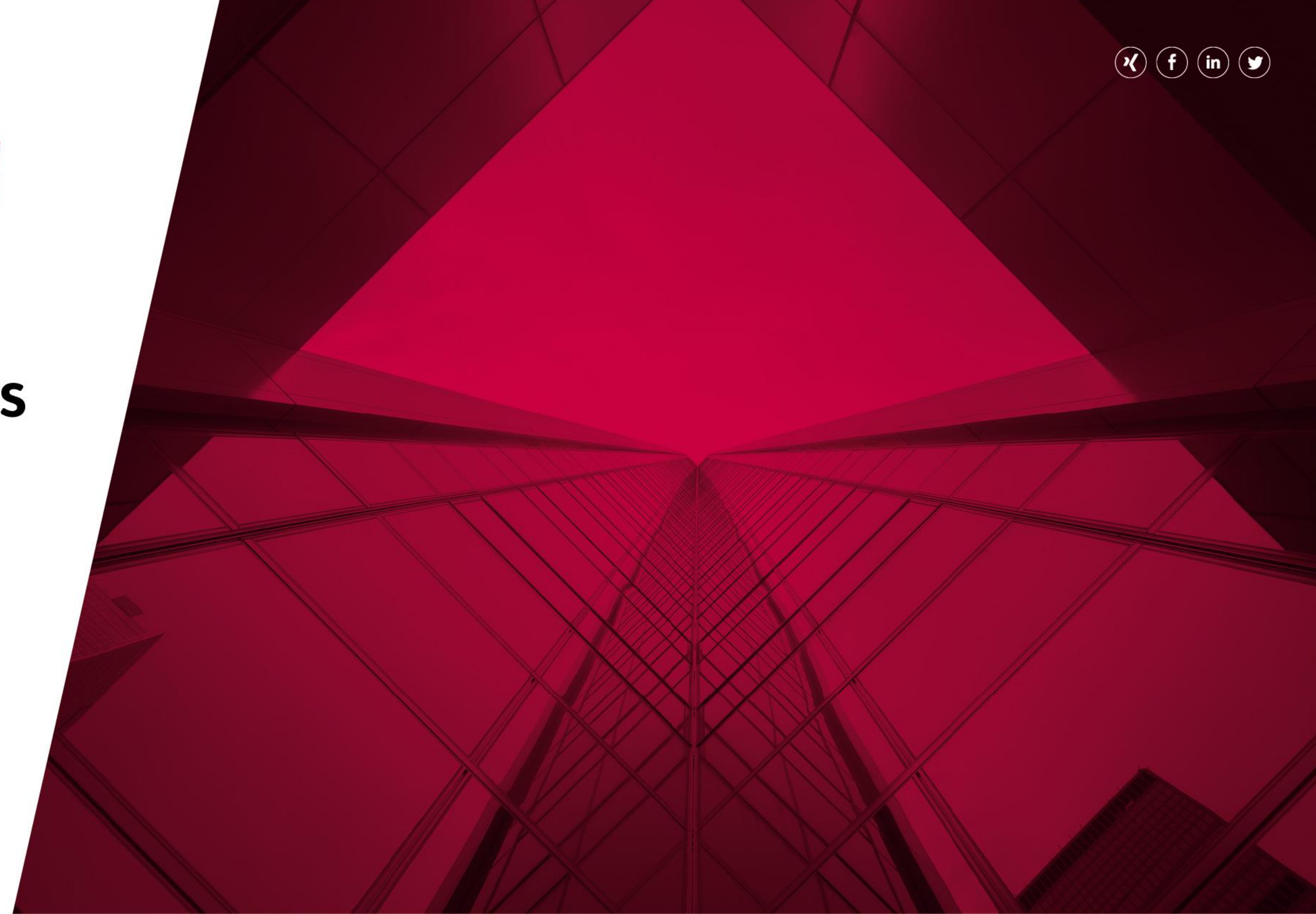
ANGULAR

ARCHITECTS

INSIDE KNOWLEDGE



Workshop Angular Testing

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Task Testing 0

Preparation for Testing Tasks

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Testing

Some core ideas

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

Testing in Angular

- Nearly every 1st class Angular Library provides an API for testing.
- Angular ships with Jasmine and Karma
- We can write Unit Tests
- We can write Template Tests for our component

