Mopidy Web Client - DiveSong

Scope

1. A 24/7 radio taking request from users and playing music at client side.
2. To make current playing song available to multiple nodes.
3. There aren't many good and interactive web-clients for local songs. DiveSong will use local music files and make it available to a remote user.
4. It will be a FOSS application.

Resource Requirement

1. Knowledge of Node JS and React JS.
2. Knowledge of using APIs.
3. Working of Servers and ports.
4. Knowing how to use and implement SMTP mail service.
5. Using SSL and TLS for encryption.
6. Knowledge of databases and querying in sqlite.
7. Configuring Mopidy, Apache and Port Forwarding.

Requirements

Functional

1. Should be playing random songs when nothing in queue based on the previous likes and dislikes.
2. Authenticate users and take inputs like likes, dislikes and requests to play songs.
3. Requested songs list should be sorted based on number of requests, time of requesting and likes/dislikes
4. Users who requested for song to receive a mail of information when their track is going to be played in 5 minutes or less.
5. The UI should be interactive.
6. Songs should categorized based on language, artists, albums and more.
7. Songs could be searched via a search bar.
8. Songs could be downloadable.
9. Most heard song list.
10. History of songs played till 100 tracks.
11. Forgot password implementation.
12. Seeker for current playing song.
13. Can face and recover tracks queue and continue playing after a power outage.

Non-Functional

1. Handling of 50 GiB of data(songs).
2. Low querying time.
3. Mail service should be fast and secure.

4. Availability of server should be high.
5. The list of local songs should refreshed every 10 seconds.
6. Dynamic IP should be configured by DuckDNS.
7. Service available to people inside and outside of local network.
8. The development process should be TDD(Test Driven Development) .
9. The development should be agile.
10. Git will be used for Version Control.
11. GitHub will be used as remote git server.
12. Failure handling.

Completion Criteria

1. The tests coverage should be more than 90%.
2. All the Requirements are satisfied (100% Statement Coverage).
3. The app should be able to run on heterogeneous platforms and browsers(except Internet Explorer) equally well.
4. Service can handle failures.