

RailMiles

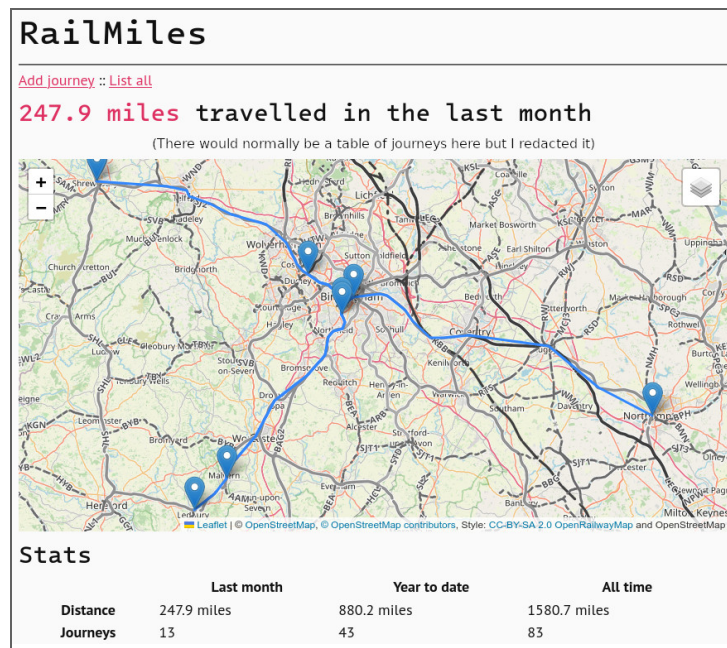
I built an app to track how far I've travelled on trains because of course I did

2023-06-22 :: 919 words

If you know me, you know that I'm kind of a nerd about trains. It's not like I have [an entire page dedicated to what I think of certain types](#), or anything.

In a similar vein to that, I decided that I wanted to keep track of how far I'd travelled by train and that I was going to build a few pages on my website to do that for me.

Before I get into how all that was built and how that works, this is what the end product looks like:



The UI is fairly simple - there's a table of all the journeys logged in the last month (not pictured), a map of those journeys and some overall stats at the bottom of the page.

To add a journey, at the very least all you have to do is tell it the date you travelled on and the stations you started, changed and ended at:

Locations should be entered with the short code (eg: SLY) and optionally the service UID (eg: SLY, C16977).
Separate locations with a newline.

Date (YYYY-MM-DD) [\(today\)](#)

Locations

Manual distance

Return?

Submit

How it's built

When a new journey is entered into the system, there are two key things that need to happen to be able to show the user a map and update the mileage totals.

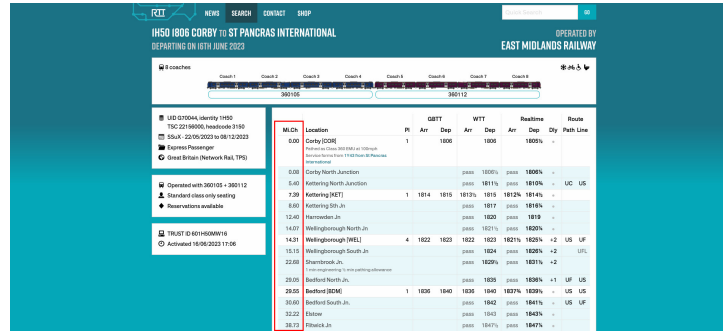
- First, the length of the journey needs to be worked out by looking up schedules
- Second, the physical places the train runs through need to be logged to generate the map

Getting data

Currently, I use [RealTimeTrains](#) to source distance data. Their API is first used to look up the schedule for the trains that the journey was comprised of. A journey can have multiple legs - for example, if I'm travelling from Corby to Brighton, I would take two trains, one from Corby to London St Pancras and one from St Pancras to Brighton. For each leg in the journey, we look up the schedule for a train that took that route on the same day that the journey took place on, then pull the stations it passed through and the distance travelled from that train.

However, the RealTimeTrains API only allows searching services by the stations they call at (as opposed to just by the origin and terminus stations) on the day they ran. This means that, sometimes, we have to manually enter the service IDs of the trains that formed the journey to get accurate results, for example if we're logging them for a journey we took yesterday. In this case, the program skips straight to requesting the routing and distance information for that train and don't bother doing any schedule lookups.

While scheduling information can (and is) pulled from the RTT API, distance information for a service is not. RealTimeTrains only serves this data in the frontend, so if we want it, we have to do some simple web scraping.



