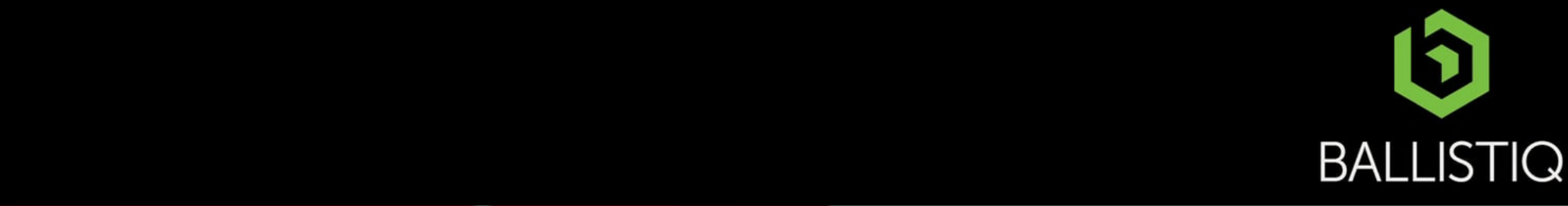
A Practical Guide to Ruby on Rails

by Louis-Olivier Guérin
http://louisolivier.com
http://github.com/louis-ver



Huge thanks to Ballistiq for making this workshop possible!

Setting up your Rails Environment

https://gist.github.com/louis-ver/7b46ec42ea2d94e01b2d0aa96b4558f4

First Things First

What is Ruby on Rails?

Ruby on Rails is…

Ruby on Rails is a Web Framework, built on top of the Ruby programming language

It allows developers to quickly build robust web applications that are easy to maintain, easy to understand, and most of all fun to write

It relies heavily on the MVC architecture, which is common in object-oriented web or mobile application design

Model–View–Controller

Central to any Rails application is the separation of the Model, View, and Controller.

Model –> Contains data, state, business logic, no knowledge of the user interface (decoupled)

View –> The interface that the user interacts with. Does not do any processing. Pulls data from the models (through the controller) and presents it to the user

Controller –> Waits for events from the user, interacts with the model, and returns the appropriate data to the view. Also performs validation of user input.

Rails App Structure

Calling `$rails new <projectname>` creates a new Rails project with all the necessary files and basic configuration. It also downloads all the necessary `gems` using bundler, Ruby's package manager.

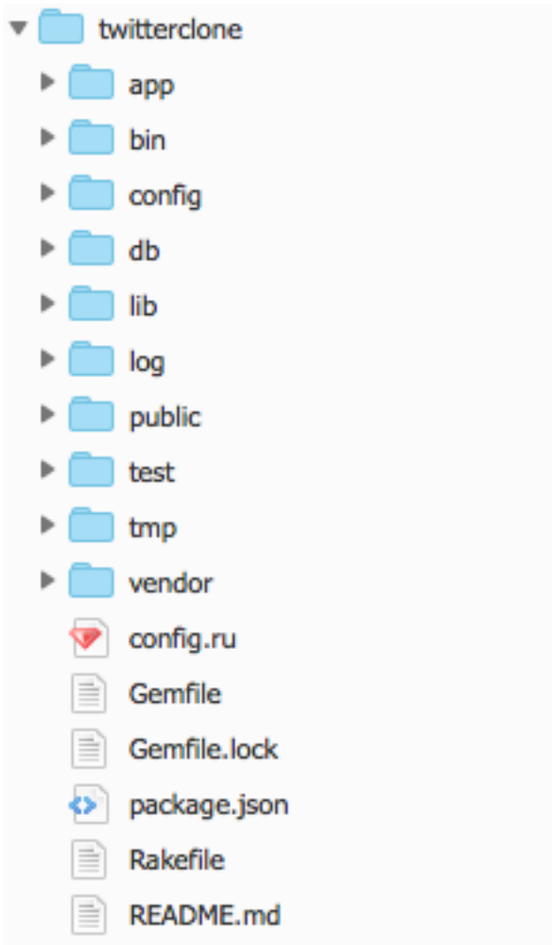
Let's run that command with the name of our application:

```
$ rails new twitterclone
```

We should now have a `twitterclone` directory containing all the files for our new application

Rails App Structure

All Rails apps have this structure:



