BMAT Technical Challenge Report

Victor Challier

January 2017

1 Task presentation

BMAT's fingerprint team collects data from all kinds of channels. Here is what the data looks like:

- Radio stations, identified by their (unique) names
- Performers, also identified by their names (which you may assume is unique)
- Songs, identified by their names and their performer
- Plays, identified by a song, a radio station, start and end time (with a precision to the second)

Our task is to build a web server that can handle simple requests to store and query the data. It will answer to GET and POST calls, that should cover the following basic functions:

- Insert a new radio station
- Insert a new artist
- Insert a new song
- Insert a new play
- Get all the plays for a given song between two dates
- Get all the songs played on a given channel between two dates
- Get a top 40 for a given week and a list of channels: a list of 40 songs ordered by the number of times they were broadcast. For each song, provide their performer, playcount and the position they had the previous week.

This report is a presentation of my deliverable. It will explain you how to run the code and my design choices. Afterwards, we will try to figure out some requirements to allow such web server to run on a production level.

2 Run the code

2.1 Download the material

Click on **this link** to download the material needed in the project. The folder BMAT_Technical includes the following files:

- app_express.js : javascript file that launches the server and has all the function definitions.
- test.py : file provided by BMAT to test the code with fake data.
- myDataBase.json : json file that we will use as a database in this project. It is currently loaded with BMAT's fake data that we can find in the test.py file.
- description.pdf : pdf provided by BMAT that explains the project.

2.2 Install Node and packages

I coded this project using node, which is an environment built on Javascript. To download and install node, follow the instructions on **this link**. Once the installation is complete, you can install packages from the command line with node's package manager:

- npm install http (should be installed by default)
- npm install express
- npm install body-parser (should be installed by default)
- npm-install node-json-db

2.3 Run the code

Once all the packages are installed, we can run the code. Go into the folder with all the files and open a command line window. Run 'node app_express.js', this should return the database initiation and the message 'App listening on port 5000!'.

Once the app is running, you can run the python file. Type in another command line 'python2.7 test.py –add-data' (runs the code with BMAT's fake data). On the python side, we should get the 'Success!' message displayed. If not, you can investigate through the error message or send me an email at challier.victor@gmail.com so we can discuss that together. And that's all for the initiation!

3 Solution design

3.1 Doodle the solution

The figure 1 is a quick graph I made to understand what are the available objects and what are their functions. The python file launches GET or POST requests.

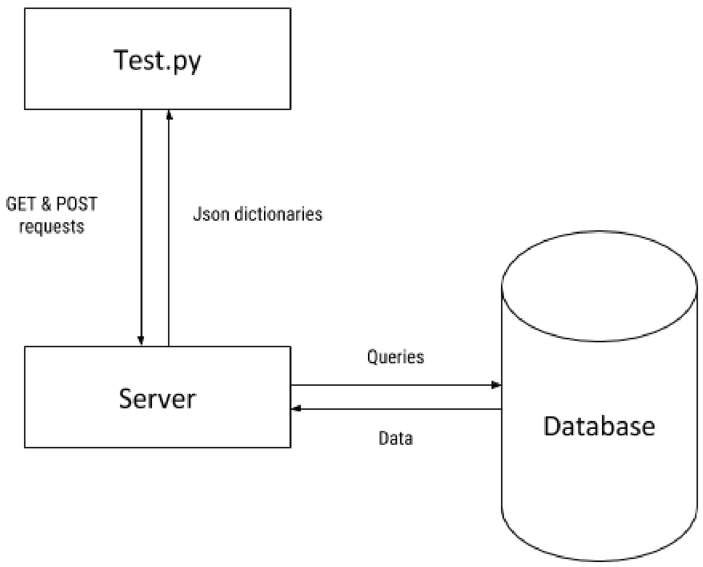

Figure 1: Simple design of the solution

A server listens to these requests through a port, and for every request returns Json dictionaries.

When receiving a request, the server will interact with a database, that stores the data. The server queries, and the database returns data that the server then processes before sending them back to the python file.

Now that we have the architecture for the problem, we can look at the tools that we want to use.

3.2 Project Management

Once I had the solution design, I defined the steps that I would follow until the end of the project. This allowed me to focus on small tasks while always having in mind a clear idea of where I was going. Here is a summary of these steps:

- Have the python file running
- Setup a web server
- Have the wen server receive requests from the python file
- Have the web server send back default answers to the python file (based on the test.py file, these default answers were {result:{}, code:0}).
- Have the web server retrieve the parameters from the GET and POST requests
- Setup the database
- Connect the database and the server

- Put fake data in the database
- Have the database and the server exchange data (at that point, all the connections are setup and we can work on processing the requests).
- Build functions for the POST requests
- Build functions for the GET requests

Once I had defined this path, I could focus on the implementation part and get to coding.