MLCR	Location	Pl	Arr	Dep	Arr	Dep	Arr	Dep	Arr	Dep	Arr	Dep	Path	Line
0.00	Corby (COB)			1806			1806			1805				
0.00	Corby North Junction				1810		1810			1805				
0.00	Kettering North Junction				1815		1815			1805				
7.29	Kettering (KET)	1	1814	1815	1815	1816	1815	1816						
8.00	Kettering (KET)				1817		1817			1805				
12.00	Wellingborough Jn				1820		1820			1805				
14.21	Wellingborough North Jn				1821		1821			1805				
14.31	Wellingborough (WEL)	4	1822	1823	1823	1824	1823	1824			+2		US LF	
15.15	Wellingborough South Jn				1824		1824			1805				US LF
22.00	Sheffoed Jn				1825		1825			1805				
29.00	Sheffoed North Jn				1825		1825			1805				
29.55	Bedford (BDM)	1	1836	1840	1836	1840	1837	1839						US LF
30.00	Bedford South Jn				1842		1842			1805				US LF
32.00	Stoke				1843		1843			1805				
38.73	St Pancras Jn				1847		1847			1805				

This is the actual code that pulls that information out of the HTML, after which the numbers are parsed and converted into miles.

```
1 var waypoints [][3]string
2 doc.Find(".location.call,.location.pas
3     shortcode := shortcodeRegexp.Find(
4         selection.Find(".location a")
5     )
6
7     if shortcode == "" {
8         // If this is a junction with
9         return
10    }
11
12    waypoints = append(waypoints, [3]:
13        shortcode,
14        strings.TrimSpace(selection.F:
15        strings.TrimSpace(selection.F:
16    })
17 }
```

Note that here, a short/CRS code one of the three letter codes used to refer to a train station - eg BHM for Birmingham New Street, KGX for Kings Cross, or VIC for London Victoria.

From all of this, we get a sequence of locations that the train passes through and the distance between them all. Now what?

Generating maps

It's all well and good talking about abstract location data, but it'd be nice to put that on a map and visualise it. Enter: [OpenStreetMap!](#)

Using OSM data, it becomes trivially easy to translate a CRS code into a latitude and longitude using the [Overpass API](#), which puts a layer on top of the raw OSM data to let you query it with a dedicated query language. In fact, to get a list of station locations (and while we're at it, names, since we only store short station codes instead of their full names in the database), all you need to run is the following query:

```
1 [out:json];
2 node["ref:crs"];
3 out geom;
```

... which gives you a load of JSON that's comprised of entries looking a little like this (you can play around with the query [here](#)).

```
1 {
2   "type": "node",
3   "id": 104734,
4   "lat": 51.5656526,
5   "lon": -1.7858762,
6   "tags": {
7     "name": "Swindon",
8     "naptan:AtcoCode": "9100SDON",
9     "network": "National Rail",
10    "operator": "First Great Western",
11    "platforms": "4",
12    "public_transport": "station",
13    "railway": "station",
14    "ref:crs": "SWI",
15    "toilets:wheelchair": "yes",
16    "wheelchair": "yes",
17    "wikidata": "Q3244572",
18    "wikipedia": "en:Swindon railway station"
19  }
20 }
```

I filtered this raw data so it only contained the stuff I was interested in, then embedded the file in the Go code that powers RailMiles.

Now we have train routes and the locations those routes correspond to, the last thing we have to do is to render those to a map.

I chose to do all the visual mapping stuff with [Leaflet](#) using the default OpenStreetMap tiles and some fancy OpenRailwayMap tiles as an overlay to emphasize the main railway routes atop that. On the main page, the routes and stations visited in the last month are compiled into a blob of GeoJSON and fed to this chunk of code:

```
1 <div id="journey-map"></div>
2 <script>
3     let map = L.map("journey-map");
4
5     L.tileLayer('https://tile.openstre
6         maxZoom: 19,
7         attribution: '...'
8     }).addTo(map);
9
10    let ormOverlay = L.tileLayer('http
11        attribution: '...',
12        minZoom: 2,
13        maxZoom: 19,
14        tileSize: 256,
15        className: "tile-orm", // this
16    });
17
18    let layerControl = L.control.layer
19
20    let gj = L.geoJSON("<< geoJSON goe
21        if (feature.properties && feat
22            layer.bindPopup(feature.pi
23        }
24    });
25    gj.addTo(map);
26    map.fitBounds(gj.getBounds())
27 </script>
```

Problems

As much as I'd like to say that this is perfect and works 100% of the time - it doesn't. Sometimes it falls flat on its face and ceases to work. The main issues I know about are:

- I'm using web scraping - RealTimeTrains doesn't have distance data in their API, probably for a reason, so if they cotton on to what I'm doing, I wouldn't be surprised if they stop me from doing it
- Sometimes, distance data just isn't available and isn't shown on the page, which stops the whole thing from working.
- It's a bit slow

I have a couple of ideas of how to fix these. More posts to follow!

© Thomas Pain 2019 - 2023. Written content licensed under the [CC BY 4.0](#). [Credits](#). [Sitemap](#).