A Simple Hello World

Open up `application_controller.rb` inside /app > controllers

Write the following function inside the ApplicationController class:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  def hello
      render html: "Hello, world!"
  end
end
```

A Simple Hello World

Open `routes.rb` inside config/

Inside the function, add the following line *:

```
root 'application#hello'
```

This says “when someone visits the root location of my application (normally “/“), invoke the `hello` method of my `application_controller`”.

* side note: in Java, this would be written as root(‘application#hello’). Parentheses are not required in Ruby

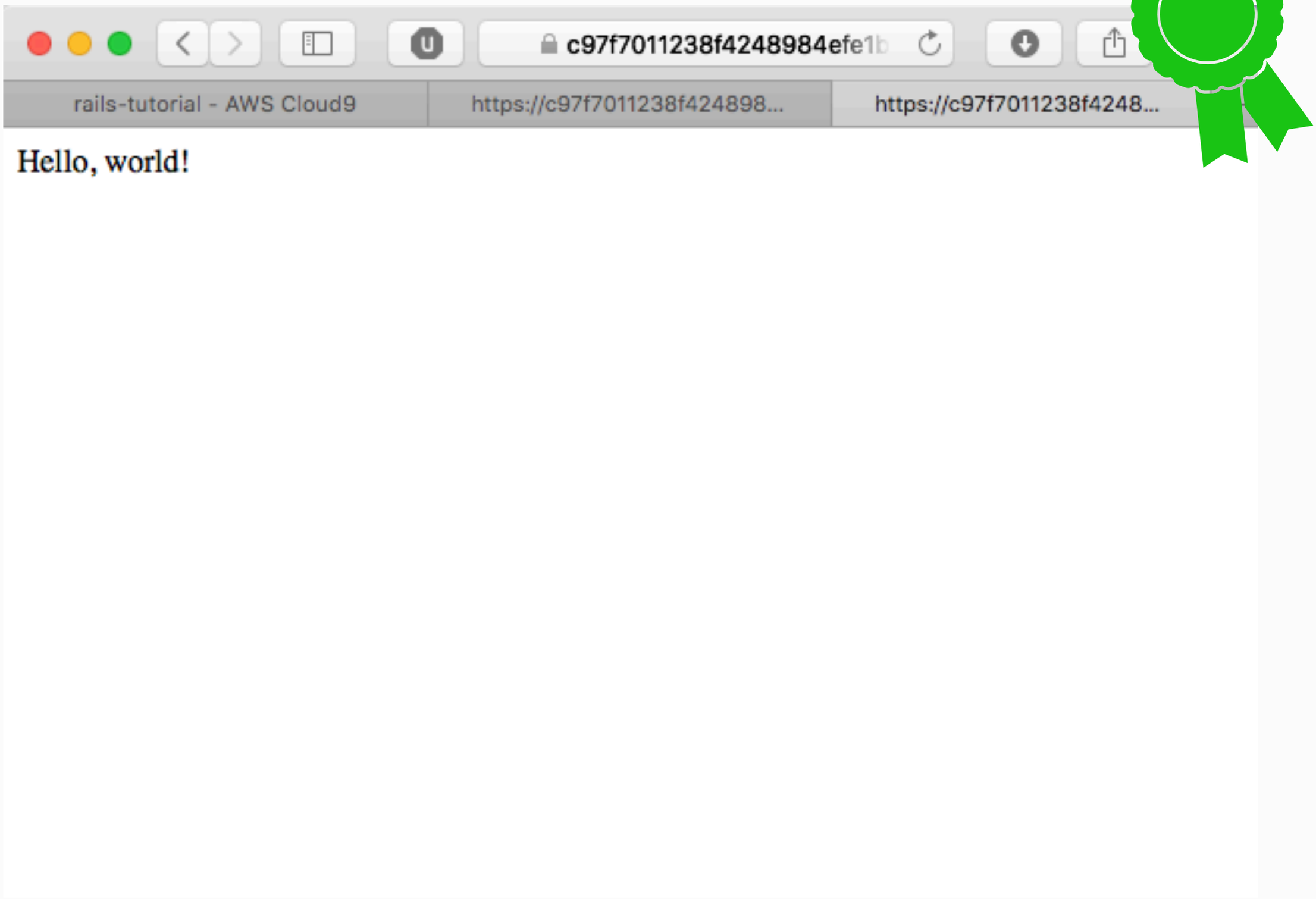
Time to Run it!

Lots of stuff with Rails happens inside the Terminal window

To run our simple application, we need to tell Rails to start the server. We do this by sending this command inside our terminal window:

```
$ rails server
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.11.0 (ruby 2.4.1-p111), codename: Love Song
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:8080
Use Ctrl-C to stop
```

Time to Run it!



Dinner time!

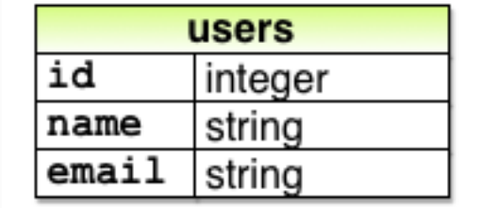
Resources

When talking about web applications, resources are objects that can be read, created, updated and deleted through HTTP protocol methods like GET, POST, PUT, DELETE.

Rails makes it super easy to create new resources, and does a lot of the routing and wiring for you.

```
GET /path/to/resource?query=rails-is-cool
apiKey: iz8Hk5hd7hHkwxHdn==
Host: www.louisolivier.com
Connection: close
```

Models



users	
id	integer
name	string
email	string

The first step when making a web application is to create a data model, which represents the structures needed by our application (`users`, `posts`, for the most part)

We'll first want to create our `User` model. Let's keep our User simple: it'll have a name that's publicly visible, as well as an email

Generating Resources

Rails makes it easy to create new components. Here are some really helpful ones:

Generating our User model

```
$ rails generate scaffold User name:string email:string
```

Note here that the id of the user is not included. Not a mistake — Rails creates one automatically for us.

Before we run our applications, we'll have to create the table in the database for our new User resource. We can do that with:

```
$ rails db:migrate
```

Testing our Resource

Let's run our server to see if everything is working

```
$ rails server
```

This command will start the server locally for you to use it.

Now let's see what URL's are available to us:

URL	Action	Purpose
/users	`index`	page to list all users
/users/1	`show`	page to show user with id `1`
/users/new	`new`	page to make a new user
/users/1/edit	`edit`	page to edit user with id `1`

Testing Endpoints

Let’s test some endpoints

```
> /users, /users/new, users/1, users/1/edit
```

As you can expect, `users/1` won’t work until we create our first user.
Same thing with `users/1/edit`

Let’s then head over to /users/new and create our first User! We can create a second one, because why not?

In /users, let’s also test the Destroy link, which should delete the resource. If it doesn’t seem to work, make sure JavaScript is enabled in your browser.

Making /users the Landing Page

Let’s make it so that when users visits our application, the first thing they see is the user list:

You do this by going to `config/routes.rb`, and changing the parameter passed to the root function:

```
Rails.application.routes.draw do
  resources :users
  root 'users#index'
end
```

What does this mean??

Making /users the Landing Page

```
Rails.application.routes.draw do
  resources :users
  root 'users#index'
end
```

This tells the application: When my users request the root "/" URL (ie www.my-cool-app.com/), direct them to the `users`'s controller index method. Let's go take a look at that controller.

The users_controller

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    .
    .
    .
  end

  def show
    .
    .
    .
  end

  def new
    .
    .
```

The users_controller

Let's start with a method that we can understand: index

```
def index
  @users = User.all
end
```

This says: Get all the users from the database, and make them visible through the `@users` instance variable.

The `index` method maps to the `GET /users/` HTTP request. Following Rails conventions, this request should show a list of all the elements of that resource.

The users_controller

All the messy SQL querying, session handling, etc. is all handled for you!

This is where Rails really shines: A lot of the configuration and lower-level stuff is done for you. *

* (But fear not, hackers and tinkerers! If you want to dig in and override the defaults, there's nothing stopping you from doing that, too)

The users_controller

Where does the HTML stuff happen?
app/views/users/index.html.erb

```
<tbody>
  <% @users.each do |user| %>
    <tr>
      <td><%= user.name %></td>
      <td><%= user.email %></td>
      <td><%= link_to 'Show', user %></td>
      <td><%= link_to 'Edit', edit_user_path(user) %></td>
      <td><%= link_to 'Destroy', user, method: :delete, data: { confi
    </tr>
  <% end %>
</tbody>
```

Weaknesses of our User resource

Can you spot a few weaknesses in our `User` resource?

