



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Testing

6 - Component & Integration: Fallback

Harnesses



Test Harnesses

- Page Object Models for Component Tests
- Available since Angular v9
- Provide a test abstraction for components
- Developed by @angular/material
- Full coverage for material since v11
- Reduces code size significantly
 - Better Readability
 - Better Maintainability



Address Validation

Address

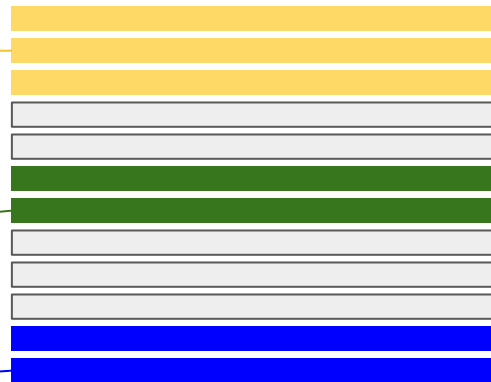
Domgasse 5



Please enter your address

Submit

Address found



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Address Validation

Address

Domgasse 5

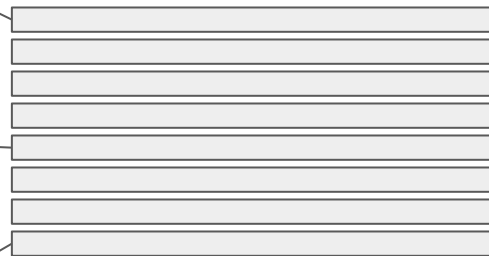
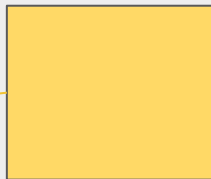


Please enter your address

Submit

Address found

Harness



- Element Selection
- Change Detection
- Asynchronicity
- Rendering



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Creating a Harness

```
export class RequestInfoComponentHarness extends ComponentHarness {  
  
    static hostSelector = "eternal-request-info";  
  
    protected getButton = this.locatorFor("[data-testid=ri-search]");  
  
    async submit(): Promise<void> {  
        const button = await this.getButton();  
        return button.click();  
    }  
}
```



Using a Harness

```
it("should use the harness", async () => {  
  // setup TestModule...  
  
  const harness = await TestBedHarnessEnvironment.harnessForFixture(  
    fixture,  
    RequestInfoComponentHarness  
  );  
  await harness.submit();  
  // expect something  
});
```



When to use it?

- Shared UI Components
- In Combination with Testing Library or Spectator
- "Page Objects" in E2E



Spectator



Spectator

- Unopinionated TestBed Wrapper
- Great Support for Mocking
- Applicable for various testing types
- Very popular in Angular
- Powered by ngneat



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Setup

```
const createComponent = createComponentFactory({  
  component: RequestInfoComponent,  
  mocks: [AddressLookuper]  
});  
  
const setup = () => {  
  return { spectator: createComponent() };  
};
```



Tests

```
it(`should show ${message} for ${address}`, fakeAsync(() => {  
  const { spectator } = setup();  
  spectator  
    .inject(AddressLookuper)  
    .lookup.mockImplementation((query: string) =>  
      scheduled([query === 'Domgasse 5'], asyncScheduler)  
    );  
  spectator.typeInElement(address, byTestId('ri-address'));  
  spectator.click(byTestId('ri-search'));  
  spectator.tick();  
  
  expect(spectator.query(byTestId('ri-message'))).toHaveText(message);  
}));
```



When to use it?

- If Cypress or Testing Library is not an option
- If you have already Spectator-based tests



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Testing Library



Testing Library

- Huge ecosystem
- Enforces a Testing Philosophy
- Superior UI Debugging Tools
- find* removes (most) Asynchrony and Change Detection
- userEvent with real-world behaviour
- Common API for multiple frameworks (Cypress,...)
- Host Components very easy



Configuring the TestingModule

Classic

```
TestBed.configureTestingModule({
  declarations: [NewsletterComponent],
  imports: [ReactiveFormsModule],
  providers: [
    { provide: NewsletterService,
      useClass: MockedNewsletterService },
  ],
});

const fixture = TestBed
  .createComponent(NewsletterComponent);
const component = fixture.componentInstance;
fixture.detectChanges()
```

Testing Library

```
render(NewsletterComponent, {
  declarations: [NewsletterComponent],
  imports: [ReactiveFormsModule],
  providers: [
    { provide: NewsletterService,
      useClass: MockedNewsletterService },
  ],
});
```

No component creation and
detectChanges() needed



Configuration with static HTML

```
render(`<eternal-newsletter email="user@host.com"></eternal-newsletter>`, {  
  declarations: [NewsletterComponent],  
  imports: [ReactiveFormsModule],  
  providers: [  
    { provide: NewsletterService, useValue: MockedNewsletterService },  
  ],  
});
```



Selecting DOM Elements

Classic

```
const button = fixture.debugElement.query(  
  By.css("[data-testid=btn-submit]")  
);
```

Testing Library

```
const button = screen.getByTestId("btn-search");
```

```
// Alternatives
```

```
// can also be null
```

```
const button = screen.queryByTestId("btn-search");
```

```
// internally triggers asynchronous tasks and CD
```

```
const button = await screen.findByTestId("btn-search"); // 👍
```



Actions

Classic

```
button.click();  
fixture.detectChanges();
```

Testing Library

```
fireEvent.click(button);
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Extension userEvent - real DOM Events

@testing-library/user-event

```
// v1
await user.click(screen.getByTestId("btn-submit"));

// v2
await user.click(await
screen.findByTestId("btn-submit"));
```

- Dispatches Events like an end-user
- Works best with 3rd party components
- Provides support for
 - **Writing to inputs**
 - Mouse movements
 - Clipboard interaction
 - ...



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Extension selectors: "mini page objects"

testing-library-selector

Before

```
screen.getByTestId<HTMLButtonElement>('btn-submit');  
screen.queryByTestId<HTMLButtonElement>('btn-submit');  
  
await screen  
  .findByTestId<HTMLButtonElement>('btn-submit');
```

After

```
const ui = {  
  button: byTestId<HTMLButtonElement>('btn-submit')  
};  
  
ui.button.get()  
ui.button.query()  
await ui.button.find()
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Setup

```
const ui = {  
  address: byTestId("ri-address"),  
  search: byTestId("ri-search"),  
  message: byTestId("ri-message"),  
};  
  
const setup = async () => {  
  const lookuper = createMock(AddressLookuper);  
  const renderResult = await render(RequestInfoComponent, {  
    providers: [{ provide: AddressLookuper, useValue: lookuper }],  
  });  
  const user = userEvent.setup();  
  
  return { ...renderResult, lookuper, user };  
};
```



Tests

```
it(`should show ${message} for ${address}`, async () => {  
  const { lookuper, user } = await setup();  
  lookuper.lookup.mockImplementation((query: string) =>  
    scheduled([query === "Domgasse 5"], asyncScheduler)  
  );  
  
  await user.type(ui.address.get(), address);  
  await user.click(screen.getByTestId("ri-search"));  
  const messageEl = await screen.findByTestId("ri-message");  
  expect(messageEl.textContent).toBe(message);  
});
```



When to use it?

- For all tests where DOM interaction is required
- If Cypress is not an option



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE