and therefore it will first of all nont throw an error and it also gives us a way of pulling out our userId.
 - if i npm test, this will still fail. but now it fails because we expected to have a property userId but we don't. we only have an object with a get method
-

```
32   };
33   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
34   });
35
36   it('should yield a userId after decoding the token', function() {
37     const req = {
38       get: function(headerName) {
39         return 'Bearer djfkalsdjfaslfjdls';
40       }
41     };
42     jwt.verify = function() {
43       return { userId: 'abc' };
44     };
45     authMiddleware(req, {}, () => {});
46     expect(req).toHaveProperty('userId');
47   });
48   });
49
```

1) Auth middleware
should yield a userId after decoding the token:
AssertionError: expected { get: [Function: get] } to have property 'userId'
at Context.<anonymous> (test/auth-middleware.js:46:25)

- because we only have this object. we don't have a userId
-
-

The screenshot shows the VS Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `auth-middleware.js` file. The code is as follows:

```
7     error.statusCode = 401;
8     throw error;
9   }
10   const token = authHeader.split(' ')[1];
11   let decodedToken;
12   try {
13     decodedToken = jwt.verify(token, 'somesupersecretsecret');
14   } catch (err) {
15     err.statusCode = 500;
16     throw err;
17   }
18   if (!decodedToken) {
19     const error = new Error('Not authenticated.');
```

The test runner output at the bottom shows the following error:

```
1) Auth middleware
   should yield a userId after decoding the token:
   AssertionError: expected [get: [Function: get]] to have property 'userId'
   at Context.<anonymous> (test/auth-middleware.js:46:25)
```

The screenshot shows the same VS Code editor window with the `auth-middleware.js` file. The code is identical to the first screenshot. The test runner output at the bottom shows the same error:

```
1) Auth middleware
   should yield a userId after decoding the token:
   AssertionError: expected [get: [Function: get]] to have property 'userId'
   at Context.<anonymous> (test/auth-middleware.js:46:25)
```

```
7      error.statusCode = 401;
8      throw error;
9    }
10    const token = authHeader.split(' ')[1];
11    let decodedToken;
12    try {
13      decodedToken = jwt.verify(token, 'somesupersecretsecret');
14    } catch (err) {
15      err.statusCode = 500;
16      throw err;
17    }
18    if (!decodedToken) {
19      const error = new Error('Not authenticated.');
```

```
npm ERR! Test failed. See above for more details.
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm test
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
```

```
7      error.statusCode = 401;
8      throw error;
9    }
10    const token = authHeader.split(' ')[1];
11    let decodedToken;
12    try {
13      decodedToken = jwt.verify(token, 'somesupersecretsecret');
```

```
✓ should throw an error if the token cannot be verified
✓ should yield a userId after decoding the token

4 passing (13ms)
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
```

- and we can look into our code and see we should add this code again to make sure we stored at userId in the request object.

- and if we npm test, then we have 4 passing tests because we replace the built-in verify method and that is common way of handling such cases.

- however instead of manually overwriting like this, there is a more elegant way.

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 structure with folders like .vscode, controllers, images, middleware, and files like is-auth.js, models, node_modules, routes, test, auth-middleware.js, start.js, .gitignore, app.js, package-lock.json, and package.json. The main editor window displays the content of auth-middleware.js, which includes a Jest test suite. The test suite has two assertions: 'should throw an error if the token cannot be verified' and 'should yield a user ID after decoding the token'. The terminal output shows the test results: '4 passing (13ms)'. The status bar at the bottom indicates the file is at line 47, column 6, with 335 characters selected. The status bar also shows the file encoding as UTF-8, line endings as LF, and the language as JavaScript.

```
32   };
33   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
34   });
35
36   it('should yield a user ID after decoding the token', function() {
37     const req = {
38       get: function(headerName) {
39         return 'Bearer djfkalsdjfaslfjdias';
40       }
41     };
42     jwt.verify = function() {
43       return { userId: 'abc' };
44     };
45     authMiddleware(req, {}, () => {});
46     expect(req).toHaveProperty('userId');
47   });
48 });
49
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

✓ should throw an error if the token cannot be verified
✓ should yield a user ID after decoding the token

4 passing (13ms)

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

Ln 47, Col 6 (335 selected) Spaces: 4 UTF-8 LF JavaScript Prettier ✓

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 structure with folders like .vscode, controllers, images, middleware, and files like is-auth.js, models, node_modules, routes, test, auth-middleware.js, start.js, .gitignore, app.js, package-lock.json, and package.json. The main editor window displays the content of auth-middleware.js, which includes a Jest test suite. The test suite has two assertions: 'should throw an error if the token cannot be verified' and 'should yield a user ID after decoding the token'. The terminal output shows the test results: '4 passing (13ms)'. The status bar at the bottom indicates the file is at line 36, column 3, with 4 characters selected. The status bar also shows the file encoding as UTF-8, line endings as LF, and the language as JavaScript.

```
32   };
33   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
34   });
35
36   it('should yield a user ID after decoding the token', function() {
37     const req = {
38       get: function(headerName) {
39         return 'Bearer djfkalsdjfaslfjdias';
40       }
41     };
42     jwt.verify = function() {
43       return { userId: 'abc' };
44     };
45     authMiddleware(req, {}, () => {});
46     expect(req).toHaveProperty('userId');
47   });
48 });
49
```

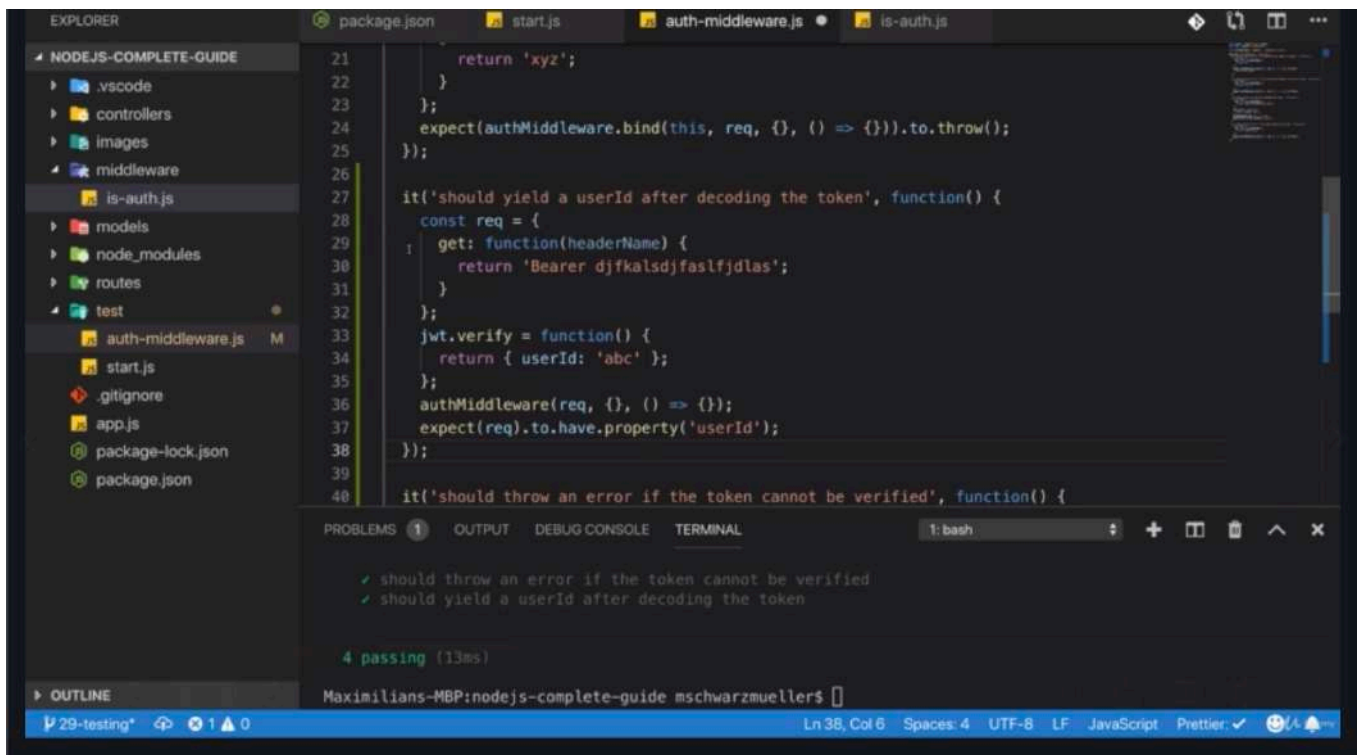
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

✓ should throw an error if the token cannot be verified
✓ should yield a user ID after decoding the token

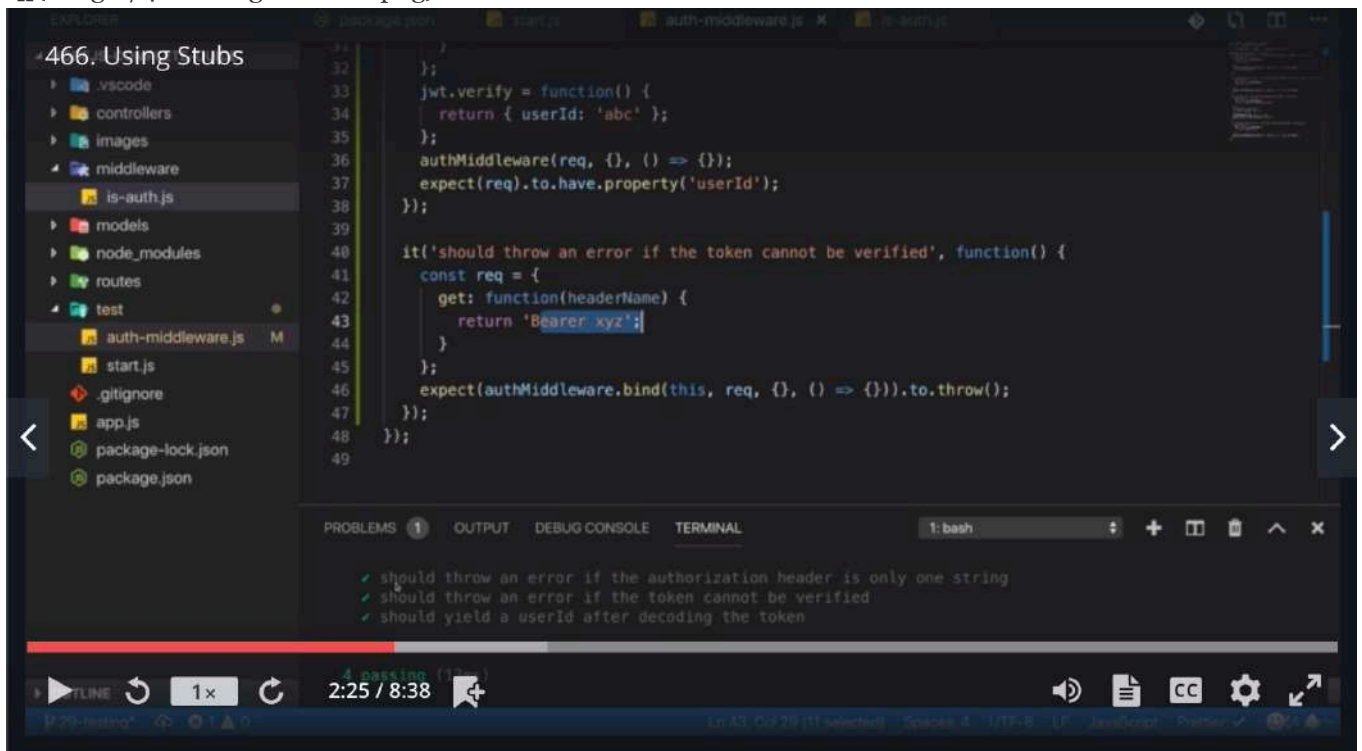
4 passing (13ms)

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

Ln 36, Col 3 Spaces: 4 UTF-8 LF JavaScript Prettier ✓



- because this has a huge downside which becomes evident if i cut this test, i put it in front of the test, we had before that 'should throw an error if the token cannot be verified',



- that is the test where it should throw an error if we have an invalid token. 'Bearer xyz'

- note that before all test passed, so we get an error for an invalid token.


```
27 it('should yield a userId after decoding the token', function() {
28   const req = {
29     get: function(headerName) {
30       return 'Bearer djfkalsdjfaslfjdldas';
31     }
32   };
33   jwt.verify = function() {
34     return { userId: 'abc' };
35   };
36   authMiddleware(req, {}, () => {});
37   expect(req).to.have.property('userId');
38 });
39
40 it('should throw an error if the token cannot be verified', function() {
41   const req = {
42     get: function(headerName) {
43       return 'Bearer xyz';
44     }
45   };
46   expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
47 });
```

3 passing (17ms)
1 failing

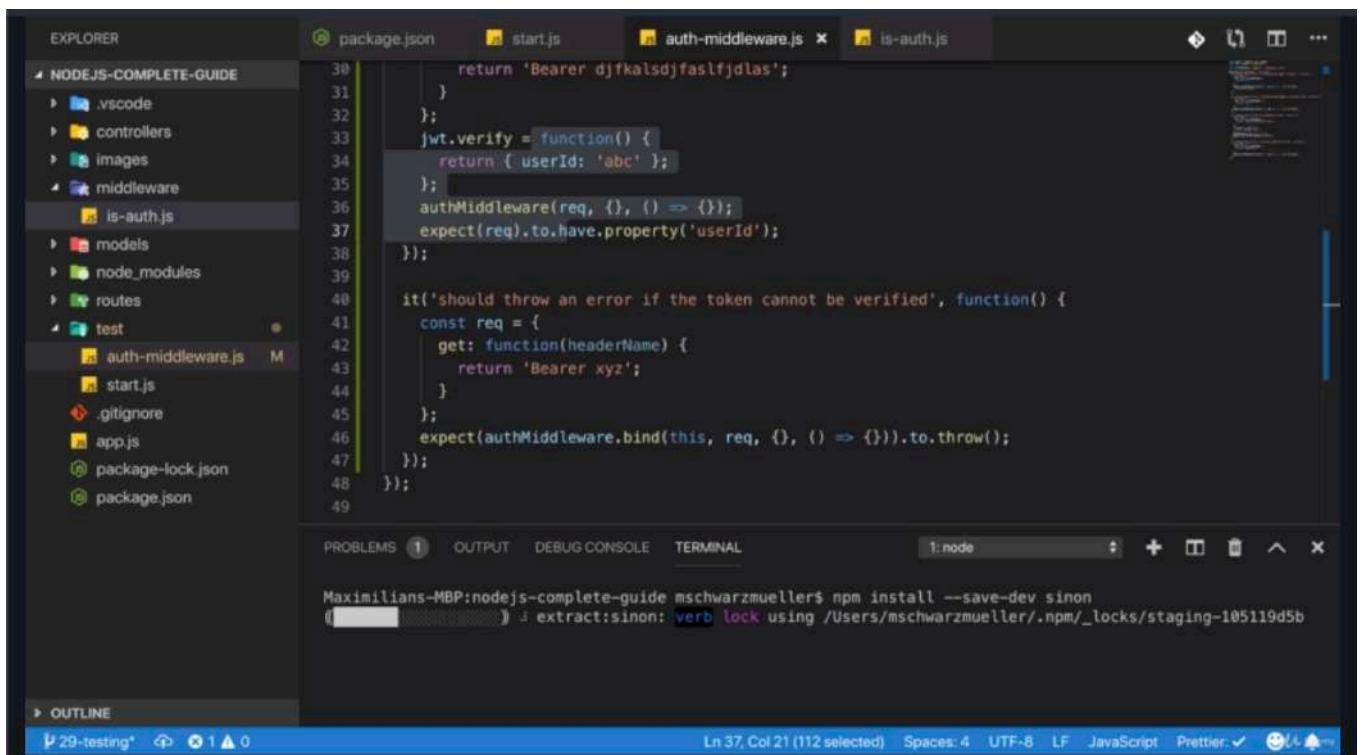
1) Auth middleware
should throw an error if the token cannot be verified:
AssertionError: expected [Function: bound] to throw an error

