CS169 Discussion 7

Cucumber & Capybara

Administrivia

- HW 6 due on Sunday, 10/20, at 11:59pm
- Iteration 0 deliverables due Friday, 10/18, at 11:59pm

Features vs. Scenarios

- In Cucumber, we have both features and scenarios.
- A feature is the new app behavior that we want to implement, typically embodied in a user story.
- Scenarios are the different ways a feature can be exercised
  - Specify and test the different behaviors, such as happy/sad paths

“Happy Path”

- This is usually the scenario where the user and app both act exactly as you’d expect it to.
- A naive implementation of BDD only tests these perfect cases

“Sad Path”

- These are scenarios where the user and app don’t act as you’d expect them to. For example…
  - The user submits a malformed input
  - The app relies on some external service, which fails
- Conceptually, you can think of these as edge/failure cases. (Perhaps consider them sad paths because the user, most unfortunately, doesn’t see what they want.)
- Robust BDD always includes sad paths for user stories

Example

Feature: user login

Scenario: user can log in with correct user/password

Scenario: user sees reset password prompt on incorrect login attempt

Scenario: user locked out after three failed login attempts

Scenario: user locked out after three failed login attempts

Given that an account with username “cs169student” and password “pg&e” exists,

And that I have unsuccessfully tried to log in to my “cs169student” account “2” times already,

When I try to log in with an incorrect password for username “cs169student”,

Then I should no longer see the login prompt,

And I should be locked out of my account.

What’s wrong with this test?

Given that an account with username “cs169student” and password “pg&e” exists,

And that I have unsuccessfully tried to log in to my “cs169student” account “2” times already,

When I try to log in with an incorrect password for username “cs169student”,

Then I should no longer see the login prompt,

And I should be locked out of my account.

Common Pitfall for Sad Path testing

- Avoid loosely testing against a negative condition
- When your test only asserts that you no longer see an element on the returned page, you're saying that ANYTHING else is okay, such as
  - Some other unexpected page, so long as it doesn't have the specified element
  - A page with no content or unexpected content
  - Errors (e.g. some rails error page)
- From before:

  Then I should no longer see the login prompt,

  And I should be locked out of my account.
- As long as the back-end is updated correctly (user is locked out) and the login prompt disappears, anything goes ==> common place for big bugs to creep in

Step Definitions

- The actual ruby code you want to be executed when the matching step is found.
- Example:

Given that an account with username “cs169student” and password “pg&e” exists

```
Given /^(?:|that )an account with username “cs169student” and password “pg&e”
exists$/ do

   User.create(:username => “cs169student”, :password => “pg&e”)

end
```

What’s wrong with this step definition?

```
Given /^(?:|that )an account with username “cs169student” and password “pg&e”
exists$/ do

    User.create(:username => “cs169student”, :password => “pg&e”)

end
```

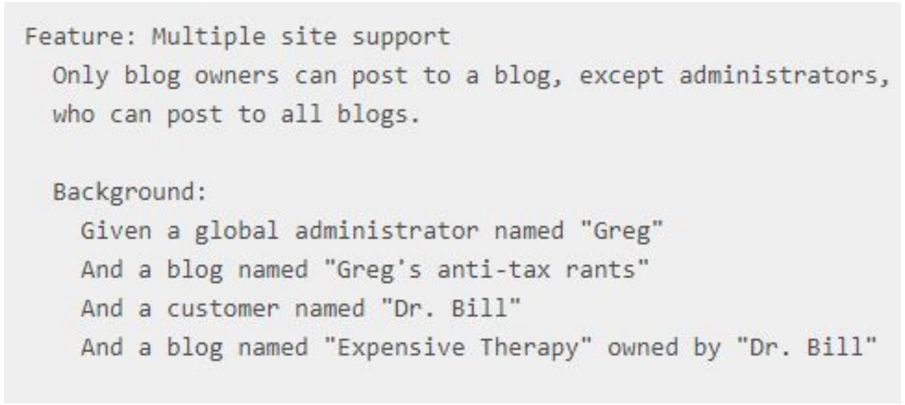
DRY Step Definitions

- Try to generalize step definitions whenever possible.
- One simple strategy is by including capture groups in the matcher and assigning these to variables
- Example:

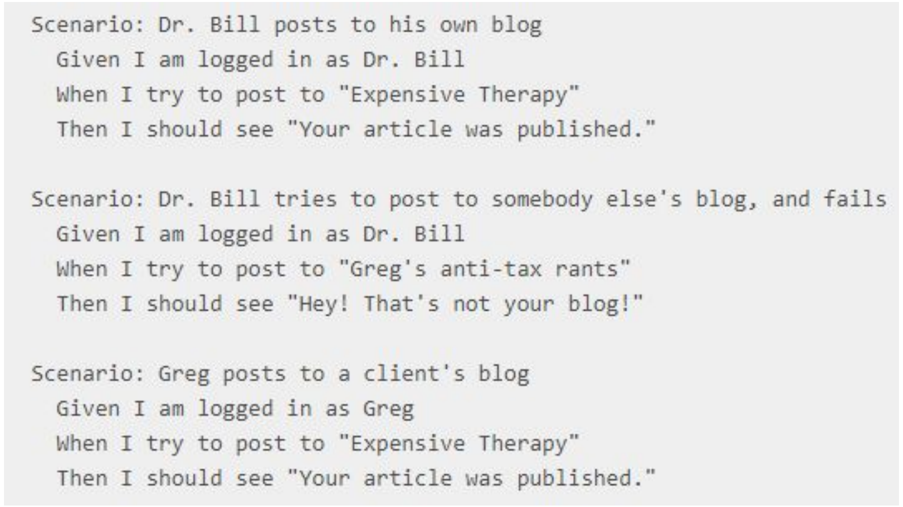
```
Given /^(?:|that )an account with username “([\S]+)” and password “([\S]+)”
exists$/ do |user|, |pass|

        User.create(:username => user, :password => pass)

end
```

Advanced Cucumber

- What if all of the scenarios for a given feature require a common subset of preconditions? Can use a Background section! Example:



```
Feature: Multiple site support
  Only blog owners can post to a blog, except administrators,
  who can post to all blogs.

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Dr. Bill"
    And a blog named "Expensive Therapy" owned by "Dr. Bill"
```

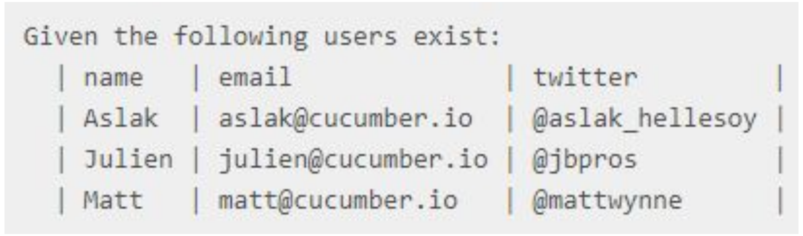


```
Scenario: Dr. Bill posts to his own blog
  Given I am logged in as Dr. Bill
  When I try to post to "Expensive Therapy"
  Then I should see "Your article was published."

Scenario: Dr. Bill tries to post to somebody else's blog, and fails
  Given I am logged in as Dr. Bill
  When I try to post to "Greg's anti-tax rants"
  Then I should see "Hey! That's not your blog!"

Scenario: Greg posts to a client's blog
  Given I am logged in as Greg
  When I try to post to "Expensive Therapy"
  Then I should see "Your article was published."
```

(Pics from Gherkin reference docs)

Advanced Cucumber

- What if you want to pass a list of values or objects to a step? Can use tables.



```
Given the following users exist:
  | name   | email              | twitter         |
  | Aslak  | aslak@cucumber.io  | @aslak_hellesoy |
  | Julien | julien@cucumber.io | @jbpros         |
  | Matt   | matt@cucumber.io   | @mattwynne      |
```

```
Given /^the following users exist:$/ do |table|
        table.hashes.each do |acct|
                User.create(. . .)
        end
end
```

Advanced Cucumber

- What if you need some value(s) to persist between steps? Can use instance variables.
- Handy for things like preserving credentials and service results (e.g. some JSON to be used by various parts of your app)

```
Given /^some step$/ do
        @state = . . .
end
...
Given /^some other step$/ do
        expect(@state).to be_valid
end
```

Advanced Cucumber

- What if you have some composite step that can be performed by executing some pre-existing step defs? Can nest steps.

```
Given /^I have unsuccessfully made two login attempts$/ do
    steps %Q{
        When I make an unsuccessful login attempt
        And I make an unsuccessful login attempt
    }
end
```

Advanced Cucumber

- Another powerful tool to use with Cucumber is xpath (although you can select elements with pure CSS, too. Read the docs)
- Can filter out parts of returned page in HTML DOM by element type and/or CSS attributes/identifiers
- Also helps mitigate the sad path pitfall by allowing for more specificity
  - Rather than testing “I do not see . . .” (anywhere on the page), you can test whether the HTML element with id [id] is present within [some known parent element]
  - Can combine this with other positive checks to better define expected behavior

```
. . .

div_to_test = page.find(:xpath, './/div[contains(@id, "bar")]')

. . .
```