Angular unit testing

Jasmin, Karma

Šta je Angular testing?

- Osnovna funkcionalnost koja je dostuna u svakom projektu koji je kreiran preko **Angular CLI**.
- Postoje dve vrste Angular tetsiranja:
  - **Unit testiranje** – je proces testiranja malih, izolovanih delova koda. Poznato kao izolovano testiranje, unit testovi ne koriste externe zavisnosti, kao što su internet ili baza podataka.
  - Funkcionalno testiranje – se odnosi na testiranje funkcionalnosti Angular aplikacije iz perspektive krajnjeg korisnika – npr. interakcija sa aplikacijom u browseru kao što bi to radio krajnji korisnik.

Šta je Angular **unit** testing?

- Unit testing u Angularu se odnosi na proces testiranja pojedinačnih jedinica koda.
- Angular jedinični testovi imaju za cilj da izolovanjem delova koda otkriju probleme kao što su netačna logika, funkcije koje se ponašaju nepredviđeno….
- Ovo je ponekad teže nego što zvuči, posebno za složene projekte gde postoji kršenje principa jedinstvene odgovornosti (ne poštuje se princip razdvajanja odgovornosti – jedna komponenta – jedna odgovornost)
- Angular je dizajniran da vam pomogne da napišete kod na način da možete da pojedinačno testirate funkcionalnosti vaše aplikacije u izolaciji.

Zašto bi trebalo da Angular aplikaciju pokrijete unit testovima?

- Angular unit testiranje vam omogućava da testirate svoju aplikaciju na osnovu ponašanja korisnika. Testiranje svakog mogućeg ponašanja korisnika bilo bi zamorno i neefikasno, dok pisanje testova za svaku komponentu u vašoj aplikaciji može pomoći da se pokaže kako se ove komponente ponašaju.
- Ne morate nužno da čekate dok se vaši korisnici ne požale na to kako se polje za unos ponaša kada se klikne na dugme. Pisanjem jediničnog testa za svoje komponente ili servise, možete lako otkriti kada je došlo do greške.
- Naš primer Angular aplikacije ima servise, komponente i asinhroni task koji simulira dobavljanje podataka sa servera.

Jasmine I Karma?

- Uvek se preporučuje da se koristi **Angular CLI** za kreiranje Angular projekata. Time se konfiguracija **Jasmine** i **Karme** automatski postavlja.
- Kada kreirate projekat preko **Angular CLI** sve zavisnosti koje su potrebne za kreiranje testova se instaliraju što se može videti u **package-lock.json** falu kreiranog projekta .

```
"jasmine-core": "~4.3.0",
"karma": "~6.4.0",
"karma-chrome-launcher": "~3.1.0",
"karma-coverage": "~2.2.0",
"karma-jasmine": "~5.1.0",
"karma-jasmine-html-reporter": "~2.0.0",
```

- **jasmine-core**. – **Jasmine** je framework koji se koristi za pisanje testova. Ima gomilu funkcionalnosti koje nam omogućavaju da pišemo različite vrste testova.
- **karma**. - **Karama** je JavaScript test runner koji pokreće testove u Angularu. Koristi konfiguracioni fajl da bi izmedju ostalog podesio stratup fajlove, reportere, framework i browser.
- Možemo im pokazati i **karma.conf.js** fajl.

Kako se piše Angular test?

- Kada kreirate novi projekat preko **Angular CLI** (**ng new appName**), po defaultu se dodaju glavna komponenta i njen test fajl. Takođe, test skripta se kreira zajedno sa komponentom ili servisom koji kreirate korišćenjem **Angular CLI**.
- Kreiranje servisa: ng generate **component** [*name*]
- Kreiranje komponente: ng generate **service** [*name*]
- Naziv test skripte se završava sa **.spec.ts**