```
31   }
32   };
33   jwt.verify = function() {
34     return { userId: 'abc' };
35   };
36   authMiddleware(req, {}, () => {});
37   expect(req).to.have.property('userId');
38 });
39
40 it('should throw an error if the token cannot be verified', function() {
41   const req = {
42     get: function(headerName) {
43       return 'Bearer xyz';
44     }
45   };
46   expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
47 });
48 });
49
```

3 passing (17ms)
1 failing

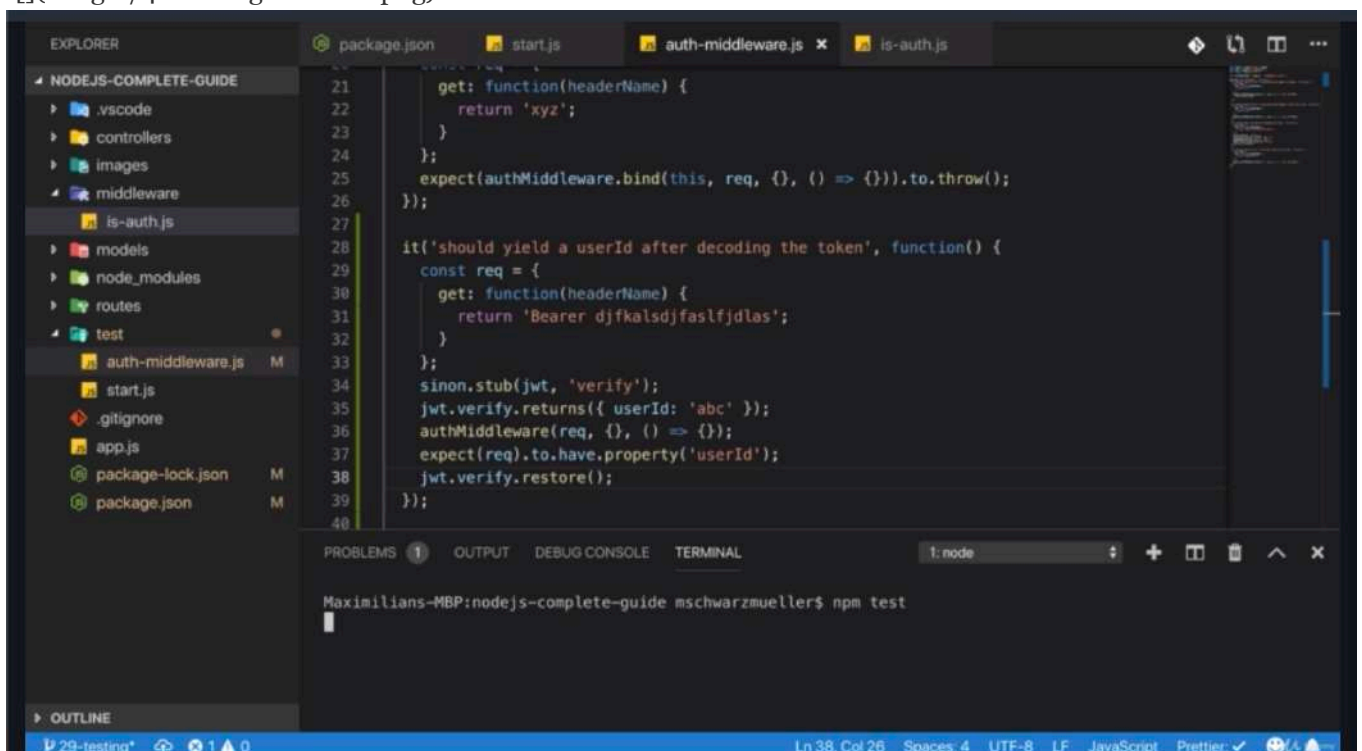
1) Auth middleware
should throw an error if the token cannot be verified:
AssertionError: expected [Function: bound] to throw an error

- here we are then overwriting verify to never throw an error. so now after i switched the order of tests, i have a problem with my last test



- because this doesn't throw an error anymore. because in this test, i globally replaced the verify method. that is not ideal. it's good for this test to succeed but it means that if i have any other test that needs the original verify method. it has no chance of getting that. because we replaced it here

- therefore instead of manually stopping or mocking functionalities and replacing them, it's good to use packages that allow you to restore the original setup.



- for that will install extra package. 'npm install --save-dev sinon' which is the package that allows us to create a so-called stub which is replacement for the original function where we can easily restore the original function.

The screenshot shows the VS Code editor with the file `auth-middleware.js` open. The code defines a `get` function that returns a token and a test suite. The test suite includes a `const req = {}` and a `get` function that returns a specific token. The test suite has two tests: one that expects the token to have a `userId` property and another that expects an error to be thrown if the token cannot be verified. The terminal shows the output of the test run: `4 passing (10ms)`.

```
21     get: function(headerName) {
22       return 'xyz';
23     }
24   };
25   expect(authMiddleware.bind(this, req, {}, () => {})).toThrow();
26 });
27
28 it('should yield a userId after decoding the token', function() {
29   const req = {
30     get: function(headerName) {
31       return 'Bearer djfkalsdjfaslfjdldas';
32     }
33   };
34   sinon.stub(jwt, 'verify');
35   jwt.verify.returns({ userId: 'abc' });
36   authMiddleware(req, {}, () => {});
37   expect(req).toHaveProperty('userId');
38   jwt.verify.restore();
39 });
40
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

✓ should yield a userId after decoding the token
✓ should throw an error if the token cannot be verified

4 passing (10ms)

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

- 4 tests passing. because if i run npm test again, all tests pass again because i restore the original function after this test where i needed a different behavior.

The screenshot shows the VS Code editor with the file `auth-middleware.js` open. The code defines a `get` function that returns a token and a test suite. The test suite includes a `const req = {}` and a `get` function that returns a specific token. The test suite has two tests: one that expects the token to have a `userId` property and another that expects an error to be thrown if the token cannot be verified. The terminal shows the output of the test run: `AssertionError: expected false to be true`.

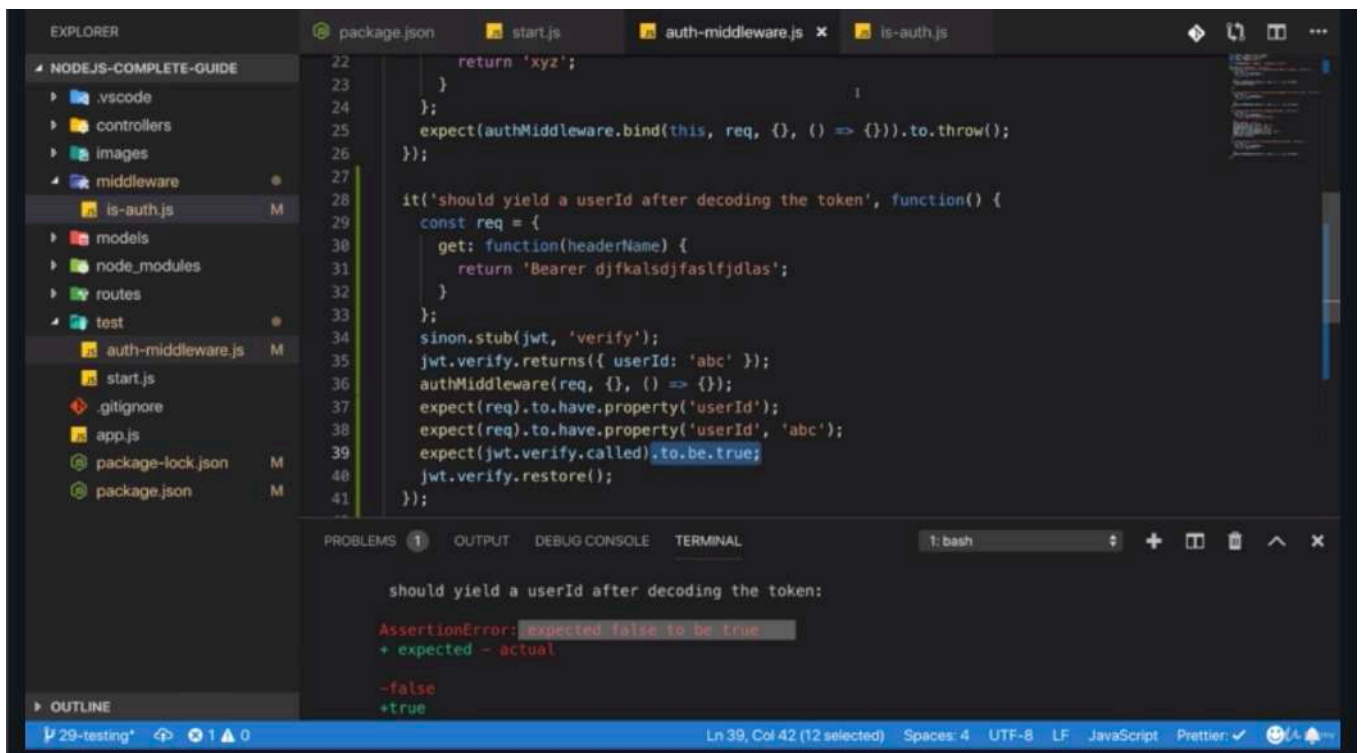
```
7     error.statusCode = 401;
8     throw error;
9   }
10   const token = authHeader.split(' ')[1];
11   let decodedToken;
12   try {
13     // decodedToken = jwt.verify(token, 'somesupersecretsecret');
14     decodedToken = { userId: 'abc' };
15   } catch (err) {
16     err.statusCode = 500;
17     throw err;
18   }
19   if (!decodedToken) {
20     const error = new Error('Not authenticated. ');
21     error.statusCode = 401;
22     throw error;
23   }
24   req.userId = decodedToken.userId;
25   next();
26 };

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

should yield a userId after decoding the token:

AssertionError: expected false to be true
+ expected - actual
-false
+true



- if so, our average test would still succeed. but we get an error because we expect that `false(jwt.verify.called` is comment now) to be true.

```

1  //./test/auth-middleware.js
2
3  const expect = require('chai').expect;
4  const jwt = require('jsonwebtoken');
5  const sinon = require('sinon');
6
7  describe('Auth middleware', function(){
8      const authMiddleware = require('../middleware/is-auth');
9      it('should throw an error if no authorization header is present', function(){
10         const req = {
11             get: function(headerName){
12                 return null;
13             }
14         };
15         expect(authMiddleware.bind(this, req, {}, () => {})).to.throw('Not authenticated.');
```

```

16     })
17
18     it('should throw an error if the authorization header is only one string', function(){
19         const req = {
20             get: function(headerName){
21                 return xyz;
22             }
23         };
24         expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
25     })
26
27     it('should yield a userId after decoding the token', function(){
28         const req = {
29             get: function(headerName){
30                 return 'Bearer fjdkslaf;d';
31             }
32         };
33         /**instead of manually replacing it like this,
```

```

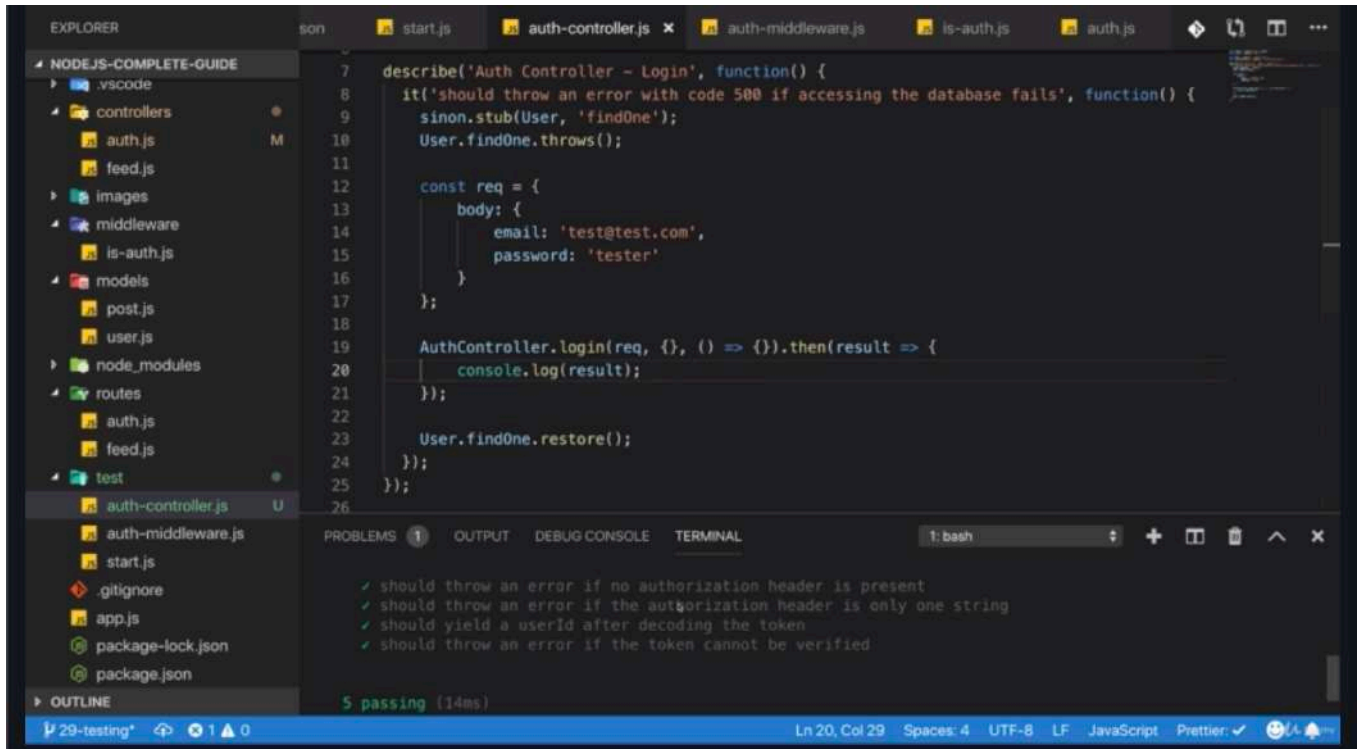
34     * i call sinon.stub()
35     * and i pass in the object
36     * where i have the method i wanna replace.
37     * that is jwt
38     * and 2nd argument is passing the object
39     * which has the method i wanna replace
40     * and then as a string the method name 'verify'
41     *
42     * sinon will replace that
43     * by default, it replace it with an empty function
44     * that doesn't do anything special though that's not entirely true
45     * it will do with things like registering function calls and so on.
46     * so you can test for things like
47     * has this function be called no matter what it executes
48     */
49     sinon.stub(jwt, 'verify')
50     /**'returns()' is by sinon
51     * that allows us to configure what this function should return
52     *
53     * this stub also register things like function calls
54     */
55     jwt.verify.returns({userId: 'abc'})
56     authMiddleware(req, {}, () => {});
57     expect(req).to.have.property('userId')
58     /**for completeness sake,
59     * we can test if request has a property userId with certain value
60     * that's optional 2nd argument you can pass to that property method
61     * so we wanna make sure the value 'abc'
62     * that's a redundant test
63     * because we defined the value down there.
64     */
65     expect(req).to.have.property('userId', 'abc')
66     /**this stub also register things like function calls
67     * verify method has been called in our authMiddleware
68     * and
69     */
70     expect(jwt.verify.called).to.be.true;
71     /**after checking our expectation,
72     * we can call jwt.verify.restore()
73     * which will restore the original function
74     * that is the big difference to our own stub
75     * where we replace as on our own.
76     */
77     jwt.verify.restore();
78 })
79
80 it('should throw an error if the token cannot be verified', function(){
81     const req = {
82         get: function(headerName){
83             /**'xyz' will be incorrect token, not a token by the JWT package
84             */
85             return 'Bearer xyz';
86         }
87     };
88     expect(authMiddleware.bind(this, req, {}, () => {})).to.throw();
89 })

```

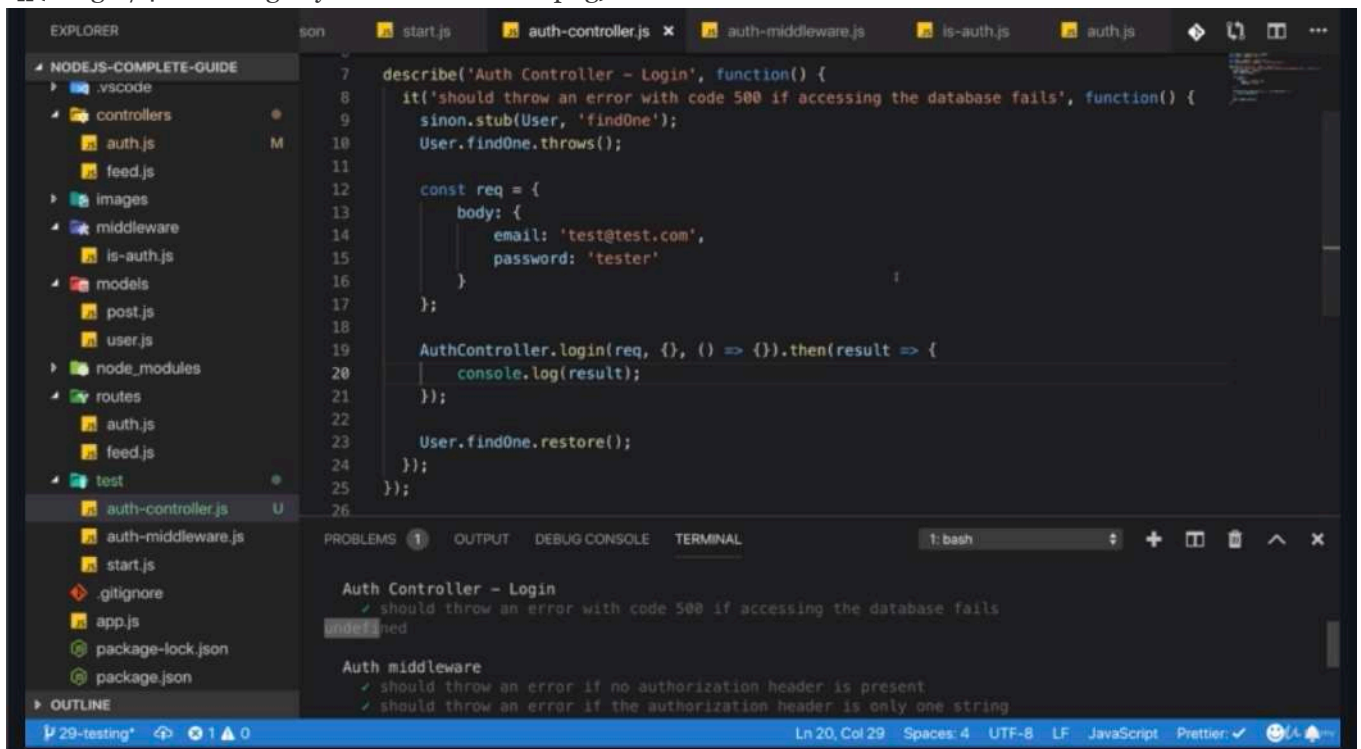
* Chapter 468: Testing Asynchronous Code

1. update

- ./routes/auth.js
- ./controllers/auth.js
- ./test/auth-controller.js



- we can see all pass which is strange



- but if we scroll up a bit, you can see 'undefined' here.


