Stefan Exner | stex.codes

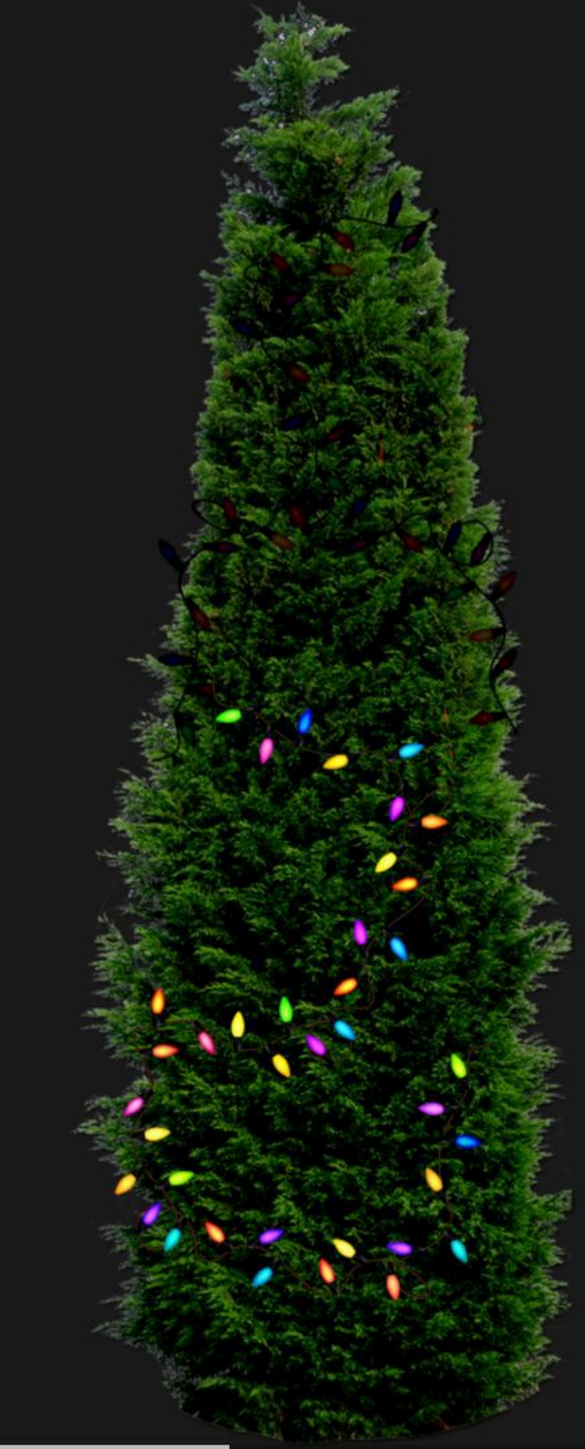
Introduction to Cypress

(You'll need this to understand the Rails integration later)

What is Cypress?

What is Cypress?

- Website agnostic end-to-end testing framework



What is Cypress?

- Website agnostic end-to-end testing framework
- Runs completely in the browser / doesn't need Selenium

What is Cypress?

- Website agnostic end-to-end testing framework
- Runs completely in the browser / doesn't need Selenium
- Tests are written in pure Javascript (on top of Mocha, Chai and others)

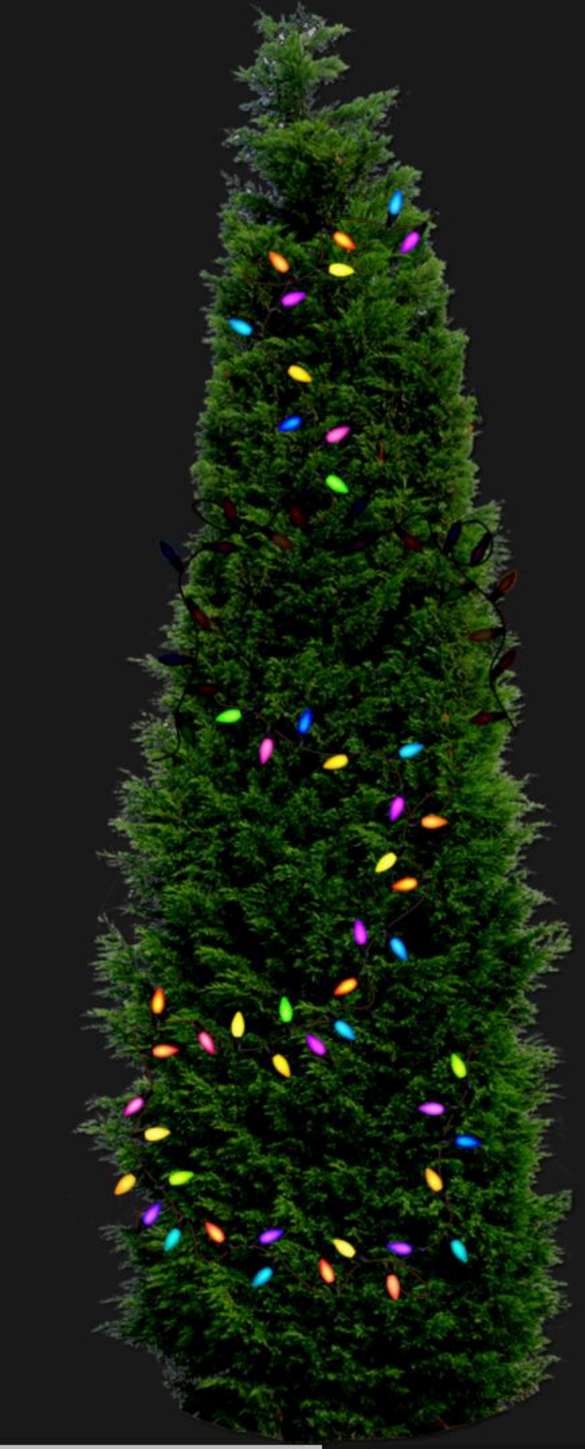
What is Cypress?

- Website agnostic end-to-end testing framework
- Runs completely in the browser / doesn't need Selenium
- Tests are written in pure Javascript (on top of Mocha, Chai and others)
- Supports Electron, Firefox and Chromium based browsers like the new Edge

Other Features

Other Features

- Live-Reload on file system changes



Other Features

- Live-Reload on file system changes
- Step-By-Step DOM snapshots

Other Features

- Live-Reload on file system changes
- Step-By-Step DOM snapshots
- Request and browser API stubbing

Other Features

- Live-Reload on file system changes
- Step-By-Step DOM snapshots
- Request and browser API stubbing
- Full Dev-Tools and `debugger` support

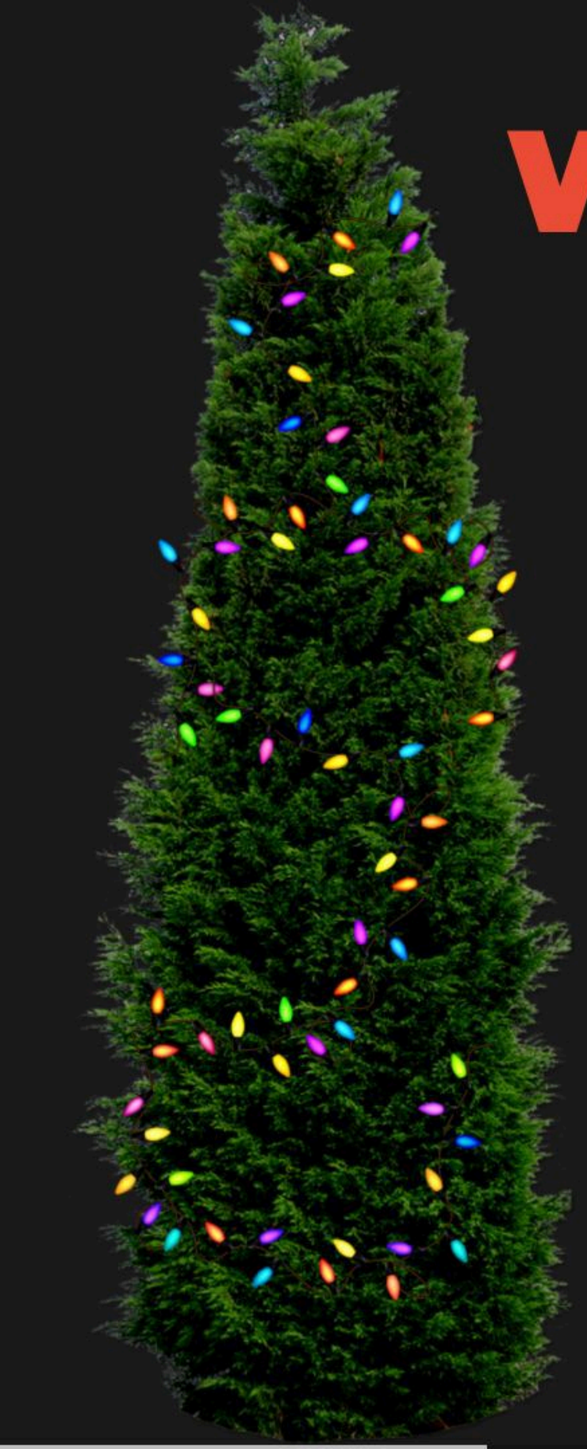
Other Features

- Live-Reload on file system changes
- Step-By-Step DOM snapshots
- Request and browser API stubbing
- Full Dev-Tools and `debugger` support
- Video recording in CI

Demonstration

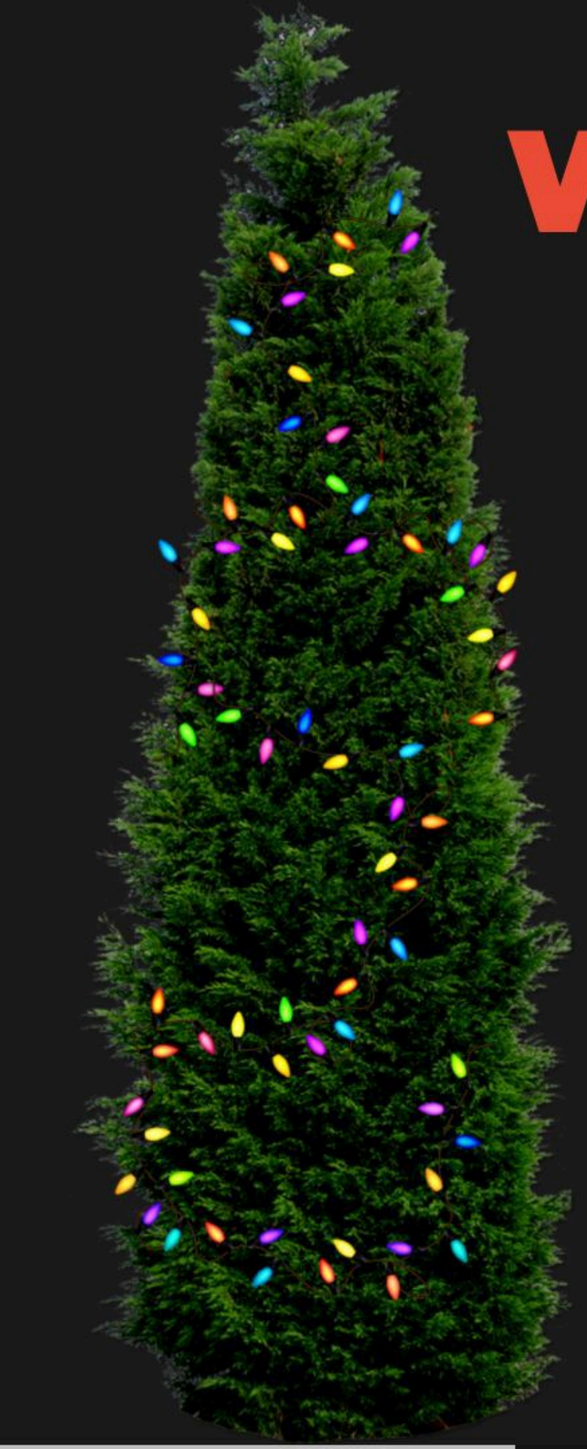
Writing a Basic Test

```
describe('Login', function() {
    beforeEach(function() {
        cy.fixture("basic_user.json").as("user")
        cy.visit("/")
    });

    it('logs me in', function() {
        cy.get('nav').contains('a', 'Login').click();

        cy.get('input[name=email]').type(this.user.email);
        cy.get('input[name=password]').type('12345678');
        cy.contains('input', 'Login').click();

        cy.get('nav').should('contain', this.user.email);
    })
})
```



Writing a Basic Test

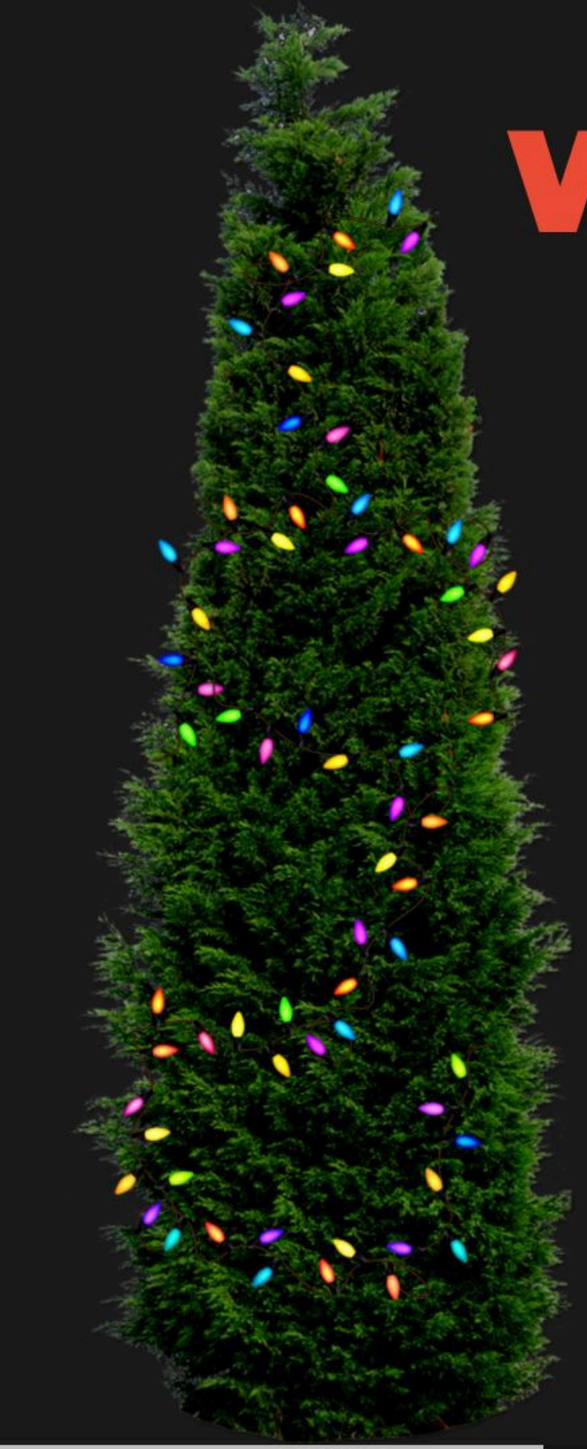
```
describe('Login', function() {
    beforeEach(function() {
        cy.fixture("basic_user.json").as("user")
        cy.visit("/")
    });

    it('logs me in', function() {
        cy.get('nav').contains('a', 'Login').click();

        cy.get('input[name=email]').type(this.user.email);
        cy.get('input[name=password]').type('12345678');
        cy.contains('input', 'Login').click();

        cy.get('nav').should('contain', this.user.email);
    })
})
```



Writing a Basic Test

```
describe('Login', function() {
    beforeEach(function() {
        cy.fixture("basic_user.json").as("user")
        cy.visit("/")
    });

    it('logs me in', function() {
        cy.get('nav').contains('a', 'Login').click();

        cy.get('input[name=email]').type(this.user.email);
        cy.get('input[name=password]').type('12345678');
        cy.contains('input', 'Login').click();

        cy.get('nav').should('contain', this.user.email);
    })
})
```

`before()`, `beforeEach()`, `after()` and `afterEach()` are available



Writing a Basic Test

```
describe('Login', function() {
    beforeEach(function() {
        cy.fixture("basic_user.json").as("user")
        cy.visit("/")
    });

    it('logs me in', function() {
        cy.get('nav').contains('a', 'Login').click();

        cy.get('input[name=email]').type(this.user.email);
        cy.get('input[name=password]').type('12345678');
        cy.contains('input', 'Login').click();

        cy.get('nav').should('contain', this.user.email);
    })
})
```

Aliased (`.as()`) values are available as `this.ALIAS`

Asynchronous Execution

```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

Asynchronous Execution

```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

Asynchronous Execution

```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

`cy` commands immediately return `Chainer` objects and are queued for later execution

Asynchronous Execution

```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

The **debugger** command is executed before either of the above are actually ran

Asynchronous Execution

```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

Asynchronous Execution

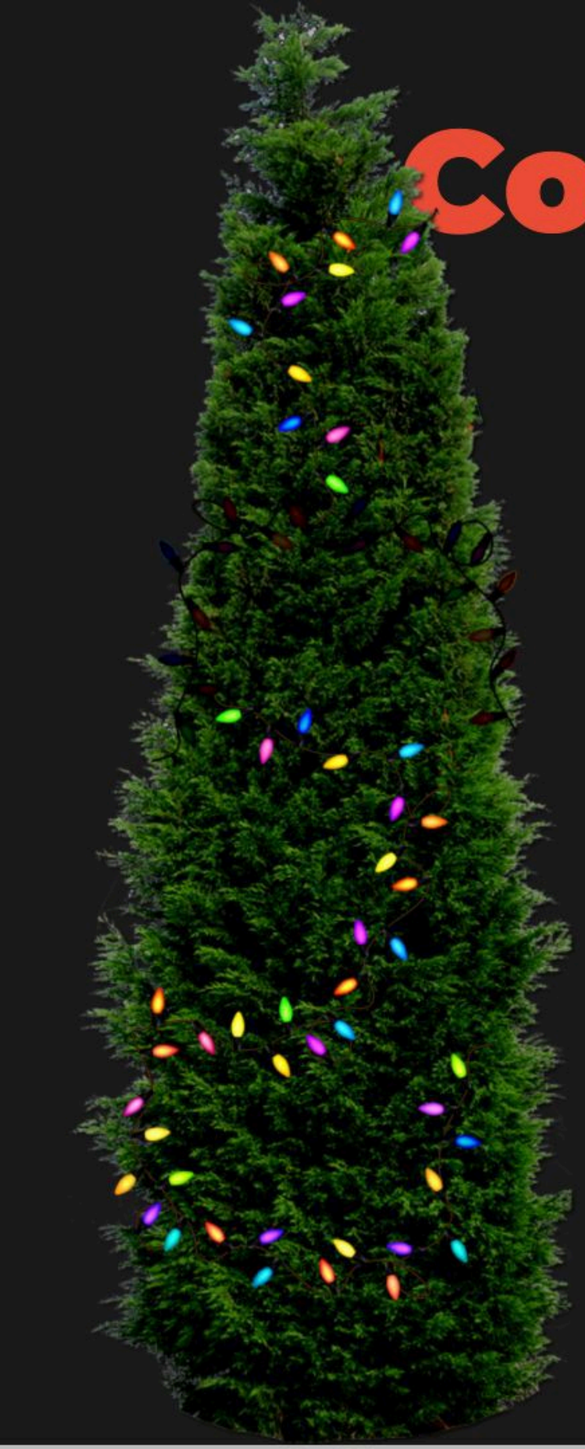
```
it('lets me debug like a fiend', () => {
  cy.visit('/my/page/path')
  cy.get('.selector-in-question')