Unit Testing

- code level
- every component can be unit tested (!)
- isolated testing
- Every dependency will be mocked and stubbed

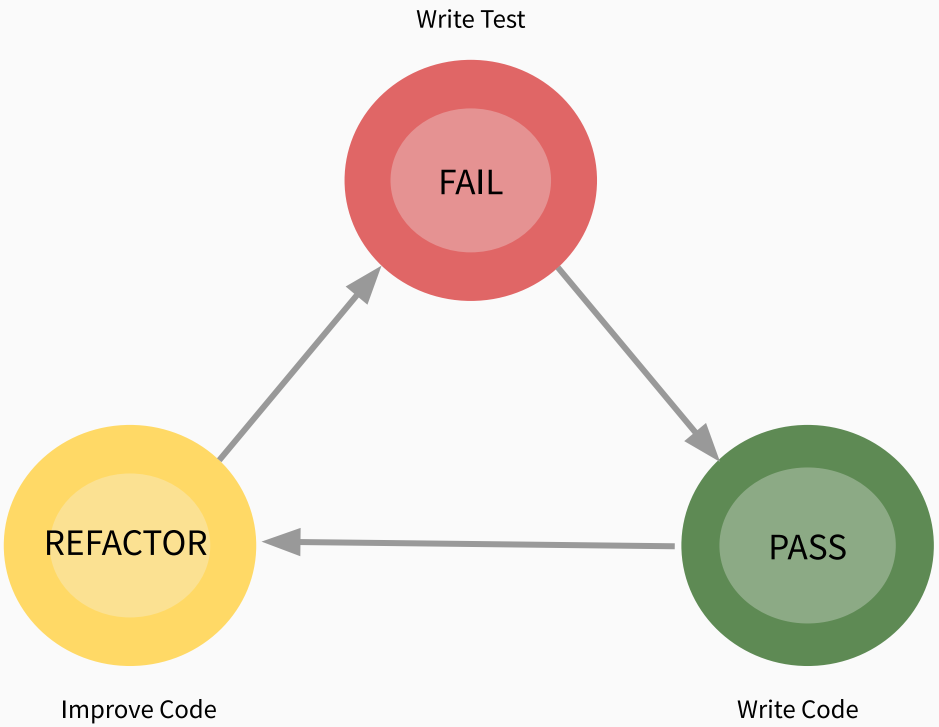
Integration Testing

- code level
- Testing component with its dependencies
- Takes sometimes a lot effort to implement
- If isolated unit test doesn’t make sense

E2E Testing

- User level (Browser)
- Browser robot
- Assertions against the document

Test Driven Development (TDD)



Test Driven Development (TDD)

1. Write a test case and make sure it fails. (red)
2. Satisfy the test case with minimal effort. (green)
3. Improve/refactor your code…
   a. Meet general code guidelines.
   b. Make it readable and comprehensible.
   c. Remove redundant code.
4. Verify that the test case is still passing. (green)

