Unit testing in Angular and NgRx

Matěj Chalk
Front-end developer @FlowUp

matejchalk
matejchalk

up

Contents

flow

up

Unit tests in general

flow

What are unit tests?

- ➔ automated tests of a *single unit* of your app *in isolation*
  - ◆ dependencies are mocked
- ➔ *describe* a unit's behaviour and *prove* it works

flow

up

Why unit test?

- ➔ increase code quality
  - ◆ encourage modularization
  - ◆ prevent bugs
  - ◆ document usage

flow

up

What to be aware of

- unit tests should be fast to run and fast to write
  - git hooks, CI
- writing good tests has a learning curve
- team must be committed and disciplined
  - establish clear rules on testing
- extra maintenance

flow

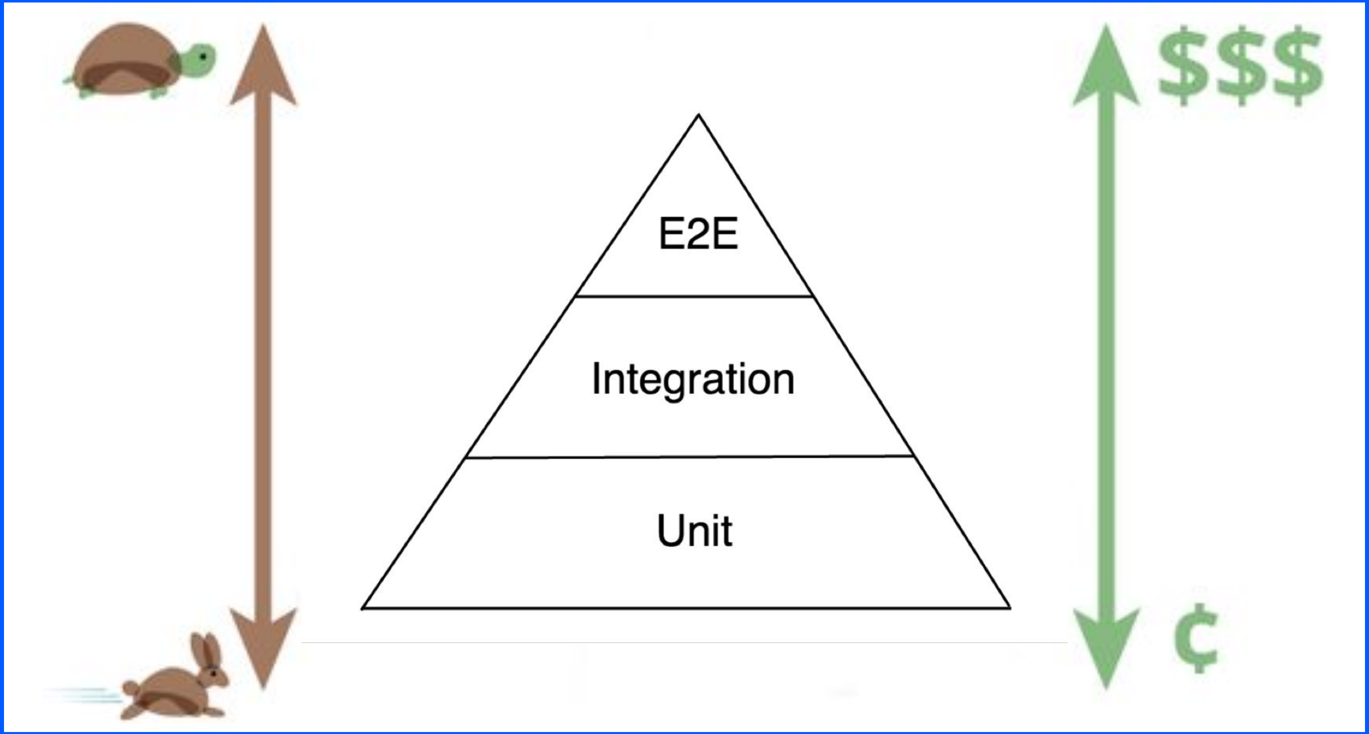
Integration tests

- ➔ test (a part of) your system, composed of multiple units
  - ◆ are they integrated correctly?

