# Web Development with Ruby on Rails

Unit 6: Validations and Testing

- Try creating a pledge without an amount
- Win? Fail?

# Validations

- Put constraints on data
- Code in the model
- Lots of built-ins
- Customizable

```
class Pledge
  validates :amount, presence: true
end
```

# Rails built-ins

*http://guides.rubyonrails.org/active_record_validations_callbacks.html*

- `presence`
- `length`
- `uniqueness`
- others

# Caveat

- Not stored down to the DB
- Mainly an issue for uniqueness
- `add_index :users, :email, :unique => true`

# Handling errors

- Models have a `#valid?` method
- Returns false and sets errors if invalid
- Use `#errors[:field_name]` to see them

```
pl = Pledge.create(issue_name: "A bug")
pl.valid?
 => false
pl.errors[:amount]
 => ["can't be blank"]
```

# Scaffolded form shows errors

```
<% if @pledge.errors.any? %>
  <div id="error_explanation">
    ...
    <% @pledge.errors.full_messages.ec
         do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
<% end %>
```

# And the form builder marks bad fields

```
<div class="field_with_errors">
  <input id="pledge_amount" ... />
</div>
```

Let's test that

# RSpec

Behavior Driven Development

# Installation

*Gemfile*

```
group :development, :test do
  gem 'rspec-rails'
end
```

Then bundle

*Terminal*

```
bundle install
```

# Generate files

*Terminal*

```
rails g rspec:install
```

# Configuration

This avoids generating view, controller and helper specs

*config/application.rb*

```
config.generators do |g|
  g.controller_specs false
  g.view_specs false
  g.helper_specs false
end
```

# Example spec files

*Terminal*

```
rails g scaffold pledge -s
```

The `-s` means "skip existing files"

*spec/models/pledge_spec.rb*

```
require 'spec_helper'

describe Pledge do
  pending "add some examples"
end
```

# Running

*Terminal*

```
rspec spec/models/pledge_spec.rb
```

# The BDD Cycle

1. Write a failing spec
2. Write code to make it pass
3. Repeat

# First, break it

```
class Pledge
  belongs_to :user
  # validates :amount, presence: true
end
```

# Write the failing spec

```
it "requires an amount" do
  subject.should_not be_valid
  # Or: subject.valid?.should == false

  subject.amount = 3.50

  subject.should be_valid
  # Or: subject.valid?.should == true
end
```

# Run it

```
rake db:test:prepare
rspec spec/models/pledge_spec.rb
```

# Debugging

*Gemfile*

```
group :development, :test do
  gem 'debugger'
end
```

then bundle

*Terminal*

```
bundle
```

Insert the debugger
before the failing line

```
it "requires an amount" do
  debugger;1
  subject.should_not be_valid
  subject.amount = 3.50
  subject.should be_valid
end
```

*the `;1` keeps the debugger on that line*

# The Debugger

*guides.rubyonrails.org/debugging_rails_applications.html#debugging-with-ruby-debug*

- `p expression` - evaluate an expression
- `l=` - show the code around the current line
- `n` - run to next line
- `c` - continue running

# Make it pass

*app/models/pledge.rb*

```
class Pledge
  belongs_to :user
  validates :amount, presence: true
end
```

# To run all tests

*Terminal*

```
rake

# This is common to most ruby projects
```

# Basic matching

```
expected.should == actual
# tests expected == actual

expected.should_not == actual
#  tests !(expected == actual)
```

# Method matching

```
expected.should be_foo
# tests expected.foo?

expected.should have_foo
# tests expected.has_foo?

expected.should have_foo :bar
# tests expected.has_foo?(:bar)
```

</unit-testing>

# <integration-testing>

# The tool: Capybara

- A DSL for driving web browsers
- RackTest
- Selenium (Firefox)
- Webkit (via Qt)
- and others

# Install it

*Gemfile*

```
group :test do
  gem 'capybara'
end
```

Then bundle

*Terminal*

```
bundle
```

# An integration test
# aka., Request spec

- Describes a user flow
- Runs in a "browser"
- Tests page content

# An example test

*spec/requests/pledge_spec.rb*

```
visit pledges_path
click_link "New Pledge"
fill_in "Issue title", with: "A bug"
click_button "Create Pledge"

error_message = "Amount can't be blank"
page.should have_content(error_message
```

# Run it just like a regular test

```
rspec spec/requests/pledge_spec.rb
```

# Standard Capybara matchers

```
click_link
click_button
click_on
fill_in "field", with: "value"
select "field", from: "value"
page.has_content? "Some text"
```

See http://cheat.errtheblog.com/s/capybara/

# Break it

```
class Pledge
  belongs_to :user
  # validates :amount, presence: true
end
```

# Debug it

```
error_message = "Amount can't be blank"
debugger;1
page.should have_content?(error_messag
```

# Put it in a browser

```
describe "Pledges" do
  describe "making a pledge" do
    it "requires an amount", js: true do
      ...
```

# Fix it

```
class Pledge
  belongs_to :user
  validates :amount, presence: true
end
```

# Test user selection

```
it "can include a user" do
  visit pledges_path
  click_link "New Pledge"
  fill_in "Amount", with: "3.50"
  select "Mat", from: "User"
  click_on "Create Pledge"

  page.should have_content "successful"
end
```

# Extract shared code

```
before do
  visit pledges_path
  click_link "New Pledge"
end
```

# No users? Fixtures!

- Set up your test data
- Fast, but skip validations
- Limited association power

*spec/fixtures/users.yml*

Maybe move it from test/fixtures

```
mat:
  name: Mat
  email: mat@schaffer.me
```

*spec/requests/pledge_spec.rb*

```
fixtures :users
```

</integration-testing>

# Next Topic: Plugins

# Midterm available today

Will include today's material

Still due 11/1

# Homework:

Add validations and testing