Primer

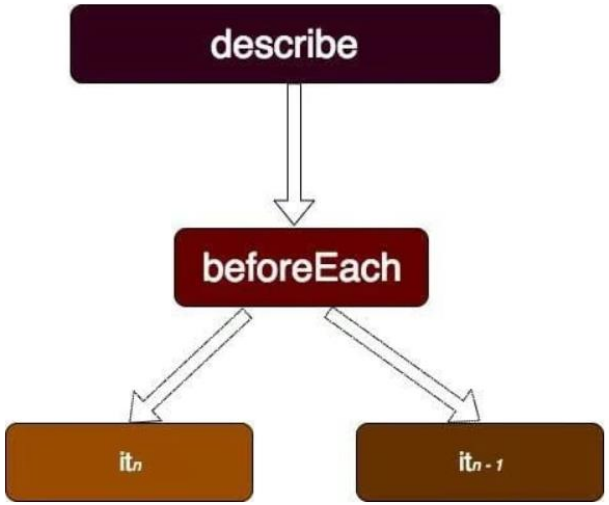
- Kada se tek kreira projekat inicijalna test skripta (**app.component.spec.ts**) izgleda ovako:

```
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'angular-app'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('angular-app');
  });

  it('should render title', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    expect(compiled.querySelector('.content span')?.textContent).toContain('angular-app app is running!');
  });
});
```

- Pokreće se sa komandom **ng test**

Kako se piše test u Angularu?

- Angular paket za testiranje uključuje dva pomoćna modula pod nazivom TestBed i async.



- **describe** kontejner se sastoji od različitih blokova (**beforeEach**, **it**)
- **beforeEach** se pokreće pre bilo kog bloka. Pokretanje ostalih blokova ne zavisi od drugih blokova.

Kako se piše test u Angularu?

- Prvi blok je **beforeEach** u **app.component.spec.ts** unutar **describe** kontejnera je jedini blok koji se pokreće pre bilo kog drugog bloka (it).
- Komponenta (AppComponent) koja je deklarisana u **beforeEach** bloku je komponenta koju želimo da testiramo.

Kako se piše test u Angularu?

- **compileComponents** je pozvana da kompajlira resurse komponente kao što su template, styles itd.

```
describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });
```

Kako se piše test u Angularu?

- Sada kada je komponenta deklarisana u bloku **beforeEach**, proverimo da li je komponenta kreirana unutar **it** bloka.

```
it('should create the app', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(app).toBeTruthy();
});
```

- **fixture.componentInstance** kreira instancu komponente (AppComponent). Testira se da li je instanca komponente zaista kreirana ili ne koristeći **toBeTruthy** metodu.

Kako se piše test u Angularu?

- Sledeći **it** blok (test) pokazuje kako možete da imate pristup svojstvima kreirane komponente (AppComponent). Jedino svojstvo koje je podrazumevano dato je naslov. Možete lako da proverite da li je naslov koji ste postavili promenjen ili ne iz kreirane instance komponente (AppComponent) :

```
it(`should have as title 'angular-app'`, () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(app.title).toEqual('angular-app');
});
```

Kako se piše test u Angularu?

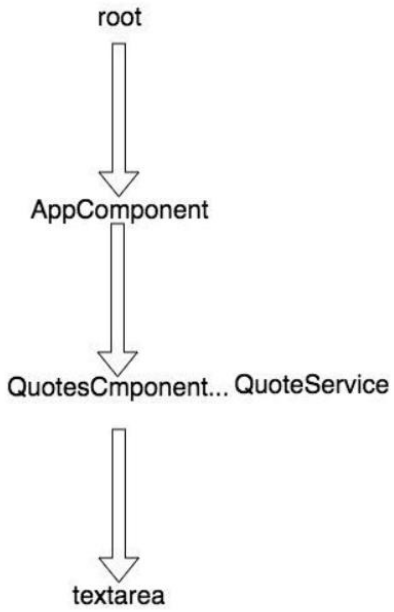
- Poslednji blok pokazuje kako se test ponaša u okruženju browsera. Nakon kreiranja komponente, poziva se **detectChanges** metoda kreirane komponente za simuliranje pokretanja na browseru.
- Sada kada je komponenta renderovana, možete imati pristup njenom **child** elementu tako što ćete pristupiti objektu **nativeElelment** komponente (**fixture.debugElement.nativeElement**):

```
it('should render title', () => {
  const fixture = TestBed.createComponent(AppComponent);
  fixture.detectChanges();
  const compiled = fixture.nativeElement as HTMLElement;
  expect(compiled.querySelector('.content span')?.textContent).toContain('angular-app app is running!');
});
```

- **fixture.detectChanges()** – simulacija renderovanja komponente u browseru.

Primer

- Sada kada smo se upoznati sa osnovama testiranja Angular komponente, hajde da testiramo našu Angular aplikaciju:



- Pokazati aplikaciju

Kako testirati Angular servise?