  debugger
});

it('lets me debug after the command executes', () => {
  cy.visit('/my/page/path')

  cy.get('.selector-in-question')
    .then(($selectedElement) => {
      debugger
    })
})
```

`Chainer` objects provide their result by calling `then()` on them. Looks familiar?

Commands aren't Promises

Commands aren't Promises

- Fully synchonous execution of commands



Commands aren't Promises

- Fully synchonous execution of commands
- No `return` (and searching for a missing one for hours)

Commands aren't Promises

- Fully synchonous execution of commands
- No `return` (and searching for a missing one for hours)
- No `.catch` (as cypress has its own error handling)

Fixtures

```
// fixtures/basic_user.json
cy.fixture("basic_user.json").as("user");
cy.fixture("basic_user.json").then(user => {
  cy.log(user);
});

// fixtures/images/beaver.jpg
cy.fixture("images/beaver.jpg").then(beaver =>  {
  // base64 representation of the image
});
cy.fixture("images/beaver.jpg", "binary").then(beaver =>  {
  // binary representation of the image
});
```

Fixtures

```
// fixtures/basic_user.json
cy.fixture("basic_user.json").as("user");
cy.fixture("basic_user.json").then(user => {
  cy.log(user);
});

// fixtures/images/beaver.jpg
cy.fixture("images/beaver.jpg").then(beaver =>  {
  // base64 representation of the image
});
cy.fixture("images/beaver.jpg", "binary").then(beaver =>  {
  // binary representation of the image
});
```

Almost every file can be used as fixture, some are automatically validated

Fixtures

```
// fixtures/basic_user.json
cy.fixture("basic_user.json").as("user");
cy.fixture("basic_user.json").then(user => {
  cy.log(user);
});

// fixtures/images/beaver.jpg
cy.fixture("images/beaver.jpg").then(beaver =>  {
  // base64 representation of the image
});
cy.fixture("images/beaver.jpg", "binary").then(beaver =>  {
  // binary representation of the image
});
```

Some file types have custom default representation formats

Fixtures

```
// fixtures/basic_user.json
cy.fixture("basic_user.json").as("user");
cy.fixture("basic_user.json").then(user => {
  cy.log(user);
});

// fixtures/images/beaver.jpg
cy.fixture("images/beaver.jpg").then(beaver => {
  // base64 representation of the image
});
cy.fixture("images/beaver.jpg", "binary").then(beaver => {
  // binary representation of the image
});
```

The default format can be changed by providing a second argument

Custom cy Commands

```
Cypress.Commands.add('login', user => {
  cy.visit("/")
  cy.get('nav').contains('a', 'Login').click();
  cy.get('input[name=email]').type(user.email);
  cy.get('input[name=password]').type('12345678');
  cy.contains('input', 'Login').click();
});

Cypress.Commands.overwrite('visit', (originalFn, url) => {
  return originalFn(
    "https://custom_host.com?original_url=" + url
  );
});
```

Custom cy Commands

```
Cypress.Commands.add('login', user => {
  cy.visit("/")
  cy.get('nav').contains('a', 'Login').click();
  cy.get('input[name=email]').type(user.email);
  cy.get('input[name=password]').type('12345678');
  cy.contains('input', 'Login').click();
});

Cypress.Commands.overwrite('visit', (originalFn, url) => {
  return originalFn(
    "https://custom_host.com?original_url=" + url
  );
});
```

Own commands are available as `cy.COMMAND_NAME`

Custom cy Commands

```
Cypress.Commands.add('login', user => {
  cy.visit("/")
  cy.get('nav').contains('a', 'Login').click();
  cy.get('input[name=email]').type(user.email);
  cy.get('input[name=password]').type('12345678');
  cy.contains('input', 'Login').click();
});

Cypress.Commands.overwrite('visit', (originalFn, url) => {
  return originalFn(
    "https://custom_host.com?original_url=" + url
  );
});
```

Existing commands can be overwritten / overloaded

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

prevSubject lets the command receive the previous chain result

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

The type of `subject` is validated based on the value of `prevSubject`

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

`subject` is guaranteed to be an element, e.g. coming from `cy.get`

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

Works as `subject` is an element

Custom cy Commands

```
Cypress.Commands.add('insertImage', {
  prevSubject: 'element'
}, (subject, imageData) => {
  let myImage = new Image();
  myImage.src = imageData;
  subject.drawImage(myImage, 0, 0);
});

cy.fixture("images/beaver.jpg").then(beaver => {
  cy.get('#myCanvas').insertImage(beaver)
  cy.wrap([]).fillWithMarkdown(beaver);
})
```

Raises a validation error as **`subject`** is not an element

Custom cy Commands

```
Cypress.Commands.add('tree', () => "🎄");

beforeEach(function() {
  cy.tree().as('myTree');
})

it('logs a tree', function() {
  cy.log(this.myTree);
  cy.tree().then(tree => {
    cy.log(tree);
  })
})
```

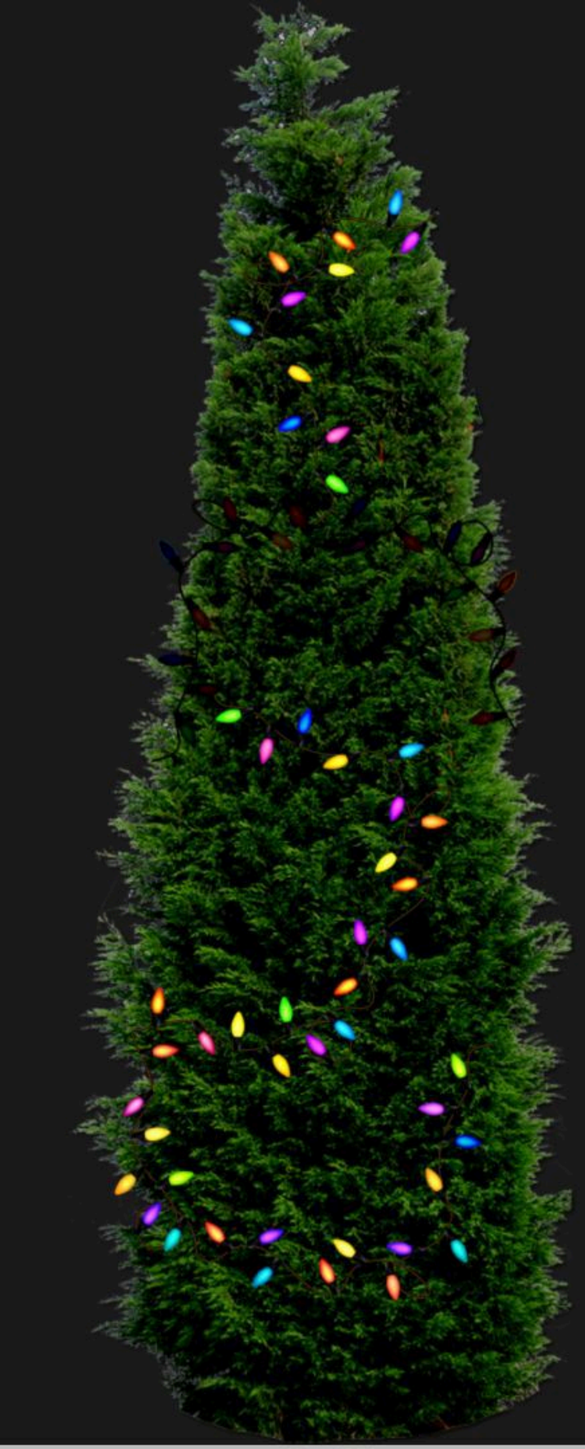
Own commands return `Chainer` instances just like the predefined ones.

Additional Goodies

Additional Goodies

- Integrated mechanisms for parallel execution*

*Requires using the Cypress Dashboard (and probably paying for it)



Additional Goodies

- Integrated mechanisms for parallel execution*
- Automatic retries per test

Additional Goodies

- Integrated mechanisms for parallel execution*
- Automatic retries per test
- Pretty good automatic timeouts for almost everything

Additional Goodies

- Integrated mechanisms for parallel execution*
- Automatic retries per test
- Pretty good automatic timeouts for almost everything
- Request-Stubbing

Additional Goodies

Request-Stubbing

```
cy.fixture('users').then((json) => {
  cy.intercept('GET', '/users/**', json)
})
```

In theory, the complete backend can be stubbed out in cypress tests by using `cy.intercept` and specifying the expected response.

It can also be used to modify requests before they are sent.

Using Cypress with Rails

Capybara Pros & Cons

Capybara Pros & Cons

- **Pros**

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests
  - Generating test data on the fly when it's required

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests
  - Generating test data on the fly when it's required
- **Cons**

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests
  - Generating test data on the fly when it's required
- **Cons**
  - Debugging in the browser is difficult

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests
  - Generating test data on the fly when it's required
- **Cons**
  - Debugging in the browser is difficult
  - It's not fun writing tests (Browser closing/opening, no intelligent auto-reruns, etc.)

Capybara Pros & Cons

- **Pros**
  - Access to the whole Rails application inside E2E tests
  - Generating test data on the fly when it's required
- **Cons**
  - Debugging in the browser is difficult
  - It's not fun writing tests (Browser closing/opening, no intelligent auto-reruns, etc.)
  - Very flaky with SPAs

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

🤷‍♀️

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

```ruby
cy.visit("/")
```

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

```
cy.contains("Login").click()
```

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

There is a cypress equivalent for all remaining commands

Capybara to Cypress

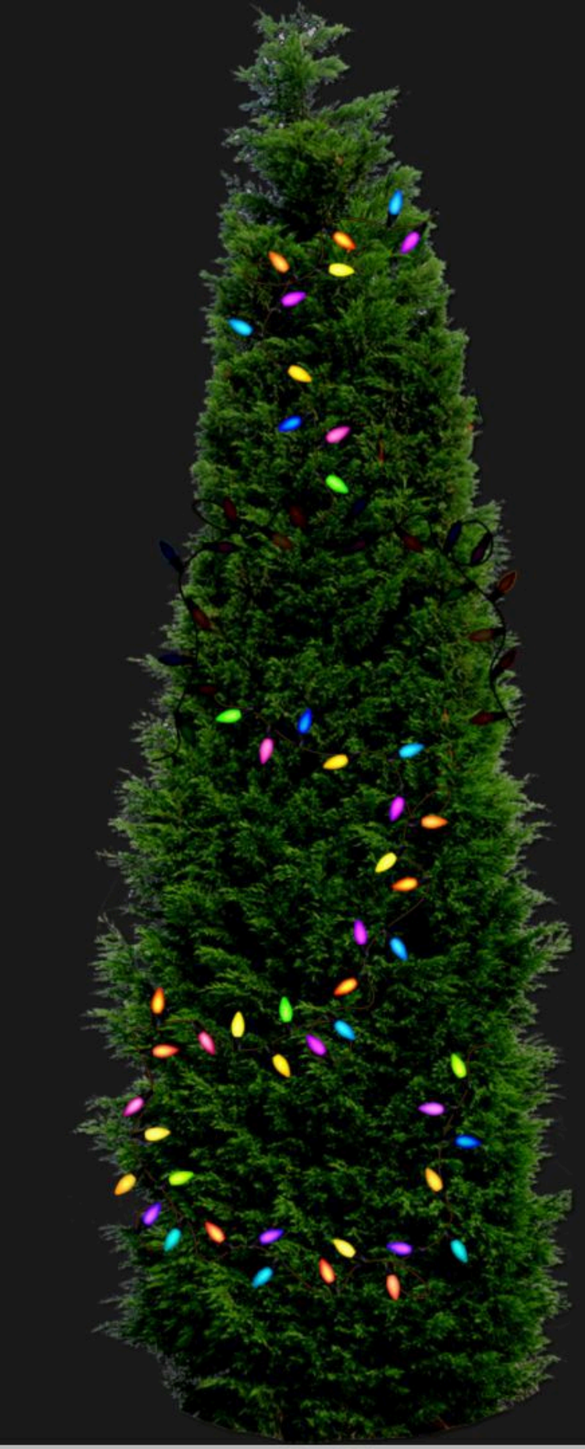
```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }

  it "logs you in" do
    visit root_path
    click_on "Login"

    fill_in "Email", with: user.email
    fill_in "Password", with: "12345678"
    click_button "Login"

    expect(find("nav").text).to include(user.email)
  end
end
```

So, how do we get test data into our Rails application on-the-fly?

Cypress on Rails

Cypress on Rails

- **The `cypress-on-rails` gem**



Cypress on Rails

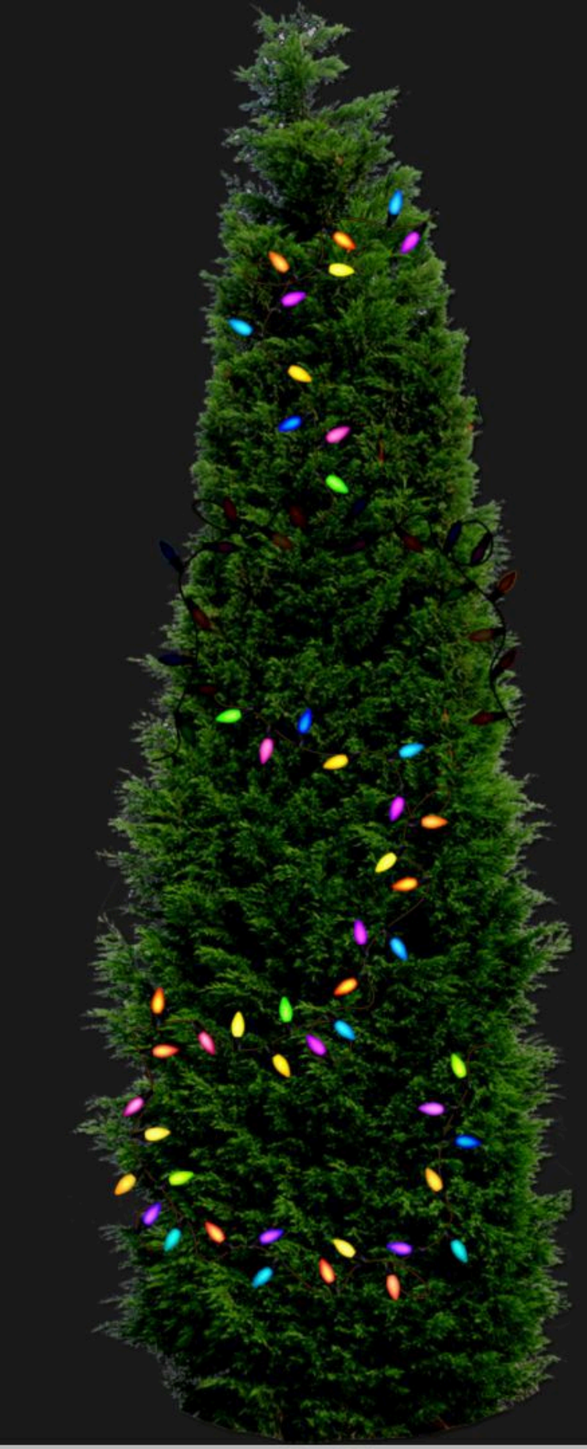
- **The `cypress-on-rails` gem**
  - Adds a `rack` middleware to send requests to the Rails server

Cypress on Rails

- **The `cypress-on-rails` gem**
  - Adds a `rack` middleware to send requests to the Rails server
  - Provides conventions on how to run code on the server from within Cypress

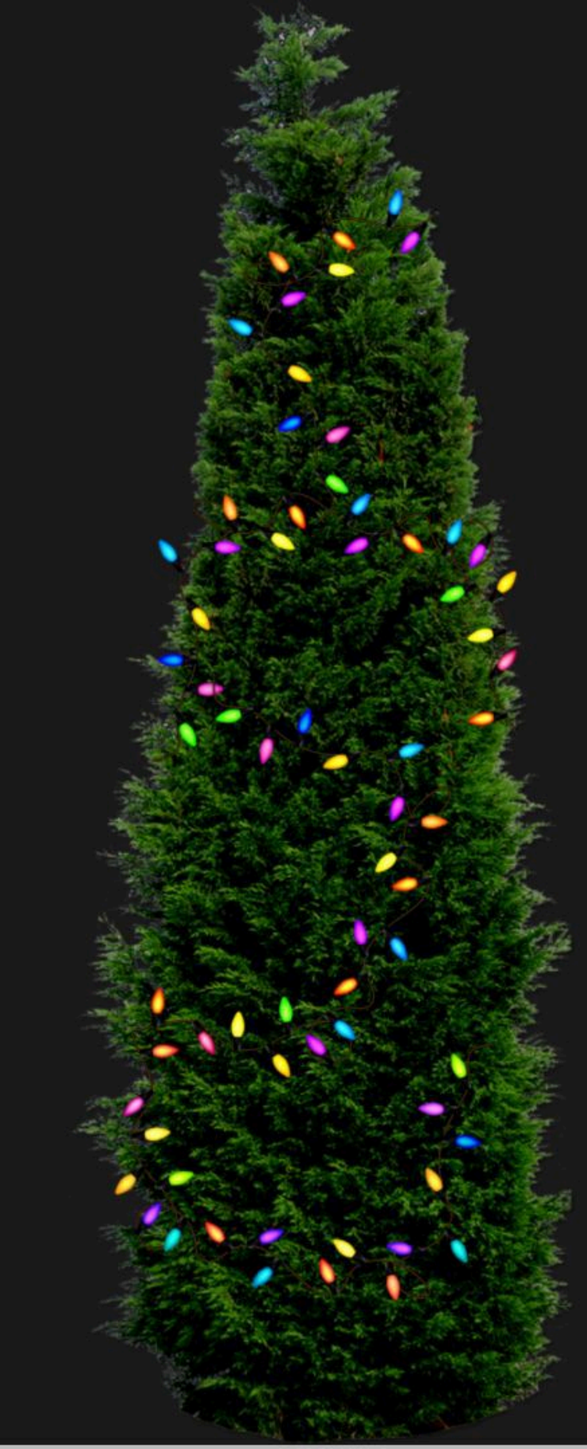
Cypress on Rails

- **The `cypress-on-rails` gem**
  - Adds a `rack` middleware to send requests to the Rails server
  - Provides conventions on how to run code on the server from within Cypress
- **Own Extensions**



Cypress on Rails

- **The `cypress-on-rails` gem**
  - Adds a `rack` middleware to send requests to the Rails server
  - Provides conventions on how to run code on the server from within Cypress
- **Own Extensions**
  - Flexible `FactoryBot` integration with association support



Cypress on Rails

- **The `cypress-on-rails` gem**
  - Adds a `rack` middleware to send requests to the Rails server
  - Provides conventions on how to run code on the server from within Cypress
- **Own Extensions**
  - Flexible `FactoryBot` integration with association support
  - Scenario-Wrapper to adjust JSON responses

cypress-on-rails

cypress-on-rails

- Adds a helper file that's ran before any cypress command

cypress-on-rails

- Adds a helper file that's ran before any cypress command
- Adds the `"/__cypress__/command"` endpoint to the application

cypress-on-rails

- Adds a helper file that's ran before any cypress command
- Adds the `"/__cypress__/command"` endpoint to the application
- This endpoint `evals` a requested Ruby file (in the application context) and responds with its return value as JSON

cypress-on-rails

```
spec
|- cypress
    |- cypress_helper.rb
    |- app_commands
        |- clean.rb
        |- factory_bot.rb
        |- ...
    |- integration
        |- login_spec.js
    |- fixtures
```

cypress-on-rails

```
spec
|- cypress
   |- cypress_helper.rb
   |- app_commands
      |- clean.rb
      |- factory_bot.rb
      |- ...
   |- integration
      |- login_spec.js
   |- fixtures
```

Is ran once at server start

cypress-on-rails

```
spec
|- cypress
   |- cypress_helper.rb
   |- app_commands
      |- clean.rb
      |- factory_bot.rb
      |- ...
   |- integration
      |- login_spec.js
   |- fixtures
```

cypress-on-rails

```
spec
|- cypress
   |- cypress_helper.rb
   |- app_commands
      |- clean.rb
      |- factory_bot.rb
      |- ...
|- integration
   |- login_spec.js
|- fixtures
```

Normal cypress folder structure

Using the command endpoint

```
Cypress.Commands.add('appCommands', function(body) {
  return cy.request({
    method: 'POST',
    url: "/__cypress__/command",
    body: JSON.stringify(body),
  }).then((response) => {
    return response.body;
  });
});

Cypress.Commands.add('app', function(name, command_options) {
  return cy.appCommands({name: name, options: command_options})
    .then(response => response[0]);
});

cy.app("factory_bot", ["create", "post"]);
```

Using the command endpoint

```
Cypress.Commands.add('appCommands', function(body) {
  return cy.request({
    method: 'POST',
    url: "/__cypress__/command",
    body: JSON.stringify(body),
  }).then((response) => {
    return response.body;
  });
});

Cypress.Commands.add('app', function(name, command_options) {
  return cy.appCommands({name: name, options: command_options})
    .then(response => response[0]);
});

cy.app("factory_bot", ["create", "post"]);
```

Basic POST request to the Rails server

Using the command endpoint

```
Cypress.Commands.add('appCommands', function(body) {
  return cy.request({
    method: 'POST',
    url: "/__cypress__/command",
    body: JSON.stringify(body),
  }).then((response) => {
    return response.body;
  });
});

Cypress.Commands.add('app', function(name, command_options) {
  return cy.appCommands({name: name, options: command_options})
    .then(response => response[0]);
});

cy.app("factory_bot", ["create", "post"]);
```

Convenience helper to issue a command

Using the command endpoint

```
Cypress.Commands.add('appCommands', function(body) {
  return cy.request({
    method: 'POST',
    url: "/__cypress__/command",
    body: JSON.stringify(body),
  }).then((response) => {
    return response.body;
  });
});

Cypress.Commands.add('app', function(name, command_options) {
  return cy.appCommands({name: name, options: command_options})
    .then(response => response[0]);
});
```

```
cy.app("factory_bot", ["create", "post"]);
```

```
evaling spec/cypress/app_commands/factory_bot.rb
```

Cleaning between Tests

```ruby
# spec/cypress/app_commands/clean.rb

# Reset the database to a clean state
DatabaseCleaner.strategy = :truncation
DatabaseCleaner.clean

# Reload factories, reset sequences
FactoryBot.reload

Rails.logger.info "APPCLEANED" # used by log_fail.rb
```

```javascript
// spec/cypress/support/index.js
beforeEach(function() {
  cy.app("clean");
})
```

Capybara to Cypress

```ruby
RSpec.describe "Logging in" do
  let!(:user) { FactoryBot.create(:user) }
  ...
end
```

```javascript
cy.describe("Logging in", function() {
  beforeEach(function() {
    cy.app("factory_bot", ["create", "user"]).as("user");
  })
});
```

Creating data on-the-fly is now possible!

FactoryBot Integration v2

```ruby
# /app/models/post.rb
has_many :likes
has_many :liking_users, through: :likes, class_name: "User"

# /spec/feature/liking_spec.rb
RSpec.describe "Liking" do
  let(:user) { create(:user) }
  let(:post) { create(:post, liking_users: [user]) }
  ...
end
```

FactoryBot Integration v2

```ruby
# /app/models/post.rb
has_many :likes
has_many :liking_users, through: :likes, class_name: "User"

# /spec/feature/liking_spec.rb
RSpec.describe "Liking" do
  let(:user) { create(:user) }
  let(:post) { create(:post, liking_users: [user]) }
  ...
end
```

FactoryBot Integration v2

