

This year (2019) I have been working on multiple projects while I study at Victoria University. This document discusses my various projects and the work I did on them. If you'd like to see some of the projects, code samples or any other details, feel free to email at contact@alexmorris.nz

Summary

I have been working on five main projects this year, using Java, Python, PHP, JavaScript & more.

- NZGrapher was my work on improving performance in a widely used graphing application in New Zealand high schools.
- With Makine I wanted to learn about recommendation algorithms and high-performance indexing services.
- GTFS/Commuter I built an API to query New Zealand transit network data through a single API.
- I used Navra to learn about Vue.js and the various frontend frameworks.
- Tetris Tie is about building a tie which can play games.

NZGrapher

After suggesting security fixes for NZGrapher (<https://grapher.jake4maths.com/>) a couple years ago I decided to try and fix some of the issues I had with the site, so in July of this year I forked the popular graphing site to include new features, performance improvements and optimisations. Since then I have moved all of the graphs to JavaScript for generation(Making it very fast), fixed a few bugs, and added new features for graphs. I have also updated it to include changes from the upstream repository if the changes are applicable.

NZGrapher is open source, and my fork is available at <https://github.com/ledship/NZGrapher>. Due to the way it was structured from the start and the need for supporting legacy browsers it is stuck using antiquated code.

However more recently I have been thinking about scrapping my fork of NZGrapher and starting from the beginning on a new grapher, using Babel to convert any ES2015 into backwards compatible versions for legacy browser compatibility. This would allow for a better structured codebase and the ability to add new features easily.

I would also like to add support for more graph types and integrate features such as t-tests and ANOVA tests, so it would be usable for university students.

Preferably such a project preferably would be made in collaboration with Jake Wills(Creator of NZGrapher).

Makine

In April I started working on a project called "Makine". Makine, Turkish for Machine has four main components. These are the indexer, tagger, website and recommendation service.

Makine will index threads on [reddit.com](https://www.reddit.com), add them to a database, tag them based on keywords and phrases, they will then be accessible via a website which recommends which threads/content you should look at.

Indexer - Java

The indexer takes a list of subreddits(different locations on reddit) that it will index, alongside any addition settings it may need to help categorise it for tagging later on, and turns that information into subreddit objects. I split the subreddits into groups based on the number of threads specified in

the configuration, these batches of subreddits are then passed into a new thread and they start indexing subreddits.

The actual indexing is just making a request to a JSON file on reddit.com which contains all the posts on that subreddit, I take this, turn it into individual post objects and pass them back to the batch of subreddits they came from.

Once a batch of subreddits have been indexed they are run through some post processing to standardise the URL's from specific websites and to parse additional information in the post.

All the posts which have been processed are added to a database of untagged posts, the indexer then starts processing a new batch of subreddits and will continue to run until stopped.

Because of the lack of good reddit libraries for Java I built my own into Indexer which was also used in Tagger.

Before I stopped working on Makine I was in the process of integrating the Tagger into the Indexer version 2. This new indexer was designed to run faster and tag the posts at the same time, it was able to complete what Indexer V1 and Tagger did in the less time.

Tagger - Java

The tagger takes in indexed posts from the database and searches the titles and content for words and phrases which may be of interest, it will then add a row into the database linking that post and that tag. Once a post has been completely searched it is removed from the indexed posts table and moved into the processed table. This makes the post available on the website.

Recommendation Service – Python

The recommendation('rec') service was made with Python, using Flask and LightFM, a library with recommendation algorithm implementations.

The rec service ran a Flask webserver, which when called with a user ID would initiate a new recommendation process to generate recommendations.

My first attempt at a rec system would build a model based on posts scored by the user, scored each post using that scores and weighted the values of each tag's score(each tag had a higher/lower score depending on what it was). It would then compare the scores with posts in the dataset and grab the highest matching 1000 posts, sort them by score and then add them to the database in a new feed. This feed could then be accessed by the website to show the recommendations. This was a rather intensive process for the results that it generated.

Before the end of Makine I started looking into using optimising this as I had rushed the implementation and wasn't focused on it very much as I was more interested in indexing/tagging. I think I could have optimised the model much further, and built proper interactions/user matrices rather than how I did it. Admittedly, I am not an expert by any means at machine learning and I am sure there are far better ways to generate recommendations for a system like this.

Website – PHP, JavaScript, & Laravel Framework

The website was made using the Laravel Framework version 5.8. It was simple, as most of the work happened behind the scenes in indexer, tagger and the recommendation service. I chose the Laravel framework because it's easy to use and I am familiar with it.

With the website you can log in, view new and trending posts from subreddits, alongside a feed of posts which the system believes you might like. The website would simply grab the posts from the

database and display them. Any ratings you made on the site would be saved to the database through the API.

Through the website you could also request new recommendations, this would send a request to the recommendation service with your user id.

Conclusion

My work on Makine started in April and ended in August, however in July worked slowed considerably. I stopped working on Makine as I lost interest in the results generated by it, and the costs of keeping it up were more than I was willing to pay for a project I had fading interests in.

By the end of the project I had indexed over 700 thousand reddit posts across 40 subreddits on my staging machine which ran for 34 days straight.

My staging server was a DigitalOcean 2GB 1 vCPU running Ubuntu 18.04 running a typical LAMP stack.

These figures do not include the posts indexed on my own machine where I was testing, or any tests I conducted before I left the machine running to see how it would perform.

I think overall the project was a success. I had set out to learn more about recommendation systems and build a service which could index reddit posts as fast as possible. I believe I achieved this.

If I was to redo Makine now, there are definitely improvements I'd make, however most of them I had started to work on near the end of the projects life. These include faster indexing, integrating the indexer & tagger, better recommendations and including more options/attributes for tagging and indexing posts.

GTFS/Commuter - Java/Spring Framework

In late August I found myself interested in public transit data, specifically Metlink's scheduling and real time data. In my search for Metlinks data I found they had no documented API for real time data and only provided Google Transit Feed Specification('GTFS') files. So I started building "Commuter".

Commuter is an API accessible through the web(<https://commuter.nz>) which allows easy access to an API for querying not only Metlink's data, but any other transit system's data in a standardised way.

I made Commuter in Java using the Spring Framework, this allows me to focus on processing the data rather than messing around with webservices and building an ORM. Because I was using the Spring framework I used the Hibernate ORM.

Commuter will automatically download new GTFS data and ingest it on a schedule, checking if the GTFS data is out of date once a day. The ingestion process is simple, we download the new data, read the files and turn them into objects which can be used across the API. This can take a little bit of work as GTFS data isn't very relational in some aspects.

Alongside the GTFS data, we need real time information, Metlink doesn't have a SIRI("Service Interface for Real Time Information" API yet(If it did I wouldn't have started this project) however it does have an undocumented real time API. I query this API every two minutes across all services, checking on their status. This is then recorded in a database.

Currently Commuter only allows access to the GTFS data as I work to make the real time data reliable and patch bugs which reduce the reliability of the service when using real time data.

So far I have only talked about using Metlink's data, however I made Commuter flexible. You can query each transit system's data separately through the same API. To add a new transit network all I need to do is add a new line to a config file. Each transit network is stored in a new database as GTFS data can be big and it simplifies data access. GTFS data contains the timetables for each individual service for multiple months. Metlink's data itself is over 500,000 rows, 297,000 of these are records for a stop on each service (Not each route!).

Once Metlink makes a documented real time API or implements SIRI I'll discontinue the Commuter project and adapt the domain name for a new project, however in the mean time I will continue to track my train and bus services through my API.

Navra

Navra is the continuation of my student management project from a couple years ago. Built using the Laravel Framework and Vue.JS it features a modern material design look, new features and is faster than before.

I chose Vue.JS for this so I could learn a new and widely used frontend framework, specifically Vue because of its ability for rapid prototyping.

I started working on Navra August after I stopped on Makine and while work on it has slowed down a bit recently its going great and I think with the way Laravel and Vue make you structure the project it should be rather easy for me to add more features and extend it to the point where eTeach(<http://dev.isolatedsoftware.nz/eteach/>). I have made some different design decisions on Navra, ones which I wished I had made on eTeach and I think that they will make it much easier to develop and allow for greater flexibility within Navra.

Such as the notice system which in eTeach when you wanted to have a notice showing for a week I would create a notice for each of those days, however in Navra it will create a notice group containing the notice details(start/stop time) and then an individual notice referencing the group for each day. This means you can easily delete/edit a notice which spans multiple days and have them all update at the same time.

Tetris Tie

I found a video a few weeks ago of someone playing tetris on their tie, and with my love of flashing lights I knew I had to make one.

I started by looking for the largest display that could fit in a tie I own. I found a 8x32 RGB display using WS2811 leds. It has 256 individually addressable leds. I want to be able to control the game on the tie, unlike the one in the video, so I bought a NodeMCU which I can connect to wireless through my phone. I have almost finished programming the game in Arduino's C variant and am just waiting on a power supply large enough to arrive so I can test it.

I own a few 18650 batteries left over from a robotics project which I'll be using to make it portable. At maximum brightness on white the display will draw 14 amps at 5 volts. I'm going to program it for more then just tetris in the future, such as snake.

Once I have built my tie I will be doing more display projects like this, as I already have a huge amount of components needed for them and some great ideas.