Unit 7: APIs, JavaScript, and Deployment

[Lesson 7.1: APIs and the Bus App](#h.2vig2cgnfeg)

[Gem of the Day](#h.8fm36kvjlaa9)

[What is an API?](#h.mvhqhz6gald1)

[MARTA App Plan](#h.w2vxzfgeskti)

[What’s Our App Idea?](#h.dh0pubaie41t)

[How Will We Make it Happen?](#h.wap52arle6fa)

[Latitude and Longitude Review](#h.wwags6dqvs7d)

[Implementation](#h.624uxmqu8fto)

[New Project](#h.shy1nkb2tmtz)

[Gems](#h.meosofe6j3av)

[Model/Scaffolds](#h.gqlq6gbnve3m)

[Helpers](#h.87kcwymsdy28)

[Controller](#h.uzcot06la495)

[Views](#h.uy4nmnk59fte)

[Bootstrap: Adding Manually Without a CDN](#h.yvf5x7acunz4)

[Homework](#h.qi6r1j9yq3pu)

[Lesson 7.2: JavaScript and JQuery](#h.wt4e21pfcbfl)

[Gem of the Day](#h.cl4cyyskr7a0)

[JavaScript](#h.432w3v5su6vj)

[JavaScript Language Basics](#h.fd18nkmu7gka)

[Variables](#h.8pzbnxvz3z4j)

[Data Types](#h.cxkb8o6zbqq0)

[Null and Undefined](#h.soybf27solvm)

[Scope](#h.hk1ih4hq4yr9)

[Operators](#h.7xol7crzq1qa)

[Conditionals](#h.rj3l0rv2t7q)

[Loops](#h.pol8e62o21l0)

[Functions](#h.p1o5dcf6h697)

[JavaScript in HTML](#h.ow8uuh10mgf)

[JQuery](#h.k452xcryo9x2)

[My First JQuery](#h.xzomxkbntw0c)

[The DOM, or Document Object Model](#h.uv4uhkdm0w56)

[CSS Selectors](#h.u1rudzeo5g58)

[Attributes, CSS, Events, and Animation/Effects](#h.udorodhpi40p)

[Homework](#h.d1o18z98kto)

[Resources](#h.2gz5vyr64vrv)

[Lesson 7.3: Let’s Map It! (secretly)](#h.d2m81n3o5v2j)

[Gem of the Day](#h.9hm1gvuv52ry)

[Hiding Secrets With Figaro](#h.fyk757t7fdxg)

[Google Maps API Key](#h.mgfv58sjmput)

[Implementing the Map With JavaScript](#h.it8kry35hih6)

[Basic Map](#h.cql1kiobmsi7)

[Center on User and Drop User Marker](#h.ren9i4o5ype)

[Add Bus Markers](#h.z0zc8deq6qqu)

[Change User Location Pin Icon](#h.uuny5f8rmqt4)

[Add Info Windows for Buses](#h.649umlgqnjaw)

[Required Homework - IMPORTANT!](#h.h6ongcli3r6n)

[Homework](#h.hw6hoy2g7g60)

[Lesson 7.4: Deploying with Heroku](#h.v50txl8329sa)

[Gem of the Day](#h.wolxsv5ssfdc)

[Objectives](#h.se1hbkpokrwc)

[What is Heroku and how does it fit into our app architecture?](#h.sqr1k5yn7y83)

[Our First Heroku App](#h.axpqmd3s3h3y)

[More Heroku Commands](#h.ukj4u04azj24)

[How to Delete a Heroku App](#h.w9j7ohtn6cjo)

[Instructions for Deploying Doggy Daycare](#h.t5hg1ztf8wbw)

[Dropbox Setup](#h.vq8uuco7lqqr)

[Heroku Deployment](#h.5p8xlg2liosw)

[Homework](#h.aj2zbata8t07)

[Lesson 7.5: AJAX](#h.q3bk55cgx21w)

[Gems of the Day](#h.vhg12fshm5ln)

[What is AJAX?](#h.pk8ylwgveejv)

[Intro Exercise: Blog](#h.j92te4x1lhkv)

[Task App](#h.8m4xg815dlrn)

[Rails Composer](#h.tc8zpnalfa9z)

[Adding Tasks](#h.kej5s3e98edw)

[Switching to AJAX: Controller](#h.xiwv25rcwpol)

[New Form Using AJAX](#h.heehy87mumya)

[Dynamically Updating Index](#h.tyxxi511e2o5)

[Updating a Task](#h.vf8e1r4pkt0v)

[Deleting a Task](#h.ks956vveukg8)

[Beautification](#h.nbdvxnczammd)

[Datepicker](#h.xwylv6tywsxl)

[Layout](#h.il2qa24ep98k)

[Fancy Deleting](#h.ih4oksl6pz4h)

[Homework](#h.vakftzj5soki)

# Lesson 7.1: APIs and the Bus App

## Gem of the Day

Geocoder <https://github.com/alexreisner/geocoder>

## What is an API?

API stands for application programming interface. It refers to a tool, or library, that assists developers in writing code that interfaces with other software. In short, it defines a way in which a computer program communicates with another computer program.

In practice, you could manually go to weather.com to look at the weather, or your app with an API could find the current weather in London by sending a message to the weather.com API (in a structured format like XML or JSON). The weather.com API then would reply with a structured response.

What is this “structured response”? Well, let’s take a look at the MARTA example - this is the response for all buses and their locations:

<http://developer.itsmarta.com/BRDRestService/RestBusRealTimeService/GetAllBus>

Structured responses are usually something like an array of hashes or a hash that contains a key for errors and a key for the array of hashes that represent the data object. Lots of free tools exist for copying in that response and returning a more readable version:

<http://jsonlint.com/>

<http://jsonformatter.curiousconcept.com/>

What is an API request? Well, if we send this URL, we are actually sending a request for the bus or buses that match the route “110”:

<http://developer.itsmarta.com/BRDRestService/RestBusRealTimeService/GetBusByRoute/110>

The request goes in the URL. Many APIs also require secure keys that you have to apply for so that their APIs don’t get bombarded by requests. You would store this key in an environment variable using a tool like Figaro so that no one could see it. Then your code would add that secure key to the URL request by only mentioning the environment variable.

Always read the API documentation for information on how their data is structured, what kind of requests (e.g., searches) you can make, and what are potential limits.

## MARTA App Plan

### What’s Our App Idea?

We want to make an app that will take a user’s address or street corner and output “nearby” buses as well as their next stop location.

### How Will We Make it Happen?

(Walk through this on board)

**What components do we need?**

* user’s address
* a way to convert the user’s address to exact longitude and latitude
* real-time bus information (from API)
* a comparison of user location and all bus locations to only return nearby ones

**What are the steps?**

1. Add the gem Geocoder which will take a user’s full approximate address and return his or her latitude and longitude.
2. Create a user’s location table with their approximate street address, city, latitude, and longitude. (state will always be Georgia)
3. In the location model, create a method which returns my\_location that includes that approximate street address, city, and state which is then passed to geocoder.
4. Update controller to loop through buses and only return those nearby, but use helper methods to make this code shorter...
5. Create helper methods
   1. fetch and parse data from API for all buses
   2. determine if a bus qualifies as nearby
6. Output nearby buses to view
7. Bootstrap using actual files

**What Data is in the Real-Time Bus Tracker?**

<http://www.itsmarta.com/developers/data-sources/marta-bus-realtime-restful-api.aspx>

* ADHERENCE: how late/early the bus is (in minutes)?
* DIRECTION: which cardinal direction is the bus headed?
* LATITUDE/LONGITUDE: exact coordinates of the bus!
* ROUTE: the route number, obvi.
* TIMEPOINT: the bus' next stop.
* VEHICLE: the ID number of that particular bus.

### Latitude and Longitude Review

First, let’s take a second to remind ourselves about latitude and longitude and distances of degrees.

<http://geography.about.com/library/faq/blqzdistancedegree.htm>

Now, let’s figure out what the length of a degree of latitude and longitude are in Atlanta:

<http://www.csgnetwork.com/degreelenllavcalc.html>

Atlanta’s latitude is about 33.7 degrees, which means 1 degree is equal to about 58 miles in longitude and the standard 69 miles in latitude.

Let’s define nearby as within about a 0.64 mile radius, which we could approximate as 0.01 degrees of either (it will be more elliptical)

## Implementation

### New Project

Let's create a new project, with one resource: Location

rails new marta\_near\_me  
cd marta\_near\_me

git init and do first commit

### Gems

Let’s add our gems now:

gem 'geocoder'

Bundle install, and commit changes.

### Model/Scaffolds

We are going to scaffold a resource (table) called Location  
Remember, for Geocoder to work we need the attributes of latitude and longitude, both as floats.  
rails g scaffold Location address:string city:string latitude:float longitude:float

rake db:migrate

Now, let’s edit our location model based on what the Geocoder gem needs. We are going to create a location called my\_location for geocoder to use that is built off of the address, city, and hard-coded state:

class Location < ActiveRecord::Base

geocoded\_by :my\_location

after\_validation :geocode # auto-fetch coordinates

def my\_location

"#{address}, #{city}, GA"

end

end

Commit your changes!

### Helpers

Let’s review first - what exactly is a helper? Helpers are used to generate code for our views, and to assist the models and controllers. Rails encourages “creating custom helpers to extract complicated logic or reusable functionality.” This helps keep our code small, focused, and uncluttered.

Let’s go ahead and make our helpers inside the Location Helper:

module LocationsHelper

# Parse the API data to store it in an array of hashes - each bus is a hash.

def fetch\_api\_data source

http = Net::HTTP.get\_response(URI.parse(source))

data = http.body

JSON.parse(data)

end

# Compare latitude/longitude of the user and all the buses to see if they are

# within 0.01 degree

def is\_nearby(lat\_user, long\_user, lat\_bus, long\_bus)

(long\_user - long\_bus).abs <= 0.01 && (lat\_user - lat\_bus).abs <= 0.01

end

end

### Controller

In the show controller, we will ping our API to grab a list of nearby buses and output them to an array. But first, we have to give them access to the Location Helper:

class LocationsController < ApplicationController

include LocationsHelper # use name of module from that file

…

def show

# MARTA API URL

source = 'http://developer.itsmarta.com/BRDRestService/BRDRestService.svc/GetAllBus'

# Use a helper method to parse the data into an array of hashes for all

# buses in system

@buses = fetch\_api\_data(source)

# Loop through all buses in system to find those that are close by and put

# them in the nearby buses array.

@nearby\_buses = []

@buses.each do |bus|

if is\_nearby(@location.latitude, @location.longitude, bus['LATITUDE'].to\_f, bus['LONGITUDE'].to\_f)

@nearby\_buses.push(bus)

end

end

@bus\_count = @nearby\_buses.length

# TODO: if no buses, return with notice and redirect to new

end

### Views

We want our landing page to be the new view for a location, so let’s set our **root** there in the config/routes file:

root 'locations#new'

**New Location**

Now, let’s edit the new location view:

<h1>Bus Me!</h1>

<p>Give us your address or corner, and we'll let you know if there is a bus nearby!</p>

<%= render 'form' %>

<%= link\_to 'Back', locations\_path %>

**Form**

Let’s also edit the form to be a bit more clear about what we want while also deleting the latitude and longitude fields since Geocoder will handle those for us:

<div class="field">

<%= f.label "Address or Approximate Location" %><br>

<%= f.text\_field :address %>

</div>

<div class="field">

<%= f.label :city %><br>

<%= f.text\_field :city %>

</div>

<div class="actions">

<%= f.submit "Find My Bus!" %>

</div>

**Showing the Buses**

Let’s show all our nearby buses in the Location Show view:

<p id="notice"><%= notice %></p>

<h2>You are currently standing at...</h2>

<p>

<%= @location.my\_location %>

</p>

<h2>The closest buses are...</h2>

<% if @bus\_count == 0 %>

<p>...not really that close. Time to walk, bike, taxi, or Uber.</p>

<% end %>

<% @nearby\_buses.each do |bus| %>

<p>

<strong>Route:</strong> <%= bus["ROUTE"] %><br>

<strong>Bus Number:</strong> <%= bus["VEHICLE"] %><br>

<strong>Next Stop:</strong> <%= bus["TIMEPOINT"] %><br>

</p>

<% end %>

<%= link\_to 'New Search', edit\_location\_path(@location) %>

Now, test the app! Try an actual intersection like Ellis St NE and Courtland St NE in Atlanta. Then, try switching the city to Savannah to make sure the logic works. Finally, commit your changes!

## Bootstrap: Adding Manually Without a CDN

Download Bootstrap and JQuery-UI files. Copy un-minified versions to your assets.

## Homework

How could you make the nearby bus calculation be a circular radius? Implement this and provide the user the option of increasing/decreasing that radius (say 0.2, 0.5, and 1 mile options).

If no buses are nearby, don’t go to the show view. Redirect back to the new view with a notice that no buses were found nearby.

Add bus direction to the output.

MARTA only runs in a select amount of cities - give the User a dropdown of cities to choose from. Maybe make it default to Atlanta.

