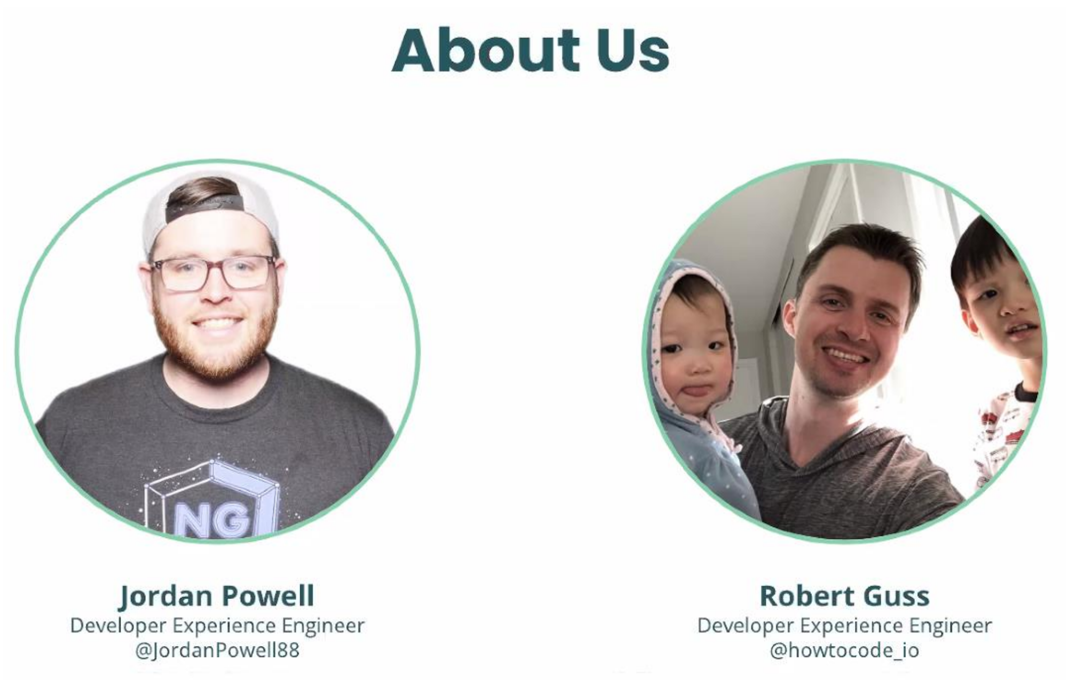
<u>Jordan Powell and Robert Guss - Cypress</u>
Developer experience engineers from Cypress



<u>Intro to Testing</u>

Limited notes as mostly recapped all we have learnt so far about testing.

Test a single "unit" in your application
They should not depend upon other parts of the system or application

Integration tests check how your functions work together

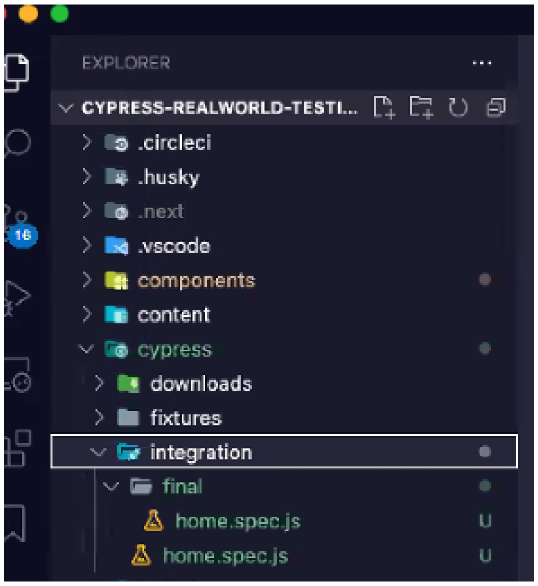
End to end tests tend to be written once your product has been built

<u>Demo of Cypress</u>

Cypress always tests inside a browser and is testing what the end user gets. This means it only deals with HTML, CSS and JS – it is language agnostic in the way that it doesn't matter what's behind it all (the tech stack) – Node, React etc.

To install:
npm I cypress
Open it using:
npx cypress open (opens a browers)
Select browser in top right (it detects what you have installed so you can use it)
Lots of built in examples that teach you how to test for certain things (such as local storage, navigation etc)

Although it run tests really quickly, but has time travel feature so you can check specific actions and check exactly what was happening at that point



Can use then click on an element and Cypress will show you the code to grab that element to use when writing your tests (and can copy to clipboard then paste into VS code)

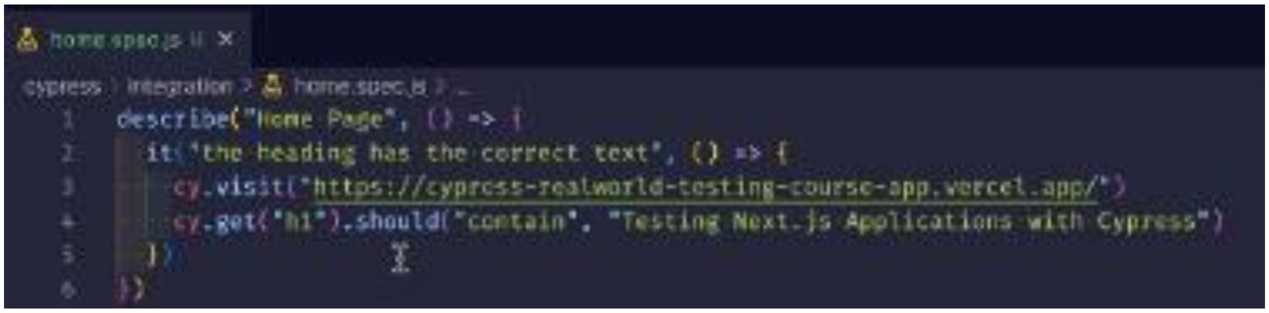
When you install cypress, it creates a cypress folder and you should then write your scripts inside the integration folder. The naming convention is *filenametotest*.spec.js. (don't need *final* folder, that was just for the demo so he had the completed code as well as starting code):

Don't need to .gitignore the Cypress folder as you want the tests included

Whenever you start a new test file, write a describe block. This is a function that takes 2 arguments – a string describing the block and a callback function. Inside callback function, define an "it" function (this is the equivalent of test in Jest) that has a string describing the test and an inline function that is the test:

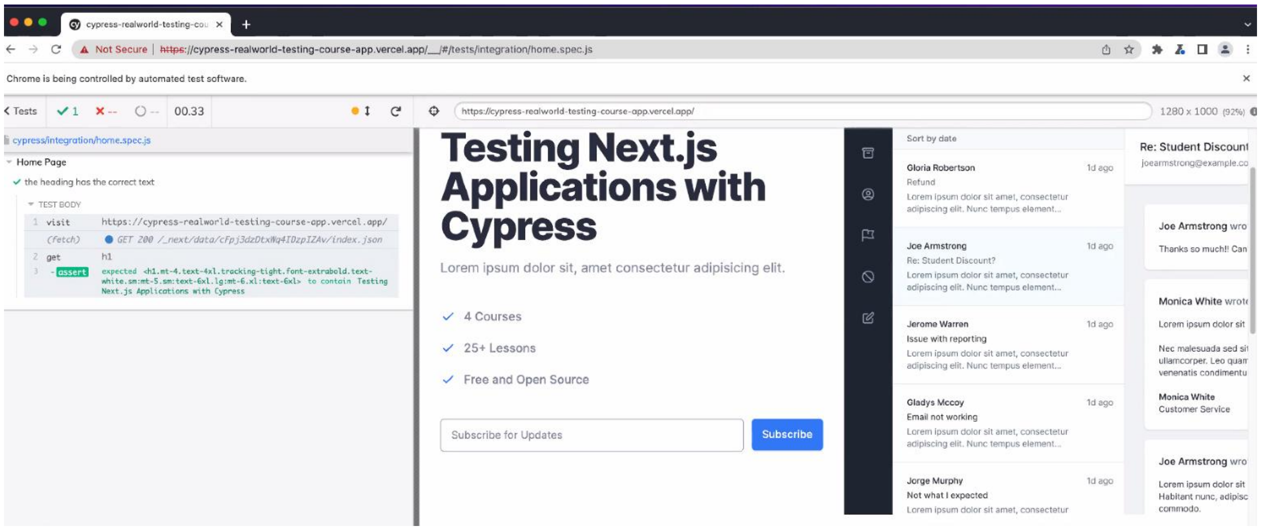
Give it the object to get (start with the URL then give it name of the elemnt you want to test

Cy.visit('*URLname');*
Cy.get('h1'); *takes CSS selectors, ID, or just element name (h1) – be as specific as needed!*
Use .*should* method for what you want to check



```
describe("Home Page", () => {
  it("the heading has the correct text", () => {
    cy.visit("https://cypress-realworld-testing-course-app.vercel.app/")
    cy.get("h1").should("contain", "Testing Next.js Applications with Cypress")
  })
})
```

The page we are testing that has the H1 tag mentioned above. You can see the UI on the left hand side and hovering over steps will show the elements/steps it took to get there.


Chrome is being controlled by automated test software.
Testing Next.js Applications with Cypress
Lorem ipsum dolor sit, amet consectetur adipisicing elit.
4 Courses
25+ Lessons
Free and Open Source
Subscribe for Updates
Subscribe

```
cypress/integration/home.spec.js
Home Page
✓ the heading has the correct text
  TEST BODY
  1  visit      https://cypress-realworld-testing-course-app.vercel.app/
     (fetch)    ● GET 200 /_next/data/cFpj3dzDtxWq4IDzpIZAv/index.json
  2  get        h1
  3  - assert   expected <h1.mt-4.text-4xl.tracking-tight.font-extrabold.text-
                white.sm:mt-5.sm:text-6xl.lg:mt-6.xl:text-6xl> to contain Testing
                Next.js Applications with Cypress
```

All their teaching is open-source, so you can visit this to get a full walkthrough:

https://learn.cypress.io/

Can use recording feature in Chrome Dev Tools, actually click on things, fill in forms etc, then can export text file with all that info and Cypress and import that and convert it for you automatically!