University of the Pacific

Web Based Music Player

http://localhost:3000

Cloud hosting TBD

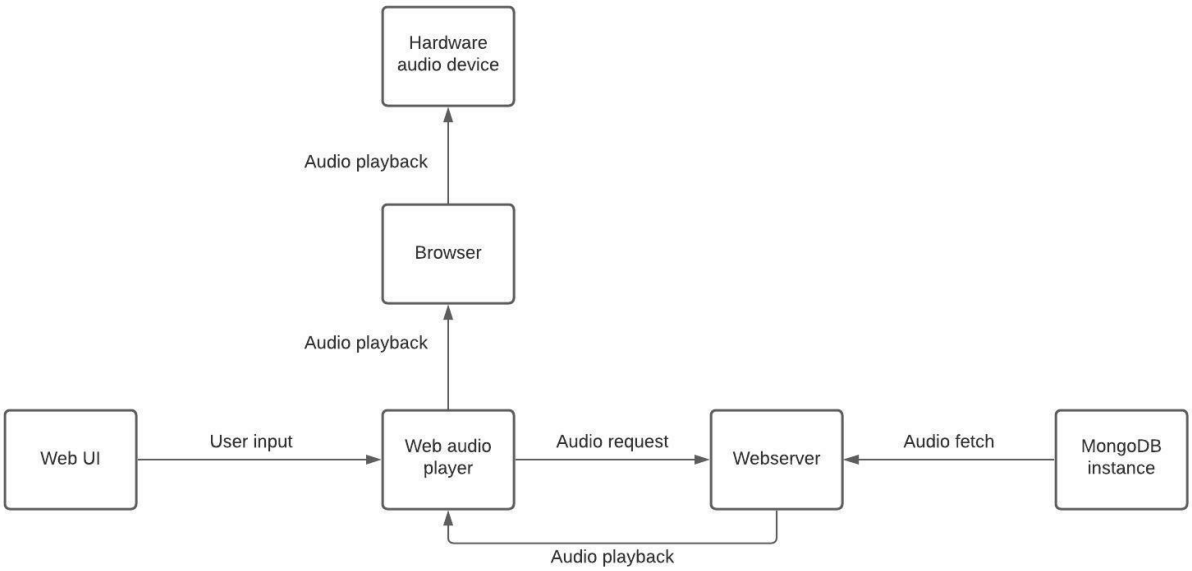
David Leavenworth

d_leavenworth@u.pacific.edu

COMP-195 Senior Project

Date of last modification: 10/24/2021

System Architecture



In brief, the web UI controls the playback of the audio player, and on a song change the audio player will make a request to the webserver, which will fetch the song from the MongoDB instance, and begin playback for the web audio player.

Hardware and Software Requirements

Hardware requirements: A computer capable of running Google Chrome or equivalent browser. Requirements provided by Google online [1]. An internet connection, ethernet or wireless, and some audio playback hardware like speakers or headphones.

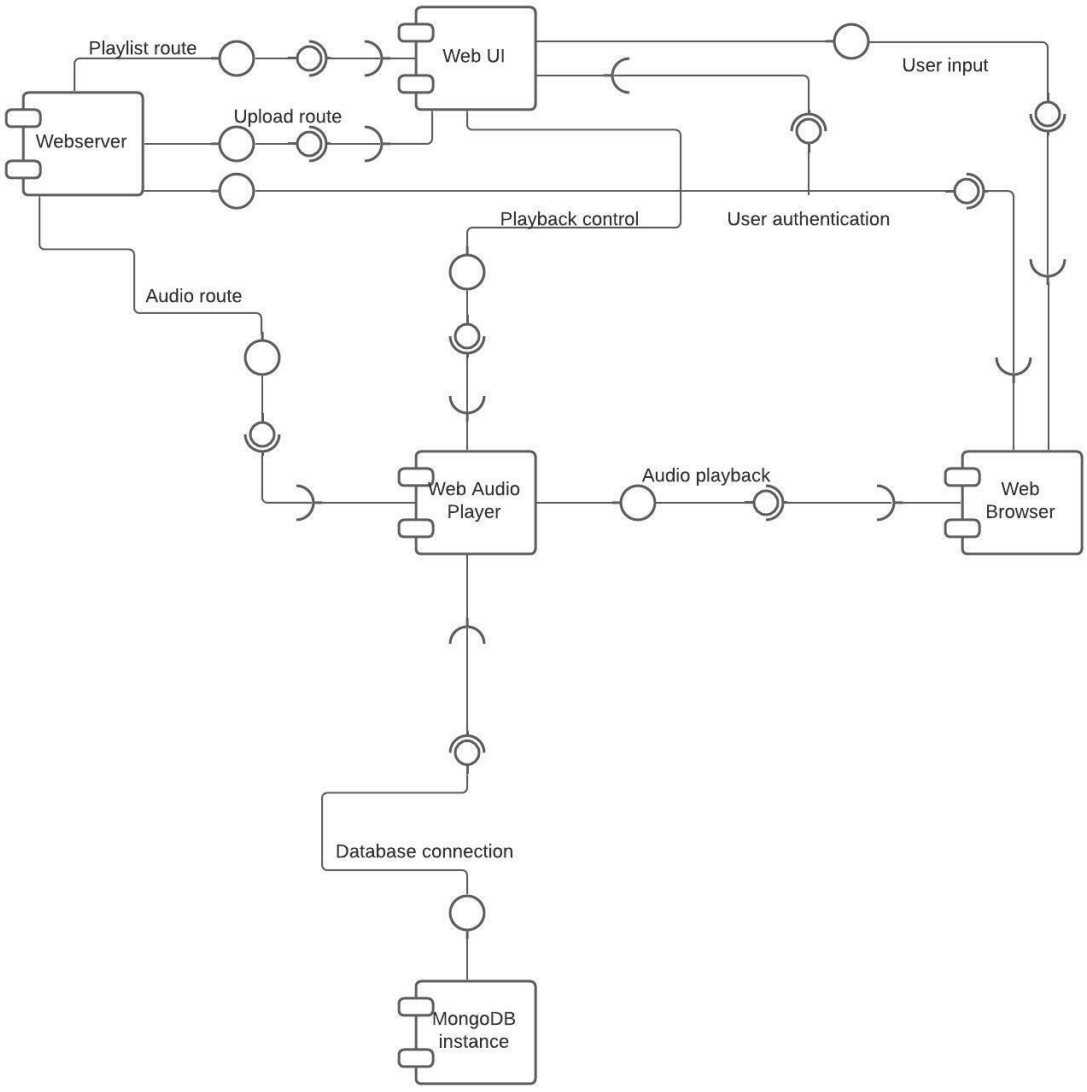
Software requirements: If building locally, Express v4.xx.x, Node.js v14.xx.x, React.js v17.x.x, MongoDB instance of v4.x, Material UI v5.x.x, Axios v0.22.x, React Router DOM v5.x.x, CORS v2.x.x, DOTENV v10.x.x, Multer v1.x.x, Readable Stream v3.x.x. Note: can be installed via $ npm -i.

External Interfaces

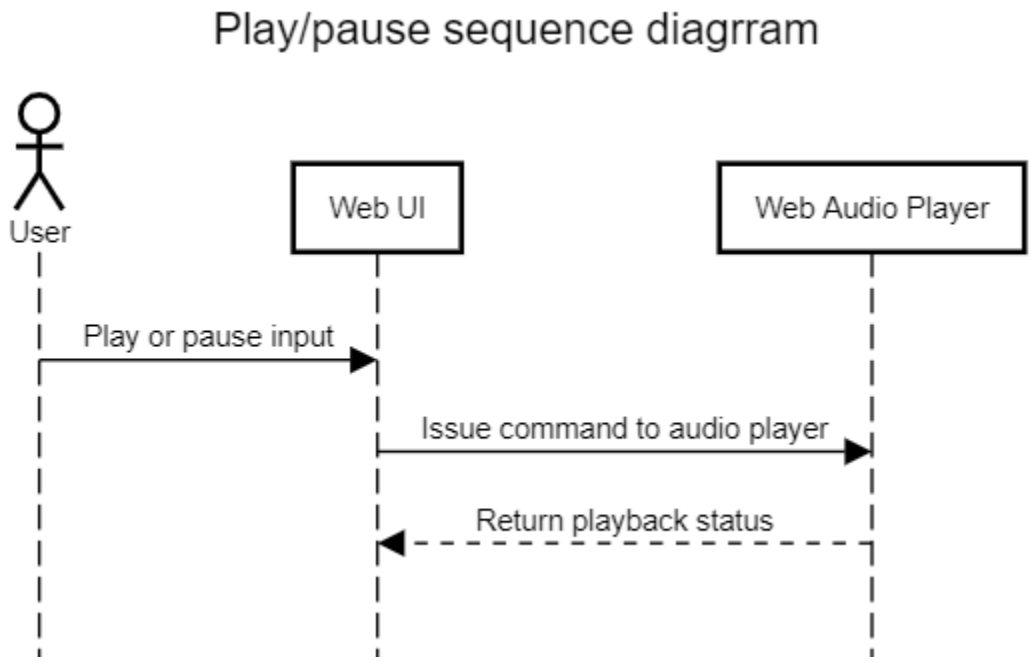
The only external interface required is the communication with the MongoDB instance, which has documentation provided online [2]. Additionally, for this specific project implementation Heroku will be used for cloud hosting. Documentation for Heroku is provided online [3].

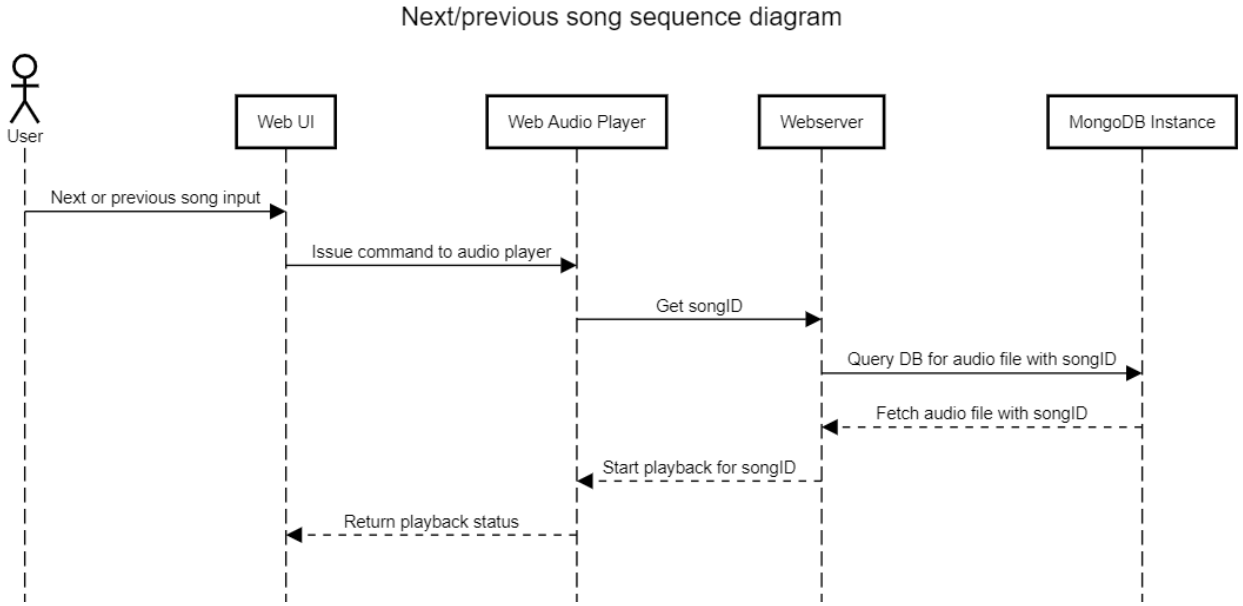
Software Design

Component diagram

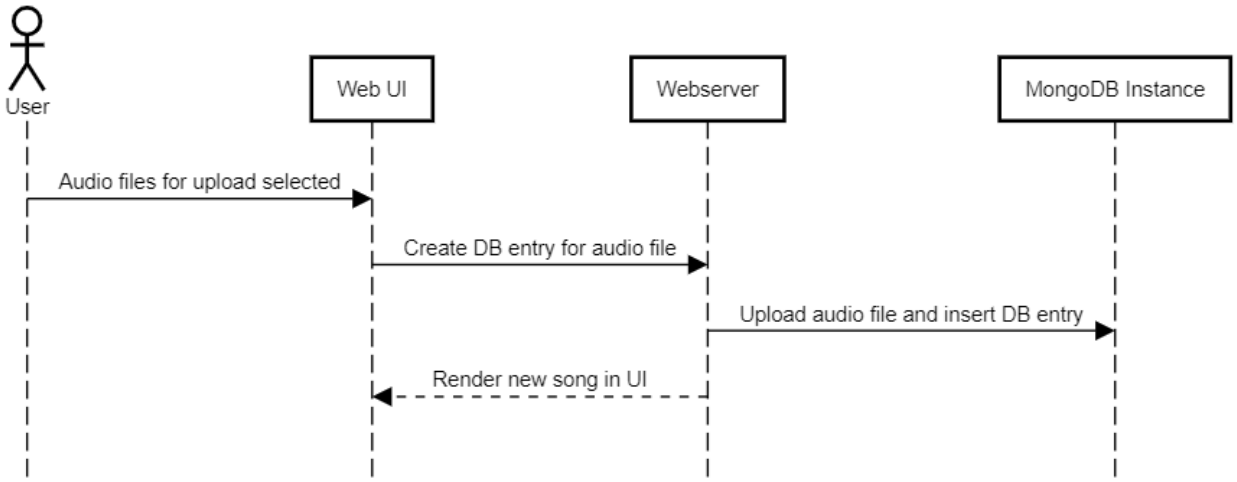


Use case sequence diagrams

