Univerzitet u Beogradu
Fakultet organizacionih nauka
Laboratorija za softversko inženjerstvo

Opis alata i tehnika za testiranje Angular okvira

Mentor:

prof dr. Saša D. Lazarević

Studenti:

Katarina Simić 2020/3701
Aleksa Pavlović 2020/3709

Beograd, 2021.

Sadržaj

Testiranje softvera....3
Angular....3
Unit testovi u Angularu....3
Mock-ovanje....4
Alati za unit testiranje u Angularu....4
Postavka i pisanje unit testa....5
Pokretanje testa i čitanje rezultata....8
Plitki (Shallow) inetgracioni testovi....10
Literatura....15

Testiranje softvera

Testiranje koda je neizostavan deo kvalitetnog softvera, a ono prvo što korisnici vide, pa čak i male greške mogu dovesti do toga da korisnici imaju manje poverenja u određeni brend [1].

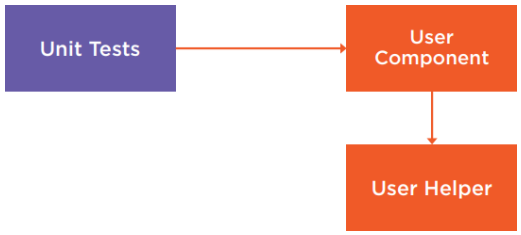
Piramida testiranja predstavlja model koji olakšava odluku o tome kako dati prednost različitim vrstama automatizovanog testiranja u aplikaciji. Na kraju, kada su na teorijskom nivou istražene sve mogućnosti, treba započneti sa testiranjem u praksi [1].

Angular

Angular je savremeni okvir i kao takav podržava pisanje testova i ima svoje specifilnosti. Jednom kreiran projekat uključuje sve potrebne alate za testiranje. Međutim, takođe postoji fleksibilnost i nije nužno koristiti okvire i alate koji su podrazumevano uključeni u Angular, ako postoji preferencija za nečim drugim [2].

Unit testovi u Angularu

Unit testiranje je jednako važno kao i razvoj projekta u današnje vreme i ono postaje sastavni deo razvoja. To zapravo poboljšava kvalitet koda, projekta i samopouzdanje čitavog tima. Unit testovi su napisani u okviru ***Jasmine***, a izvršava ih ***karma***, ***Test runner*** i izvršavaju se u pregledaču. Ponekad se pišu testovi i pre nego što se počne sa razvojem, što se naziva ***Test Driven Development (TDD)*** [1].



Slika 1

Uzima se samo jedna jedinica koda, ***User Component***, i pišu testovi za tu jedinicu koda. Unit testiranje je obično vrsta testiranja koja se najviše koristi, i generalno u razvoju, pišu se više unit testova nego što se piše bilo koja druga vrsta testa. Nekada nije baš najjasnije šta je tačno jedna jedinica koju treba testirati [2]. Na primer, ako bismo imali klasu User Helper, mogli bismo ovu komponentu uz User Component jednom jedinicom koda i zajedno ih testirati.

Angular zapravo ima nekoliko različitih vrsta unit testova i važno je znati šta je svaki od njih i kako se međusobno razlikuju.

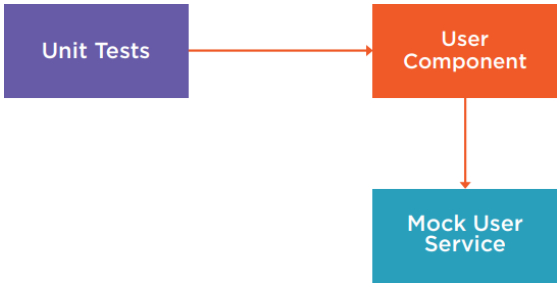
Osnovni unit test je **izolovani** test. U izolovanom testu se jednostavno vežba jedna jedinica koda, bilo klasa komponente, bilo klasa usluge ili pipeline-a, ta se klasa konstruiše ručno i daju joj se parametre konstrukcije [2].

Test integracije je malo složeniji. U testu integracije se zapravo kreira modul u koji se stavlja samo kod koji će se testirati, uglavnom samo jedna komponenta, ali to se zapravo testira u kontekstu Angular modula. Ovo se koristi da bismo mogli da testiramo komponentu pomoću njenog templejta. Postoje dve vrste testova integracije podržane u Angularu, plitke (shallow) integracije, gde testiramo samo jednu komponentu i testovi duboke (deep) integracije. Razlika je u tome što mnoge komponente zapravo imaju podređene komponente. Ponekad je poželjno testirati i roditeljsku komponentu i podređenu komponentu, ali i kako rade zajedno, što čini jedan dubok test integracije [2].