```
56     email: loadedUser.email,
57     userId: loadedUser._id.toString()
58   },
59   'somesupersecretsecret',
60   { expiresIn: '1h' }
61 );
62 res.status(200).json({ token: token, userId: loadedUser._id.toString() });
63 } catch (err) {
64   if (!err.statusCode) {
65     err.statusCode = 500;
66   }
67   next(err);
68 }
69 return;
70 };
71
72 exports.getUserStatus = async (req, res, next) => {
73   try {
74     const user = await User.findById(req.userId);
75     if (!user) {
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

1: bash

Auth Controller - Login

- ✓ should throw an error with code 500 if accessing the database fails

Auth middleware

- ✓ should throw an error if no authorization header is present
- ✓ should throw an error if the authorization header is only one string

Ln 69, Col 10 Spaces: 2 UTF-8 LF JavaScript Prettier: ✓

- because go back to ./controllers/auth.js file, i return undefined and that is what i return as a value of that promise that gets returned.


```
56     email: loadedUser.email,
57     userId: loadedUser._id.toString()
58   },
59   'somesupersecretsecret',
60   { expiresIn: '1h' }
61 );
62 res.status(200).json({ token: token, userId: loadedUser._id.toString() });
63 return;
64 } catch (err) {
65   if (!err.statusCode) {
66     err.statusCode = 500;
67   }
68   next(err);
69   return err;
70 }
71 };
72
73 exports.getUserStatus = async (req, res, next) => {
74   try {
75     const user = await User.findById(req.userId);
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

1: bash

Auth Controller - Login

- ✓ should yield a userId after decoding the token
- ✓ should throw an error if the token cannot be verified

Auth middleware

- ✓ should throw an error if no authorization header is present
- ✓ should throw an error if the authorization header is only one string

5 passing (15ms)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

Ln 69, Col 16 (12 selected) Spaces: 2 UTF-8 LF JavaScript Prettier: ✓


```
56     email: loadedUser.email,
57     userId: loadedUser._id.toString()
58   },
59   'somesupersecretsecret',
60   { expiresIn: '1h' }
61 );
62 res.status(200).json({ token: token, userId: loadedUser._id.toString() });
63 return;
64 } catch (err) {
65   if (!err.statusCode) {
66     err.statusCode = 500;
67   }
68   next(err);
69   return err;
70 }
71 };
72
73 exports.getUserStatus = async (req, res, next) => {
74   try {
75     const user = await User.findById(req.userId);
```

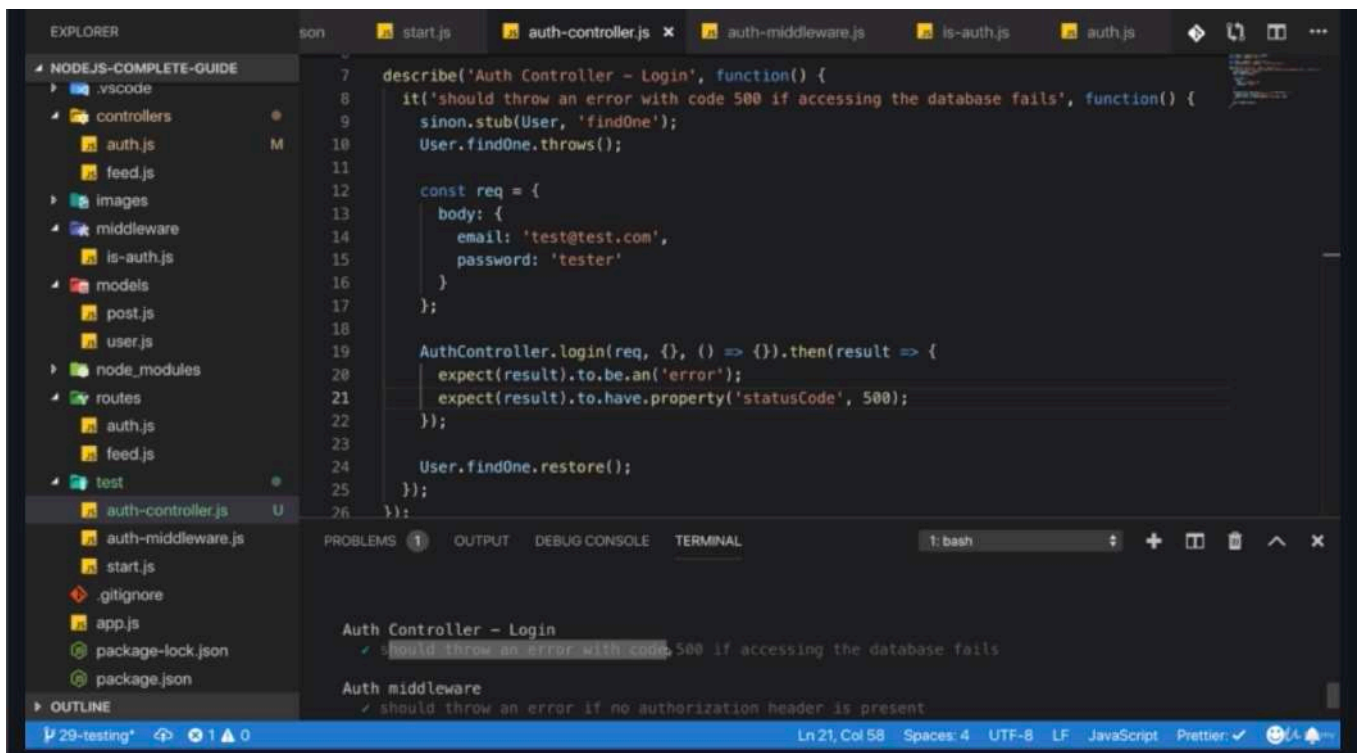
Auth Controller - Login
✓ should throw an error with code 500 if accessing the database fails
{ Error: Error
 at Object.fake.exceptionCreator (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide

- now if i rerun npm test, we see that error object being log up.

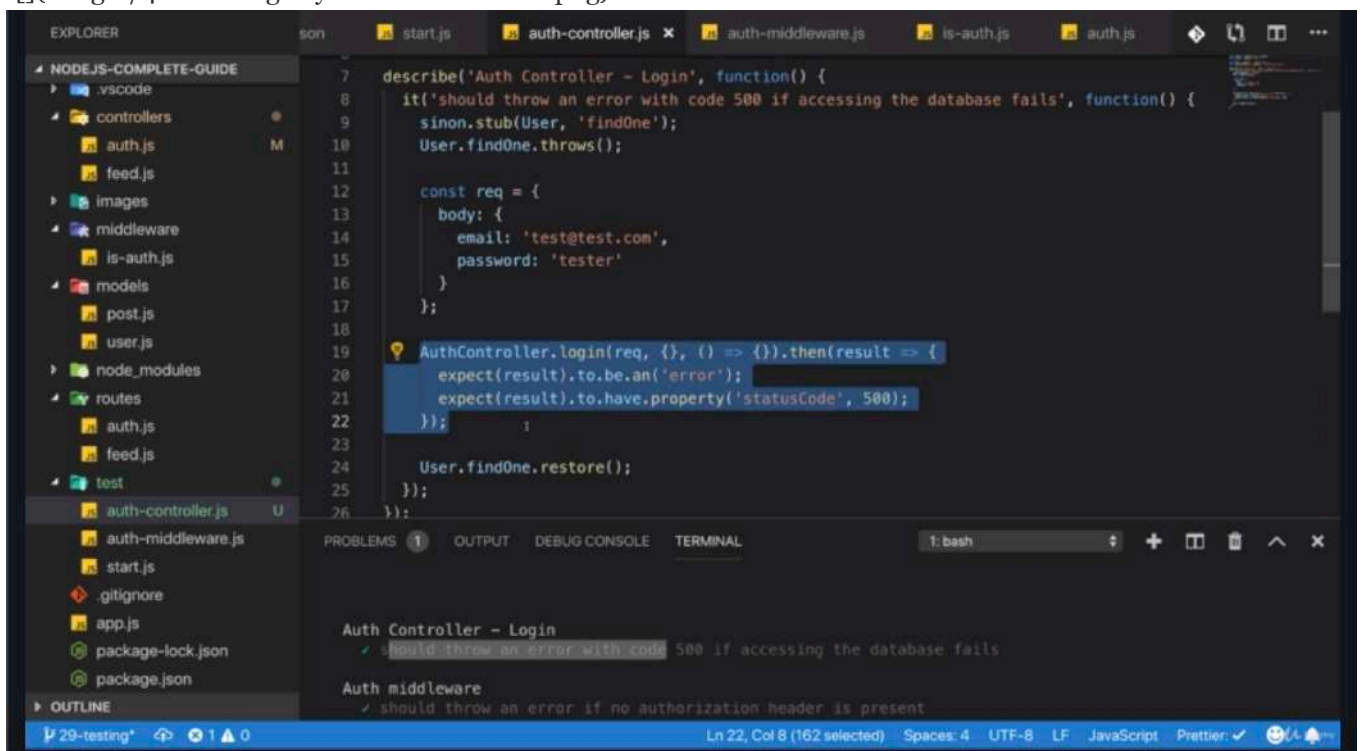

```
7 describe('Auth Controller - Login', function() {
8   it('should throw an error with code 500 if accessing the database fails', function() {
9     sinon.stub(User, 'findOne');
10    User.findOne.throws();
11
12    const req = {
13      body: {
14        email: 'test@test.com',
15        password: 'tester'
16      }
17    };
18
19    AuthController.login(req, {}, () => {}).then(result => {
20      expect(result).toBe.an('error');
21      expect(result).to.have.property('statusCode', 500);
22    });
23
24    User.findOne.restore();
25  });
26 });
```

✓ should yield a userId after decoding the token
✓ should throw an error if the token cannot be verified
5 passing (11ms)

- this looks good everything passes.

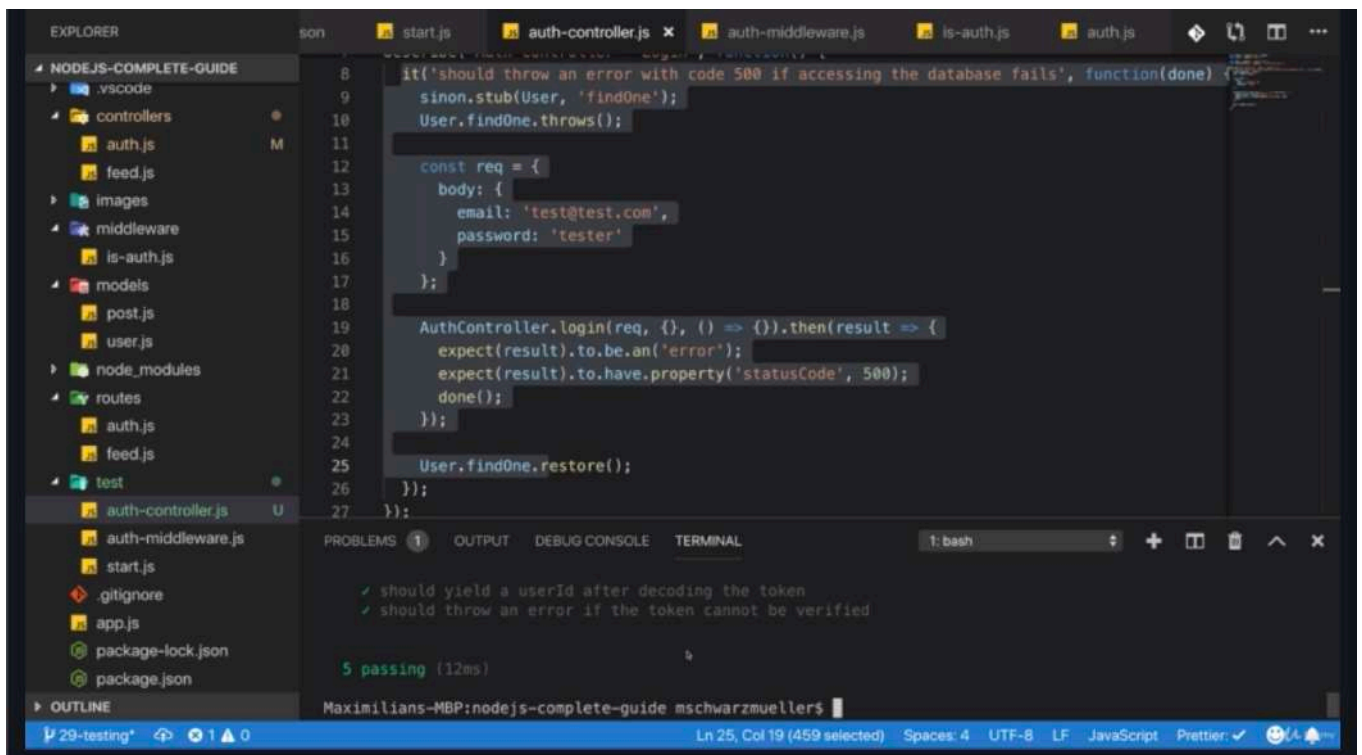


- but this is false pass here.



- it passes because mocha doesn't wait for this test case to finish because we have async code. and by default it doesn't wait for that async code to resolve. it doesn't wait for this promise to resolve no matter how fast is this.

- we can tell mocha to wait and we do this by adding extra argument in this function. we pass to it and that's the 'done' argument.



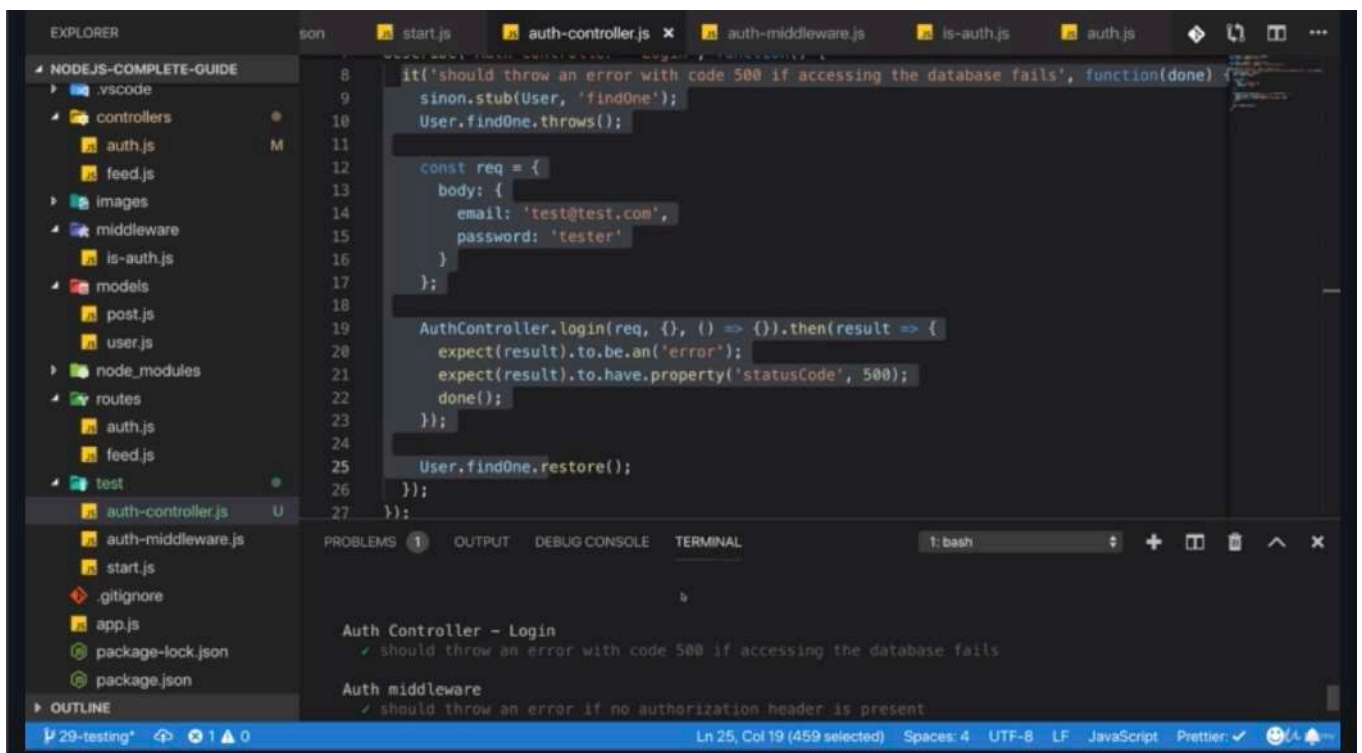
```
it('should throw an error with code 500 if accessing the database fails', function(done) {
  sinon.stub(User, 'findOne');
  User.findOne.throws();

  const req = {
    body: {
      email: 'test@test.com',
      password: 'tester'
    }
  };