The tools

Jasmine

Karma

Cypress

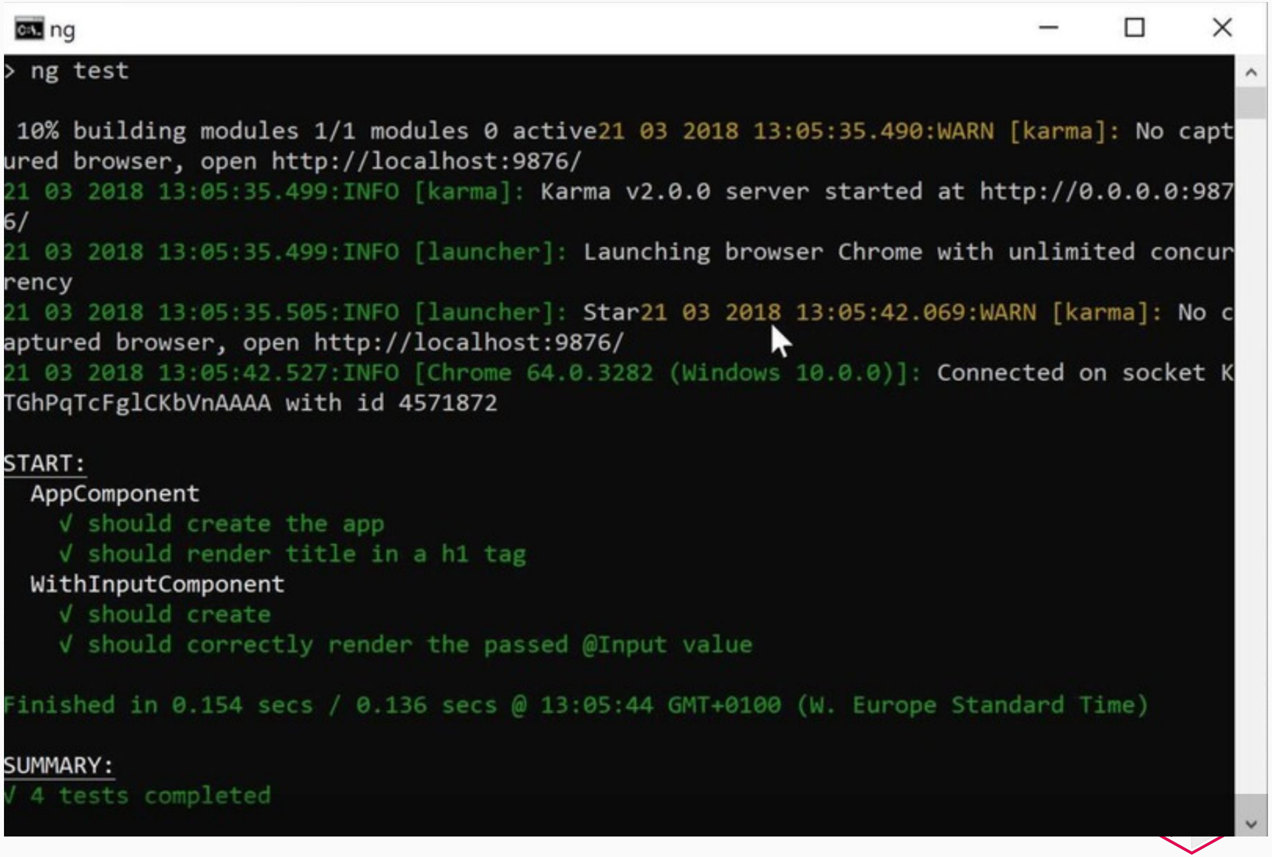
Karma

Angular Test runner

Karma

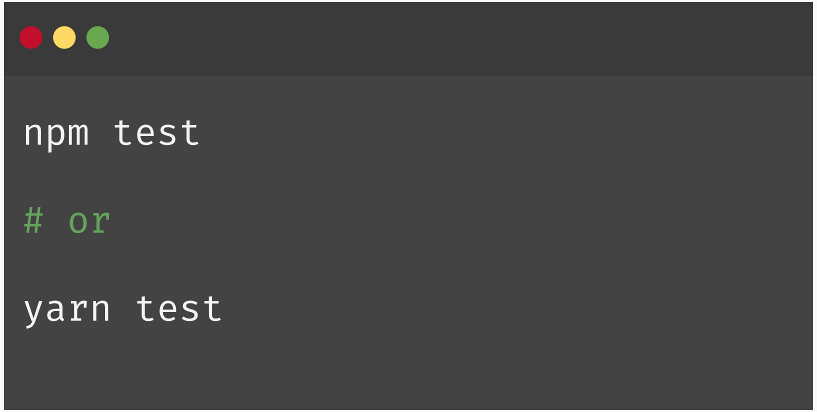
- Test runner
- Spawns browser & runs tests
- Also on command line

Karma



```
> ng test

 10% building modules 1/1 modules 0 active21 03 2018 13:05:35.490:WARN [karma]: No capt
ured browser, open http://localhost:9876/
21 03 2018 13:05:35.499:INFO [karma]: Karma v2.0.0 server started at http://0.0.0.0:987
6/
21 03 2018 13:05:35.499:INFO [launcher]: Launching browser Chrome with unlimited concur
rency
21 03 2018 13:05:35.505:INFO [launcher]: Star21 03 2018 13:05:42.069:WARN [karma]: No c
aptured browser, open http://localhost:9876/
21 03 2018 13:05:42.527:INFO [Chrome 64.0.3282 (Windows 10.0.0)]: Connected on socket K
TGhPqTcFglCKbVnAAAA with id 4571872

START:
  AppComponent
    √ should create the app
    √ should render title in a h1 tag
  WithInputComponent
    √ should create
    √ should correctly render the passed @Input value

Finished in 0.154 secs / 0.136 secs @ 13:05:44 GMT+0100 (W. Europe Standard Time)

SUMMARY:
√ 4 tests completed
```