Mock-ovanje

Jedan od vrlo važnih koncepata u unit testiranju je mock-ovanje koje omogućava da testeri i programeri budu sigurni da istovremeno testiraju samo jednu jedinicu koda. Klasa uglavnom ne radi izolovano, već na klasa ili komponenata ima zavisnosti [1].

U primeru iznad, User Component komponenta koristi User Helper komponentu kroz Dependency Injection princip. Kada se pišu unit testovi, testira se User Componentkomponent, ali ne i User Helper komponenta. Umesto korišćenja stvarne User Component komponente, kreira se lažna (mock-ovana) komponenta. Lažna je klasa koja izgleda kao prava klasa, ali je moguće kontrolisati šta radi i šta njene metode vraćaju.



Slika 2

Alati za unit testiranje u Angularu

Alati koji se najviše koristite pri testiranju u Angularu su podrazumevani alati koji se koriste prilikom testiranja aplikacije Angular [3].

CLI postavlja sva neophodna podešavanja za testiranje za i koristi dva različita alata [3]:

- **Karma** - pokretač testa, to je ono što zapravo izvršava testove u pregledaču

- **Jasmin** - alat koji se koristi za kreiranje mock-ova i to je alat koji se koristi kako bi testeri i programeri bili sigurni da testovi rade onako kako se očekuje.

Postoji još nekoliko alata za unit testiranje koji su dostupni za jedinstveno testiranje sa Angular-om [3]:

- Postoji vrlo popularna biblioteka pod nazivom ***Jest*** koju je **Facebook** objavio i zaista je popularna u drugim okvirima, ali se može koristiti i sa Angular-om.
- **Mocha i Chai** su zamena za Jasmin i lako ih je podesiti i zameniti Jasmin.
- **Sinon** je specijalizovana biblioteka za mock-ovanje I koristi se ako se Jasmin ne pokaže kao dovoljno dobar za mock-ovanje.

Još neki popularni alati su [3]:

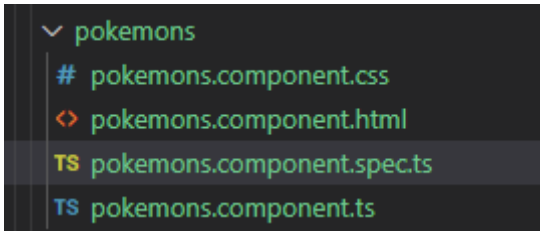
- **TestDouble**
- **Vallabi**
- **Cipress**

Postavka i pisanje unit testa

Prvo će biti prikazano kako izgleda aplikacija nakon pokretanja npm start komande kako bi se stekla šira slika o sistemu za koji se pišu testovi:

Pre nego što se započne sa pisanjem testova potrebno je pokrenuti **npm-test** u okviru Angluar CLI-ja.

Unutar direktorijuma aplikacija potrebno je kreirati novi fajl sa ekstenzijom **.spec. ts** . Specifikacija je važna i to je ono što alatu za unit testiranje, odnoso Karmi govori da je ovo fajl za testiranje [4]. Spec je skraćenica za specifikaciju, to je uobičajena reč koja se koristi prilikom pisanja unit testova, tako da treba obraiti pažnju da se svi unit testovi završavaju sa **.spec.ts**.



Slika 3

Početak pisanja testa podrazumeva opisivanje fukcije. Ovo je funkcionalnost **Jasmine**-a. **Jasmine** omogućava grupisanje testova [4].

Describe funkcija ima dva parametra: prvi je string, a drugi je funkcija povratnog poziva (callback) koja će sadržati naše testove [4]:

```
describe('PokemonsComponent', () => {
  let component: PokemonsComponent;
  let POKEMONS;
  let mockPokemonService;
```

Slika 4

U okviru describe funkcije naći će se sva logika testiranja kao i sva neophodna mock-ovanja.

```
describe('PokemonsComponent', () => {
  let component: PokemonsComponent;
  let POKEMONS;
  let mockPokemonService;

  beforeEach(() => {
    POKEMONS = [
        { id: 1, name: 'Bulbasaur', strength: 10 },
        { id: 2, name: 'Chikorita', strength: 15 },
        { id: 3, name: 'Togepi', strength: 18 },
    ]
    mockPokemonService = jasmine.createSpyObj(['getPokemons', 'addPokemon', 'delete
Pokemon']);
    component = new PokemonsComponent(mockPokemonService);
  })
```

Slika 5

Metoda beforeEach() resetuje stanje testa i na taj način obezbeđuje da svaki test radi sa podacima i postavkama koji su konfigurisani u ovoj metodi [4]. Na ovaj način prethodni testovi ne mogu imati efekat na izvršavanje budućih testova. Na primer, ukoliko test metoda delete() obriše jedan element liste, metoda test update() će raditi sa inicijalnom listom, a ne listom koja je nastala kada se test metoda delete() izvršila.

