TESTING

📖 SERVICE TESTS

```
// Straight Jasmine testing without Angular's testing support

describe('ValueService', () => {

  let service: ValueService;

  beforeEach(() => { service = new ValueService(); });


  it('#getValue should return real value', () => {

    expect(service.getValue()).toBe('real value');

  });
```

TBC

📖 SERVICE TESTS

```
it('#getObservableValue should return value from observable',
  (done: DoneFn) => {
  service.getObservableValue().subscribe(value => {
    expect(value).toBe('observable value');
    done();
  });
});
```

TBC

📖 COMPONENT TESTING

```
@Component({
  selector: 'lightswitch-comp',
  template: `
    <button (click)="clicked()">Click me!</button>
    <span>{{message}}</span>`
})
export class LightswitchComponent {
  isOn = false;
  clicked() { this.isOn = !this.isOn; }
  get message() { return `The light is ${this.isOn ? 'On' : 'Off'}`; }
}
```

TBC

📖 COMPONENT TESTING

```
describe('LightswitchComp', () => {
  it('#clicked() should toggle #isOn', () => {
    const comp = new LightswitchComponent();
    expect(comp.isOn).toBe(false, 'off at first');
    comp.clicked();
    expect(comp.isOn).toBe(true, 'on after click');
    comp.clicked();
    expect(comp.isOn).toBe(false, 'off after second click');
  });
```

TBC

📖 MOCK SERVICE

```
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.';
  }
}
```

TBC

📖 MOCK SERVICE

```
class MockUserService {
  isLoggedIn = true;
  user = { name: 'Test User'};
};
```

TBC

📖 MOCK SERVICE

```
beforeEach(() => {
  TestBed.configureTestingModule({
    // provide the component-under-test and dependent service
    providers: [
      WelcomeComponent,
      { provide: UserService, useClass: MockUserService }
    ]
  });
  // inject both the component and the dependent service.
  comp = TestBed.get(WelcomeComponent);
  userService = TestBed.get(UserService);
});
```

📖 MOCK SERVICE

```
it('should welcome logged in user after Angular calls ngOnInit', () => {
  comp.ngOnInit();
  expect(comp.welcome).toContain(userService.user.name);
});
```

TBC

📖 COMPONENT TEST

```
describe('BannerComponent', () => {
  let component: BannerComponent;
  let fixture: ComponentFixture<BannerComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ BannerComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(BannerComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

TBC

📖 COMPONENT TEST

```
describe('BannerComponent (minimal)', () => {
  it('should create', () => {
    TestBed.configureTestingModule({
      declarations: [ BannerComponent ]
    });
    const fixture = TestBed.createComponent(BannerComponent);
    const component = fixture.componentInstance;
    expect(component).toBeDefined();
  });
});
```

TBC

📖 COMPONENT TEST

```
it('should have <p> with "banner works!"', () => {
  const bannerElement: HTMLElement = fixture.nativeElement;
  const p = bannerElement.querySelector('p');
  expect(p.textContent).toEqual('banner works!');
});
```

TBC

📖 COMPONENT TEST

```
it('should find the <p> with fixture.debugElement.query(By.css)', () => {
  const bannerDe: DebugElement = fixture.debugElement;
  const paragraphDe = bannerDe.query(By.css('p'));
  const p: HTMLElement = paragraphDe.nativeElement;
  expect(p.textContent).toEqual('banner works!');
});
```

TBC

HERE GOES!

TBC