Lab - Unit Testing Angular 14 Application with Jasmine & Karma

We will learn how to write unit tests for services and components in an Angular app using the Jasmine framework and Karma (JavaScript Test Runner).

We always ship our final product after thorough testing, It is a fundamental step in software development, and it can be done via various testing methods.

There are many methods to test our code, such as automated testing, manual testing, performance testing, automated unit testing. These testing methods are chosen as per the testing requirements of an application.

**There are 3 types of tests:**

* Unit tests
* Integration tests
* End-to-End (e2e) tests

In this lab, we will focus on unit testing with Jasmine and Karma.

*Jasmine is an open-source behavior-driven testing framework crafted by Pivotal Labs. It is installed via Angular CLI and offers the hassle-free testing experience to test an Angular and JavaScript code.*

Jasmine provides several valuable functions to write tests. Here are the main Jasmine methods:

* **it()**: Declaration of a particular test
* **describe()**: It’s a suite of tests
* **expect()**: Expect some value in true form

Writing tests with Jasmine and Karma is very easy, so, we will create a basic Angular application, then create a simple Angular component and service. Then, we will write some test cases for Angular component, and also unit test a service with HttpTestingController.

Let’s get started testing an Angular component with the Jasmine test framework.

## ****Karma in Angular****

Karma is a test runner tool, it creates a browser instance, and runs tests to provide the expected results.

The benefit of using Karma is that it can be operated via command line and It refreshes the browser automatically whenever we make even minor changes in our app.

## ****Setting Up Angular App****

Next, install the Angular project by running the below command:



Head over to the project folder by using the following command:



Start the app on the browser:



Now, you can view your app on the browser on the following port: [localhost:4200](http://localhost:4200/).

## ****Angular Component Testing Example****

An Angular component is a an HTML template and a TypeScript class. So, to test a component first, we need to create a component.

Let’s name it pizza and run the below command to create the component.



Above command has created a **pizza** folder, and inside the pizza folder create a **title** variable and assign some value to it.

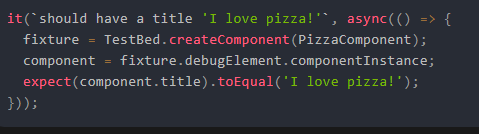


You can see there is another file created **pizza.component.spec.ts** and this is a test file which is responsible for testing in Angular and testing file looks like this.



Writing tests in Angular is easy, now we are going to write a simple test within the **describe()** function.





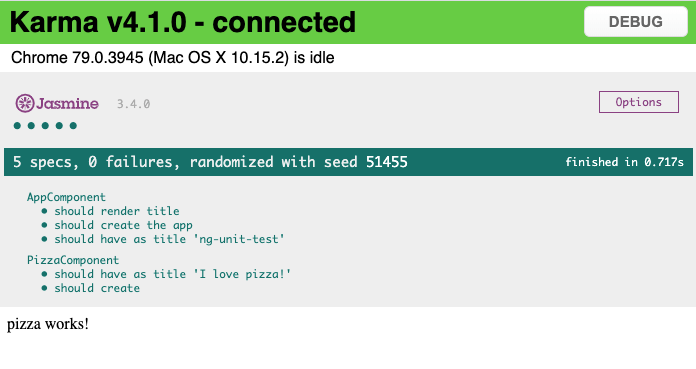
You need to use the **ng test** command to start testing an Angular component.

Above command built the app in watch mode and launched the karma.

The **ng test** command opened the watch mode in karma.

We added some content in the pizza component. Then we created the pizza component instance to verify its properties and functions inside of it for testing purpose.

Now, as you can see in the screenshot 5 specs and 0 failures, this means we passed the test for pizza as well as for AppComponent’s properties.



The fixture creates a wrapper around a component instance, The fixture **TestBed.createComponent()** method allows accessing the component and its template.

## ****Unit Testing with Angular Service with HttpClient & HttpTestingController API****

Next, we are going to look at how to unit test a service that handles the http requests in Angular.

Run the following command to create service inside the shared folder:



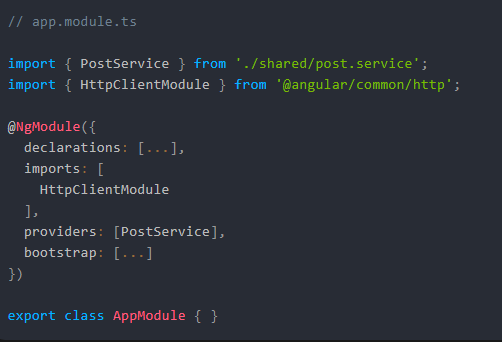
We are using free REST API from [JSONPlaceholder](https://jsonplaceholder.typicode.com/" \t "https://www.positronx.io/angular-unit-testing-application-with-jasmine-karma/_blank), a big thanks to them for providing such a beautiful collection of REST APIs.

After running the above command, we have got the following files:

***app/shared/post.service.spec.ts***

***app/shared/post.service.ts***

Next, import and register the PostService in app.module.ts file, also import and register **HttpClientModule** in the main app module file.



Add the following code in the post.service.ts file, The api is called via getPosts() method and returns post list Observable and we subscribe to it.

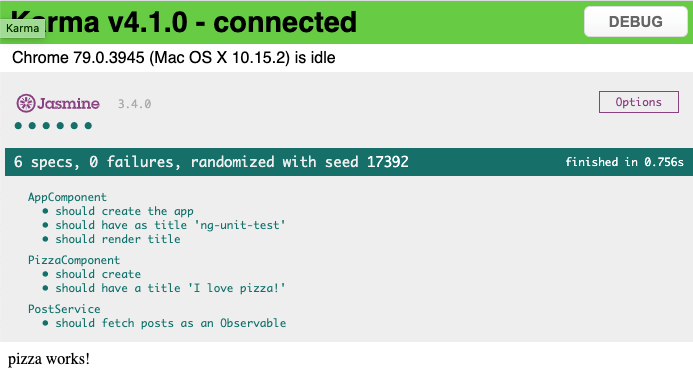


**HttpTestingController** service are beneficial in mocking the HTTP requests, and this process can not be followed without taking the help of **HttpClientTestingModule**.

In this lab, we are going to write a unit test for mocking the HTTP GET request by taking the help of the **HttpTestingController** service.

Add the following code in the **shared/post.service.spec.ts** file.

**See this downloaded folder for the code.**



Import the **HttpClientTestingModule** and **HttpTestingController** and inject inside the **TestBed** method and also set up the [Angular service](https://www.positronx.io/angular-service-tutorial-with-example/), which we are going to unit test.

We also declared the **Verify** method via **HttpTestingController**, and it makes sure that there are no unmatched requests that are outstanding.

We defined the posts data in the postItem array and mocking the requests using the **HttpTestingController**. We subscribe to Observable returned from HTTP GET request and set up the expectations using the response.

### Conclusion

Finally, Unit Testing Angular 12 Application with Jasmine & Karma lab is over. In this lab, we covered the following topics:

* What are the types of tests?
* What is unit testing?
* How to unit test an Angular component?
* How to test an Angular application from scratch?
* How to unit test an Angular Service with HttpClient and HttpTestingController?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*