U Angularu se takođe prati Act, Arrange, Assert patern.

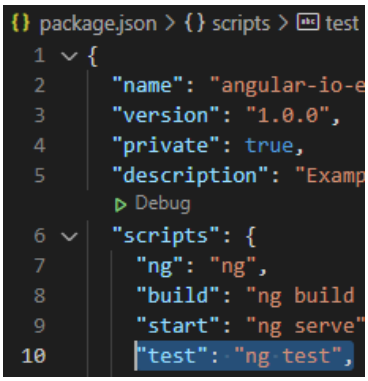
```
describe('delete', () => {

    it('should remove the indicated pokemon from the pokemons list', () => {
      //act
      mockPokemonService.deletePokemon.and.returnValue(of(true))
      component.pokemons = POKEMONS;
      //arange
      component.delete(POKEMONS[2]);
      //assert
      expect(component.pokemons.length).toBe(2);
    })
```

Slika 6

Pokretanje testa i čitanje rezultata

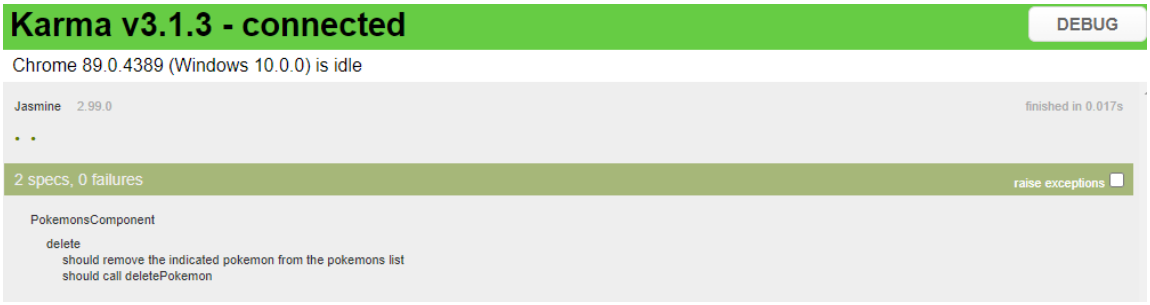
U okviru fajla package.json može se pogledati koju komandu treba koristiti za pokretanje testa:



Slika 7

Moguće je pokrenuti i test komandom npm test.

Nakon pokretanja testa čitaju se rezultati:



Slika 8

```
Chrome 89.0.4389 (Windows 10.0.0): Executed 2 of 2 SUCCESS (0.021 secs / 0.011 secs)
TOTAL: 2 SUCCESS
TOTAL: 2 SUCCESS
```

Slika 9

Spec 2 označava da su se izvršila dva testa koja su prošla uspešno: 0 failures 😊

Primer testova i odgovarajučeg servisa:

```
describe('addPokemon', () => {

    it('should add pokemon in the pokemons list', () => {
      //act
      mockPokemonService.addPokemon.and.returnValue(of(true))
      component.pokemons = POKEMONS;
      //arange
      component.add('Pikachu',20);
      //assert
      expect(component.pokemons.length).toBe(4);
    })
```

Slika 10

```
import { Component, OnInit } from '@angular/core';

import { Pokemon } from '../pokemon';
import { PokemonService } from '../pokemon.service';

@Component({
  selector: 'app-pokemons',
  templateUrl: './pokemons.component.html',
  styleUrls: ['./pokemons.component.css']
})
export class PokemonsComponent implements OnInit {
  pokemons: Pokemon[];

  constructor(private PokemonService: PokemonService) { }

  ngOnInit() {
    this.getPokemons();
  }

  getPokemons(): void {
    this.PokemonService.getPokemons()
    .subscribe(Pokemones => this.pokemons = Pokemones);
  }
```

```
  add(name: string, strength: number): void {
    name = name.trim();
    if (!name) { return; }
    this.PokemonService.addPokemon({ name, strength } as Pokemon)
      .subscribe(Pokemon => {
        this.pokemons.push(Pokemon);
      });
  }

  delete(Pokemon: Pokemon): void {
    this.pokemons = this.pokemons.filter(h => h !== Pokemon);
    this.PokemonService.deletePokemon(Pokemon).subscribe();
  }

}
```

Slika 11

Plitki (Shallow) inetgracioni testovi

U nastavku će biti opisani plitki testovi integracije, što znači da će se testirati samo jednu komponentu i nijednu njenu podređenu komponentu ili direktivu [4]. Kako bi se naznačilo da se radi o shallow testu, fajl treba kreirati u formatu **naziv_komponente.shallow.spec**.

