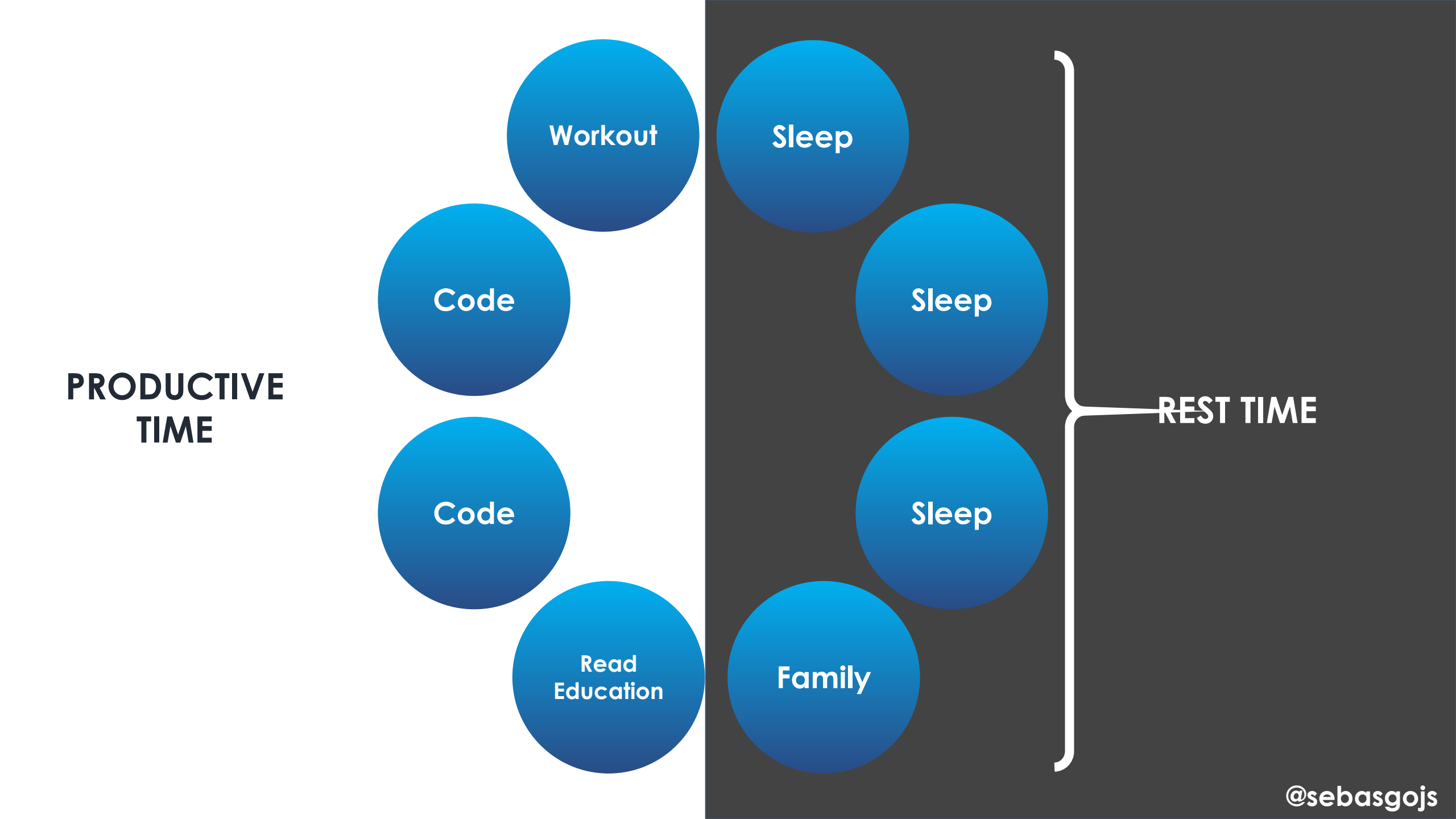
HAVE A GOOD NIGTH

my dear friend

@sebasgojs


PRODUCTIVE TIME
Workout
Code
Code
Read Education
Sleep
Sleep
Sleep
Family
REST TIME
@sebasgojs


Dude, wtf?
...
Wait what?
...
We need to fix it
Move on
Zzz


@sebasgojs

Angular TestBed and Have a good night

@sebasgojs
Sebastian-gomez.com
Front-end Architect at @globant

GDG Medellín

NG COLOMBIA

- Backend problems.
- No Continuous integration.
- No Continuous delivery.
- Quality analysts forgotten to review something.
- No team building activities like “asado”, “breakfast”, “party”...
- **No Unit Tests.**
- **No TDD.**

Why don’t developers write Unit Tests?

