Risk Analysis Document - Database Layer

Team 2

Main Risks

1. If care is not observed in maintaining the username/email_address used to sign in a user unique, users could end up facing issues in logging in/signing up.
2. It's possible that a user could abuse the playlist by accessing other users' accounts. To resolve this, password restrictions should be imposed.
3. The server might not be running at any given time due to it's prototype nature.
4. There's a possibility of database entries becoming stale if the link is taken down (IE: Youtube video disabled or Soundcloud file removed).
5. The shareable link for each playlist could be used by anyone to access the playlist if it is active forever, or if access permissions aren't specified by the host
6. The search for songs requires a search by genre, title and artist. Such searches could become extremely slow if the database grows too large or if titles/artists/genres cease to be unique to songs and we are forced to create composite keys.
7. Each user access multiple playlists and each playlist contains multiple songs. Using a traditional RDBMS database would force us to create multiple rows for each user, and multiple rows for each playlist, making the data very large to efficiently handle.
8. Given our limited knowledge or databases, especially NoSql databases right now, we could up end up under or overestimating our time and manpower required for any of the user stories, possibly delaying our deliverables.

Risk Mitigation

1. Restrict usernames to unique names.
2. Passwords should be restricted to at least 8 character including at least one number
3. Care should be taken to ensure the server is always running. The app accessing the database should carefully expect timeouts.
4. There's no easy or obvious way to check if a Youtube video is available as the page will still load. On Soundcloud the old link will take you to a "removed" page, making it obvious to any script that the audio was removed. A script should be written that will occasionally check the Soundcloud links for this.
5. Shareable links to playlists should only be valid for 24 hours or a manually enterable number of hours.
6. Since our user base is small, we likely will never encounter a significant database slowdown. Depending on the type of queries, we could create indexes on titles/genres/artists to speedup our queries. Also, if we are aware of the kind of queries

we would make, we could use a query-based database like Cassandra or ElasticSearch for such purposes.

7. To address the issue of repetitive rows, we could use a NoSQL database like MongoDB that stores information in JSON formats. However, since we're only storing links and song metadata, our small user base should prevent entry width from being an issue. Even if each entry was 1MB we could easily handle the data requirements of thousands of entries. If we stored entire songs this would be a much bigger issue.
8. We can try to overestimate timelines whenever possible. This is crude but will help prevent issues with deadlines. We should also take a close look at our prototype development time to use as an indicator of actual development time.

Possible Prototypes:

- Create test data and use it for each of the different kinds of databases like MySql, Cassandra and MongoDB. Use these databases to compare access times in order to select the final database used.
- Managing user authentication is a huge task. So, we could try both strategies, one of storing the unique username/email_address and password to our own database, and two, using Google API (OAuth) and assess which one of the two is easily manageable and works more securely and efficiently.
- Compare strategies of storing songs in an in-house databases or using an external service like SoundCloud etc. and assess the time taken and number of songs returned accurately in each one of them. Accordingly make a call on the best method.