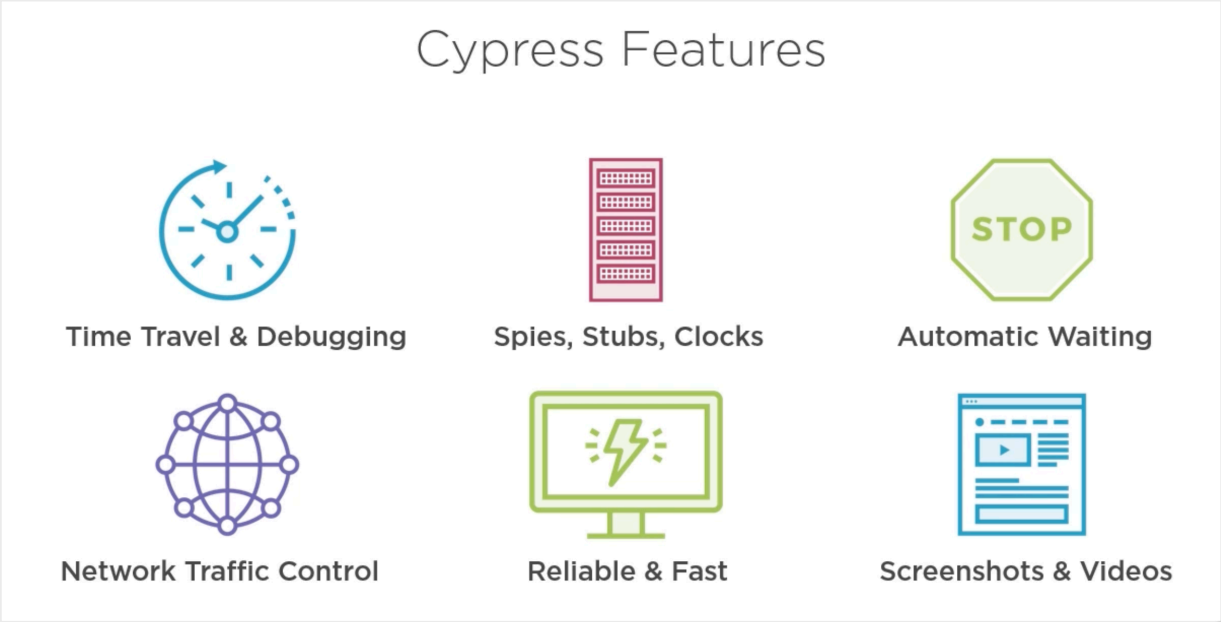
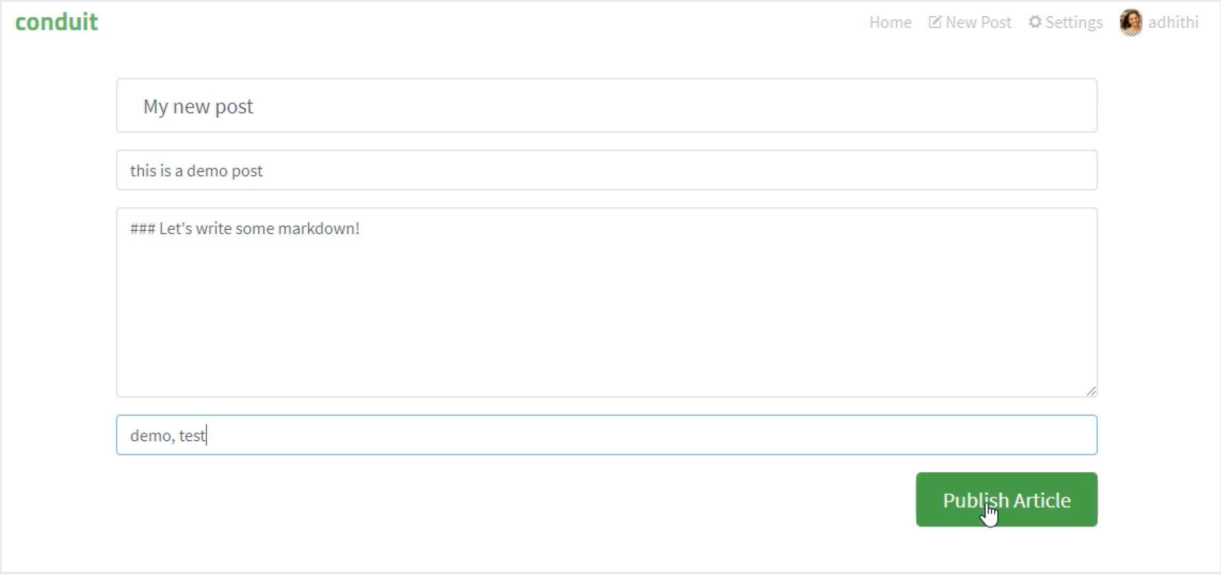
Cypress



