# PFR: Intro to Rails – Lab Projects

## Description

For the Unit 2 Project, students will create a Twitter clone in Rails, called “Shouter”. The completed application will allow users to sign up, post “Shouts” (tweets), follow other users, and customize the look and feel of their profile. There will be two ways to consume the content within the app: a global stream of all shouts, and a stream of shouts from followed users.

## Feature Specification

* User signup (/signup)
  + Users sign up with an email address, username, and password
  + Once signed up, a user is automatically logged in and taken to the global stream
  + A user shouldn’t be able to sign up with an email or username that is in use (these are case insensitive)
  + Usernames should only contain letters, numbers, and \_, and should start with a letter
  + Usernames should be between 1 and 40 characters long, inclusive
  + Email addresses should be valid (just check format)
  + Passwords must be at least 6 characters long
  + Passwords will be stored encrypted (hint: ActiveModel::SecurePassword)
* User login (/login)
  + A user can log in with their email address or username
  + If successful, the user is taken to the global stream
  + If unsuccessful, an error message is displayed
  + The login form should remember the username/email on failure, but not the password
  + Users should stay logged in via sessions
* User profile (/profile/USERNAME)
  + Each user should be able to access their user profile at /profile/USERNAME
  + The user’s last 20 shouts will be displayed in reverse chronological (reverse chron) format, and should be paginated using the will\_paginate gem
  + The profile will also show a Follow/Unfollow button, unless you are logged in and looking at your own profile
    - The button state will reflect whether the logged in user follows the viewed user
  + User profiles can have a custom background and text color set by its owner
    - These colors must be valid hex colors (#fff or #f0f0f0)
  + Users can also upload a background image for their profile
    - It must be a jpg, png, or gif
* Streams
  + Global stream (/shouts)
    - This is a list of all shouts from all users of the app, paginated 20 per page and in reverse chron
  + Following stream (/shouts?from=following)
    - This is a list of shouts in reverse chron from all users the logged in user follows. It should include the user’s shouts as well
    - Shouts should be paginated (20 per page)
    - This page is not available when logged out
* Followers
  + Each user displayed will have an Follow/Unfollow button next to it
    - Clicking Follow will create a new Follow record for the two users
    - Clicking Unfollow will destroy an existing Follow record
  + Users can only follow a user one or no times (i.e. don’t allow multiple Follow records for a [user, follower] pair); users can’t follow themselves
* Shouts
  + A shout is a piece of content that contains up to 140 characters of text and a timestamp
  + Shouts must not be longer than 140 characters, and also can’t be empty
    - Shouts that don’t abide by this restriction should not be saved
  + Shouts belong to a user

## **Gem Dependencies**

* will\_paginate

(This list excludes gems required by Rails itself)

## **Data Model**

Shout

id, timestamps (automatically added by Rails migrations)

user\_id (int) [use the `references` method in your migration]

content (text)

index on user\_id

Belongs to User, keyed on user\_id

User

id, timestamps (automatically added by Rails migrations)

email (string)

username (string)

password (string)

profile\_bg (string, default “fff”)

profile\_fg (string, default “000”)

profile\_image (string)

unique index on email

unique index on username

Has many Shouts, use dependent: :destroy, default order of “created\_at DESC”

Follow

id, timestamps (automatically added by Rails migrations)

user\_id (int)

follow\_id (int)

unique index on (user\_id, follow\_id)

Belongs to User, keyed on user\_id

Belongs to Follower, keyed on follow\_id, of class User

# Lab Tasks by Lesson

## Lesson 1 Lab

In this lab, you will verify that Rails is installed, learn about Ruby gems, create a fresh Rails application from the command line using the `rails` command, look at and understand Rails’ file structure, learn about and use routing, and finally, generate your first controller and get a simple “hello world” page to render.

### Tasks

1. Using the command line, verify that Rails is installed and determine the version
2. Using the command line, print a list of installed gems
3. Create a new Rails app called *shouter* using the `rails new` command
4. Ensure the app’s database is created by running `rake db:create` (You may have to edit config/database.yml with your system’s database host, username, etc. first)
5. Create a new controller called PagesController by running `rails generate controller pages`
6. Start the built-in Rails development server by running `rails server` and make sure it’s running
7. Add an action called “home” to your pages controller and add an ERB view template for it
8. Add the markup <h1>Hello World</h1> to this new template file
9. Point your browser to <http://localhost:3000/> and make sure the app is running
10. Enable Rails’ default routes (/controller/action/:id) and load your new home action in the browser

## Lesson 2 Lab

In this Lab, you will look at how Rails handles CSS stylesheets, add a custom layout template to your Pages controller, and set a default route.

