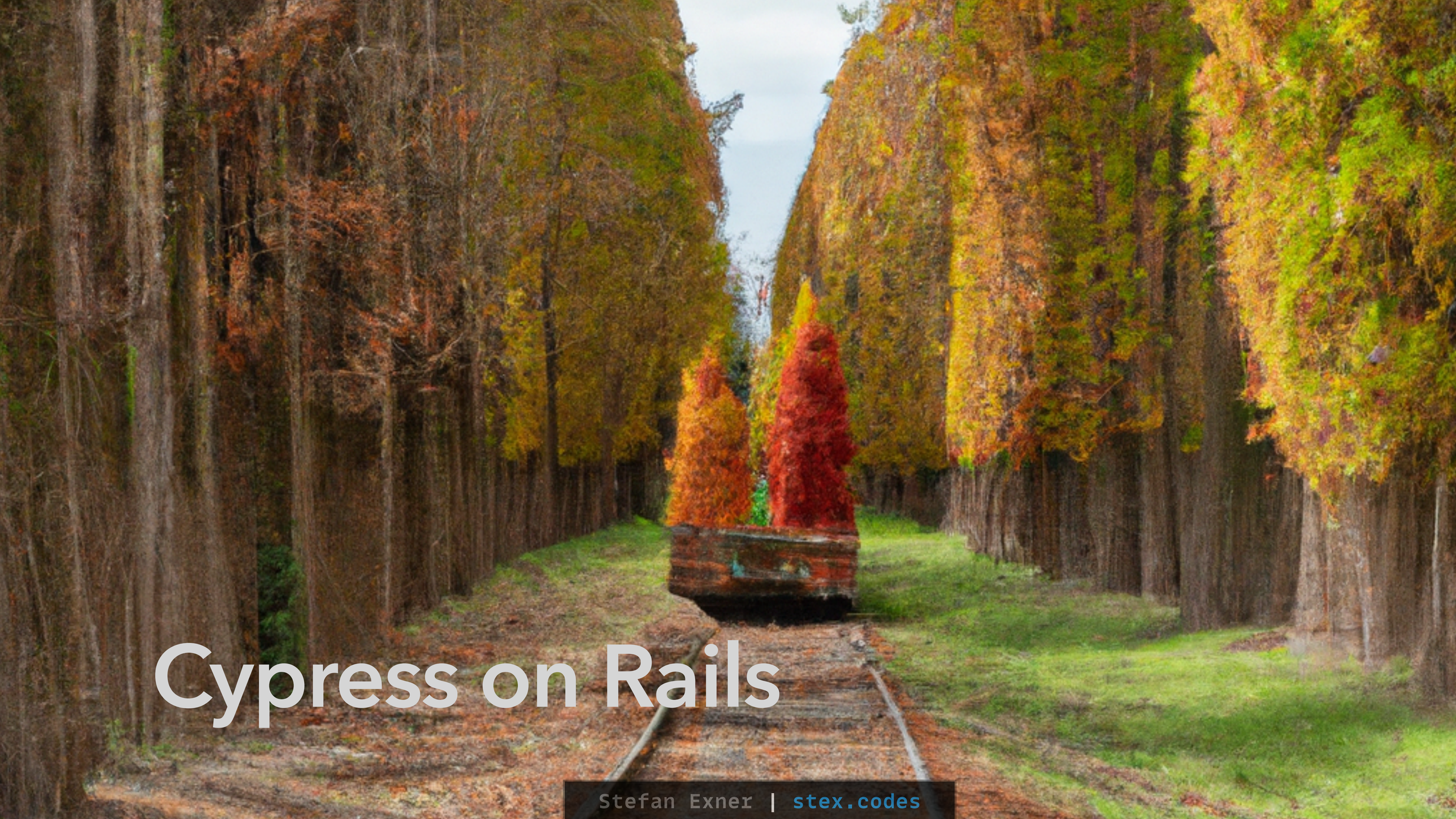


Cypress on Rails

Stefan Exner | stex.codes

Overview

Overview

- Introduction to Cypress

Overview

- Introduction to Cypress
- The Pros and Cons of Capybara

Overview

- Introduction to Cypress
- The Pros and Cons of Capybara
- Bringing Capybara Functionality to Cypress

Overview

- Introduction to Cypress
- The Pros and Cons of Capybara
- Bringing Capybara Functionality to Cypress
- Containerisation and CI

Overview

- Introduction to Cypress
- The Pros and Cons of Capybara
- Bringing Capybara Functionality to Cypress
- Containerisation and CI
- Questions / Custom Application Requirements

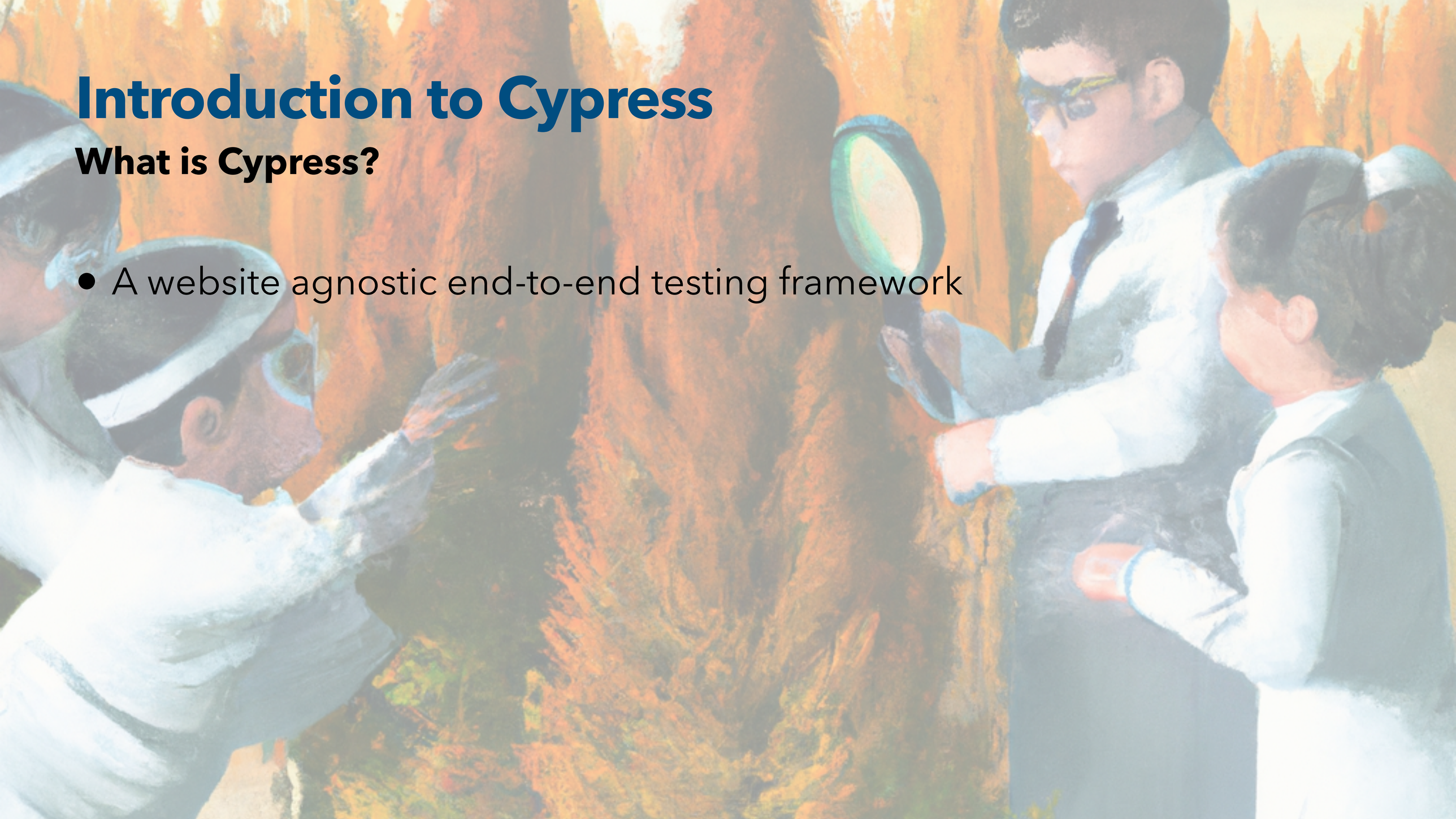
Most of the code from this presentation can be found at

https://github.com/stex/cypress_demo

Introduction to Cypress

Introduction to Cypress

What is Cypress?



Introduction to Cypress

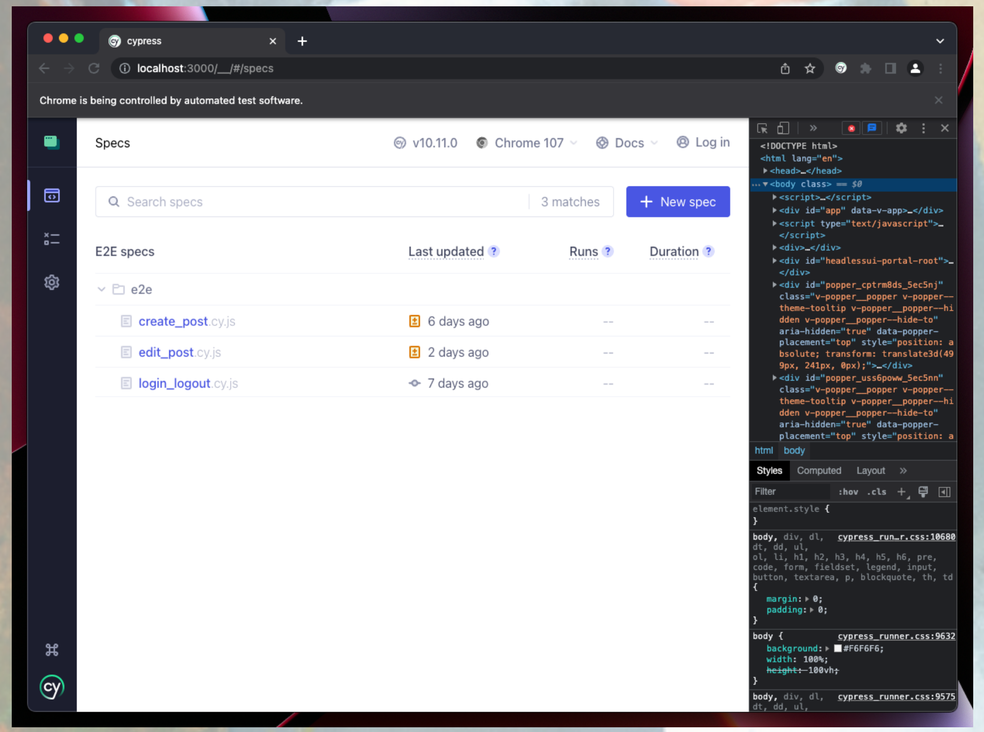
What is Cypress?

- A website agnostic end-to-end testing framework

Introduction to Cypress

What is Cypress?

- A website agnostic end-to-end testing framework
- Runs in the browser, doesn't need Selenium



Introduction to Cypress

What is Cypress?

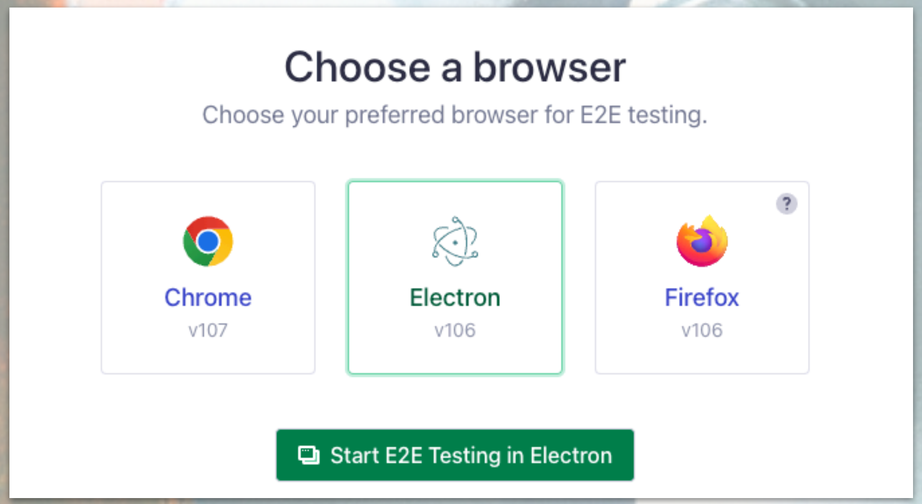
- A website agnostic end-to-end testing framework
- Runs in the browser, doesn't need Selenium
- Tests are written in Javascript
  (with Mocha, Chai and other caffeine puns)

```
describe('Login and Logout', function () {
 it('allows me to log in and out', function () {
    cy.visit('/login')

    cy.get('input[name=login]').type('user@example.com')
    cy.get('input[name=password]').type('12345678')
    cy.contains('input', 'Login').click()

    cy.get('#navbarNav').should('contain', 'Logout')

    cy.contains('button', 'Logout').click()
    cy.contains('.nav-link', 'Login')
  })
})
```

Introduction to Cypress

What is Cypress?