The client says ¨no unit tests¨.

Seriously?, how old are you?

My skill is extreme programming

They pay me to write code that works, not to write tests

Ok, I will do the tests later

The budget is too small.

@sebasgojs

I'm not going to waste my life doing tests no ones cares about.

@sebasgojs

Unit Tests are not for:

The project

The client

The company

The money

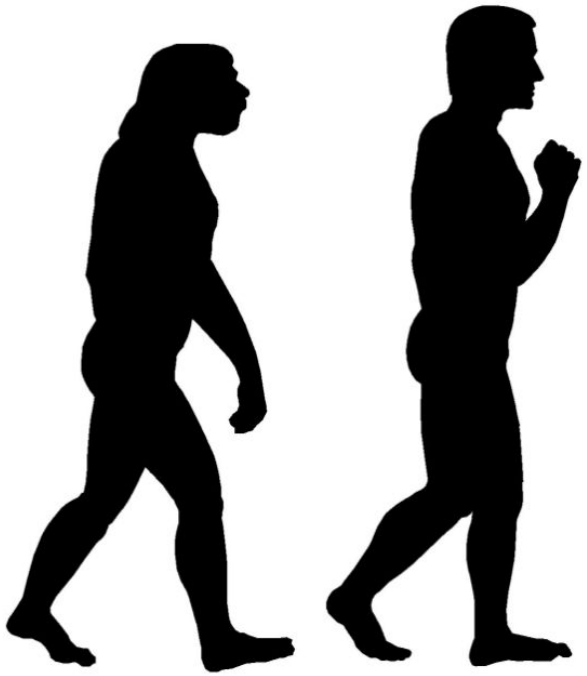
The team

Unit Tests are:

For you

- $scope and $rootScope
- Dependency injection
- Filters
- $compile

- No global scope.
- Better foundation.



Angular TestBed

"The main and most important Angular testing utility"

Configures and initializes environment for unit testing.

Provides methods for creating components and services in unit tests.

Our service

```
@Injectable()
export class MasterService {
    constructor(private valueService: ValueService)
    { }
    getValue() {
        return this.valueService.getValue();
    }
}
```

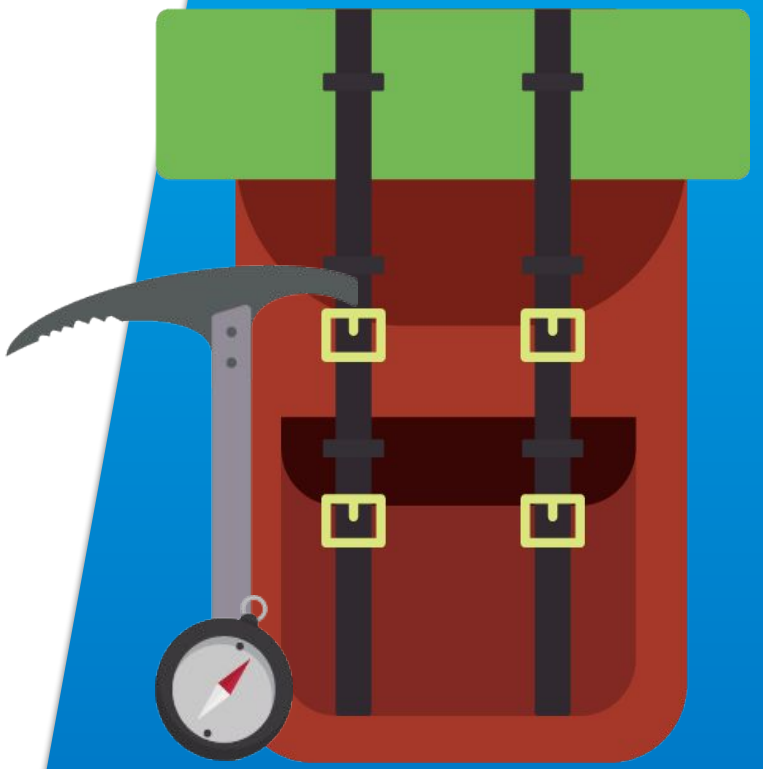
Typical blocks

```
describe( "Test Suite #1" , ()=>{

    beforeEach(()=>{
      // Here we set the preconditions for the tests
    });
    afterEach(()=>{
      // Here we set the postconditions for the tests
    });
     beforeAll, afterAll ....

    it("Should be the first test to implement", ()=>{
    });

});
```



