**Useful Sites**

Ruby on rails guide - <http://edgeguides.rubyonrails.org/index.html>

Getting started, the basics, and installing rails: <http://guides.rubyonrails.org/getting_started.html>

For validations used on the models - <https://apidock.com/rails/v2.3.8/ActiveRecord/Validations/ClassMethods/validates_length_of>

For more validations - <http://guides.rubyonrails.org/active_record_validations.html>

Basics on queries in the rails console - <http://guides.rubyonrails.org/active_record_querying.html#ordering>

Faker Library - <https://github.com/stympy/faker#fakergameofthrones>

rspec documentation - <http://www.rubydoc.info/gems/rspec-expectations/file/README.md>

Self-referential associations (model stuff, basically joining a table to itself) - <http://railscasts.com/episodes/163-self-referential-association?view=asciicast>

creating polymorphic relationships in the models (db stuff) - <http://culttt.com/2016/01/13/creating-polymorphic-relationships-in-ruby-on-rails/>

Rails migration cheetsheat - <https://gist.github.com/amejiarosario/2950888>

**WORD DOCS TO LOOK AT**

* starting\_a\_project\_commands\_and\_refresher - for the basics
* rspec\_basics - for rspec basics, configurations, and examples
* rails\_console\_query\_examples\_and\_basics - exactly like it sounds
* using\_bootstrap\_with\_rails - to set up and configure bootstrap
* model\_basics\_validations\_and\_callbacks - exactly like it sounds

**Useful Notes**

* Once a controller is created, a folder in the views, and files in both the javascript and stylesheets folder in assets are also created. So if I create a users\_controller, there will be a “users” folder in views, a “users.css” file in stylesheets, and “users.js.cofee” in the javascripts folder. If you label a view file/page the same as a method, then rails automatically knows that they are linked. Ex: label a file “index.html.erb” and rails will know that it goes with the “index” method
* When querying (specifically creating), you should never reference the foreign key (id). For example: if you’re creating a post that is owned by a user and a blog, you should do something like this: Post.create(user: User.first, blog: Blog.find(2)) you shouldn’t do something like this: Post.create(user: User.first, blog\_id:2)

**Useful terminal commands (see starting new app for specifics on that)**

**rails s** - starts the server for a project at <http://0.0.0.0:3000/>

can shut it down with ctrl+c, can specify port by adding “ -e production -p 4000” after the “s” in rails s

**rails c** - opens rails interactive console

**Hirb.enable** - type this in rails console after installing the gem to make it easier to look at data from the db

**RAILS PROJECTS TO LOOK AS FOR REMINDERS**

**BASICS OF RAILS**

Open great\_number\_game in rails\_basics for the very basics, including

* example of session use
* routes page
* flash messages
* getting and using the paramaters passed in from url
* redirects in rails
* using flash messages in the view and basic forms
* if statement in the view (html)

Next, open survey\_form in rails\_basics for

* very basics of forms in rails
* getting the data from forms and using it in the controller
* passing the data from the form as a single chunk of data instead of individual pieces or fields (as an associative array)
* accessing the session data in the view, in the format of an associative array

Next, open integrating\_models\_controllers in rails\_basics for

* Basic look at models (db stuff)
* accessing the db in the controller
* example of plugging the db data into a variable in the controller than can be used in the view
* example of the faker gem for fast create
* using render json:

Next open management\_system the in rails\_basics folder for examples on

* partials
* full RESTful routes and controller operations
* form\_for
* very basic database validations (models folder in the user.rb page)
* if statements as validations for creating/updating data in the db
* and private params

Next open products\_dashboard in rails\_basics for

* a look at more complicated models and db
* basic relationships between different models (belongs\_to, has\_many, etc.)
* querying the db with tables that are connected and passing them to the view
* understanding and examples of how querying the db gets you an array of objects
* further use of strong params
* example of the each iterator being used on queried data
* examples of RESTful anchor tags, and how to specify the method (such as delete for the destroy route)
* using forms instead of form\_for
* using hidden inputs in the form to determine the method (post, patch, etc.)

**RAILS MODELS BASICS**

Open rails\_console\_query\_examples\_and\_basics word doc for a rundown

Open blogs\_posts\_messages.txt in Rails\_models for:

* examples on generating rails models from the terminal
* further examples of validations and relationships between models (can also see in the rails project of same name)
* examples of creating multiple rows for a model in the rails console at the same time using the Faker gem
* queries affected one model by going through another (ex: Blog.first.posts.create…..)
* more complex queries, such as using order (sort by basically), where, and update
* See how to use methods in the model, such as a method to delete all messages related to a post running automatically when you delete that post (can also see in the rails project of same name)

Next, open users.txt in Rails\_models for:

* slightly more complex validations in the model
* more queries in the console

Next, open blogs2 (both txt and project) in Rails\_models for:

* more complex model/db relationships, such as multiple references, and a many to many relationship (needs an intermediary table)
* queries with many to many tables/models

**RSPEC**

* Look at rspec\_basics word doc
* Then look at belt\_review\_events in full rails for the basics, examples, and comments

**ADVANCED RAILS**

See belt\_review\_events in full\_rails for:

* See how to use passwords, create models with them, hash them, etc.
* See how to create users with email validations, passwords, and how to log in with those users.
* see more advanced model validations, including email regex
* see more complicated relationships among your models and how to use has\_many :whatever, through: :table, source: :table
* See how to use helper methods in the controllers and how to use those helper methods with callbacks like “before\_action”
* using strong params with .merge
* example of staying logged in
* Using .strftime to format date, examples can be seen at <https://apidock.com/ruby/DateTime/strftime>
* .each and conditionals that compare query results in the view
* rspec basics, comments, configuration, etc.

**OTHER USEFUL INFO OR EXAMPLES**

Example of what the ‘resources’ gives you when used in the routes page

Ex from routes page:

Rails.application.routes.draw do

resources :users

More Examples, using more specific resources:

resources :sessions, only: [:new, :destroy, :create]

resources :events, except: [:index]

Ex:

| **Helper** | **HTTP Verb** | **Path** | **Controller#Action** |
| --- | --- | --- | --- |
| [**Path**](http://0.0.0.0:3000/userss)**/**[**Url**](http://0.0.0.0:3000/userss) |  |  |  |
| users\_path | GET | /users(.:format) | users#index |
|  | POST | /users(.:format) | users#create |
| new\_user\_path | GET | /users/new(.:format) | users#new |
| edit\_user\_path | GET | /users/:id/edit(.:format) | users#edit |
| user\_path | GET | /users/:id(.:format) | users#show |
|  | PATCH | /users/:id(.:format) | users#update |
|  | PUT | /users/:id(.:format) | users#update |
|  | DELETE | /users/:id(.:format) | users#destroy |