NO. 1

Intro

NO. 2

Request Spec

GOALS FOR REQUEST SPEC

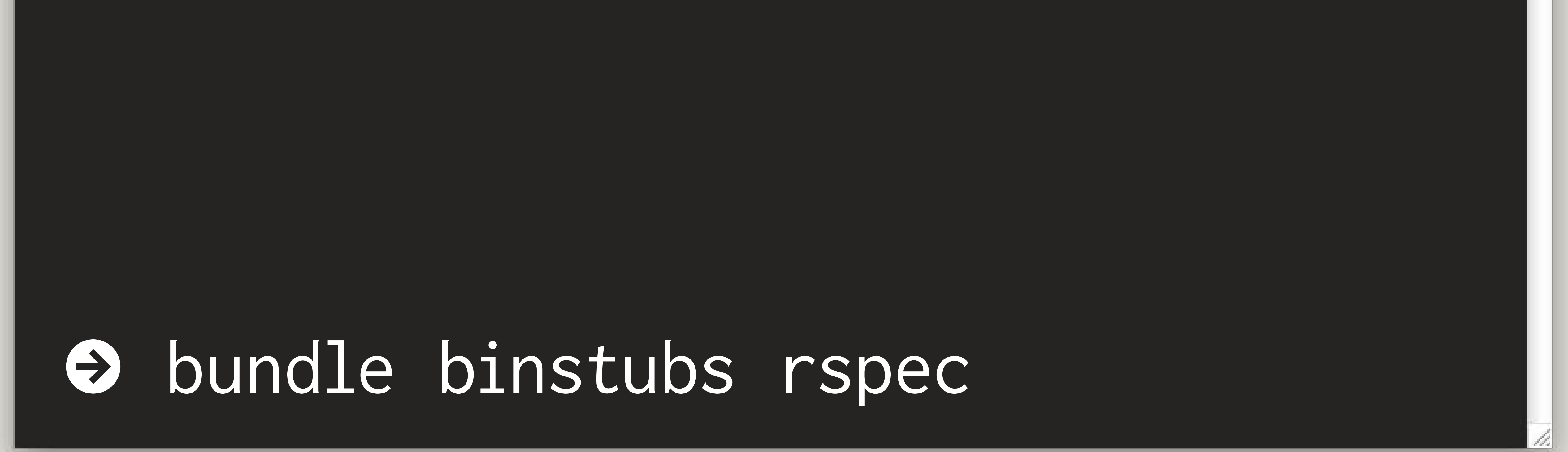
- ✓ Setup the project
- ✓ Write a simple request spec
- ✓ Configure and use Capybara
- ✓ Set up a Rails controller, view, and route

STEPS TO COMPLETE

- ✓ Configure Capybara
- ✓ Write a request spec
- ✓ Configure routes
- ✓ Make a Rails controller and view

TIP: Use our Gemfile

For the smoothest experience, use the complete Gemfile we've provided for you. It specifies version numbers that were compatible at the time this video was filmed.



```
➔ bundle binstubs rspec
```

Use the binstubs sub-command

To create commands in `bin` with the best compatibility with Rails, use the `binstubs` command, *not* the `--binstubs` flag. The final argument is the name of the RubyGem for which commands should be created.

NO. 3

Refactor

GOALS FOR REFACTOR

- ✓ Brief refactoring
- ✓ Move setup to a before block
- ✓ Use a context

NO. 4

HTML Content

GOALS FOR HTML CONTENT

- ✓ Write examples that examine rendered HTML
- ✓ Create a masthead showing a title and subtitle
- ✓ Create helper methods
- ✓ Use the have_selector matcher to find HTML content
- ✓ Use have_title to verify the page title

TASK

Write an example for and implementation of a helper that displays a subtitle. It should say “Read comments from your favorite blogs.”

NO. 5

Big Steps, Small Steps

GOALS FOR BIG STEPS, SMALL STEPS

- ✓ Understand the edges of what request specs can do
- ✓ Mark an example as pending
- ✓ Move on to other kinds of tests

KINDS OF SPECS

- ✓ Request: Runs the full stack, from controller to model to rendered view.
- ✓ View: Specifies behavior on a single view. Rarely used.
- ✓ Controller: Specs the controller in isolation from models and views.
- ✓ Model: Operates the methods on a model, separate from its usage in a controller. Very useful, and used often.
- ✓ Helper: Executes a single view helper method.

NO. 6

Automated Spec Runner

GOALS FOR AUTOMATED SPEC RUNNER

- ✓ Setup a tool: automated spec runner
- ✓ Install, configure, and use Guard
- ✓ Generate a guard command alias in the bin directory

WARNING! WARNING! WARNING!

Never use `bundle install --binstubs`.

Only use `bundle binstubs [gem_name]`.

NO. 7

Write a Model Spec

GOALS FOR WRITE A MODEL SPEC

- ✓ Start writing the Blog model
- ✓ Create files for the Blog model
- ✓ Write a model spec
- ✓ Implement the model

NO. 8

Test Model Features with Shoulda

GOALS FOR TEST MODEL FEATURES WITH SHOULDA

- ✓ Use *the* `shoulda` gem
- ✓ Implement *the* `Blog` model
- ✓ Use automatically generated descriptions

TASK

Write an example and expectation for `comments_feed_url`. It should be both present and unique.

NO. 9

A Custom Model Method

GOALS FOR A CUSTOM MODEL METHOD

- ✓ Write examples for the Blog model
- ✓ Implement a custom method on the Blog model
- ✓ Write fast tests by skipping the database

NO. 10

Comment Model

GOALS FOR COMMENT MODEL

- ✓ Augment the Blog model
- ✓ Introduce a related Comment model
- ✓ Generate the Comment model and migration

TASK

The `link` attribute should have a `unique` constraint. Write an RSpec example and the matching implementation.

NO. 11

Association Methods

GOALS FOR ASSOCIATION METHODS

- ✓ Use the WordPress comments client code from the Blog model
- ✓ Write and test association methods
- ✓ Create new Comment objects

NO. 12

Stub the Network

GOALS FOR STUB THE NETWORK

- ✓ Use a helper method to fake network access
- ✓ Speed up tests by providing local data

stub

A stub is a temporary replacement for a single method on a single object. It doesn’t keep track of how many times it is called, or if it ever is. It’s a generic placeholder.

EXAMPLE

A method that accepts any API request to a URL and returns reasonable data for further processing.

TIP: Test your network isolation

Tests shouldn't touch the network. See if you've properly stubbed your RSpec examples by turning off your wifi card, then running your test suite.

```
expect(Object).to receive(:method).and_return(value)

allow(Object).to receive(:method).and_return(value)
```

New syntax

Mocks use `expect` with `receive` and other options. Stubs use `allow`.

NO. 13

Factories for Easy Data Creation

GOALS FOR FACTORIES FOR EASY DATA CREATION

- ✓ Use a testing concept called a factory
- ✓ Refactor commonly used data into a single class
- ✓ Store attributes for creating a new object

NO. 14

Home Request Spec

GOALS FOR HOME REQUEST SPEC

- ✓ Return to the Home request spec
- ✓ Use model code to implement the home page with data

STEPS TO COMPLETE THIS FEATURE

- ✓ Write a view template
- ✓ Load data in the controller
- ✓ Add a custom model scope to retrieve relevant data
- ✓ Configure Rails routes

WARNING: It gets complicated

If you have many object that all need data, fixtures can become difficult to work with.

On the other hand, *any* complicated data structure can be difficult to work with.

Benefits of fixtures

Fixtures load quickly because they bypass the callbacks you've built into your models.

If you need that data, hard-code it into the fixture when you write it the first time.

NO. 15

Visual Debugging with Capybara

GOALS FOR VISUAL DEBUGGING WITH CAPYBARA

- ✓ Use save_and_open_page to view test output

NO. 16

Blog Detail

GOALS FOR BLOG DETAIL

- ✓ Implement a simple controller
- ✓ Implement a view to display comments for a single blog
- ✓ Fix a rendering bug by specifying the character set

```
describe BlogsController do

  before :each do
    post :create, title:’Example’, comments_feed_url:’http://...’
  end

  let(:blog) { Blog.find_by_permalink ‘example’ }

  it “creates a new blog record” do
    expect(blog).to be_valid
  end

end
```

Controller Spec

Quickly tests a single action on a controller. Fast, but is detached from user activity. Doesn’t render view templates.

```
describe BlogsController do
  integrate_views

end
```

Testing views with controllers

The `integrate_views` option forces a controller spec to render the associated view template with the spec's data.

NO. 17

New Blog Form

GOALS FOR NEW BLOG FORM

- ✓ Write a form for creating new blogs
- ✓ Provide a title and comments feed URL to save to the database
- ✓ Use Capybara to fill out and submit forms
- ✓ Create a Blog record in the controller

NO. 18

Conclusion

GUIDELINES FOR GREAT SPECS

- ✓ Take small steps
- ✓ Pay attention to speed and other metrics
- ✓ Build for humans
- ✓ Use Ruby

Take small steps

Try to be in red for as short a time as possible.

How to minimize time in red

Write expectations that only need a small amount of implementation code to be satisfied.

Write the smallest amount of implementation code needed to satisfy each expectation.

Pay attention to speed and other results

Does the suite take significantly more time to run than before you implemented that feature? That could signal a problem in the implementation code or the examples.

Code for humans

Keep in mind that people will be using your applications (in some way). How does it affect them?

Use Ruby

Try to implement features (in code or RSpec) with the Ruby language. Only reach for third-party dependencies when necessary.