Sign Up!

Tuesday, September 20, 2016 6:47 PM

- Setting up a sign up form with HTML and Ruby; Using REST architecture to handle calls

Showing Users

- Create a branch (like always)

Debugging & Rails Environments

- The **debug** method with **params** will allow for debugging specific pieces

  ```
  <%= debug(params) if Rails.env.development? %>
  ```

  From <https://www.railstutorial.org/book/sign_up>

- We can access the method Rails.env.development? To only provide debugging when we are building…
- Rails allows for environments in which you can build and test (In console run Rails.env)
  - Production: What is live
  - Development: What we are working on
- To make debug pretty, add CSS in listing 7.2

Users Resource

- If we have no users in the database, we are going to need to add one (see Section 6.3.4)
- We are going to focus on POST,GET,PATCH,DELETE of REST
- To allow for URL calls to work with key fields, add **resources :users** to the routes.rb file
  - This will allow for all dynamic calls to the Users controller
- Adding the View
  - We'll need to add a Users.Show view, since we don't have a template available.
  - Update the controller to pass the necessary User over to the model and view

    ```
    @user = User.find(params[:id])
    ```

    From <https://www.railstutorial.org/book/sign_up>
    - This passes the specific USER ID called into the find method and stores it within the @user
- Debugging Some More
  - Add debugger to the show method of the UsersController
  - We can now interact with a prompt as a console through the browser
  - Close With Ctlr+D

- Gravatar Image
  - Adding a globally recognized avatar to the site
  - Requires just a hash value of the email address
  - Use **gravatarfor @user**
  - You then can implement a necessary helper function into users_helper.rb

- Sidebar
  - Update the necessary users.html.erb file to drop in the new contents via typical ruby commands
  - Note: We are using built in bootstrap classes here: col-md-4
  - Note: We are using an HTML5 tag, aside (Typically for secondary content, sidebars, etc)

Signup Form

- We will need to build a form that will pass over the necessary fields into **form_for** and then the Active Record
- Let's update the users_controller.rb with **@User = User.new**
- Digging into the HTML
    - Form_for(@user) do |f|……end
    - Let's loop through each variable and process
    - The code f.label :name => Creates the necessary label and field for the form automagically (And all other fields)
    - Ruby is also pretty smart -> Uses HTML5 Email field for client side validation without the need for JS (If disabled) as well as special keyboards for mobile
    - **Name** attribute: The unique ID for the form field. Ruby will grab this and make it a part of the user object
    - Ruby sees the @user object and handles the form tag with easy magically
    - Additional tags are added for validation and character encodings

Unsuccessful signups

- Where the user fails to do something right
- When we create users we call the **create** action via POST
- We can store the User.new into @user and see if the .save was completed.
- Processing
    - Ruby pulls all the fields into **params** of the UserController. This is set into hash maps

- Strong Params
    - Initializing the entire params has is dangerious!
    - When in need of more data, we can use administrative flags
    - Within the controller we sepecify what params are required now

```
params.require(:user).permit(:name, :email, :password, :password_confirmation)
```

From <https://www.railstutorial.org/book/sign_up>

    - We can add a private User_params as a method to pass the required

- Sending Error Messages
    - In order to add validation messages, we'll need to update the views
    - Adding a render will help display the necessary fields

```
<%= render 'shared/error_messages' %>
```

From <https://www.railstutorial.org/book/sign_up>

    - Along with the other fields (Below is a template)

```
<%= f.text_field :name, class: 'form-control' %>
```

From <https://www.railstutorial.org/book/sign_up>

    - To fully implement add a new directory and template to link up the logic and display
    - To make text more pleasant, we can use **pluralize** method

- Testing
    - We can create automated tests to check our form! (Unlike the olden days)
    - Generate a test file:

```
rails generate integration_test users_signup
```

From <https://www.railstutorial.org/book/sign_up>

    - **Assert no difference** comparison between the user count before and after the block for **assert)no_difference**

Successful Signups

- We'll need to update the redirect on success
- Using redirect_to user_url(@user) will take the user to their specific profile page

- The Flash
    - A Flash is a quick message displayed after an action occurs
    - We can add them into the control, such as

      ```
      flash[:success] = "Welcome to the Sample App!"
      ```

      From <https://www.railstutorial.org/book/sign_up>
    - The messages can be design tweaked within the HTML ERB files
    - **:success** is a symbol that is converted to success during template insertion
    - We have others too…. **Alert-danger, info, warning, danger**

    - After changes, don't forget to migrate!

    - Testing valid submissions
        - With **assert_difference** we can add **follow_redirect!** To check the redirect off to the next page.

Deployment

- Let's git commit and push up, it's time!

- SSL
    - Secure Sockets Layer, a method of secure transport on the internet
    - We can add a setting to the config to enforce SSL

      ```
      # Force all access to the app over SSL, use Strict-Transport-Security,
       # and use secure cookies.
       config.force_ssl = true
      ```

      From <https://www.railstutorial.org/book/sign_up>

- PUMA
    - To increase performance on the web, we'll switch to Puma on Heroku.
    - Need to add a new gem, puma (Default in Rails 5)
    - We'll need to update the file contents within config/puma.rb; ./procfile
    - Then push up the changes to git and heroku!