  AuthController.login(req, {}, () => {}).then(result => {
    expect(result).toBe.an('error');
    expect(result).toHave.property('statusCode', 500);
    done();
  });

  User.findOne.restore();
});
```

5 passing (12ms)



```
Auth Controller - Login
  ✓ should throw an error with code 500 if accessing the database fails

Auth middleware
  ✓ should throw an error if no authorization header is present
```

- now it passes again but now this is a valid test and we can confirm this by changing this expected statusCode to 401


```
8 it('should throw an error with code 500 if accessing the database fails', function(done)
9   sinon.stub(User, 'findOne');
10   User.findOne.throws();
11
12   const req = {
13     body: {
14       email: 'test@test.com',
15       password: 'tester'
16     }
17   };
18
19   AuthController.login(req, {}, () => {}).then(result => {
20     expect(result).toBe.an('error');
21     expect(result).to.have.property('statusCode', 401);
22     done();
23   });
24
25   User.findOne.restore();
26 });
27 });
```

1) Auth Controller - Login
should throw an error with code 500 if accessing the database fails:
Error: Timeout of 2000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves. (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/test/auth-controller.js)

```
8 it('should throw an error with code 500 if accessing the database fails', function(done)
9   sinon.stub(User, 'findOne');
10   User.findOne.throws();
11
12   const req = {
13     body: {
14       email: 'test@test.com',
15       password: 'tester'
16     }
17   };
18
19   AuthController.login(req, {}, () => {}).then(result => {
20     expect(result).toBe.an('error');
21     expect(result).to.have.property('statusCode', 401);
22     done();
23   });
24
25   User.findOne.restore();
26 });
27 });
```

Auth Controller - Login
1) should throw an error with code 500 if accessing the database fails
(node:48766) UnhandledPromiseRejectionWarning: AssertionError: expected [Error: Error] to have property 'statusCode' of 401, but got 500
at AuthController.login.then.result (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/test/auth-controller.js:21:30)

- if i now run npm test, it fails. it should throw an error code 500 and if you scroll up here, you see you got an error code 401.

The screenshot shows a VS Code editor with a project named 'nodejs-complete-guide'. The file explorer on the left shows a directory structure with files like .vscode, controllers, auth.js, feed.js, images, middleware, is-auth.js, models, post.js, user.js, node_modules, routes, auth.js, feed.js, test, auth-controller.js, auth-middleware.js, start.js, gitignore, app.js, package-lock.json, and package.json. The main editor window shows the file 'auth-controller.js' with the following code:

```
8 it('should throw an error with code 500 if accessing the database fails', function() {
9   sinon.stub(User, 'findOne');
10  User.findOne.throws();
11
12  const req = {
13    body: {
14      email: 'test@test.com',
15      password: 'tester'
16    }
17  };
18
19  AuthController.login(req, {}, () => {}).then(result => {
20    expect(result).to.be.an('error');
21    expect(result).to.have.property('statusCode', 401);
22    // done();
23  });
24
25  User.findOne.restore();
26 });
27 });
```

The terminal at the bottom shows the output of the tests:

```
1: bash
✓ should yield a userId after decoding the token
✓ should throw an error if the token cannot be verified
5 passing (15ms)
```

The status bar at the bottom indicates the file is at line 22, column 17, with 166 characters selected. The editor is using the JavaScript language and the Prettier formatter.

- without 'done', all tests pass even though it gets error. so it's important to pass 'done' and then call it once you are done to make sure that your tests were correctly.

```
1 //./routes/auth.js
2
3 const express = require('express');
4 const { body } = require('express-validator/check');
5
6 const User = require('../models/user');
7 const authController = require('../controllers/auth');
8 const isAuth = require('../middleware/is-auth');
9
10 const router = express.Router();
11
12 router.put(
13   '/signup',
14   [
15     body('email')
16       .isEmail()
17       .withMessage('Please enter a valid email.')
18       .custom((value, { req }) => {
19         return User.findOne({ email: value }).then(userDoc => {
20           if (userDoc) {
21             return Promise.reject('E-Mail address already exists!');
22           }
23         });
24       })
25     .normalizeEmail(),
26     body('password')
27       .trim()
28       .isLength({ min: 5 }),
29     body('name')
30       .trim()
31       .not()
32       .isEmpty()
33   ],
```

```

34   authController.signup
35 );
36
37 /**what will not test is the routing here '/login'
38 * we will not test whether we can send a request to login
39 * and we execute the login function in the ./controllers/auth.js
40 * because that entire forwarding of the request the execution of this method here
41 * 'authController.login'
42 * that is all handled by express.js
43 * as i mentioned earlier, you don't wanna test our libraries.
44 * you wanna test your own code
45 * so we will just test 'login' or 'singup' in ./controllers/auth.js
46 */
47 router.post('/login', authController.login);
48
49 router.get('/status', isAuth, authController.getUserStatus);
50
51 router.patch(
52   '/status',
53   isAuth,
54   [
55     body('status')
56       .trim()
57       .not()
58       .isEmpty()
59   ],
60   authController.updateUserStatus
61 );
62
63 module.exports = router;
64

```

```

1  //./controllers/auth.js
2
3  const { validationResult } = require('express-validator/check');
4  const bcrypt = require('bcryptjs');
5  const jwt = require('jsonwebtoken');
6
7  const User = require('../models/user');
8
9  exports.signup = async (req, res, next) => {
10   const errors = validationResult(req);
11   if (!errors.isEmpty()) {
12     const error = new Error('Validation failed. ');
13     error.statusCode = 422;
14     error.data = errors.array();
15     throw error;
16   }
17   const email = req.body.email;
18   const name = req.body.name;
19   const password = req.body.password;
20   try {
21     const hashedPw = await bcrypt.hash(password, 12);
22
23     const user = new User({
24       email: email,
25       password: hashedPw,

```

```

26     name: name
27   });
28   const result = await user.save();
29   res.status(201).json({ message: 'User created!', userId: result._id });
30 } catch (err) {
31   if (!err.statusCode) {
32     err.statusCode = 500;
33   }
34   next(err);
35 }
36 };
37
38 exports.login = async (req, res, next) => {
39   const email = req.body.email;
40   const password = req.body.password;
41   let loadedUser;
42   try {
43     /**User model is based on mongoose and MongoDB
44     * so how can we test our Database?
45     *
46     * strategy 1 for testing code that involves database operations
47     * is that we stub or mock the part that rely on dataase access
48     * when we execute findOne,
49     * we again create a step that returns a predefined result
50     * and we then test if our code behaves correctly.
51     * for example,
52     * we might be interested in finding out
53     * how our code behaves when findOne throws error.
54     * so if we are having trouble interacting with the database
55     * or how our code behaves if we don't have a user with that email address when logging
in
56     * these are 2 different scenarios
57     * and we can write 2 different tests for that.
58     *
59     * we throw error on 'no user' manually
60     * and if findOne fails, it will throw an error
61     * but statusCode for example should be different
62     * because the statusCode should be 500
63     * if the database fails
64     * because we use our default code
65     * or we set our own 401 code.
66     * if we have no user
67     */
68     const user = await User.findOne({ email: email });
69     if (!user) {
70       const error = new Error('A user with this email could not be found.');
```



```

81     const token = jwt.sign(
82       {
83         email: loadedUser.email,
84         userId: loadedUser._id.toString()
85       },
86       'somesupersecretsecret',
87       { expiresIn: '1h' }
88     );
89     res.status(200).json({ token: token, userId: loadedUser._id.toString() });
90     /**so in the success case,
91     * i return nothing in that promise
92     */
93     return;
94   } catch (err) {
95     if (!err.statusCode) {
96       err.statusCode = 500;
97     }
98     next(err);
99     /**or which in the end gets returned the error. */
100    return err;
101  }
102 };
103
104 exports.getUserStatus = async (req, res, next) => {
105   try {
106     const user = await User.findById(req.userId);
107     if (!user) {
108       const error = new Error('User not found. ');
109       error.statusCode = 404;
110       throw error;
111     }
112     res.status(200).json({ status: user.status });
113   } catch (err) {
114     if (!err.statusCode) {
115       err.statusCode = 500;
116     }
117     next(err);
118   }
119 };
120
121 exports.updateUserStatus = async (req, res, next) => {
122   const newStatus = req.body.status;
123   try {
124     const user = await User.findById(req.userId);
125     if (!user) {
126       const error = new Error('User not found. ');
127       error.statusCode = 404;
128       throw error;
129     }
130     user.status = newStatus;
131     await user.save();
132     res.status(200).json({ message: 'User updated.' });
133   } catch (err) {
134     if (!err.statusCode) {
135       err.statusCode = 500;
136     }

```

```
137     next(err);
138   }
139 };
140
```

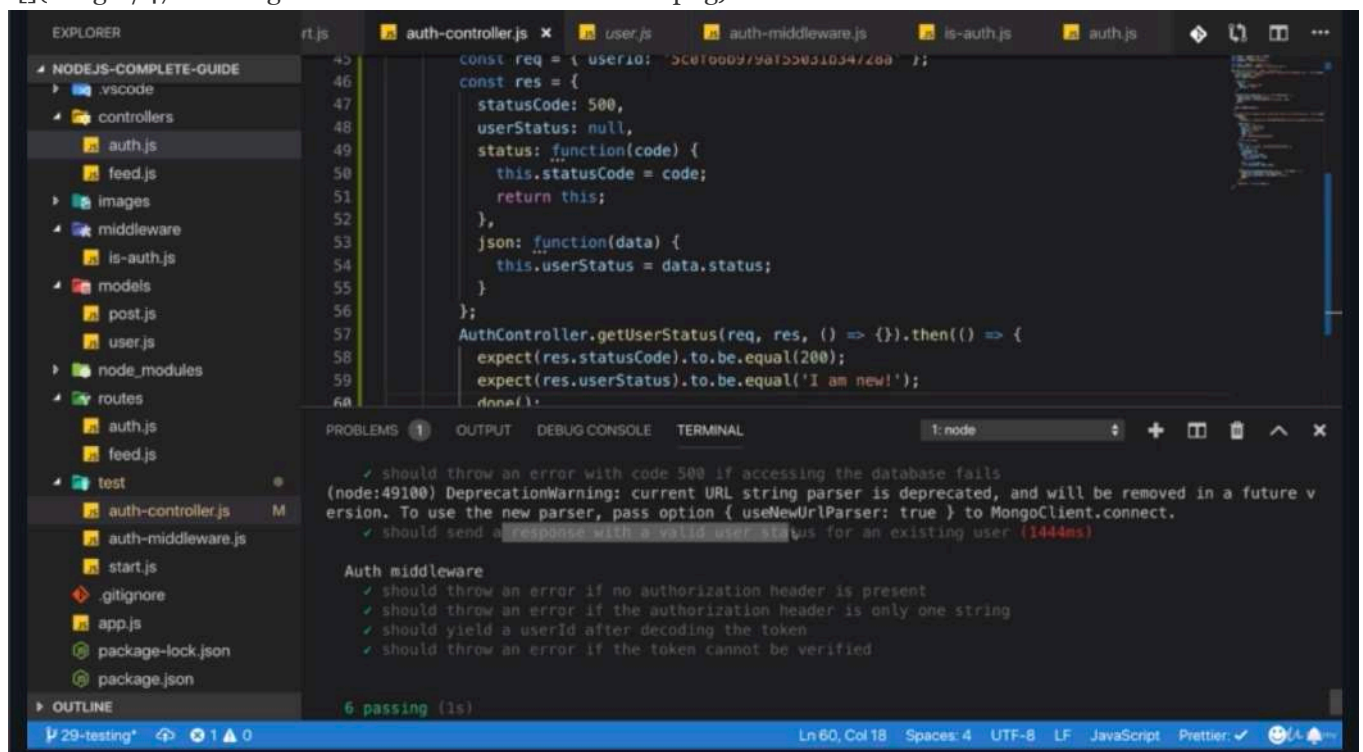
```
1  //./test/auth-controller.js
2
3  const expect = require('chai').expect;
4  const sinon = require('sinon');
5
6  const User = require('../models/user');
7  const AuthController = require('../controllers/auth');
8
9  describe('Auth Controller - Login', function() {
10    /**mocha doesn't wait for this test case to finish
11     * because we have async code
12     * and by default, it doesn't wait for async code to resolve
13     * it doesn't wait for promises to resolve no matter how fast is this
14     *
15     * we can tell mocha to wait
16     * and we do this by adding extra argument in this function
17     * we pass to it
18     * and that's 'done' argument
19     * this is optional
20     * and it's the function which you can call
21     * so mocha gives you a function 'done'
22     * which you can call once this test case is done
23     * by default, it's done once the execute the code top to bottom
24     * but if you accept this argument,
25     * it will wait for you to call it
26     * and then you can call it in a asynchronous code snippet.
27     */
28    it('should throw an error with code 500 if accessing the database fails', function(done) {
29      /**the important thing is that
30       * i'm faking that database fails
31       * because i replace findOne method with stub
32       * that throw an error
33       * because i wanna check we should throw an error with code 500
34       * so i wanna check whether our default statusCode 500 get applied correctly
35       */
36      sinon.stub(User, 'findOne');
37      User.findOne.throws();
38      /**in login in ./controllers/auth.js file,
39       * we use async code
40       * which means in the end we use promises in there
41       * and that means we have asynchronous code
42       * which is a some complexity we have to deal with
43       * because the execution of that code will not happen synchronously
44       * that means by default our expectation won't work the way you might expect it to
45       work
46       *
47       * we wanna check whether that promises we had there
48       * eventually returns an error for this test
49       * for that, let's make a tiny adjustment in ./controllers/auth.js file
50       */
51      //expect(AuthController.login)
52      const req = {
```