- Servisi često zavise od drugih servisa koje Angular ubrizgava (injectuje) u konstruktor. U mnogim slučajevima, lako je kreirati i ubrizgati ove zavisnosti dodavanjem **providedIn: root**, što ga čini dostupnim bilo kojoj komponenti ili servisu:

```
@Injectable({
  providedIn: "root"
})
export class QuoteService {
```

Testiranje QuoteService

- Imamo nekoliko načina da testiramo **QuoteService**
- U **beforeEach** bloku kreira se instanca **QuoteService**
- Ovo se radi u beforeEach metodi da bi se osiguralo da bi se izbeglo ponavljanje u drugim blokovima:

```
describe("QuoteService", () => {
  let service: QuoteService;

  beforeEach(() => {
    service = new QuoteService();
  });
```

Testiranje QuoteService

- Prvi blok testira da li je citat **QuoteModel** kreiran, tako što proverava dužinu liste. Očekuje se da će dužina liste biti 1:

```
it("should create a post in an array", () => {
  const qouteText = "This is my first post";
  service.addNewQuote(qouteText);
  expect(service.quoteList.length).toBeGreaterThanOrEqual(1);
});
```

- Drugi blok kreira citat u nizu i odmah ga uklanja pozivom funkcije **RemoveQuote** nad objektom servisa. Očekuje se da će dužina liste citata biti 0.

```
it("should remove a created post from the array of posts", () => {
  service.addNewQuote("This is my first post");
  service.removeQuote(0);
  expect(service.quoteList.length).toBeLessThan(1);
});
```

Kako testirati Angular komponentu?

- U našoj aplikaciji, **QuoteService** se ubrizgava (injectuje) u **QuoteComponent** :

```
export class QuoteComponent implements OnInit {
  public quoteList: QuoteModel[] = [];
  public fetchedList: QuoteModel[] = [];
  public quoteText: String = "";

  constructor(private service: QuoteService) {}
```

Testiranje QuoteComponent

- Prva dva **beforeEach** bloka u **describe** kontejneru se pokreću uzastopno.
- U prvom bloku, **FormsModule** se importuje u **configureTestingModule**. Na taj način se osigurava korišćenje srodnih direktiva, kao što je **ngModel**. Takođe, **QuotesComponent** se deklariše u **configureTestingModule**.
- Drugi blok kreira **QuoteComponent** i njegovu instancu koju će koristiti drugi blokovi:

```
describe("QuotesComponent", () => {
  let component: QuoteComponent;
  let fixture: ComponentFixture<QuoteComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [FormsModule],
      declarations: [QuoteComponent]
    });
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(QuoteComponent);
    component = fixture.debugElement.componentInstance;
  });
```

Testiranje QuoteComponent

- Ovaj **it** blok (test) testira da li je definisana instanca komponente koja je kreirana:

```
it("should create Quote component", () => {
  expect(component).toBeTruthy();
});
```

- **QuoteService** manipuliše citatima (dodavanje, uklanjanje, dobavljanje) – promenljiva **quoteService**.
- Komponenta će biti renderovana kada se pozove metoda **detectChanges**:

```
it("should use the quoteList from the service", () => {
  const quoteService = fixture.debugElement.injector.get(QuoteService);
  fixture.detectChanges();
  expect(quoteService.getQuote()).toEqual(component.quoteList);
});
```

- U metodi **ngOnInit QuoteComponent-e** se poziva **quoteService.getQuote()** i njen rezultat semešta u **quoteList** – polje komponente – zato čekamo da se komponenta renderuje, da bi se pozvala **ngOnInit** metoda.

Testiranje QuoteComponent

- Hajde da testiramo da li možemo uspešno da napravimo citat.
- Svojstvima komponente može se pristupiti nakon instanciranja, kada se citat prosledi u **quoteText** svojstvo potrebno je pozvati **detectChanges** da bi komponenta detektovala nove promene.
- Objekat **nativeElement** daje pristup prikazanom HTML elementu, što olakšava proveru da li je dodat citat deo prikazanih citata:

```
it("should create a new post", () => {
  component.quoteText = "I love this test";
  fixture.detectChanges();
  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.innerHTML).toContain("I love this test");
});
```

