

Practical BDD with Behat and Mink



Jeremy Mikola (@jmikola)



In order to verify application behavior
As a software developer
I need **tests**

In order to verify application behavior
As a software developer
I need **tests**

Preferably **automated tests**

Test-Driven Development

...is an iterative design process

- Write a test



Test-Driven Development

...is an iterative design process

- Write a test
- Ensure the new test **fails**



Test-Driven Development

...is an iterative design process

- Write a test
- Ensure the new test **fails**
- Write code to satisfy the test



Test-Driven Development

...is an iterative design process

- Write a test
- Ensure the new test **fails**
- Write code to satisfy the test
- Ensure all tests **pass**



Test-Driven Development

...is an iterative design process

- Write a test
- Ensure the new test **fails**
- Write code to satisfy the test
- Ensure all tests **pass**
- Refactor



Test-Driven Development

...is an iterative design process

- Write a test
- Ensure the new test **fails**
- Write code to satisfy the test
- Ensure all tests **pass**
- Refactor
- Repeat



Dan North Introduces BDD



I had a problem. While using and teaching agile practices like test-driven development (TDD) on projects in different environments, I kept coming across the same confusion and misunderstandings. Programmers wanted to know:

- Where to start
- What to test and what not to test
- How much to test in one go
- What to call their tests
- How to understand why a test fails

Dan North Introduces BDD



I started using the word “behavior” in place of “test” in my dealings with TDD and... I now had answers to some of those TDD questions:

- What to call your test is easy – it’s a sentence describing the next behavior in which you are interested.
- How much to test becomes moot – you can only describe so much behavior in a single sentence.
- When a test fails, simply work through the process described above – either you introduced a bug, the behavior moved, or the test is no longer relevant.

Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike
 - Helps relate domain language of requirements to the code



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike
 - Helps relate domain language of requirements to the code
- Do this with user stories and scenarios



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike
 - Helps relate domain language of requirements to the code
- Do this with user stories and scenarios
 - User stories describe a feature's benefit in context



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike
 - Helps relate domain language of requirements to the code
- Do this with user stories and scenarios
 - User stories describe a feature's benefit in context
 - Scenarios are executable acceptance criteria



Behavior-Driven Development

...builds upon TDD

- Write test cases in a natural language
 - Understood by developers and business folks alike
 - Helps relate domain language of requirements to the code
- Do this with user stories and scenarios
 - User stories describe a feature's benefit in context
 - Scenarios are executable acceptance criteria

“

A story's behavior is simply its acceptance criteria – if the system fulfills all the acceptance criteria, it's behaving correctly; if it doesn't, it isn't.



So what does this look like?

Example: A Contact Form

contact.feature

Feature: Contact form

In order to contact an email address

As a visitor

I need to be able to submit a contact form

Scenario: Successfully submit the contact form

Given I am on "/demo/contact"

When I fill in "Email" with "user@example.com"

And I fill in "Message" with "Hello there!"

And I press "Send"

Then I should see "Message sent!"

Example: A Contact Form

contact.feature

Feature: Contact form

Benefit In order to contact an email address

Role As a visitor

Feature I need to be able to submit a contact form

Scenario: Successfully submit the contact form

Context Given I am on "/demo/contact"

Events {
When I fill in "Email" with "user@example.com"
And I fill in "Message" with "Hello there!"
And I press "Send"

Outcome Then I should see "Message sent!"

This is where Behat and Mink come in.

Acceptance testing (any tests)

Tests a feature by executing its scenarios' steps in a context.

This is where Behat and Mink come in.

Web acceptance testing (functional tests)

Drivers for Goutte, Sahi and Symfony2's test client.

