Meetify Test Plan

Overall Test Plan

Our primary testing strategy is to test individual components of our project using mock data to ensure that the functionality of each component is working and stable. This can be further split up based on the two primary components of the project: **front-end**, **back-end**, and **integration.**

For **front-end testing**, it needs to act independently of the back-end, but it also needs to know what to expect. As such, while the integration API is still being developed, basic mock data will be incorporated to see how data will be presented. As the server API becomes more refined, a server “emulator” containing other mock data in the exact expected form of the server will be made, where we can then visually verify that data is being displayed as designed. The front-end logic is not likely to be complicated, but unit testing may be found to be useful in the future, including automated UI testing. However, this is a stretch goal and not a priority at the time of writing.

In **back-end testing**, data insertion and retrieval needs to be correct so that the rest of the system can handle said data as it needs to. This will be tested using mock data that will be used to fulfill requests made to the server. The matching algorithm, handled by the back-end, will be tested for accuracy and tuned to our liking using mock users with varying Spotify listening history. Other features, such as the messaging system, will also be tested as individual components so that the functionality can be tuned as needed.

Integration testing incorporates both of these components and their interactions. We can of course act upon the server itself, but it will be within our best interest to make a test server for the purposes of testing changes with other mock data. In any case, the front-end will need to be made aware of this configuration, with which we can make changes through the server and test the full operation by making sure we see the appropriate data on the front-end. This will likely be basic confirmation tests to ensure that the front-end can indeed handle the actual communication protocols that the server will provide.