Run tests with @angular/cli



```
npm test

# or

yarn test
```

Unit Tests

Jasmine

Test method names should be sentences:

```
describe("Testsuite", () => {
  it("should assert something", () => {
    // ...
  });
});
```

Jasmine Basics

Test Suite:	`describe()`
Test Case:	`it()`
Setup:	`beforeEach()`
Tear Down:	`afterEach()`
Assert:	`expect()`

Test Suites can be nested!

Jasmine

<code>

```
describe("Testsuite", () => {
 beforeEach(() => { });
 beforeAll(() => { });
 afterEach(() => { });
 afterAll(() => { });

it("should do sth correctly", () => {
   expect(true).toBe(true);
  });
});
```

Jasmine Matchers

- `toBe()`
- `toEqual()`
- `toContain()`
- `toBeUndefined()`
- `toBeTruthy()`
- `toBeFalsy()`
- `toThrow()`
- `toBeGreaterThan()`
- `toBeLessThan()`
- `toBeCloseTo()`
- `...`

Mocks and Stubs

Mocks:

- Can meet expectations and can cause your tests to fail
- Are mostly part of the framework or library
- Comparing start and end state

Stubs:

- Are objects or classes to let your tests run in general
- Are mostly implemented by yourself
- Testing the correct behaviour as well

Task Testing 1

Component Unit

Testing Pipes

Basics

Creating Pipe instance before each test is running

```
let pipe: FilterPipe;

beforEach(() => {
    pipe = new FilterPipe();
});
```

Testing the transform function

<code>

```
it("return correct filtered Books for searchTerm", () => {
    const filteredBooks = pipe.transform(books, "searchTerm");
    expect(filteredBooks.length).toBe(1);
  });
});
```

Task Testing 2

Pipe

Integration Tests

TestBed for creating TestModules

Helper Function - Testbed

- Configuration and initialisation of the environment to unit test Angular apps
- Generates NgModule
- Methods to create and use services and components

Helper Function - Arrange

<code>

Testbed - generate NgModule with declarations and services

```
TestBed.configureTestingModule({
  declarations: [
    BookComponent
  ],
  providers: [{
    provide: BookApiService,
    useFactory: () => bookApiMock
  }]
});
```

Helper Function - Act

<code>

```
beforEach(() => {
    fixture = TestBed.createComponent(BookComponent);
    component = fixture.componentInstance
    fixture.detectChanges();
});
```

Mock dependencies

Dependencies to Mock

- Child components need to be either mocked mocked or declared
- Pipes and Services needs to be mocked or the module providing them imported

Mocking Child Components 1

Mock them

```
@Component({
 selector: 'app-book-card',
 template: '<div></div>'
})
class DummyBookCardComponent {
 @Input() content!: Book;
}
```

Mocking Child Components 2

Ignore them: NO_ERRORS_SCHEMA

```
beforeEach(async () => {
   await TestBed.configureTestingModule({
       declarations: [ BookComponent ],
       schemas: [ NO_ERRORS_SCHEMA ]
   })
       .compileComponents();
});
```

Mocking Child Components 3

Integrate them

```
beforeEach(async () => {
   await TestBed.configureTestingModule({
       declarations: [ BookComponent, BookCardComponent ],
   })
       .compileComponents();
});
```

Mocking services

Type-safe mocks

<code>

Jasmine ships with a type-safe mock API.

```
let bookApiMock: jasmine.SpyObj<BookApiService>;

// ...

bookApiMock = jasmine.createSpyObj<BookApiService>(['getAll']);
```

Type-safe mocks

The outcome of a property or method can be set.

```
bookApiMock.getAll.and.returnValue(of(book));
```

Testing Observables

<code>

```
it('call observable', () => {
   component.books$.subscribe((book) => {
           expect(book.length).toBe(2)
       }
   )
});
```