### Tasks

1. In the app/assets/stylesheets directory, create a new stylesheet called “base.css.scss” and add it to the top of the list of included stylesheets in application.css.scss
2. Refresh the app in browser and view source. Check out how the stylesheet definitions are changed.
3. Use CSS to make your page have a grey background with black text (do this in your new base CSS file)
4. Create a new layout template called “pages.html.erb” within the app/views/layouts directory, and add enough markup to it so that it will render your pages
5. Set this new layout as the default layout for the pages controller
6. Make the home action in your pages controller the default route (hint: you will need to remove public/index.html)

## Lesson 3 Lab

In this Lab, you’ll create your first ActiveRecord model, run a set of database migrations, utilize ActiveRecord to read and write data to the database, and use Rails’ built-in model generator to generate various model files, including tests and fixtures.

### Tasks

1. Create a new model called User
2. Update the create\_users migration to reflect the data model for users per the project spec
3. Run migrations using rake
4. Make email, password\_digest, and username accessible attributes in your user model
5. Read about has\_secure\_password; you’ll come back to this later
6. Open the rails console and create a user record
7. Find the user record you just created and change the email address
8. Create a users controller with scaffolding (use `rails g scaffold\_controller users`) and update the routes config to generate restful routes for the users controller
9. Update the app/views/users/\_form.html.erb partial to include fields for email, username, and password
10. Update the app/views/users/show.html.erb template to display the user’s ID, email, and username
11. Update app/views/users/index.html.erb to display the user’s email address in the list

## Lesson 4 Lab

In this Lab, you will create a Shout model and build a form that lets users post new shouts (posts) to Shouter. You will add fields to a form and understand how form\_for works. You will also display date/time information, as well as use helpers. Finally, you’ll display a page with paginated shouts using the will\_paginate gem.

### Tasks

1. Create a new Shout model per the project data model
2. Create an Shouts controller with new, create, and index actions
3. Enable restful routes for the shouts controller
4. Use form\_for to create a form that lets you post a new shout (for now, set the user\_id to -1); implement the create action
5. Set up the index action in your new shouts controller to find all shouts ordered newest first
6. Display these shouts in an index.html.erb view, including a relative timestamp for their created\_at field (e.g. “posted 4 minutes ago”). Make sure to display the user\_id and content
7. Install the will\_paginate gem by adding it to your Gemfile
8. Change the index action to use will\_paginate; display 20 shouts at a time. Also update your view to include pagination controls
9. Add a helper method to the shouts helper that outputs the shout creation date like “October 10, 2012”, and update your view appropriately
10. For fun, set the body background color CSS to #eee and change the font to Helvetica

## Lesson 5 Lab

In this Lab, you will work with Validations. You’ll add built-in validations to your new user model, as well as customize their behavior. You will use regular expressions to validate user data, and you’ll also create a custom validation method. You’ll also add validations to bring your shouts up to spec.

### Tasks

1. Add validations to require the user provide an email, password, and username as per the project spec
2. Add validations to ensure the email and username are unique
3. Customize the password validation so that it only runs when creating a new user
4. Add a validation to ensure that the username follows the format outlined in the project spec
5. Also add a validation to make sure the user’s email address is valid
6. Add a custom validation method to ensure that the user’s username is not admin, kyle, or leo
7. Now update your shout model to validate per the project spec; display error messages on the new shout form if there are errors.

## Lesson 6 Lab

In this Lab, you will handle file uploads, and use FormBuilder to create a form to edit user records. You will allow users to upload a “cover image” for their profile, as well as set a custom background and text color. You’ll display these with CSS. You will also use String.html\_safe.

### Tasks

1. Create a migration to add profile\_bg, profile\_fg, and profile\_image fields to the users table (per the spec)
2. Make sure these attributes are accessible in the User model
3. Update the user form view partial to include text fields for the new fg/bg colors; also add a file field for profile\_image. Also hide the password\_digest field if the user is being edited. Use FormBuilder methods for this.
4. Add an attribute accessor for profile\_image\_file to user; this is what will contain the uploaded file instance
5. Update the users#edit action to upload the cover image (if present). It should live in the public/ directory. Also store this filename in the users.profile\_image field.
6. Update the users show action to use the user’s profile colors and display the profile image (if present). Also display user data (like email and username).
7. Add custom css to the users form so that it has a white background, 20px of padding, and a 3px solid #ddd border
8. Since we check/restrict the format of usernames, display the username wrapped in <u> tags using String.html\_safe

## Lesson 7 Lab

In this Lab, you will wire users and shouts together to learn about one to many associations. You will create a following system so that users can follow each other. You will also add tags to shouts, so that you can learn about many to many associations. You will also add a view of shouts for shouts from users you follow. You will update the user, shout, and (new) follow models accordingly.