Sample APP



Cypress Network Testing Strategy

Don't Stub Responses

Stub Responses

Stub Responses

Can control every aspect of response (body, status, headers, delay)

Extremely fast, responses in less than 20 ms

Perfect for JSON responses

No test coverage on some server endpoints

```
cy.server()
cy.route('**/users').as('getUsers')
cy.visit('/users')
cy.wait('@getUsers')
```

Without Stubbing

No response is passed to route

Cypress will pass the request without stubbing to the server

Wait for request to resolve later

```
cy.server()
cy.route('https://localhost:7777/surveys/customer?email=john@doe.com', [
  {
    id: 1,
    name: 'john'
  }
])
```

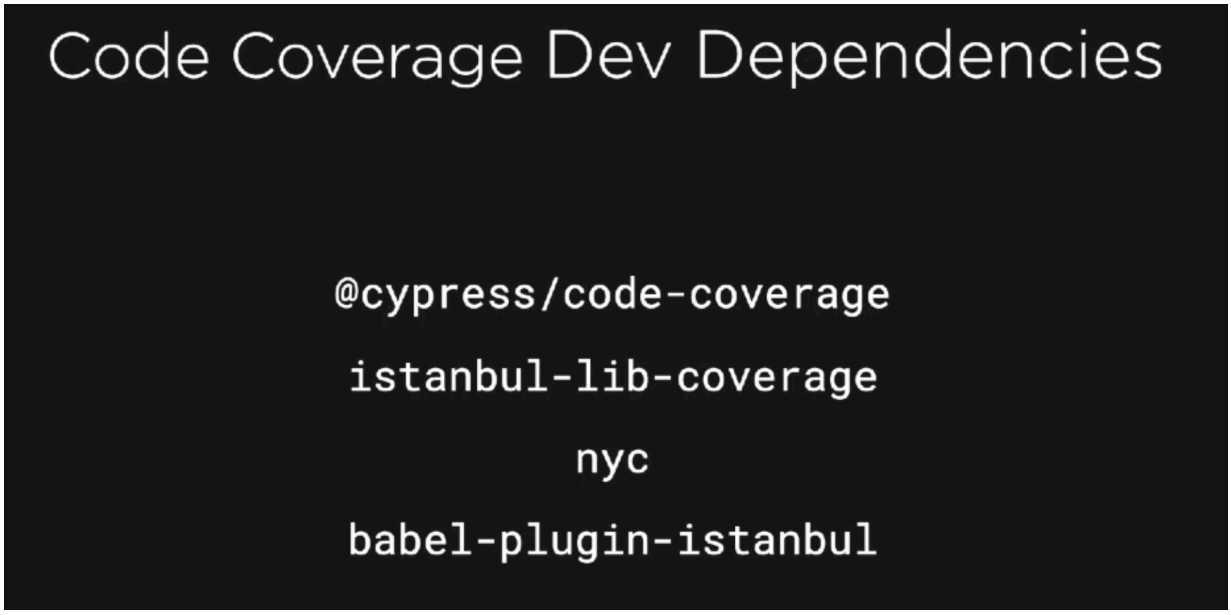
With Stubbing

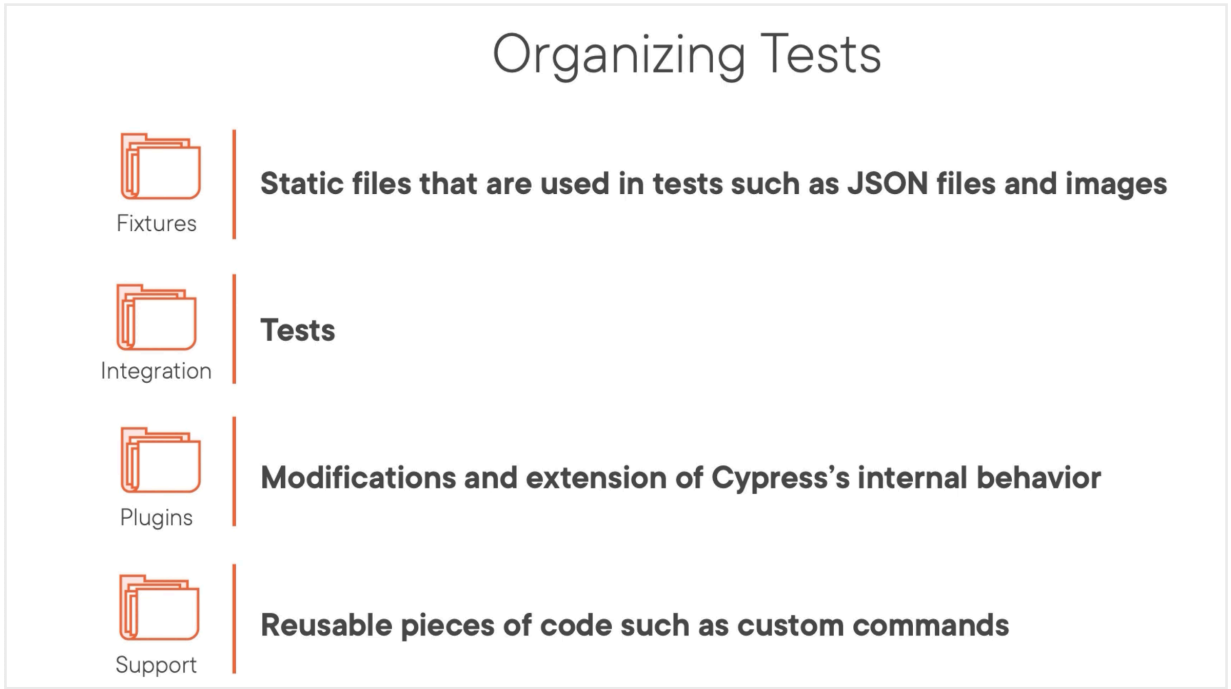
Pass a response to *cy.route()*

Cypress will stub the response in the request

. The first command that you would need is the cy.server command. This is used to start a server to begin routing responses to cy.route and to change the behavior of your network requests. You won't need the server command if you directly decide to hit the server and get an actual response back from the server. The next command that we need is a cy.route command. This is used to manage the behavior of the network request, and you can also pass the response within the cy.route command. The third command is a cy.request command. This is the command that makes an HTTP request directly to the server, and this can be used if you don't want to stub your responses. Alright. Let's take a look at how we use these. In this example, you can notice that we started with the cy.server command. The next command we've used is the cy.route command where we pass the /users URL, so if you have large routes, you can always define aliases that you can use later. That way you don't repeat your entire URL. And then we also do cy.visit, which goes into the /users URL, and we've added a wait for the request to resolve later. If you notice here, we did not define any response inside the cy.route command. What this means is without a response being passed to the route, Cypress will pass the request without stubbing to the server. So this essentially acts as a call that is made to the server and waits for a response back from the server. In this situation, we have not stubbed any response at all. In this example, we're going to use cy.route with stubbing. In this case, we have initially called the cy.server, and then we've used the cy.route command to pass a URL, and if you notice within the square brackets, we've passed a response JSON right there. If the response JSON has an ID and the name, and Cypress will now stub the response in the request so it doesn't wait for the server to give back the response instead, it's going to stub this response as if the server returned this response. You saw now how we can use the same command cy.route for both making calls to the server and waiting for the server and for also stubbing a response. The cy.route can also make calls where you send an empty response. This could be

useful when you're trying to mimic a delete scenario. In this example, the cy.route command is going to match all DELETE requests to the URL /users, and it's going to stub it with an empty JSON indicating that it is an empty response. This is going to simulate basically deleting a bunch of users and returning a response, which is empty, so you could do things like this with the cy.route command as well. In the next clip, we'll write some Cypress tests for our Conduit app that that'll demonstrate making network requests using Cypress.





