|  |
| --- |
| TrialTracker |
| Making Clinical Research Easier |
|  |

# Setting Up a localhost

1) Our site uses Ruby on Rails. To run it on localhost you must have the following set up on your computer:

a. The Ruby programming language

b. Ruby Gems

c. A relational database, such as SQLite3.

d. Rails.

Getting this properly set up is rather involved, and the details are different depending on your local machine. Detailed instructions can be found at:

<http://guides.rubyonrails.org/getting_started.html>

2) Now that you have rails set up, download our source code. Navigate into the highest level folder (the first 2 items in it should be the app and config folders) and run the following commands to set up your database:

*rake db:reset* (wipes any current database you might have)

*rake db:migrate* (runs the rails’ migrate files to set up your db/schema file, which describes the database)

*rake db:setup* (creates the database. It also seeds the database. You can view and change your seeds in db/seeds)

3) Now run

*rails server*

Your server is live. Navigate in a browser to localhost:3000 to view the site.

# The Account

## Getting Started

Creating an account is simple. The splash page includes a registration box where one chooses a username, email, and password. Clients can login with either their username or their email.

Entering a wrong username or password will take the client to an error page. The client can click on “Forgot Password?”, enter their email, and receive an email with an updated (random) password.

Note that after a user is logged in, at all times the following session data must be set: session[:username], session[:userID], session[:current\_tab], and session[:current\_trial]. All these are initialized at moment of login.

# Implementation

From back-end to front-end:

SQL-lite database: Stores the following objects, each specified as a rails model:

Trial

Entry

User

Ruby-on-Rails framework:

Objects are defined in model classes. Each model has an associated controller class that is used to connect it to the database for things like: creating new objects, deleting objects, modifying existing objects.

routes.rb to map URL requests to controller methods.

Catch-all controller (home\_controller) that sets session variables and passes data to the views to be displayed.

Front end rails helper methods (such as form\_for) to generate html and link it to the routes file.

Javascript, Jquery, Google Charts: Our graphs are generated with google chart API (details below). Javascript and Jquery provide dynamic content and pretty effects like modal and fade-in.

HTML/CSS: We also use Twitter’s Bootstrap library liberally to help with styling. This includes Bootstrap’s Javascript functionality.

### Content

At all times the user is looking at a trial, and a tab within that trial.

The currently viewed trial is maintained in two places:

1. **session[:current\_trial]**, which should always be the integer ID of the trial being viewed.
2. **@current\_trial**, which should be the rails object representing the trial.

The currently viewed tab is maintained in **session[:current\_tab]**, which can be either ‘settings’, ‘edit\_data’, or ‘view\_trends’.

### Settings

The code for the settings page is in ‘views/home/\_settings.html.erb’.

Trial Form

This form is linked to the *update* method in *trials\_controller*. The form uses rails’ form\_for on @current\_trial, so that the controller can simply update\_attributes on the existing trial object.

Note: Here and in similar places, date fields return dates in dd/mm/yyyy format, while the database wants them in mm/dd/yyyy. Be sure to update the date’s format in the controller method before sending anything to the database.

Leaving the trial

By clicking the logout button, the routes takes the user to the splash page. The logout function exists in the home controller. In the login function, the session variables are all set to null so that they can be filled when another user logs in.

Administrator’s powers:

Inviting people by invite – The idea is to send an email containing trial code. This is taken care of in the home controller invite function. Using the mailer properties, the params[:email], and session[:current\_trial], we can invite users to trials of our choice. The mailer properties contain a single user\_mailer ruby file that contains multiple functions. Each of these functions sends out email. The function of interest here is invite\_new\_user.

Inviting people by trial code – This takes the trial code and stores it in params[:trial\_id]. With the join function in the trials controller, the corresponding user who has added the id will be added to the trial.

Removing people from the trial – Removing people from the trial is done by clicking the remove button next to them. This simply removes the user based on params[:id] from the Trial database.

### Edit Data

The code for the edit data page is in ‘views/home/\_edit\_data.html.erb’.

New Data

To create the first row, the row where a user can input a new line of data, we use rails’ form\_for on @entry. @entry is an empty entry object created at the end of *home\_controller’s index* method. For the convenience of the user, we populate this form with the data from @last\_entry, which is this trial’s most recent line of data. The new entry is created in *entries\_controller’s create* method.

Note: if there is already an entry for today, inputting an entry will erase the previous one from today.

The data is displayed by looping through @entries\_recent\_first, which is populated in *home\_controller’s index* method.

Delete

Delete calls *entries\_controller’s destroy* method, giving it the id of the entry to destroy.

Edit

Clicking edit calls the javascript function editEntry. This does two things: 1) it hides the row of the selected entry, and 2) it reveals a hidden row containing a form for that entry. On page load, for every entry in the table we create a hidden row with a form for that entry. This form *calls entries\_controller’s update* method, and uses rails’ update\_attributes.

### View Trends

Bar Chart

Google’s Bar Chart API:

<https://developers.google.com/chart/interactive/docs/gallery/barchart>

1) The bar chart is generated in view/home/index.html.erb, in the javascript function drawBarChart().

2) We draw a bar for each category in @categories, which is set in *home\_controller’s index* method. The bar’s size comes from @last\_entry- set in the same method.

Line Chart

Google’s Line Chart API:

<https://developers.google.com/chart/interactive/docs/gallery/linechart>

1) The data for this line chart is quite complicated. There are three possible lines: *progress, target* and *projected*, for a total of 8 possible graphs. Google’s API doesn’t allow any sophisticated way to handle this – so for every category we generate 8 possible data sets. They include the blank graph.

**Progress line**: loop through @entries\_oldestFirst. For each entry, x is the entry’s date and y is its count for that category. There is a point for every entry.

**Target line**: home\_controller’s index method makes a hash called @target\_averages. This maps enrolled and completed (the only categories with goals) to the target number the trial should be adding every day to be on target (goal/length of trial). There is a point for every day of the trial.

**Projected line**: home\_controller’s index method makes a hash called @averages. This maps each category to the average number of subjects marked per day in that category (latest entry total/length of trial). There is a point for every day of the trial.

2) Toggling between categories: On page load, the html is generated for every category. Clicking ‘enrolled’ activates the div for enrolled.

3) Toggling lines within a category is managed by toggling the buttons below the graph. Enrolled and Completed each have 3 lines and 3 buttons, while every other category has 2. Each button has a javascript onclick function tied to it. There is a counter which tracks the state of the three buttons at all times using the following abstraction: if the three buttons are each a bit where pressed means 1 and unpressed means 0, the counter is the integer represented. On clicking a button, the counter is updated and the appropriate graph is displayed using the draw function.

4) The table on the right is generated from @averages and data fields from @current\_trial. For example, Average/month is simply @averages \* 30.