No data validations

No authentication

Not much style/layout

For the moment, we're gonna let these problems slide. Just be aware that they will need to be addressed in the future.

The Micropost resource

Remember how to generate a scaffold for a resource?

```
$ rails generate scaffold Micropost content:text
user_id:integer
```

Microposts have content, and belong to a user. That's why we include both a `content` field, as well as a `user_id`

Again, to create the table in the database, we need to run:

```
$ rails db:migrate
```

microposts	
id	integer
content	text
user_id	integer

The Micropost resource

Like we did with users, let's explore the Micropost resource's public pages:

HTTP request	URL	Action	Purpose
GET	/microposts	index	page to list all microposts
GET	/microposts/1	show	page to show micropost with id 1
GET	/microposts/new	new	page to make a new micropost
POST	/microposts	create	create a new micropost
GET	/microposts/1/edit	edit	page to edit micropost with id 1
PATCH	/microposts/1	update	update micropost with id 1
DELETE	/microposts/1	destroy	delete micropost with id 1

The Micropost resource

We called them Microposts...but is there anything actually micro about them? Not yet.

Let’s fix that. Head over to:

`app/models/micropost.rb`

And add this line:

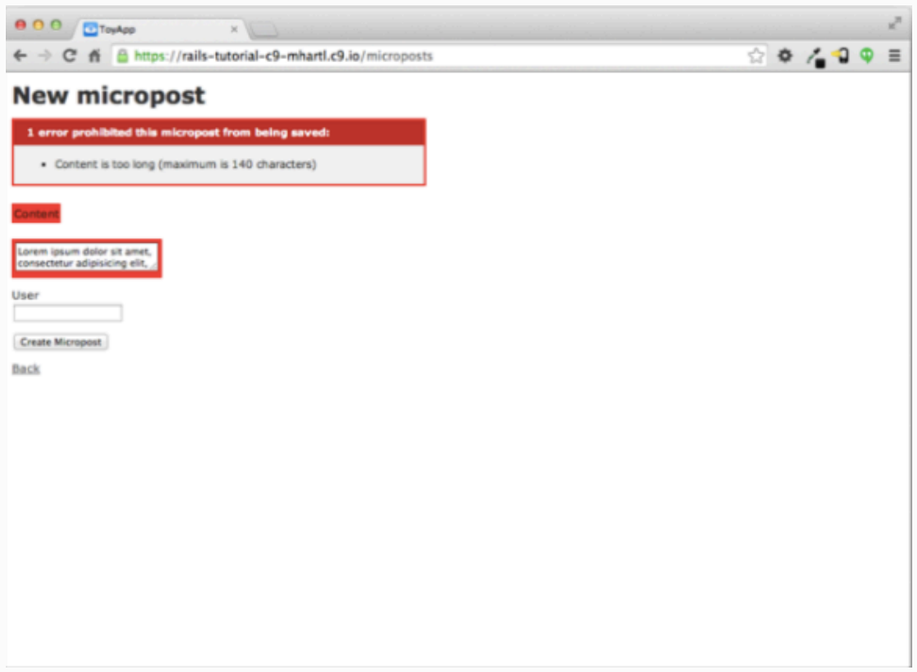
```
class Micropost < ApplicationRecord
    validates :content, length: { maximum: 140 }
end
```

The Micropost resource

Let’s test out our newly written code by creating a micropost with more than 140 characters.

Head over to this URL to try it out:

/microposts/new



The Micropost resource

We mentioned that Microposts should belong to a user...
but we haven't done anything about it. Let's fix that:

/app/models/user.rb

```
class User < ApplicationRecord
    has_many :microposts
end
```

The Micropost resource

/app/models/micropost.rb

```
class Micropost < ApplicationRecord
    belongs_to :user
    validates :content, length: { maximum: 140 }
end
```

The Micropost resource

Let's see what effects this has on our program:

```
$ rails console
>> first_user = User.first
>> first_user.microposts
>> first_user.microposts.first
```

As we can see, the `micropost` belongs to a user, and can be retrieved simply from the `User` object!

