1 Introduction

As we enter the information age, we are inundated with various types and formats of data that must be interpreted in a positive manor. There is tons of data readily available to the average person provided by Network Rail and the UK Goverment, and it is important to use this data when making informed decisions on about the network such as timetabling and routing. The masses of data can be used to find artificial choke points that may not be obviously be a problem region, or problem trains that cause may not themselves be delayed, but cause other trains to become delayed, for example a Manchester Airport to Middlesbrough train may not itself be delayed, but due to the track it requires at Ordsall Junction, means that 5 tracks are blocked at once. Without data, it may not be possible to realise this.

2 Sources

This chapter relies on quality data sources, with consistent formats that can easily be interpreted automatically and filtered automatically. Equally a high level of detail is required for them to be of any use. This project utilises two main sources of data.

2.1 Network Rail

Network Rail provide a number of operational data feeds available to developers. They state on their website (*Network Rail Data Feeds* 2020) that by providing access to these feeds, they hope to encourage the development of new products of interest to those who use the railway.
The available feeds are:

- SCHEDULE
- MOVEMENT
- TD
- TSR (Temporary Speed Restrictions)
- VSTP (Very Short Term Plan)
- RTPPM (Real-Time Public Performance Measure)
- SMART
- Corpus
- BPLAN
- Train Planning Network Model

This paper will utilise most of these, and will be described later. More information on these data streams and how they are used is in the additional file

2.2 National Rail (Darwin)

Darwin is the primary train tracking system in the UK, and is used in all stations for their Departure Boards. It uses many data sources including Network Rail's to provide real-time arrival and departure predictions. The information from Darwin is far more detailed than that from Network Rail, however is significantly over-detailed and complicated for the scope of this project. The Historic Service Performance will be used to provide additional reliability metrics that were not collected with the system created for this paper (due to the HSP giving a time period of up to one year). The data streams provided by the Darwin system are:

- LDB Webservice (PV)
- LDB Webservice (Staff Version)
- Darwin Timetable
- Darwin Push Port
- Historic Service Performance (HSP)

More information on these, and how they are used are also in the additional file.

2.3 CIF

The Common Interface File (CIF) format is the industry standard for transfer of schedules electronically from Network Rail's Integrated Train Planning System (ITPS) to downstream operational and information systems. Data feeds usually supply both json and CIF files, the CIF data is more suited to advanced users of the service and requires additional parsing compared to the JSON data.

The CIF file is a text file containing one record per row. Each record has a fixed length of 80 characters. The first two characters of a row identify the Record Type.

Although JSON is used in other areas of the project, CIF is used for this section due to it being the standard format in this area, as well as being more detailed.

3 System Design

The systems designed for this project follow a service-oriented architecture, and all operate independent of each other. This allows for greater flexibility when testing or creating new features. There are 4 separate systems that integrate with each other using REST

1. Streamer
2. REST Service
3. Website
4. Android App

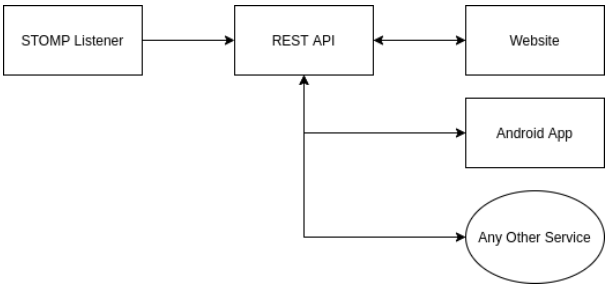

Figure 1: Server-Oriented Architecture

The service-oriented architecture allows for massive scalability as each system can be replaced individually as long as the produce the same output.

