Testing in Web Applications

So we've learned the basics of **testing** and **Web applications**.

But how can we tie them together?

Let’s take a look at how to bring a Week 1 exercise to the Web.

Let’s build a

Sinatra blog system.

We are going to give each piece a **distinct role** in our application.

These roles will follow the MVC pattern. (model-view-controller)

First let’s compare Week 1’s blog to Week 2’s blog.

Week 1 blog vs. Week 2

In our Week 1 blog we had two main classes: `Post` and `Blog`.

We are still keeping those.

Week 1 blog vs. Week 2

But the function of `Post` and `Blog` will change slightly.

In Week 1, they were partially in charge of **presentation**, how they looked on the screen.

Week 1 blog vs. Week 2

On our Web blog, we are going to let the **views** handle presentation concerns (HTML).

The `Post` and `Blog` classes are only going to handle our data.

Week 1 blog vs. Week 2

Since the `Post` and `Blog` classes are only going to handle our data, they are in essence our **models**.

They represent what our application is all about.

Week 1 blog vs. Week 2

Both `Post` and `Blog` need to have **unit tests** that make sure they function as expected throughout the development of the blog.

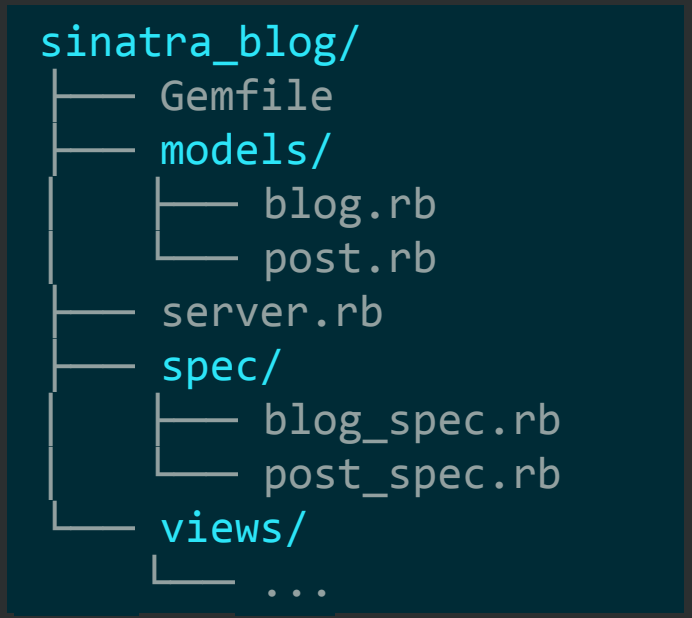
Week 1 blog vs. Week 2

You don’t necessarily have to use TDD (testing first), but every feature you add **needs tests**.

If you add any more classes,
they **need tests** as well.

Week 1 blog vs. Week 2

Our project structure might look like this:



So our classes are our models. How do we use them in a Web application?

Classes in a Web app

For now, the main instances of your `Blog` and `Post` classes will live in your `server.rb`.

They will be defined outside any route but they will all be accessible in the routes.

Classes in a Web app

For example:

```ruby
# server.rb
require "sinatra"
require_relative "lib/blog.rb"
require_relative "lib/post.rb"

blog = Blog.new(...)
blog.add_post Post.new(...)
blog.add_post Post.new(...)

get "/" do
  # Use blog in some way
  blog.posts
end
```

Classes in a Web app

Start with the Sinatra part of the app
until the point you either:

(1) need to call a method a `Blog` or `Post`
(like if you need data)

or

(2) need to make a new instance of a `Post`
(you only need the one instance of `Blog`)

Classes in a Web app

Then add the method or functionality you need on `Blog` or `Post` and create a test for that method!

Let’s help you along for the first tests.

Starting with the blog's home page, you will need the list of posts from the `Blog`.

List of posts

Maybe you have a `#posts` method on `Blog` that provides the array of posts.

```ruby
# server.rb
# [...]

get "/" do
  @posts = blog.posts

  erb(:home)
end
```

List of posts

That needs a test, of course!

```ruby
# spec/blog_spec.rb
require_relative("../lib/blog.rb")

RSpec.describe Blog do
  before(:each) do
    @blog = Blog.new(...)
  end

  it("#posts returns list of posts") do
    expect(@blog.posts).to  # ??? which matcher should we use
  end
end
```

List of posts

Your `Blog` instance is full of `Post` instances that provide the post information like the title.

```
<!-- views/home.erb -->
<% @posts.each do |post| %>
  <%= post.title %>
<% end %>
```

Yup, that needs a test as well.

List of posts

```ruby
# spec/post_spec.rb
require_relative("../lib/post.rb")

RSpec.describe Post do
  before(:each) do
    @post1 = Post.new(...)
    @post2 = Post.new(...)
  end

  it("#title returns title") do
    expect(@post1.title).to # ??? which matcher
    expect(@post2.title).to # ??? which matcher
  end

  it("#date returns date") do
  end

  it("#text returns text") do
  end
end
```

Remember in tests we test **methods** and their **return values**.

In Web applications, think about what methods you need for your routes.

What should the method be **named**?

What should the method **return**? An array? A string? Nothing?

What should the method **receive as a parameter**?

You decide what you need!

Remember, classes are your models. They represent data.

They shouldn’t have any HTML inside them.

Onto the exercise!