Testing Observables

```
it('call observable', () => {
   component.books$.subscribe((book) => {
           expect(book.length).toBe(2);
       }
   )
});
```

Might not be triggered and might result in a false passing

Testing Observables

The done functions tells when a test is completed

```
it('call observable', (done) => {
   component.books$.subscribe((book) => {
           expect(book.length).toBe(2);
           done();
       }
   )
});
```

Task Testing 3

Component Mock Dependency

Mock HTTP Backend

Testing Services HttpClient

<code>

Setup for HttpClient testing

```
import { HttpClientTestingModule } from '@angular/common/http/testing';

beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpClientTestingModule],
    providers: [BookApiService]
  });

  httpMock = TestBed.inject(HttpTestingController);
  bookApi = TestBed.inject(BookApiService);
});
```

Testing Services HttpClient

<code>

HttpTestingController

```
import { HttpTestingController } from '@angular/common/http/testing';

let httpMock: HttpTestingController;

// Response
httpMock.expectOne('<endpoint>').flush(responseData);
```

Testing Services HttpClient

<code>

Mock errors

```
// Network Error
httpMock.expectOne('<endpoint>').error(new ErrorEvent('Network error.'));

// API Error
httpMock.expectOne('<endpoint>').flush(
  'No books', { status: 500, statusText: 'The API hung up'
});
```

Testing Services

<code>

Asynchronous tests | Variant 1

```
it('provides books', done => {
  const booksExpected = [<mock test data>];

  bookApi.getAll().subscribe(booksFromApi => {
    expect(booksFromApi).toBe(booksExpected);
    done();
  });

  httpMock.expectOne('<endpoint>').flush(booksExpected);
});
```

ARCHITECTS
INSIDE KNOWLEDGE

Testing Services

<code>

Asynchronous tests | Variant 2

```
it('provides books', async () => {
  const books = [<mock test data>];
  const books$ = bookApi.getAll().toPromise();

  httpMock.expectOne('<endpoint>').flush(books);

   // success
  await expectAsync(books$).toBeResolvedTo(books);
   //failure
   await expectAsync(books$).toBeRejectedWithError('sorry');
});
```

Testing Services HttpClient

Verify no unhandled Http Request is left

```
afterEach(() => httpMock.verify());
```

Task Testing 4

HTTP Mock Backend

@angular/material
Component Harness

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Testing complex components like *calendars*, *steppers* or even *form-fields with validation* can be hard.

The Angular team provides an abstraction layer to make testing their component library easier.

https://material.angular.io/cdk/test-harnesses/overview

Component Harness

<code>

Setup Harness Environment

```
import { TestbedHarnessEnvironment } from '@angular/cdk/testing/testbed';

let fixture: ComponentFixture<BookNewComponent>;
let loader: HarnessLoader;

// ...

fixture = TestBed.createComponent(BookNewComponent);
loader = TestbedHarnessEnvironment.loader(fixture);
```

Component Harness

<code>

Get Harness for specific material component

```
import {
  MatFormFieldHarness
} from '@angular/material/form-field/testing';

const isbnFormField = await loader.getHarness(
  MatFormFieldHarness
);
```

Component Harness

<code>

Specify resilient selectors

```
import {
  MatFormFieldHarness
} from '@angular/material/form-field/testing';

const isbnFormField = await loader.getHarness(
  MatFormFieldHarness.with({ selector: '[data-test=isbn-field]' })
);
```

Component Harness

Access child material component

```
import { MatInputHarness } from '@angular/material/input/testing';

const isbnFormField = await loader.getHarness(/* ... */);

const isbnInput = (await isbnFormField.getControl()) as MatInputHarness;
```

Component Harness

<code>

Interact with material component

```
await isbnInput.setValue('12');

await isbnInput.blur();
```