3.1 Finding the Data-stream

Designing the Streamer (or more accurately the listener) was based on the snippet provided by the Open Rail Wiki (*Open Rail Data Wiki* 2020) and is shown in Listing. 1

```
var prettyjson = require('prettyjson'),
    StompClient = require('stomp-client').StompClient;

var destination = '/topic/TRAIN_MVT_ALL_TOC',
    client = new StompClient('datafeeds.networkrail.co.uk', 61618, 'your-email', 'your-
    password', '1.0');

client.connect(function(sessionId) {
    console.log('Trying to connect...');
    client.subscribe(destination, function(body, headers) {
        console.log(prettyjson.render(JSON.parse(body)));
    });
});
```

Listing 1: Node.JS Example Stomp Listener

Listing.1 shows a simple STOMP application in JavaScript that simple listens for all TOC changes, and then prints them as they come. Using a program similar to this allows us to confirm the data format shown on Open Rail Wiki is correct and as such the databases that will store the data must be designed, as simply printing the data is not the final solution. The system requires some static information, which

are 'reference','smart' and 'corpus'. These are direct imports and are described in more detail in the additional file.

The 'schedule' collection is initialised with a full import, but then uses a daily import to keep up to date.

The 'trains' collection records every train that's been seen on the network. This includes trains which have been TRUST activated, have been seen by a Train Describer. The records are a linking of the SCHEDULE, TD and Train Movement feeds, as well as the SMART/CORPUS data about their current location.

4 System Implementation

The system implementation

4.1 Tools Used

All tools used in development of NRDF Service are described here, since its a complex problem with various purposes and requirements.

4.1.1 Javascript

Javascript forms the entire basis of the Streamer, REST Service and Website. It was chosen because it was able to perform all 3 tasks well. If more data needed to be processed (if Darwin was to be used) then Javascript could not keep up, and a solution would need to be made using a more strongly typed program in C or a memory-optimsed Python program

4.1.2 STOMP

Network Rail API uses STOMP to transmit all data, STOMP (Simple (or Streaming) Text Orientated Messaging Protocol) provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to provide easy and widespread messaging interoperability among many languages, platforms and brokers.*STOMP* 2020

4.1.3 Node.js

Node.JS is a JavaScript runtime environment, all JavaScript in this paper uses Node.JS. Additionally, due to the limited time-scope of this project it was infeasible to create a project devoid of dependencies, so Node.js manages the required dependencies including mongoStream (MongoDB interface), bunyan (JSON Logger)and also stompit(STOMP interface)

4.1.4 CIF

CIF is one of the two data file types used in this paper, and is described in more detail in Section. 2.3. CIF is used for all live data, including parsing the TRUST and TD data streams.

4.1.5 JSON

JSON (JavaScript Object Notation) is a datainterchange text format (*JSON* 2020) and is the second data file type used in this paper. It is used for static data, such as the streamer config file, and logging. It is easy to read and write and is language independent. Although Network Rail also uses JSON for sending out their movement messages, it is not the preferred choice. It is also used in the REST Service when returning the data after receiving a request.

4.1.6 MongoDB

MongoDB was chosen for its versatility as a NoSQL table, and is the core database for the project. The streamer saves data to the table, and the REST Service queries the database.

4.1.7 Google Maps API

Google Maps is the leading map provider, and has fantastic JavaScript integration as well as lots of documentation.

4.1.8 Postman

Postman was used to test all REST Service endpoints, and was key for development of the REST system.

4.2 Listener

The Streamer uses Node.JS and Javascript with a STOMP Listener to collect data from Network Rail and inputs that data into a MongoDB database.

4.3 Core Database

The core database has multiple collections that all have their purpose briefly described here. For a more detailed description look at the additional document.

4.3.1 Association

Stores schedule information relating to what days a specific schedule entry (route) will run .

4.3.2 Berths

Stores all train headcodes that have passed through that specific berthfor a given berth.

