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.

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 order to implement the API, the Apache Thrift library was chosen, since this allows for the easy definition of the interface of the API which is then used to automatically generate the majority of the RPC code involved in the API client and library. A Although it's complete as it is, the API is still subject to change if any new features are added last-minute to the project, and due to the fact that this would only require incrementing a version counter and regenerating the source from the definition, this would not involve a lot of work.

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. This functionality has gone through limited testing through the use of a test client for the API, which creates the objects in the API and calls it's methods on the remote server, to which the server returns the arguments provided (if they're valid). This is only a basic test to ensure that the serialisation/deserialisation in the API works correctly, and that it is possible to transfer data over the network.

Once the entire system is integrated, more comprehensive unit tests for the entire system can be created, which would involve creating a new database, populating it with test data, and then testing the functionality the API together with DATM.

1.4 DATM

1.5 Analytics