Component Harness

<code>

Component-Harness-API is asynchronous

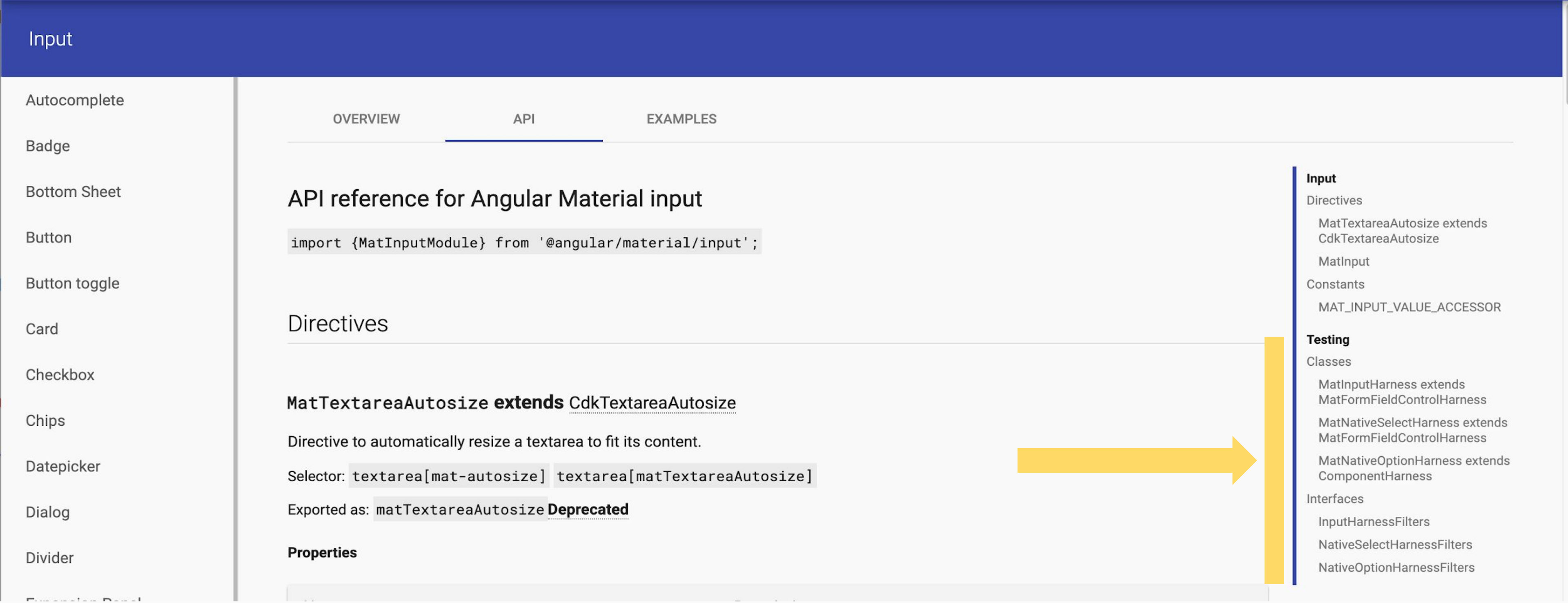
```
it('test with Angular Material component', async () => {

  const isbnFormField = await loader.getHarness(MatFormFieldHarness);

  const isbnErrors = await isbnFormField.getTextErrors();

})
```

Component Harness

<code>

Get state information from component.

```
const isbnErrors = await isbnFormField.getTextErrors();



expect(isbnErrors).toContain(
  'ISBN has to be at least 3 characters long.'
);
```


Input
Autocomplete
Badge
Bottom Sheet
Button
Button toggle
Card
Checkbox
Chips
Datepicker
Dialog
Divider
Expansion Panel
OVERVIEW
API
EXAMPLES
Input
Directives
MatTextareaAutosize extends CdkTextareaAutosize
MatInput
Constants
MAT_INPUT_VALUE_ACCESSOR
Testing
Classes
MatInputHarness extends MatFormFieldControlHarness
MatNativeSelectHarness extends MatFormFieldControlHarness
MatNativeOptionHarness extends ComponentHarness
Interfaces
InputHarnessFilters
NativeSelectHarnessFilters
NativeOptionHarnessFilters

API reference for Angular Material input

```
import {MatInputModule} from '@angular/material/input';
```

Directives

`MatTextareaAutosize` extends CdkTextareaAutosize

Directive to automatically resize a textarea to fit its content.

Selector: `textarea[mat-autosize]` `textarea[matTextareaAutosize]`

Exported as: `matTextareaAutosize` **Deprecated**

Properties

💁 Each Material Component ships with its Test-API. Check out the Documentation to learn about the details.

Task Testing 5

Material Component Harness

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE