Unit and Integration tests for Angular components. Part 1 out of 3.

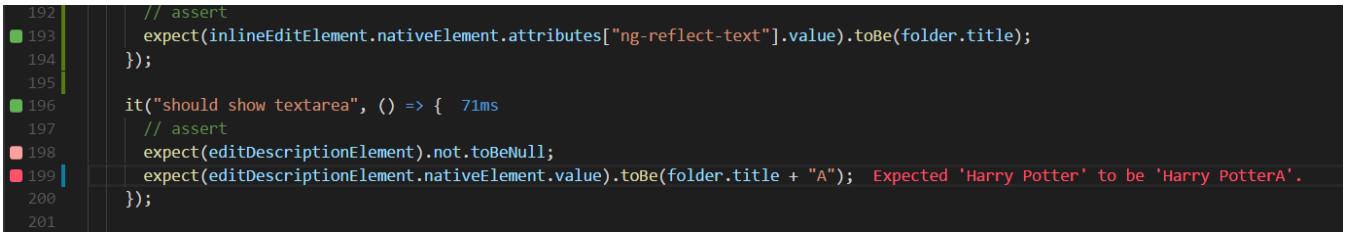
Anastasia Dvorkina
May 8, 2018 · 4 min read

As application grows we ask ourselves how to structure the code so that it's easier to understand and maintain. Unit tests sometimes also become the hell on Earth if you don't define the clear vision on how they should be written.

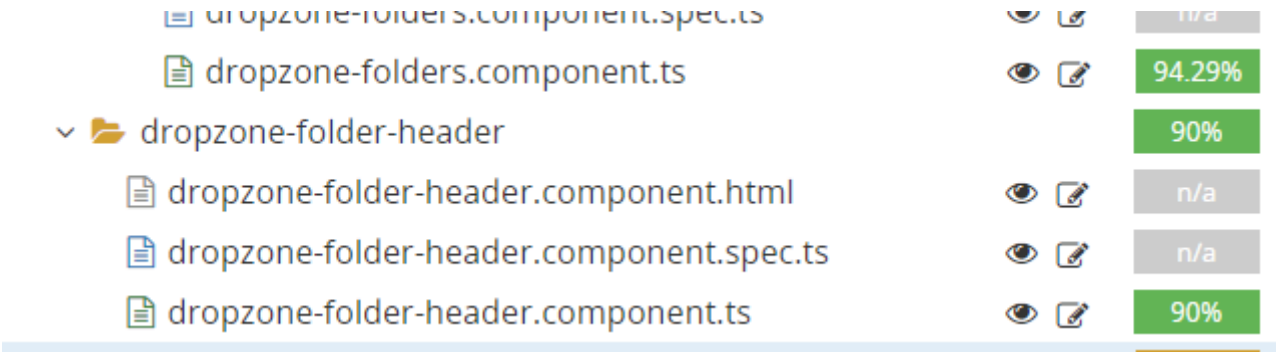
Photo by Samuel Zeller on Unsplash

Technological stack of our team is pretty usual I guess: Karma + Jasmine.

Both Karma and Jasmine are cool and nice to use, but there is one tool in our daily life that makes my heart melt—*wallaby* (https://wallabyjs.com/). You can read more about it on their website but shortly wallaby is an integrated test runner which provides instant feedback on your tests, so TDD in JavaScript has never been so easy. Another benefit of using wallaby is having constant feedback on the test coverage.



VS Code integration



wallaby coverage app

Wallaby updates the report instantly and gives you the detailed information on each file. Wonderful tool, definitely recommend!

There are 2 types of tests that developers normally write:

1. **unit tests**: testing one thing at a time, tests should not care about the points of interaction between different units of the application
2. **integration tests**: checking that multiples units are interacting with each other correctly

U**nit tests** for Angular components can be split into 2 categories:

1. **isolated tests**: when we just test component functions.
2. **shallow unit tests**: testing components template without rendering its children. It may be achieved by simply not registering child components inside of the TestBed function.

Now let's talk about these types of tests in a little bit more details.

Isolated unit tests

Let's assume we have following function in folder.component.ts

```
1    public selectFolder(folder: IFolder): void {
2      this.selectedFolder = folder;
3    }
```

folder.component.ts hosted with ❤ by GitHub view raw

folder.component.ts hosted with ❤ by GitHub view raw

Its unit test where we are checking if the selectedFolder is set correctly will look smth like that:

```
1    describe("selectFolder", () => {
2      it("should update selectedFolder property with passed folder", () => {
3        // arrange
4        component.selectedFolder = undefined;
5        let folder: IFolder = new Folder(1, "Harry Potter");
6
7        // act
8        component.selectFolder(folder);
9
10       // assert
11       expect(component.selectedFolder).toBe(folder);
12     });
13   });
```

folders.component.spec.ts hosted with ❤ by **GitHub** **view raw**

In this case we don't touch the template at all and concentrate only on the inner functionality of the component.

