

# Front End Testing: Cypress.io

Install Cypress by following the instructions in the installation guide<sup>1</sup>. Once this is done, a good place to start working would be the Cypress IO's documentation where they have a guide to writing your first test<sup>2</sup>. Some of the things explained in the guide are summarised in the next section but try to go through the entire Getting Started guide for Cypress ([documentation](#)). You can open Cypress once it's been installed by using the command:

```
./node_modules/.bin/cypress open
```

This tutorial will continue using the Movie Manager MERN app used in previous tutorials ([Github](#), [deployed Heroku link](#)). The Cypress.io [Youtube channel](#) also has a couple of pretty helpful video tutorials.

## A Simple Passing and Failing Test

If you're done installing Cypress, you should see a new folder in your application called "cypress". The contents of that folder will initially look like this (below). Create a new file called `basic_tests.js`.

```
├── fixtures
│   └── example.json
├── integration
│   ├── examples
│   └── basic_tests.js (create this file)
├── plugins
│   └── index.js
└── support
    ├── commands.js
    └── index.js
```

Type out the below code in the new file:

```
/// Basic tests - from the CypressIO documentation

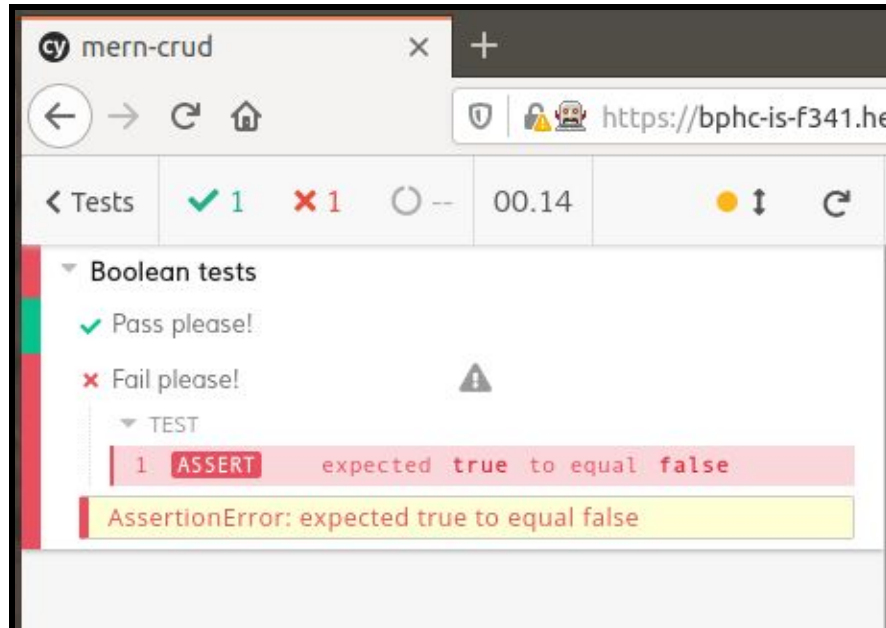
// a passing and a failing test
describe('Boolean tests', function() {
  it('Pass please!', function() {
    expect(true).to.equal(true)
  });
  it('Fail please!', function() {
    expect(true).to.equal(false)
  });
});
```

---

<sup>1</sup> [Installing Cypress](#)

<sup>2</sup> [Cypress: Writing Your First Test](#)

The first test should pass and the second test should fail. After opening the Cypress window (by running `./node_modules/.bin/cypress open`) click on the `sample_spec.js` item in the list to run the test. A Firefox window will open, that'll look something like this:

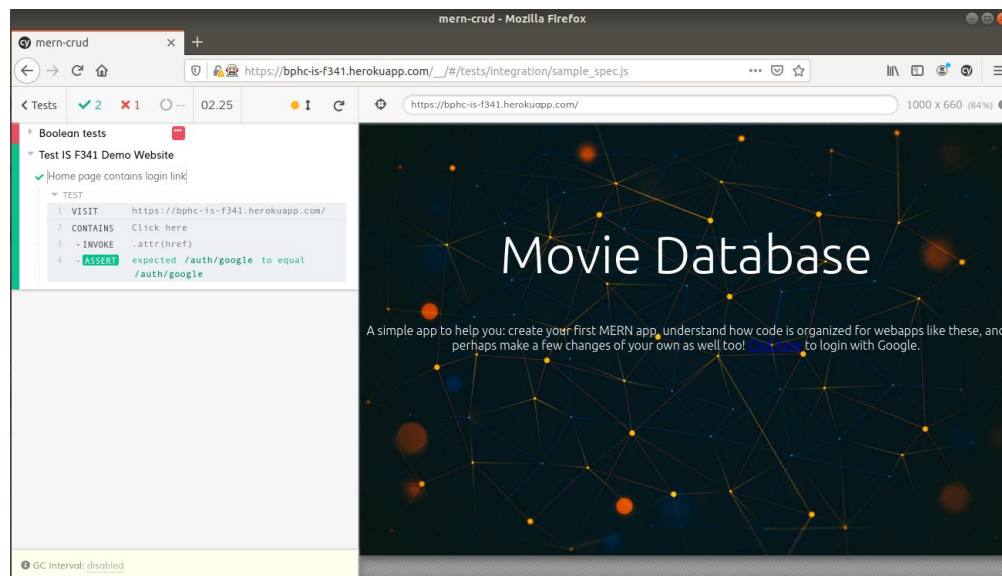


Now, we will try to write a test that will visit the course website, and check if the home page has a link to the login page.

```
describe('Test IS F341 Demo Website', function() {
  it('Home page contains login link', function() {
    cy.visit('https://bphc-is-f341.herokuapp.com/');
    cy.contains('Click here')
      .invoke('attr', 'href')
      .then(href => {
        expect(href).to.equal("/auth/google");
      });
  });
});
```

The `cy.contains` statement finds the web element that contains the text “Click here”, gets the “href” or destination URL of that link and checks if it is equal to “/auth/google”. /auth/google is the API endpoint of our web app that handles Google sign-in.

Running this test gives you the following output. We see that the test dashboard contains the list of all steps that we have used in the test.



The aim of this tutorial will be to test the **front-end** only. We'll test if the **login**, **all movies view**, and **add movie view** are being displayed as intended.

## The Frontend Tests

All the code and fixtures for the front end tests can be found in `cypress/integration/front_end_tests.js` and `cypress/fixtures/movies_list.json` respectively. Let's start by taking a look at the first set of tests inside the `front_end_tests.js` file:

```
1 describe('App Home Page', function() {
2   it('successfully loads', function() {
3     cy.visit('/') // change URL to match your dev URL
4   });
5   it('Home page contains login link', function() {
6     cy.contains('Click here')
7       .invoke('attr', 'href')
8       .then(href => {
9         expect(href).to.equal("/auth/google");
10      });
11   });
12 });
```

The first test (code snippet on the previous page) tries to load the home page and checks if there is a link to the login page. Once a user is logged in, he is taken to the all movies page which shows the list of movies in the database.

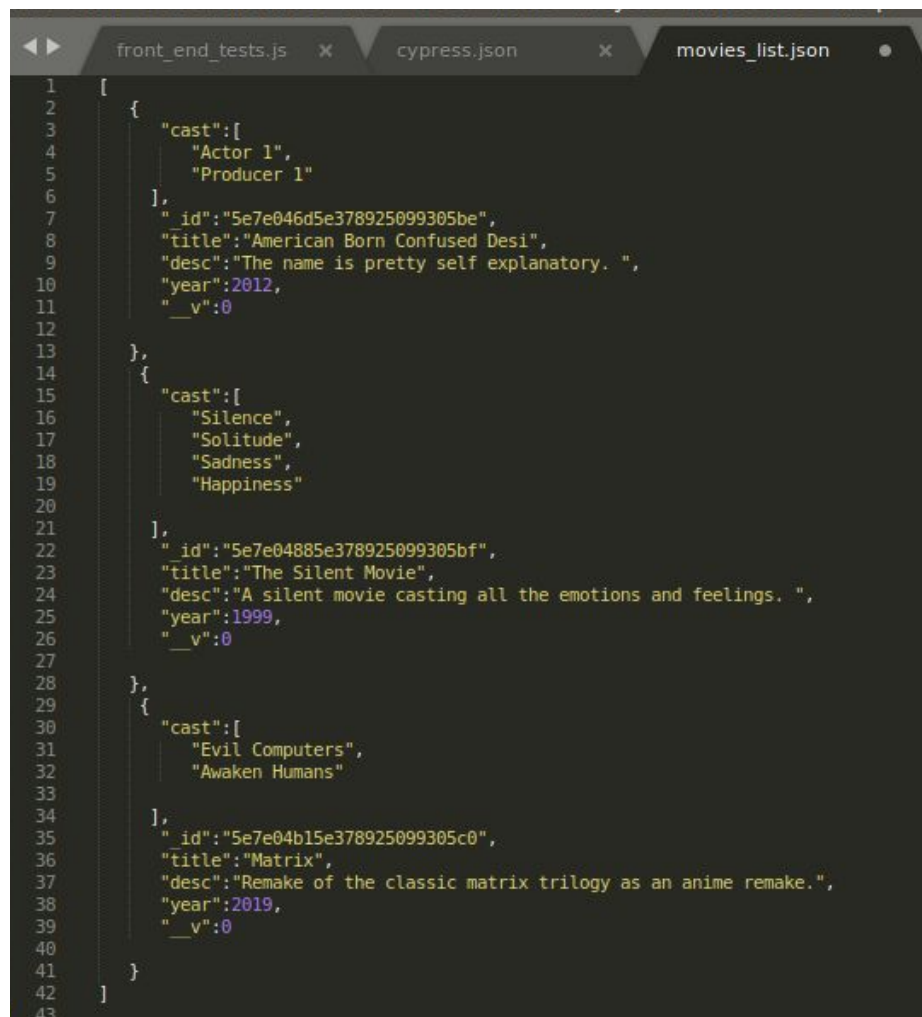
To set up these tests for your own app, you need to configure a couple of things in the beginning. Firstly, in the code snippet above, we see on line 3, the statement `cy.visit('/')`. We need to configure the base URL (the address at which our web app is running) in the `cypress.json` file - this file can be found in the root directory of your project. Add this the file:

```
{
  "baseUrl": "http://localhost:5050"
}
```