up

flow

Testing strategy



up

flow

up

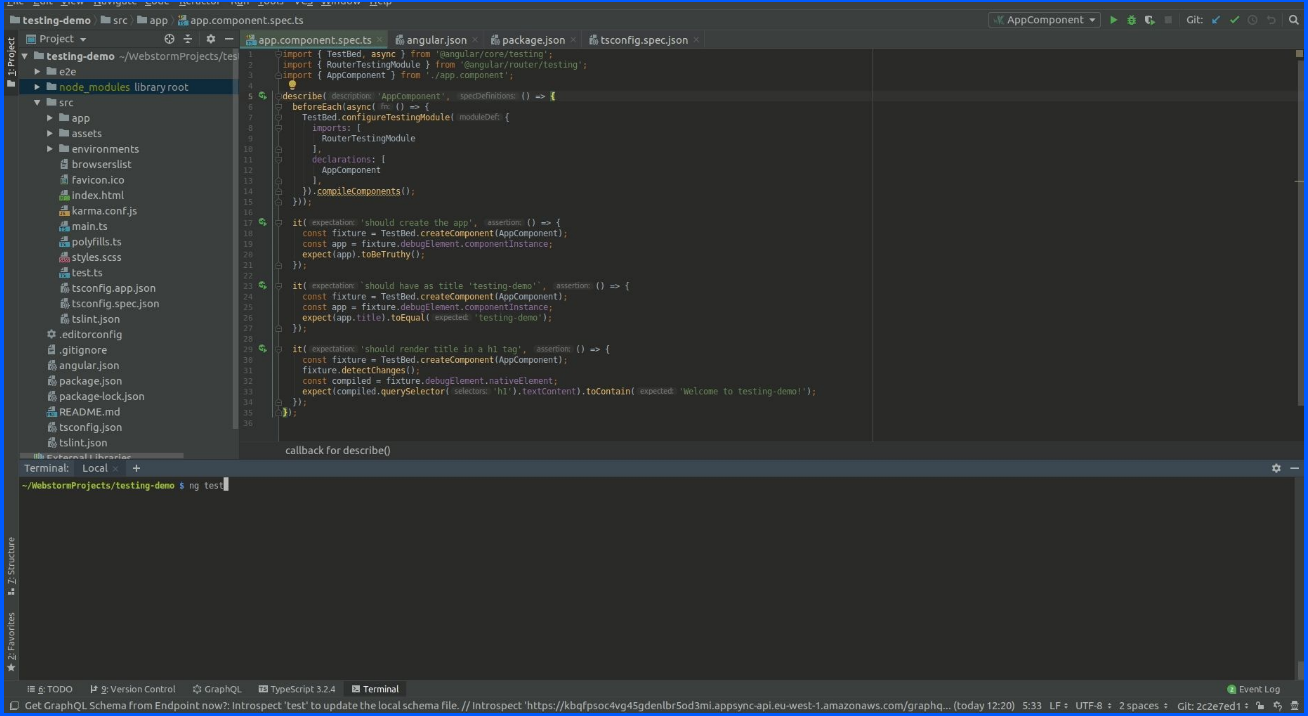
Unit (& integration) testing in Angular

flow

Default tech stack - Karma & Jasmine

➔ *Karma* - test runner, runs in browser on local server

➔ *Jasmine* - testing framework



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

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

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

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

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to testing-demo!');
  });
});
```

callback for describe()

```
~/WebstormProjects/testing-demo $ ng test
```

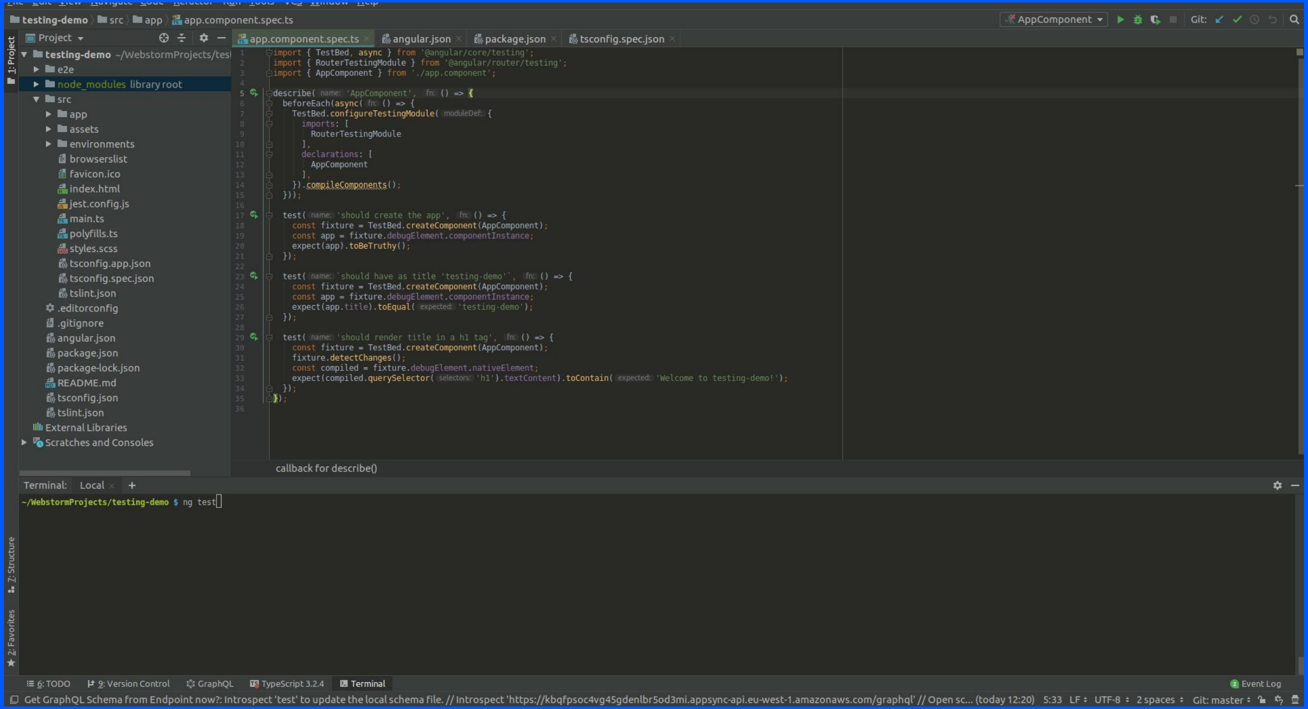
Get GraphQL Schema from Endpoint now?: Introspect 'test' to update the local schema file. // Introspect 'https://kbqfpsoc4vg45gdenlbr5od3mi.appsync-api.eu-west-1.amazonaws.com/graphq... (today 12:20)

flow

Alternative tech stack - Jest

→ testing framework by Facebook

→ runs in Node.js, simulates DOM API via JSDom



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

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

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

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

  test('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to testing-demo!');
  });
});
```

```
~/WebstormProjects/testing-demo $ ng test
```

