**Quote Keeper: Server Layer**

In this project, you will write the server tests for a web application that allows users to save and display a collection of their favorite quotes. We will call this app “Quote Keeper”.

The app currently has a feature level test which submits a quote, source, and who it is attributed to. It then asserts that it is displayed to the user.

Since the existing tests were solely at the feature level, it was unable to pass. We are diving into the server level. The steps below guide you through a TDD approach to implement the corresponding server behavior.

As you work through the tests, ensure that you are running npm test to check on your test status as you implement.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/12Complete**

Mark the tasks as complete by checking them off

**Quote Keeper Server Test**

**1.**

The scope of our test is that a user should be able to submit a quote and have it displayed back to the user upon submission. The feature version of this test is located at **test/features/user-visits-root-test.js**.

Run this test and observe the results.

Hint

Type npm test to run the test suite.

**2.**

Let’s add a server level test in **test/routes/index-test.js**.

Set up a describe block for '/', with a nested describe block for POST. Inside of this, place an it block with 'creates a new quote'. This block should have the async keyword since we’ll be making server requests.

Hint

describe('/', () => {

describe('POST', () => {

it('creates a new message', async () => {

});

});

});

**3.**

For the setup, declare and assign strings to variables for quote, attributed, and source.

Hint

const quote = 'Our deepest fear is not that we are inadequate. Our deepest fear is that we are powerful beyond measure.';

const attributed = 'Marianne Williamson';

const source = 'A Return to Love: Reflections on the Principles of A Course in Miracles';

**4.**

For the exercise, make a POST request to '/', submitting these variables. Make sure to include .type('form') to submit the data as a form.

Hint

const response = await request(app)

.post('/')

.type('form')

.send({quote, attributed, source});

**5.**

For our assertions, let’s check the status code is 200 and that the response contains the text of our quote, the attribution and the source.

Use the parseTextFromHTML helper. We are expecting this text to be located in a #quotes element.

Hint

assert.equal(response.status, 200);

assert.include(parseTextFromHTML(response.text, '#quotes'), quote);

assert.include(parseTextFromHTML(response.text, '#quotes'), attributed);

assert.include(parseTextFromHTML(response.text, '#quotes'), source);

**6.**

The status code assertion is passing, but the rest of the server level test is failing just like the corresponding feature level test. When the quote is submitted (via HTTP POST) the response does not include the submitted quote.

We’ve added Handlebars to the project to allow you to respond dynamically. You can see the current response is empty in **views/index.handlebars**.

Copy the current code from **public/index.html** into **views/index.handlebars**.

The test failure messages should be different now.

Hint

Now the right element is returned with the POST, but it is empty.

**7.**

Since we don’t have a database set up yet, for now, let’s respond with the submitted quote to the user. The data won’t be persistent, but it will demonstrate the proper behavior. (this would be refactored when we move to the model layer).

The code for handling the data POST is in **routes/index.js**. Within the handling for a POST to ‘/‘, capture the submitted data by destructuring req.body.

Hint

const {source, attributed, quote} = req.body;

**8.**

Now we need to render the template with this data. Again in **routes/index.js**, pass these values to the .render method.

Hint

res.render('index', {source, attributed, quote});

**9.**

Use Handlebars to render the quote within the #quotes section at the top. This can be done how you like; our test is just checking that the text exists in this section.

Hint

<section id="quotes">

{{quote}} - {{attributed}} - {{source}}

</section>

**10.**

Our test is now passing! However, it seems we have almost exactly the same code in **public/index.html** as we do in **views/index.handlebars**.

Lets do a refactor so we aren’t repeating ourselves.

Add an empty handler for a GET of ‘/‘ within **routes/index.js**.

Hint

router.get('/', (req, res) => {

});

**11.**

Our test should now be failing with a timeout since this route hasn’t been implemented. See if you can implement the code to get the test into the green.

Hint

router.get('/', (req, res) => {

res.render('index');

});

**12.**

Now that we have a passing test, we can remove our old implementation. Delete the contents of **public/index.html**.

Make sure the tests are still passing.

We’ve successfully implemented server behavior to pass both the feature and server level tests. However, this behavior isn’t quite right. The objective application should really save the data somewhere - it’s time to look at the model layer.