# Lab Activity 1.2

## First Test Code

Now is the time to create your first Cypress code. To create a test file in the Cypress flow, follow the following steps.

Step 1 : open cypress. To open cypress there is two methods

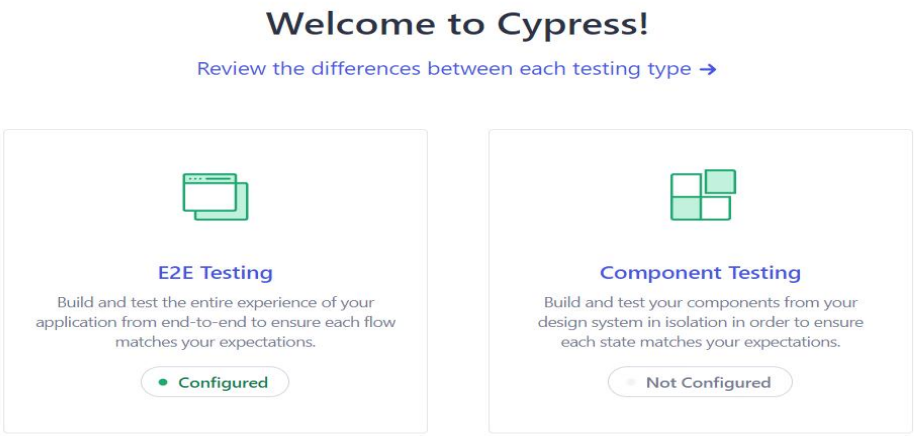
1. On test Runner
2. On Terminal.

On Test runner:- to open cypress on Test Runner run this command on terminal.

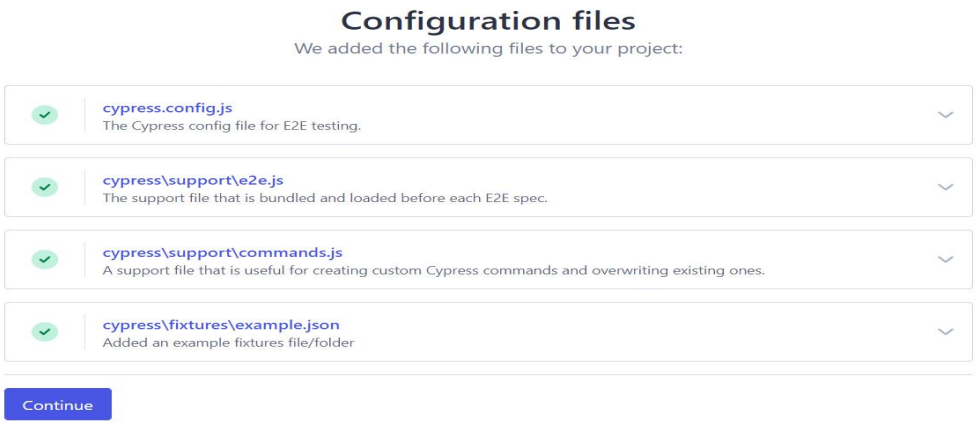
```
npx cypress open
```

For the first time when open cypress we will see the follwing screen.

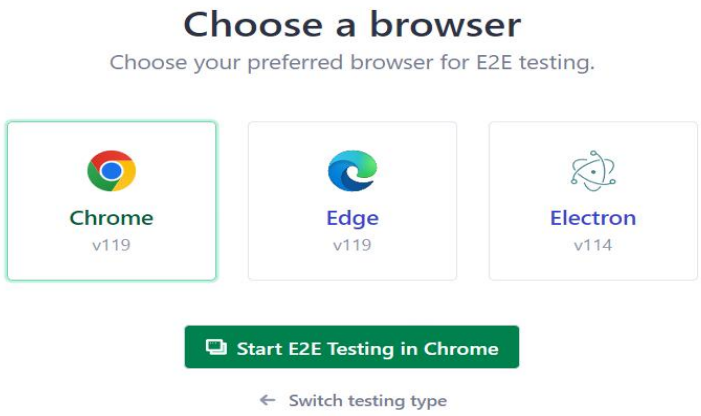
- ✧ This is the screen to select test type. We have two test type E2E aand Conmponet testing. Select E2E testing. To do end to end testing.



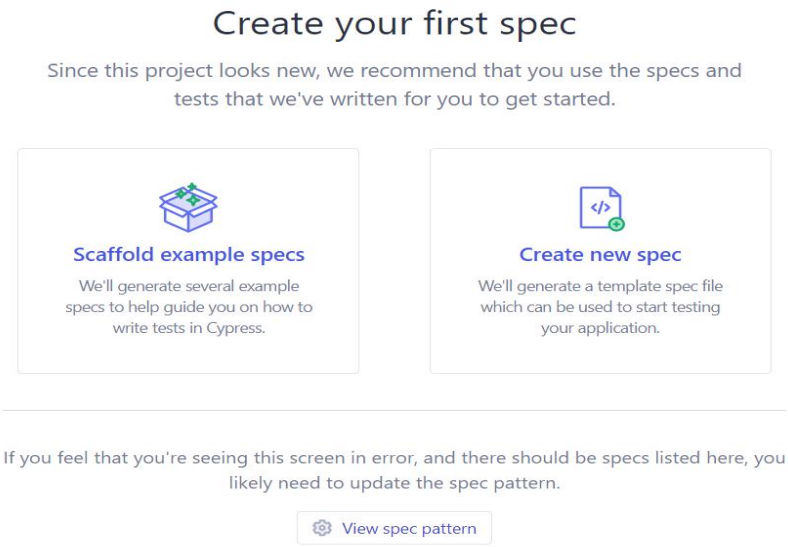

- ✧ Quick Configuration and let's skep as it is and press "Continue" button.