Budući da se radi o testiranju komponente shallow testom, u okviru describe funkcije treba to i naznačiti. Kao i kod unit testova i u okviru ovakvih tipova testova treba setovati **beforeEach**.

TestBed je ono što omogućava testiranje i komponente i njenog templejta koji rade zajedno [5]. TestBed ima nekoliko različitih metoda, od kojih se izdvaja **configureTestingModule** koja ima jedan parametar - objekat koji se tačno podudara sa izgledom kada se kreira modul aplikacije. Pozivanjem metode **createComponent**, konstruiše se komponenta PokemonComponent, a ova fukncija zapravo
vraća ComponentFixture.

ComponentFixture je u osnovi omotač za komponentu koja se koristi u testiranju i ima nekoliko drugih svojstava, više od onoga što sama komponenta ima [5]. Jednom kada se setuje fixture promenljiva, moguće je pristupiti pristupiti npr. instanci komponente., metodi onDeleteClick…

NO_ERRORS_SCHEMA će reći Angular-u da za ovaj modul ne greši ako naiđe na nepoznati atribut ili nepoznati element u HTML-u (npr. router-link), već samo da ga zanemari [5]. Ovaj pristup treba koristiti isključivo kada komponenta nema svoju podređenu komponentu.

```
import { TestBed, ComponentFixture } from "@angular/core/testing";
import { NO_ERRORS_SCHEMA } from "@angular/core";
import { By } from "@angular/platform-browser";
import { PokemonComponent } from "./pokemon.component";

describe('PokemonComponent (shallow tests)', () => {
  let fixture: ComponentFixture<PokemonComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [PokemonComponent],
      schemas: [NO_ERRORS_SCHEMA]
    });
    fixture = TestBed.createComponent(PokemonComponent);
  });
```

Slika 12

```
  it('should have the correct pokemon', () => {
    fixture.componentInstance.pokemon = { id: 1, name: 'Togepi', strength: 3};

    expect(fixture.componentInstance.pokemon.name).toEqual('Togepi');
  });

  it('should render the pokemon name in an anchor tag', () => {
    fixture.componentInstance.pokemon = { id: 1, name: 'Togepi', strength: 3};
    fixture.detectChanges();

    let deA = fixture.debugElement.query(By.css('a'));
    expect(deA.nativeElement.textContent).toContain('Togepi');

    expect(fixture.nativeElement.querySelector('a').textContent).toContain('Togepi'
);
  })
```

Slika 13

Pozivanjem detectChanges metode komponenti govori da izvrši otkrivanje promena i ažurira sve veze koje mogu postojati na komponenti [5]. U ovom slučaju postoji vezivanje identiteta i vezivanje naziva. Snaga integracionog testa je testiranje stvarnog templejta. U testu iznad je zapravo uhvaćen DOM element, a zatim je usledila provera da li je nešto u DOM-u tačno, a to je nešto što nije moguće uraditi u izolovanom testu [5].

Svojstvo **debugElement** ima funkciju upita: query, queryAll i queryAllNodes. Prve dve funkcije se obično koriste sa elementom za otklanjanje grešaka. Upit omogućava da se dobije referencu na jedan čvor, a queryAll omogućava dobijanje referenci na više čvorova. Kada se radi sa komponentom, tražimo se od fixture da vrati komponentu Instance, ali ponekad ako se radi sa više komponenti, potrebno je znati komponentu kojoj dati element pripada [5]. Ovo je još jedan scenario gde **debugElement** nailazi svoju primenu.