### Tasks

1. Update the user and shout model associations so that they match the spec (a user has many shouts; a shout belongs to a user)
2. Display a paginated list of a user’s shouts on their profile (users#show)
3. Update the shouts#create method to associate the first user in the database with a shout (you’ll update this later). Use `User.first`
4. Display the shout’s user’s username in the list of shouts on a user’s profile as well as in the shouts#index view (make this a partial since you’re now displaying the same markup twice; you’ll also need to delete all existing shouts from the database -- use the Rails console)
5. Create a follow model per the project spec, then add a follow controller
6. Add a has\_many follower and follow association to User. Followers should be users following that user; follows should be *other* users the user follows
7. Add a method to the user model that let you check if another user is following them
8. Add custom, named routes for follows#create and follows#destroy
9. Add a follow/unfollow link on user profiles (Use `User.first` in place of a logged in user, which we’ll update later)
10. In the follows controller, implement following between two users per the spec
11. Update the shouts#index action so that passing from=followed as a URL parameter will only show shouts from users the current user follows (use User.first as a placeholder)
12. Create a Tag model and add a join table for shouts\_tags; Shouts and Tags should have a many to many relationship using has\_and\_belongs\_to\_many
13. Allow users to type single word tags when creating a shout, which should be translated into tag records; display these with the shout
14. Allow browsing shouts by tag name using a tags controller

## Lesson 8 Lab

In this Lab, you will learn about Rails exception handling, logging, the Ruby debugger, unit, functional, and integration testing, as well as write a few unit tests.

### Tasks

1. Add the **debugger** gem to your Gemfile and run `bundle install` (you’ll have to restart the Rails server once you do this)
2. Read about the debugger here: http://guides.rubyonrails.org/debugging\_rails\_applications.html
3. Pick 2-3 spots in your code to hook up to the debugger
4. Run the rails console in debug mode and experiment with some debugger commands
5. Add logger info output when a shout is created in the following format: “USERNAME just posted a shout: SHOUT TEXT”; Tail the development logs and look for this text
6. Add a fatal logger call if a shout is not saved; output the errors object
7. In the follows controller, make sure the code in the create and destroy methods is wrapped in a begin/rescue block; in the rescue branch of each block, write a logger debug message with the exception text; raise an exception to test this out
8. Comment out the Follow.create statement, and add a new statement without arguments; observe what happens when you try to save
9. Comment out the new line and enable the original create statement
10. Read about functional, unit, and integration tests: <http://guides.rubyonrails.org/testing.html>; check out users\_controller\_test.rb and understand how it works
11. Write a unit test for User.check\_reserved\_usernames to make sure it works as expected. If you get an exception running tests, fix it. (you’ll probably need to clear out the four yml files in test/fixtures first, then add a fixture named “one” in users.yml); Finally, run tests with `rake test`.
12. Get the rest of your test suite passing.

## Lesson 9 Lab

In this lesson, you will implement a simple authentication system with signup, login, logout, and secure passwords (using the bcrypt algorithm). You’ll refactor your existing code to utilize the logged in user, and add checks for a logged-in user. Before you start, delete all data in the database (call destroy\_all for User and Tag).

### Tasks

1. Uncomment the line in Gemfile requiring bcrypt-ruby, install gems, and restart the server
2. Add the has\_secure\_password helper method to your user model
3. Update your user form to display a field for password, instead of password\_digest; also update your validations and tests to reference password instead.
4. Add a sessions controller with a new, create, and destroy action; add named routes for these actions per the spec
5. Add a view for sessions#new that has a username\_or\_email and password field; it should POST to auth\_path
6. Implement create and destroy in the sessions controller. Create should find and authenticate a user by email/username + password and if successful, store their user\_id in the session. Destroy should reset the session.
7. Update users#create so that it stores the user id in session when successful (this will give you auto-login after signup)
8. Implement the follow helper methods in application controller:
   1. current\_user — this should return either nil or the User instance if a user is logged in
   2. logged\_in? — this should return true if logged in, false otherwise
   3. require\_user — this should redirect to login\_path unless someone is logged in
   4. require\_no\_user — this should redirect to root\_path if someone is logged in
9. Change the path for /users/new to /signup and sign up; add require\_user/require\_no\_user as before\_filters where required to implement the project spec (i.e. /login shouldn’t be available to already logged in users)
10. Change the path for users#show per the spec (/profile/USERNAME). Use the to\_param method in User to return the user’s username
11. Update code for creating a shout so that you are creating a shout which belongs to the logged in user
12. Read about authentication libraries (gems) like Devise and OmniAuth.

## Suggested Bonus Exercises

* Get users controller tests passing again (some will fail from the auth stuff/url changes)
* Allow deleting your own shouts
* Add a nav bar (login/logout/signup links, etc.)
* Add search for shouts and users
* Link up @usernames to their profile
* Auto-link urls in shouts
* Remember the last URL when redirecting to login (e.g. from require\_user)