# Lesson 7.2: JavaScript and JQuery

## Gem of the Day

RuboCop/RefactorCop <http://refactorcop.com/>

## JavaScript

We're going to cover the basics of JavaScript: how to create variables and functions, write loops and conditionals, and including scripts in your HTML page. We'll also cover jQuery, the most popular JavaScript library on the web today.

JavaScript was developed by Netscape in the early 1990s as a way to provide interactivity in web pages. Before JavaScript, web pages were essentially static documents: once you loaded a page, the content didn't change.

Since then, JavaScript has become standardized and evolved tremendously. You can develop entire applications, from the server side to the client side, using only JavaScript.

## JavaScript Language Basics

Double-check that you have node.js installed - run node -v on your command line and confirm that it returns a version (if no version, use Cloud9 or Nitrous as your IDE for now). Then, go to your front-end folder, and create a new file called practice.js. Inside that file, let’s write a very basic program:

var message = "Hello, World!";

console.log(message);

Then, on the command line, type node practice.js. You should see your message print out to the command line like when we used “puts” in Ruby.

### Variables

Creating a variable in JavaScript is called "declaring" a variable. You declare a JavaScript variable with the var keyword (won’t work until we use “var”!). Just like Ruby, JavaScript is an untyped language, so any variable can hold any value at any time.

var x = "Hey";

var y = 10;

console.log(x);

console.log(y);

console.log(x + y);

y = " is for Horses!";

console.log(x + y);

You can declare many variables in one statement:

var x = 1, y = 2, z = "3";

### Data Types

Like Ruby, JavaScript can handle integers, floats, strings, booleans, arrays, and hashes. Actually, because of the nature of JavaScript, every object in JS is in fact a hash. The hash data type just let’s you do the more typical hash methods.

// Numbers

var integer\_num = 1;

var float\_num = 1.23;

console.log(integer\_num + float\_num);

// Strings

var message = "Let's learn JavaScript!";

console.log(message);

// Booleans

var is\_cool = true;

console.log(is\_cool + " dat");

// Arrays

var my\_stuff = [integer\_num, message, is\_cool];

console.log(my\_stuff);

// Hashes

var capitals = {

LA: "Baton Rouge",

TX: "Austin",

GA: "Atlanta"

};

console.log(capitals["LA"]);

### Null and Undefined

JavaScript has two different concepts of “emptiness” - null and undefined:

var x;

// Variables that haven't been initialized are undefined.

console.log(x); // undefined

// But they are not null:

console.log(x === null); // false

// Unless you make them null:

x = null;

console.log(x); // null

console.log(x === null); // true

console.log(x === undefined); // false

// Or use 'type coercion' - 2 equal signs will change the value rather than

// simply compare in Javascript

console.log(x == undefined); // true

### Scope

In JavaScript, objects and functions are also variables. **Scope** is the set of variables, objects, and functions you have access to. JavaScript has **function scope**: The scope changes inside functions.

**Local JavaScript Variables**

Variables declared within a JavaScript function, become LOCAL to the function. Local variables have local scope: They can only be accessed within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.

**Global JavaScript Variables**

A variable declared outside a function, becomes GLOBAL. A global variable has global scope: All scripts and functions on a web page can access it.

**Automatically Global**

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable. This is why you should always declare variables with var to avoid overwriting a global variable with the same name.

// This variable is in the Global scope:

var x = "I'm a global variable called x!";

console.log(x);

// Defining a function called someFunction

function someFunction(){

// This variable only exists inside the function:

var y = "I'm a local variable called y!";

console.log(x);

console.log(y);

// This is bad. Don't do it.

x = "I'm now a BAD local variable called x."

console.log(x);

}

// Calling someFunction

someFunction();

console.log(x); // Returns BAD local x

console.log(y); // Results in an error

### Operators

JavaScript has arithmetic and logical operations, like Ruby:

var x = 10,

y = 5;

console.log(x + y);

console.log(x - y);

console.log(x \* y);

console.log(x / y);

console.log(x % y); // modulus, or remainder

console.log(x > y);

console.log(x < y);

console.log(x === y); // is equal to

console.log(x !== y); // is not equal to

var a = true,

b = false;

console.log(a && b); // and

console.log(a || b); // or

### Conditionals

JavaScript conditional statements are similar to Ruby conditionals, but you need parentheses and curly brackets, and it's else if, not elsif:

var x = 10;

y = 5;

if (x > y){

console.log("x is greater than y");

} else if (x < y){

console.log("x is less than y");

} else {

console.log("x is equal to y");

}

### Loops

JavaScript looks like C when it comes to loops, though modern browsers also support a cleaner forEach loop:

// C-style JavaScript loop

var x = [1, 2, 3, 4, 5];

for (var i = 0; i < x.length; i++){

console.log(x[i]);

}

// For each loop

x.forEach(function (element){

console.log(element);

});

// While loop

var numbers = [1, 2, 3, 4],

i = 0;

while (i < numbers.length){

console.log(numbers[i]);

i++;

}

### Functions

JavaScript functions are like Ruby methods, but they have a few more tricks.

* They create their own local scope.
* They can be stored in variables and passed to other functions, just like any other data type.
* They create closures, which means they keep their surrounding scope with them as they get passed around.
* They can receive an arbitrary number of parameters, regardless of how the function is declared.

Earlier when we talked about scopes, we saw how to define a function and call it. Let’s also learn how to pass some arguments to a function:

// A function with explicit arguments

function sumExplicitly(a, b){

console.log(a + b);

}

sumExplicitly(10, 5); //=> 15

// A function with implicit arguments

function sumImplicitly(){

var total = 0,

i;

for (i = 0; i < arguments.length; i++){

total += arguments[i];

}

console.log(total);

}

sumImplicitly(1, 4, 7, 3, 20); //=> 35

Functions can return values to be used elsewhere. A very important difference to Ruby is that if you don't have a return statement a function, then its return value is undefined.

function sum(a, b){

return a + b;

}

var x = sum(10, 5);

var y = sum(x, sum(20, 30));

console.log(x); //=> 15

console.log(y); //=> 65

As you can see from this last example, you can pass functions as arguments into other functions.

## JavaScript in HTML

When we use JavaScript in HTML, we include it inside <script></script> tags. You can technically place <script> tags anywhere in the<head> or <body> element, but there are a few caveats:

* Scripts block further rendering of the document while they execute, so it's generally a good idea to load them near the end of the <body>.
* Scripts that depend on other scripts need to be loaded in the correct order — e.g., load the jQuery library before your script that uses jQuery.

What’s cool about JavaScript is that console.log is still available to you when we start using the actual browser - find it in DevTools under the Console tab.