3.3 Tool choices

3.3.1 The server

Choosing the tools to build the server was not an easy task for me since I am not very familiar with this topic. I knew that I wouldn't have a lot of time to do that project. I had already used node once to run a quick local server and have a bit of Javascript programming skills. Given this, Node would certainly be a good way to start this project.

I visited several blogs and websites and quickly discovered the 'express' package, a simple way to setup local servers. Not long after that, I had the web server set up and could receive the requests from the python files, sending back my default answers.

3.3.2 The database

Choosing the correct tool for the database was more painful. The project presentation suggested that the data architecture would include several tables and that we might have to do several joins at some points. Based on this, I thought there would be two criterias and the tool choice:

- Easy to connect with Node
- Easy querying methods

I have some good SQL skills, so given the parameters I decided that using MySQL would be a good idea, since I should just figure out how to connect Node and the MySQL database.

After struggling a bit with the connection, I realised that we wouldn't need the table join, hence that the query wouldn't be complicated. When retrieving the information we would only look into the "Plays" table, and adding some javascript filtering on the result would allow us to get the data we want. The presentation pdf suggested to use a Json file as a database so I started looking in that direction.

I first worked with the package 'jsonfile', that allows to write and read pdf. It wasn't very convenient though, so I looked for different solutions. That's when I found the package 'node-json-db', which given its name was likely to be very helpful. Indeed, it perfectly suited my needs and I decided to work with it. In addition, working with a Json was convenient since I had to return Json dictionaries to the python file, so I may have avoided a lot of conversions. The only issue is that it doesn't seem to handle the idea of primary key. I had to manually code extra functions to make sure the same song/channel/performer wasn't added twice to the database.

3.4 Questions

a. What you provided is a prototype and therefore not a production ready software. But, of course, the marketing team doesn't know about that and already sold the finished product to three different customers. What do you think needs to be improved in your software in order to be used in production?

The most important point according to me is that strong assumptions were made for the sake of this project. It is not true that title names and performer names are unique. To solve this issue, we should add meta data in our records, such as the trademark that helps identify the performer. When looking at the data architecture I made sure that when we add a play, if the performer in the play is not available in the performer table then we should add it. Same for channels and title. Having some primary keys set up on each tables and foreign keys between different tables is clearly the most important point. A MySQL instance could definitely work her, and I just realised that this is something we can do with Json and that I could have included in my project.

We should also look at the data consistency. On boarding a new customer requires some heavy work to make sure that the data he sends is correctly formatted and consistent. For example, I didn't check all the 'start' and 'end' dates were actually dates. It might happen that a record was mismatched and we get the title instead of the date, which could lead to an error in the data processing.

I think these are the main point of concerns when looking at what I produced. On a more technical level, to ease the maintenance and debugging, I would probably add more comments and add error messages before pushing to prod.

b. How do you think your program would behave if the fingerprint team starts going crazy, inserts ten million songs and starts monitoring two thousand channels?

A server lifespan is about 3-5 years, so let's see if my computer could handle 3 years of music played by two thousand channels among ten million songs.

If the channels play music non stop and each music is three minutes long, then two thousand channels will produce over three year :

$$n_{songs} = \frac{numberofchannels * years * minutesperyear}{songlength} = 1,051,200,000$$

So all these channels working non stop would produced more than one billion play records over three years (on a side note, it helps to realise the meaning of a youtube video with one billion views).

Let's look at my computer settings here. We are trying to answer these three questions : 'can it process all the data?', 'can it store all the data?' and 'can it run queries on the data?'.

Can it process all the data? Node can handle thousands of calls per second, so there is no problem to process the data.

Can it store all the data? I think having ten million different songs is not important parameter here, since the play record is the largest one (has 5 attributes) and there will be one billion of it, which represents 99% of the data we want to store. One billion row in a Json represents approximately 100GB. I currently have 170GB of free storage, which means I could maybe hold such information, not taking into account that we should have a copy of the data and other good practice measures.

Can it run queries on the data? Let's assume we have 100GB of data stored in our database. The maximum json size that can be handled in one node request is around 2GB, so we could not use my prototype. Even if we only want to take one sample of the data (for example one day of plays), my data query system would not work. The thing is that if I want all the plays from last week, I first pull all the plays from the json file, then I filter the values I want with some javascript. This means that at one point, all the plays are stored in a javascript array which is not optimized at all. Using a MySQL instance could solve that issue, because the query would only pick the records we need from our database.

References