Acceptance Testing with wallaby

Feb 4, 2019
Christian Wesselhoeft
hi@xtian.us
https://github.com/xtian/talks

What are acceptance tests?

What are acceptance tests?

- Automated test cases that exercise your whole system, written in terms of high-level user interactions
- Key term is “user interactions”
- Aside from setup and teardown, test cases should only utilize APIs that are accessible to a user

Why write acceptance tests?

Why write acceptance tests?

Automated tests create a feedback cycle

- Focus our decision making
- Validate design decisions
- Catch defects

Why write acceptance tests?

- Unit tests give great feedback on *units*, but say *nothing* about the health of the overall application

- Acceptance tests give little feedback on implementation, but are great at focusing decision making and catching defects at the boundaries of interfaces

Why write acceptance tests?

- Acceptance tests give you a place to start
- Good stories are generally easy to translate into acceptance tests

Why write acceptance tests?

Good stories: “INVEST”

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

(https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/)

Why write acceptance tests?

- Elixir compiler + tools like Dialyzer, GraphQL, and TypeScript can help catch integration defects at compile time

- Ultimately there will still be defects that will only surface at run time

- Acceptance tests help catch those bugs and protect against regressions

Why don’t we write acceptance tests?

Why don’t we write acceptance tests?

Historically, tooling has been poor

- Selenium
- Capybara-WebKit
- PhantomJS

Why don’t we write acceptance tests?

- Overreliance on manual QA
- False sense of security from unit tests
- Common mistakes make acceptance tests difficult to integrate and maintain
- They are fundamentally slower and more difficult to write and debug than unit tests
- Understand the tradeoffs as they relate to your context!

Common Mistakes

Common Mistakes

Coupling to implementation details

- Acceptance testing tools commonly offer both XPath and CSS selector APIs
- Very dangerous to rely on!
- E.g., `` `#signup form > .field:nth-child(1)` ``
- What does this mean to a user??

Common Mistakes

Avoiding coupling:

- Use input labels, button text to interact with forms
- Add accessibility text to icons
- In cases where info really is communicated visually or contextually, use a test-specific helper
- Abstract repeated selectors/actions into “page wrappers”

Common Mistakes

```elixir
defmodule AcceptanceDemoWeb.ViewHelpers do
  if Mix.env == :test do
    def tid(id), do: “data-t-#{id}"
  else
    @compile {:inline, tid: 1}
    def tid(_), do: []
  end
end
```

```elixir
defmodule AcceptanceDemo.TestHelpers do
  use Wallaby.DSL

  def test_find(session, id), do: find(session, Query.find(“[data-t-#{id}]"))
end
```

Common Mistakes

```
<h1 <%= tid :group_name %>>
  <%= link @group.name, to: group_path(@conn, :show, @group) %>
</h1>
```

```
session
|> visit(group_path(@endpoint, :new))
|> fill_in(Query.text_field("Group Name"), with: " #{group_name} ")
|> click(Query.text("Create Group"))

assert_text test_find(session, :group_name), group_name
```

Common Mistakes

```elixir
defmodule MyPage do
  use Wallaby.DSL

  @name_field Query.text_field("Name")
  @email_field Query.text_field("email")
  @save_button Query.button("Save")

  def register(session) do
    session
    |> visit("/registration.html")
    |> fill_in(@name_field, with: "Chris")
    |> fill_in(@email_field, with: "c@keathly.io")
    |> click(@save_button)
  end
end
```

Common Mistakes

Not using a monorepo

- If different components of your application live in different repositories, writing tests across them is basically intractable

- The myriad benefits of monorepos is a larger topic but check out this article: https://www.drmaciver.com/2016/10/why-you-should-use-a-single-repository-for-all-your-companys-projects/

Common Mistakes

Not getting first acceptance test in early

Obstacles creep in:

- Slow UI
- Complex system startup procedure
- Proliferation of unwrapped external dependencies

Wallaby

Wallaby

- https://github.com/keathley/wallaby
- Written by Chris Keathley of Bleacher Report
- Started in Feb 2016
- 779 stars, 23 releases, latest release Nov 2018

Using Wallaby

Setup

Setup

Install Chrome, then: `brew install chromedriver`

mix.exs

```
defp deps do
  [
    # …
    {:wallaby, "~> 0.21", only: :test}
  ]
end

defp aliases do
  [
    "test.acceptance": ["cmd ACCEPTANCE=true mix test --only acceptance"]
  ]
end
```

Setup

test/test_helper.exs

```
if System.get_env("ACCEPTANCE") == "true" do
  Application.put_env(:wallaby, :base_url, AcceptanceDemo.Endpoint.url())
  {:ok, _} = Application.ensure_all_started(:wallaby)
else
  ExUnit.configure(exclude: [acceptance: true])
end
```

Setup

lib/acceptance_demo_web/endpoint.ex

```
defmodule AcceptanceDemoWeb.Endpoint do
  use Phoenix.Endpoint, otp_app: :acceptance_demo

  if Application.get_env(:acceptance_demo, :sql_sandbox) do
    plug Phoenix.Ecto.SQL.Sandbox
  end

  # …
end
```

Setup

config/test.exs

```
config :acceptance_demo, AcceptanceDemoWeb.Endpoint,
  http: [port: 4002],
  server: System.get_env(“ACCEPTANCE") == "true",
  watchers: [
    npm: ["start", cd: Path.expand(“../assets", __DIR__)]
  ]

config :acceptance_demo, sql_sandbox: true

config :wallaby,
  driver: Wallaby.Experimental.Chrome
```

Setup

- That’s it!
- On CI you will need to have Chrome installed
- Circle has a Docker image with it pre-installed: circleci/elixir:1.8-node-browsers

Demo