UNIT TEST

OPDRACHT 1.1 – BESTAANDE TESTS MET JASMINE EN KARMA

Repareer de automatisch gegenereerde spec van app.component

a. Voeg in de TestBed configuratie een schema toe (geïmporteerd uit @angular/core):

```
declarations: [AppComponent],
schemas: [NO_ERROR_SCHEMA]
```

b. Vervang de querySelector met een element dat in de DOM staat, zoals een div element met de class 'container':

```
expect(compiled.querySelector('div.container')).toBeTruthy();
```

c. Run de tests met het commando dat in package.json staat:

```
npm run  test
```

De test resultaten worden weergegeven in Karma.

OPDRACHT 1.2 – JEST CONFIGUREREN

In plaats van Jasmine en Karma willen we nu Jest gebruiken voor de test.

a. Installeer de benodigde dependencies:

```
npm install jest jest-preset-angular @types/jest
```

b. Daarnaast installeren we nog twee extra helper libraries:

```
 npm install jest-createspyobj @testing-library/jest-dom
```

c. Om Jest werkend te krijgen is wat configuratie nodig. Maak een jest.config.js file met daarin de volgende configuratie:

```
// jest.config.js
module.exports = {
    preset: 'jest-preset-angular',
    setupFilesAfterEnv: ['<rootDir>/setup-jest.ts'],
    globalSetup: 'jest-preset-angular/global-setup',
};
```

d. Maak een setup-jest.ts file met daarin de volgende imports:

```
import '@testing-library/jest-dom';
import 'jest-preset-angular/setup-jest';
```

e. Pas ook de compilerOptions aan in tsconfig.spec.json:

```
"compilerOptions": {
  "outDir": "./out-tsc/spec",
  "module": "CommonJS",
  "types": [
    "jest",
    "@testing-library/jest-dom"
  ]
}
```

f. Tenslotte kan het test script in package.json worden aangepast:

```
"test": "jest --coverage"
```

De test van app.component werkt nog altijd, maar wordt nu uitgevoerd met Jest.

OPDRACHT 1.3 – UNIT TEST VOOR TRIPS COMPONENT

Maak een eerste simpele unit test voor het Trips component

a. Maak een nieuw bestand trips.component.spec.ts. Maak hierin een describe() blok aan, en definieer de variabelen die we nodig hebben voor de test:

```
describe('TripsComponent', () => {
    let component: TripsComponent;
    let citiesServiceSpy: jest.Mocked<CitiesService>;
    let routerSpy: jest.Mocked<Router>;
});
```

b. In een beforeEach() blok wordt het component geïnitialiseerd:

```
beforeEach(() => {
    citiesServiceSpy = createSpyObj(CitiesService);
    routerSpy = createSpyObj(Router);
    component = new TripsComponent(citiesServiceSpy, routerSpy);
});
```

c. Een eerste simpele test zou nu moeten slagen:

```
it('should create', () => {
    expect(component).toBeDefined();
});
```

d. Voeg een nieuw describe() blok toe, en definieer hierin een constante cities:

```
describe('addCities()', () => {
    const cities: City[] = [];
});
```

e. In een eerste test voor addCities() kunnen we controleren dat er geen call naar de service wordt gemaakt wanneer een lege cities array aan de methode wordt meegegeven:

```
it('should not call citiesService.addCity() if input has length 0', () => {
    component.addCities(cities);
    expect(citiesServiceSpy.addCity).not.toHaveBeenCalled();
});
```

f. In een tweede test voegen we een City object aan de array toe. Als we nu de methode aanroepen met deze array, wordt er wel een call naar de service gemaakt:

```
it('should call citiesService.addCity() once if input has length 1', () => {
    cities.push(new City('test-name', 'test-country', 'test-code'));
    component.addCities(cities);
    expect(citiesServiceSpy.addCity).toHaveBeenCalledWith(cities[0]);
});
```

g. In een derde test voegen we nogmaals een City object aan de array toe. Deze keer moet in de methode meerdere calls naar de service gemaakt worden:

```
it('should call citiesService.addCity() 2x if input has length 2', () => {
    cities.push(new City('test-name', 'test-country', 'test-code'));
    component.addCities(cities);
    expect(citiesServiceSpy.addCity).toHaveBeenCalledTimes(2);
});
```

In de coverage file (coverage/lcov-report/index.html) is te zien dat alle regels van het Trip component nu geraakt worden met de test.

