Cypress in Action
How to integrate Cypress in your development cycle

Magdalena Sawyer

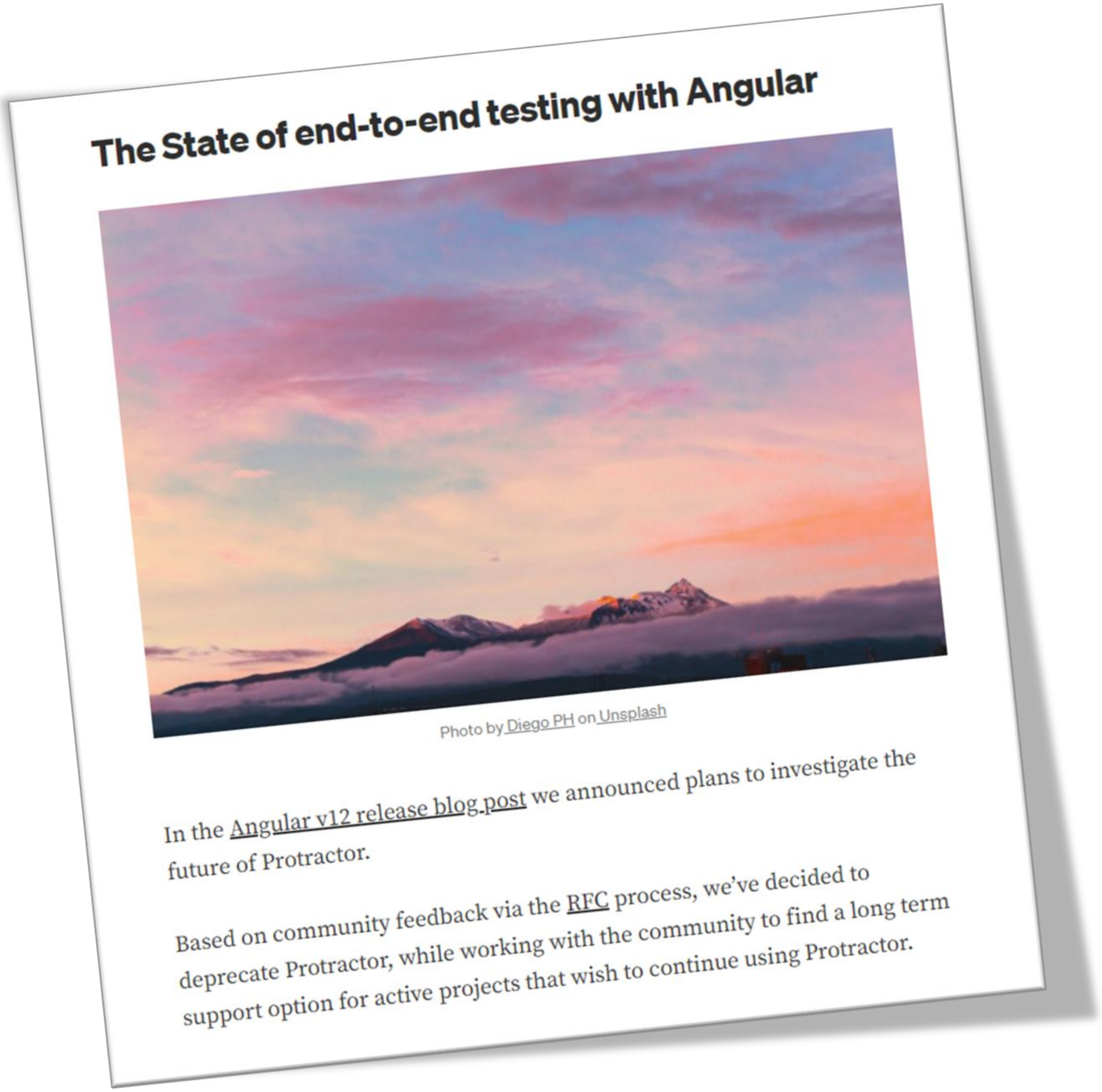
Senior Consultant @ 2BIT

2BIT
IT's our passion.

Hi, I’m Magda!