Tech comparison

Jasmine + KARMA

- 😀 already set up in Angular
- 😀 full browser API

- 😞 slower
  - ◆ starts up local server (unsuitable for CI)
  - ◆ each test requires full app build

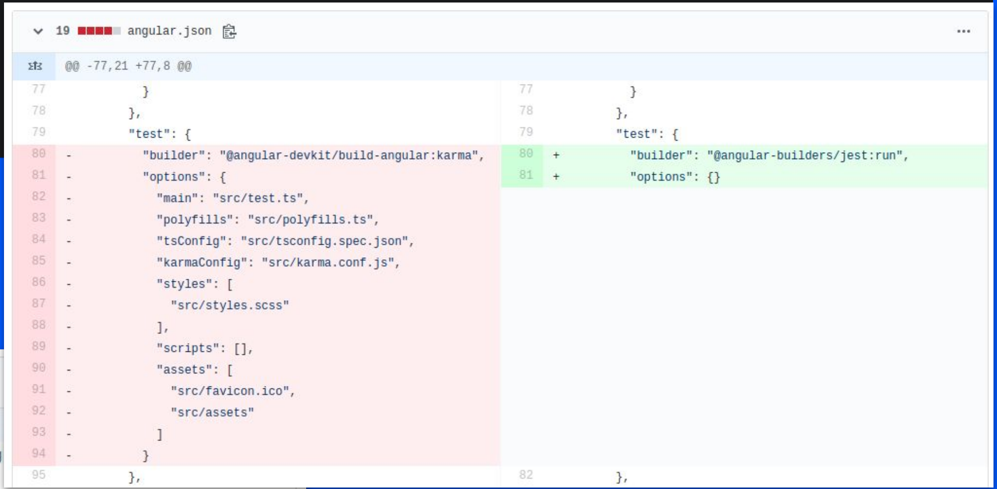
Jest

- 😞 requires setup for Angular
- 😞 JSDom has some limitations

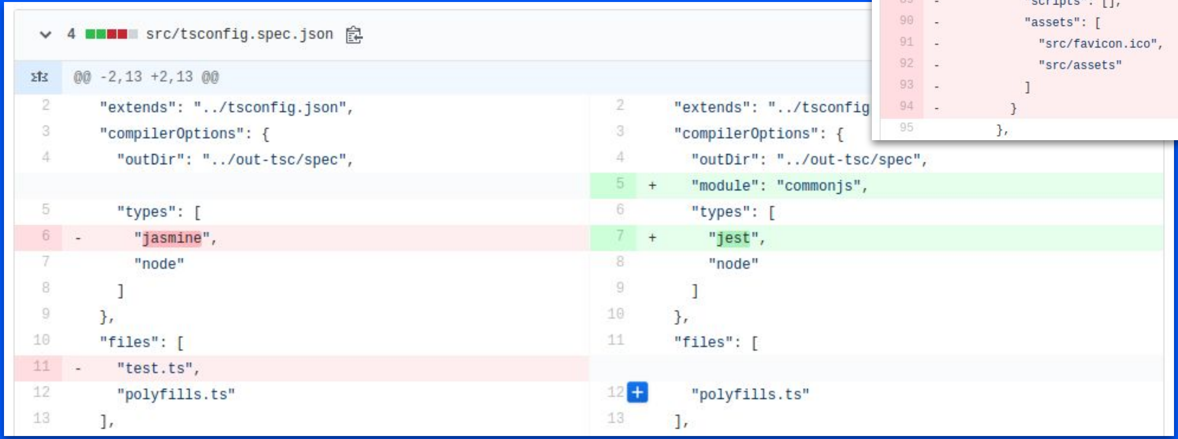
- 😀 faster
  - ◆ Node.js
  - ◆ parallelization
- 😀 code coverage
- 😀 snapshot testing

From Karma/Jasmine to Jest

```
npm remove karma karma-chrome-launcher karma-coverage-istanbul-reporter karma-jasmine karma-jasmine-html-reporter
rm src/karma.conf.js src/test.ts

npm i -D jest @angular-builders/jest @types/jest
echo "module.exports = {};" >> src/jest.config.js
```

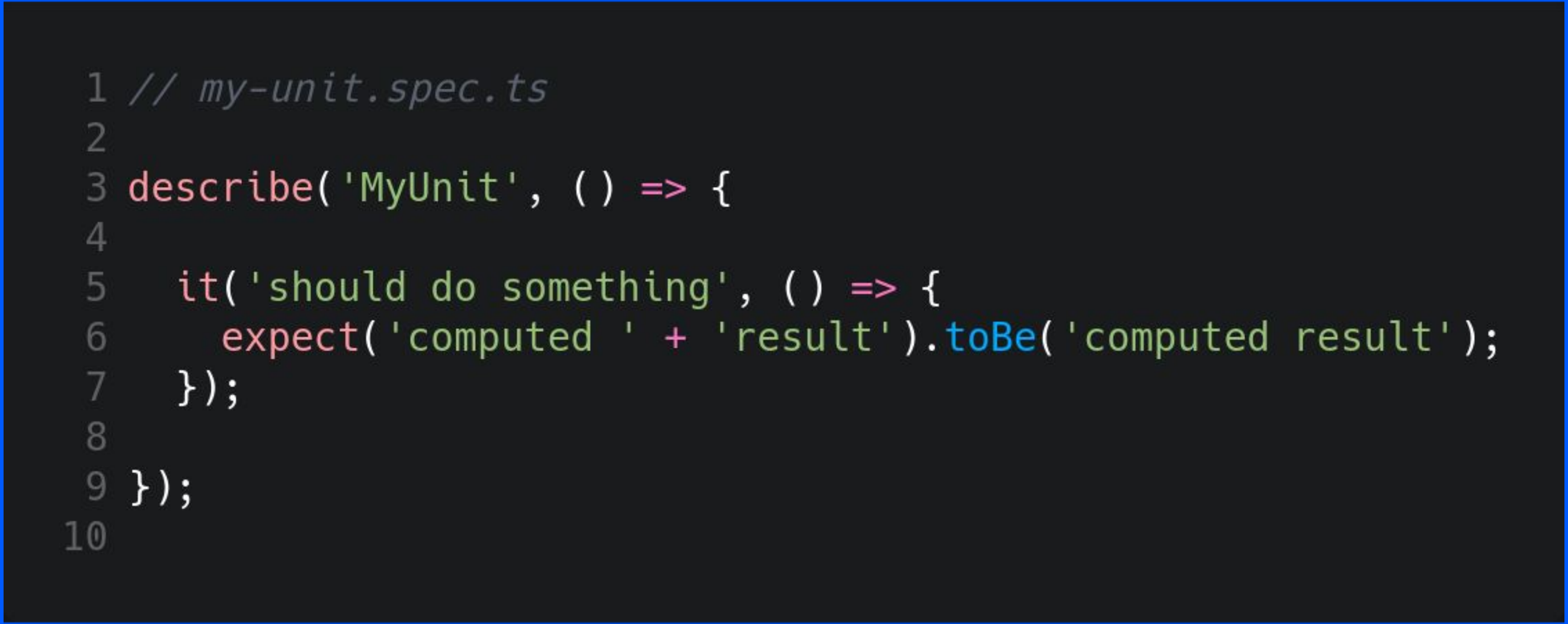


```
19 angular.json
@@ -77,21 +77,8 @@
              }                                                   }
            },                                                  },
            "test": {                                           "test": {
-             "builder": "@angular-devkit/build-angular:karma", +     "builder": "@angular-builders/jest:run",
-             "options": {                                      +     "options": {}
-               "main": "src/test.ts",
-               "polyfills": "src/polyfills.ts",
-               "tsConfig": "src/tsconfig.spec.json",
-               "karmaConfig": "src/karma.conf.js",
-               "styles": [
-                 "src/styles.scss"
-               ],
-               "scripts": [],
-               "assets": [
-                 "src/favicon.ico",
-                 "src/assets"
-               ]
-             }
            },                                                  },
```



```
4 src/tsconfig.spec.json
@@ -2,13 +2,13 @@
    "extends": "../tsconfig.json",          "extends": "../tsconfig
    "compilerOptions": {                    "compilerOptions": {
      "outDir": "../out-tsc/spec",            "outDir": "../out-tsc/spec",
                                        +     "module": "commonjs",
      "types": [                              "types": [
-       "jasmine",                      +       "jest",
        "node"                                  "node"
      ]                                       ]
    },                                      },
    "files": [                              "files": [
-     "test.ts",
      "polyfills.ts"                          "polyfills.ts"
    ],                                      ],
```

Structure of a test



```
// my-unit.spec.ts

describe('MyUnit', () => {

  it('should do something', () => {
    expect('computed ' + 'result').toBe('computed result');
  });

});

```

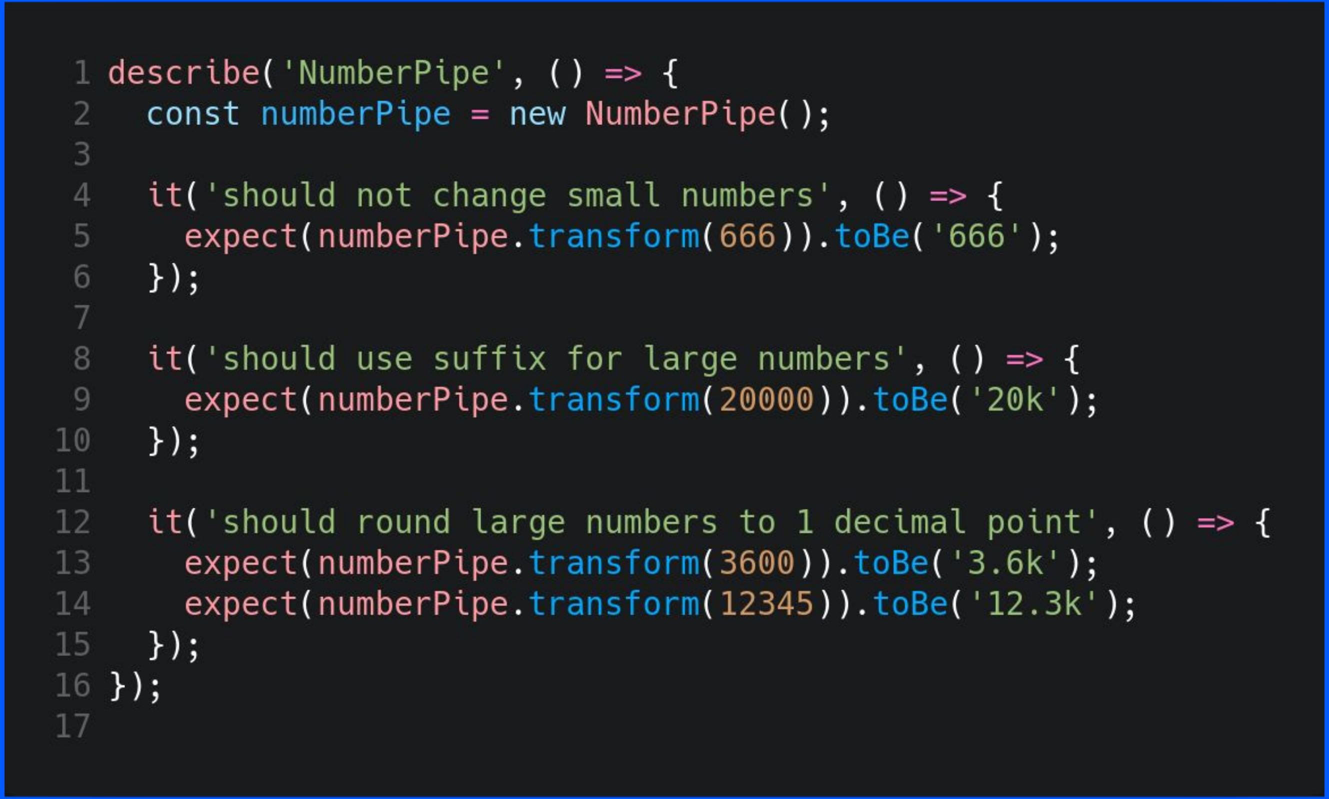
up

What to test in Angular?

- components
- services
- directives
- pipes
- ...