The Micropost resource

Finally, what's a micropost without any content? Let's make sure that a micropost always contains some text:

/app/models/micropost.rb

```
class Micropost < ApplicationRecord
    belongs_to :user
    validates :content, length: { maximum: 140 },
                        presence: true
end
```

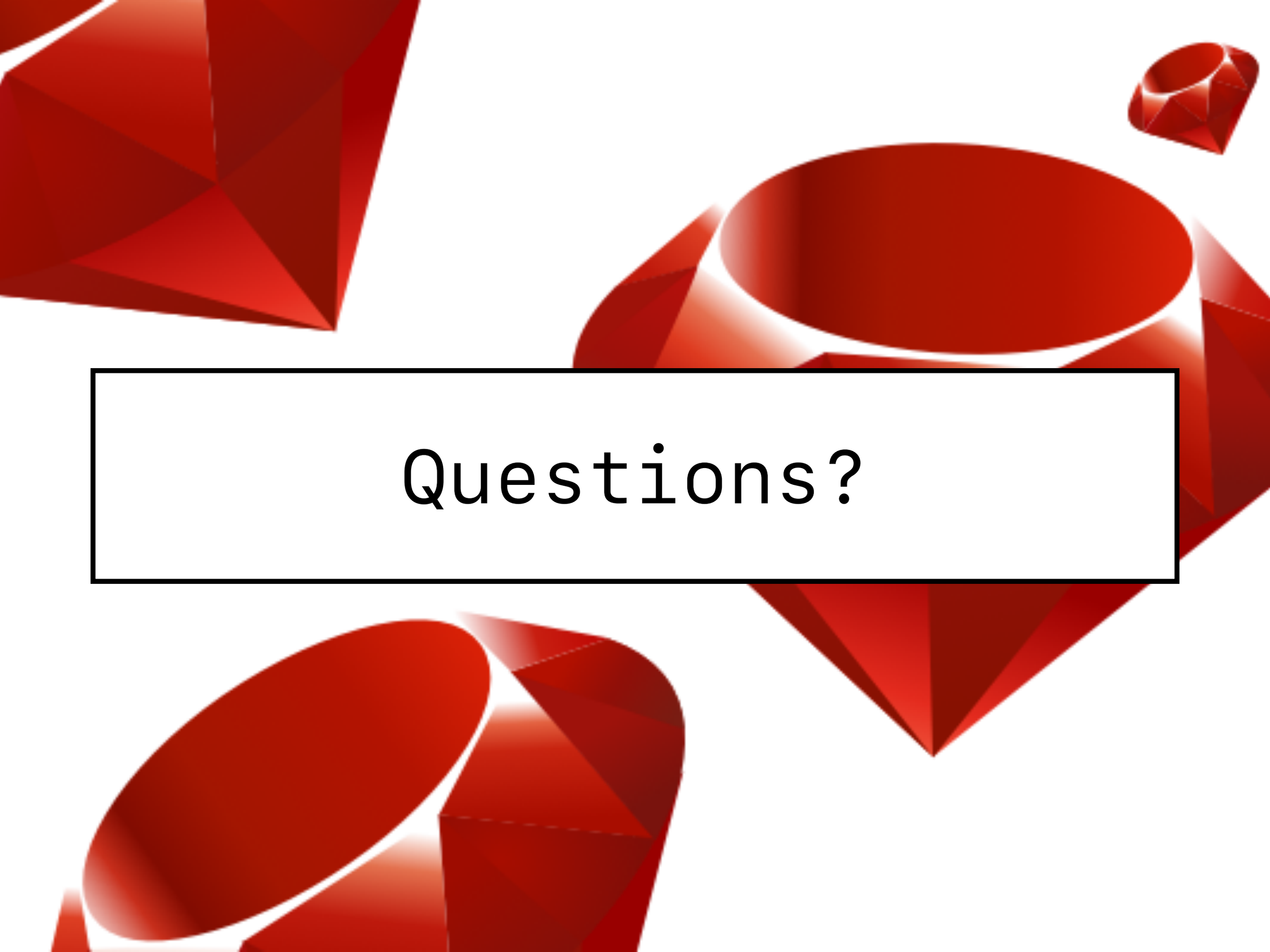
We have a (very) minimal Twitter Clone!

We've managed to make an application that has a concept of users and microposts. Obviously there is a lot missing in our application, but we've went over the high-level basics of a Rails application.

To continue building your application on your own free time, visit the amazing tutorial on which this workshop was based: https://www.railstutorial.org/book/

Happy Coding!

Questions?



Want to learn how to apply this in the real world?

Apply for an internship at Ballistiq!

Send an email to intern@ballistiq.com with your resume