Initialize Our Bundle With Behat

```
$ app/console behat --init @AcmeDemoBundle
+d src/Acme/DemoBundle/Features
  - place your *.feature files here
+f src/Acme/DemoBundle/Features/Context/FeatureContext.php
  - place your feature related code here
```

- We now have a directory to hold AcmeDemoBundle's features
- Behat also creates an empty FeatureContext class, which extends BehatBundle's BehatContext
 - Features describe our behavior, but the context tells Behat how to evaluate our feature as an executable test

Let's Have Behat Analyze Our Feature

```
$ app/console behat src/Acme/DemoBundle/Features/contact.feature
```

```
Feature: Contact form
```

```
    In order to contact an email address
```

```
    As a visitor
```

```
    I need to be able to submit a contact form
```

```
Scenario: Successfully submit the contact form # contact.feature:6
```

```
    Given I am on "/demo/contact"
```

```
    When I fill in "Email" with "user@example.com"
```

```
    And I fill in "Message" with "Hello there!"
```

```
    And I press "Send"
```

```
    Then I should see "Message sent!"
```

```
1 scenario (1 undefined)
```

```
5 steps (5 undefined)
```

Behat Creates the Glue

...but the rest is up to you

You can implement step definitions for undefined steps with these snippets:

```
/**
 * @Given /^I am on "([^"]*)"$/
 */
public function iAmOn($argument1)
{
    throw new PendingException();
}
```

```
/**
 * @When /^I fill in "([^"]*)" with "([^"]*)"$/
 */
public function iFillInWith($argument1, $argument2)
{
    throw new PendingException();
}
```

```
/**
 * @Given /^I press "([^"]*)"$/
 */
public function iPress($argument1)
{
    throw new PendingException();
}

/**
 * @Then /^I should see "([^"]*)"$/
 */
public function iShouldSee($argument1)
{
    throw new PendingException();
}
```

Not so fast. What about Mink?

MinkContext Defines Steps

...for making requests

Pattern	Description
Given /^I am on "(?P<page>[^\"]+)"/	Opens specified page
When /^I go to "(?P<page>[^\"]+)"/	Opens specified page
When /^I reload the page\$/	Reloads current page
When /^I move backward one page\$/	Moves backward one page in history
When /^I move forward one page\$/	Moves forward one page in history
When /^I press "(?P<button>(?:[^\"] \\"))*"/	Presses button with specified id name title alt value
When /^I follow "(?P<link>(?:[^\"] \\"))*"/	Clicks link with specified id title alt text

MinkContext Defines Steps

...for interacting with forms

Pattern	Description
When /^I fill in "(?P<field>(?:[^\] \\")*)" with "(?P<value>(?:[^\] \\")*)"\$/	Fills in form field with specified id name label value
When /^I fill in "(?P<value>(?:[^\] \\")*)" for "(?P<field>(?:[^\] \\")*)"\$/	Fills in form field with specified id name label value
When /^I fill in the following:\$/	Fills in form fields with provided table
When /^I select "(?P<option>(?:[^\] \\")*)" from "(?P<select>(?:[^\] \\")*)"\$/	Selects option in select field with specified id name label value
When /^I check "(?P<option>(?:[^\] \\")*)"\$/	Checks checkbox with specified id name label value
When /^I uncheck "(?P<option>(?:[^\] \\")*)"\$/	Unchecks checkbox with specified id name label value
When /^I attach the file "(?P<path>[^\]*)" to "(?P<field>(?:[^\] \\")*)"\$/	Attaches file to field with specified id name label value

MinkContext Defines Steps

...for interacting with forms

Pattern	Description
When /^I fill in "(?P<field>(?:[^\] \\")*)" with "(?P<value>(?:[^\] \\")*)"\$/	Fills in form field with specified id name label value
When /^I fill in "(?P<value>(?:[^\] \\")*)" for "(?P<field>(?:[^\] \\")*)"\$/	Fills in form field with specified id name label value
When /^I fill in the following:\$/	Fills in form fields with provided table
When /^I select "(?P<option>(?:[^\] \\")*)" from "(?P<select>(?:[^\] \\")*)"\$/	Selects option in select field with specified id name label value
When /^I check "(?P<option>(?:[^\] \\")*)"\$/	Checks checkbox with specified id name label value
When /^I uncheck "(?P<option>(?:[^\] \\")*)"\$/	Unchecks checkbox with specified id name label value
When /^I attach the file "(?P<path>[^\]*)" to "(?P<field>(?:[^\] \\")*)"\$/	Attaches file to field with specified id name label value

What's missing here?