<!DOCTYPE html>

<html>

<head>

<title>Practice JavaScript</title>

</head>

<body>

<script type="text/javascript">

alert("It worked!");

console.log("The console still works too!");

</script>

</body>

</html>

## JQuery

jQuery is a fast and concise JavaScript library created by John Resig in 2006 with a nice motto: Write less, do more. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

Features include:

* DOM manipulation
* Event handling
* Asynchronous requests (Ajax)
* Animations
* Cross-browser compatibility
* Light weight

Just like Bootstrap, you can either download jQuery and include it locally in your project, or you can use the version hosted by their CDN. We'll use the CDN version, so add this to your page just above the existing <script> tag:

<script type="text/javascript" src="<http://code.jquery.com/jquery-1.11.3.min.js>"></script>

### My First JQuery

Let’s write a bit of code first, then talk through it:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Practice</title>

<script type="text/javascript" src="http://code.jquery.com/jquery-1.11.3.min.js"></script>

</head>

<body>

<div id="my-element">HI</div>

<script type="text/javascript">

$("#my-element")

.show()

.addClass("selectable")

.click(function (e){

alert("You clicked something!");

})

console.log("hi");

</script>

</body>

</html>

Almost everything you'll do in jQuery is implemented through the $() function. You'll also frequently chain function calls together.

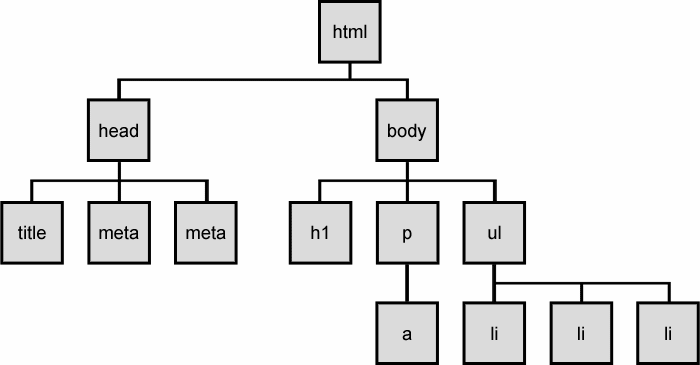
For example, we did these three things to the div element with an ID of “my-element”:

* showed a particular element on the page
* added a CSS class name to it, and
* attached a click event handler to it

### The DOM, or Document Object Model

jQuery's primary purpose is interacting with the HTMLdocument object model (DOM). The DOM is the representation of your HTML document in your computer's memory. You’re already familiar with the DOM though we haven’t really used that terminology yet...

(draw this on board):



The DOM is referred to as a tree since it has many branches. Some terms we need to be familiar with (draw on board with previous image):

* node: a single element in the tree
* root: the top-most node in the tree
* parent: the node exactly one level above me
* child: any node exactly one level below me
* sibling: any node at my level
* ancestor: a node at any level above me
* descendant: a node at any level below me

### CSS Selectors

JQuery uses CSS selectors (who can remember the 3 ways we reference our elements from CSS?)

$("a") //=> all <a> elements  
  
$("li a") //=> only <a> elements that are  
 //=> descendants of <li> elements  
  
$(".special") //=> all elements with a class of "special"  
  
$("#heading") //=> the element with an id of "heading"

### Attributes, CSS, Events, and Animation/Effects

JQuery can manipulate attributes - let’s add some more to our code:

$("#my-element")

.show()

.addClass("selectable")

.click(function (e){

alert("You clicked something!");

})

// Get an attribute's value

var id = $("div").attr("id");

console.log(id); //=> "my-element"

// Set an attribute's value

$("div").attr("id", "your-element");

Now, refresh your page and inspect the div element. What is the id of it? Here are some more [JQuery Attributes](http://api.jquery.com/category/attributes/).

JQuery can also manipulate CSS classes. Let’s play with some [CSS manipulation](http://api.jquery.com/category/css/) (do each one-by-one):

// Set an attribute's value

$("div").attr("id", "your-element");

// Add a class

$("div").addClass("special");

// Remove a class

$("div").removeClass("special");

// Add or remove a class, depending on whether

// it's currently present

$("div").toggleClass("special");

// Check whether a class exists

if ($("div").hasClass("special")){

alert("It's special alright!");

} else {

alert("It's normal.");

}

Now let’s play with [Events](http://api.jquery.com/category/events/) by making our div show the div’s text whenever it is clicked (delete the previous script):

$(function (){

$("#my-element").click(function (e){

var paragraphText = $(e.target).text();

alert(paragraphText);

});

});

Finally, let’s play with Animation, or [Effects](http://api.jquery.com/category/effects/), by making our div slide up and away when we click on it:

$(function (){

$("#my-element").click(function (e){

$(e.target).slideUp();

});

});

Did you notice how in the last 2 examples, we used $(function (){});? Since JavaScript frequently manipulates elements in the DOM, you need to wait until those elements are rendered before you try manipulating them. By including our JQuery code inside that function, it will know to wait until the DOM elements are rendered.

## Homework

1. Create a page that hides all of its <p> tags after 5 seconds. You'll want to use [delay()](http://api.jquery.com/delay/) and [hide()](http://api.jquery.com/hide/).
2. Create a button that shows a [Bootstrap Modal](http://getbootstrap.com/javascript/#modals) when you click it. They list several ways to do it, so try the one under the Via JavaScript heading.

## Resources

JQuery documentation: [JQuery Attributes](http://api.jquery.com/category/attributes/), [CSS manipulation](http://api.jquery.com/category/css/), [Events](http://api.jquery.com/category/events/), [Effects](http://api.jquery.com/category/effects/)

JSFiddle: <http://jsfiddle.net/>

# Lesson 7.3: Let’s Map It! (secretly)

Our Bus Me app is great, but it would be even better if we could show a map with the user’s location and the buses near him or her.

## Gem of the Day

Pagination! <https://www.ruby-toolbox.com/categories/pagination>

* Will Paginate <https://github.com/mislav/will_paginate>
* Kaminari <https://github.com/amatsuda/kaminari>

## Hiding Secrets With Figaro

Figaro <https://github.com/laserlemon/figaro>

First, let’s create a new **branch** called “maps” where we will work for today.

Then, let’s add the Figaro gem:

gem 'figaro'

bundle

Then, we have to explicitly install per the readme:

figaro install

Now check your config folder - you should have a new file called application.yml which was also added to your .gitignore file so no one will ever see it - it won’t push anywhere else.

## Google Maps API Key

Now, let’s request an API key.

1. Go to <https://developers.google.com/maps/web/>
2. Click “get started with the javascript api”
3. Follow the directions to get your key.