```

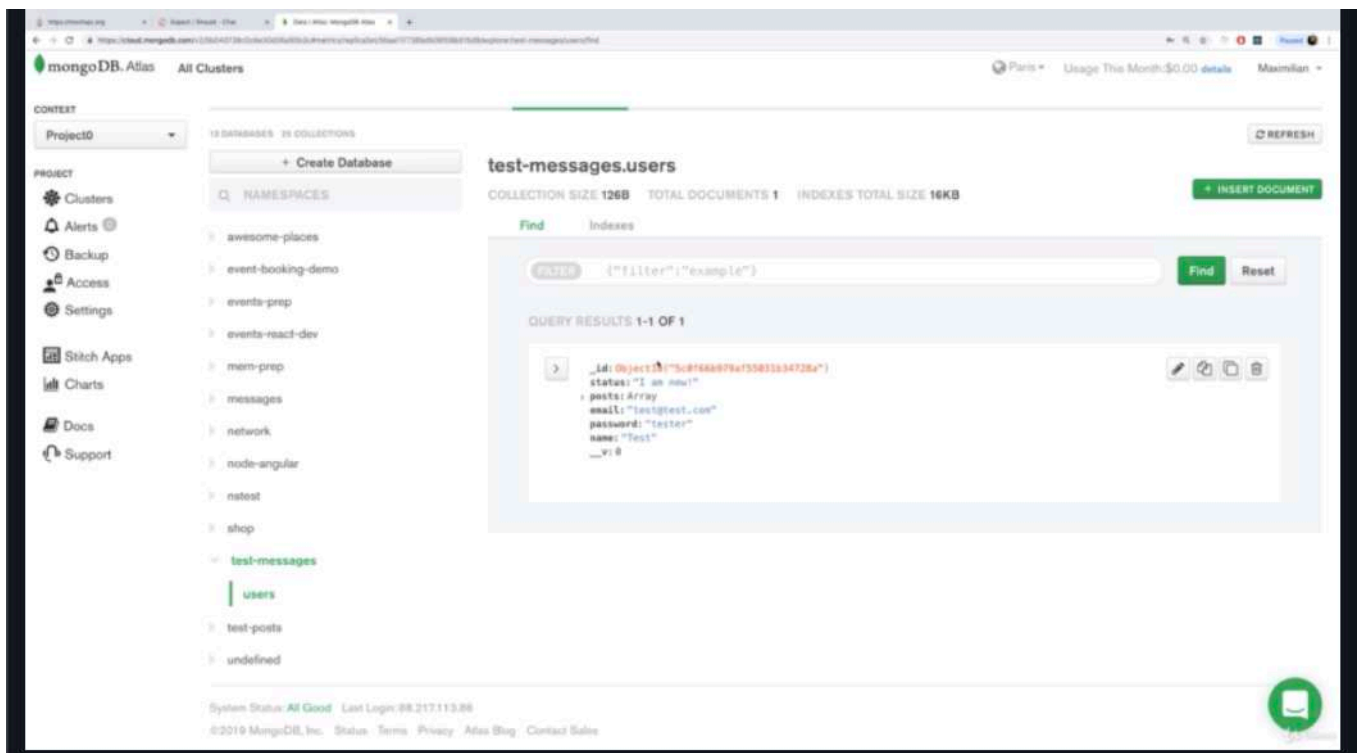
52     body: {
53       email: 'test@test.com',
54       password: 'tester'
55     }
56   };
57
58   AuthController.login(req, {}, () => {}).then(result => {
59     /**Chai is able to detect a couple of types of data
60      * and error is one of them*/
61     expect(result).to.be.an('error');
62     expect(result).to.have.property('statusCode', 500);
63     /**this is the signal
64      * that i want mocha to wait for this code to execute
65      * because before this async code,
66      * this test above case is done
67      */
68     done();
69   });
70
71   User.findOne.restore();
72 });
73 });
74

```

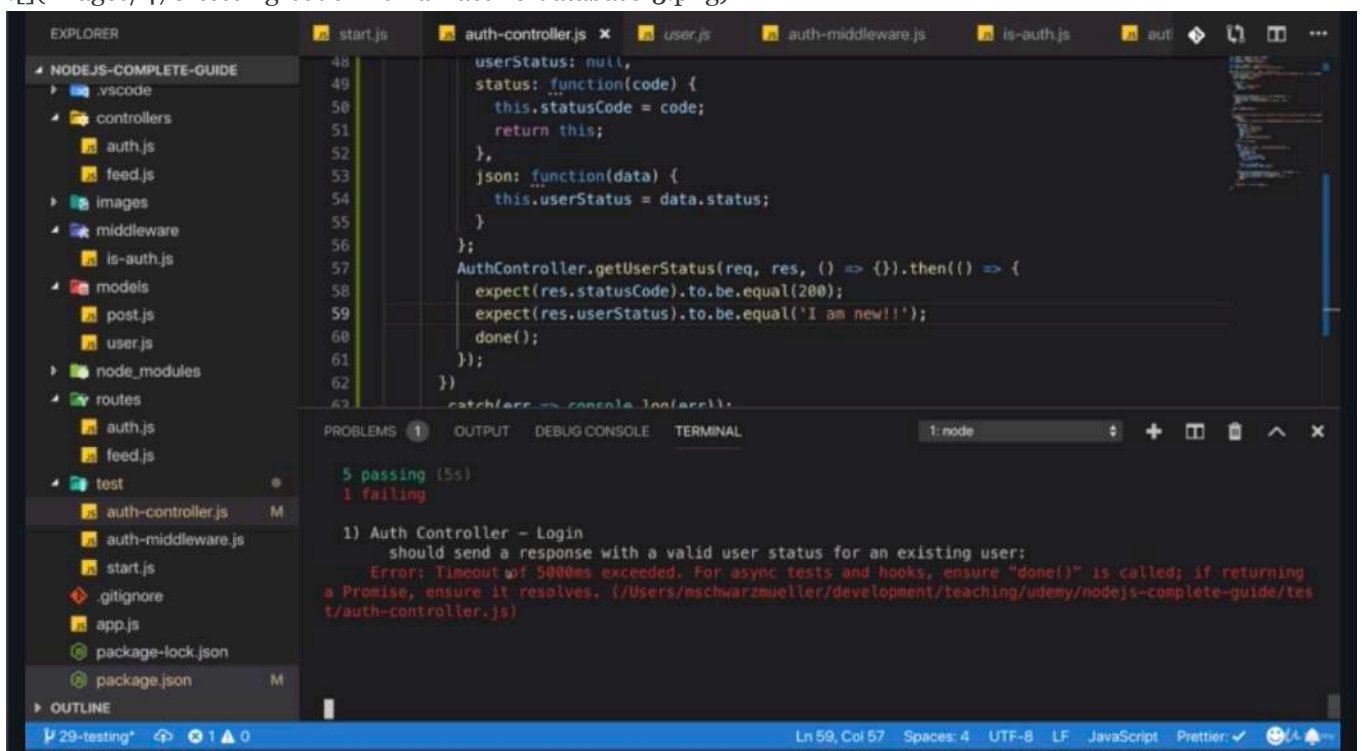
* Chapter 470: Testing Code With An Active Database



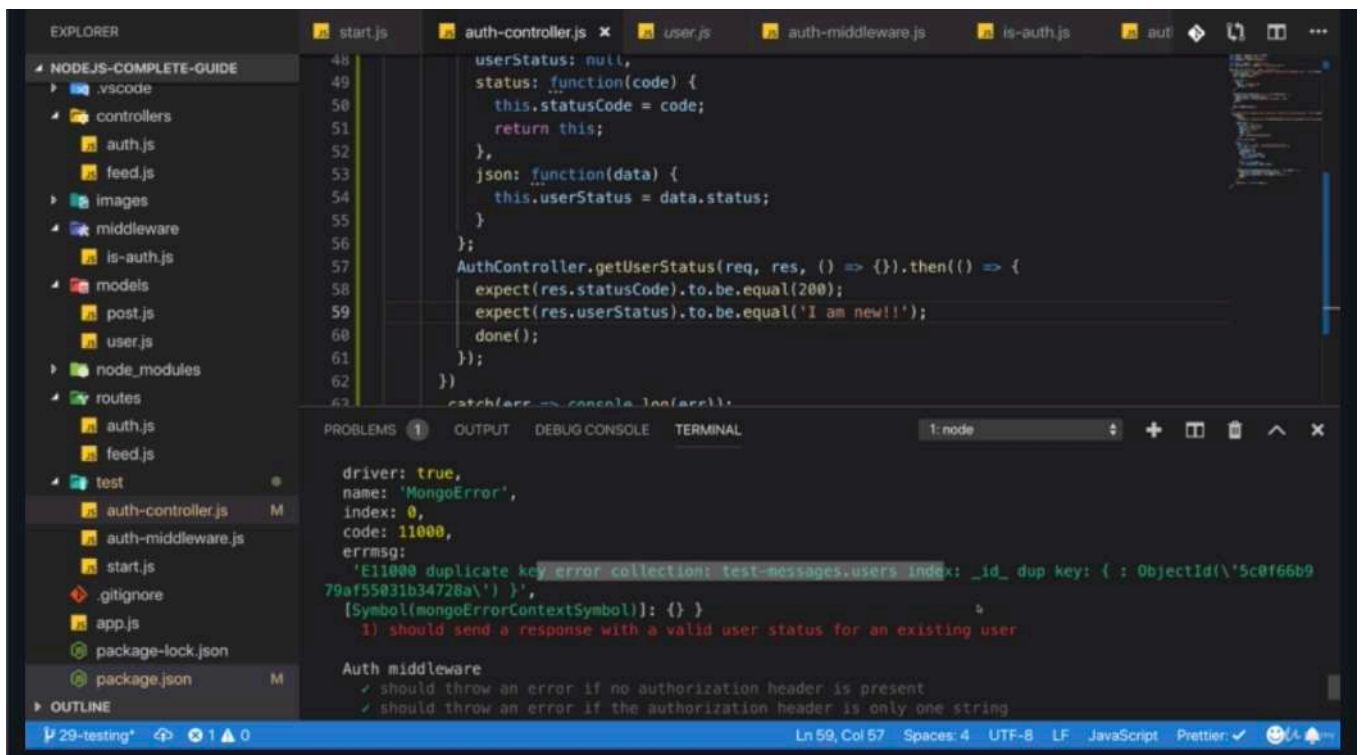
- if i run npm test, we have 6 passing tests including this one here. it took quite long.



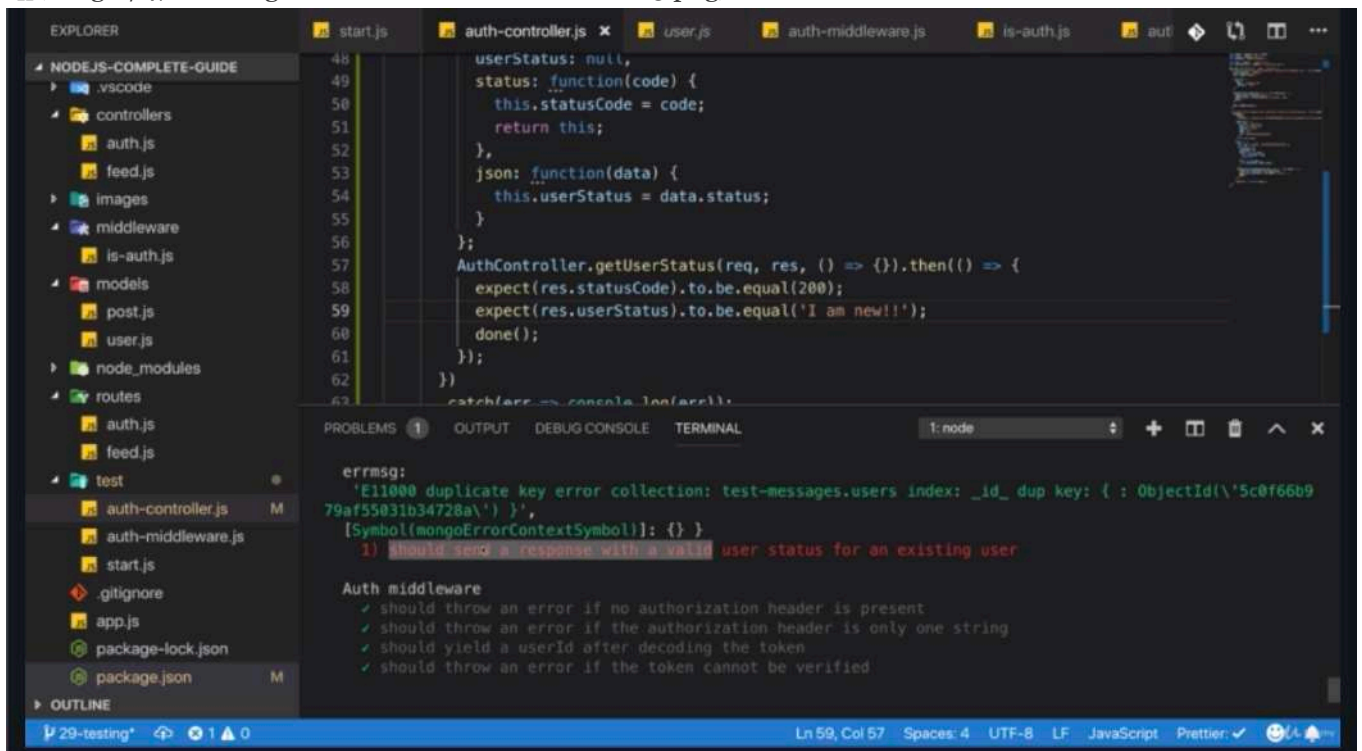
- by the way, if this time's out, you can define a longer timeout period by going to your scripts and adding '—timeout 5000' 5second because this is milliseconds. default is 2000 milliseconds.



- if i go to database, i have that dummy user.



- if you wanna really be sure, if i check for a different status not 'I am new!' but 'I am new!!' which is incorrect and rerun this.



- then we get a timeout error because we got one failing test where we have that test up there failing.

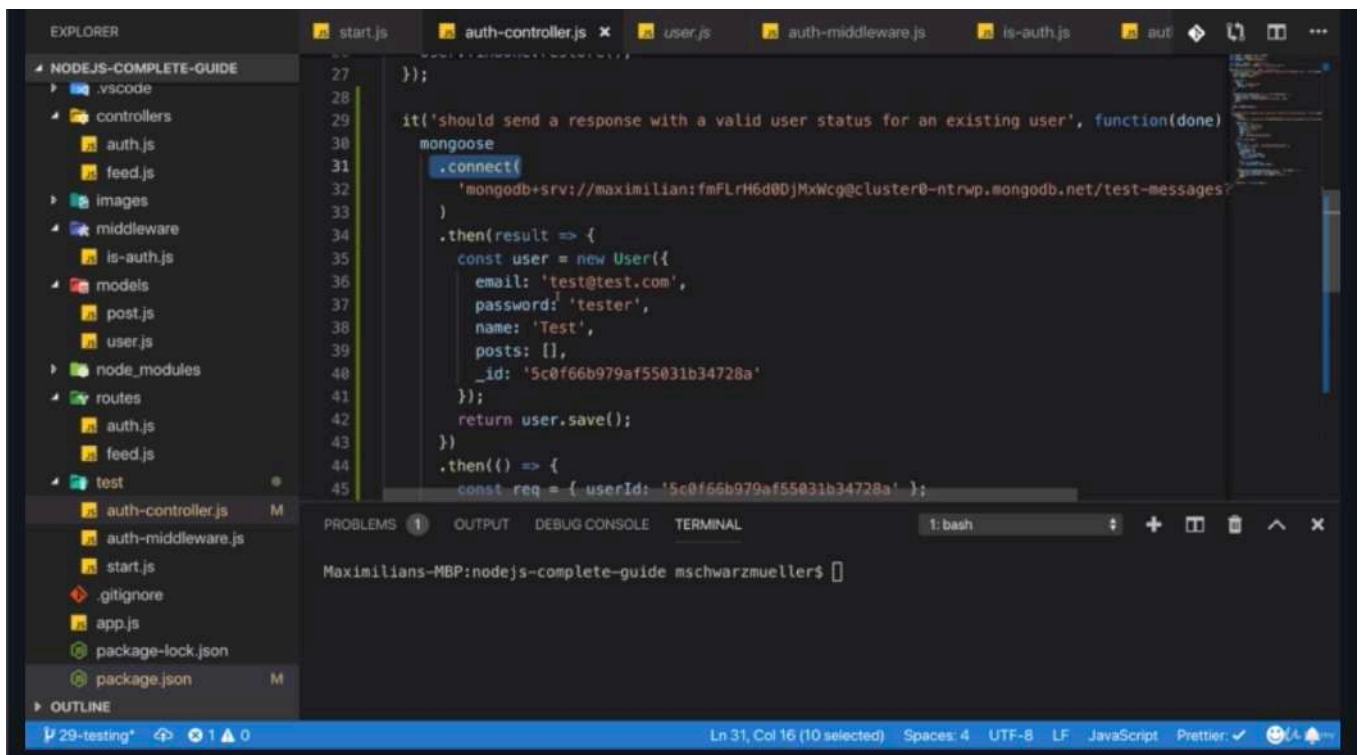
The screenshot shows a VS Code editor with a project named '29-testing'. The Explorer sidebar on the left shows a file structure with folders like 'controllers', 'middleware', 'models', 'routes', and 'test'. The 'auth-controller.js' file is open in the editor, showing a function that updates a user's status. The terminal at the bottom displays a MongoDB error: 'E11000 duplicate key error collection: test-messages.users index: _id, dup key: { : ObjectId(\'5c0f66b979af55031b34728a\') }'. The error message indicates that a user with the same ID is being created again. The status bar at the bottom shows 'Ln 59, Col 57'.

- however it fails for a different reason if you watch closely, you will see it fails because we have a duplicate key issue

This screenshot shows the same VS Code editor, but now the 'auth-controller.js' file is open at line 40, where a new user is being created. The terminal at the bottom shows the same MongoDB error: 'E11000 duplicate key error collection: test-messages.users index: _id, dup key: { : ObjectId(\'5c0f66b979af55031b34728a\') }'. The error message indicates that a user with the same ID is being created again. The status bar at the bottom shows 'Ln 40, Col 42 (52 selected)'.

- and that stems from our setup code. create a new user. that now for the second test run, already exists and they are all different issues. this process doesn't quit as it did before we manually have to do this with control+c let's fix in the next lecture.

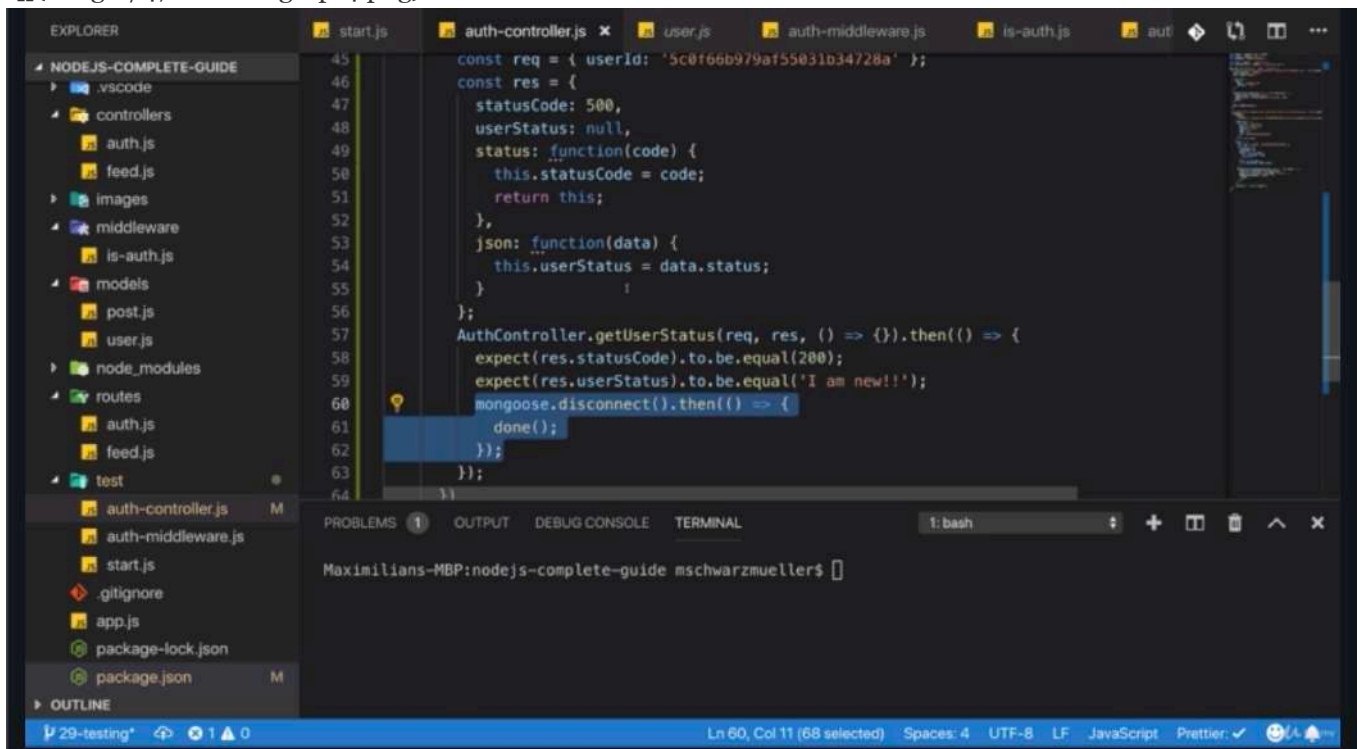
* Chapter 471: Cleaning Up