Gherkin, the DSL Behat uses to define behaviors, specifies two multi-line argument types: tables and pystrings

<http://docs.behat.org/guides/1.gherkin.html#multiline-arguments>

```
When I fill in the following:
```

```
| email      | user@example.com |
| message    | Hello There!     |
```

```
Given lorem ipsum:
```

```
"""
```

```
This can be a multi-line string
```

```
"""
```

MinkContext Defines Steps

...for querying the DOM

Pattern	Description
Then /^I should see "(?P<text>(?:[^\"] \\")*)" in the "(?P<element>[^\"]*)" element\$/	Checks that element with specified CSS contains specified text
Then /^the "(?P<element>[^\"]*)" element should contain "(?P<value>(?:[^\"] \\")*)"\$/	Checks that element with specified CSS contains specified HTML
Then /^I should see an? "(?P<element>[^\"]*)" element\$/	Checks that element with specified CSS exists on page
Then /^I should not see an? "(?P<element>[^\"]*)" element\$/	Checks that element with specified CSS doesn't exist on page
Then /^the "(?P<field>(?:[^\"] \\")*)" field should contain "(?P<value>(?:[^\"] \\")*)"\$/	Checks that form field with specified id name label value has specified value
Then /^the "(?P<field>(?:[^\"] \\")*)" field should not contain "(?P<value>(?:[^\"] \\")*)"\$/	Checks that form field with specified id name label value doesn't have specified value
Then /^the "(?P<checkbox>(?:[^\"] \\")*)" checkbox should be checked\$/	Checks that checkbox with specified id name label value is checked
Then /^the "(?P<checkbox>(?:[^\"] \\")*)" checkbox should not be checked\$/	Checks that checkbox with specified id name label value is unchecked

MinkContext Defines Steps

...for examining responses

Pattern	Description
Then /^I should be on "(?P<page>[^\"]+)"\$/	Checks that current page path is equal to specified
Then /^the url should match "(?P<pattern>(?:[^\"] \\"))*"\$/\$/	Checks that current page path matches pattern
Then /^the response status code should be (?P<code>\d+)\$/\$/	Checks that current page response status is equal to specified
Then /^I should see "(?P<text>(?:[^\"] \\"))*"\$/\$/	Checks that page contains specified text
Then /^I should not see "(?P<text>(?:[^\"] \\"))*"\$/\$/	Checks that page doesn't contain specified text
Then /^the response should contain "(?P<text>(?:[^\"] \\"))*"\$/\$/	Checks that HTML response contains specified string
Then /^the response should not contain "(?P<text>(?:[^\"] \\"))*"\$/\$/	Checks that HTML response doesn't contain specified string
Then /^print last response\$/	Prints last response to console
Then /^show last response\$/	Opens last response content in browser

Take Advantage of MinkContext

...for features that require web acceptance testing

```
<?php

namespace Acme\DemoBundle\Features\Context;

use Behat\BehatBundle\Context\BehatContext;

/**
 * Feature context.
 */
class FeatureContext extends BehatContext
{
}
```

Take Advantage of MinkContext

...for features that require web acceptance testing

```
<?php

namespace Acme\DemoBundle\Features\Context;

use Behat\BehatBundle\Context\MinkContext;

/**
 * Feature context.
 */
class FeatureContext extends MinkContext
{
}
```

Let's Execute Our Feature With Behat

```
$ app/console behat @AcmeDemoBundle --env=test
Feature: Contact form
  In order to contact an email address
  As a visitor
  I need to be able to submit a contact form

Scenario: Successfully submit the contact form # contact.feature:6
  Given I am on "/demo/contact" # FeatureContext::visit()
  When I fill in "Email" with "user@example.com" # FeatureContext::fillField()
    Form field with id|name|label|value "Email" not found
  And I fill in "Message" with "Hello there!" # FeatureContext::fillField()
  And I press "Send" # FeatureContext::pressButton()
  Then I should see "Message sent!" # FeatureContext::assertPageContainsText()

1 scenario (1 failed)
5 steps (1 passed, 3 skipped, 1 failed)
```

Let's Execute Our Feature With Behat

```
$ app/console behat @AcmeDemoBundle --env=test
```

```
Feature: Contact form
```

```
    In order to contact an email address
```

```
    As a visitor
```

```
    I need to be able to submit a contact form
```

```
Scenario: Successfully submit the contact form # contact.feature:6
```

```
    Given I am on "/demo/contact" # FeatureContext::visit()
```

```
    When I fill in "Email" with "user@example.com" # FeatureContext::fillField()
```

```
        Form field with id|name|label|value "Email" not found
```

```
    And I fill in "Message" with "Hello there!" # FeatureContext::fillField()
```

```
    And I press "Send" # FeatureContext::pressButton()
```

```
    Then I should see "Message sent!" # FeatureContext::assertPageContainsText()
```

```
1 scenario (1 failed)
```

```
5 steps (1 passed, 3 skipped, 1 failed)
```

Of course it fails. We haven't written any code yet!

