Sinatra

Sinatra is a gem that provides methods for creating Web applications in Ruby.

Gemfile

As such, you start a Sinatra project by creating a `Gemfile`.

```
source "https://rubygems.org"

gem "sinatra"
```

Now we can create our `server.rb` file that will contain our Web application code.

```ruby
# server.rb
require "sinatra"

get "/" do
  "My first Sinatra app."
end
```

Sinatra's methods correspond to HTTP verbs

HTTP GET

```ruby
get "/" do
  "Hello, world!"
end
```

GET

```ruby
get "/hi" do
  "Hello, world!"
end
```

Exercise

Add a the `/about` URL to your Sinatra app.

Have that page show a brief bio.

Of course Web applications don't send random strings to the browser.

They send the browser HTML.

HTML

The way we are handling sending the browser a response doesn't lend itself to HTML.

```
get "/author" do
  "<h1>My First Sinatra</h1>
  <p>
    <b>Author:</b>
    Johnny Appleseed
  </p>"
end
```

HTML

Especially when you consider that a page could have hundreds of lines of HTML.

```
get "/author" do
  "<h1>My First Sinatra</h1>
  <p>
    <b>Author:</b>
    Johnny Appleseed
  </p>"
end
```

Enter views, separate files that will contain our HTML code.

Views

Create a folder called `views`.
Inside it, create a file called `author.erb`.

```
project/
├── server.rb
└── views/
    └── author.erb
```

ERB stands for embedded Ruby. It's HTML that can have Ruby inside it.

ERB

Let's put that author HTML in the `author.erb` file.

```
<h1>My First Sinatra</h1>
<p>
  <b>Author:</b>
  Johnny Appleseed
</p>
```

ERB

That works but where's the embedded Ruby? Let's add some.

```
<h1>My First Sinatra</h1>
<p>
  <b>Author:</b>
  Izzy Ironside
</p>
```

ERB

You can use `if..else` and other structures with `<% %>`.

```
<% if Date.today.monday? %>
  <p>
    Izzy Ironside is <strong>NOT</strong>
    a fan of Mondays like today.
  </p>
<% end %>
```

ERB

Don’t forget the end.

```
<% if Date.today.monday? %>
  <p>
    Izzy Ironside is <strong>NOT</strong>
    a fan of Mondays like today.
  </p>
```

ERB

You can show Ruby values in the HTML with <%= %>.

```
<p>
  Izzy Ironside was born on
  <%= Date.today %>.
</p>
```

ERB

Don’t leave out the equal sign.

```
<p>
  Izzy Ironside was born on
  <% Date.today %>.
</p>
```

ERB

```erb
<h1>My First Sinatra</h1>
<h2>
  <b>Author:</b>
  Izzy Ironside
</h2>
<p>
  Izzy Ironside was born on
  <%= Date.today %>.
</p>
<% if Date.today.monday? %>
  <p>
    Izzy Ironside is <strong>NOT</strong>
    a fan of Mondays like today.
  </p>
<% end %>
```

Views and @ variables

Use instance variables to send data from Sinatra to a view.

```ruby
get "/hi" do
  @greeting = "Hello World"
  erb(:hipage)
end
```

Views and @ variables

In the view just use the instance variable inside an ERB tag.

```
<h1>Welcome page!</h1>
<%= @greeting %>
```

Views and @ variables

A local (regular) variable will *not* work.

```
get "/hi" do
  greeting = "Hello World"
  erb(:hipage)
end
```

```
<h1>Welcome page!</h1>
<%= greeting %>
```

Views and @ variables

Avoid errors, use instance variables.

```
get "/hi" do
  @greeting = "Hello World"
  erb(:hipage)
end
```

```
<h1>Welcome page!</h1>
<%= @greeting %>
```

Views and @ variables

Use instance variables to send data from Sinatra to a view.

```ruby
get "/hi" do
  @greeting = "Hello World"
  erb(:hipage)
end
```

Exercise

Use views to add a page that shows the current time.
Use some fancy date formatting.

Loops in ERB

It might be tricky at first, but loops work in a similar way that ifs do in ERB.

```
<ul>
  <% @ingredients.each do |ingr| %>
    <li><%= ingr %></li>
  <% end %>
</ul>
```

Loops in ERB

Notice a few things. The each line doesn't have an = in the ERB tag.

```
<ul>
  <% @ingredients.each do |ingr| %>
    <li><%= ingr %></li>
  <% end %>
</ul>
```

Loops in ERB

The code inside the loop is the one that shows things in HTML.

```erb
<ul>
  <% @ingredients.each do |ingr| %>
    <li><%= ingr %></li>
  <% end %>
</ul>
```

Loops in ERB

And we are using instance variables to send the array to the view.

```erb
<ul>
  <% @ingredients.each do |ingr| %>
    <li><%= ingr %></li>
  <% end %>
</ul>
```

Loops in ERB

That means the route had something like this:

```ruby
get "/pizza" do
  @ingredients = [ "cheese", "pepperoni", "mushrooms" ]

  erb(:pizza)
end
```

In the Web, you will often need to send some files to the browser directly.

We call these static assets. Usually they mean images, fonts, CSS and JavaScript.

Static assets

Any files you put in the `public` folder will be sent as is to the browser.

Put any image, font, CSS and JavaScript files in there and they will be accessible.

Static assets

Create a folder called `public`.
Download your favorite cool image and put it inside `public`.

```
project/
├── public/
│   └── cool_image.png
├── server.rb
└── views/
    └── author.erb
```

Static assets

Your cool image will be accessible at `localhost:4567/cool_image.png`.

You can use just `/cool_image.png` in your HTML though.

Static assets

Like in this `<img>` tag, for example.

```
<img src="/cool_image.png"
     alt="A very cool image">
```

Exercise

Create a CSS file in your public folder that styles the homepage

The idea of a Web application is to have pages be dynamic.

That means that the same code will be able to generate different content in the page.

Twitter profiles

Think of a Twitter profile.

You visit `twitter.com/ironhack`
and you get Ironhack's profile.

You visit `twitter.com/starwars`
and you get Star Wars' profile.

Twitter profiles

The same code generates those and all Twitter profiles.

The content of the profile page is dynamic.

Twitter profiles

How does Twitter know what content to show?

It makes use of the URL. The username in the URL determines the content of the page.

Dynamic elements in the URLs of Web applications are called URL parameters.

URL parameters

You can make routes use URL parameters in Sinatra too.

```ruby
get "/users/:username" do
  erb(:user_profile)
end
```

URL parameters

Just put a : before a URL segment.

```ruby
get "/users/:username" do
  erb(:user_profile)
end
```

URL parameters

The value of the parameter on any given request will be available in a special hash called `params`.

```ruby
get "/users/:username" do
  params

  erb(:user_profile)
end
```

URL parameters

The name of the URL parameter will be the key inside the `params` hash.

```ruby
get "/users/:username" do
  params[:username]

  erb(:user_profile)
end
```

URL parameters

Let’s use that value in our view with an instance variable.

```ruby
get "/users/:username" do
  @username = params[:username]

  erb(:user_profile)
end
```

URL parameters

Maybe our ERB looks something like this.

```
<h1>
  Profile for
  <%= @username %>
</h1>
```

Exercise

Add a dynamic page that shows the time from a number of hours ago.

Example URLs:

`/hours/ago/5` (the time 5 hours ago)
`/hours/ago/7` (the time 7 hours ago)