Shallow unit tests

Template shallow tests allow us to test *Input()* and *Output()* properties of child components.

testing input properties

Let's assume we have the following simple *folder-name.component.html* that has a child *<custom-inline-edit/>* component in it.

```
1  <span *ngIf="folder" class="folder-name">
2              <custom-inline-edit [text]="folderTitle"
3                          (editConfirmed)="editTitleConfirmed($event)"
4                          (canceled)="editFolderCanceled.emit($event)">
5              </custom-inline-edit>
6  </span>
```

folder-name.component.html hosted with ❤ by **GitHub** **view raw**

Let's test if the input property ***text*** is set correctly in the HTML template.

```
it("should set input property 'text' equal to folder title in <custom-inline-edit/>", (
    // arrange
    component.folder = folder;
    fixture.detectChanges();

    // act
    let inlineEditElement: DebugElement = editDescriptionElement = fixture.debugElement
        By.css("custom-inline-edit"));

    // assert
    expect(inlineEditElement.nativeElement.attributes["ng-reflect-text"].value).
        toBe(folder.title);
  });
```

folder-name.component.spec.ts hosted with ❤ by GitHub view raw

So we've inspected the child element to have special Angular attribute populated with the expected value.

testing output events

As we remember in the HTML of the folder-name.component.html we've had the *Output()* event *(editConfirmed)*

```
<span *ngIf="folder" class="folder-name">
            <custom-inline-edit [text]="folderTitle"
                        (editConfirmed)="editTitleConfirmed($event)"
                        (canceled)="editFolderCanceled.emit($event)">
            </custom-inline-edit>
</span>
```

folder-name.component.html hosted with ❤ by GitHub view raw

And here is the code of this event in *folder-name.component.ts*

```
  public editTitleConfirmed(title: string): void {
    if (!title) {
      return;
    }
    this.folderUpdated.emit({ ...this.folder, title: title });
  }
```

folder-name.component.ts hosted with ❤ by GitHub view raw

To test the correct binding of the event and inner functionality we'll use *triggerEventHandler* and pass there *Output*() property name:

```
it("should emit folder with the new title on editTitleConfiremd", () => {
    // arrange
    component.folder.title = "Harry Potter";
    fixture.detectChanges();
    spyOn(component.folderUpdated, "emit");

    // act
    let inlineEditElement: DebugElement = editDescriptionElement = fixture.debugElement
        By.css("custom-inline-edit"));
    inlineEditElement.triggerEventHandler("editConfirmed", "Albus Dumbledore");

    // assert
    let emittedFolder: IFolder = component.folderUpdated.emit.calls.mostRecent().args[
    expect(emittedFolder.title).toEqual("Albus Dumbledore");
});
```

folder-name.component.spec.ts hosted with ❤ by **GitHub** **view raw**

When using this technique we'll be able to proper unit test child bindings in the parent component.

. . .

Integration tests

Let's test same functionality of populating *<custom-inline-edit/>* with one value and changing it into another withing one integration test.

```
it("should emit folder with the new title on Enter key", fakeAsync(() => {
    // arrange
    spyOn(component.folderUpdated, "emit");

    // act
    let inlineEditInputDebugElement: DebugElement = fixture.debugElement.query(
        By.css("custom-inline-edit #title-input"));
    let inlineEditInputElement: any = editDescriptionElement.nativeElement;

    inlineEditInputElement.value = "Albus Dumbledore";
    inlineEditInputElement.dispatchEvent(new Event("input"));
    inlineEditInputDebugElement.triggerEventHandler("keydown.enter", null);
```

```
13
14      fixture.detectChanges();
15
16      // assert
17      let emittedFolder: IFolder = (component.folderUpdated.emit as any).calls.mostRecent
18      expect(emittedFolder.title).toEqual("Albus Dumbledore");
19  });
```

folder-name.component.integration-spect.ts hosted with ❤ by GitHub view raw

folder-name.component.spec.ts

Here we've accessed the input field inside of the child input component, changed its value and trigger Enter keyboard event so that *Output()* event *(editConfirmed)* was triggered.

