Model Specs - Example 1

Example 1

Lets write a Model Spec.

Create a new Rails app with RSpec

Create a new rails application (name it to whatever you prefer) and install rspec using the installation instructions we outlined earlier.

Generate a model and migrate your database

Terminal

```
$ rails generate model User first_name:string last_name:string email:string
$ rake db:migrate
```

Notice that the *"/spec"* folder has been updated. RSpec created a user_spec.rb file for you when you created the *User* model.

If you are running a Rails version older than v4.1.0, then always make sure to test your database by running `rake db:test:prepare` after your migration -- otherwise you may get errors.

Run our first spec test

Terminal

```
$ rspec spec
```

After running the tests you should see an ouput that looks like this

Terminal Output

```
Pending: (Failures listed here are expected and do not affect your suite's status)

  1) User add some examples to (or delete) /Users/.../spec/models/user_spec.rb
     # Not yet implemented
     # ./spec/models/user_spec.rb:4


Finished in 0.00097 seconds (files took 4.76 seconds to load)
1 example, 0 failures, 1 pending
```

That's it, you have run your first rails spec test!

Understanding our first spec test

Lets take a look at what we just did.

The command rspec spec runs all of the spec tests in our rails app. Later in the course we will go over how to run just the spec tests in a specific files, groupings of tests within a file, as well as just a single individual test.

Open the user_spec.rb file. It should look like this

user_spec.rb

```
require 'rails_helper'

RSpec.describe User, type: :model do
  pending "add some examples to (or delete) #{__FILE__}"
end
```

`require 'rails_helper'` is something you should have at the top of each of your spec files. It sets up your testing environment. It tells rails to import the *"/spec/rails_helper.rb"* file when running your rspec test.

`RSpec.describe ... do` is a grouping that tells rails thats you are going to be describing a set of tests that you want to run, which will be included within the `RSpec.describe` block

`User` tells us that the set of tests in the `RSpec.describe ... do` block relate to testing the User class. This is a developer designated description though and it could say pikachu if you wanted (even if you weren't testing a Pikachu class).

`type: :model` tells rails that the tests are model specs (as opposed to say a feature or controller spec). This is important because depending on the type of spec test, sometimes different methods are available.

`pending "add some examples ..."` is the test itself. The part in quotations is a developer designated

description of the test, again it could say "pikachu" if you wanted. `pending` tells rails that the test is currently marked as pending, meaning you are not done with writing the test yet, and it should neither be considered a passing or a failing test, and that rails can skip over it when running your spec tests.

Understanding our Terminal Output

Now lets look at the terminal output.

Terminal Output (referenced from above):

```
Pending: (Failures listed here are expected and do not affect your suite's status)

  1) User add some examples to (or delete) /Users/.../spec/models/user_spec.rb
     # Not yet implemented
     # ./spec/models/user_spec.rb:4


Finished in 0.00097 seconds (files took 4.76 seconds to load)
1 example, 0 failures, 1 pending
```

The terminal output starts with `Pending: (Failures listed here are expected and do not affect your suite's status)` This means that the following tests listed after the `Pending` header are all pending tests that rails is skipping over. We subsequently see a reference to the pending test that we ran listed as `1) User add some examples...`

`1 example, 0 failures, 1 pending` means that we ran a total of 1 test(s), 1 of which was a pending test.

Adding a Failing Test

Let's simulate a failing test.
Add the following code to the user_spec.rb file.

app/spec/models/user_spec.rb

```
require 'rails_helper'
RSpec.describe User do
  it "should not save if first_name field is blank" do
    user = User.new(first_name: '')
    expect(user).to be_invalid
  end

  it "should not save if last_name field is blank"

  it "should not save if email already exists"

  it "should contain a valid email"
end
```

Do not modify our User model yet. We want to make sure that whenever a user submits a blank first_name field, it should not be saved to the User model. Modify *"user_spec.rb"* file to:

Run the tests.

Test result should be something like the screenshot below:

```
D:\ror\rspec_model>rspec spec/models/user_spec.rb
*F*

Pending:
  User must be a valid email
    # Not yet implemented
    # ./spec/models/user_spec.rb:11
  User should not save if email already exists
    # Not yet implemented
    # ./spec/models/user_spec.rb:4

Failures:

  1) User should not save if first_name field is blank
     Failure/Error: expect(user).to be_invalid
       expected invalid? to return true, got false
     # ./spec/models/user_spec.rb:9:in `block (2 levels) in <top (required)>'

Finished in 0.065 seconds
3 examples, 1 failure, 2 pending

Failed examples:

rspec ./spec/models/user_spec.rb:5 # User should not save if first_name field is
 blank

Randomized with seed 53163
```

The reason why it failed is because we expected it to `be_invalid` (`be_invalid` is a RSpec method which runs all the validations within the specified object. Returns true if there are errors found, false otherwise), but it is `valid` ! That means our user model can insert a user's `first_name` even if we submitted a blank first_name value which you can see on line 4: `first_name: ''` .

Passing our test

Now lets make our test pass. We need to fix our code so that a user object cannot be saved if the first_name field is blank. Put another way:

How can we *not* allow the User model to save anything if first_name field is blank?

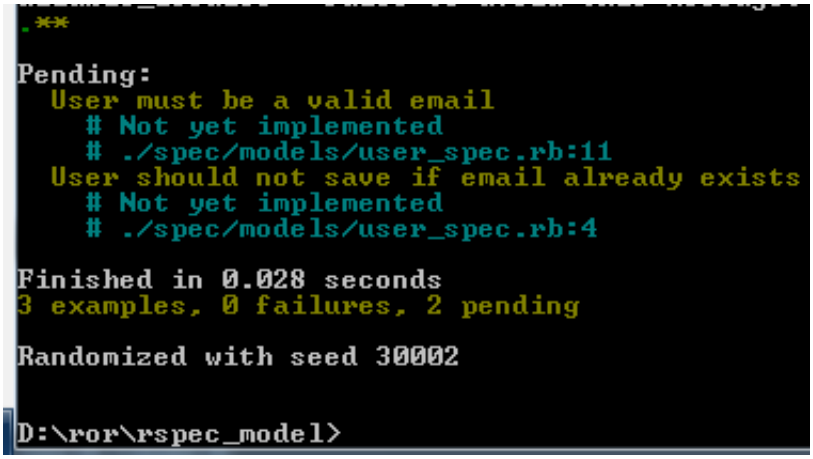
We can do this by adding a validation to our User model.
Open the *"/app/models/user.rb"* file and add this code

app/models/user.rb

```
class User < ActiveRecord::Base
  validates :first_name, :presence => true
end
```

Run the tests again.

Your test result should now be *passed:*



We can now take a deep breath and relax. We now know for sure that, whenever a user tries to submit a blank first_name, it will never save it to our User model / Database!