```ruby
# /app/models/post.rb
has_many :likes
has_many :liking_users, through: :likes, class_name: "User"

# /spec/feature/liking_spec.rb
RSpec.describe "Liking" do
  let(:user) { create(:user) }
  let(:post) { create(:post, liking_users: [user]) }
  ...
end
```

While `post.liking_user_ids=` is available here, this wouldn't work with polymorphic associations. So, again: 🤷‍♀️

FactoryBot Integration v2

```
beforeEach(function() {
  cy.factory("user").as("author");
  cy.factory("user").as("user");
});

it("creates a post", function() {
  cy.factory('post', [], {
    body: "I am a post!",
    author_identifier: this.author.gid,
    liking_users_identifiers: [this.user.gid]
  }).as('post');
});
```

FactoryBot Integration v2

```
beforeEach(function() {
  cy.factory("user").as("author");
  cy.factory("user").as("user");
});

it("creates a post", function() {
  cy.factory('post', [], {
    body: "I am a post!",
    author_identifier: this.author.gid,
    liking_users_identifiers: [this.user.gid]
  }).as('post');
});
```

Everything created by the `factory_bot` app command contains a Global ID

FactoryBot Integration v2

```
beforeEach(function() {
  cy.factory("user").as("author");
  cy.factory("user").as("user");
});

it("creates a post", function() {
  cy.factory('post', [], {
    body: "I am a post!",
    author_identifier: this.author.gid,
    liking_users_identifiers: [this.user.gid]
  }).as('post');
});
```

Every parameter that ends with `_identifier(s)` will be converted to an AR instance

Conclusion

Conclusion

- On-The-Fly Test data generation ☑️

Conclusion

- On-The-Fly Test data generation ☑️
- Database Cleaning between tests ☑️

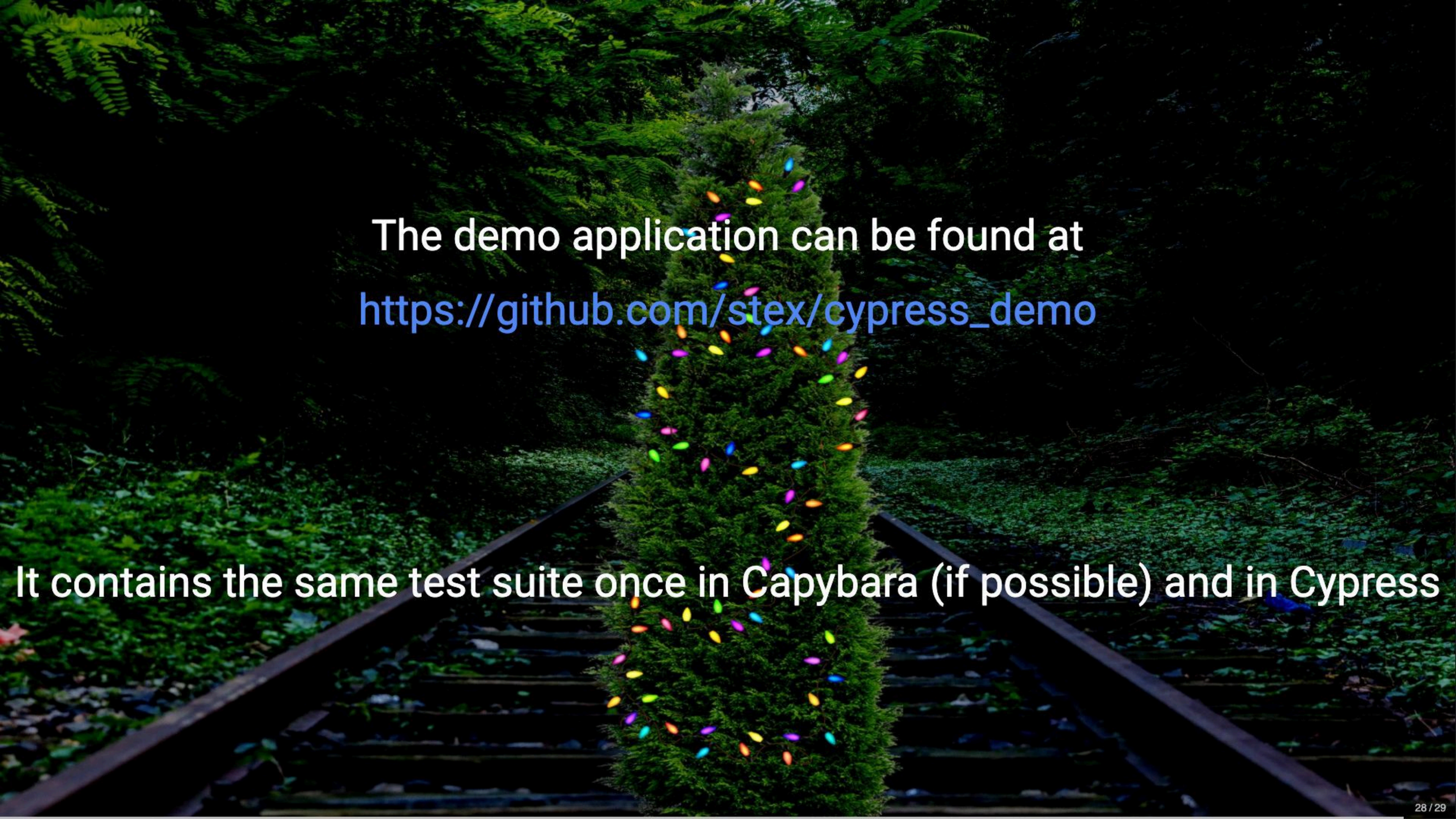
Conclusion

- On-The-Fly Test data generation ☑️
- Database Cleaning between tests ☑️
- Server-Side stubbing (e.g. Geocoder), etc. ☑️

Conclusion

- On-The-Fly Test data generation ✅
- Database Cleaning between tests ✅
- Server-Side stubbing (e.g. Geocoder), etc. ✅
- Everything I could do with Capybara ✅


Additional Topics
AppCommands and AppScenarios (v2)
Usage in CI (with paralellization)
How to organize/build spec files
Something else?


The demo application can be found at
https://github.com/stex/cypress_demo
It contains the same test suite once in Capybara (if possible) and in Cypress


Thanks!