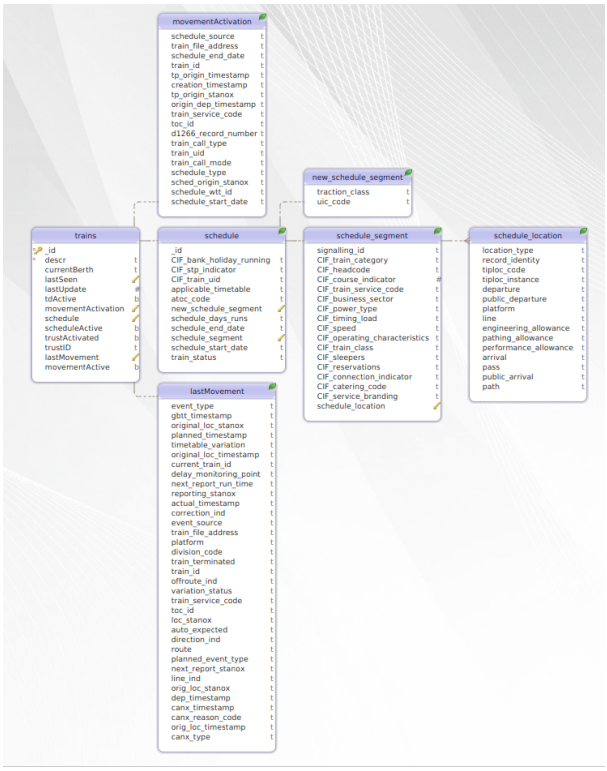

Figure 2: Relation Diagram of 'trains' db

4.3.3 Corpus

Stores all entries from the CORPUS system (Codes for Operations, Retail & Planning – a Unified Solution). This can be used to translate STANOX, TIPLOC, NLC, UIC and 3-alpha (CRS) codes to location latitude and longitude.

4.3.4 Reference

Stores schedule entry (route) information such as Operator and Timetable Start/End data.

4.3.5 Schedules

Stores every timetable entry, and all related information. It indicates all stations it will call at, and their respective times as well various operational data.

4.3.6 Signals

Stores the current state of each signal (Not all signals are in the system, usually only signals in junctions are available).

4.3.7 Smart

Stores berth to berth information, to understand how all berths link together.

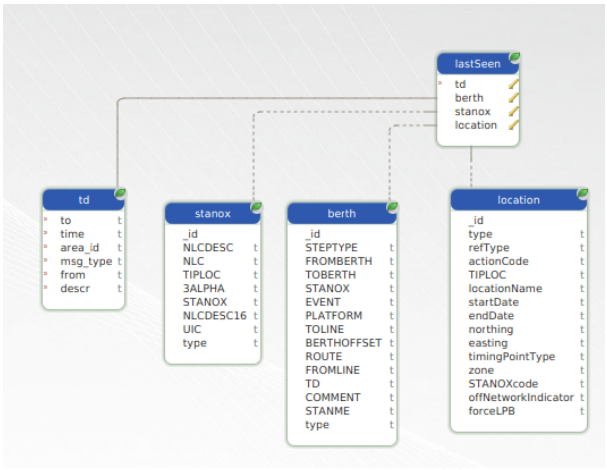

Figure 3: Relation Diagram of 'lastSeen' db entry

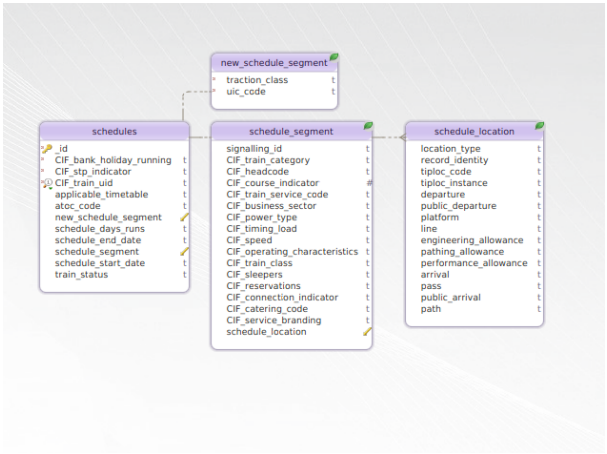

Figure 4: Relation Diagram of 'schedules'

4.3.8 Tiplocs

Stores all TIPLOC (Timing Point Location) entries.