```
27 });
28
29 it('should send a response with a valid user status for an existing user', function(done) {
30   mongoose
31     .connect('mongodb+srv://maximilian:fmFLRH6d0DJMxWcg@cluster0-ntrwp.mongodb.net/test-messages?')
32     .then(result => {
33       const user = new User({
34         email: 'test@test.com',
35         password: 'tester',
36         name: 'Test',
37         posts: [],
38         _id: '5c0f66b979af55031b34728a'
39       });
40       return user.save();
41     })
42     .then(() => {
43       const req = { userId: '5c0f66b979af55031b34728a' };
44     });
45 });
```

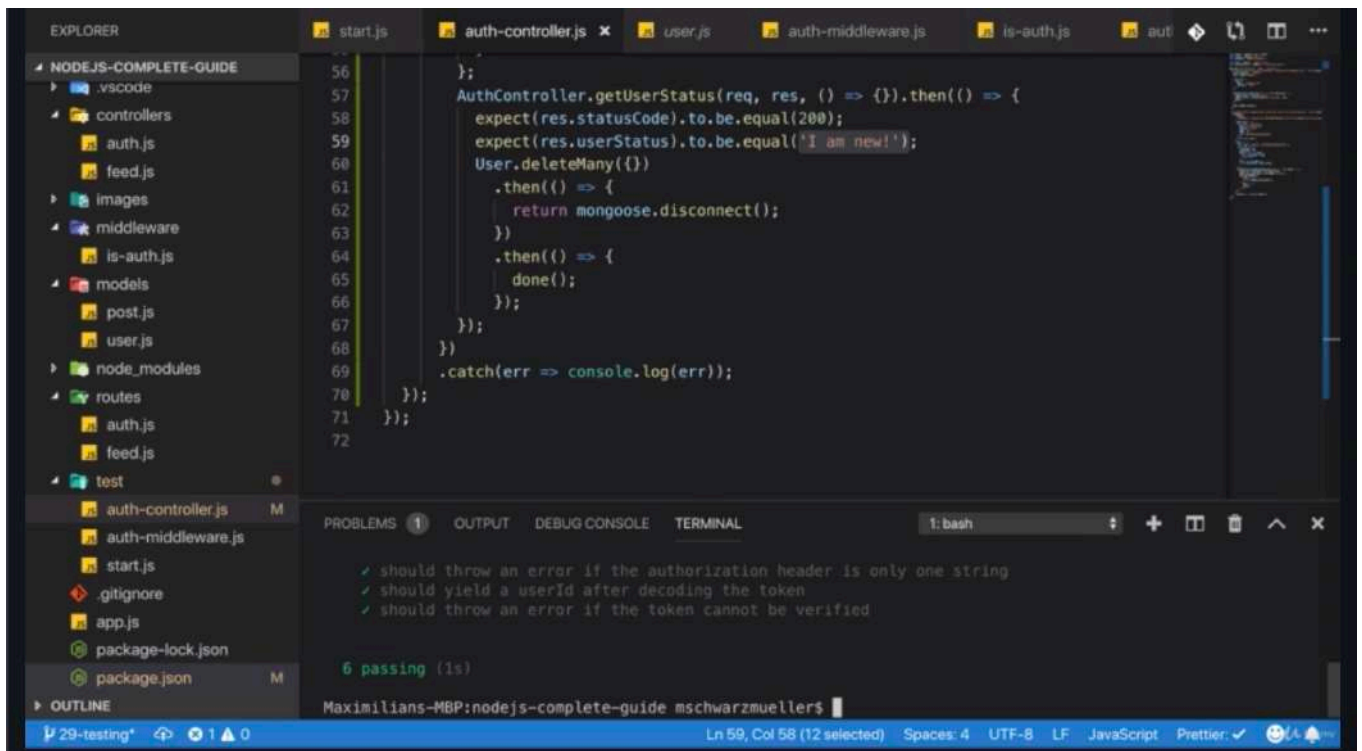
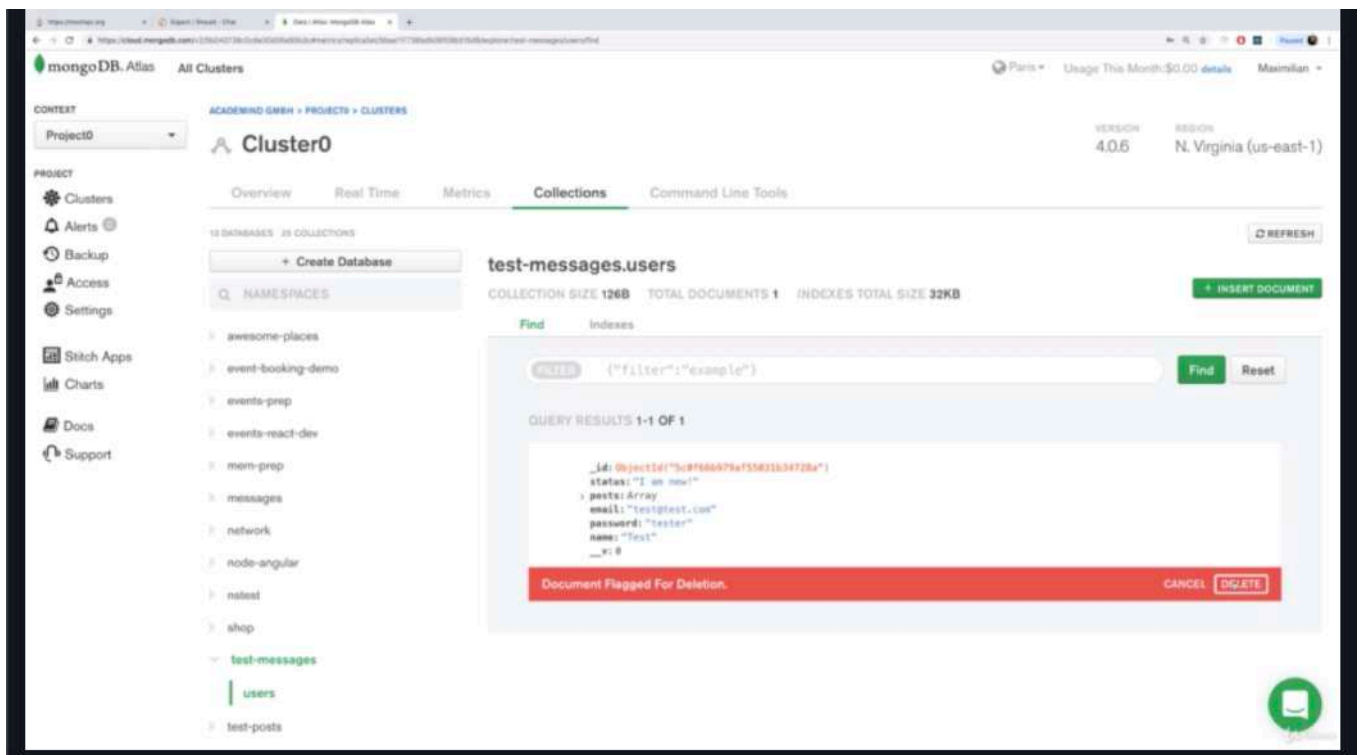
Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$

- why do i need to quit this process with control+c? the reason is despite me calling done mocha detects that there is still some open process in the event loop and indeed there is our database connection which we open but never close.

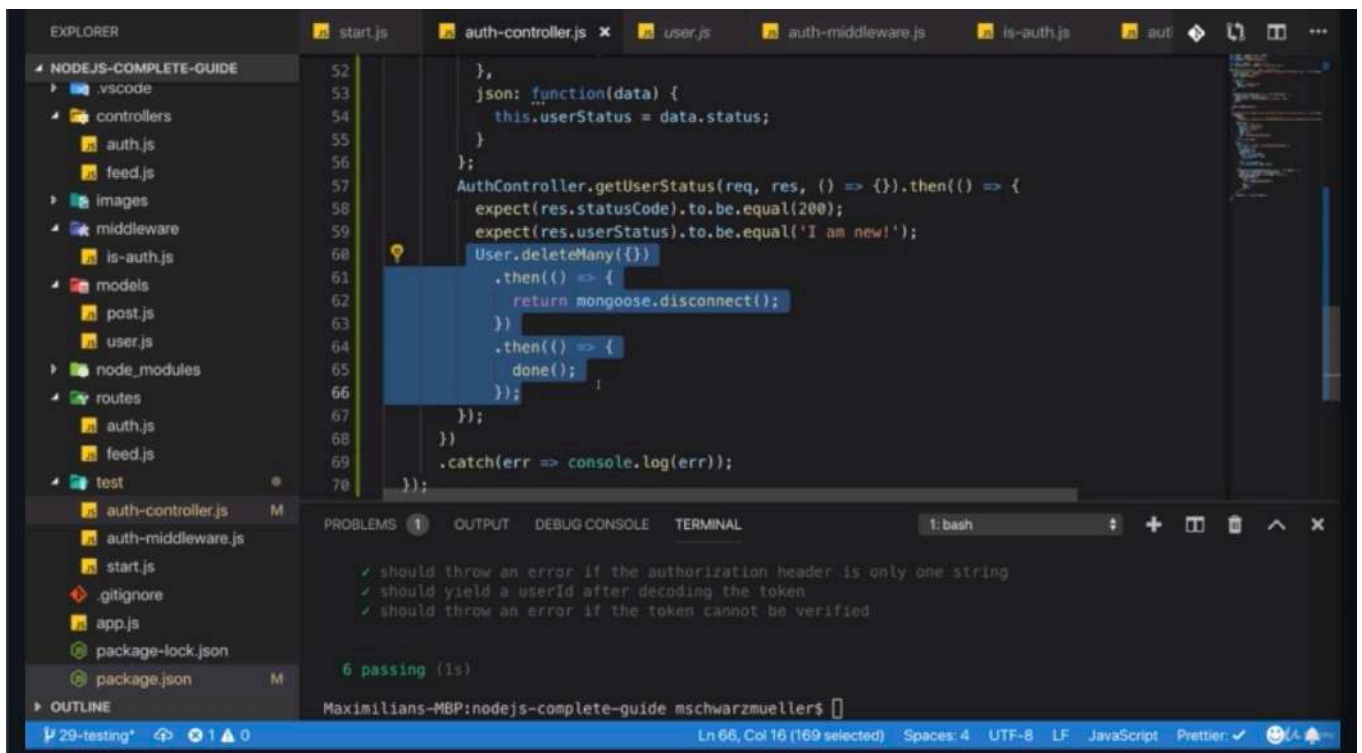


```
45 const req = { userId: '5c0f66b979af55031b34728a' };
46 const res = {
47   statusCode: 500,
48   userStatus: null,
49   status: function(code) {
50     this.statusCode = code;
51     return this;
52   },
53   json: function(data) {
54     this.userStatus = data.status;
55   }
56 };
57 AuthController.getUserStatus(req, res, () => {}).then(() => {
58   expect(res.statusCode).to.be.equal(200);
59   expect(res.userStatus).to.be.equal('I am new!!');
60   mongoose.disconnect().then(() => {
61     done();
62   });
63 });
64 });
```

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



- so one thing we should do is when we are done with our expectations, we might wanna call Mongoose disconnect and only when this is done. but it wouldn't work because we have an overall error in our test case.



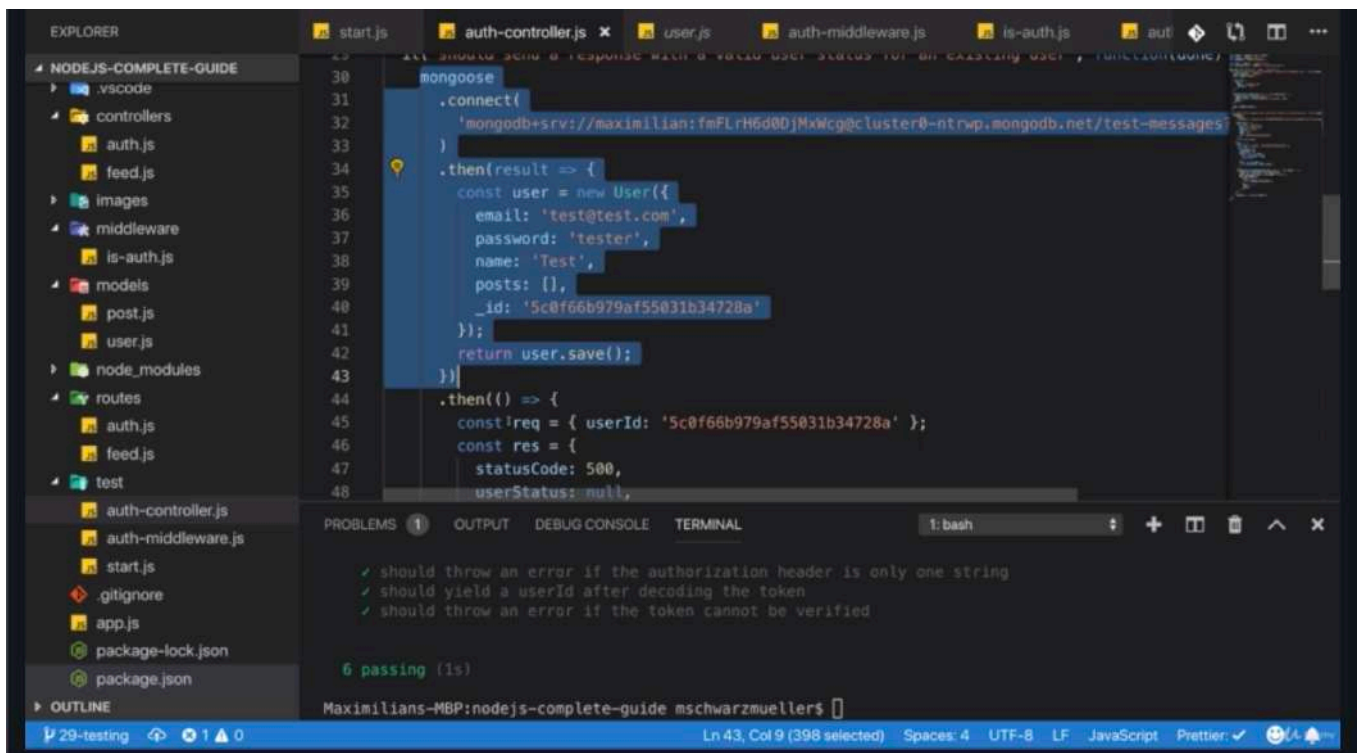
- this is not perfect as you can tell whenever a test fails, we don't make it into this cleanup phase because that will throw an error and therefore we would have to add a catch phrase overall so we don't make it into this cleanup part here. if an expectation fails and in general this is pretty clunky and pretty hard to read code and if we have a test that require a MongoDB database or our dummy set up, then we have to repeat all that code and therefore there is a cleaner solution to all of that.

* Chapter 472: Hooks

1. update
- ./test/auth-controller.js
- package.json

- the cleaner solution comes in the form of lifecycle hooks provided by mocha. inside describe() function, we have a certain extra functions we can call that will run before each tests at the same for after and after each what do i mean with that?

- let's say connecting to the database and creating one dummy user is something we wanna do when our tests run not before every test. so we don'tn wanna reconnect and recreate a user before every test. but initially when our test run starts.