Senior Consultant @ 2BIT

Fullstack software engineer



The State of end-to-end testing with Angular

Photo by Diego PH on Unsplash

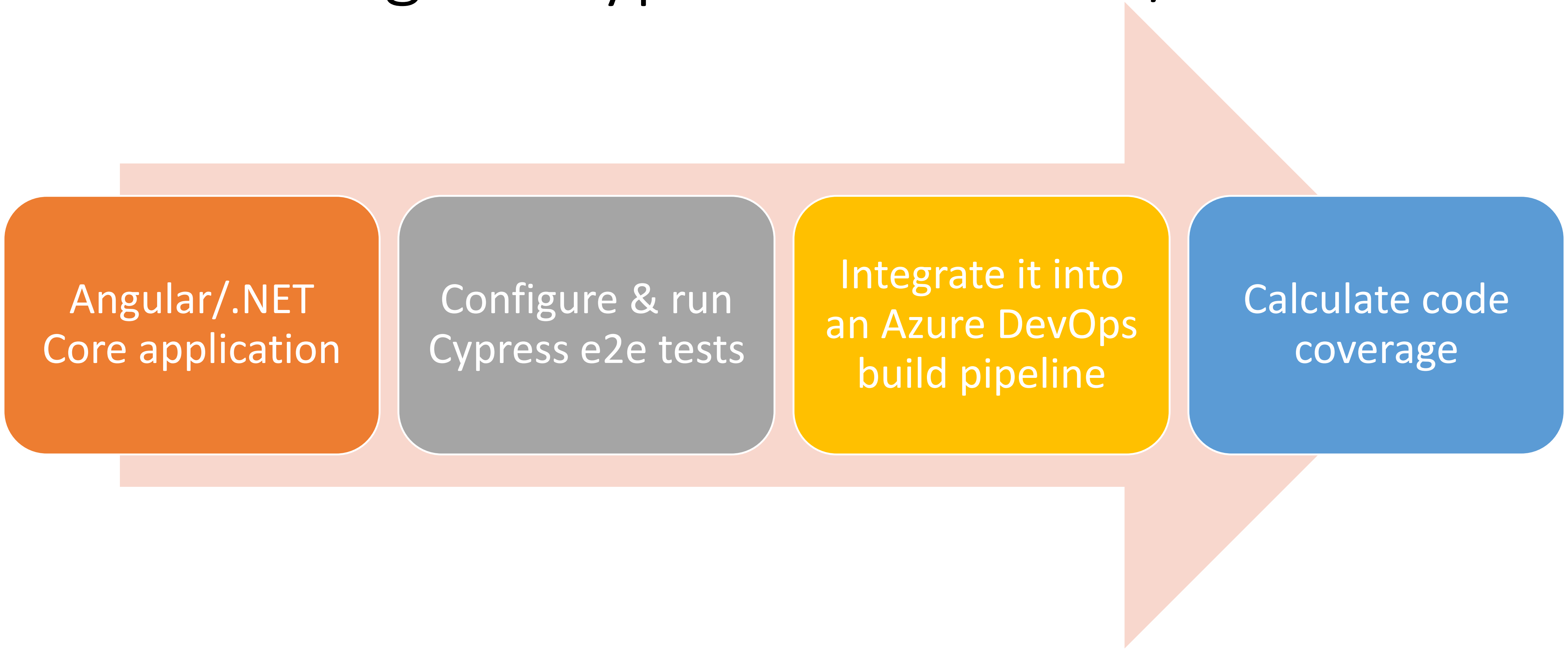
In the Angular v12 release blog post we announced plans to investigate the future of Protractor.

Based on community feedback via the RFC process, we've decided to deprecate Protractor, while working with the community to find a long term support option for active projects that wish to continue using Protractor.

Why are we talking about Cypress?

https://blog.angular.io/the-state-of-end-to-end-testing-with-angular-d175f751cb9c

Goal: Configure Cypress tests in CI/CD


Angular/.NET Core application
Configure & run Cypress e2e tests
Integrate it into an Azure DevOps build pipeline
Calculate code coverage

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

What is Cypress?