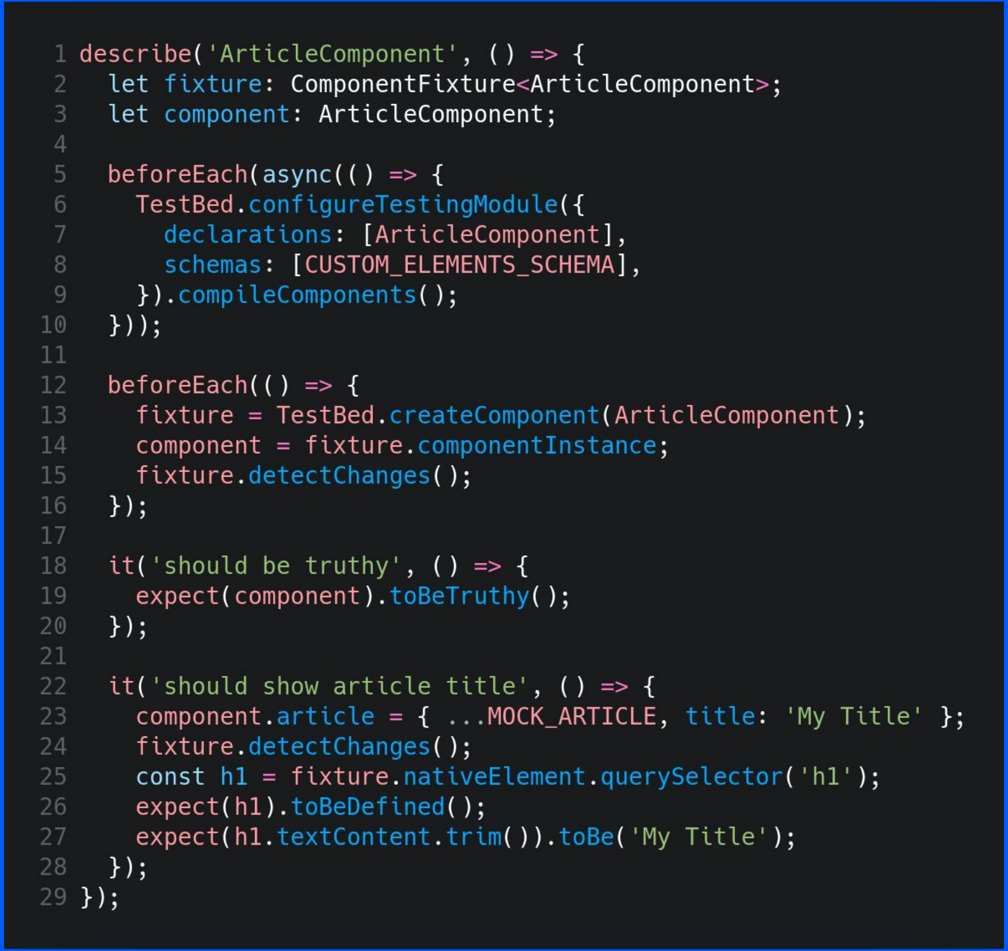
flow

Pipe UT



```
describe('NumberPipe', () => {
  const numberPipe = new NumberPipe();

  it('should not change small numbers', () => {
    expect(numberPipe.transform(666)).toBe('666');
  });

  it('should use suffix for large numbers', () => {
    expect(numberPipe.transform(20000)).toBe('20k');
  });

  it('should round large numbers to 1 decimal point', () => {
    expect(numberPipe.transform(3600)).toBe('3.6k');
    expect(numberPipe.transform(12345)).toBe('12.3k');
  });
});

```

Component UT



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

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ArticleComponent],
      schemas: [CUSTOM_ELEMENTS_SCHEMA],
    }).compileComponents();
  }));

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

  it('should be truthy', () => {
    expect(component).toBeTruthy();
  });

  it('should show article title', () => {
    component.article = { ...MOCK_ARTICLE, title: 'My Title' };
    fixture.detectChanges();
    const h1 = fixture.nativeElement.querySelector('h1');
    expect(h1).toBeDefined();
    expect(h1.textContent.trim()).toBe('My Title');
  });
});
```

up

flow

Mocking providers via dependency injection

```
TestBed.configureTestingModule({
  // ...
  providers: [
    {
      provide: APIClient,
      useValue: {
        getArticles: () => of(MOCK_ARTICLES)
      }
    }
  ]
});
```

```
mockApiClient = TestBed.get(APIClient);
```

Integration testing

```
TestBed.configureTestingModule({
  declarations: [ArticleComponent, AuthorComponent],
  // ...
});
```

```
const authorName = fixture.nativeElement.querySelector(
  'app-author h1'
);
expect(authorName.textContent.trim())
  .toBe(component.article.author.name);
```

up

flow

Unit testing with NgRx

up

flow

Testing with NgRx

- ➔ store can be mocked in components, etc.
- ➔ reducers and selectors are pure functions
  - ◆ perfect for unit testing
- ➔ RxJS marble tests may be used for effects

up

flow

Providing store in NgRx 1-7

```
TestBed.configureTestingModule({
  // ...
  imports: [
    // ...
    StoreModule.forRoot(reducers)
  ]
}).compileComponents();
```

```
store = TestBed.get(Store);
```

```
store.dispatch(new GetArticlesSuccessAction(MOCK_ARTICLES));
// fixture.detectChanges();
```

up

flow

Providing store in NgRx 8

```
TestBed.configureTestingModule({
  // ...
  providers: [
    // ...
    provideMockStore({
      selectors: [
        {
          selector: $articles,
          value: []
        }
      ]
    })
  ]
}).compileComponents();
```

```
$articles.setResult(MOCK_ARTICLES);
// fixture.detectChanges();
```

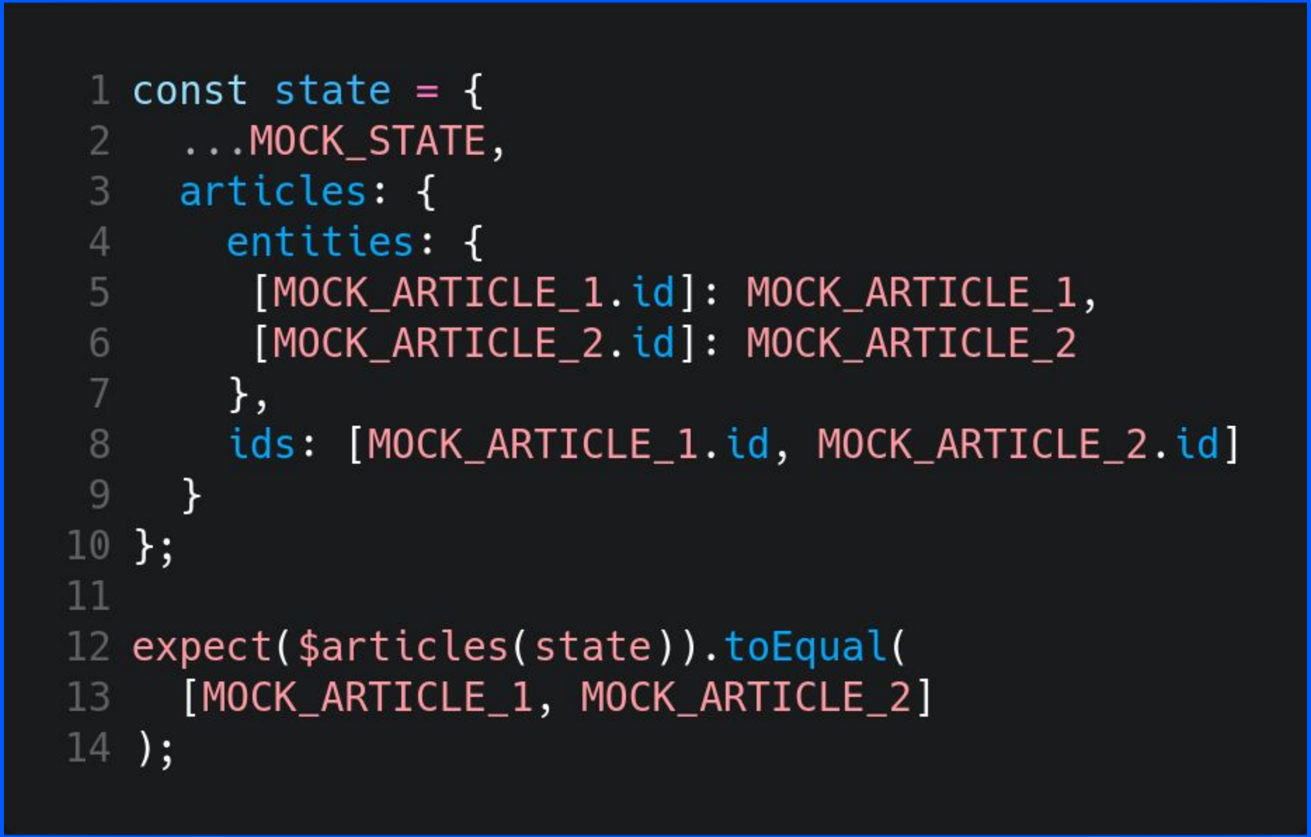
up

flow

Reducer UT