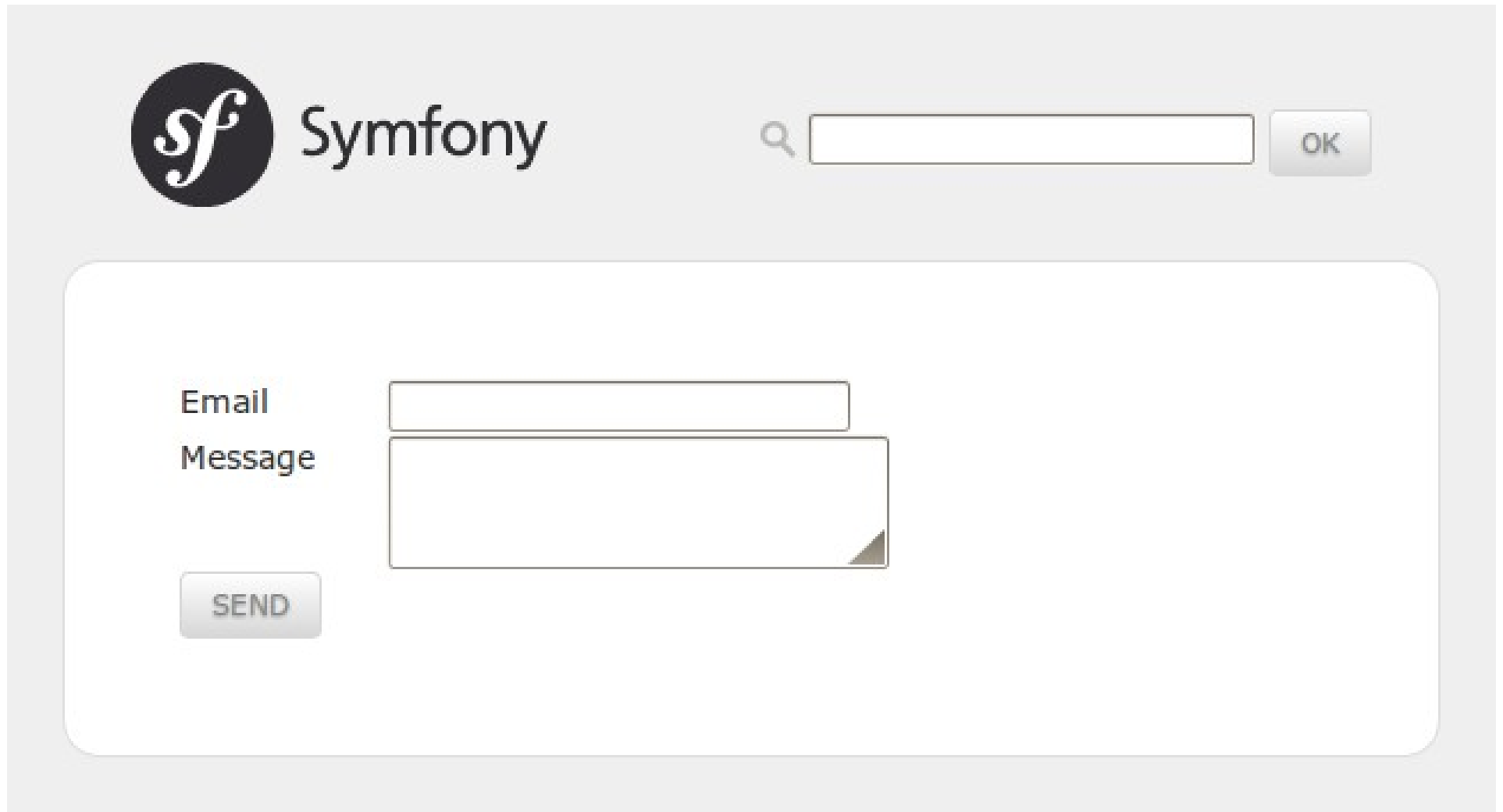
This is the **red** step of TDD.

