Create and Serve the Category Screen

30 minutes

Create And Serve the Category Screen

You'll now write the tests for the part of the application that shows the list of categories. That code, in **server.js** looks like this.

```
const filePath = path.join(__dirname, 'category-list-screen.html');
const template = await fs.promises.readFile(filePath, 'utf-8');
const html = mergeCategories(template, categories, 'li');
res.setHeader('Content-Type', 'text/html');
res.writeHead(200);
res.write(html);
```

You need to write tests for the function `mergeCategories()` for the portion that outputs the HTML for list items. Open the file **test/merge-categories-spec.js**. You will see

```
describe("mergeCategories()", () => {
  context("Using <li> tags", () => {
    const template = `
      <div>
        <ul>
          <!-- Content here -->
        </ul>
      </div>
    `;

    it("should return no <li>s for no categories", () => {
      expect.fail('please write this test');
    });

    it("should return a single <li> for one category", () => {
      expect.fail('please write this test');
    });

    it("should return an <li> for each category", () => {
      expect.fail('please write this test');
    });
  });

  // more code ...
```

The `context` block is for writing tests for when we use `mergeCategories()` and pass it `<li>` tags.

You will need to write tests in all the `it` blocks. Just replace the `expect.fail` calls with your own tests. (`expect.fail` is a chai assertion to force a spec to fail so we are using it for all the unwritten tests so that when you run `npm test` you will see all the tests you haven't written failing)

Open **merge-categories.js** to review the code before writing the tests.

The `mergeCategories` function takes a string through its `template` parameter, a list of strings through its `categories` parameter and an HTML tag through it's `tagName` parameter.

It then replaces the HTML comment `<!-- Content here ..>` with the newly created `<li>` tags (one for each catagory) and returns a new string of HTML.

Use the `template` variable that is available to you for these tests.

The first test

The first test reads

```
it("should return no <li>s for no categories", () => {
  expect.fail('please write this test');
});
```

Replace the `expect.fail` line with a test that properly follows the *Three As* of unit testing.

In the *arrange* section, you will need to create an empty array for the `categories` and store it in a variable. You will use the variable in the action.

In the *act* section, you will invoke the `mergeCategories` function with the `template` as the first argument, the variable that contains an empty array as the second argument, and the string 'li' for the tag name as the third argument. Store the return value in a variable.

In the *assert* section, assert that each of the following are true using the include assertion provided by Chai:

- To make sure that the method doesn't *remove* the wrong things
    - Assert that it contains the string " "
    - Assert that it contains the string " "
    - Assert that it contains the string "<ul>"
    - Assert that it contains the string "
"

- To make sure that the method doesn't *add* the wrong things
  - Assert that it does not contain the string "<li>"
  - Assert that it does not contain the string "
"

- To make sure it replaces what you expect it to replace
  - Assert that it does not contain the string "<!-- Content here -->"

Run the test to make sure it passes.

Here's what the test could look like.

```
it("should return no LIs for no categories", () => {
  const categories = [];
  const result = mergeCategories(template, categories, 'li');
  expect(result).to.contain('<div>');
  expect(result).to.contain('</div>');
  expect(result).to.contain('<ul>');
  expect(result).to.contain('</ul>');
  expect(result).to.not.contain('<li>');
  expect(result).to.not.contain('</li>');
});
```

Notice we are using `contain` *here instead of* `include`. `contain` *is an alias to* `include` *that chai provides, and it reads better here than* `include`.

The second test

The second test reads

```
it("should return a single <li> for one categories", () => {
  expect.fail('please write this test');
});
```

Replace the `expect.fail` line with a test that properly follows the *Three As* of unit testing.

In the *arrange* section, you will need to create an array for the `categories` argument that contains a single string and store it in a variable. You will use the variable in the action and the value that you typed in the assertion.

In the *act* section, you will invoke the `mergeCategories` function with the `template` as the first argument, the variable that contains the array with the single value as the second argument, and the string 'li' for the tag name as the third argument. Store the return value in a variable.

In the *assert* section, assert that each of the following are true using the include

assertion provided by Chai:

- To make sure that the method doesn't *remove* the wrong things
  - Assert that it contains the string " "
  - Assert that it contains the string " "
  - Assert that it contains the string "<ul>"
  - Assert that it contains the string "
"
- To make sure that the method *adds* the right things
  - Assert that it does contain the string "
your string here
" where "your string here" is the single value that you placed in the array
- To make sure it replaces what you expect it to replace
  - Assert that it does not contain the string "<!-- Content here -->"

Run the test to make sure it passes.

Here's what the test could look like.

```
it("should return a single LI for one categories", () => {
  const categories = ['Cat 1'];
  const result = mergeCategories(template, categories, 'li');
  expect(result).to.contain('<div>');
  expect(result).to.contain('</div>');
  expect(result).to.contain('<ul>');
  expect(result).to.contain('</ul>');
  expect(result).to.contain('<li>Cat 1</li>');
});
```

The third test

The third test reads

```
it("should return an <li> for each category", () => {
  expect.fail('please write this test');
});
```

Replace the `expect.fail` line with a test that properly follows the *Three As* of unit testing.

In the *arrange* section, you will need to create an array for the `categories` argument that contains multiple strings and store it in a variable. You will use the variable in the action and the values that you typed in the assertion.

In the *act* section, you will invoke the `mergeCategories` function with the `template` as the first argument, the variable that contains the array with the multiple values as the second argument, and the string 'li' for the tag name as the third argument. Store the return value in a variable.

In the *assert* section, assert that each of the following are true using the include assertion provided by Chai:

- To make sure that the method doesn't *remove* the wrong things
    - Assert that it contains the string " "
    - Assert that it contains the string " "
    - Assert that it contains the string "<ul>"
    - Assert that it contains the string "
"

- To make sure that the method *adds* the right things, for *each* of the values that you put in your categories array:
    - Assert that it does contain the string "
value n
" where "value n" is one of the values in your array

- To make sure it replaces what you expect it to replace
    - Assert that it does not contain the string "<!-- Content here -->"

Run the test to make sure it passes.

Here's what that code could look like.

```
it("should return an LI for each category", () => {
  const categories = ['Cat 1', 'Cat 2', 'Cat 3'];
  const result = mergeCategories(template, categories, 'li');
  expect(result).to.contain('<div>');
  expect(result).to.contain('</div>');
  expect(result).to.contain('<ul>');
  expect(result).to.contain('</ul>');
  expect(result).to.contain('<li>Cat 1</li>');
  expect(result).to.contain('<li>Cat 2</li>');
  expect(result).to.contain('<li>Cat 3</li>');
});
```

You have won this round!

✓ Mark As Complete

Finished with this task? Click **Mark as Complete** to continue to the next page!