✧ Here, all browsers that you have on your computer will be displayed. Select your preferred browser to browse the Test Runner. In my case, I have Chrome and Edge, but Electron has the default browser. After selecting the browser then press the "Start E2E Testing in Chrome" button.

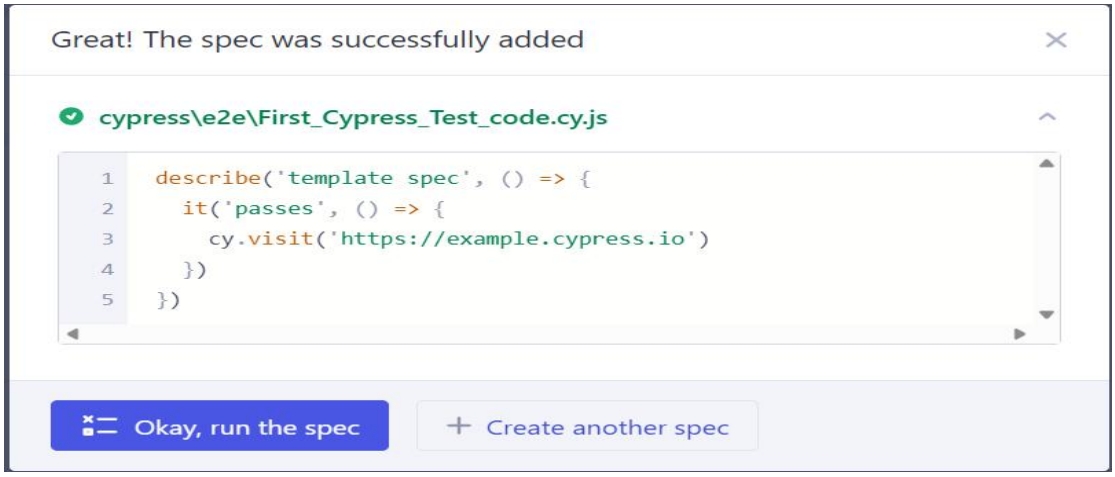



✧ Here, we have two options to create our first code. We can either use an example test code file or create a new test code file. Let's select **"Create new spec**" to create a new spec (test file).

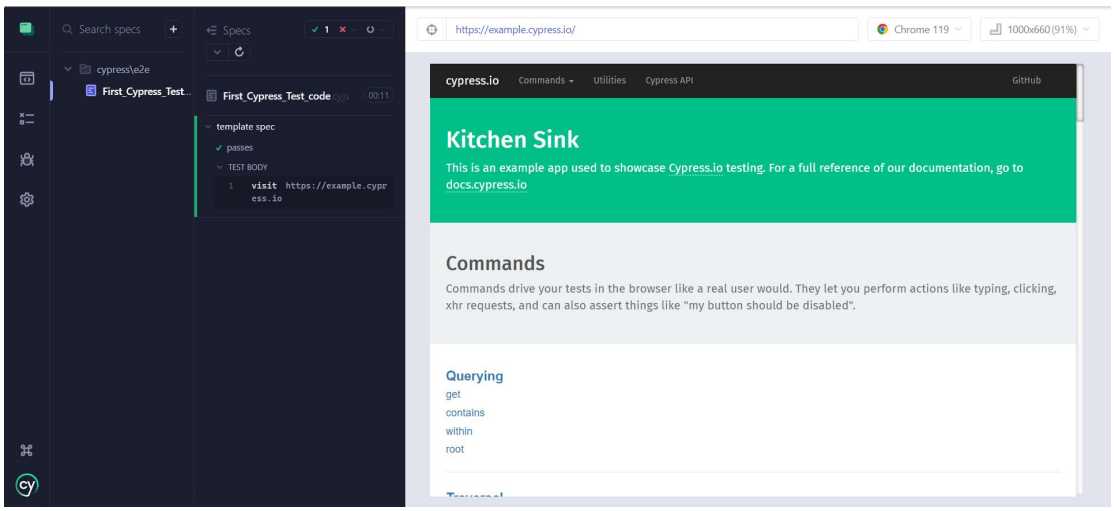



✧ Our test file path is "**cypress/e2e/spec.cy.js**". The test file is located under the "cypress" folder and we have an "e2e" folder within that. The test file name is "spec.cy.js". Let's modify the test file name to "**First_Cypress_Test_code.cy.js**".

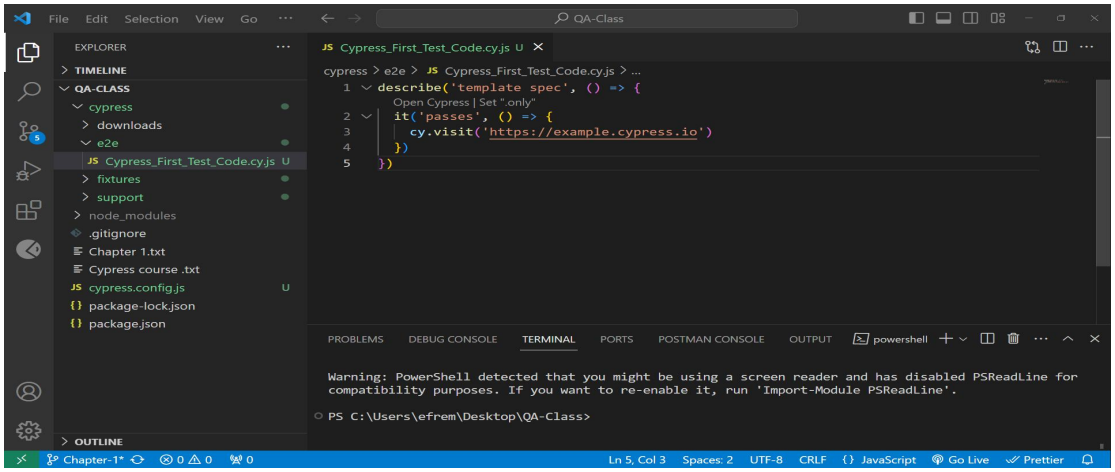
✧ This is our test file templet. Let's "**click Okay, run the spec"** button**.**




✧ Now our first test is opend with test runner. Let's go to viscode see what happen and modifay our test code.




✧ Here, we can see that a "cypress" folder has been created. Inside the "cypress" folder, there are several subfolders including "download," "e2e," "fixtures," and "support." All our test code will be placed under the "e2e" folder, which is why we are seeing our first test file located under the "e2e" folder.



Let's discuss and modify the code.

```
describe('template spec', () => {
  it('passes', () => {
    cy.visit('https://example.cypress.io')
  })
})
```

1. **describe**:-
   - ✧ This keyword is used to create a test suite or a group of related test cases.
   - ✧ It takes two arguments: a descriptive string that explains what is being tested and a callback function that contains the test cases that belong to this suite.
   - ✧ The purpose of using `describe` is to organize the tests and provide a clear structure to the testing code.
2. **it**:-
   - ✧ This keyword is used to define a single test case.
   - ✧ It also takes two arguments: the name of test case and a callback function that contains the actual test logic (test steps).
   - ✧ The purpose of using `it` is to clearly indicate what specific functionality is being tested.

All callback function writen in arrow function on the above exaple, but we can use namelesss function too.

**Example with nameless function.**

```
describe('template spec', function(){
  it('passes', function(){
    cy.visit('https://example.cypress.io')
  })
})
```

Note:-

- ✧ In a test file, we can have multiple "**describe"** blocks, each containing multiple "**it**" blocks. The "**describe**" blocks help in organizing the tests into logical groups, while the "**it**" blocks define the individual test cases within each group.

- ✧ When running the test file, we have the option to selectively skip certain "**describe**" blocks or "**it**" blocks by using the `.skip` method. This allows us to exclude specific tests from running, which can be useful when debugging or temporarily disabling certain tests.

- ✧ To skip a `describe` block, we can add `.skip` after the `describe` keyword, like this:

**Example using skiping describe block.**

```
describe.skip('template spec', function(){
  it('passes', function(){
    cy.visit('https://example.cypress.io')
  })
})
```

To skip an "**it**" block, we can add ".**skip**" after the "**it**" keyword, like this:

**Example using skiping it block.**

```
describe('template spec', function(){
  it.skip('passes', function(){
    cy.visit('https://example.cypress.io')
  })
})
```

- ✧ In Cypress, the `.only` keyword is a powerful tool that allows you to focus exclusively on running specific `describe` or `it` blocks, while ignoring all other tests in the file.
- ✧ When you add `.only` after a `describe` block, only that specific `describe` block and its nested `it` blocks will be executed, excluding all other `describe` and `it` blocks.
- ✧ Similarly, when you use `.only` after an `it` block, only that specific `it` block will be executed, regardless of its parent `describe` block.

**Example using only in describe block.**

```
describe.only('template spec', function(){
  it('passes', function(){
    cy.visit('https://example.cypress.io')
  })
})
```

**Example using only in it block.**

```
describe('template spec', function(){
  it.only('passes', function(){
    cy.visit('https://example.cypress.io')
  })
})
```

**Home work**

1.Create floder with name **“Qa_Cypress”** on desktope and open it with viscode install cypress.

2.Create cypress test file named **“my_first_cypress_code_home_work”**

3. Please wite 2 “describe” block on test file under each “describe” block create 5 “it” blocks.

4. Using the skip keyword please skip the first “describe” block and using only keyword please execute the first two “it” block under the seconde “describe” block.