Example of a Custom Command

```
Cypress.Commands.add('clickLink', (label) => {
    cy.get('a').contains(label).click()
});


cy.clickLink('Buy Now');
```

Mixing Synchronous and Asynchronous Code

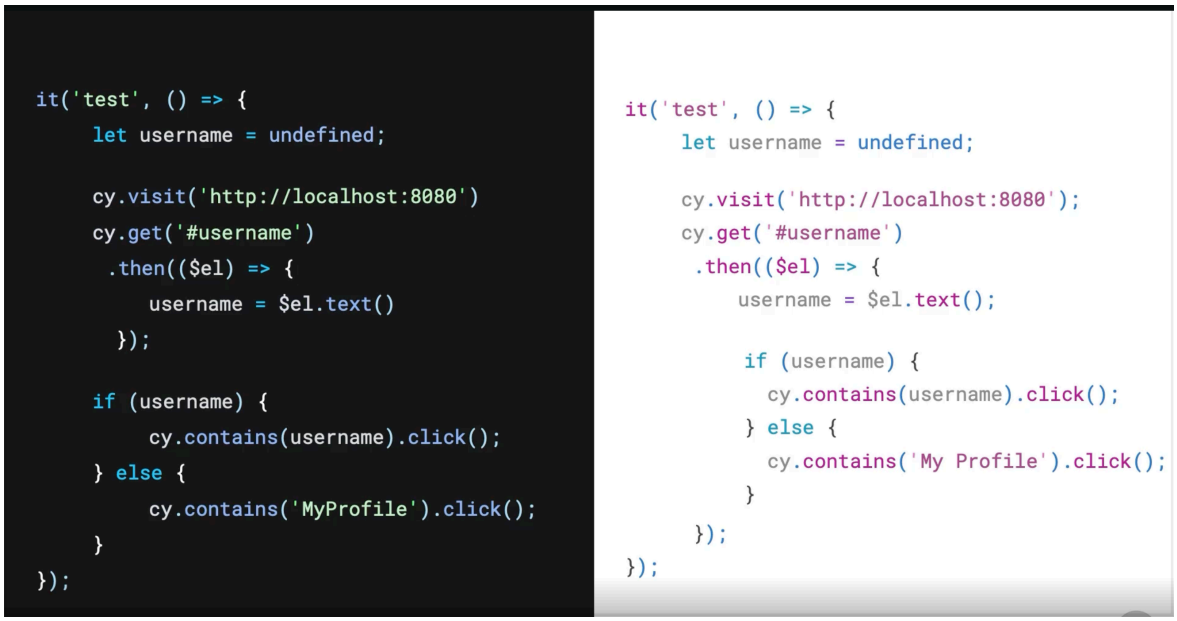
Cypress commands are asynchronous.

```
cy.get('button.primary').click();
```

Cypress commands do not return their subjects. They yield them.

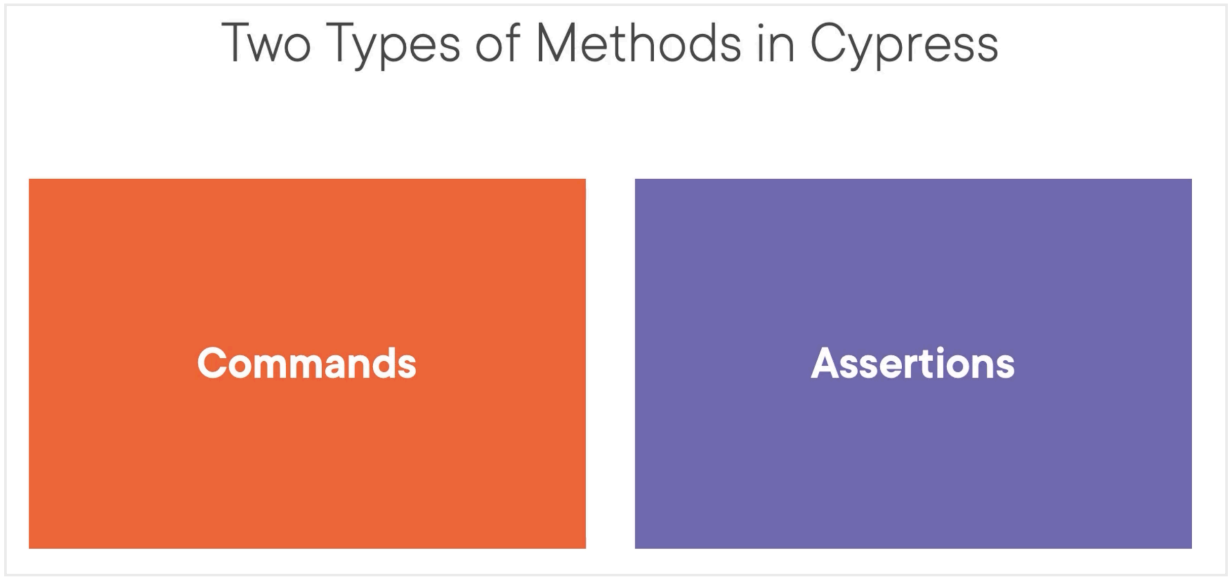
“Left side code” conditions always goes to False because “”username” is undefined and cypress will not wait for cy.get(“#username)

“Right side code” conditions always goes to true because cypress will wait until “” username” gets some value
So right side code correct



```
it('test', () => {
     let username = undefined;

     cy.visit('http://localhost:8080')
     cy.get('#username')
      .then(($el) => {
          username = $el.text()
       });

     if (username) {
          cy.contains(username).click();
     } else {
          cy.contains('MyProfile').click();
     }
});
```

```
it('test', () => {
     let username = undefined;

     cy.visit('http://localhost:8080');
     cy.get('#username')
      .then(($el) => {
          username = $el.text();

          if (username) {
            cy.contains(username).click();
          } else {
            cy.contains('My Profile').click();
          }
       });
});
```



```
cy.get('.main-list li') // command
  .should('have.length', 3) // assertion
```

Cypress only retries commands that query the DOM.

.get() .find() .contains()

Commands that are not retried are the ones that could potentially change the state of the application.

Changing the Timeout

The default timeout can be changed on a command level or globally.

cypress.json

```
{
    "defaultCommandTimeout": 0
}
```

homepage.spec.js

```
cy.get('button.primary')
   .click({ timeout: 0 });
```

```
cy.get('.main-list li') // yields only one <li>
  .find('label') // retries multiple times on a single <li>
  .should('contain', 'My Item')
```

Retry-ability is a core feature of Cypress, where Cypress automatically retries

certain methods before delivering the outcome. It works behind the scenes and it's usually not noticeable. Let's first make a difference between two types of methods that you call in your Cypress tests. Those are commands and assertions. Let's look at this command and assertion pair. If the assertion that follows the cy.get command passes, then the command finishes successfully. And if the assertion that follows the cy.get command fails, then the cy.get command will query the DOM again. Then Cypress will try the assertion against the elements yielded from the cy.get. If the assertion fails again, cy.get will query the DOM until the cy.get timeout is reached. The retry-ability allows the tests to complete each command without hardcoding waits

```
    cy.request('http://localhost:8080');

    cy.visit('http://localhost:8080/list-events');
    cy.request('events/1.json'); // url is http://localhost:8080/events/1.json

    cy.request('DELETE', 'http://localhost:8080/events/1');

    cy.request('POST', 'http://localhost:8888/events', { name: 'New Event' });

    cy.request({
        method: 'POST',
        url: 'http://localhost:8888/events',
        body: { name: 'New Event' },
        headers: { 'Content-Type': 'application/json' }
    });
```

```
cy.request('POST', 'http://localhost:8080/events', { name: 'New Event' })
 .then((response) => {
       expect(response.status).to.eq(201);
       expect(response.body).to.have.property('name', 'New Event');
  }
);
```

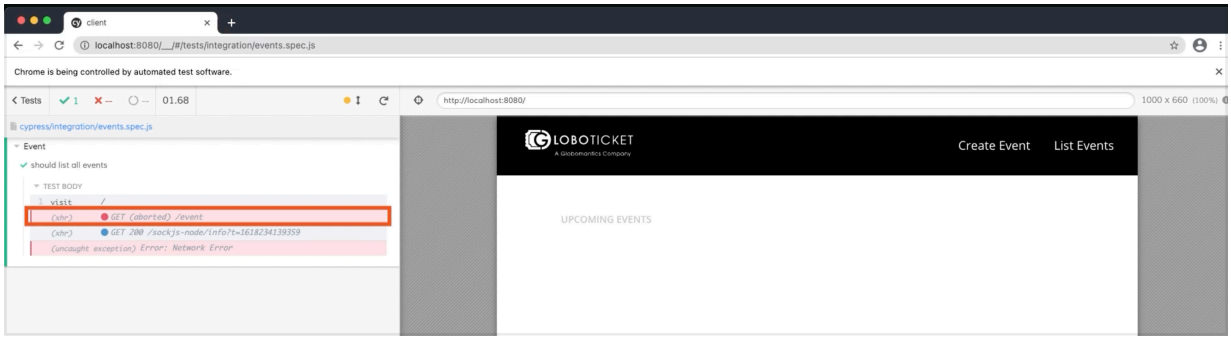
Intercepting HTTP Requests

You may be wondering how is that useful. That's useful in many situations. For example, suppose you are doing end-to-end testing of your application and you

have the feature ready on the front and side, but the server-side implementation is not ready yet. In that case, you can still implement your test scenario by marking the responses from the server, or if you're only interested in testing the presentation layer, you can intercept all the requests your front end application makes and provide responses depending on your test scenarios. Let's see how to intercept the request. The first thing we need to do is to call the intercept command. Once the request is made from the client application, it will be intercepted

```
cy.intercept(url, routeMatcher, routeHandler).as('alias');

cy.wait('@alias');
```

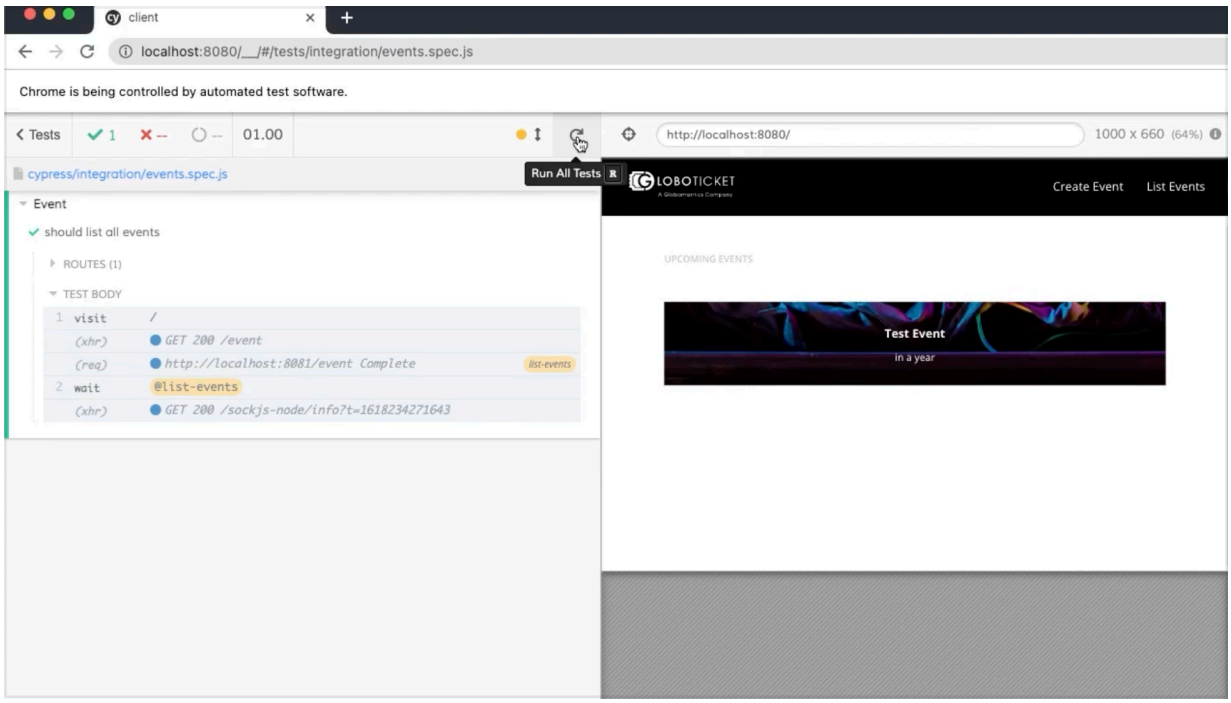
```
describe('Event', () => {
  it('should list all events', () => {
    cy.visit('/');
  })
})
```

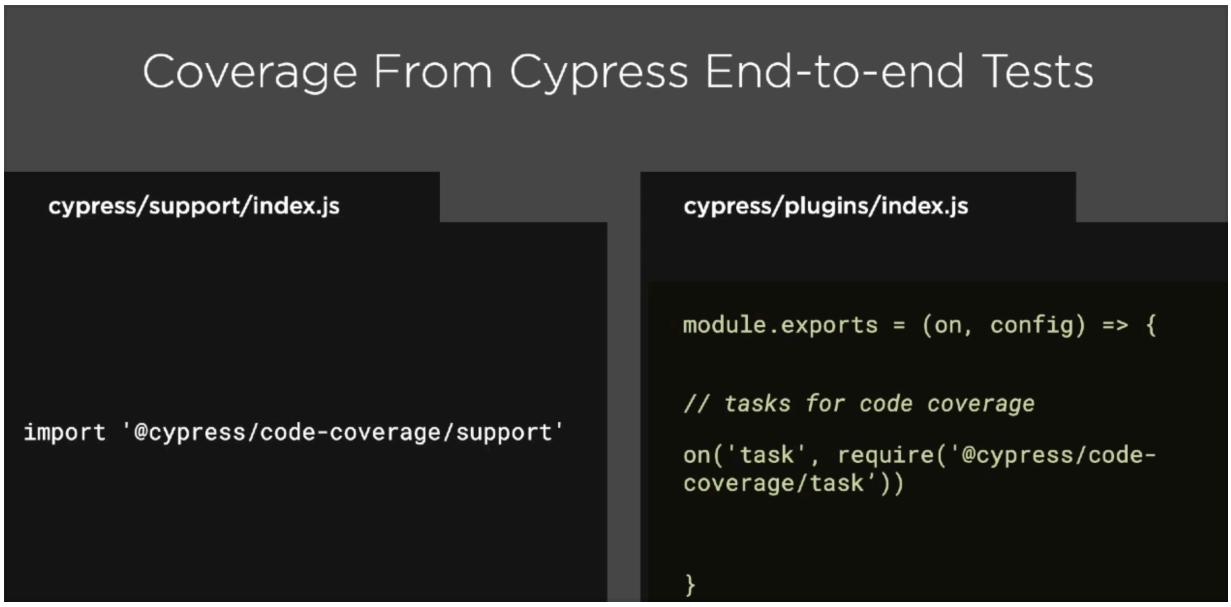



```
/// <reference types="cypress" />

describe('Event', () => {
  beforeEach(() => {
    cy.intercept('GET', 'http://localhost:8081/event', { fixture: 'list-events.json' }).as
    ('list-events');
  });

  it('should list all events', () => {
    cy.visit('/');
    cy.wait('@list-events', { timeout: 10000 });
  })
})
```

t > cypress > fixtures > {..} list-events.json > ...

```
[
  {
    "_id": "1",
    "name": "Test Event",
    "venue": "Test Venue",
    "date": "2022-05-05",
    "time": "12:00AM",
    "ticketQuantity": 100,
    "isCancelled": false,
    "image": "./assets/event-1.jpg"
  }
]
```





#pluralsight