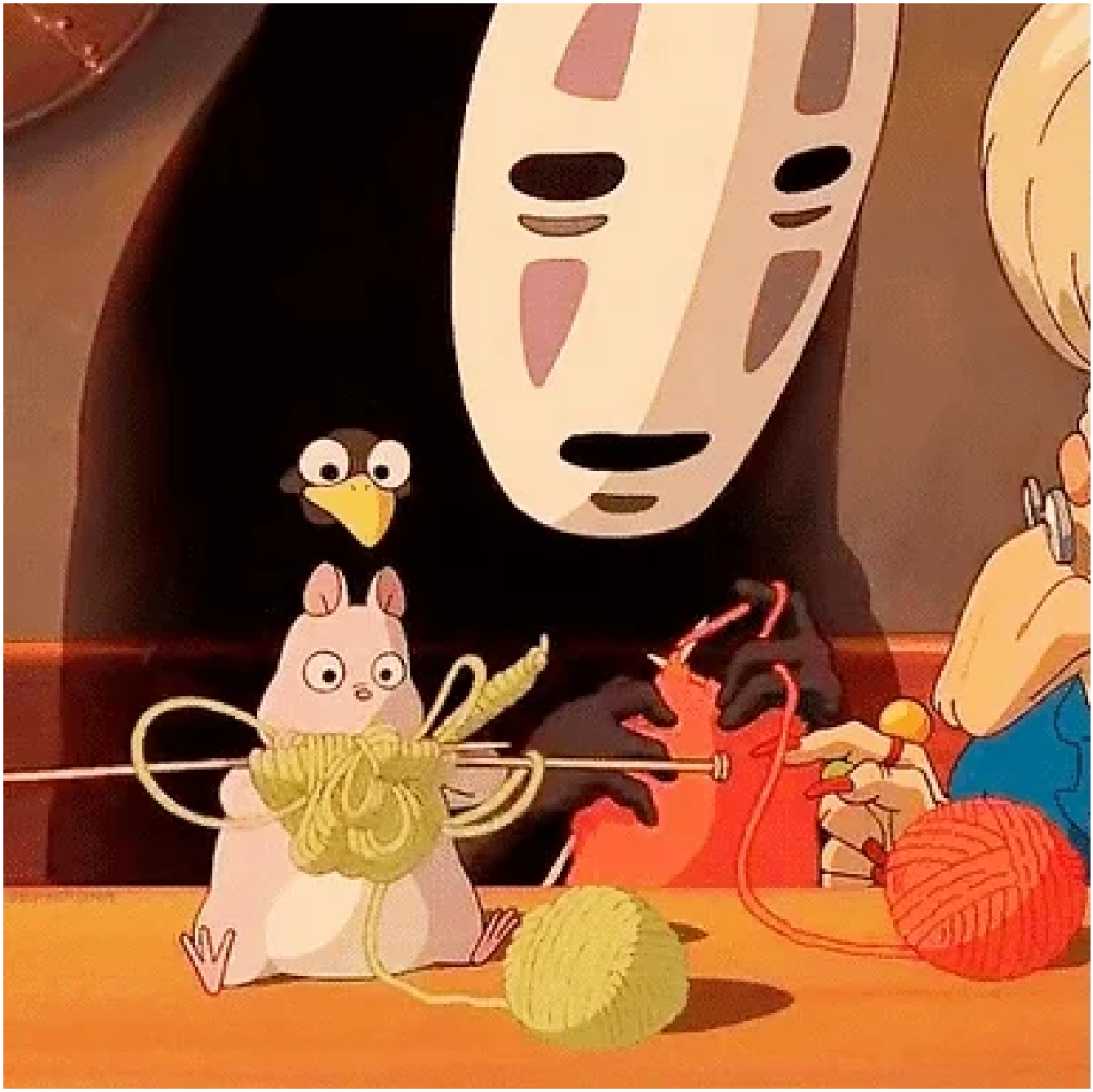
That's the difference for me between **isolated** unit tests, **shallow** unit tests and **integration** tests. Let's see what is the proposed structure of tests by Angular team:

```
- component-name
    - component-name.component.html
    - component-name.component.ts
    - component-name.component.spec.ts
```

So we have to fit into one single spec file:

1. **isolated** unit tests
2. **shallow** unit tests
3. **integration** tests

Well, actually for components I would not write isolated tests at all. Imagine HTML template to be public method and the component to be inner private implementation. So we should write just shallow and integrations tests.



carefully working on tests

I personally don't like big files so I would rather split integration and shallow tests into 2 different files with a shared configuration (we'll talk about it later).

. . .

The questions that are still open for me:

1. how deep integration tests should be?
2. how to write tests DRY if keeping integration and unit tests separate?

To be continued…

. . .

Links that were helpful for me during this investigation:

Three Ways to Test Angular Components

Victor Savkin is a co-founder of nrwl.io, providing Angular consulting to enterprise teams. He was...

vsavkin.com

Nice lecture by Julie Ralph: https://www.youtube.com/watch?v=f493Xf0F2yU

with the repository mentioned in the video: https://github.com/juliemr/angularconnect-2016/blob/master/src/app/game.spec.ts

Unit and integration testing in an Angular wonderland

In my previous blog, I described the possibilities of layered test automation in an Angular 4...

www.capgemini.com

Some of coding guidelines for testing of our team can be found here: https://github.com/advorkina/guide-to-javascript-unit-tests/blob/master/README.md

JavaScript Angular Angular2 Testing Jasmine