Neki shallow integracioni testovi zahtevaju mock-ovanje odgovarajućeg servisa. Sledi primer testa koji sada testira PokemonsComponent koja ima svoju podređenu komponentu:

```
import { ComponentFixture, TestBed } from "@angular/core/testing";
import { PokemonsComponent } from "./pokemons.component";
import { Component, Input } from "@angular/core";
import { PokemonService } from "../pokemon.service";
import { of } from "rxjs";
import { Pokemon } from "../pokemon";
import { By } from "@angular/platform-browser";

describe('PokemonesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<PokemonsComponent>;
  let mockPokemonService;
  let POKEMONS;

  @Component({
    selector: 'app-pokemon',
    template: '<div></div>',
  })
  class FakePokemonComponent {
    @Input() pokemon: Pokemon;
  }

  beforeEach(() => {
    POKEMONS = [
      {id:1, name: 'Togepi', strength: 8},
      {id:2, name: 'Pikachu', strength: 24},
      {id:3, name: 'Chikorita', strength: 55}
    ]
    mockPokemonService = jasmine.createSpyObj(['getPokemons', 'addPokemon', 'delete
Pokemon']);

    TestBed.configureTestingModule({
      declarations: [
        PokemonsComponent,
        FakePokemonComponent
      ],
      providers: [
        { provide: PokemonService, useValue: mockPokemonService }
      ],
    })
    fixture = TestBed.createComponent(PokemonsComponent);
  });

  it('should set pokemons correctly from the service', () => {
    mockPokemonService.getPokemons.and.returnValue(of(POKEMONS))
    fixture.detectChanges();
```

```
    expect(fixture.componentInstance.pokemons.length).toBe(3);
  });

  it('should create one li for each pokemon', () => {
    mockPokemonService.getPokemons.and.returnValue(of(POKEMONS))
    fixture.detectChanges();

    expect(fixture.debugElement.queryAll(By.css('li')).length).toBe(3);
  })
})
```

Slika 14

```
import { TestBed, ComponentFixture } from "@angular/core/testing";
import { PokemonDetailComponent } from "./pokemon-detail.component";
import { ActivatedRoute } from "@angular/router";
import { PokemonService } from "../pokemon.service";
import { Location } from '@angular/common';
import { of } from "rxjs";
import { FormsModule } from "@angular/forms";

describe('PokemonDetailComponent', () => {
  let fixture: ComponentFixture<PokemonDetailComponent>;
  let mockActivatedRoute, mockPokemonService, mockLocation;

  beforeEach(() => {
    mockActivatedRoute = {
      snapshot: { paramMap: { get: () => { return '3'; }}}
    }
    mockPokemonService = jasmine.createSpyObj(['getPokemon', 'updatePokemon']);
    mockLocation = jasmine.createSpyObj(['back']);

    TestBed.configureTestingModule({
      imports: [FormsModule],
      declarations: [PokemonDetailComponent],
      providers: [
        {provide: ActivatedRoute, useValue: mockActivatedRoute},
        {provide: PokemonService, useValue: mockPokemonService},
        {provide: Location, useValue: mockLocation},
      ]
    });
    fixture = TestBed.createComponent(PokemonDetailComponent);

    mockPokemonService.getPokemon.and.returnValue(of({id: 3, name: 'Togepi', streng
th: 100}));
  });
```

```
  it('should render pokemon name in a h2 tag', () => {
    fixture.detectChanges();

    expect(fixture.nativeElement.querySelector('h2').textContent).toContain('TOGEPI
');
  })
})
```

Slika 15

Testovi se pokreću analogno unit testovima.

Rezultati se takođe čitaju analogno unit testovima.

Testovi su prošli uspešno. 😊

Literatura

[1] Hutcheson, M. L. (2003). Software Testing Fundamentals: Methods and Metrics (1st ed.). Wiley. https://download.e-bookshelf.de/download/0000/5840/13/L-G-0000584013-0002360984.pdf

[2] T. (2021, March 5). Angular Testing Tutorial: What You Need and How to Start. AI-Driven E2E Automation with Code-like Flexibility for Your Most Resilient Tests. https://www.testim.io/blog/angular-testing-tutorial/

[3] Bachina, B. (2020, May 30). Angular — A Comprehensive guide to unit-testing with Angular and Best Practices. Medium. https://medium.com/bb-tutorials-and-thoughts/angular-a-comprehensive-guide-to-unit-testing-with-angular-and-best-practices-e1f9ef752e4e

[4] Palmer, J., Cohn, C., Giambalvo, M., & Nishina, C. (2018). Testing Angular Applications (1st ed.). Manning Publications.

[5] RayRay, D. B. (2020, December 10). Introduction to Angular Testing - Level Up Coding. Medium. https://levelup.gitconnected.com/introduction-to-angular-testing-c596aff78a3a