- Cypress is a complete end-to-end testing framework that allows you to write, run and record tests.
- No Selenium
- Uses many well-known open-source testing libraries (Mocha, Chai, Sinon.JS)
- Executed in the browser
- Native access to <u>everything</u>!
- Read and alter network traffic on the fly
- Access your operating system to take screenshots and videos
- Works with any front-end framework or website

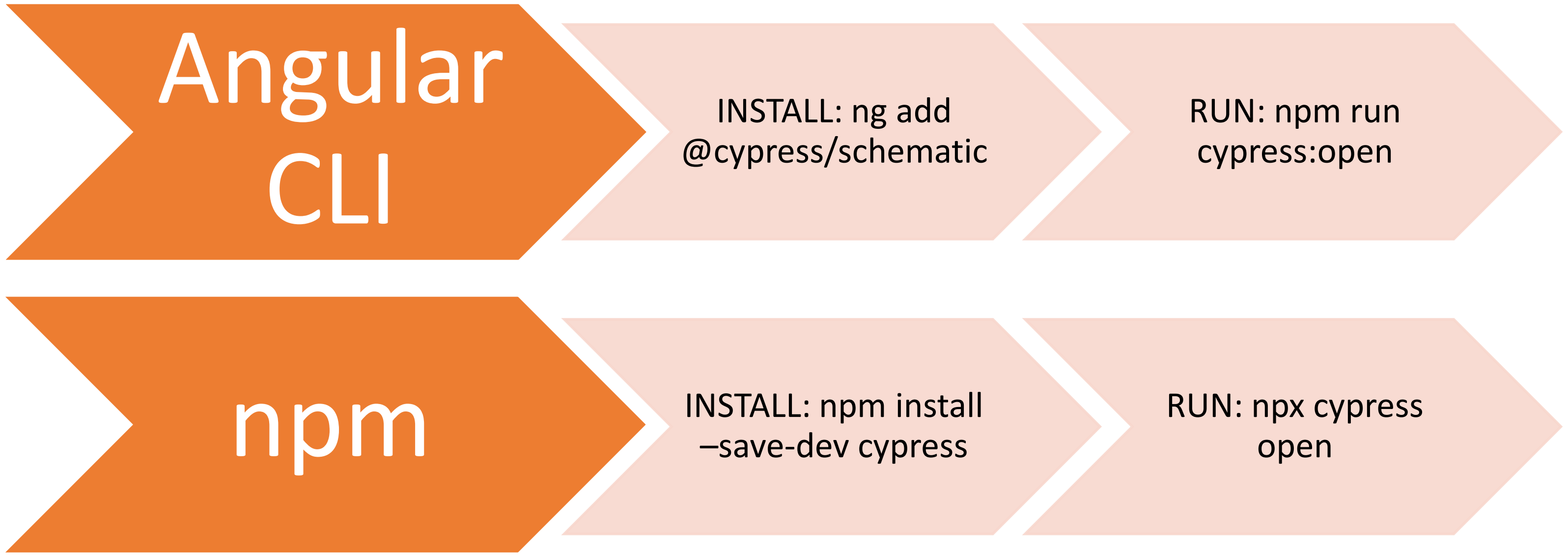
Why should you use it?

- Much faster than Selenium based frameworks
- Not flaky
- Simple enough to be used by developers as well as QA engineers
- Real time reloads
- Easy visual interface that allows you to examine tests step by step
- Easy to debug
- Allows to stub REST responses
- Easy to integrate with Angular