- A website agnostic end-to-end testing framework
- Runs in the browser, doesn't need Selenium
- Tests are written in Javascript
  (with Mocha, Chai and other caffeine puns)
- Supports Electron, Firefox and Chromium-based browsers (like the new Edge)



Introduction to Cypress

Cypress Features

Introduction to Cypress

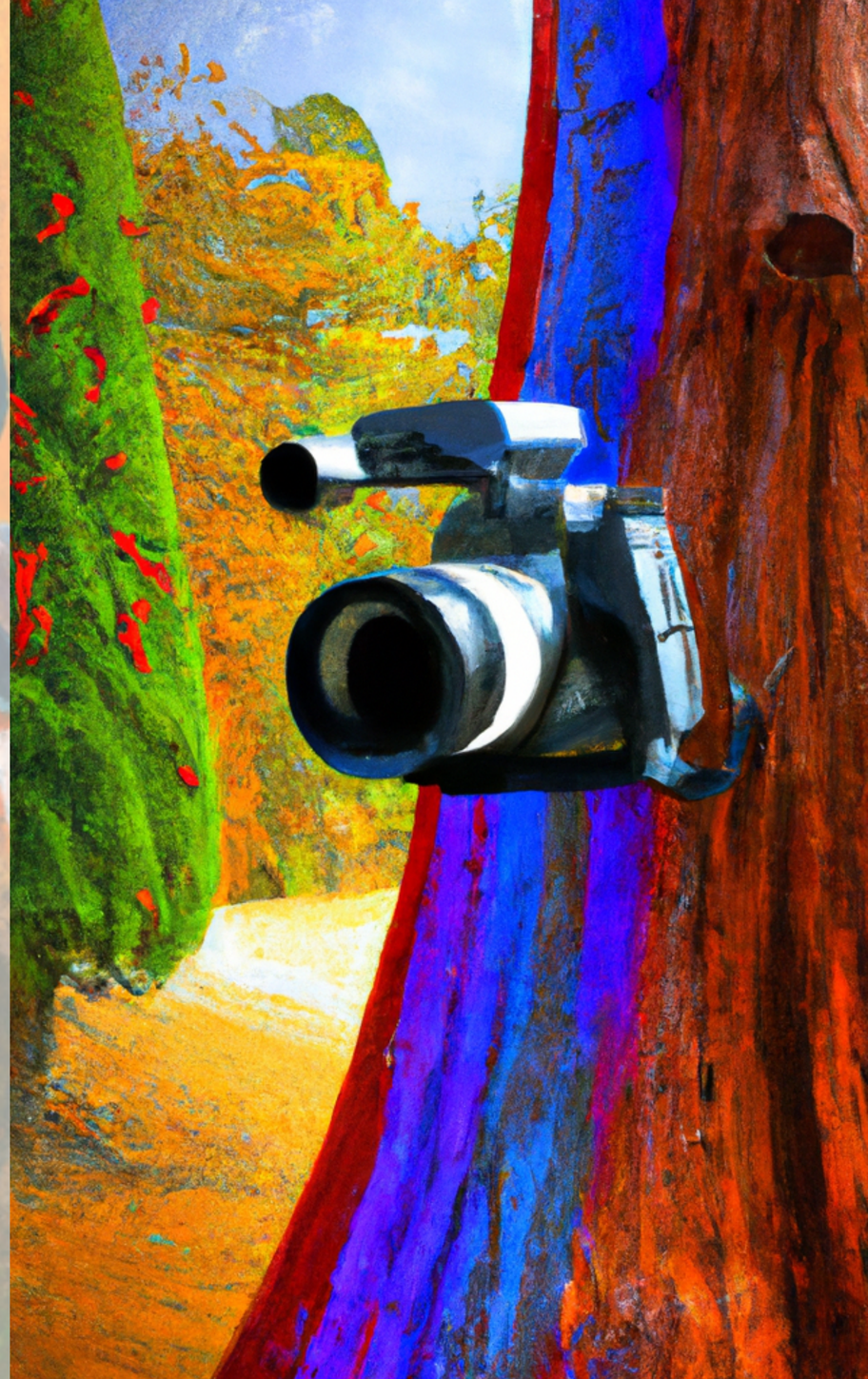
Cypress Features

- Automatic retries of single commands, perfect for SPAs which need a bit of time to load their content

Introduction to Cypress

Cypress Features

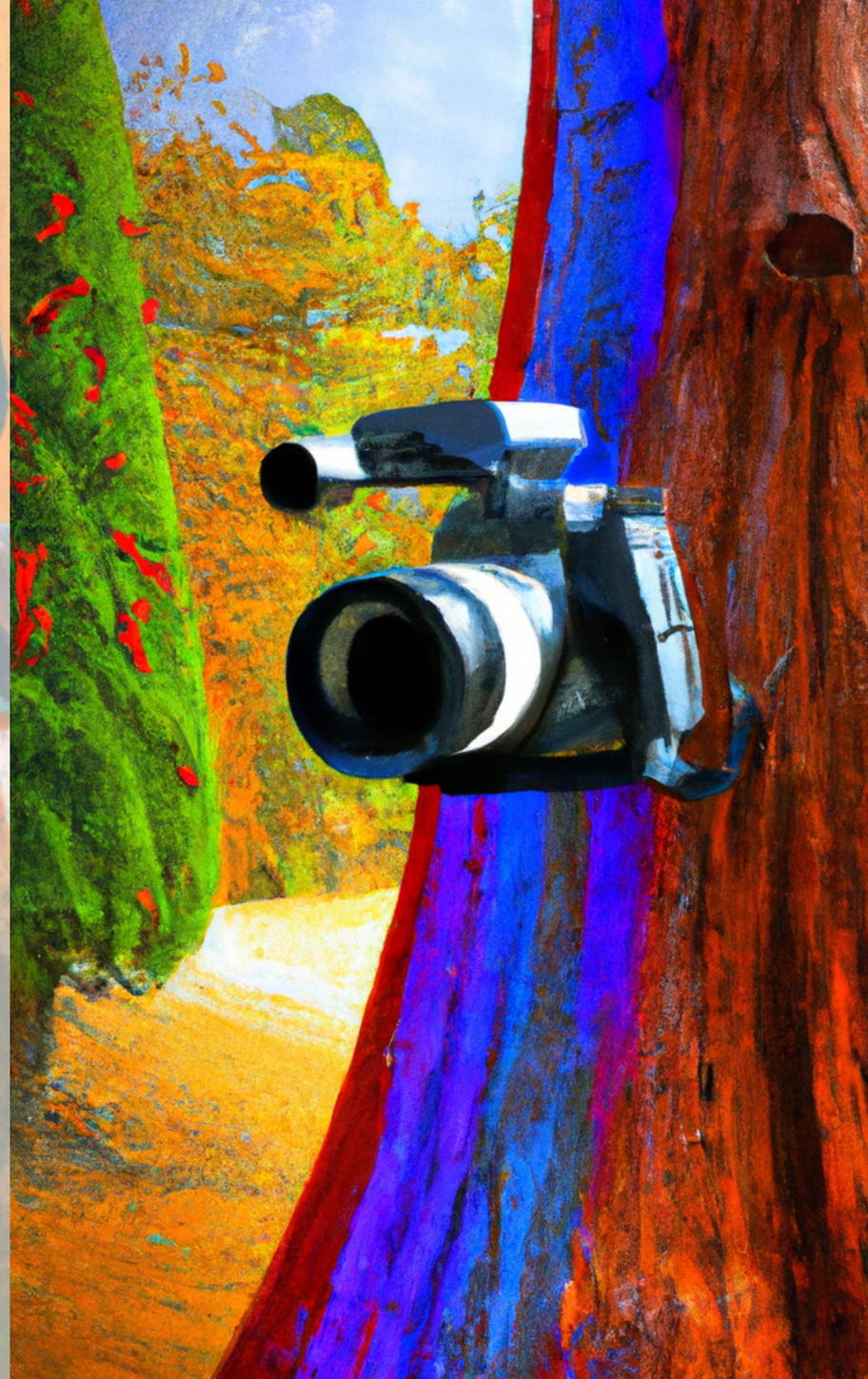
- Automatic retries of single commands, perfect for SPAs which need a bit of time to load their content
- Saves DOM snapshots for each step in the test, allows going back to each one in the UI



Introduction to Cypress

Cypress Features

- Automatic retries of single commands, perfect for SPAs which need a bit of time to load their content
- Saves DOM snapshots for each step in the test, allows going back to each one in the UI
- Each test can be saved as video, perfect for CI usage



Introduction to Cypress

Cypress Features

- Automatic retries of single commands, perfect for SPAs which need a bit of time to load their content
- Saves DOM snapshots for each step in the test, allows going back to each one in the UI
- Each test can be saved as video, perfect for CI usage
- Integrated mechanisms for parallel execution*

*requires paying for Cypress Dashboard

Introduction to Cypress

Hooks and Fixtures

Introduction to Cypress

Hooks and Fixtures

- Supports `before/after` hooks

```
// fixtures/user.json
{
   "email": "user@example.com",
   "password": "12345678"
}
```

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

  it("logs me in", function () {
    cy.contains("a", "Login").click()

    cy.get("input[name=email]").type(this.user.email)
    cy.get("input[name=password").type(this.user.password)
    cy.get("input[type=submit]").click()

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

Introduction to Cypress

Hooks and Fixtures

- Supports `before/after` hooks
- Supports fixture files, makes them available inside tests

```
// fixtures/user.json
{
  "email": "user@example.com",
  "password": "12345678"
}
```

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

  it("logs me in", function () {
    cy.contains("a", "Login").click()

    cy.get("input[name=email]").type(this.user.email)
    cy.get("input[name=password").type(this.user.password)
    cy.get("input[type=submit]").click()

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

Introduction to Cypress

Hooks and Fixtures

- Supports `before/after` hooks
- Supports fixture files, makes them available inside tests
- Most file types are supported out-of-the-box as fixtures.
  Some are automatically validated.

```
// fixtures/beaver_angry.png
```

```
describe("Image Fixtures", function () {
  beforeEach(function () {
    cy.fixture("beaver_angry.png").as("beforeBeaver")
  })

  it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver) // base64 representation of beaver image
    })

    // or

    cy.log(this.beforeBeaver)
  })

  it("supports binary representation", function () {
    cy.fixture("beaver_angry.png", "binary").then(beaver => {
      // binary representation of beaver, may e.g.
      // be attached to DOM elements / file lists / ...
    })
  })
})
```

Introduction to Cypress

Request Stubbing

Introduction to Cypress

Request Stubbing

- Allows stubbing out requests made by the application

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

// or

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

Introduction to Cypress

Request Stubbing

- Allows stubbing out requests made by the application
- In theory, the complete backend could be stubbed out this way.

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

// or

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

Introduction to Cypress

Interlude: Commands are not Promises...

```
describe("Image Fixtures", function () {
 it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver)
    })
  })
})
```

Introduction to Cypress

Interlude: Commands are not Promises...

- Even though `.then()` looks a lot like a Promise,

```
describe("Image Fixtures", function () {
 it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver)
    })
  })
})
```

Introduction to Cypress

Interlude: Commands are not Promises...

- Even though `.then()` looks a lot like a Promise,
  - everything is executed synchronously

```
describe("Image Fixtures", function () {
 it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver)
    })
  })
})
```

Introduction to Cypress

Interlude: Commands are not Promises...

- Even though `.then()` looks a lot like a Promise,
  - everything is executed synchronously
  - `return` is not supported

```
describe("Image Fixtures", function () {
 it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver)
    })
  })
})
```

Introduction to Cypress

Interlude: Commands are not Promises...

- Even though `.then()` looks a lot like a Promise,
  - everything is executed synchronously
  - `return` is not supported
  - there is no `catch`

```
describe("Image Fixtures", function () {
 it("supports base64 representation", function () {
    cy.fixture("beaver_angry.png").then(beaver => {
      cy.log(beaver)
    })
  })
})
```

Introduction to Cypress

Interlude: ...but what are they?

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")
    let elem = cy.get(".submit-button")

    debugger

    elem.click()
  })
})
```

Introduction to Cypress

Interlude: ...but what are they?

- Each Cypress command returns a `Chainer` instance

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")
    let elem = cy.get(".submit-button")

    debugger

    elem.click()
  })
})
```

Introduction to Cypress

Interlude: ...but what are they?

- Each Cypress command returns a `Chainer` instance
- All commands are enqueued before anything is actually executed and then ran asynchronously

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")
    let elem = cy.get(".submit-button")

    debugger

    elem.click()
  })
})
```

Introduction to Cypress

Interlude: ...but what are they?

- Each Cypress command returns a `Chainer` instance
- All commands are enqueued before anything is actually executed and then ran asynchronously
  - `elem` is not correctly assigned in the final execution

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")
    let elem = cy.get(".submit-button")

    debugger

    elem.click()
  })
})
```

Introduction to Cypress

Interlude: ...but what are they?

- Each Cypress command returns a `Chainer` instance
- All commands are enqueued before anything is actually executed and then ran asynchronously
  - `elem` is not correctly assigned in the final execution
  - `debugger` is executed way before anything else is

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")
    let elem = cy.get(".submit-button")

    debugger

    elem.click()
  })
})
```

Introduction to Cypress

Interlude: Closures and Synchronous Execution

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")

    cy.get(".submit-button").then(elem => {
      debugger

      cy.wrap(elem).click()
    })
  })
})
```

Introduction to Cypress

Interlude: Closures and Synchronous Execution

- To get the element matched by `cy.get()`, we use `.then()` on the returned `Chainer` object

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")

    cy.get(".submit-button").then(elem => {
      debugger

      cy.wrap(elem).click()
    })
  })
})
```

Introduction to Cypress

Interlude: Closures and Synchronous Execution

- To get the element matched by `cy.get()`, we use `.then()` on the returned `Chainer` object

- Now we can inspect it in the debugger and run further commands on it

```
describe("Debugging", function () {
  it("let's me debug like a pro!", function () {
    cy.visit("/")

    cy.get(".submit-button").then(elem => {
      debugger

      cy.wrap(elem).click()
    })
  })
})
```

Introduction to Cypress

Interlude: Closures and Synchronous Execution

- To get the element matched by `cy.get()`, we use `.then()` on the returned `Chainer` object
- Now we can inspect it in the debugger and run further commands on it
- Important: `elem` is no longer a `Chainer` instance here, so we have to `wrap()` it in one before running other commands on it.

```
1  describe("Debugging", function () {
2    it("let's me debug like a pro!", function () {
3      cy.visit("/")
4
5      cy.get(".submit-button").then(elem ⇒ {
6        debugger
7
8        cy.wrap(elem).click()
9      })
10   })
11 })
```

Introduction to Cypress

Custom Commands

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

cy.tree().then(result => {
   expect(result).to.equal("🎄")
}
```

Introduction to Cypress

Custom Commands

- Cypress allows adding own commands which can be called directly on the `cy` object...

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

cy.tree().then(result => {
   expect(result).to.equal("🎄")
}
```

Introduction to Cypress

Custom Commands

- Cypress allows adding own commands which can be called directly on the `cy` object...
- ...or on a previous `Chainer` instance to get access to its result.

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

cy.tree().then(result => {
   expect(result).to.equal("🎄 ")
}
```

```
Cypress.Commands.add(
  "typeTrees",
  {
    prevSubject: "element"
  },
  (subject, treeCount) => {
    Array(treeCount).fill().forEach(() => {
      cy.wrap(subject).type("🎄 ")
  })
})


cy.get("input").typeTrees(3)
```

Introduction to Cypress

Custom Commands

- Cypress allows adding own commands which can be called directly on the `cy` object...
- ...or on a previous `Chainer` instance to get access to its result.
- The latter also allows certain validations. In this case, subject has to be a DOM element

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

cy.tree().then(result => {
   expect(result).to.equal("🎄 ")
}
```

```
Cypress.Commands.add(
  "typeTrees",
  {
    prevSubject: "element"
  },
  (subject, treeCount) => {
    Array(treeCount).fill().forEach(() => {
      cy.wrap(subject).type("🎄 ")
  })
})


cy.get("input").typeTrees(3)
```

Introduction to Cypress

Custom Commands

- A real-world example would be a login request that can be executed before the actual tests

```
1 Cypress.Commands.add('login', ({ email, password }) => {
2   cy.request('POST', '/login', {
3     login: email,
4     password
5   })
6 })
7
8 cy.login({email: "user@example.com", password: "12345678"})
```

The Pros and Cons of Capybara

The Pros and Cons of Capybara

Pros

```
class LoginLogoutsTest < ApplicationSystemTestCase
  setup do
    @account = FactoryBot.create(:account, :verified)
  end

  test "visiting the index" do
    visit "/login"

    page.find("input[name=login]").fill_in(with: @account.email)
    page.find("input[name=password]").fill_in(with: @account.password)
    click_button "Login"

    page.find("#navbarNav", text: "Logout")

    click_button "Logout"

    page.find(".nav-link", text: "Login")
  end
end
```

The Pros and Cons of Capybara

Pros

- Access to the whole Rails application

```ruby
class LoginLogoutsTest < ApplicationSystemTestCase
  setup do
    @account = FactoryBot.create(:account, :verified)
  end

  test "visiting the index" do
    visit "/login"

    page.find("input[name=login]").fill_in(with: @account.email)
    page.find("input[name=password]").fill_in(with: @account.password)
    click_button "Login"

    page.find("#navbarNav", text: "Logout")

    click_button "Logout"

    page.find(".nav-link", text: "Login")
  end
end
```

The Pros and Cons of Capybara

Pros

- Access to the whole Rails application
- Access to `FactoryBot` to generate test data on the fly

```
1  class LoginLogoutsTest < ApplicationSystemTestCase
2    setup do
3      @account = FactoryBot.create(:account, :verified)
4    end
5
6    test "visiting the index" do
7      visit "/login"
8
9      page.find("input[name=login]").fill_in(with: @account.email)
10     page.find("input[name=password]").fill_in(with: @account.password)
11     click_button "Login"
12
13     page.find("#navbarNav", text: "Logout")
14
15     click_button "Logout"
16
17     page.find(".nav-link", text: "Login")
18   end
19 end
```

The Pros and Cons of Capybara

Pros

- Access to the whole Rails application
- Access to `FactoryBot` to generate test data on the fly
- Rails comes pre-configured with it

```ruby
class LoginLogoutsTest < ApplicationSystemTestCase
  setup do
    @account = FactoryBot.create(:account, :verified)
  end

  test "visiting the index" do
    visit "/login"

    page.find("input[name=login]").fill_in(with: @account.email)
    page.find("input[name=password]").fill_in(with: @account.password)
    click_button "Login"

    page.find("#navbarNav", text: "Logout")

    click_button "Logout"

    page.find(".nav-link", text: "Login")
  end
end
```

The Pros and Cons of Capybara

Pros

- Access to the whole Rails application
- Access to `FactoryBot` to generate test data on the fly
- Rails comes pre-configured with it
- It's ruby.

```ruby
class LoginLogoutsTest < ApplicationSystemTestCase
  setup do
    @account = FactoryBot.create(:account, :verified)
  end

  test "visiting the index" do
    visit "/login"

    page.find("input[name=login]").fill_in(with: @account.email)
    page.find("input[name=password]").fill_in(with: @account.password)
    click_button "Login"

    page.find("#navbarNav", text: "Logout")

    click_button "Logout"

    page.find(".nav-link", text: "Login")
  end
end
```

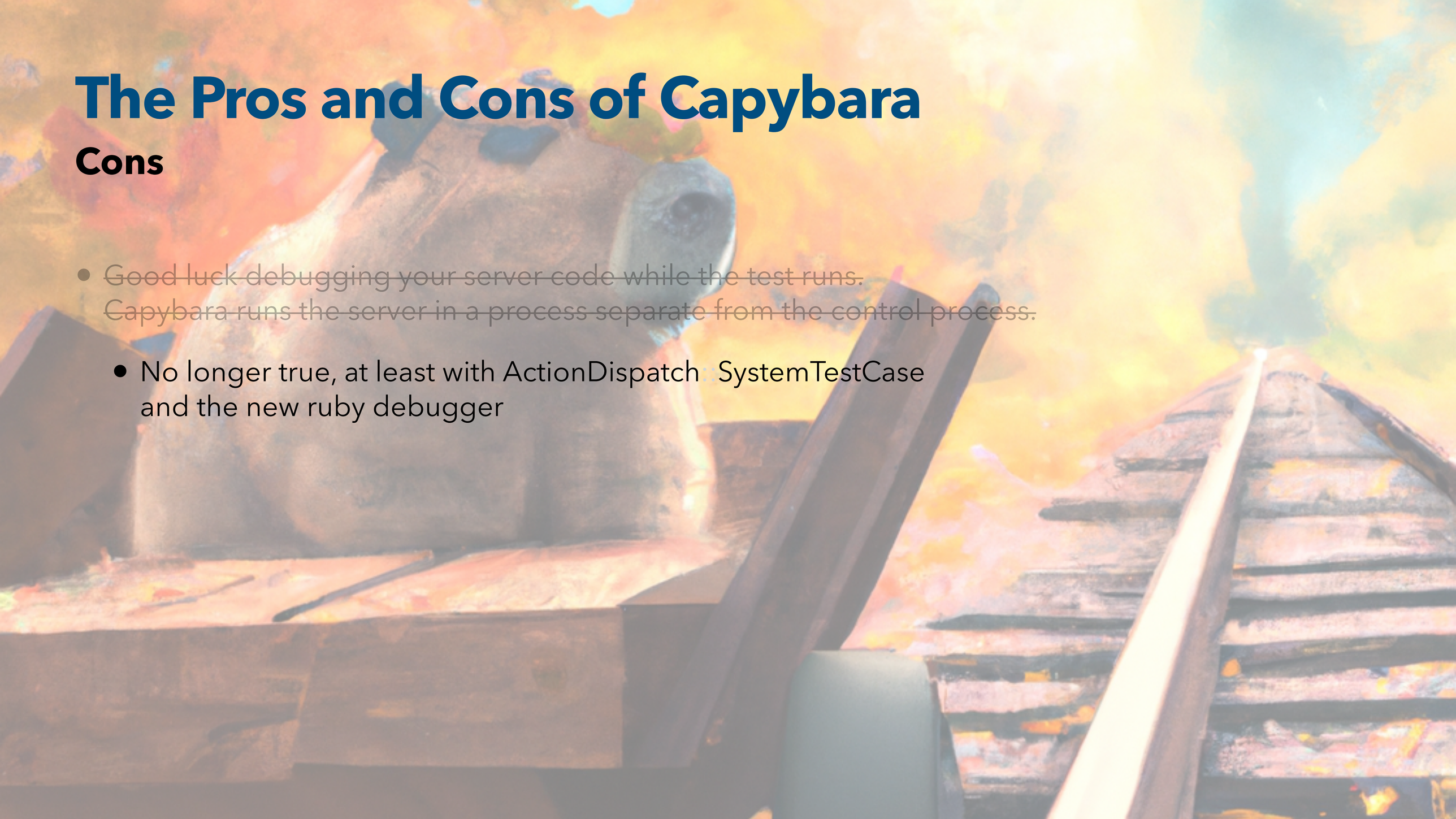
The Pros and Cons of Capybara

Cons

The Pros and Cons of Capybara

Cons

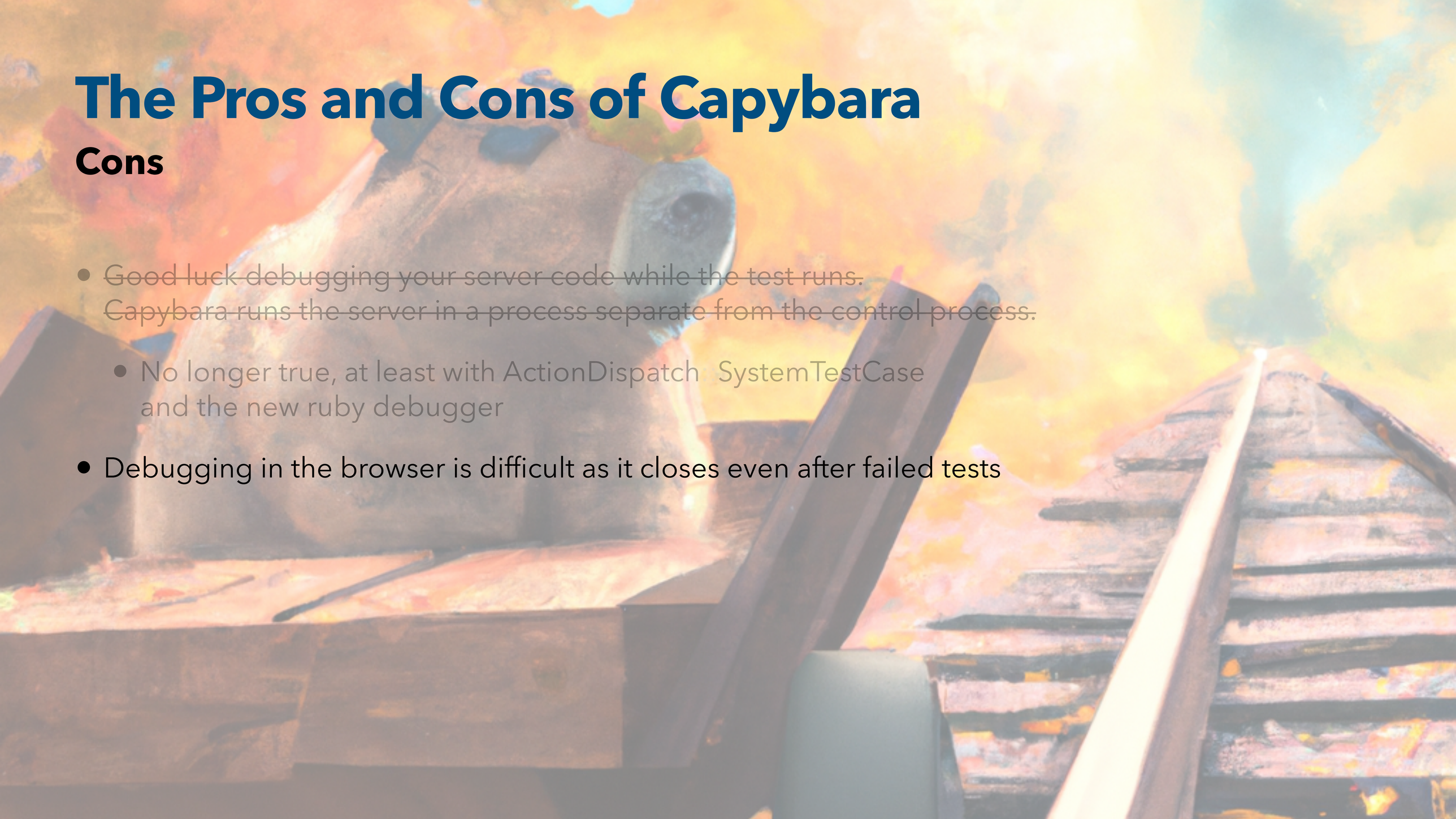
- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~



The Pros and Cons of Capybara

Cons

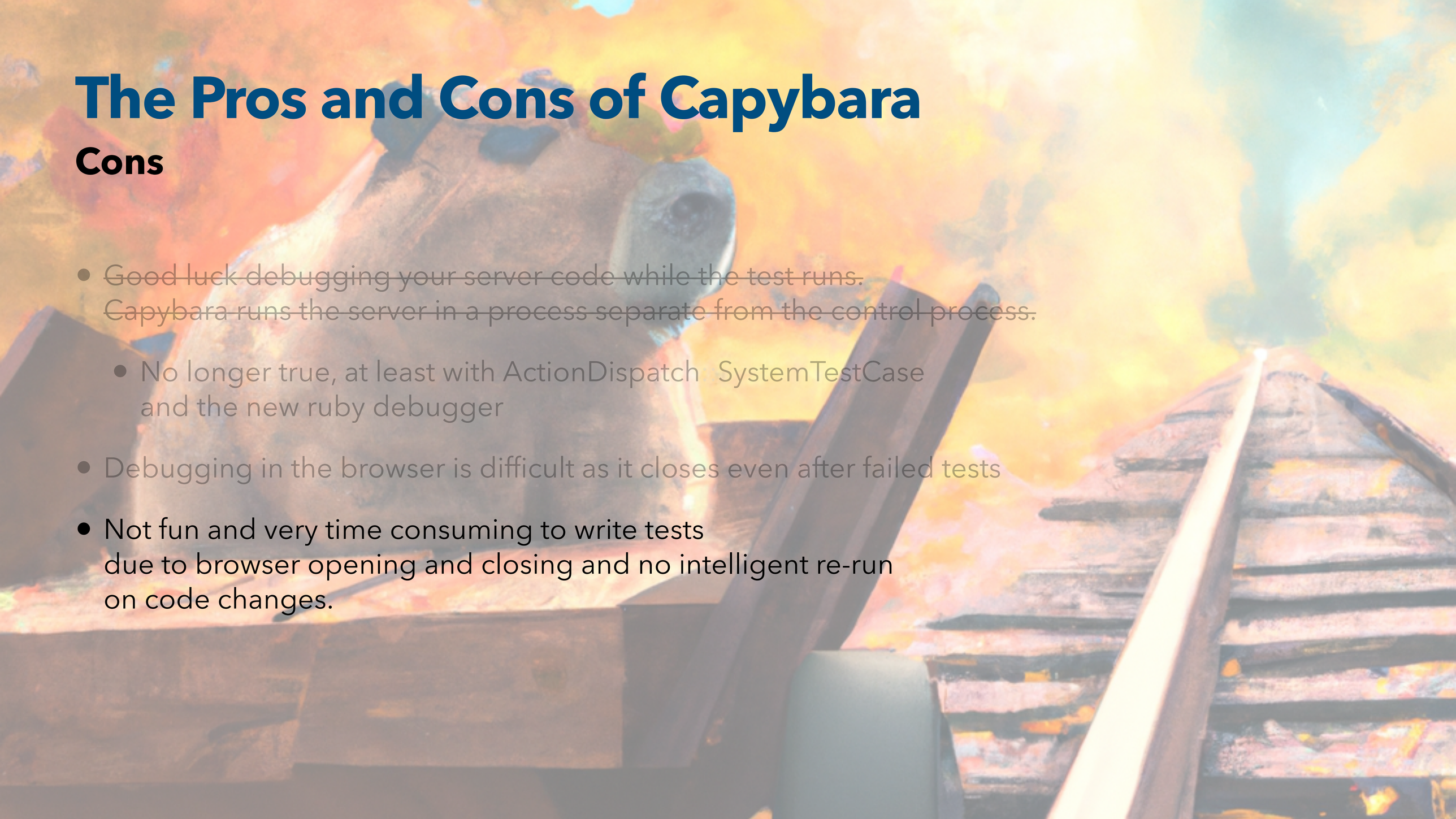
- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~
  - No longer true, at least with ActionDispatch::SystemTestCase and the new ruby debugger



The Pros and Cons of Capybara

Cons

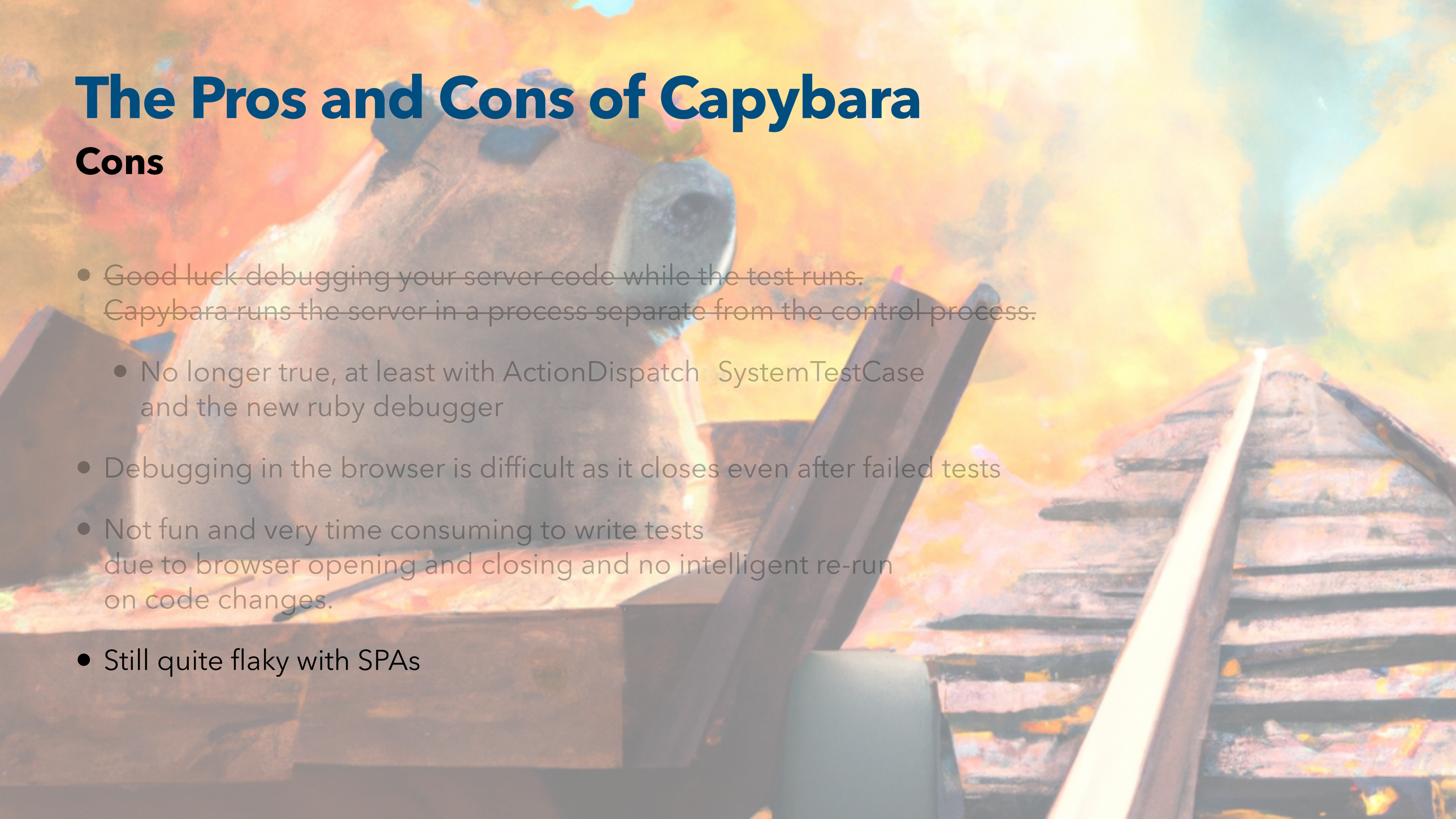
- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~
  - No longer true, at least with ActionDispatch::SystemTestCase and the new ruby debugger
- Debugging in the browser is difficult as it closes even after failed tests



The Pros and Cons of Capybara

Cons

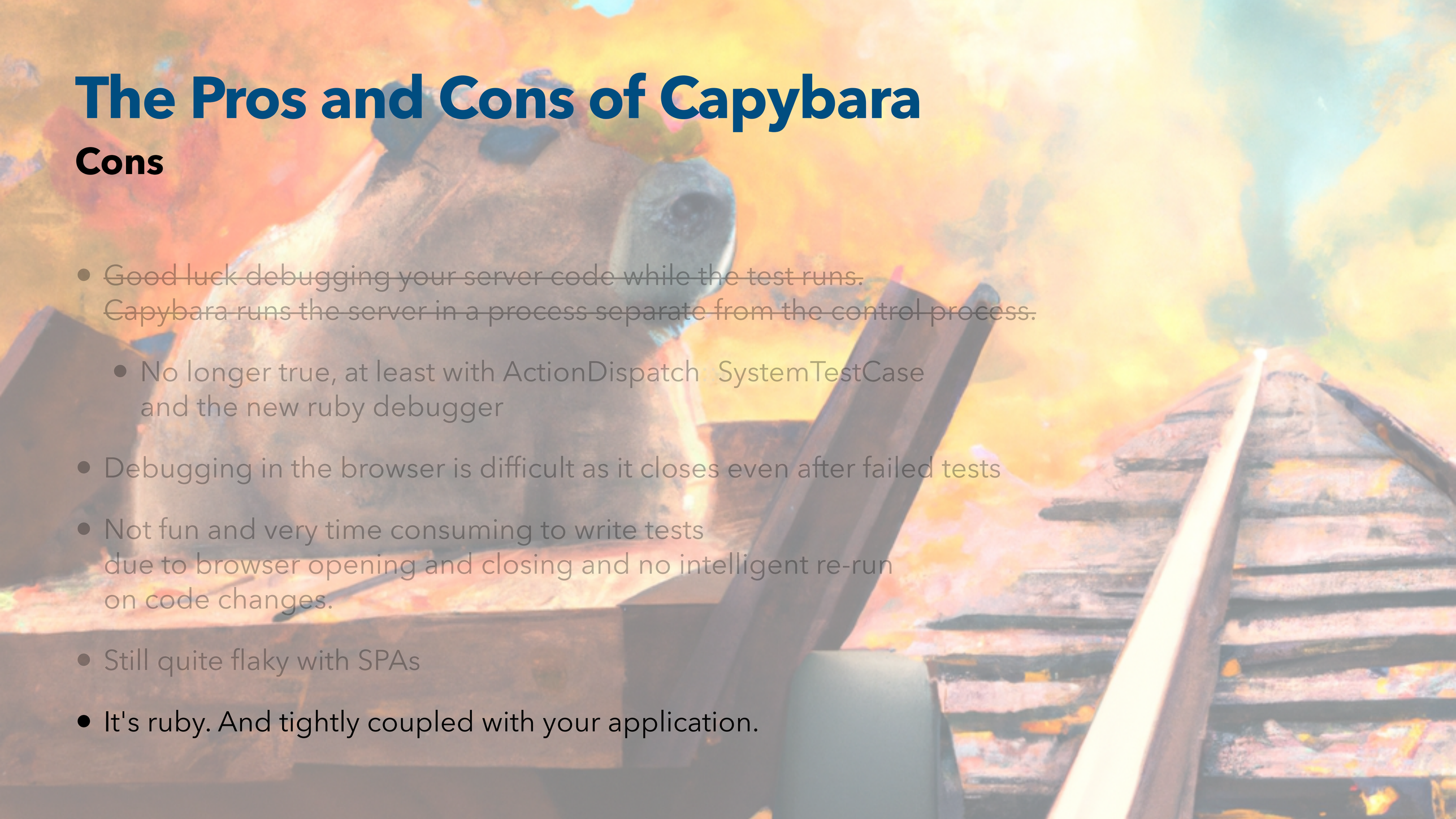
- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~
  - No longer true, at least with ActionDispatch::SystemTestCase and the new ruby debugger
- Debugging in the browser is difficult as it closes even after failed tests
- Not fun and very time consuming to write tests due to browser opening and closing and no intelligent re-run on code changes.



The Pros and Cons of Capybara

Cons

- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~
  - No longer true, at least with ActionDispatch::SystemTestCase and the new ruby debugger
- Debugging in the browser is difficult as it closes even after failed tests
- Not fun and very time consuming to write tests due to browser opening and closing and no intelligent re-run on code changes.
- Still quite flaky with SPAs



The Pros and Cons of Capybara

Cons

- ~~Good luck debugging your server code while the test runs.~~
  ~~Capybara runs the server in a process separate from the control process.~~
  - No longer true, at least with ActionDispatch::SystemTestCase and the new ruby debugger
- Debugging in the browser is difficult as it closes even after failed tests
- Not fun and very time consuming to write tests due to browser opening and closing and no intelligent re-run on code changes.
- Still quite flaky with SPAs
- It's ruby. And tightly coupled with your application.


Bringing Capybara
Functionality to Cypress

Capybara to Cypress

Translating a Capybara Spec to a Cypress Spec

```ruby
class LoginLogoutsTest < ApplicationSystemTestCase
  setup do
    @account = FactoryBot.create(:account, :verified)
  end

  test "visiting the index" do
    visit "/login"

    page.find("input[name=login]").fill_in(with: @account.email)
    page.find("input[name=password]").fill_in(with: @account.password)
    click_button "Login"

    page.find("#navbarNav", text: "Logout")

    click_button "Logout"

    page.find(".nav-link", text: "Login")
  end
end
```

```javascript
describe('Login and Logout', function () {
  beforeEach(function () {
    cy.???.as('account')
  })

  it('allows me to log in and out', function () {
    cy.visit('/login')

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

    cy.get('#navbarNav').should('contain', 'Logout')

    cy.contains('button', 'Logout').click()

    cy.contains('.nav-link', 'Login')
  })
})
```

Cypress on Rails

The `cypress_on_rails` gem

Cypress on Rails

The `cypress_on_rails` gem

- `rack` middleware, allowing communication between cypress and the application

Cypress on Rails

The `cypress_on_rails` gem

- `rack` middleware, allowing communication between cypress and the application
- Conventions on how to execute code on the server from cypress tests

Cypress on Rails

The `cypress_on_rails` gem

- `rack` middleware, allowing communication between cypress and the application
- Conventions on how to execute code on the server from cypress tests
- Helper file that's executed once before all cypress tests

Cypress on Rails

The `cypress_on_rails` gem

- `rack` middleware, allowing communication between cypress and the application
- Conventions on how to execute code on the server from cypress tests
- Helper file that's executed once before all cypress tests
- The concept of "Scenarios" for often used test preparations or data

Cypress on Rails

The `cypress_on_rails` gem

- `rack` middleware, allowing communication between cypress and the application
- Conventions on how to execute code on the server from cypress tests
- Helper file that's executed once before all cypress tests
- The concept of "Scenarios" for often used test preparations or data
- **Important**: This doesn't mean that cypress and the rails server have to be running on the same machine. You could use an external staging server to run your E2E tests against*

*later more regarding why that might be quite dangerous though

Cypress on Rails

Opinionated Commands for `factory_bot` and scenarios (by me)

Cypress on Rails

Opinionated Commands for `factory_bot` and scenarios (by me)

- Custom commands to make Cypress feel like Rails tests (if wished)

Cypress on Rails

Opinionated Commands for `factory_bot` and scenarios (by me)

- Custom commands to make Cypress feel like Rails tests (if wished)
- Conventions for server responses

Cypress on Rails

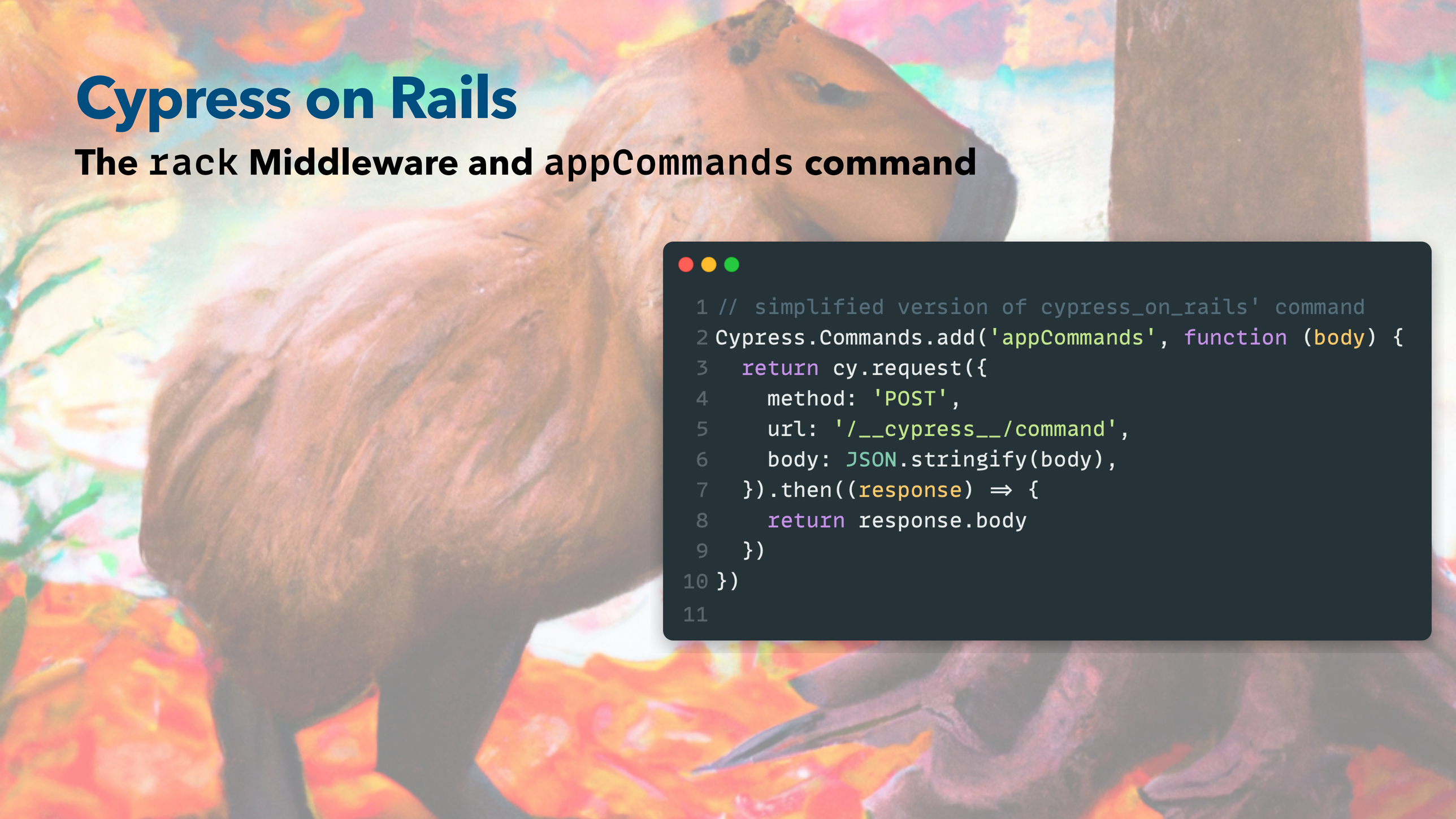
Opinionated Commands for `factory_bot` and scenarios (by me)

- Custom commands to make Cypress feel like Rails tests (if wished)
- Conventions for server responses
- Allow passing polymorphic associations to factories

Cypress on Rails

Opinionated Commands for `factory_bot` and scenarios (by me)

- Custom commands to make Cypress feel like Rails tests (if wished)
- Conventions for server responses
- Allow passing polymorphic associations to factories
- The gem works great without those, they are mostly sugar



Cypress on Rails

The `rack` Middleware and `appCommands` command

```
1 // simplified version of cypress_on_rails' command
2 Cypress.Commands.add('appCommands', function (body) {
3   return cy.request({
4     method: 'POST',
5     url: '/__cypress__/command',
6     body: JSON.stringify(body),
7   }).then((response) => {
8     return response.body
9   })
10 })
11
```

Cypress on Rails

The `rack` Middleware and `appCommands` command

- Middleware registers route on rack level

```
1 // simplified version of cypress_on_rails' command
2 Cypress.Commands.add('appCommands', function (body) {
3   return cy.request({
4     method: 'POST',
5     url: '/__cypress__/command',
6     body: JSON.stringify(body),
7   }).then((response) => {
8     return response.body
9   })
10 })
11
```

Cypress on Rails

The `rack` Middleware and `appCommands` command

- Middleware registers route on rack level
- Custom cypress command to send POST requests to the middleware

```
1 // simplified version of cypress_on_rails' command
2 Cypress.Commands.add('appCommands', function (body) {
3   return cy.request({
4     method: 'POST',
5     url: '/__cypress__/command',
6     body: JSON.stringify(body),
7   }).then((response) => {
8     return response.body
9   })
10 })
11
```

Cypress on Rails

The `rack` Middleware and `appCommands` command

- Middleware registers route on rack level
- Custom cypress command to send POST requests to the middleware
- Middleware can (and should!) be configured to only be loaded in test environments

```
1 // simplified version of cypress_on_rails' command
2 Cypress.Commands.add('appCommands', function (body) {
3   return cy.request({
4     method: 'POST',
5     url: '/__cypress__/command',
6     body: JSON.stringify(body),
7   }).then((response) => {
8     return response.body
9   })
10 })
11
```

Cypress on Rails

The `rack` Middleware and `appCommands` command

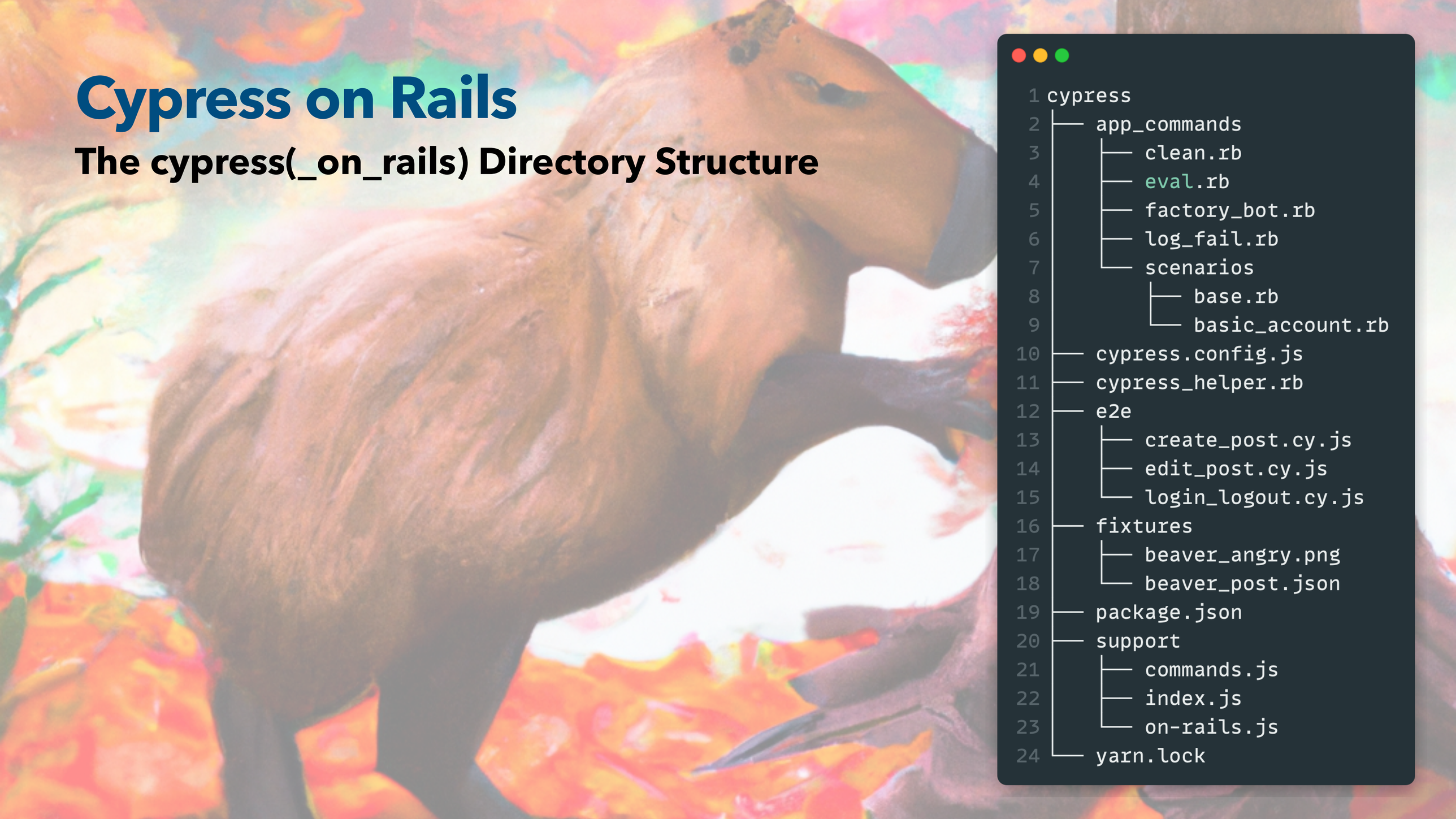
- Middleware registers route on rack level
- Custom cypress command to send POST requests to the middleware
- Middleware can (and should!) be configured to only be loaded in test environments

```
1 // simplified version of cypress_on_rails' command
2 Cypress.Commands.add('appCommands', function (body) {
3   return cy.request({
4     method: 'POST',
5     url: '/__cypress__/command',
6     body: JSON.stringify(body),
7   }).then((response) => {
8     return response.body
9   })
10 })
11
```

The middleware allows remote code execution! That's what it's made for!

Cypress on Rails

The cypress(_on_rails) Directory Structure

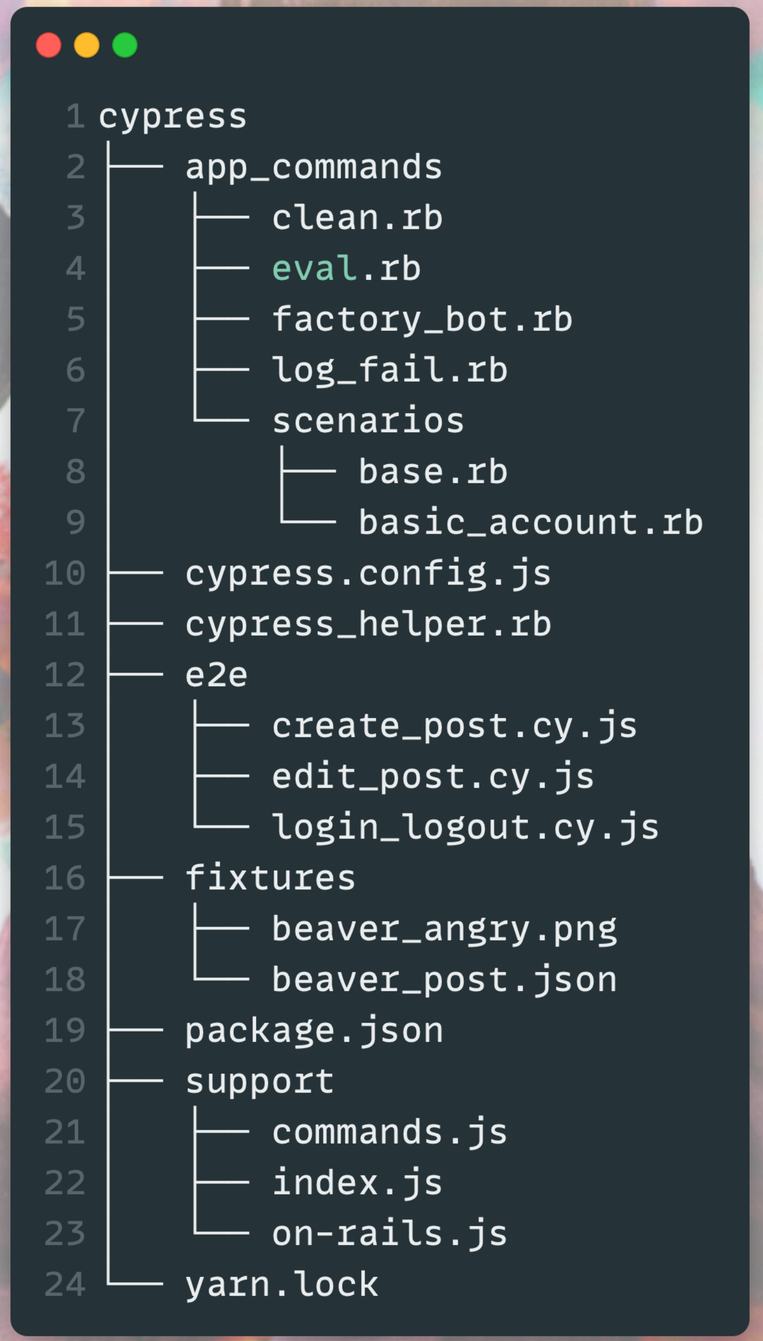


```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
├── cypress.config.js
├── cypress_helper.rb
├── e2e
│   ├── create_post.cy.js
│   ├── edit_post.cy.js
│   └── login_logout.cy.js
├── fixtures
│   ├── beaver_angry.png
│   └── beaver_post.json
├── package.json
├── support
│   ├── commands.js
│   ├── index.js
│   └── on-rails.js
└── yarn.lock
```

Cypress on Rails

The cypress(_on_rails) Directory Structure

- All test and support files for Cypress
  Used by the Cypress app and browser extension



Cypress on Rails

The cypress(_on_rails) Directory Structure

- All test and support files for Cypress
  Used by the Cypress app and browser extension

- Fixtures
  Potentially used by both server and Cypress

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
├── cypress.config.js
├── cypress_helper.rb
├── e2e
│   ├── create_post.cy.js
│   ├── edit_post.cy.js
│   └── login_logout.cy.js
├── fixtures
│   ├── beaver_angry.png
│   └── beaver_post.json
├── package.json
├── support
│   ├── commands.js
│   ├── index.js
│   └── on-rails.js
└── yarn.lock
```

Cypress on Rails

The cypress(_on_rails) Directory Structure

- All test and support files for Cypress
  Used by the Cypress app and browser extension
- Fixtures
  Potentially used by both server and Cypress
- Support files for `cypress_on_rails`
  Used/executed on the Rails server

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
├── cypress.config.js
├── cypress_helper.rb
├── e2e
│   ├── create_post.cy.js
│   ├── edit_post.cy.js
│   └── login_logout.cy.js
├── fixtures
│   ├── beaver_angry.png
│   └── beaver_post.json
├── package.json
├── support
│   ├── commands.js
│   ├── index.js
│   └── on-rails.js
└── yarn.lock
```

Cypress on Rails

App Commands

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   └── log_fail.rb
```

```ruby
# app_commands/eval.rb
Kernel.eval(command_options) unless command_options.nil?
```

```javascript
Cypress.Commands.add('app', function (name, command_options) {
  return cy.appCommands({ name, options: command_options })
})

cy.app('eval', 'User.delete_all')
cy.app('eval', 'system("rm -rf /")')
```

Cypress on Rails

App Commands

- Ruby files that can be executed in the context of the Rails server by passing their filename to the `rack` endpoint

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   └── log_fail.rb
```

```ruby
# app_commands/eval.rb
Kernel.eval(command_options) unless command_options.nil?
```

```javascript
Cypress.Commands.add('app', function (name, command_options) {
  return cy.appCommands({ name, options: command_options })
})

cy.app('eval', 'User.delete_all')
cy.app('eval', 'system("rm -rf /")')
```

Cypress on Rails

App Commands

- Ruby files that can be executed in the context of the Rails server by passing their filename to the `rack` endpoint

- The endpoint expects a `name` and an optional `options` parameter which is available inside the ruby script as `command_options`.

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   └── log_fail.rb
```

```ruby
# app_commands/eval.rb
Kernel.eval(command_options) unless command_options.nil?
```

```javascript
Cypress.Commands.add('app', function (name, command_options) {
  return cy.appCommands({ name, options: command_options })
})

cy.app('eval', 'User.delete_all')
cy.app('eval', 'system("rm -rf /")')
```

Cypress on Rails

App Commands

- Ruby files that can be executed in the context of the Rails server by passing their filename to the `rack` endpoint
- The endpoint expects a `name` and an optional `options` parameter which is available inside the ruby script as `command_options`.
- The script's return value is passed back to cypress as JSON

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   └── log_fail.rb
```

```ruby
# app_commands/eval.rb
Kernel.eval(command_options) unless command_options.nil?
```

```
Cypress.Commands.add('app', function (name, command_options) {
  return cy.appCommands({ name, options: command_options })
})

cy.app('eval', 'User.delete_all')
cy.app('eval', 'system("rm -rf /")')
```

Cypress on Rails

Scenarios

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
```

```ruby
# app_commands/scenarios/basic_account.rb
FactoryBot.create(:account, :verified)
```

```javascript
Cypress.Commands.add('appScenario', function (name, options = {}) {
  return cy.app('scenarios/' + name, options)
})

```

Cypress on Rails

Scenarios

- Just app commands in a subfolder

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
```

```ruby
# app_commands/scenarios/basic_account.rb
FactoryBot.create(:account, :verified)
```

```
Cypress.Commands.add('appScenario', function (name, options = {}) {
  return cy.app('scenarios/' + name, options)
})

```

Cypress on Rails

Scenarios

- Just app commands in a subfolder
- Contain by convention code that performs server side operations needed by many different tests

```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
```

```ruby
# app_commands/scenarios/basic_account.rb
FactoryBot.create(:account, :verified)
```

```javascript
Cypress.Commands.add('appScenario', function (name, options = {}) {
  return cy.app('scenarios/' + name, options)
})

```

Cypress on Rails

Scenarios

- Just app commands in a subfolder
- Contain by convention code that performs server side operations needed by many different tests
- Example: If your application needs a logged in user to work, its creation can be placed in a scenario.

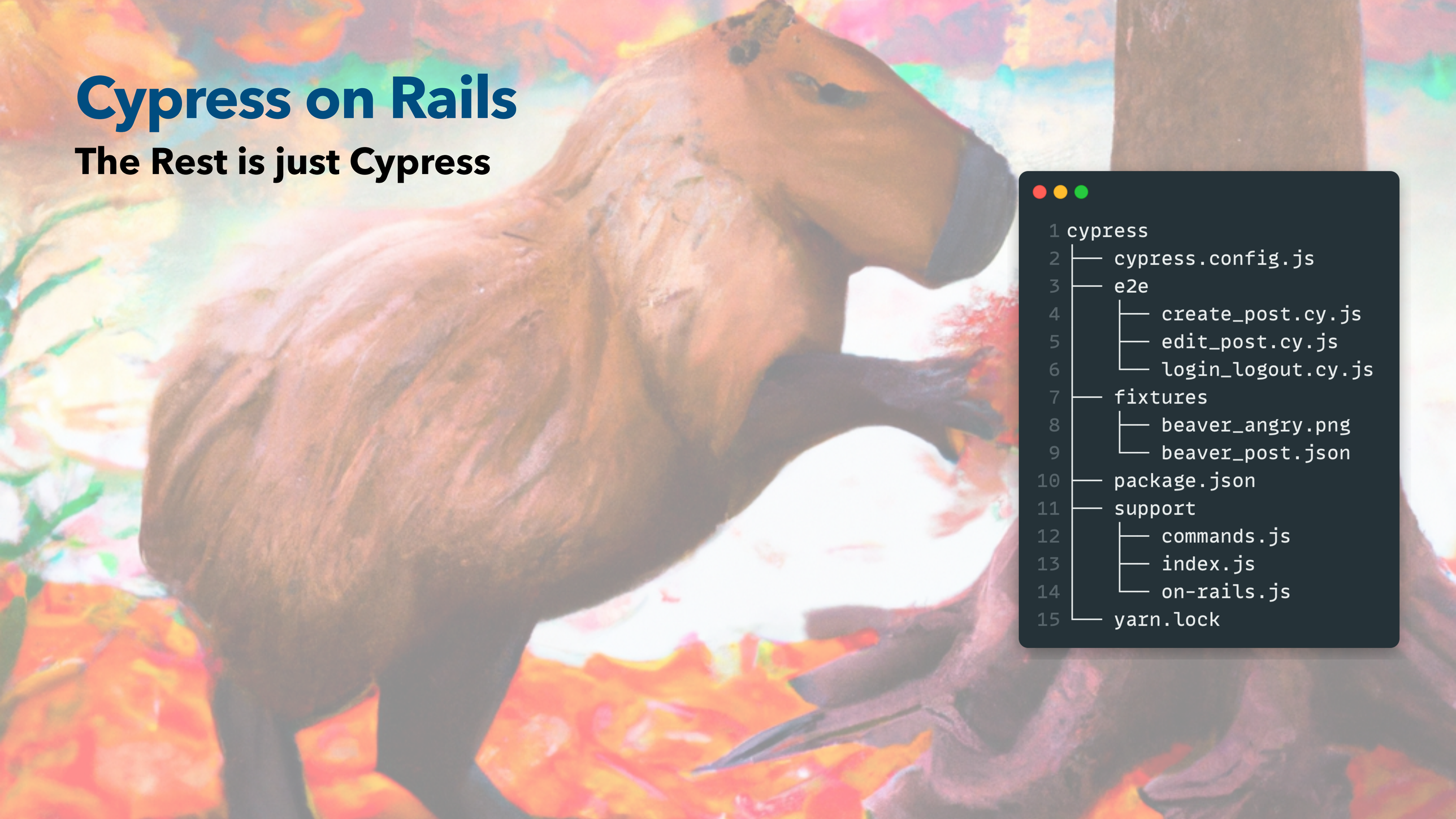
```
cypress
├── app_commands
│   ├── clean.rb
│   ├── eval.rb
│   ├── factory_bot.rb
│   ├── log_fail.rb
│   └── scenarios
│       ├── base.rb
│       └── basic_account.rb
```

```ruby
# app_commands/scenarios/basic_account.rb
FactoryBot.create(:account, :verified)
```

```javascript
Cypress.Commands.add('appScenario', function (name, options = {}) {
  return cy.app('scenarios/' + name, options)
})

```

Cypress on Rails

The Rest is just Cypress

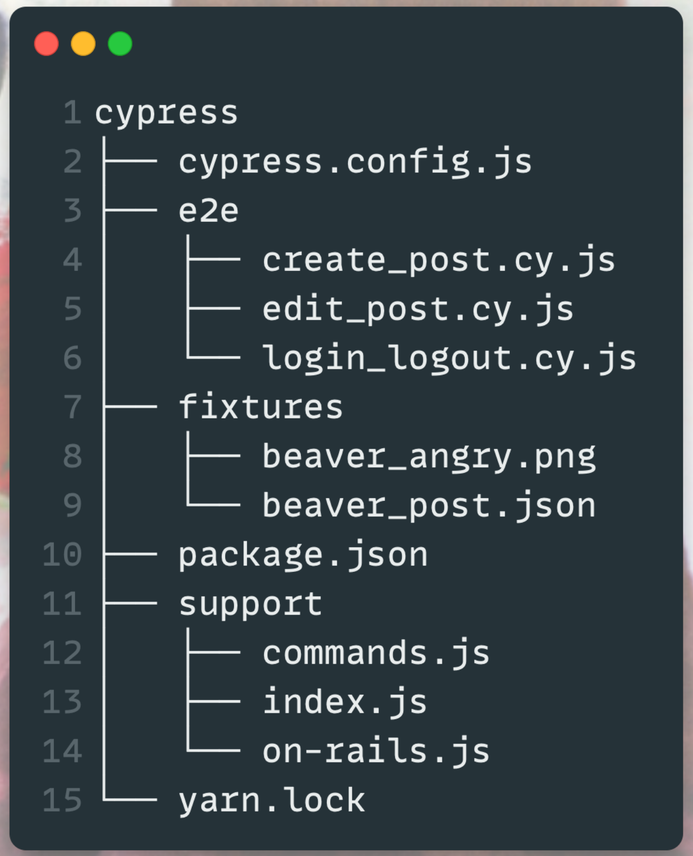


```
cypress
├── cypress.config.js
├── e2e
│   ├── create_post.cy.js
│   ├── edit_post.cy.js
│   └── login_logout.cy.js
├── fixtures
│   ├── beaver_angry.png
│   └── beaver_post.json
├── package.json
├── support
│   ├── commands.js
│   ├── index.js
│   └── on-rails.js
└── yarn.lock
```

Cypress on Rails

The Rest is just Cypress

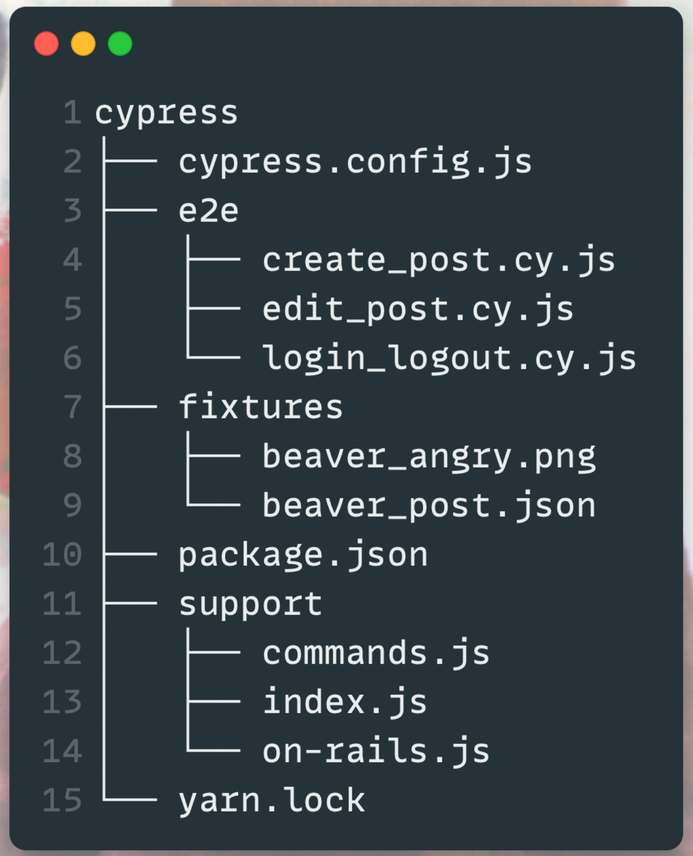
- Config contains test and fixture directories, server address and other cypress options



Cypress on Rails

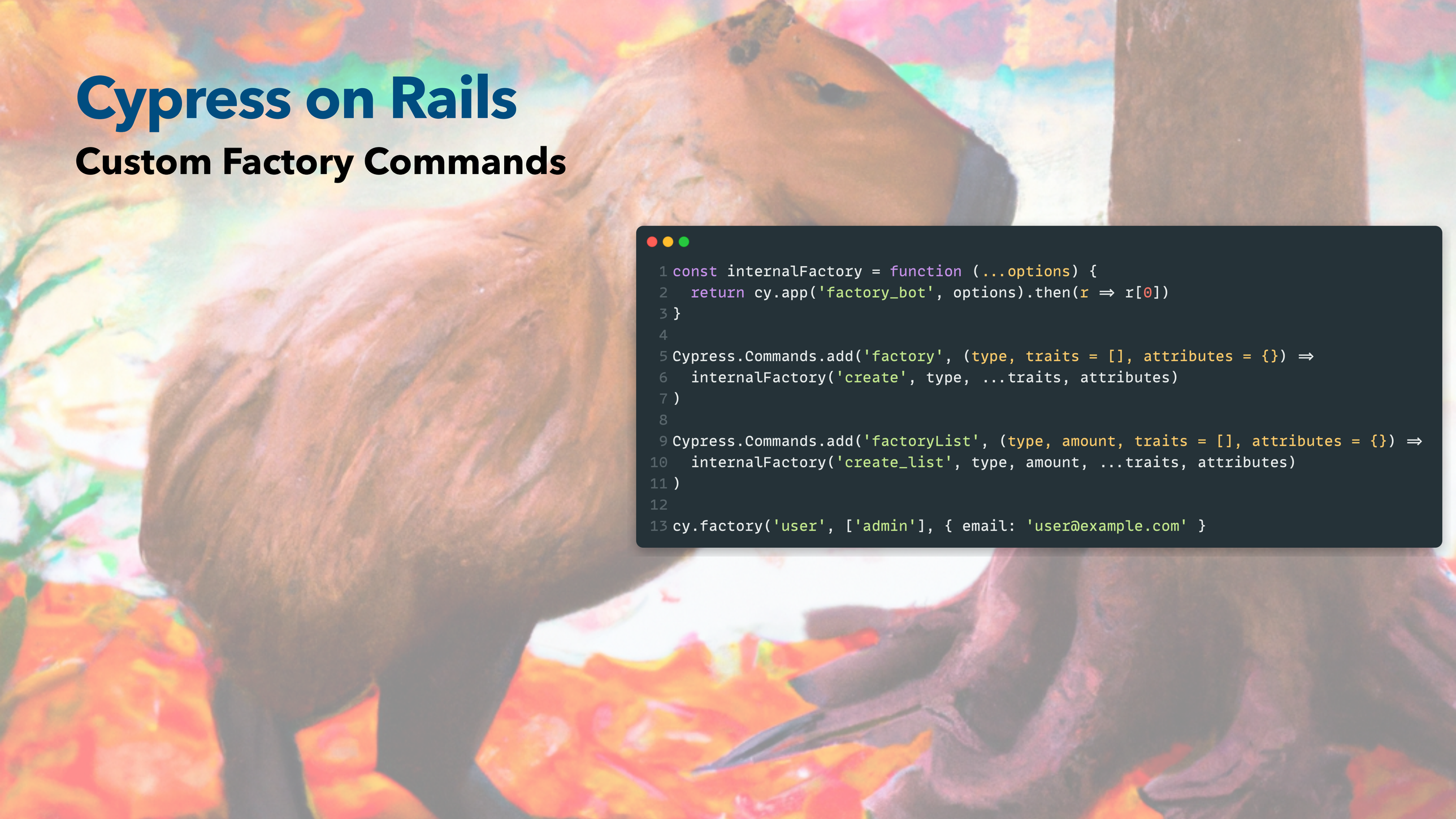
The Rest is just Cypress

- Config contains test and fixture directories, server address and other cypress options
- Support directory contains custom commands and plugin imports



Cypress on Rails

Custom Factory Commands



```
const internalFactory = function (...options) {
  return cy.app('factory_bot', options).then(r => r[0])
}

Cypress.Commands.add('factory', (type, traits = [], attributes = {}) =>
  internalFactory('create', type, ...traits, attributes)
)

Cypress.Commands.add('factoryList', (type, amount, traits = [], attributes = {}) =>
  internalFactory('create_list', type, amount, ...traits, attributes)
)

cy.factory('user', ['admin'], { email: 'user@example.com' }
```

Cypress on Rails

Custom Factory Commands

- Helper commands to access `FactoryBot.create` and `FactoryBot.create_list` through the app command

```
const internalFactory = function (...options) {
  return cy.app('factory_bot', options).then(r => r[0])
}

Cypress.Commands.add('factory', (type, traits = [], attributes = {}) =>
  internalFactory('create', type, ...traits, attributes)
)

Cypress.Commands.add('factoryList', (type, amount, traits = [], attributes = {}) =>
  internalFactory('create_list', type, amount, ...traits, attributes)
)

cy.factory('user', ['admin'], { email: 'user@example.com' }
```

Cypress on Rails

Custom Factory Commands

- Helper commands to access `FactoryBot.create` and `FactoryBot.create_list` through the app command

```
const internalFactory = function (...options) {
  return cy.app('factory_bot', options).then(r => r[0])
}

Cypress.Commands.add('factory', (type, traits = [], attributes = {}) =>
  internalFactory('create', type, ...traits, attributes)
)

Cypress.Commands.add('factoryList', (type, amount, traits = [], attributes = {}) =>
  internalFactory('create_list', type, amount, ...traits, attributes)
)

cy.factory('user', ['admin'], { email: 'user@example.com' }
```

- I decided to have traits as an own argument instead of a `...traitsAndAttributes` approach to make the command less magical.

Cypress on Rails

`factory_bot` App Command

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```
cy.factory('image', [], { resource_type: "Post", resource_id: 5 })
```

Cypress on Rails

`factory_bot` App Command

- cypress_on_rails comes with a basic app command which supports `factory_bot`

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```javascript
cy.factory('image', [], { resource_type: "Post", resource_id: 5 })
```

Cypress on Rails

`factory_bot` App Command

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```
cy.factory('image', [], { resource_type: "Post", resource_id: 5 })
```

Cypress on Rails

`factory_bot` App Command

- Repo contains an extended version

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```
cy.factory('image', [], { resource_type: "Post", resource_id: 5 })
```

Cypress on Rails

`factory_bot` App Command

- Repo contains an extended version
- It returns global IDs
  for all generated records

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```
cy.factory('image', [], { resource_type: "Post", resource_id: 5 })
```