Once you have your API key, add it to your **application.yml** file like so:

google\_maps\_api\_key: iAmnOtaReAlkEy12FrIdAy

Then, add the script link to the Google Maps API in the HEAD of your HTML:

<!DOCTYPE html>

<html>

<head>

<title>PreBusMe</title>

<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?key=<%= ENV['google\_maps\_api\_key'] %>"></script>

<%= stylesheet\_link\_tag 'application', media: 'all', 'data-turbolinks-track' => true %>

<%= javascript\_include\_tag 'application', 'data-turbolinks-track' => true %>

<%= csrf\_meta\_tags %>

</head>

When we deploy to Heroku, we will have to set this environment variable manually inside Heroku - instructions are [here](https://github.com/laserlemon/figaro#deployment) for future reference.

## Implementing the Map With JavaScript

### Basic Map

First, let’s attempt to put a basic map on our page - here is the simple example from the API documentation - put it in your show view above the loop, and see if it works:

<div id="map-canvas" style="height:500px;width:500px;margin:0;padding:0;"></div>

<script type="text/javascript">

function initialize() {

var mapOptions = {

center: { lat: -34.397, lng: 150.644},

zoom: 8

};

var map = new google.maps.Map(document.getElementById('map-canvas'),

mapOptions);

}

google.maps.event.addDomListener(window, 'load', initialize);

</script>

### Center on User and Drop User Marker

Cool! Now, let’s make it more interesting by making it centered on the user’s location and drop a marker for that user. While we are at it, let’s go ahead and put the script tag at the bottom of the file, after all the html and erb.

To do this, we need to set a new variable for the user’s latitude and longitude. You can see how to do this with the Google Maps API [here](https://developers.google.com/maps/documentation/javascript/tutorial#latitudes-and-longitudes). Then, we edit the mapOptions to center on that location. Finally, we drop a marker. See more about markers [here](https://developers.google.com/maps/documentation/javascript/markers).

function initialize() {

var myLatlng = new google.maps.LatLng(<%= @location.latitude %>, <%= @location.longitude %>);

var mapOptions = {

center: myLatlng,

zoom: 15

};

var map = new google.maps.Map(document.getElementById('map-canvas'),

mapOptions);

var myMarker = new google.maps.Marker({

position: myLatlng,

map: map,

animation: google.maps.Animation.DROP,

title: 'My Location'

});

}

google.maps.event.addDomListener(window, 'load', initialize);

### Add Bus Markers

How might we add buses to the map? With a loop. However, first we need the Ruby @nearby\_buses to be passed into a JavaScript array variable. Luckily, ERB and Rails have a fairly simple command for this:

var nearbyBuses = <%= raw @nearby\_buses.to\_json %>;

Now, we can loop through that buses object to grab the latitudes and longitudes, then drop markers, just like we did for the user location:

nearbyBuses.forEach(function (element){

var busLatlng = new google.maps.LatLng(element['LATITUDE'], element['LONGITUDE']);

var marker = new google.maps.Marker({

position: busLatlng,

map: map,

animation: google.maps.Animation.DROP,

title: element['VEHICLE']

});

});

### Change User Location Pin Icon

Nice! But, this is getting a little confusing since we can’t tell the difference between the user’s location and the bus locations. Let’s change the icon for the user. I’ll send you this icon to use for now, but feel free to change this icon in your homework.

var personIcon = "<%= image\_path 'man\_icon.png' %>";

var myMarker = new google.maps.Marker({

position: myLatlng,

map: map,

animation: google.maps.Animation.DROP,

title: 'My Location',

icon: personIcon

});

### Add Info Windows for Buses

You know what would be even better? If we could click on a bus marker and get that bus’ information. We can do that with [infowindows](https://developers.google.com/maps/documentation/javascript/infowindows).

First, we have to define the content for that window in html and put it in a variable:

var contentString = '<div id="content">'+

'<h2>Bus '+ element['VEHICLE']+' '+element['DIRECTION']+'</h2>'+

'<div id="bodyContent">'+

'<p>Next Stop: '+ element['TIMEPOINT'] +'</p>'+

'</div>'+

'</div>';

Then, we have to create the info window:

var infowindow = new google.maps.InfoWindow({

content: contentString

});

Finally, we have to add the event listener:

google.maps.event.addListener(marker, 'click', function() {

infowindow.open(map,marker);

});

Classroom challenge: Add an info window for the user marker that tells them their address.

## Required Homework - IMPORTANT!

1. Sign up for an account on <https://www.heroku.com/> and confirm your account through the email they send you.
2. Download the Heroku Toolbelt. You can get it from the Setup page of the Ruby tutorial for Heroku: [https://devcenter.heroku.com/articles/getting-started-with-ruby#set-up](https://devcenter.heroku.com/articles/getting-started-with-ruby" \l "set-up)
3. Test to make sure Heroku was installed correctly by opening your command line and typing heroku login and log in using your credentials. You should get something like this:  
   

## Homework

Continue optimizing your Bus Me app and making the Google Maps integration even better. Clean up the Show view especially. Add fake social media links, as if this were a real app. Also add a link to your repo on GitHub. We are going to share it with the world tomorrow!

# Lesson 7.4: Deploying with Heroku

Share out any cool features students added to their Google Maps for Bus Me.

## Gem of the Day

Dashing - cool dashboards! <https://github.com/Shopify/dashing>

## Objectives

* Understand how Heroku and Git are connected
* Understand all steps to successfully push an app to Heroku
* Know how to delete an app on Heroku (only 5 free)

## What is Heroku and how does it fit into our app architecture?

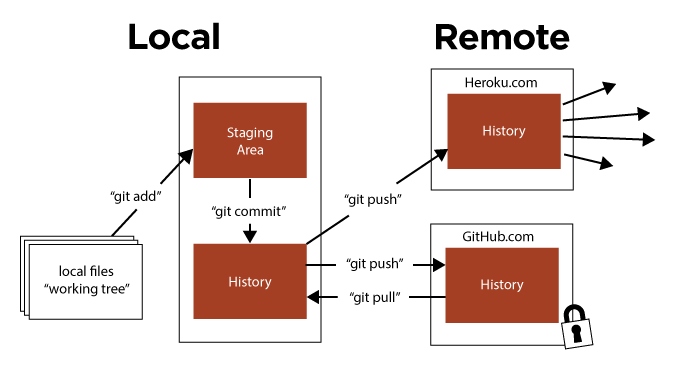
Heroku is a scalable cloud platform service for hosting your web applications.

* **Easy Deployment** - You can instantly deploy your app from the command line using a git push command.
* **Isolation** - Heroku uses “containers” called dynos that allow multiple isolated operating systems/processes to run on a shared host. A process in one container cannot see anything about another process in another container.
* **Scalability** - You can scale your app to handle more concurrent requests by adding more dynos or to handle more information at a faster speed (more RAM/CPU) with bigger dynos.
* **Full Logging and Visibility** - easy access to all logging output from every component of your app and each process (dyno)

Competitors to Heroku include:

* Amazon AWS
* Google App Engine (no Rails though)
* DigitalOcean (cheaper but more work)
* Linode (mid-range on price)
* and many others!

**How does Heroku fit in?**  Draw this on board:



## 

## Our First Heroku App

Let’s push Bus Me to Heroku.. The app must have it’s own git repository - not combined with any other app.

What files do we probably need to change if we want to change the production db? (Answer: gemfile, database.yml)

Steps :

1. Change **gem file** to use postgres (gem 'pg') in production and gem ‘sqlite’ only in dev and test. NOTE that Heroku doesn’t like using different servers. Also need to add to production gem 'rails\_12factor'. bundle install. Commit.
   1. If you get a bundle error related to pg, run this bundle command which says to ignore production gems:  
      bundle install --without production
2. Update **config/database.yml** file to replace everything in production with:   
   production:  
    <<: \*default  
    adapter: postgresql
3. Commit.
4. Once your app is ready to be released to production (db’s are migrated, everything is working, you’ve committed all changes) continue with heroku login
5. heroku create (and note your randomly generated app name and website)
6. git push heroku master (show diagram again)
   1. Purposefully attempt heroku open now. Ask how many got “can’t find page” (rake routes) vs “Something went wrong” (no db migrate)
   2. Get everyone to where something went wrong - ask for guesses as to what went wrong
7. If you have a database, you will need to migrate your db on heroku with heroku run rake db:migrate
8. Set any environment variables you may have created using [Figaro](https://github.com/laserlemon/figaro#deployment):  
   figaro heroku:set -e production
9. (optional) Ensure that at least one instance of the app is running: heroku ps:scale web=1
10. Open (or refresh) your app with the nifty heroku open (opens your live app in the browser!)

### More Heroku Commands

heroku logs -t

heroku config:add TZ=America/Chicago

Each time you make changes to your project, commit them as usual. Then, when you’re ready to release them to production on Heroku, run git push heroku master.

If you want to rename your app, it’s easiest to do from the command line. See [these instructions](https://devcenter.heroku.com/articles/renaming-apps).

This is a great tutorial - use it with their files, or use it as a general guide with your own app:

<https://devcenter.heroku.com/articles/getting-started-with-ruby>

## How to Delete a Heroku App

Go to heroku.com and log in. You will see your dashboard. Click on the app you no longer want. Click on Settings, scroll down, and then click on Delete app and continue with the instructions provided to confirm the delete.

## Instructions for Deploying Doggy Daycare

We used Paperclip for Doggy Daycare, which means we need to set up a service to host our images. You can use Amazon S3, Dropbox, or some other solutions. These instructions will cover using Dropbox. You can learn more about both options in the Paperclip readme, or specifically for [Dropbox](https://github.com/janko-m/paperclip-dropbox) and [Amazon S3](http://www.rubydoc.info/gems/paperclip/Paperclip/Storage/S3).

### Dropbox Setup

Before we can use Dropbox to host our photos, we need to set up a Dropbox Platform App. First, decide whether you want to use Full Dropbox access (public folder) or a special App Folder. Read about the differences [here](https://github.com/janko-m/paperclip-dropbox/wiki/Access-types). The rest of these instructions are for doing the App Folder.

1. Go to <https://www.dropbox.com/developers/apps/create>
2. Select Dropbox API App, select ‘yes’, give it a name, and submit.
3. Add gems to gemfile, and bundle (or bundle install --without production):

gem "paperclip-dropbox", ">= 1.1.7"

gem 'figaro'

group :production do

gem 'pg'

gem 'rails\_12factor'

end

group :development, :test do

gem 'sqlite3'

...

1. Create a new file called dropbox.yml in your config folder, and inside it, put the below content but replace app\_key and app\_secret (as well as make sure app\_folder is the access type):

app\_key: "..."  
app\_secret: "..."  
access\_token: "..."  
access\_token\_secret: "..."  
user\_id: "..."  
access\_type: "app\_folder"

1. Now we need to do a rake task to save our app key and secret which will authorize our app and provide the other 3 missing fields for the above file. Type this command in command line, but replace with your app key and secret, not in quotes:

rake dropbox:authorize APP\_KEY=your\_app\_key APP\_SECRET=your\_app\_secret ACCESS\_TYPE=app\_folder

1. Once you hit enter, it will tell you to go to a specific url to authorize your app. Do this, then hit “y”. It will spit back your access token, secret, and user\_id. Fill out the rest of **config/dropbox.yml** with that info.
2. Now, run figaro install to get your application.yml file. Copy everything in dropbox.yml over to your application.yml. Then, make your dropbox.yml file look like this - this will make it easier to push to production using Figaro’s capabilities:

app\_key: <%= ENV['app\_key'] %>

app\_secret: <%= ENV['app\_secret'] %>

access\_token: <%= ENV['access\_token'] %>

access\_token\_secret: <%= ENV['access\_token\_secret'] %>

user\_id: <%= ENV['user\_id'] %>

access\_type: <%= ENV['access\_type'] %>

1. Once your secrets are no longer in dropbox.yml, you are free to commit your changes!
2. Now, in both our **dog** and **products** **models**, we need to append some code to our **has\_attached\_file** validation (put a comma after the default file location item, hit enter, then paste this):

:storage => :dropbox,  
 :dropbox\_credentials => Rails.root.join("config/dropbox.yml")

1. It’s probably safest to rake db:drop and then both re-migrate and seed. You’re welcome to attempt skipping this step though in production you will kind of have to do this anyway since the database itself doesn’t port to production.
2. Run your server to test that you can add a photo to a dog or product. So cool. Now commit your changes.

### Heroku Deployment

1. Update **config/database.yml** file to replace everything in production with:   
   production:  
    <<: \*default  
    adapter: postgresql
2. Commit.
3. Once your app is ready to be released to production (db’s are migrated, everything is working, you’ve committed all changes) continue with heroku login
4. heroku create (and note your randomly generated app name and website)
5. git push heroku master
6. Migrate your db on heroku with heroku run rake db:migrate, then seed it with heroku run rake db:seed.
7. Set any environment variables you may have created using [Figaro](https://github.com/laserlemon/figaro#deployment):   
   figaro heroku:set -e production
8. Open (or refresh) your app with the nifty heroku open (opens your live app in the browser!)

## Homework

Try to deploy another one of your apps - especially a special project that you’ve been working on. Clean up your apps and add them to your portfolio page!