Testiranje QuoteComponent

- Osim što imate pristup HTML sadržaju, možete da dobijete i element pomoću njegovog CSS svojstva.
- Testiramo: kada je quoteText prazan (empty) ili bez vrednosti (null), očekuje se da će dugme biti disabled:

```
it("should disable the button when textArea is empty", () => {
  fixture.detectChanges();
  const button = fixture.debugElement.query(By.css("button"));
  expect(button.nativeElement.disabled).toBeTruthy();
});

it("should enable button when textArea is not empty", () => {
  component.quoteText = "I love this test";
  fixture.detectChanges();
  const button = fixture.debugElement.query(By.css("button"));
  expect(button.nativeElement.disabled).toBeFalsy();
});
```

Testiranje QuoteComponent

- Baš kao i način na koji pristupamo elementu sa njegovim CSS svojstvom, možemo pristupiti i elementu preko naziva klase. Višestrukoj klasi možete pristupiti u isto vreme koristeći metode **By** objekta npr. **By.css('.className.className')**.
- Klik na dugme se simulira pozivanjem **triggerEventHandler** metode. Mora biti naveden tip događaja koji je, u ovom slučaju, na klik.
- Očekuje se da će prikazan citat biti izbrisan sa liste citata kada se na njega klikne:

```
it("should remove post upon card click", () => {
  component.quoteText = "This is a fresh post";
  fixture.detectChanges();

  fixture.debugElement
    .query(By.css(".row"))
    .query(By.css(".card"))
    .triggerEventHandler("click", null);
  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.innerHTML).toContain("This is a fresh post");
});
```

Kako testirati async operacije u Angularu?

- Neizbežno je da ćete morati da dobavljate podatke sa servera. Ova operacija se najbolje obavlja kao asinhroni task.
- **fetchQuotesFromServer** metoda koja se nalazi u **QuoteService** predstavlja async task koji vraća niz citata nakon dve sekunde:

```
fetchQuotesFromServer(): Promise<QuoteModel[]> {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve([new QuoteModel("I love unit testing", "Mon 4, 2018")]);
    }, 2000);
  });
}
```

Testiranje asinhrone metode

- **spyOn** simulira rad **fetchQuotesFromServer** metode. Prihvata dva argumenta: **quoteService** i **fetchQuotesFromServer** metod čiji rad treba da simulira.
- Očekuje se da će **fetchQuotesFromServer** vratiti **Promise**. **spyOn** koristi **and** metodu sa lažnim **Promise** pozivom, koji se vraća koristeći **returnValue**.
- Da bi simulirali rad **fetchQuotesFromServer** metode, moramo da prosledimo **Promise** sa listom citata.
- Pozove se **detectChanges** metod da bi dobili ažurirane promene. **whenStable** metoda dozvoljava pristup rezultatima svih asinhronih taskova tek kada se oni završe:

```
it("should fetch data asynchronously", async () => {
  const fakedFetchedList = [
    new QuoteModel("I love unit testing", "Mon 4, 2018")
  ];
  const quoteService = fixture.debugElement.injector.get(QuoteService);
  let spy = spyOn(quoteService, "fetchQuotesFromServer").and.returnValue(
    Promise.resolve(fakedFetchedList)
  );
  fixture.detectChanges();
  fixture.whenStable().then(() => {
    expect(component.fetchedList).toBe(fakedFetchedList);
  });
});
```

Lažne implementacije

- Ne želimo da koristitimo prave servise već njihovu lažnu implementaciju (mock).
- **UserComponent**:

```
@Component({
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.sass']
})
export class UserComponent {
  text = 'user page';
  users;

  constructor(private userService: UserService) {
    this.users = this.userService.getUsers();
  }
}
```

I način – kreiranje mock klase

- Ovo je jednostavna komponenta koja dobija listu korisnika iz servisa.
- Stvarna implementacija servisa nije važna ona bi mogla da bude dobijanje korisnika sa bilo kog mesta.
- Kako napraviti lažnu implementaciju servisa (mock)?
- Možemo napraviti **UserServiceMock** u **user.service.mock.ts**:

```
import { Injectable } from '@angular/core';

@Injectable()
export class UserServiceMock {
  constructor() { }

  getUsers(): Array<{}> {
      return [
          {
              name: 'user1',
              surname: 'usersurname1'
          }
      ];
  }
}
```

