Jasmine

Behavior-Driven JavaScript

Angular Unit Testing

Status	writing
BeginnerTuto	https://www.youtube.com/watch?v=HBaid2cPT98&list=PL1ano0qwNuBxyiYXCmO_OjaPwc-GV-L9O&index=1
Summary	This is a tutorial documentation for angular unit testing
Author	Ⓐ Ahmed Hrabi
Type	Tutorial documentation
gitHub repository	https://github.com/Hrabi80/AngularTutorials/tree/main/Unit%20Testing

Unit testing is a software testing technique that allows individual units of code to be tested in isolation from the rest of the application. It is a code that test a code.

Why unit testing:

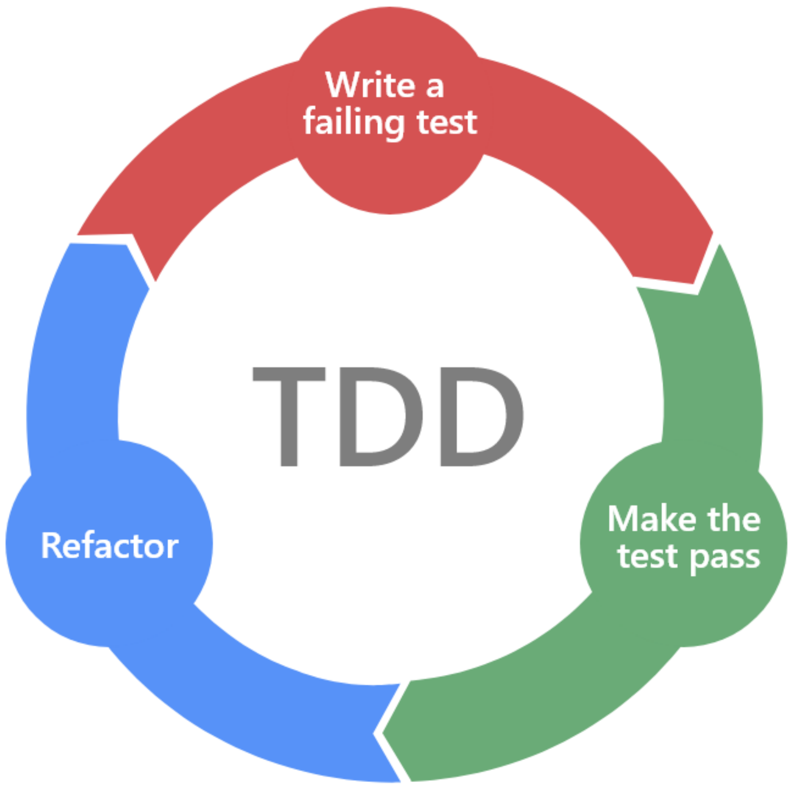
- To deploy the application with confidence, ensure that every part of the application is working as intended.
- Enable developers to confidently apply refactoring while avoiding undesired code behavior.
- Unit tests can detect early flaws in cod

Test Driven Development (TDD):

Is a development process that emphasizes writing automated tests before writing the code. This ensures that the code is testable and that all requirements are met. The process involves three main steps:

1. **Red**: Write a test that fails. This test should capture the intended behavior of the code.
2. **Green**: Write the simplest code that passes the test.
3. **Refactor**: Refactor the code to improve its design while ensuring that all tests pass.

Here is a flowchart that represents the TDD process:



Behavior Driven Development (BDD):

Is a software development methodology that emphasizes collaboration among developers, testers, and business stakeholders to define, implement, and verify system behavior based on real-world scenarios expressed as understandable specifications and tests.

! BDD is a part of TDD.

! The way writing test cases does not require a technical knowledge in order to understand it.

▼ Simple examples:

Hello world example

In AppComponent.ts :

```
hello(){
    return 'hello World!'
  }
```

In AppComponent.spec.ts

```
describe('clicking in hellow funtion',()=>{  // (1) describe of the scenario
  it('must return Hello World!',()=>{    // (2) describe the behavior
    const com = new AppComponent();
    expect(com.hello()) // (3) the methode to be tested
    .toBe('Hello World!');  // (3) expect the return of the method
  })
})
```

Run the command :

```
ng test
// or
ng test --code-coverage
```

Testing a service example:

auth.service:

```
isAuth():boolean {
    return !!localStorage.getItem('token');
  }
```

auth.service.spec.ts:

```
describe('check returning value for isAuth',()=>{
  let auth : AuthService;
  beforeEach(()=>{
    auth = new AuthService();  //a new instance before each test
  });
  afterEach(()=>{
    localStorage.removeItem('token'); // remove token after each test
  })
  it('must return true if there is a token in local storage ',()=>{
    localStorage.setItem('token','mytoken');
    expect(auth.isAuth()).toBeTruthy(); //expect to return true
  });

  it('must return false if there is no token in local storage ',()=>{
    expect(auth.isAuth()).toBeFalsy(); //expect to return false
  })
})
```

Angular Test Bed (ATB):

Angular component is a combination of HTML template and typescript class. An adequate component test should test both template and typescript work together as intended.