# Lesson 7.5: AJAX

## Gems of the Day

* Rails Composer <https://github.com/RailsApps/rails-composer>
* Simple Form <https://github.com/plataformatec/simple_form>
* Bootstrap Datepicker Rails <https://github.com/Nerian/bootstrap-datepicker-rails> - even has a [sandbox](https://github.com/Nerian/bootstrap-datepicker-rails) to build your datepicker JS code

## What is AJAX?

AJAX stands for Asynchronous JavaScript and XML.

* ASYNCHRONOUS means that the client can request new pieces of information from the server at ANY TIME
* This means content on a page is updated without having to re-render the entire page, making it a “seamless” experience.

## Intro Exercise: Blog

Create a new rails project called ajax-sample inside your rails-practice folder.

$ rails new ajax-sample

$ cd ajax-sample  
$ rails g scaffold Post content:text  
$ rake db:migrate

Create a method inside your Posts controller called ajax

# posts\_controller.rb  
def ajax  
end

Create the corresponding ajax view and route.

# routes.rb  
root 'posts#ajax'

<!-- ajax.html.erb -->  
<h1>Ajax Testing 1, 2, 3</h1>

<!-- This div element will be replaced by actual posts. First, we just put a link to the JavaScript function that will do that behavior. -->

<div id="posts">

<a href="javascript:loadPosts();">This is where blog posts go</a>

</div>

<!-- Here is the JavaScript function that actually loads posts once it's initiated by clicking the hyperlink above. -->

<script type="text/javascript">

function loadPosts(){

// Go grab the JSON data which contains our posts. Rails creates the JSON version of our data for us (index.json.jbuilder).

$.getJSON("/posts.json", function(data){

// Put the post data in an HTML string to make it bold and italic

var html = "";

$.each(data, function (index){

// alert(data[index].content);

html += "<p><strong><em>" + data[index].content + "</em></strong><p>";

}); // close .each

// update our div with the content of the var html

$("#posts").html(html);

}); // close .getJSON

} // close function

</script>

Add a few posts (/posts), then go back to the root and see what happens. Notice that the entire page did not have to reload in order to return the data for the posts. This is AJAX coolness!

## Task App

### Rails Composer

Let’s use Rails Composer to create our new task app. From the readme, we know to run this command (changing the app name to what you want your app to be named):

rails new myapp -m https://raw.github.com/RailsApps/rails-composer/master/composer.rb

Windows users will probably have an SSL error - here is the fix:

<https://gist.github.com/fnichol/867550>

Here are the options I generally choose for an app like this - feel free to experiment. However, in the beginning, I recommend you use less of the functionality until you get comfortable working with each of these tools.

* Build a starter application? 3) Custom application (experimental)
* Web server for development? 4) Puma
* Web server for production? 1) Same as development
* Database used in development? 1) SQLite
* Template engine? 1) ERB
* Test framework? 1) None (use default minitest)
* Front-end framework? 2) Bootstrap 3.3
* Add support for sending email? 1) None
* Authentication? 2) Devise
* Devise modules? 1) Devise with default modules
* Authorization? 2) Simple role-based
* Admin interface for database? 1) None (add ActiveAdmin later)
* Use a form builder gem? 2) SimpleForm
* Prepare for deployment 1) none or 2) Heroku
* Add gem and file for environment variables? 3) Add application.yml with Figaro

Notice that in both secrets.yml and application.yml, you have some default roles set up. These provide the admin login details you would need to log in as an admin - your database is currently seeded with this fake user (eventually you want to change this otherwise anyone can guess).

Go ahead and run your server and take a look at the app. Commit your changes.

### Adding Tasks

Let’s add Tasks with a description and deadline (“task” is a reserved word in Rails):

rails g scaffold UserTask description:string due:date

rake db:migrate

Commit your changes, then run your server and add some data.

### Switching to AJAX: Controller

Let’s hijack our “new” link so that we can load a hidden form. Here we pass the remote: true option to disable the default Rails mechanism that would have otherwise navigated us to /tasks/new.

<%= link\_to 'New Task', new\_user\_task\_path, remote: true %>

Now, we are going to edit our controller:

1. First, let’s create a **before action** to use an all\_tasks method which will set an instance variable @user\_tasks = UserTask.all:  
   class UserTasksController < ApplicationController  
    before\_action :all\_tasks, only: [:index, :create, :update, :destroy]  
    before\_action :set\_user\_task, only: [:show, :edit, :update, :destroy]  
    …  
   private  
    def all\_tasks  
    @user\_tasks = UserTask.all  
    end
2. Since that is essentially the same code that is in the **index** method, we no longer need that method, so we can remove it from the file along with the **show** and **edit** methods, which are also blank. We are also going to delete the **destroy** methods since we don’t want to do those responses whenever a user updates or deletes.
3. Finally, let’s edit the **create** method to do UserTask.create rather than UserTask.new, and delete the response stuff:  
    def create  
    @user\_task = UserTask.create(user\_task\_params)  
    end

You should only have 3 non-private methods left - new, create, and update.

### New Form Using AJAX

Now, let’s go to the index view and put in a div that we will populate using AJAX, similar to the Google Maps div for the map canvas. This div will hold our task form, so we will give it an id of task-form:

<div id="task-form"></div>

In your stylesheet, set the display to none since we want it hidden when the page initially loads:

~~div~~#task-form {

display: none;

}

Now we need to add the JavaScript code to return that form. In your views/user\_tasks folder, create a new file called **new.js.erb**. In that file, we will tell JS to look for the div with id of task-form, and in that div render the form, then add a bit of animation:

$('#task-form').html("<%= j (render 'form') %>");

$('#task-form').slideDown(350);

This script essentially will take the place of the “render form” part of new.html.erb. Run your server and check it out. You can’t actually create a new task yet but you can see the form render immediately. Commit your changes.

Let’s make the form more user-friendly while we fix it. Check out the documentation for [Simple Form](https://github.com/plataformatec/simple_form). We already installed the gem with Rails Composer, and it already conveniently used the Simple Form syntax for our form. We just need to add the remote: true to the form loop:

<%= simple\_form\_for(@user\_task, remote: true) do |f| %>

### Dynamically Updating Index

We aren’t quite finished yet because when we hit submit, it will want to go to a create view. Upon creation, we want it to re-render our full list dynamically. Let’s edit the index to update the list with JavaScript. First we need the placeholder in **index.html.erb**:

<h1>Tasks</h1>

<%= link\_to 'New Task', new\_user\_task\_path, remote: true, class: 'btn btn-default' %>

<div id="task-form"></div>

<div id="tasks">

<ul>

<%= render @user\_tasks %>

</ul>

</div>

We should practice more partials, so we referenced a new partial that will return the description and due dates. Create a new file in that folder called **\_user\_task.html.erb**:

<li>

<%= user\_task.description %>

<%= user\_task.due %>

</li>

Now, let’s create the JavaScript that will populate the tasks. What do you think it should be called? “create”. Create a new file in the same folder called **create.js.erb** - this is where we will say to render the tasks in the list as well as animate the form away.

