[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 7.1: 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.