```
30 mongoose
31 .connect(
32   'mongodb+srv://maximilian:fmFLrH6d0DJMxWcg@cluster0-ntrwp.mongodb.net/test-messages'
33 )
34 .then(result => {
35   const user = new User({
36     email: 'test@test.com',
37     password: 'tester',
38     name: 'Test',
39     posts: [],
40     _id: '5c0f66b979af55031b34728a'
41   });
42   return user.save();
43 })
44 .then(() => {
45   const req = { userId: '5c0f66b979af55031b34728a' };
46   const res = {
47     statusCode: 500,
48     userStatus: null,
```

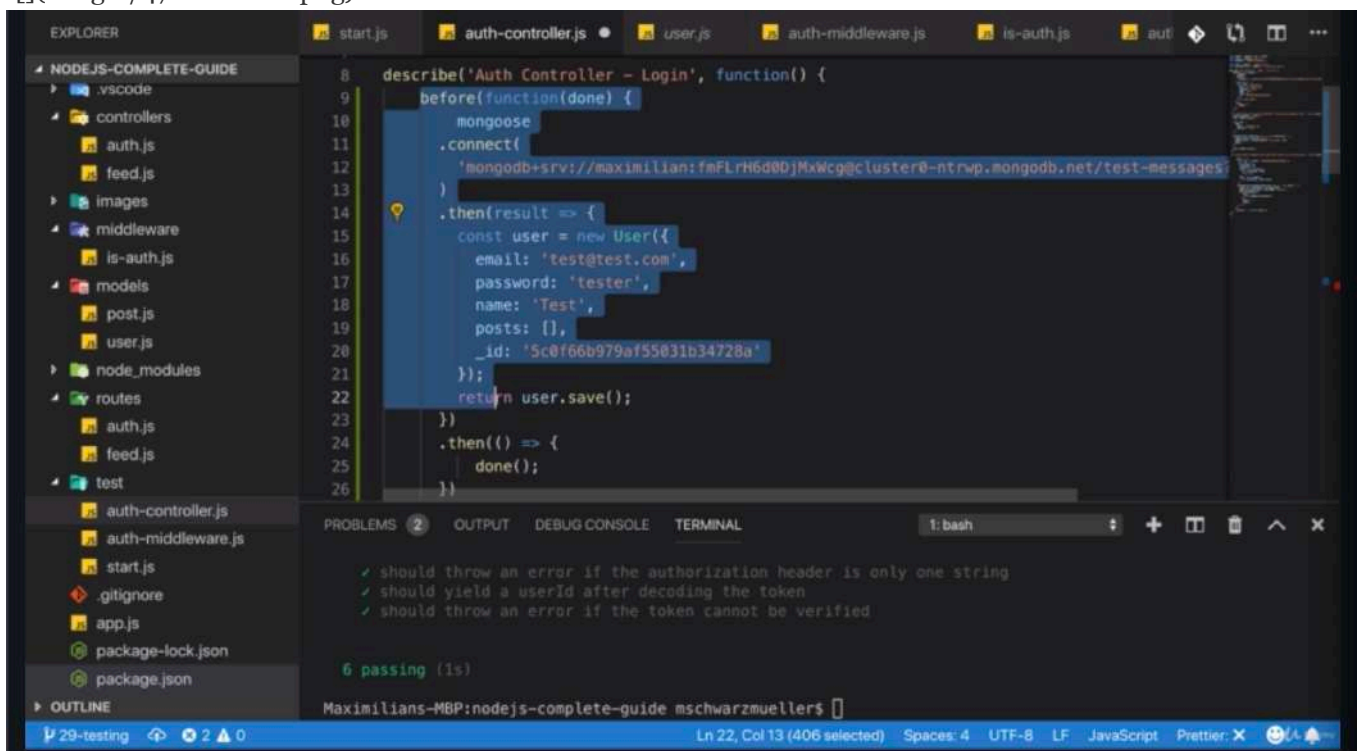
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

- ✓ should throw an error if the authorization header is only one string
- ✓ should yield a userId after decoding the token
- ✓ should throw an error if the token cannot be verified

6 passing (1s)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

- so i wanna run this code. i will cut it before every test.



```
8 describe('Auth Controller - Login', function() {
9   before(function(done) {
10     mongoose
11     .connect(
12       'mongodb+srv://maximilian:fmFLrH6d0DJMxWcg@cluster0-ntrwp.mongodb.net/test-messages'
13     )
14     .then(result => {
15       const user = new User({
16         email: 'test@test.com',
17         password: 'tester',
18         name: 'Test',
19         posts: [],
20         _id: '5c0f66b979af55031b34728a'
21       });
22       return user.save();
23     })
24     .then(() => {
25       done();
26     })
27   })
```

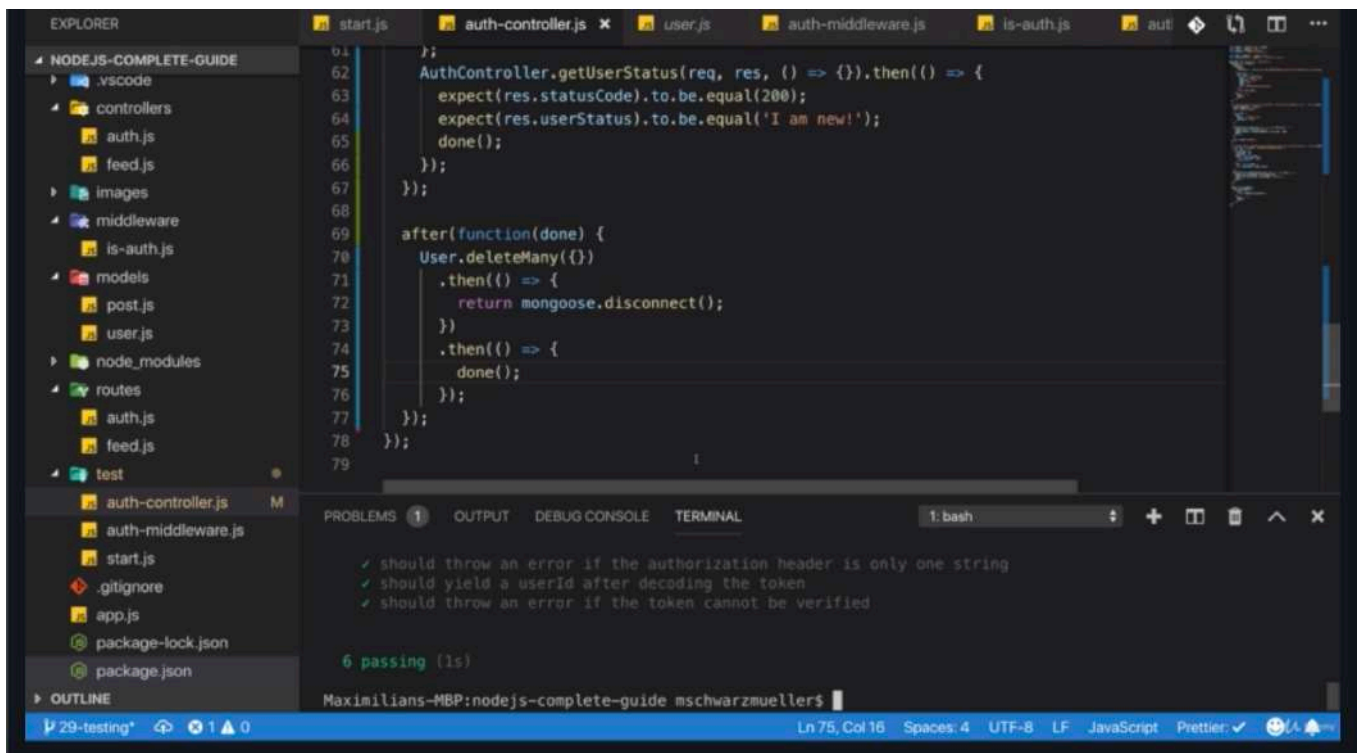
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

- ✓ should throw an error if the authorization header is only one string
- ✓ should yield a userId after decoding the token
- ✓ should throw an error if the token cannot be verified

6 passing (1s)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

- you can be achieved by 'before()' function. once you are done, mocha knows you are done with your initialization and it will start running your test cases. so it runs all your test cases after 'before()' and before() only executes once not before every test case but before all test cases.



- besides 'before()' and 'after()', there also are 'beforeEach()' and 'afterEach()'. the differences is that 'beforeEach' is initialization work that it runs before every test case. before runs before all test case. it's not repetitive. it only runs once per test. so it's useful if you need to reset something before every test case or if you wanna have some initialization work that has to run before every test case and there are all those 'afterEach()' in case, there is some functionality which need to run after every test case. so cleanup work which needs to be done after every test case.

```
1 //package.json
2
3 {
4   "name": "nodejs-complete-guide",
5   "version": "1.0.0",
6   "description": "",
7   "main": "index.js",
8   "scripts": {
9     "test": "mocha --timeout 5000",
10    "start": "nodemon app.js"
11  },
12  "repository": {
13    "type": "git",
14    "url": "https://git-codecommit.us-east-1.amazonaws.com/v1/repos/udemy-course-nodejs-complete"
15  },
16  "author": "",
17  "license": "ISC",
18  "dependencies": {
19    "bcryptjs": "^2.4.3",
20    "body-parser": "^1.18.3",
21    "express": "^4.16.3",
22    "express-validator": "^5.3.0",
23    "jsonwebtoken": "^8.3.0",
24    "mongoose": "^5.3.2",
25    "multer": "^1.4.0"
26  },
27  "devDependencies": {
```

```

28     "chai": "^4.2.0",
29     "mocha": "^6.1.4",
30     "nodemon": "^1.18.4",
31     "sinon": "^7.3.2"
32   }
33 }
34

```

```

1  //./test/auth-controller.js
2
3  const expect = require('chai').expect;
4  const sinon = require('sinon');
5  const mongoose = require('mongoose');
6
7  const User = require('../models/user');
8  const AuthController = require('../controllers/auth');
9
10 describe('Auth Controller', function() {
11   before(function(done){
12     mongoose
13       .connect(
14         /**don't use the production database
15          * so 'test-messages' not the 'messages'
16          */
17         'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/test-messages?
retryWrites=true&w=majority'
18       )
19     .then(result => {
20       const user = new User({
21         email: 'test@test.com',
22         password: 'tester',
23         name: 'Test',
24         posts: [],
25         /**this format of string matters
26          * consider to be a valid Id by MongoDB
27          * so i will give this user my own Id
28          * so taht i can pass the userId
29          */
30         _id: '5c0f66b979af55031b34728a'
31         /**status doesn't have to be set
32          * because there is a default defined
33          * then we call save
34          * and i return this
35          * because this will always return a promise
36          * so we can now add another '.then()' block in this function
37          */
38       });
39       return user.save();
40     })
41     .then(() => {
42       done();
43     });
44   });
45
46   /**besides 'before()' and 'after()',
47    * there also are 'beforeEach()' and 'afterEach()'.
48    * the differences is that

```

```

49  * 'beforeEach' is initialization work
50  * that it runs before every test case.
51  *
52  * before runs before all test case.
53  * it's not repetitive.
54  * it only runs once per test.
55  * so it's useful if you need to reset something
56  * before every test case
57  * or if you wanna have some initialization work
58  * that has to run before every test case
59  * and there are all those 'afterEach()' in case,
60  * there is some functionality
61  * which need to run after every test case.
62  * so cleanup work which needs to be done after every test case.
63  */
64  beforeEach(function(){});
65
66  afterEach(function(){});
67
68  it('should throw an error with code 500 if accessing the database fails', function(done) {
69      sinon.stub(User, 'findOne');
70      User.findOne.throws();
71      const req = {
72          body: {
73              email: 'test@test.com',
74              password: 'tester'
75          }
76      };
77
78      AuthController.login(req, {}, () => {}).then(result => {
79          expect(result).to.be.an('error');
80          expect(result).to.have.property('statusCode', 500);
81          done();
82      });
83
84      User.findOne.restore();
85  });
86
87  it('should send a response with a valid user status for an existing user', function(done){
88      const req = { userId: '5c0f66b979af55031b34728a' }
89      const res = {
90          statusCode: 500,
91          userStatus: null,
92          status: function(code){
93              this.statusCode = code;
94              /**so that this status returns this response object again */
95              return this;
96          },
97          json: function(data){
98              this.userStatus = data.status;
99          }
100      };
101      AuthController.getUserStatus(req, res, () => {}).then(() => {
102          expect(res.statusCode).to.be.equal(200);
103          /**'i am new!' is from ./models/user.js file */
104          expect(res.userStatus).to.be.equal('I am new!');

```

```

105     /**i wanna call 'User.deleteMany()'
106     * and apss on an empty object
107     * which means all users are deleted
108     * and that is not the worst idea
109     * if you have a test for use setup dummy data
110     * clean up everything after that test
111     * so that you can be sure
112     * that you have a clean setup for the next test
113     * and also for the next test run
114     * which is the issue here for our second test run
115     */
116     /**clearing our users and disconnecting is not something i wanna do here
117     * i just wanna extra expectations and call 'done()'
118     */
119     done();
120   });
121 });
122 /**'after()' will run after all your test cases.
123  * you execute your synchronous or async code
124  * if it's async,
125  * you must not forget to call done once you are done.
126  */
127 after(function(done){
128   User.deleteMany({}).then(() => {
129     return mongoose.disconnect();
130   })
131   .then(() => {
132     done();
133   })
134 })
135 });

```

* Chapter 473: Testing Code That Requires Authentication

1. update
 - ./test/feed-controller.js
 - ./controllers/feed.js


```
68     next(err);
69     return err;
70   }
71 };
72
73 exports.getUserStatus = async (req, res, next) => {
74   try {
75     const user = await User.findById(req.userId);
76     if (!user) {
77       const error = new Error('User not found.');
```

- how can we make this work if we only wanna test the controller and not the full flow? it's important that you configure your tests you pass in a request response object and the next function.