INTEGRATION TEST

OPDRACHT 2.1 – NIEUWE TEST VOOR CITIES-EDIT-FORM

Maak een nieuwe file cities-edit-form.component.int.spec.ts

a. Maak een describe() blok aan, en definieer de variabelen die nodig zijn voor de test:

```
describe('CitiesEditFormComponent', () => {
    let component: CitiesEditFormComponent;
    let fixture: ComponentFixture<CitiesEditFormComponent>;
    let countryCodeValidatorSpy: jest.Mocked<CountryCodeValidator>;
});
```

b. Voeg een beforeEach() blok toe, en initialiseer daar de CountryCodeValidatorSpy:

```
beforeEach(() => {
    countryCodeValidatorSpy = createSpyObj(CountryCodeValidator);
    countryCodeValidatorSpy.validate.mockReturnValue(of(null));
});
```

c. Gebruik vervolgens Angular Testbed om de test module te configureren:

```
TestBed.configureTestingModule({
    declarations: [
        CitiesEditFormComponent
    ],
    imports: [
        ReactiveFormsModule
    ],
    providers: [
        { provide: CountryCodeValidator, useValue: countryCodeValidatorSpy }
    ]
});
```

d. Geef tenslotte waardes aan de variabelen die nodig zijn om de tests uit te voeren:

```
fixture = TestBed.createComponent(CitiesEditFormComponent);
component = fixture.componentInstance;
```

e. Een eerste simpele test zou nu moeten slagen:

```
it('should create', () => {
    expect(component).toBeDefined();
});
```

OPDRACHT 2.2 – ELEMENTEN IN DE DOM

Maak een test om input elmenten in de DOM te vinden

a. Maak een nieuw it() blok:

```
it('should have all input fields present on the page', () => {

});
```

b. Trigger de ngOnChanges() methode om de template voor te bereiden:

```
component.ngOnChanges(change);
fixture.detectChanges();
```

c. Het change object dat hier wordt gebruikt, wordt boven in de test gedefinieerd, en is hierdoor herbruikbaar in andere tests:

```
const change = {
    city: {
        previousValue: null,
        currentValue: new City('name', 'country', 'code', [2001]),
        firstChange: true
    } as SimpleChange
};
```

d. Voeg de expects toe voor de verschillende elementen:

```
expect(fixture.nativeElement.querySelector('#country')).toBeVisible();
expect(fixture.nativeElement.querySelector('#name')).toBeVisible();
expect(fixture.nativeElement.querySelector('#countryCode')).toBeVisible();
expect(fixture.nativeElement.querySelector('#yearsVisited')).toBeVisible();
```

OPDRACHT 2.3 – INTERACTIE MET DE DOM

Door interactie met de DOM elementen kunnen wijzigingen in de UI getest worden

a. Maak een nieuw it() blok:

```
it('should add multiple yearsVisited input fields on the page', () => {

});
```

b. Trigger de ngOnChanges() methode om de template voor te bereiden:

```
component.ngOnChanges(change);
fixture.detectChanges();
```

c. In een eerste expect() kun je testen hoe vaak het element nu in de DOM voor komt:

```
expect(fixture.nativeElement.querySelectorAll('#yearsVisited').length).toBe(
1);
```

d. Voeg een nieuw id toe in de template zodat het button element in de test makkelijk gevonden kan worden. Vervolgens kan een click() event op dit element uitgevoerd worden:

```
const addButton = fixture.nativeElement.querySelector('#addYearsVisited');
addButton.click();
fixture.detectChanges();
```

e. In een nieuwe expect() kan nu getest worden hoe vaak het element in de DOM voor komt na de klik:

```
expect(fixture.nativeElement.querySelectorAll('#yearsVisited').length).toBe(
2);
```