4.3.9 Trains

Trains is the key database that is used for the REST API for live information. Every TD Activated Train gets an entry, and trains can also be Schedule Activated, Movement Activated and Trust Activated. A train entry can store all required information for any operation.

4.4 Auxillary Database

4.4.1 Tocs

Stores all Train Operators key information

4.4.2 Routes

Stores some routes (This is a database being populated manually, and as such is no where near complete)

4.5 REST Service

The REST Service is the basis for all other interactions with the data, and conforms with standard RESTful practices. The REST Service is a Node.JS and Express application to deliver fast and responsive requests. If the service was to become public asynchronous implementation would be a priority, but currently the system works as desired. Requests are made via HTTP.

Example Requests:

`http://~REST-API~/live/schedules/nt`

`http://~REST-API~/live/service/W12345`

`http://~REST-API~/live/all/mco/to/wgn`

The first request will return the next 25 trains that NT (Northern Rail) will run. The second request will return detailed information about the train with UID W12345 that exists on the day. The third request will return the next 10 trains that arrive/depart at MCO (Manchester Oxford Road) to WGN (Wigan North Western).

4.6 Website

The Website utilises the REST Service, as well as Huxley, to allow a view into the data. Using Node.JS and Express (as well as alot of other dependencies) as a basis for the website backend. It uses an MVC Project Structure, and uses a REST-Style template engine, Pug.JS, to deliver web pages.

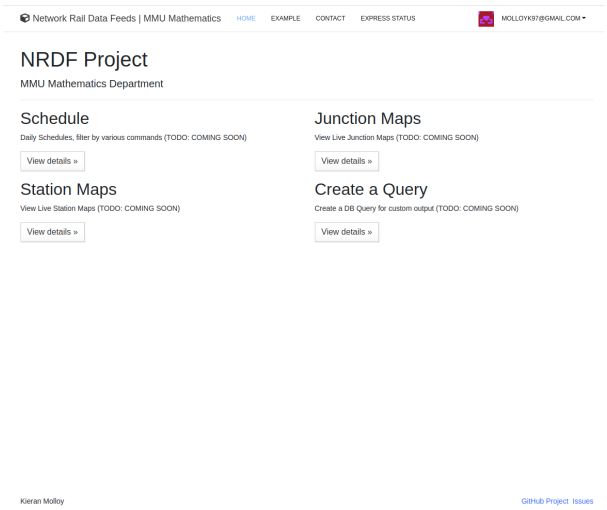

Figure 5: Homepage of NRDF Website

4.6.1 User Functionality

The website has user login functionality including session storage and remembering previous searches, user security used includes password encryption and salting. There are two databases required for functionality, 'sessions' and 'users'

4.7 Android App

The Android App interfaces the same as the website.

5 Results

This project has delivered 3 key programs, and 2 side programs. The key program is the REST API which powers all services, but this could not work without the Streamer and Databases. The website and Android App are simple interfaces for easily accessing the API. The Android app can be run on any Android device running Oreo 8.0 or higher. However due to limitations with the database and API, must also be connected to MMU Eduroam.

6 Discussion

This project has generally been successful, and has delivered everything that was set out. Further work would include extending the API to improve functionality, potentially include more security features to it or tokenisation. The database design could be improved by using referencing instead of what currently happens where data is copied into new documents. As for website and Android App, they could improved in every way, simple improvements could include UI updates. It would interesting to include user geolocation to provide a list of nearby stations

References

JSON (2020). JSON. URL: `https://www.json.org/json-en.html` (visited on 02/17/2020).

Network Rail Data Feeds (2020). Network Rail. URL: `https://datafeeds.networkrail.co.uk/ntrod/login` (visited on 01/06/2020).

Open Rail Data Wiki (2020). URL: `https://wiki.openraildata.com//index.php?title=Main_Page` (visited on 01/06/2020).

STOMP (2020). STOMP. URL: `https://stomp.github.io/` (visited on 02/17/2020).