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

Working with Cypress locally



Angular CLI

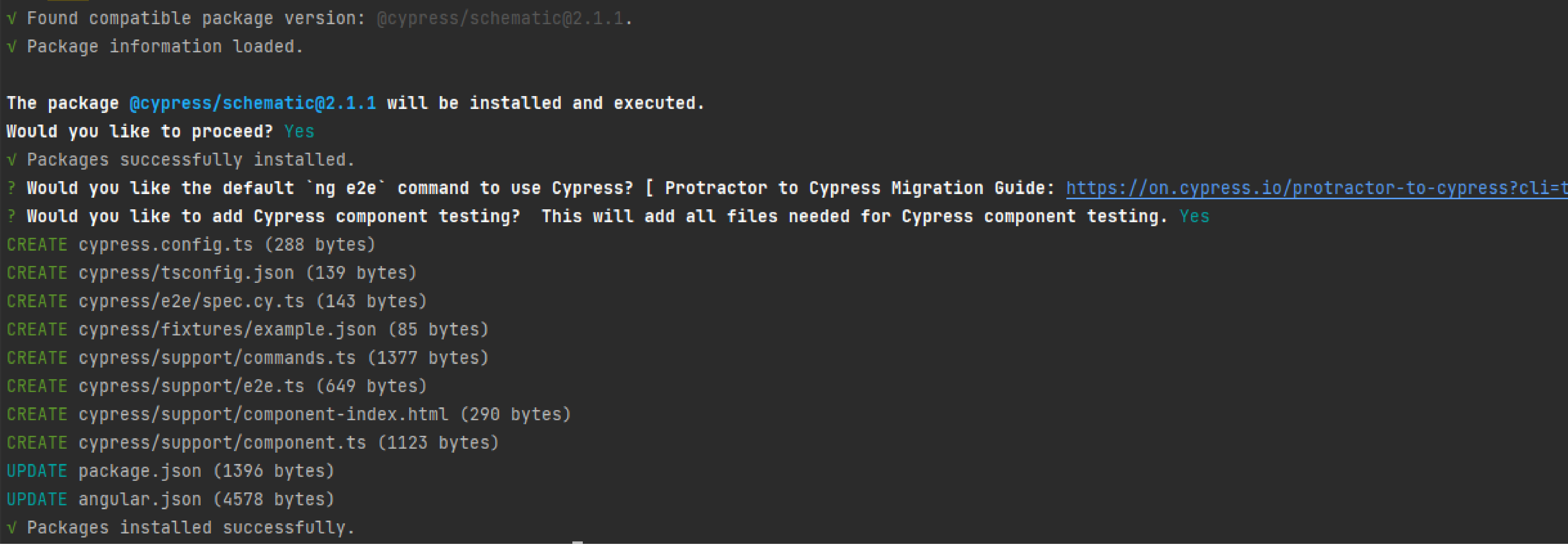
INSTALL: ng add @cypress/schematic

RUN: npm run cypress:open

npm

INSTALL: npm install –save-dev cypress

RUN: npx cypress open



```
√ Found compatible package version: @cypress/schematic@2.1.1.
√ Package information loaded.

The package @cypress/schematic@2.1.1 will be installed and executed.
Would you like to proceed? Yes
√ Packages successfully installed.
? Would you like the default `ng e2e` command to use Cypress? [ Protractor to Cypress Migration Guide: https://on.cypress.io/protractor-to-cypress?cli=t
? Would you like to add Cypress component testing?  This will add all files needed for Cypress component testing. Yes
CREATE cypress.config.ts (288 bytes)
CREATE cypress/tsconfig.json (139 bytes)
CREATE cypress/e2e/spec.cy.ts (143 bytes)
CREATE cypress/fixtures/example.json (85 bytes)
CREATE cypress/support/commands.ts (1377 bytes)
CREATE cypress/support/e2e.ts (649 bytes)
CREATE cypress/support/component-index.html (290 bytes)
CREATE cypress/support/component.ts (1123 bytes)
UPDATE package.json (1396 bytes)
UPDATE angular.json (4578 bytes)
√ Packages installed successfully.
```

ng add @cypress/schematic

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

Testing strategies

Use server responses

Pros:

- More likely to work in producion
- Allows you to test server endpoint
- Works with server-side HTML rendering

Cons:

- Requires seeding data
- Much slower
- Hard to test edge cases

Stub responses

Pros:

- Control of response bodies, status, and headers
- Can simulate network delay or failure
- No code changes to your server or client code

Cons:

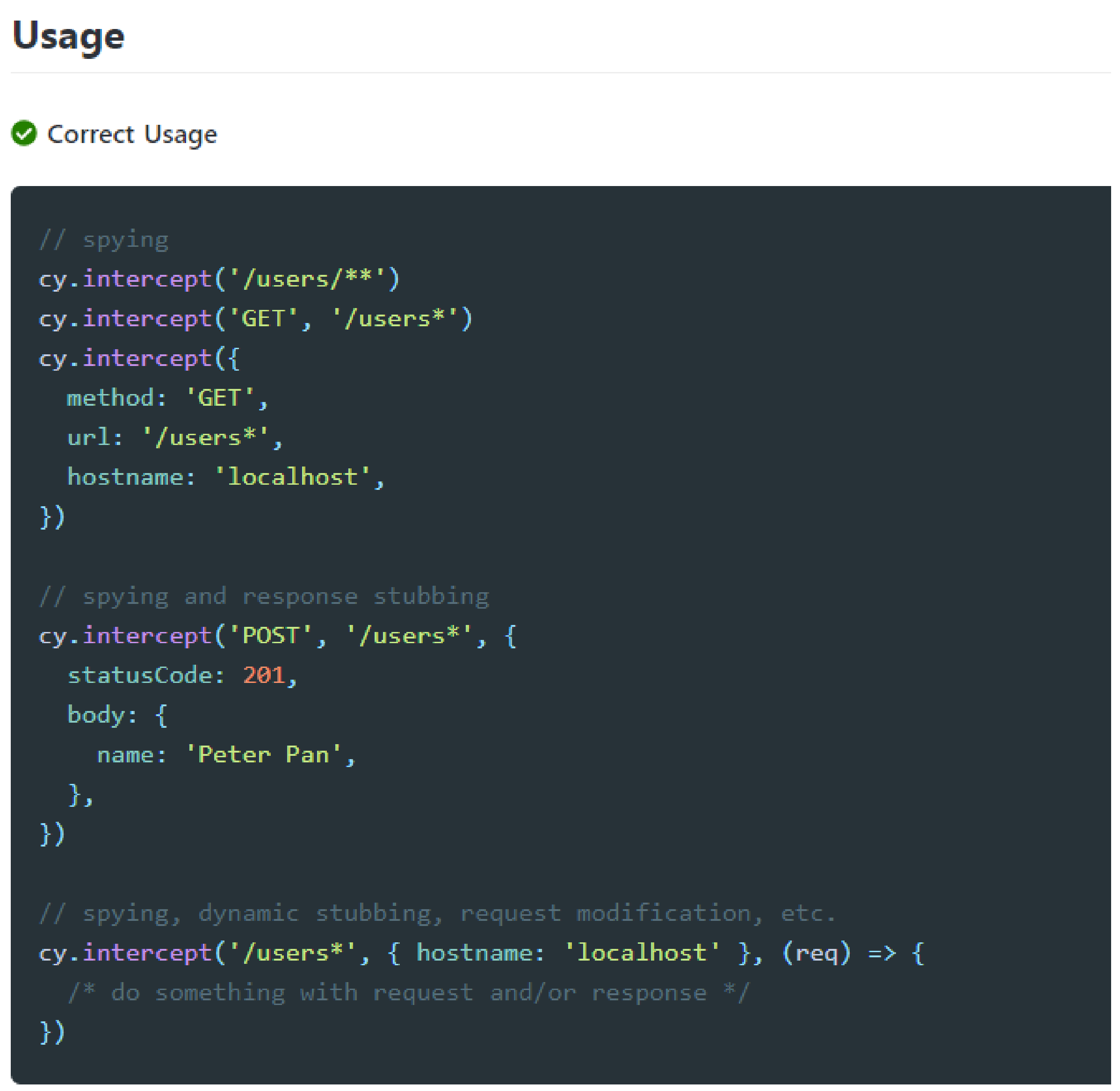
- No guarantee your stubbed responses match the actual data
- No test coverage on some server endpoints
- Not as useful if you're using traditional server-side HTML rendering

Which testing strategy to use?

USE BOTH!

- Mix and match, typically have one true end-to-end test, and then stub the rest
- Use server responses to test critical paths of your application
- Use stubbed responses to for majority of tests, making sure you cover edge cases

Intercepting & stubbing requests - DEMO



Usage

✓ Correct Usage

```
// spying
cy.intercept('/users/**')
cy.intercept('GET', '/users*')
cy.intercept({
  method: 'GET',
  url: '/users*',
  hostname: 'localhost',
})

// spying and response stubbing
cy.intercept('POST', '/users*', {
  statusCode: 201,
  body: {
    name: 'Peter Pan',
  },
})

// spying, dynamic stubbing, request modification, etc.
cy.intercept('/users*', { hostname: 'localhost' }, (req) => {
  /* do something with request and/or response */
})
```

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

CI/CD setup – where to run the app?

Inside the pipeline

Pros:

- The simplest way

Cons:

- Messy when you have a complex environment setup
- Need to configure everything again when changing your CI/CD server

Deployed on external test server

Pros:

- Easiest to setup
- Could be an option if you don’t need instant feedback and you want to run your tests e.g. nightly

Cons:

- complicates the workflow: cannot be part of the same test pipeline, you need to deploy your application first and then run tests
- Requires a separate test environment

Inside a Docker container

Pros:

- Can use same/simmilar config as production
- Isolated environment

  Portable - easy to migrate to different CI/CD

Cons:

- Requires more time and understanding Docker

Demo – how to integrate Cypress in the CI/CD pipeline using Docker

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

Code Coverage

It is not enough to know your tests are passing – you need to know if you are testing enough

Code coverage gives you a metric to assess if you need more tests

Next steps – combined code coverage from unit and e2e tests

Code Coverage – prerequisites

- Instrument your application code using e.g. Istanbul.js plugin
- DO NOT run your application from production build
- The example from Cypress website uses a React app, so instrumentation part is different for Angular

Code Coverage - setup

1. Instrument your code using Istanbul plugin:
   - Install npm libraries: npm i -D @jsdevtools/coverage-istanbul-loader @istanbuljs/nyc-config-typescript istanbul-lib-coverage nyc webpack
   - Create .nycrc configuration file for your
   - Create cypress/coverage.webpack.ts file
   - Modify angular.json – use custom webpack config for CI e2e setup
   - Add e2e:ci script to package.json
   - Add code coverage to Cypress:
   - npm i -D @cypress/code-coverage
   - Add to e2e.ts: import '@cypress/code-coverage/support';
   - Add to cypress.config.ts: require('@cypress/code-coverage/task')(on, config)
2. Run your app and tests using the script

Add code coverage to Azure DevOps pipeline

- Azure DevOps requires coverage to be pubished in a JaCoCo or Cobertura reporter format
  - nyc supports Cobertura out of the box – just need to add it as a reporter in your .nycrc file
- Mount coverage folders in docker-compose.yml

Agenda

- What is Cypress and why use it?
- Working with Cypress locally
- Testing strategies
- Integrating Cypress tests in Azure DevOps using Docker
- Code coverage
- Lessons learned
- Discussion

Lessons learned

- No one-fits-all setup possible: choices depend on frameworks used, amount of config needed, login method, external tools used, team skillset
- Each approach has trade-offs – take time to analyze which ones are acceptable for your usecase
- Writing tests is easy – it is the setup that takes a lot of time and experience
- Taking breaks and working in small chunks can be more productive than pushing yourself to finish something in one go, especially when you are stuck

Resources

- Cypress homepage: https://www.cypress.io/
- Angular blog on state of testing 2022: https://blog.angular.io/the-state-of-end-to-end-testing-with-angular-d175f751cb9c
- Testing Angular: https://testing-angular.com/
- Using docker-compose with Cypress: https://github.com/cypress-io/cypress-example-docker-compose
- Configuring code coverage: https://lukas-klement.medium.com/implementing-code-coverage-with-angular-and-cypress-6ed08ed7e617
- Combined code coverage: https://dev.to/muratkeremozcan/combined-unit-e2e-code-coverage-case-study-on-a-real-life-system-using-angular-jest-cypress-gitlab-35nk

GitHub repository

Thank you!

...tell me what you think!

LET'S STAY IN TOUCH

www.2bit.ch

https://ch.linkedin.com/company/2bit-gmbh

https://www.meetup.com/zurich-progressive-web-app-meetup/

2BIT

IT's our passion.