Our ¨always¨ skeleton

```
describe("Test Suite example", () => {
    let masterService: MasterService;
    let valueServiceSpy: jasmine.SpyObj<ValueService>;
      // … as many spies as you need

  beforeEach(() => {
      const spy = jasmine.
                    createSpyObj('ValueService', ['getValue']);

     TestBed.configureTestingModule({

       providers: [

        MasterService,
  …{ provide: ValueService, useValue: spy }
]});
  masterService = TestBed.get(MasterService);
  valueServiceSpy = TestBed.get(ValueService);
```

Our ¨always¨ skeleton

```
describe("Test Suite example", () => {
    let masterService: MasterService;
    let valueServiceSpy: jasmine.SpyObj<ValueService>;
     // … as many spies as you need

  beforeEach(() => {
     const spy = jasmine.
                  createSpyObj('ValueService', ['getValue']);

    TestBed.configureTestingModule({

      providers: [

       MasterService,
  …{ provide: ValueService, useValue: spy }
]});
  masterService = TestBed.get(MasterService);
  valueServiceSpy = TestBed.get(ValueService);
```

Our ¨always¨ skeleton

```
describe("Test Suite example", () => {
    let masterService: MasterService;
    let valueServiceSpy: jasmine.SpyObj<ValueService>;
     // ... as many spies as you need

  beforeEach(() => {
     const spy = jasmine.
                createSpyObj('ValueService', ['getValue']);

    TestBed.configureTestingModule({
      providers: [
       MasterService,
  ...{ provide: ValueService, useValue: spy }
]});
  masterService = TestBed.get(MasterService);
  valueServiceSpy = TestBed.get(ValueService);
```

Our ¨always¨ skeleton

```
describe("Test Suite example", () => {
    let masterService: MasterService;
    let valueServiceSpy: jasmine.SpyObj<ValueService>;
      // … as many spies as you need

  beforeEach(() => {
      const spy = jasmine.
                  createSpyObj('ValueService', ['getValue']);

     TestBed.configureTestingModule({

       providers: [

        MasterService,
   …{ provide: ValueService, useValue: spy }
]});
  masterService = TestBed.get(MasterService);
  valueServiceSpy = TestBed.get(ValueService);
```

Our ¨always¨ skeleton

```
describe("Test Suite example", () => {
    let masterService: MasterService;
    let valueServiceSpy: jasmine.SpyObj<ValueService>;
      // … as many spies as you need
  beforeEach(() => {
     const spy = jasmine.
                  createSpyObj('ValueService', ['getValue']);
     TestBed.configureTestingModule({
       providers: [
        MasterService,
  …{ provide: ValueService, useValue: spy }
]});
  masterService = TestBed.get(MasterService);
  valueServiceSpy = TestBed.get(ValueService);
```

The expects

```
describe("Test Suite example", () => {
...
it('Should return stubbed value from a spy', () => {

  const stubValue = 'stub value';
  valueServiceSpy.getValue.and.returnValue(stubValue);

  expect(masterService.getValue())
    .toBe(stubValue, 'service returned stub value');

  expect(valueServiceSpy.getValue.calls.count())
    .toBe(1, 'spy method was called once');

});
```

1. Imports as usual.

```
import masterService from '...';
import valueService from '...';

describe("Test Suite example", () => {
```

2. Spies on every injected dependency used by service.

```
// masterService.spec.ts
  let valueServiceSpy: jasmine.SpyObj<ValueService>;
  // ...
  let anyServiceSpy: jasmine.SpyObj<AnyService>;
  // ... as many spies as you need

// masterService.ts
@Injectable()
export class MasterService {
   constructor(private valueService: ValueService,
               ..., private anyService: AnyService) { }
...
```