Cypress on Rails

`factory_bot` App Command

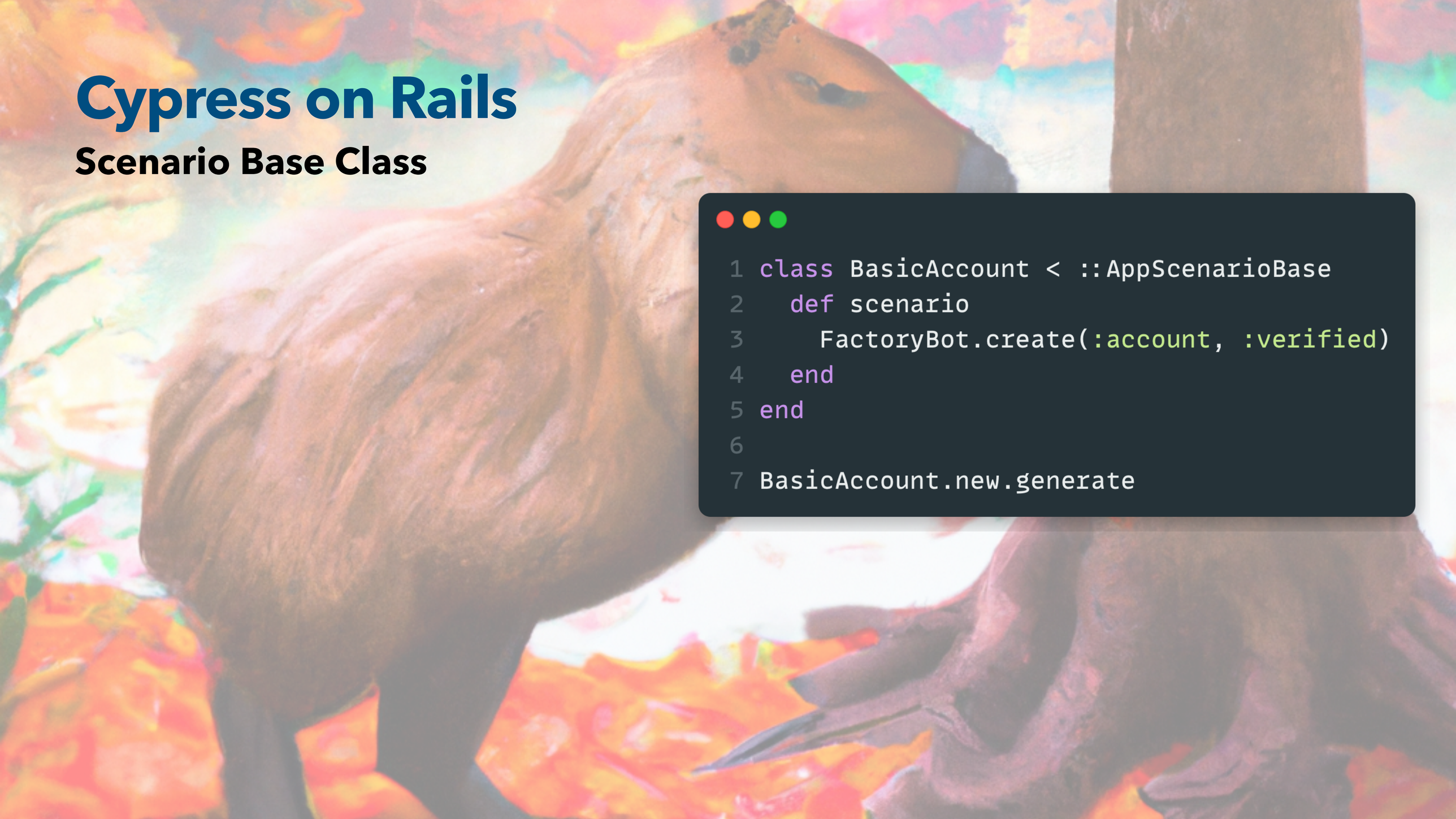
- Repo contains an extended version
- It returns global IDs
  for all generated records
- It accepts global IDs as associations,
  emulating assigning a record
  in `factory_bot` instead of setting
  attributes

```ruby
class Image < ApplicationRecord
  belongs_to :resource, polymorphic: true
  # resource_id and resource_type columns
end

class Post < ApplicationRecord
  has_many :images, as: :resource
end

class Comment < ApplicationRecord
  has_many :images, as: :resource
end

```

```
cy.factory('post').then(post ⇒ {
  cy.factory('image', [], { resource_identifier: post.gid })
})
```

Cypress on Rails

Scenario Base Class



```ruby
class BasicAccount < ::AppScenarioBase
  def scenario
    FactoryBot.create(:account, :verified)
  end
end

BasicAccount.new.generate
```

Cypress on Rails

Scenario Base Class

- The repo contains a base class for app scenarios

```ruby
class BasicAccount < ::AppScenarioBase
  def scenario
    FactoryBot.create(:account, :verified)
  end
end

BasicAccount.new.generate
```

Cypress on Rails

Scenario Base Class

- The repo contains a base class for app scenarios
- It adds global IDs to all generated records...

```ruby
class BasicAccount < ::AppScenarioBase
  def scenario
    FactoryBot.create(:account, :verified)
  end
end

BasicAccount.new.generate
```

```javascript
cy.appScenario('basic_user').then(user => {
  cy.factory('post', [], { author_identifier: user.gid })
})
```

Cypress on Rails

Scenario Base Class

- The repo contains a base class for app scenarios
- It adds global IDs to all generated records...
- ...and allows some JSON response manipulations if needed

```ruby
class BasicAccount < ::AppScenarioBase
  def scenario
    FactoryBot.create(:account, :verified)
  end
end

BasicAccount.new.generate
```

```javascript
cy.appScenario('basic_user').then(user => {
  cy.factory('post', [], { author_identifier: user.gid })
})
```


Containerisation and CI

Containerisation and CI

Tools provided by cypress

Containerisation and CI

Tools provided by cypress

- Great demo project by the cypress folks with a node server and demo CI configurations for all major providers

CI	Build status	basic config file	full parallel config
AWS Amplify Console		amplify.yml	
AWS CodeBuild		basic/buildspec.yml	buildspec.yml
AppVeyor	build passing	appveyor.yml	
Azure CI	Azure Pipelines never built	basic/azure-ci.yml	azure-ci.yml
Buddy		buddy.yml	
Buildkite		.buildkite/pipeline.yml	
Circle	FAILED	basic/.circleci /config.yml	.circleci/config.yml
Codeship Pro		basic/codeship-pro	
GitHub Actions	Cypress parallel tests passing	single.yml	parallel.yml
GitLab	pipeline failed	basic/.gitlab-ci.yml	.gitlab-ci.yml
Heroku CI		basic/app.json	
Jenkins		basic/Jenkinsfile	Jenkinsfile
Netlify	netlify Failed	netlify.toml	
Semaphore v2	Passed	basic/.semaphore.yml	.semaphore/semaphore.yml
Shippable	Shippable CI	shippable.yml	
Travis	build passing	basic/.travis.yml	.travis.yml

Containerisation and CI

Tools provided by cypress

- Great demo project by the cypress folks with a node server and demo CI configurations for all major providers

- Own Github helper action

```yaml
name: Chrome

on: push

jobs:
  chrome:
    runs-on: ubuntu-18.04
    steps:
      - name: Checkout
        uses: actions/checkout@v1
      - name: Chrome
        uses: cypress-io/github-action@v4
        timeout-minutes: 10
        with:
          build: npm run build
          start: npm start
          browser: chrome
```

Containerisation and CI

Tools provided by cypress

- Great demo project by the cypress folks with a node server and demo CI configurations for all major providers
- Own Github helper action
- Docker image with node + cypress pre-installed

```yaml
# .gitlab-ci.yml
test:
  image: cypress/base:10
  stage: test
  script:
    # start the server in the background
    # ./bin/node_server.sh
    - npm run start &
    # run Cypress
    - npm run cypress run
  artifacts:
    paths:
      - cypress/videos
      - cypress/screenshots
    expire_in: 1 day
```

Containerisation and CI

We likely need a little more complex setup

```yaml
# .gitlab-ci.yml
test:
  image: cypress/base:10
  stage: test
  script:
    # start the server in the background
    # ./bin/node_server.sh
    - npm run start &
    # run Cypress
    - npm run cypress run
  artifacts:
    paths:
      - cypress/videos
      - cypress/screenshots
    expire_in: 1 day
```

Containerisation and CI

We likely need a little more complex setup

- The demo project uses a very simple node server that's just ran as a background process

```yaml
# .gitlab-ci.yml
test:
  image: cypress/base:10
  stage: test
  script:
    # start the server in the background
    # ./bin/node_server.sh
    - npm run start &
    # run Cypress
    - npm run cypress run
  artifacts:
    paths:
      - cypress/videos
      - cypress/screenshots
    expire_in: 1 day
```

Containerisation and CI

We likely need a little more complex setup

- The demo project uses a very simple node server that's just ran as a background process

- Examples do not include additional services or dependencies for the server

```
# .gitlab-ci.yml
test:
  image: cypress/base:10
  stage: test
  script:
    # start the server in the background
    # ./bin/node_server.sh
    - npm run start &
    # run Cypress
    - npm run cypress run
  artifacts:
    paths:
      - cypress/videos
      - cypress/screenshots
    expire_in: 1 day
```

Containerisation and CI

We likely need a little more complex setup

- The demo project uses a very simple node server that's just ran as a background process
- Examples do not include additional services or dependencies for the server
- Not really how a Rails app works

```yaml
# .gitlab-ci.yml
test:
  image: cypress/base:10
  stage: test
  script:
    # start the server in the background
    # ./bin/node_server.sh
    - npm run start &
    # run Cypress
    - npm run cypress run
  artifacts:
    paths:
      - cypress/videos
      - cypress/screenshots
    expire_in: 1 day
```

Containerisation and CI

Separate Docker Approach (very opinionated)

Containerisation and CI

Separate Docker Approach (very opinionated)

- Build own docker images for your backend and cypress

Containerisation and CI

Separate Docker Approach (very opinionated)

- Build own docker images for your backend and cypress
- Add your backend as a service when running the cypress image in CI

Containerisation and CI

Separate Docker Approach (very opinionated)

- Build own docker images for your backend and cypress
- Add your backend as a service when running the cypress image in CI
- Keep a separation between your application and the cypress tests

Containerisation and CI

Adding the Backend Service to CI

```yaml
# .gitlab-ci.yml
cypress:
  image: $CI_REGISTRY_IMAGE/cypress:$CI_COMMIT_REF_SLUG
  services:
    - postgres:14
    - redis
    - name: $CI_REGISTRY_IMAGE/development:$CI_COMMIT_REF_SLUG
      alias: backend
      command: ["/app/docker/setup_cypress_server.sh"]
  variables:
    RAILS_ENV: test
    BASE_URL: "backend:4567"
  script:
    - cd /app
    - yarn run cypress run
```

Containerisation and CI

Adding the Backend Service to CI

- A similar approach with Github Actions can be found in the Repo

```yaml
# .gitlab-ci.yml
cypress:
  image: $CI_REGISTRY_IMAGE/cypress:$CI_COMMIT_REF_SLUG
  services:
    - postgres:14
    - redis
    - name: $CI_REGISTRY_IMAGE/development:$CI_COMMIT_REF_SLUG
      alias: backend
      command: ["/app/docker/setup_cypress_server.sh"]
  variables:
    RAILS_ENV: test
    BASE_URL: "backend:4567"
  script:
    - cd /app
    - yarn run cypress run
```

Containerisation and CI

Adding the Backend Service to CI

- A similar approach with Github Actions can be found in the Repo

- I had to add a lot of workarounds as Github e.g. does not allow setting custom commands for services

```yaml
# .gitlab-ci.yml
cypress:
  image: $CI_REGISTRY_IMAGE/cypress:$CI_COMMIT_REF_SLUG
  services:
    - postgres:14
    - redis
    - name: $CI_REGISTRY_IMAGE/development:$CI_COMMIT_REF_SLUG
      alias: backend
      command: ["/app/docker/setup_cypress_server.sh"]
  variables:
    RAILS_ENV: test
    BASE_URL: "backend:4567"
  script:
    - cd /app
    - yarn run cypress run
```

Containerisation and CI

Building an own Cypress Docker Image

```
# cypress/Dockerfile

FROM cypress/base:16

WORKDIR /app

# Install npm dependencies
COPY ./package.json ./yarn.lock ./
RUN npm install -g yarn
RUN yarn install

ARG BASE_URL=localhost:3000
ENV BASE_URL=$BASE_URL

COPY ./ ./
ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["yarn", "run", "cypress", "run"]
```

```
# cypress/docker-entrypoint.sh

set -e

echo "Waiting for backend at http://${BASE_URL}..."
npx wait-on -t 120000 "http-get://${BASE_URL}"

exec "$@"
```

Containerisation and CI

Building an own Cypress Docker Image

- Allows us to only include the actual cypress tests

```
# cypress/Dockerfile

FROM cypress/base:16

WORKDIR /app

# Install npm dependencies
COPY ./package.json ./yarn.lock ./
RUN npm install -g yarn
RUN yarn install

ARG BASE_URL=localhost:3000
ENV BASE_URL=$BASE_URL

COPY ./ ./
ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["yarn", "run", "cypress", "run"]
```

```
# cypress/docker-entrypoint.sh

set -e

echo "Waiting for backend at http://${BASE_URL}..."
npx wait-on -t 120000 "http-get://${BASE_URL}"

exec "$@"
```

Containerisation and CI

Building an own Cypress Docker Image

- Allows us to only include the actual cypress tests
  - Everything belonging to `cypress_on_rails` can be excluded via `.dockerignore`

```
# cypress/Dockerfile

FROM cypress/base:16

WORKDIR /app

# Install npm dependencies
COPY ./package.json ./yarn.lock ./
RUN npm install -g yarn
RUN yarn install

ARG BASE_URL=localhost:3000
ENV BASE_URL=$BASE_URL

COPY ./ ./
ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["yarn", "run", "cypress", "run"]
```

```
# cypress/docker-entrypoint.sh

set -e

echo "Waiting for backend at http://${BASE_URL}..."
npx wait-on -t 120000 "http-get://${BASE_URL}"

exec "$@"
```

Containerisation and CI

Building an own Cypress Docker Image

- Allows us to only include the actual cypress tests
  - Everything belonging to `cypress_on_rails` can be excluded via `.dockerignore`
- The cypress base URL can be set via the environment

```
# cypress/Dockerfile

FROM cypress/base:16

WORKDIR /app

# Install npm dependencies
COPY ./package.json ./yarn.lock ./
RUN npm install -g yarn
RUN yarn install

ARG BASE_URL=localhost:3000
ENV BASE_URL=$BASE_URL

COPY ./ ./
ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["yarn", "run", "cypress", "run"]
```

```
# cypress/docker-entrypoint.sh

set -e

echo "Waiting for backend at http://${BASE_URL}..."
npx wait-on -t 120000 "http-get://${BASE_URL}"

exec "$@"
```

Questions / Custom Application Requirements?


Thank you