I način – kreiranje mock klase

- U deklaraciji provajdera test modula kažemo modulu da kada se servis **UserService** unjektuje treba da koristi **UserServiceMock**.
- **UserServiceMock** je mock servis koji smo kreirali. Ovaj mock servis vraća lažne podatke za potrebe testiranja komponente.
- Ovako treba da mokujete servise prilikom testiranja komponente.

```
beforeEach(() => {


    TestBed.configureTestingModule({
        declarations: [
            UserComponent
        ],
        providers: [
            { provide: UserService, useClass: UserServiceMock }
        ]
    }).compileComponents();
});

beforeEach(() => {
    fixture = TestBed.createComponent(UserComponent);
    comp = fixture.componentInstance; // UserComponent test instance
});
```

II način – direktno stubovanje

- Možemo odmah napravti spyObjekat i stubovati njegovu metodu

```
beforeEach(() => {
    const users : Array<UserModel> = [
        {
            name: 'user1',
            surname: 'usersurname1'
        }
    ];
    const userServiceSpy = jasmine.createSpyObj<UserService>(['getUsers']);
    userServiceSpy.getUsers.and.returnValue(users);

    TestBed.configureTestingModule({
        declarations: [
            UserComponent
        ],
        providers: [
            { provide: UserService, useValue: userServiceSpy }
        ]
    }).compileComponents();
});

beforeEach(() => {
    fixture = TestBed.createComponent(UserComponent);
    comp = fixture.componentInstance; // UserComponent test instance
});
```

Testiranje reaktivnih formi (Contact komponenta)

- contact.component HTML

```
<div>
    {{ text }}
</div>

<form id="contact-form" [formGroup]="contactForm" (ngSubmit)="onSubmit()" novalidate>
  <div class="form-group">
    <label class="center-block">Name:
      <input class="form-control" formControlName="name">
    </label>
    <label class="center-block">Email:
      <input class="form-control" formControlName="email">
    </label>
    <label class="center-block">Text:
      <input class="form-control" formControlName="text">
    </label>
  </div>
    <button type="submit"
            [disabled]="!contactForm.valid" class="btn btn-success">Save</button>
</form>
```

Testiranje reaktivnih formi (Contact komponenta)

- **contact.component.ts**
- Testiramo: Dugme za prosleđivanje je onemogućeno ako forma nije validna.

```
@Component({
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.sass']
})
export class ContactComponent {
    text = 'contact page';
    contactForm: FormGroup;
    contact = {
        name: '',
        email: '',
        text: ''
    };
    submitted = false;

    constructor() {
        this.contactForm = new FormGroup({
            'name': new FormControl(this.contact.name, [
                Validators.required,
                Validators.minLength(4)
            ]),
            'email': new FormControl(this.contact.email, [
                Validators.required,
                Validators.email
            ]),
            'text': new FormControl(this.contact.text, Validators.required)
        });
    }

    onSubmit(): void {
        this.submitted = true;
    }
}
```

Testiranje reaktivnih formi (Contact komponenta)

```
describe('ContactComponent', () => {
  let comp: ContactComponent;
  let fixture: ComponentFixture<ContactComponent>;
  let de: DebugElement;
  let el: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        ContactComponent
      ],
      imports: [
        BrowserModule,
        FormsModule,
        ReactiveFormsModule
      ]
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(ContactComponent);

    comp = fixture.componentInstance; // ContactComponent test instance
      // query for the title <h1> by CSS element selector
    de = fixture.debugElement.query(By.css('form'));
    el = de.nativeElement;
  });



  it(`should have as text 'contact page'`, () => {
    expect(comp.text).toEqual('contact page');
  });

  it(`should set submitted to true`, () => {
    comp.onSubmit();
    expect(comp.submitted).toBeTruthy();
  });
```

Testiranje reaktivnih formi (Contact komponenta)

```
it(`should call the onSubmit method`,() => {
  spyOn(comp, 'onSubmit');
  el = fixture.debugElement.query(By.css('button')).nativeElement;
  el.click();
  expect(comp.onSubmit).toHaveBeenCalledTimes(0);
});

it(`form should be invalid`, () => {
  comp.contactForm.controls['email'].setValue('');
  comp.contactForm.controls['name'].setValue('');
  comp.contactForm.controls['text'].setValue('');
  expect(comp.contactForm.valid).toBeFalsy();
});

it(`form should be valid`, () => {
  comp.contactForm.controls['email'].setValue('asd@asd.com');
  comp.contactForm.controls['name'].setValue('aada');
  comp.contactForm.controls['text'].setValue('text');
  expect(comp.contactForm.valid).toBeTruthy();
});
```