The class-only tests can tell you about only : the class behavior.

they cannot

- Tell you If the component is going to render properly
- Respond to user input and gesture
- integrate with parent or child component.

ATB is a higher level *Angular Only* testing framework that allows us to easily test behaviours that depend on the Angular Framework.

When to use ATB :

We use ATB because:

- It allows us to test the interaction of a directive or component with its template.
- It allows us to easily test change detection.
- It allows us to test and use Angular’s DI framework.
- It allows us to test using the `NgModule` configuration we use in our application.
- It allows us to test user interaction via clicks and input fields

▼ Simple example:

`AppComponent.ts:`

```
export class AppComponent {
  title = 'ngUnitTest';
  constructor(private authService:AuthService){
  }
  canLogin(username:string,password:number):boolean{
    return this.authService.isAuthenticated(username,password);
  }
```

`AuthService.service.ts:`

```
isAuthenticated(username:string,password:number):boolean{
    if(username&&password){
      return true;
    }else{
      return false;
    }
  }
```

`App.component.spec.ts:`

```
//import testBed
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';
import { AuthService } from './services/auth.service';

// testBed
describe('App component',()=>{
  let component: AppComponent;
  //a fixute is a wrapper (emballage) for a component and its template
  let fixture:ComponentFixture<AppComponent>;
  let authService: AuthService;
  beforeEach(function(){
    TestBed.configureTestingModule({

    })
   // create component and test fixture
    fixture = TestBed.createComponent(AppComponent);
    // get test component from the fixture
    component = fixture.componentInstance;
    // the service provided to the TestBed
    authService = TestBed.inject(AuthService);
  });
  it('should should create app component',()=>{
    expect(component).toBeTruthy();
  });
  it('should have title ng unit testing',()=>{
    expect(component.title).toEqual('ngUnitTest')
  })
  it('can login',()=>{
    expect(component.canLogin('my uuser',123)).toBeTruthy();
    expect(component.canLogin('',123)).toBeFalsy();
  })
})
```

Testing Http Service example:

`Auth.service.ts`

```
getPost(postId:number):Observable<Post>{
    return this.http.get<Post>(`https://jsonplaceholder.typicode.com/posts/${postId}`);
  }
```

`Auth.service.spec.ts`

```
interface Post {
  userId: number;
  id: number;
  title: string;
  body: string;
}
describe('Auth service post',()=>{
  let service : AuthService;
  beforeEach(()=>{
    TestBed.configureTestingModule({
      imports:[HttpClientModule], // import httpClientModule in test
    });
    service = TestBed.inject(AuthService);
  })
  //DoneFn is methode to be called when the asy methode is done
  it('should get the data succesfully',(done:DoneFn)=>{
    service.getPost(1).subscribe((post:Post)=>{
      expect(post.id).toEqual(1);
      done(); // call the done function here
    })
  })
})
```

Testing Http Service with mock data example:

```
import { HttpClientTestingModule, HttpTestingController} from '@angular/common/http/testing';
// mock data is to make fake data and expect the return data to be like the mock
describe('AuthService with mock data',()=>{
  let service :AuthService;
  let httpMock: HttpTestingController;
  const mockpost = {
    userId: 1,
    id: 2,
    title: "my title",
    body: "my body",
  };
  beforeEach(()=>{
    TestBed.configureTestingModule({
      imports:[HttpClientTestingModule]
    });
    service = TestBed.inject(AuthService);
    httpMock = TestBed.inject(HttpTestingController);
  });

  it('getPost must get data as expected',()=>{
    service.getPost(1).subscribe((data:Post)=>{
      console.log("data is ", data);
      expect(data).toEqual(mockpost);
    })

    // Simulating a request.
    const req = httpMock.expectOne('https://jsonplaceholder.typicode.com/posts/1');
    console.log("req is ====> ",req);
    // Other test example
    expect(req.request.method).toEqual('GET');
    // resolve the request
    req.flush(mockpost);
    // verify that there is no unmatched outstanding requests
    httpMock.verify();
  })
})
```

Testing components:

Testing component example #1:

`login.component.ts:`

```
isLoggedIn:boolean;
  constructor() {
    this.isLoggedIn = false;
  }
  login():void{
    this.isLoggedIn = !this.isLoggedIn;
  }
  get loginState():string{
    return `User is ${this.isLoggedIn ? 'logged in' : 'logged out'}`;
  }
```

`login.component.spec.ts:`

```
describe('Login Component',()=>{
  let com:LoginComponent;
  beforeEach(()=>{
    com = new LoginComponent();
  });
  it('#login() should toggle isLoggedIn',()=>{
    expect(com.isLoggedIn).toBe(false,'false at first');
    com.login();
    expect(com.isLoggedIn).toBe(true,'true after click login');
    com.login();
    expect(com.isLoggedIn).toBe(false,'false after second click');
  });
  it('#login() should toggle loginState() message',()=>{
    expect(com.loginState).toMatch(/out/);
    com.login();
    expect(com.loginState).toMatch(/in/);
  })
})
```