```typescript
const startState: ArticlesState = {
  entities: {},
  ids: [],
  loading: true
};
const action = new GetArticlesSuccessAction(MOCK_ARTICLES);
const endState = articlesReducer(startState, action);

expect(endState.ids).toHaveLength(MOCK_ARTICLES.length);
expect(endState.loading).toBe(false);
expect(endState.entities[MOCK_ARTICLES[0].id])
  .toEqual(MOCK_ARTICLES[0]);
```

Selector UT



```
const state = {
  ...MOCK_STATE,
  articles: {
    entities: {
      [MOCK_ARTICLE_1.id]: MOCK_ARTICLE_1,
      [MOCK_ARTICLE_2.id]: MOCK_ARTICLE_2
    },
    ids: [MOCK_ARTICLE_1.id, MOCK_ARTICLE_2.id]
  }
};

expect($articles(state)).toEqual(
  [MOCK_ARTICLE_1, MOCK_ARTICLE_2]
);
```

Effect UT

```typescript
let effects: ArticlesEffects;
let actions$: Observable<Action>;

beforeEach(() => {
  TestBed.configureTestingModule({
    providers: [
      ArticlesEffects,
      provideMockActions(() => actions$),
      {
        provide: APIClient,
        useValue: { getArticles: () => of(MOCK_ARTICLES) }
      }
    ]
  });
  effects = TestBed.get(ArticlesEffects);
});

test('getArticles$', () => {
  actions$ = hot('--r-', {
    r: new GetArticlesRequestAction()
  });
  const expected = cold('--s', {
    s: new GetArticlesSuccessAction(MOCK_ARTICLES)
  });
  expect(effects.getArticles$).toBeObservable(expected);
});
```

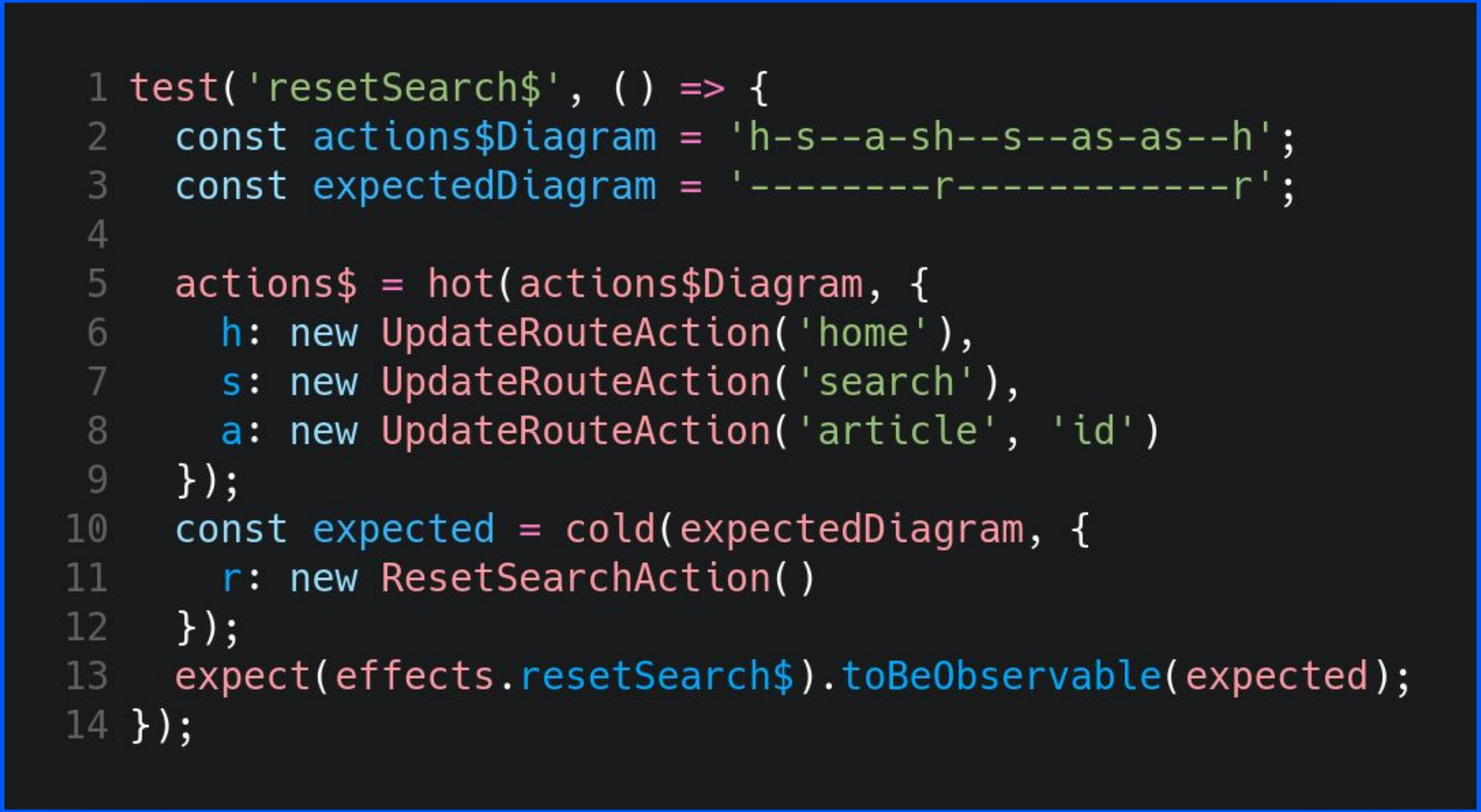
Advanced effect UT 1/2 - the code

```
@Effect() resetSearch$ = this.actions$.pipe(
  ofType(UpdateRouteAction.type),
  map(({ page }) => page),
  filter(page => page !== 'article'),
  distinctUntilChanged(),
  pairwise(),
  filter(([previousPage]) => previousPage === 'search'),
  mapTo(new ResetSearchAction()),
);
```

up

flow

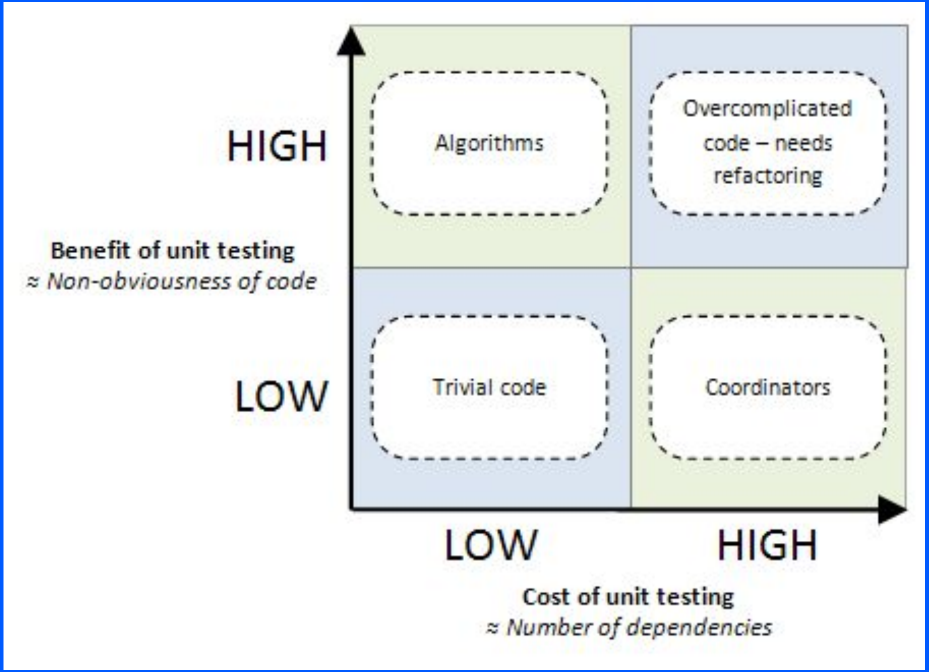
Advanced effect UT 2/2 - the test



```
test('resetSearch$', () => {
  const actions$Diagram = 'h-s--a-sh--s--as-as--h';
  const expectedDiagram = '--------r------------r';

  actions$ = hot(actions$Diagram, {
    h: new UpdateRouteAction('home'),
    s: new UpdateRouteAction('search'),
    a: new UpdateRouteAction('article', 'id')
  });
  const expected = cold(expectedDiagram, {
    r: new ResetSearchAction()
  });
  expect(effects.resetSearch$).toBeObservable(expected);
});
```

Selective unit testing

- → not all code is well-suited for unit testing
- → best to focus on logic heavy code (e.g. NgRx)
- → more dependencies mean more mocking and test maintenance



Q&A

Matěj Chalk

Front-end developer @FlowUp

matejchalk

matejchalk