3. Use providers property in TestBed.configureTestingModule. to add the main provider to be tested and useValue to substitute the providers with the spies.

```
TestBed.configureTestingModule({
   providers: [
    MasterService,
    { provide: ValueService, useValue: spy }
```

4. Stub values could be declared in each it statement or in the beforeEach.

```
const stubValue = 'stub value';
valueServiceSpy.getValue.and.returnValue(stubValue);

let stub;

beforeEach(()=>{


});
```

```
it('...', () => {


});
```

5. Add asserts to verify the service only get called the times you consider should be.

```
expect(masterService.getValue())
  .toBe(stubValue, 'service returned stub value');
expect(valueServiceSpy.getValue.calls.count())
  .toBe(1, 'spy method was called once');
```

Never ever tests more than one thing

ValueService could be:

- Async
- Failing
- Third party

You only test code units
so always use spy

Our component

```
@Component({
    selector: 'app-welcome',
    template: '<h3 class="welcome"> <i>{{welcome}}</i></h3>'
})
export class WelcomeComponent  implements OnInit {
   welcome: string;
   constructor(private userService: UserService) { }
   ngOnInit(): void {
      this.welcome =
      this.userService.isLoggedIn ?
        'Welcome, ' + this.userService.user.name :
        'Please log in.';
   }
}
```

Our ¨pre-skeleton¨

```
TestBed.configureTestingModule({
   declarations: [ WelcomeComponent ],
   // providers:    [ UserService ]
   // NO! Don't provide the real service!

   // Provide a test-double instead
   providers:    [
      {provide: UserService, useValue: userServiceStub }
   ]
});
```

Our ¨always¨ skeleton

```
let userServiceStub: Partial<UserService>;
  beforeEach(() => {
    // stub UserService for test purposes
    userServiceStub = {
      isLoggedIn: true,
      user: { name: 'Test User'}
    };
    TestBed.configureTestingModule({...});
    fixture = TestBed.createComponent(WelcomeComponent);
    comp    = fixture.componentInstance;
    userService = TestBed.get(UserService);
    el = fixture.nativeElement.
                  querySelector('.welcome');
});
```

@sebasgojs

Our ¨always¨ skeleton

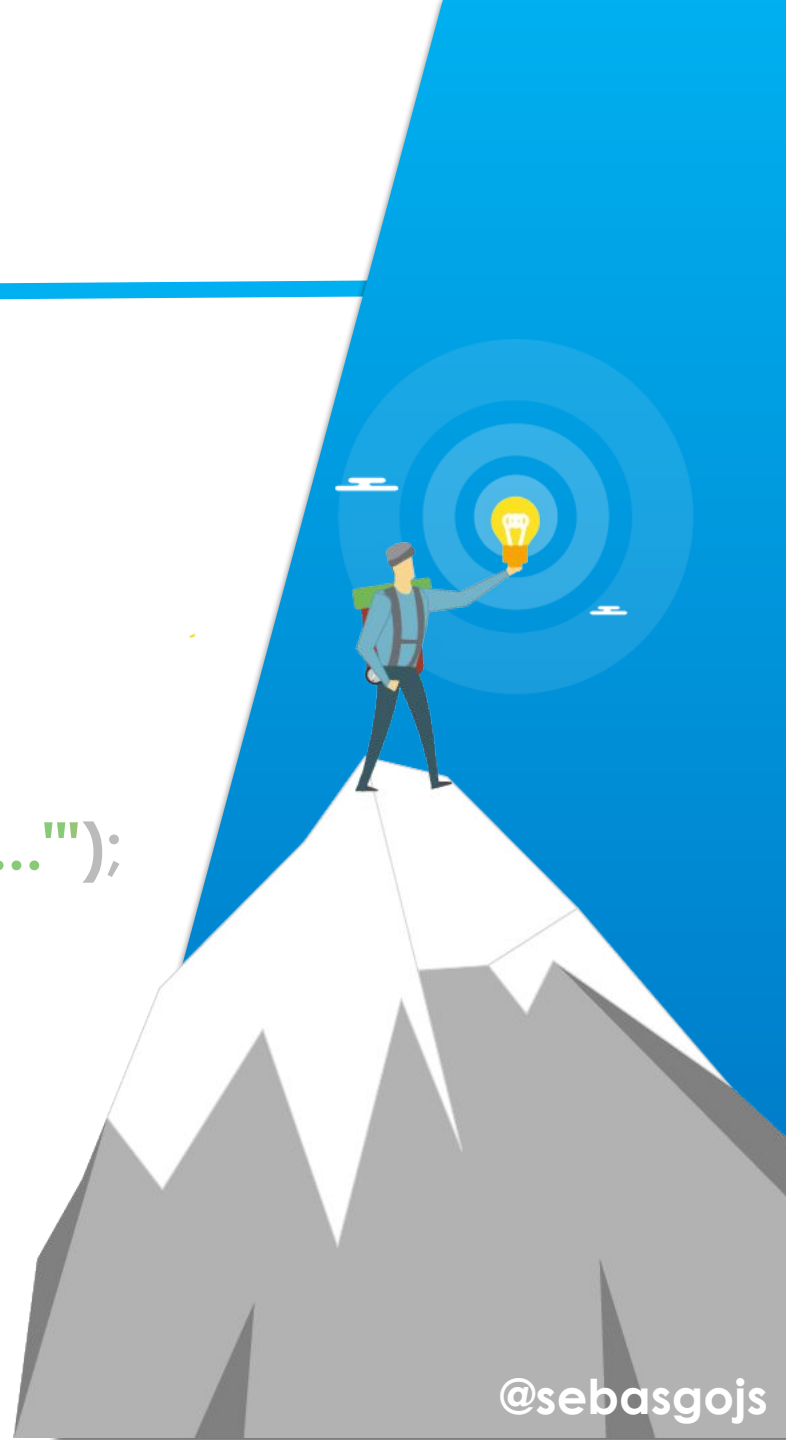
```
let userServiceStub: Partial<UserService>;
  beforeEach(() => {
    // stub UserService for test purposes
    userServiceStub = {
      isLoggedIn: true,
      user: { name: 'Test User'}
    };
    TestBed.configureTestingModule({...});
    fixture = TestBed.createComponent(WelcomeComponent);
    comp    = fixture.componentInstance;
    userService = TestBed.get(UserService);
    el = fixture.nativeElement.
                  querySelector('.welcome');
});
```

Fixture is the key 🔑

```
let userServiceStub: Partial<UserService>;
  beforeEach(() => {
    // stub UserService for test purposes
    userServiceStub = {
      isLoggedIn: true,
      user: { name: 'Test User'}
    };
    TestBed.configureTestingModule({...});
    fixture = TestBed.createComponent(WelcomeComponent);
    comp    = fixture.componentInstance;
    userService = TestBed.get(UserService);
    el = fixture.nativeElement.
                  querySelector('.welcome');
});
```

The expects

```
it('should welcome the user', () => {
    fixture.detectChanges();
    const content = el.textContent;
    expect(content).toContain('Welcome', '"Welcome ..."');
    expect(content).toContain('Test User', 'expected
name');
});
```



The expects

```
it('should welcome the user', () => {
    fixture.detectChanges();
    const content = el.textContent;
    expect(content).toContain('Welcome', ""Welcome ..."");
    expect(content).toContain('Test User', 'expected
name');
});
```

1. Do not use real services

```
let userServiceStub: Partial<UserService>;

....
userServiceStub = {
   isLoggedIn: true,
   user: { name: 'Test User'}
 };

....

 providers:    [ {provide: UserService, useValue:
userServiceStub } ]
```

2. Get the component fixture to access component DOM.

```
fixture = TestBed.
            createComponent(WelcomeComponent);
 comp = fixture.componentInstance;
....
 el = fixture.nativeElement.
                querySelector('.welcome');
```

3. Fixture.detectChanges triggers the data binding process

```
fixture.detectChanges();
```

- Angular TestBed PROVIDES us the way TO initialize anything REQUIRED for our test suite.
- TestBed.get()
- TestBed.ConfigureTestingModule
- TestBed.createComponent is all you need.

- Only focus on the main thing to test and FOR the other ELEMENTS use spies, mocks or fakes.
- Components and providers have a different testing strategy: Fixture vs Direct
- You can assert as much as you want but avoiding no valuable tests.

Angular TestBed and Have a good night

Thank you!

- @sebasgojs
- Sebastian-gomez.com
- Front-end Architect at @globant

GDG Medellín

NG COLOMBIA