$('#tasks').html("<%= j (render @user\_tasks) %>");

$('#task-form').slideUp(350);

Run your server and check it out! Commit your changes.

### Updating a Task

First, let’s go to the controller and edit the update method to call for a new (as yet to be created) JavaScript method:

def update

@user\_task.update(user\_task\_params)

end

Where do we want a user to be able to edit a task? Probably in the task list. Let’s edit our helper for the user\_tasks:

<li>

<%= user\_task.description %>

<%= user\_task.due %>

<%= link\_to "Edit", edit\_user\_task\_path(user\_task), remote: true %>

</li>

Now, create an **edit.js.erb** that is exactly the same as new.js.erb. Similarly, create an **update.js.erb** that is exactly the same as create.js.erb. Run your server and check that it worked. Excellent! Commit your changes.

### Deleting a Task

What do you think we need to change? (destroy method in controller, link in partial, js method) We need to call a new JavaScript method in the controller:

def destroy

@user\_task.destroy

end

And, we need to add a delete link to the \_user\_task partial:

delete

And create **destroy.js.erb** which will simply re-render the list of tasks:

$('#tasks').html("<%= j (render @user\_tasks) %>");

Run your server and test that it worked. So awesome - we can actually do all tasks from this one page without having to re-render the entire page! Commit your changes.

## Beautification

Let’s make our app prettier and more like real apps.

First, go to your **scaffolds** style sheet and delete everything. Notice how your page changes. You may want to do this in your other apps as well.

We already have Bootstrap in our app, courtesy of Rails Composer, but we still need to set up some **structure**. Go to your application.html.erb and add a container, row, and columns:

<main role="main">

<div class="container-fluid">

<div class="row">

<div class="col-sm-6 col-sm-offset-3">

<%= render 'layouts/messages' %>

<%= yield %>

</div>

</div>

</div>

</main>

### Datepicker

Let’s go ahead and add our [datepicker gem](https://github.com/Nerian/bootstrap-datepicker-rails) and bundle:

gem 'bootstrap-datepicker-rails'

Now we need to update some configurations. Add this line to app/assets/stylesheets/application.css:

\*= require bootstrap-datepicker3

Add this line to app/assets/javascripts/application.js

//= require bootstrap-datepicker

Now, go play in the sandbox to build the code with your preferred options:

<http://eternicode.github.io/bootstrap-datepicker/>

Once you’re satisfied, let’s go ahead and put the placeholder in our \_form (and optimize it a bit) - the most important part is the “as string” (See Simple Form documentation):

<div class="form-inputs">

<%= f.input :description, as: :text, placeholder: "Make groceries" %>

<%= f.input :due, as: :string, placeholder: "Today!" %>

</div>

We use “as string” so that the input type will be changed to text which is what the datepicker element needs. If you run your server and inspect element, you will notice that Simple Form automatically gives that form input an id of “user\_task\_due”, so we will just user that id for our JavaScript.

Now let’s create a partial that will hold our code for the actual datepicker - put **\_datepicker.js.erb** in the same views/user\_tasks folder. Copy your sandbox code, but change the id that it is looking for to the id of our input element, and wrap it inside a document ready function like below. Note that you also must have the format set like below otherwise Rails won’t save the date.

$(document).ready(function(){

$('#user\_task\_due input').datepicker({

autoclose: true,

todayHighlight: true,

toggleActive: true,

format: "yyyy-mm-dd",

orientation: "top auto"

});

});

Now we need to call this partial in our new and edit js.erb files:

$('#task-form').html("<%= j (render 'form') %>");

$('#task-form').slideDown(350);

<%= render 'datepicker' %>

Run your server and test it out!

### Layout

We can order our tasks by updating the controller:

private

# Set @all\_tasks

def all\_tasks

@user\_tasks = UserTask.order("due")

end

Let’s go ahead and add the Font Awesome gem, then bundle:

gem "font-awesome-rails"

Then add this to your application.css in the top area with similar require statements:

\*= require font-awesome

Now, take the row and column divs out of application.html.erb and instead make 2 columns in our task index, and use a plus icon in the add task button (add a top margin to the Add task button in your css):

<div class="row">

<div class="col-sm-4">

<%= link\_to new\_user\_task\_path, remote: true do %>

<button class="btn btn-default add-task"><%= fa\_icon "plus" %> Add Task</button>

<% end %>

<div id="task-form"></div>

</div>

<div class="col-sm-8">

<h1>Tasks</h1>

<div id="tasks">

<%= render @user\_tasks %>

</div>

</div>

</div> <!-- row -->

Now let’s update the user\_task partial. See also Bootstrap [wells](http://getbootstrap.com/components/#wells):

<div class="well">

<%= user\_task.description %>

<span class="pull-right">

<%= user\_task.due %>

<%= link\_to edit\_user\_task\_path(user\_task), remote: true do %>

<%= fa\_icon 'pencil' %>

<% end %>

<%= link\_to user\_task, remote: true, method: :delete, data: { confirm: 'Are you sure?'} do %>

<%= fa\_icon 'trash-o' %>

<% end %>

</span>

</div>

### Fancy Deleting

What if we wanted to delete a task just by clicking on it? First, let’s put the description inside it’s own span:

<div class="well">

<span class="task">

<%= link\_to user\_task.description, user\_task, remote: true, method: :delete, data: { confirm: 'Are you sure?'} %>

</span>

<%= link\_to edit\_user\_task\_path(user\_task), remote: true do %>

<%= fa\_icon 'pencil' %>

<% end %>

<span class="pull-right">

<%= user\_task.due %>

</span>

</div>

Now, let’s warn the user by showing a strike-through when hovering on a task by adding this css:

a:hover {

text-decoration: none;

}

.well .task a:hover {

text-decoration: line-through;

}

## Homework

1. Add the stamp gem to format the date prettier in the view. Also research how we could make that date look prettier in the form itself. You can find some clues in [this stackoverflow post](http://stackoverflow.com/questions/7181220/rails-3-jquery-date-picker-date-not-saving-to-database).
2. Add a cancel button to the new and edit forms.
3. If multiple tasks fall on the same date list the date once with each task underneath.
4. Push to GitHub.

Optional Add Ons:

1. Add a condition to check if there are any tasks. If none are listed display a silly image and instruct the user to add a task!
2. Add validation and get error messages to print to the page.
3. Add the pagination gem
4. Push to Heroku