```
38   if (!req.file) {
39     const error = new Error('No image provided.');
```



```
11 let decodedToken;
12 try {
13   decodedToken = jwt.verify(token, 'somesupersecretsecret');
14 } catch (err) {
15   err.statusCode = 500;
16   throw err;
17 }
18 if (!decodedToken) {
19   const error = new Error('Not authenticated. ');
20   error.statusCode = 401;
21   throw error;
22 }
23 req.userId = decodedToken.userId;
24 next();
25 };
26
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

1: bash

- ✓ should throw an error if the authorization header is only one string
- ✓ should yield a userId after decoding the token
- ✓ should throw an error if the token cannot be verified

6 passing (1s)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers

- and if you have code that tries to get something out of the request object like the `userId` in `'creator'`. in the real app that is set by the middleware and we are testing this middleware but for the controller, we can just fake that and we can pass on a request object that has a `userId` and we are done. let's now write a test for `'createPost'` in `./controllers/feed.js` file

Actually, we'll still need the "User" model. I'll revert this later, feel free to simply not change it to "Post".

```
5 const User = require('../models/user');
6 const { password, posts } = User;
7 describe('createPost', function() {
8   before(function(done) {
9     mongoose
10      .connect(
11        'mongodb+srv://maximilian:fmFLrH6d0JmXWcg@cluster0-ntrwp.mongodb.net/test-messages?
12      )
13      .then(result => {
14        const user = new User({
15          email: 'test@test.com',
16          password: 'tester',
17          name: 'Test',
18          posts: [],
19        });
20      });
21   });
22   it('should throw an error if the authorization header is only one string', function() {
23     // ...
24   });
25 });
```

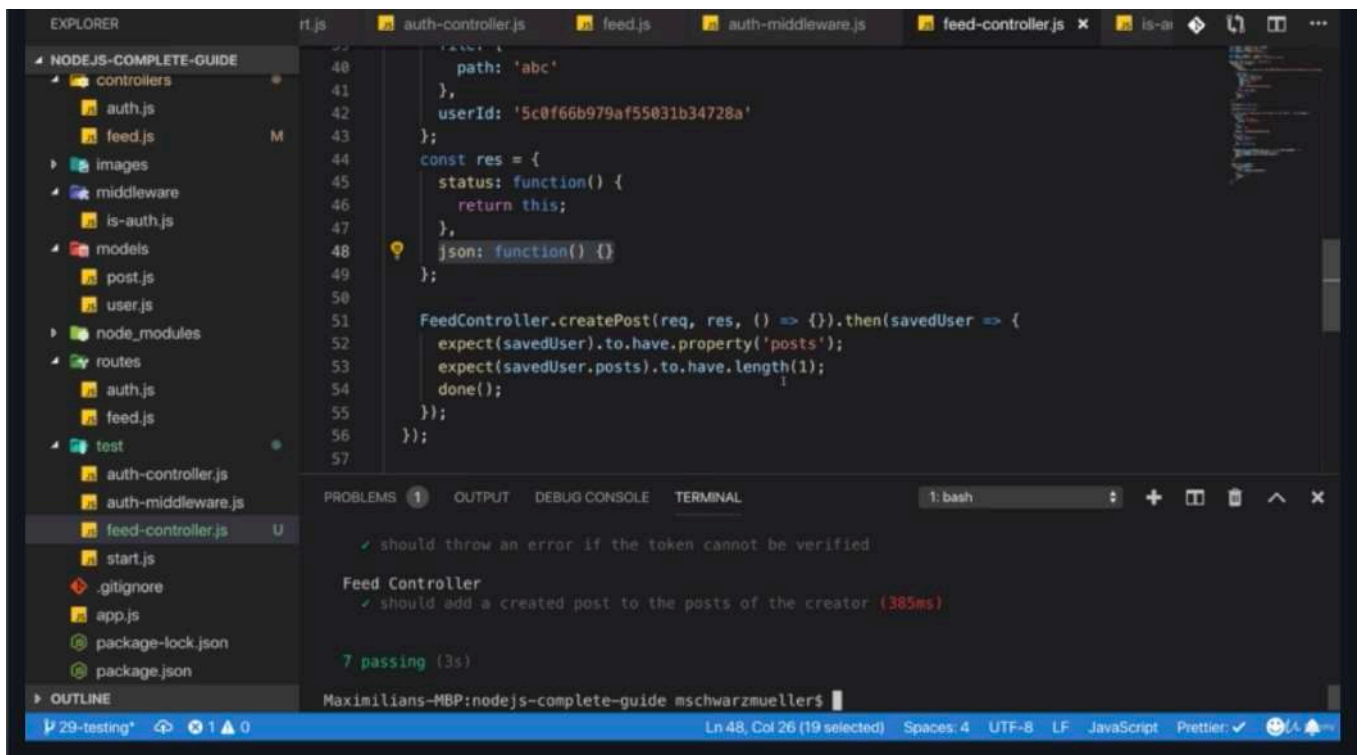
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

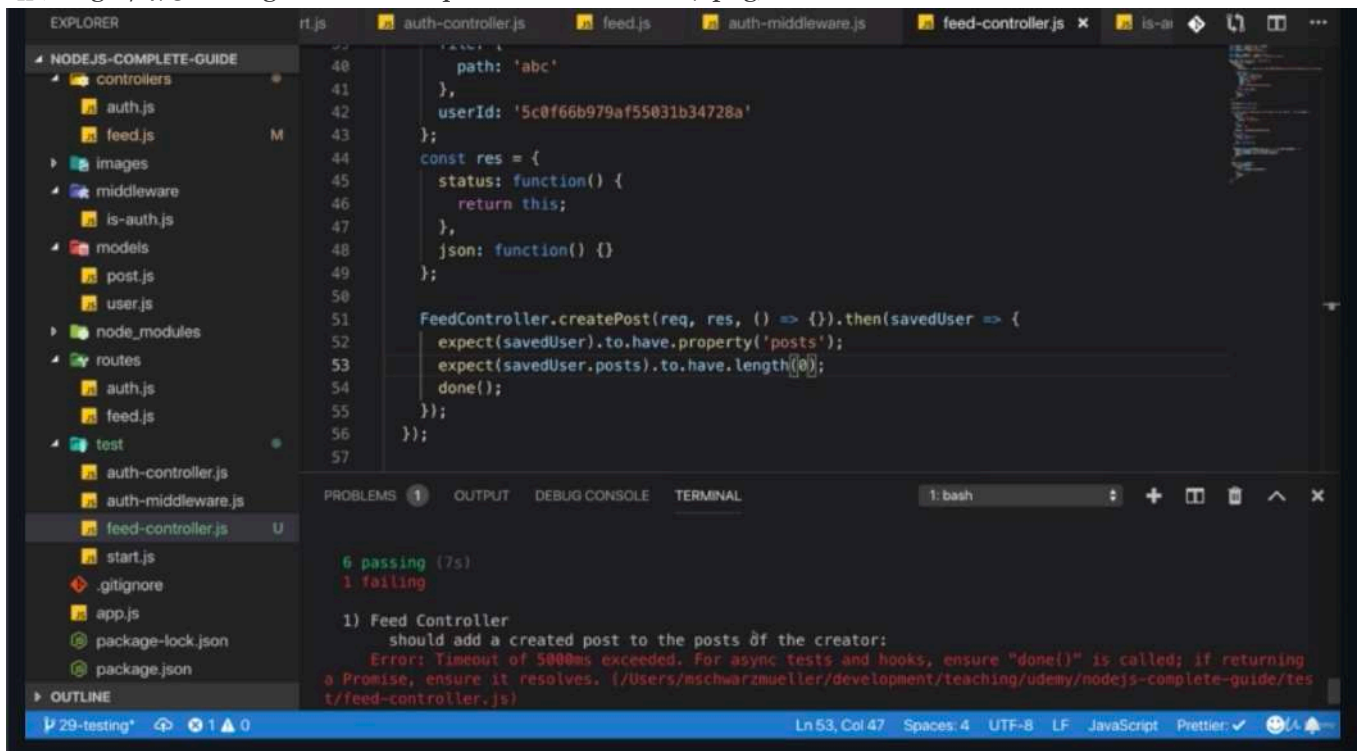
1: bash

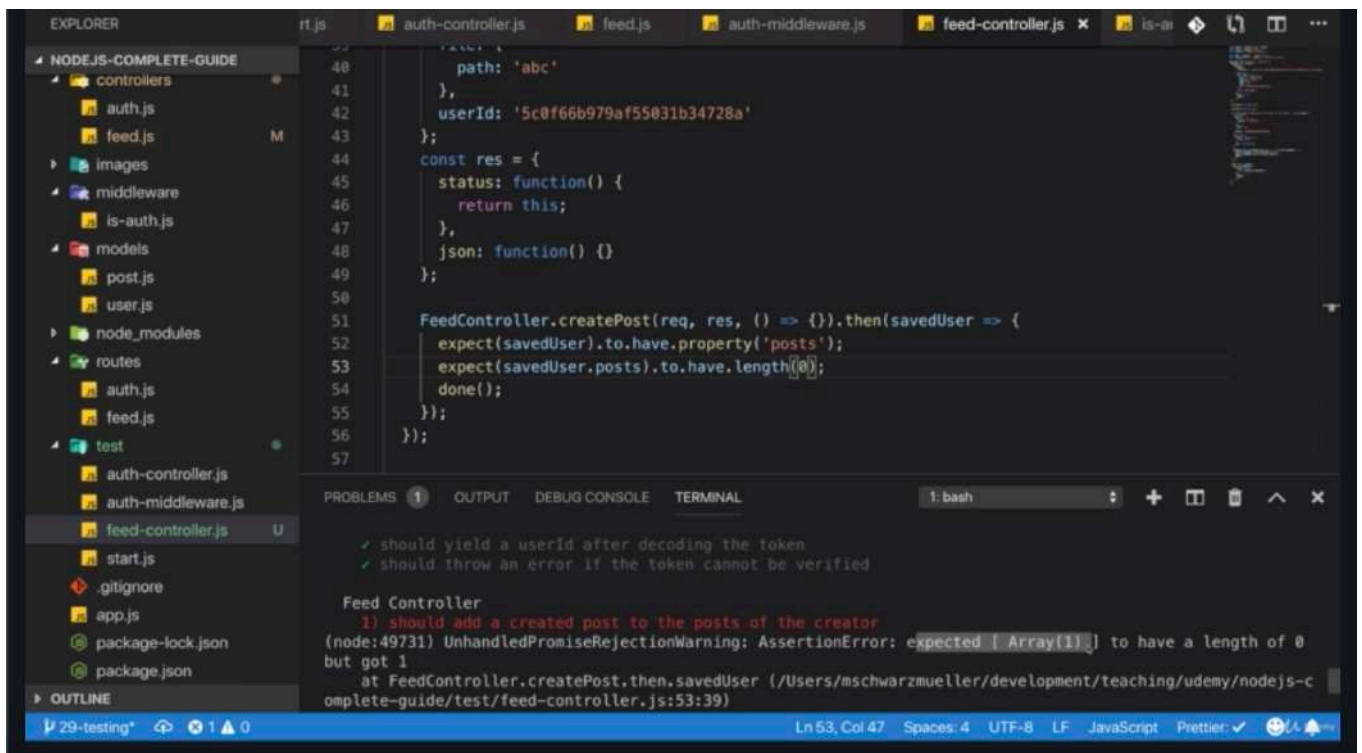
- ✓ should throw an error if the authorization header is only one string
- ✓ should yield a userId after decoding the token
- ✓ should throw an error if the token cannot be verified

6 passing (1s)

Maximilians-MBP:nodejs-complete-guide mschwarzmuellers







- but if we change from 'length(1)' to 'length(0)', then fail. if we scroll up, we see that it failed it expected array with a length(1) but we did length(0) but got length(1).

```

1  //./test/feed-controller.js
2
3  const expect = require('chai').expect;
4  const sinon = require('sinon');
5  const mongoose = require('mongoose');
6
7  const User = require('../models/user');
8  const FeedController = require('../controllers/feed');
9
10 describe('Feed Controller', function() {
11   before(function(done) {
12     mongoose
13       .connect(
14         'mongodb+srv://maximilian:rlbnjs12@cluster0-z3v1k.mongodb.net/test-messages?
15         retryWrites=true&w=majority'
16       )
17       .then(result => {
18         const user = new User({
19           email: 'test@test.com',
20           password: 'tester',
21           name: 'Test',
22           posts: [],
23           _id: '5c0f66b979af55031b34728a'
24         });
25         return user.save();
26       })
27       .then(() => {
28         done();
29       });
30   });
31   beforeEach(function() {});
32

```

```

33 afterEach(function() {});
34
35 it('should add a created post to the posts of the creator', function(done) {
36     /**we need these fields
37      * because we are using them in the ./controllers/feed.js file
38      */
39     const req = {
40         body: {
41             title: 'Test Post',
42             content: 'A Test Post'
43         },
44         file: {
45             path: 'abc'
46         },
47         /**userId matter because the creator we assign here doesn't matter
48          * but to really work later
49          * or to be connected to the user object the database,
50          * we need a real userId
51          */
52         userId: '5c0f66b979af55031b34728a'
53     };
54     const res = {
55         status: function() {
56             /**the tiny problem is that
57              * we are calling '.json()' method, not on the response object
58              * but on the result of the status method call
59              * so for that to work,
60              * back in ./test/feed-controller.js file,
61              * we need to make sure that
62              * in this status function,
63              * we call first to return 'this'
64              * so that we return to other reference at the entire object
65              * which then has this json function below
66              */
67             return this;
68         },
69         json: function() {}
70     };
71     /**for the response object,
72      * we need to make sure
73      * that we don't get an error.
74      * so we need to provide a status method and adjacent method
75      * even if we don't care about what they do but we need them
76      * so i pass in a dummy response object you are queue
77      * which is an object which needs these 2 methods
78      * so they can be called without throwing error.
79      * why are they not doing anything?
80      * because for this test, i don't care
81      *
82      * in ./controllers/feed.js,
83      * 'creatPost' has async keyword
84      * and therefore it returns a promise automatically
85      * and hence into '.then' block.
86      */
87     FeedController.createPost(req, res, () => {}).then(savedUser => {
88         expect(savedUser).to.have.property('posts');

```

```

89     /**'length(1)' is because there should be one new post added to it.
90     */
91     expect(savedUser.posts).to.have.length(1);
92     done();
93   });
94 });
95
96 after(function(done) {
97   User.deleteMany({})
98     .then(() => {
99     return mongoose.disconnect();
100   })
101     .then(() => {
102     done();
103   });
104 });
105 });
106
107

```

```

1  //./controllers/feed.js
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9  const User = require('../models/user');
10
11 exports.getPosts = async (req, res, next) => {
12   const currentPage = req.query.page || 1;
13   const perPage = 2;
14   try {
15     const totalItems = await Post.find().countDocuments();
16     const posts = await Post.find()
17       .skip((currentPage - 1) * perPage)
18       .limit(perPage);
19
20     res.status(200).json({
21       message: 'Fetched posts successfully.',
22       posts: posts,
23       totalItems: totalItems
24     });
25   } catch (err) {
26     if (!err.statusCode) {
27       err.statusCode = 500;
28     }
29     next(err);
30   }
31 };
32
33 exports.createPost = async (req, res, next) => {
34   const errors = validationResult(req);
35   if (!errors.isEmpty()) {
36     const error = new Error('Validation failed, entered data is incorrect.');

```



```

38     throw error;
39 }
40 if (!req.file) {
41     const error = new Error('No image provided. ');
42     error.statusCode = 422;
43     throw error;
44 }
45 const imageUrl = req.file.path;
46 const title = req.body.title;
47 const content = req.body.content;
48 const post = new Post({
49     title: title,
50     content: content,
51     imageUrl: imageUrl,
52     creator: req.userId
53 });
54 try {
55     await post.save();
56     /**userId matter because the creator we assign here doesn't matter
57     * but to really work later
58     * or to be connected to the user object the database,
59     * we need a real userId
60     */
61     const user = await User.findById(req.userId);
62     user.posts.push(post);
63     const savedUser = await user.save();
64     /**the tiny problem is that
65     * we are calling '.json()' method, not on the response object
66     * but on the result of the status method call
67     */
68     res.status(201).json({
69         message: 'Post created successfully!',
70         post: post,
71         creator: { _id: user._id, name: user.name }
72     });
73     return savedUser;
74 } catch (err) {
75     if (!err.statusCode) {
76         err.statusCode = 500;
77     }
78     next(err);
79 }
80 };
81
82 exports.getPost = async (req, res, next) => {
83     const postId = req.params.postId;
84     const post = await Post.findById(postId);
85     try {
86         if (!post) {
87             const error = new Error('Could not find post. ');
88             error.statusCode = 404;
89             throw error;
90         }
91         res.status(200).json({ message: 'Post fetched.', post: post });
92     } catch (err) {
93         if (!err.statusCode) {

```

```

94     err.statusCode = 500;
95   }
96   next(err);
97 }
98 };
99
100 exports.updatePost = async (req, res, next) => {
101   const postId = req.params.postId;
102   const errors = validationResult(req);
103   if (!errors.isEmpty()) {
104     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

```

107   }
108   const title = req.body.title;
109   const content = req.body.content;
110   let imageUrl = req.body.image;
111   if (req.file) {
112     imageUrl = req.file.path;
113   }
114   if (!imageUrl) {
115     const error = new Error('No file picked.');
```

error.statusCode = 422;

throw error;

```

118   }
119   try {
120     const post = await Post.findById(postId);
121     if (!post) {
122       const error = new Error('Could not find post.');
```

error.statusCode = 404;

throw error;

```

125     }
126     if (post.creator.toString() !== req.userId) {
127       const error = new Error('Not authorized!');
```

error.statusCode = 403;

throw error;

```

130     }
131     if (imageUrl !== post.imageUrl) {
132       clearImage(post.imageUrl);
133     }
134     post.title = title;
135     post.imageUrl = imageUrl;
136     post.content = content;
137     const result = await post.save();
138     res.status(200).json({ message: 'Post updated!', post: result });
139   } catch (err) {
140     if (!err.statusCode) {
141       err.statusCode = 500;
142     }
143     next(err);
144   }
145 };
146
147 exports.deletePost = async (req, res, next) => {
148   const postId = req.params.postId;
149   try {
```

```

150     const post = await Post.findById(postId);
151
152     if (!post) {
153         const error = new Error('Could not find post. ');
154         error.statusCode = 404;
155         throw error;
156     }
157     if (post.creator.toString() !== req.userId) {
158         const error = new Error('Not authorized!');
159         error.statusCode = 403;
160         throw error;
161     }
162     // Check logged in user
163     clearImage(post.imageUrl);
164     await Post.findByIdAndRemove(postId);
165
166     const user = await User.findById(req.userId);
167     user.posts.pull(postId);
168     await user.save();
169
170     res.status(200).json({ message: 'Deleted post.' });
171 } catch (err) {
172     if (!err.statusCode) {
173         err.statusCode = 500;
174     }
175     next(err);
176 }
177 };
178
179 const clearImage = filePath => {
180     filePath = path.join(__dirname, '..', filePath);
181     fs.unlink(filePath, err => console.log(err));
182 };
183

```

* Chapter 474: Wrap Up & Mastering Tests

- always ask yourself are you testing something that you are responsible for with your code regarding the status code. we don't need to test whether the status code is set on the response but if that exact status code you are looking for is correct that is something you test. is it status code 201 or 500
- if you have problem testing large functions, try splitting them in smaller more testable functions.