A Note About Step Results

...of which there are seven

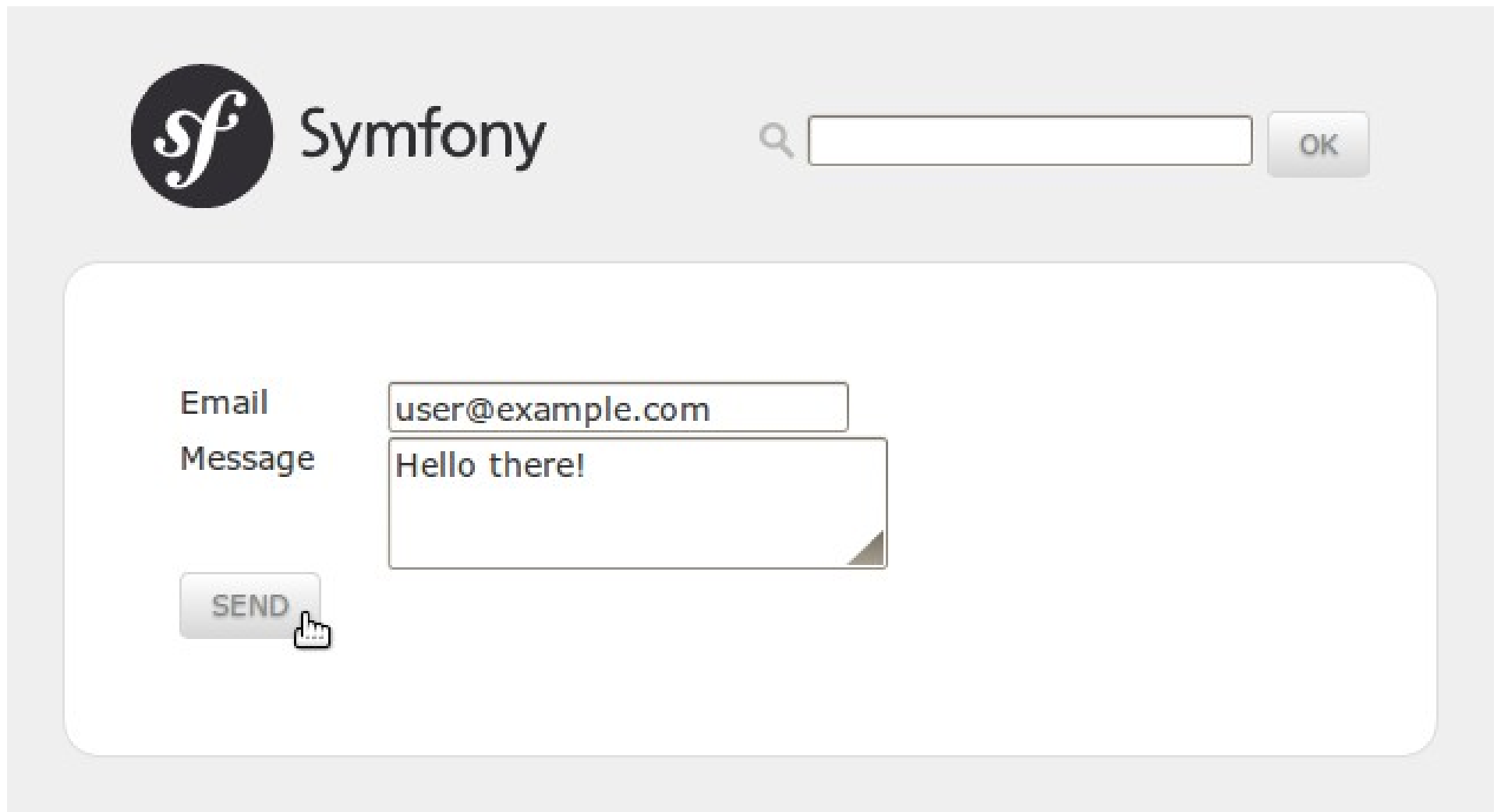
- **Success:** a definition was found and executing it did not throw an Exception
- **Undefined:** a definition couldn't be found; all subsequent steps will be skipped
- **Pending:** the definition threw the special PendingException, which means you have work to do; skip remaining steps
- **Failure:** a definition throws an Exception; Behat will skip remaining steps and terminate with exit status 1
 - By default, Behat relies on PHPUnit for assertions, but you can roll your own.
- **Skipped:** steps which were never executed
- **Ambiguous:** multiple definitions matched a step
- **Redundant:** multiple definitions share the same pattern

Implement the Contact Form



The image shows a contact form on the Symfony website. At the top left is the Symfony logo, a black circle with a white 'sf' monogram, followed by the word 'Symfony' in a sans-serif font. To the right is a search bar with a magnifying glass icon, an empty text input field, and an 'OK' button. Below this is a large white rounded rectangle containing the contact form. On the left side of this rectangle, the text 'Email' and 'Message' are stacked vertically. To the right of 'Email' is a single-line text input field. To the right of 'Message' is a larger multi-line text area with a small dark triangle in the bottom right corner. Below the 'Message' label is a 'SEND' button.

Implement the Contact Form



The image shows a web interface for a contact form. At the top left is the Symfony logo, a black circle with a white 'sf' monogram, followed by the text 'Symfony'. To the right is a search bar with a magnifying glass icon and an 'OK' button. Below this is a large rounded rectangle containing the contact form. The form has two labels: 'Email' and 'Message'. The 'Email' input field contains 'user@example.com'. The 'Message' input field contains 'Hello there!'. Below the 'Email' label is a 'SEND' button with a mouse cursor pointing at it.

Email

Message

Implement the Contact Form



Symfony



OK

Notice: Message sent!

[Hello World >](#)

[Access the secured area >](#) [Go to the login page >](#)

Let's Try That Again

```
$ app/console behat @AcmeDemoBundle --env=test
```

```
Feature: Contact form
```

```
    In order to contact an email address
```

```
    As a visitor
```

```
    I need to be able to submit a contact form
```

```
Scenario: Successfully submit the contact form # contact.feature:6
    Given I am on "/demo/contact" # FeatureContext::visit()
    When I fill in "Email" with "user@example.com" # FeatureContext::fillField()
    And I fill in "Message" with "Hello there!" # FeatureContext::fillField()
    And I press "Send" # FeatureContext::pressButton()
    Then I should see "Message sent!" # FeatureContext::assertPageContainsText()
```

```
1 scenario (1 passed)
```

```
5 steps (5 passed)
```

Let's Try That Again

```
$ app/console behat @AcmeDemoBundle --env=test
```

```
Feature: Contact form
```

```
  In order to contact an email address
```

```
  As a visitor
```

```
  I need to be able to submit a contact form
```

```
Scenario: Successfully submit the contact form # contact feature:6
```

```
  Given I am on "/demo/contact" # FeatureContext::visit()
```

```
  When I fill in "Email" with "user@example.com" # FeatureContext::fillField()
```

```
  And I fill in "Message" with "Hello there!" # FeatureContext::fillField()
```

```
  And I press "Send" # FeatureContext::pressButton()
```

```
  Then I should see "Message sent!" # FeatureContext::assertPageContainsText()
```

```
1 scenario (1 passed)
```

```
5 steps (5 passed)
```

Great! Our tests pass.

This is the **green** step of TDD.

What Else Can Mink Do?

- Provide a single API for browser behavior
 - HTTP authentication, cookies, headers, sessions
 - Page examination via XPath or CSS selectors
 - Page manipulation (e.g. complete forms, click, hover, drag-and-drop)
- Existing drivers can be used interchangeably
 - Symfony2 test client – simulated request serving
 - Goutte – headless, PHP web scraper
 - Sahi – browser-control toolkit (necessary for JS)
- Possible future drivers
 - Selenium – another browser-control toolkit
 - PhantomJS – headless WebKit with JS support!

Thanks!

<http://behat.org/>

<http://mink.behat.org/>

<http://github.com/Behat>