1 Progress Report

1.1 Overview

Since the beginning of the project, we have made steady progress to achieving our initial goals. As things stand, the majority of the modules that are independent have been completed and have gone through a limited amount of testing, and are now waiting on integration into a single unified system. The last module, analytics, has not yet been implemented largely because it's entire action depends on the existence of the other three modules as a single functional whole, and hence the majority of the work that has gone into that has been on deciding on the functions and algorithms involved in the module.

1.2 Web Interface

The web interface has a basic structure and style established, with the most important functionality available to the end users. To create the web frontend, the Python Django framework was chosen, which handles much of the heavy lifting yet allows for flexibility regarding additional plugins and functionality that we may wish to add. While much of the data is generated on the serverside with Django, to improve the user experience we are utilizing Javascript to handle events which are better suited to the clientside such as buying, selling, and updating information asynchronously. This results in an interface which is as frictionless as possible.

Stylewise, we have tried to mimic the main Last.fm website to a certain degree, keeping aspects which work well (display of images, colours, layout) while also augmenting this design with new additions. This allows for a look and feel which is familiar to users yet is fresh and stylish.

With regards to Django, dummy data has now been written to mock up the objects that will eventually be received from the API, with matching naming conventions, and made available to the templates. Also, Last.fm authentication is now supported, so that users are redirected to a Last.fm page where they can give the Scrobble Exchange app permissions to access their profile. Last.fm then redirects back to us, with a token that can be used for API calls. Functionality to redirect back to the specific page that the user was on previously still needs to be added.

Testing on the frontend Javascript and HTML will potentially use a suite such as Selenium, but depending on the extent and complexity of the code, this may be put to one side. However, Django controller testing will definitely take place once the full system has been integrated with regards to the API.

1.3 API

The API has been written in it's entirety, and tested roughly. In terms of functionality, the API currently supports getting various artist data (both from the game's database and from the last.fm API), getting various user data (primarily through the game's database, but also through the last.fm API), as well as the ability to authenticate a user with last.fm. In addition to simply fetching data, the API also supports the buy/sell mechanics that are fundamental to the game.

With regards to the implementation of the API, the Apache Thrift library was chosen to be used, rather than writing the entire library from scratch. Thrift allows for a declarative approach to the API: a thrift file is created (essentially C-style definitions), which is then used to automatically generate code in your chosen language (in this case python) for the client and server. This has a number of benefits, the first being that changing the API by adding or removing functions becomes vastly easier (a version counter can be incremented as well to maintain client/server consistency). Secondly, using an established library to handle the serialisation and transfer of data over the network means that extensive testing has already been carried out in this area and it's unlikely to have any (major) bugs in the code. Thirdly, the code not require much documentation - the thrift file essentially contains all of the class and methods definitions, commented with their purpose/function and their method signature.

Since the API is difficult to test in a standalone fashion (it would involve essentially testing that thrift works and setting up dummy responses from the API server), no extensive testing to see if the API functions correctly. Once the entire system is integrated and DATM returns actual data from a database, proper unit testing can be carried out by creating a new database, populating it with test data, creating a dummy testing client and then checking that all the methods that the API server provides take the correct arguments and reject any input outside of what was expected.

1.4 DATM

In accordance with the project's structure, DATM has been implemented in response to demand from those working on the frontend and API - that is, the latter have been given free rein to request absolutely any methods that they require and it has been considered DATM's responsibility to declare a suitable signature for use immediately and implement such methods as soon as possible after this fact.

Due to the sheer scope of its implementation, DATM is not yet complete, but it does currently have...

- ... a complete and final public interface, such that all external code (API, Analytics) which need call it in any way may be written against it and will receive either dummy data or a NotImplementedException (which is not caught in the implementing code as it shall not in any way form a part of the final implementation).

  This approach has already been used to great success in implementing the API in its entirety.

- ... a complete data model, to be used to store *all* data persisted by the Scrobble Exchange service as a whole, implemented as objects ready to be mapped into the database as tables by the Object-Relational Mapper used by DATM.

- ... tests written and *run* validating the syntax, internal consistency (foreign key references, etc.) and expected functionality of the aforementioned models.

  These demonstrate that the data model as implemented in objects *is* ready-for-use and only awaits data entry.

- ... a suite of tests and test stubs pre-empting the implementation of the the public interface as used by the API and frontend.

- ... a full set of objects to configure DATM on an application- and session- specific basis, prepared for thread-safety and transactions (using Python context managers).

1.5 Analytics

Analytics as an entity is divided into two parts.

The first calculates the price and the seed number of copies available for artists that have not been sold on the market yet, adjusts the price of artist according to the supply and demand and forms a part of the API.

The second is a semi-external, long-running utility application that runs daily to analyze and update the server's database using data fetched from the last.fm API. It rewards players by paying royalties for their artists' performance as well as aggregating score for a performance metric, used in leaderboards. It also resets data for time-based leaderboards as well as adjusting the number of available copies of artists based upon the size of the playerbase at the time of its execution.

Since the analytics module had been dependent on other modules and more importantly the *absolute* finalisation of mechanics it has yet to be implemented and we expect that its algorithms, upon implementation, will require further testing and balancing. The current iteration of these can be found in the analytics pdf file that comes with this progress report.