

Workshop – Day 2

GOAL: DEEPEN YOUR EXPERIENCE WITH CUCUMBER AND LEARN MORE ADVANCED BDD TECHNIQUES





Regex

3 usages

- Simple Step-Definitions
- Reduce the number of step-definition lines
- Pass data to Step-definition
- Validate data



Repeat Repeat Repeat

There's a lot of duplication in our content right now.

- Story / Feature class
- Duplicate Given statements
- Nearly identical Then statements
- Steps class
- Repetitive methods

There are several tools in our tool belt for refactoring our code to deal with the repetition. We'll look at them each individually.



Background

Feature: Sunday afternoon is my favorite

Scenario: Friday is not my favorite

Given It is sunday

And It is afternoon

When I ask if this friday is your favorite

Then The answer should be "Nope"

Scenario: Sunday is my favorite

Given It is sunday

And It is afternoon

When I ask if this sunday is your favorite

Then The answer should be "Yep"





Background - Feature File

Feature: Sunday afternoon is my favorite

Scenario: Friday is not my favorite

Given It is sunday

And It is afternoon

When I ask if this friday is your favorite

Then The answer should be "Nope"

Scenario: Sunday is my favorite

Given It is sunday

And It is afternoon

When I ask if this sunday is your favorite

Then The answer should be "Yep"

See what's duplicated?

There's a better way.



Background - Feature File

Feature: Sunday afternoon is my favorite

Background:

Given It is sunday

And It is afternoon

Scenario: Friday is not my favorite

When I ask if this friday is your favorite

Then The answer should be "Nope"

Scenario: Sunday is my favorite

When I ask if this sunday is your favorite

Then The answer should be "Yep"

Ahhh...much better.





Cucumber RegEx

Cucumber does not have an alias construct, preferring RegEx (Regular Expressions) instead:

```
Given(/^a .* account$/, async function(){
```

```
Given a user has an account
Given a customer with an existing account
```



RegEx Wildcards

•	one of any character (letter, number, space, etc.)
.+	matches at least one character, but could be any quantity
.*	matches anything (or nothing) 0 or more times
[0-9]* or \d*	matches a series of digits (or nothing)
[0-9]+ or \d+	matches one or more digits
an?	matches a or an (the question mark makes the <i>n</i> optional)



Wildcard Examples – 1/3

Character matching:

Then(/^ .* found \$/, async function(){ }

This would match?

- 1) Then 2 movies should have been found
- 2) Then I found my purpose
- 3) Then check the lost and found



Wildcard Examples – 1/3

Character matching:

Then(/^ .* found \$/, async function(){ }

Correct Answer (red)

- 1) Then 2 movies should have been found
- 2) Then I found my purpose
- 3) Then check the lost and found



Wildcard Examples – 2/3

Numeric matching:

```
Then(/^\d+ movies should have been found" $/, async function(){
```

This would match?

- 1) Then 2 movies should have been found
- 2) Then 1 movie should have been found
- 3) Then two movies found
- 4) Then a movie should have been found



Wildcard Examples – 2/3

Numeric matching:

```
Then(/^\d+ movies should have been found" $/, async function(){
```

Correct Answer (red)

- 1) Then 2 movies should have been found
- 2) Then 1 movie should have been found
- 3) Then two movies found
- 4) Then a movie should have been found



Wildcard Examples – 3/3

Specify an optional character:

Then/^\d+ movies? should have been found\$/, async function(){

This would match?

- 1) Then 2 movies should have been found
- 2) Then 1 movie should have been found
- 3) Then a movie should have been found



Wildcard Examples – 3/3

```
Specify an optional character:
```

```
Then/^\d+ movies? should have been found$/, async function(){
```

Correct Answer (red)

- 1) Then 2 movies should have been found
- 2) Then 1 movie should have been found
- 3) Then a movie should have been found



Capturing Part of the Input

Parenthesis allow you to group and capture part of the input for later use (such as storing in a parameter).

Then(/^(\d+) movies? should have been found \$/, async function(variable){ }



Capturing multiple Inputs

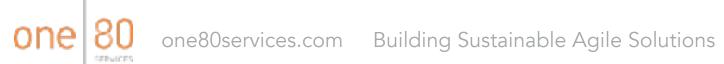
Parenthesis allow you to group and capture part of the input for later use (such as storing in a parameter).

Feature:

When I select Dog house and a Cat toy

Steps:

When(/^I have a (Dog) house and a (Cat) barn\$/, async function(variable 1, variable 2){





Grouping RegEx characters (1)

Parenthesis are useful for grouping content within the RegEx to support advanced handling.

Grouping – Supports more flexible matching of optional terms (for when a simple? Will not suffice:

Feature:

When she does accept the call

When she doesn't accept the call

Steps:

(/^she (does | doesn't) accept the call \$/, async function(){}





Non Capture Groups –

Parenthesis are used for grouping content together, capturing that data and passing it into the code.

Then(/^A book with title (.*) is found \$/, async function(variable name){

However, we may want to group content for a purpose other than passing it to the underlying code.



? Non Capture Groups

? makes the preceding element optional. This means that the pattern will match whether or not that element is present.

When(/^(Matching colo?r\$/, async function(){}

Will both "color" and "colour".



?:Non Capture Groups

?: when placed at the start of a group, makes that group non-capturing.

Scenario Steps:

When I log in as an 'Admin'

When User logs in as a 'Manager'

Annotation Matching

When(/^(I log|User logs) in as an? (.*) \$/, async function(){}

Annotation Matching

When(/^(?:I log|User logs) in as an? (.*) \$/, async function(){}