Next, it might be a good idea to add `cypress/screenshots` to your `.gitignore`. One thing you'll have to keep in mind is that you need to build and run your development server BEFORE trying to run the tests.

## Stubbing

In our application, the front-end gets a list of movies from the backend by sending a request to the `/api/movies` endpoint. Since our aim is to independently test the front-end, we need to write a **stub** that sends data without involving the server. A stub with three movies in it was created for the purposes of testing (`cypress/fixtures/movies_list.json`):



```
1  [
2    {
3      "cast": [
4        "Actor 1",
5        "Producer 1"
6      ],
7      "id": "5e7e046d5e378925099305be",
8      "title": "American Born Confused Desi",
9      "desc": "The name is pretty self explanatory. ",
10     "year": 2012,
11     "__v": 0
12   },
13   {
14     "cast": [
15       "Silence",
16       "Solitude",
17       "Sadness",
18       "Happiness"
19     ],
20     "id": "5e7e04885e378925099305bf",
21     "title": "The Silent Movie",
22     "desc": "A silent movie casting all the emotions and feelings. ",
23     "year": 1999,
24     "__v": 0
25   },
26   {
27     "cast": [
28       "Evil Computers",
29       "Awaken Humans"
30     ],
31     "id": "5e7e04b15e378925099305c0",
32     "title": "Matrix",
33     "desc": "Remake of the classic matrix trilogy as an anime remake.",
34     "year": 2019,
35     "__v": 0
36   }
37 ]
```

Now, we are in a position to write code to test the all-movies-view:

```
89 describe('All movies page', function() {
90   beforeEach(function(){
91     cy.server();
92     cy.route({
93       method: "GET",
94       url: "/api/movies",
95       response: "fixture:movies_list.json"
96     });
97   });
98   it('successfully loads', function() {
99     cy.visit('/movies'); // change URL to match your dev URL
100   });
101
102   it('movie titles are visible', function() {
103     cy.contains('American Born Confused Desi').should('be.visible');
104     cy.contains('The Silent Movie').should('be.visible');
105     cy.contains('Matrix').should('be.visible');
106   });
107
108   it('add movie button visible', function() {
109     cy.contains('Add a Movie').should('be.visible');
110     cy.get('a[id="addm"]').should('have.attr', 'href', '/movies/add');
111   });
112
113   it('delete buttons visible', function(){
114     // we should have three delete buttons, one for each movie
115     cy.get('button:contains(Delete)').should('have.length', 3).should('be.visible');
116   });
117 });
```

---

There's a lot happening in this code, so let's try to break it down bit by bit:

1. The `beforeEach` function on Line 90 is executed before every test. In that function, we are setting up a Cypress server, and telling the test environment to return the static list inside `movies_list.json` whenever the app makes a request to `/api/movies`. So, instead of sending a request to the server, this data (called a stub) is directly provided to the front-end.
2. Each `it()` block is one test.
3. We are testing if the webpage loads, whether all the movies in the stub are being displayed in the webpage and if the add and delete buttons are visible.

Similarly, tests have been written for the "add movie" view. Check them out on the [Git repo](#). A video recording of this test being performed can be found in [this Youtube video](#) I uploaded. Once you're done writing tests in Cypress for your application, you should be able to make a similar video too.

## Reference and Notes

1. [Cypress Network Tests: Stubbing vs End-to-End Testing](#): This link explains how stubbing works, and the tradeoffs between doing end-to-end tests and stubbing.
2. To convert this front end test to an end-to-end test, we will have to remove the stubs and allow the front-end to send requests to the backend. Since some of your functionality might require login you will have to write code in your scripts to handle login and sessions. The next tutorial will probably contain a very brief description of how to do end-to-end testing.

## Queries

For any queries about this tutorial, you may contact the TA for the course, Rohit Dwivedula (2017A7PS0029H) through email at [f20170029@hyderabad.bits-pilani.ac.in](mailto:f20170029@hyderabad.bits-pilani.ac.in).