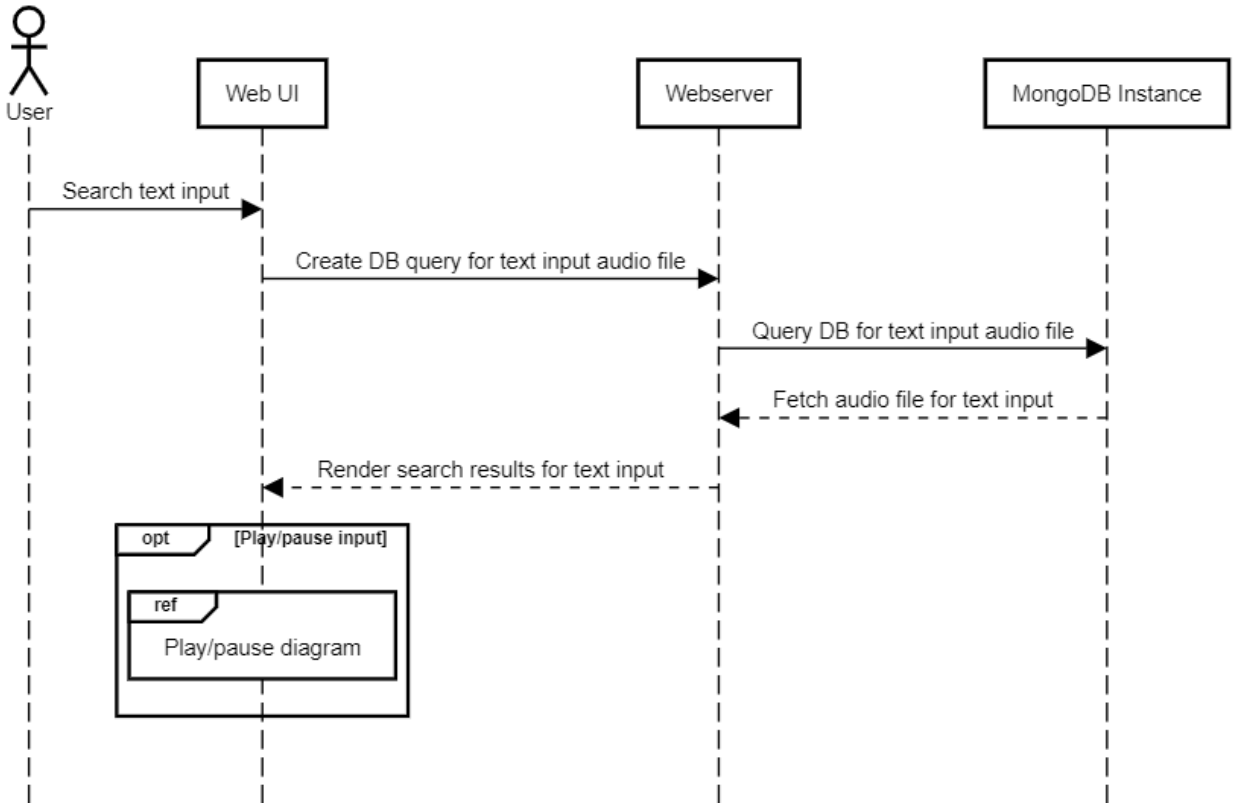


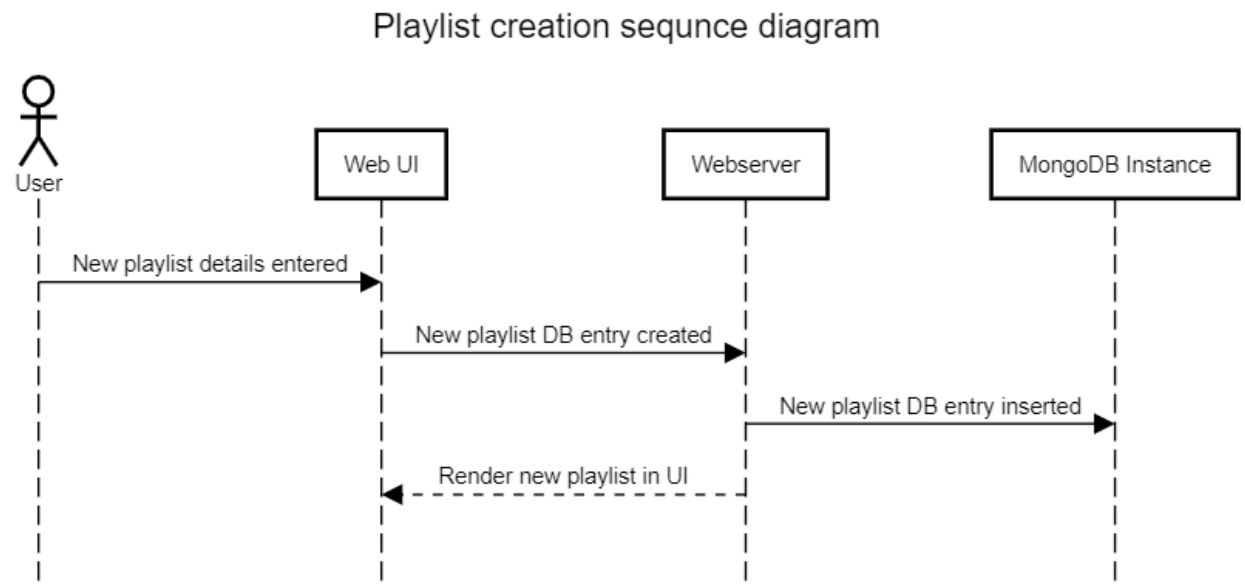


Upload sequence diagram



Search sequence diagram





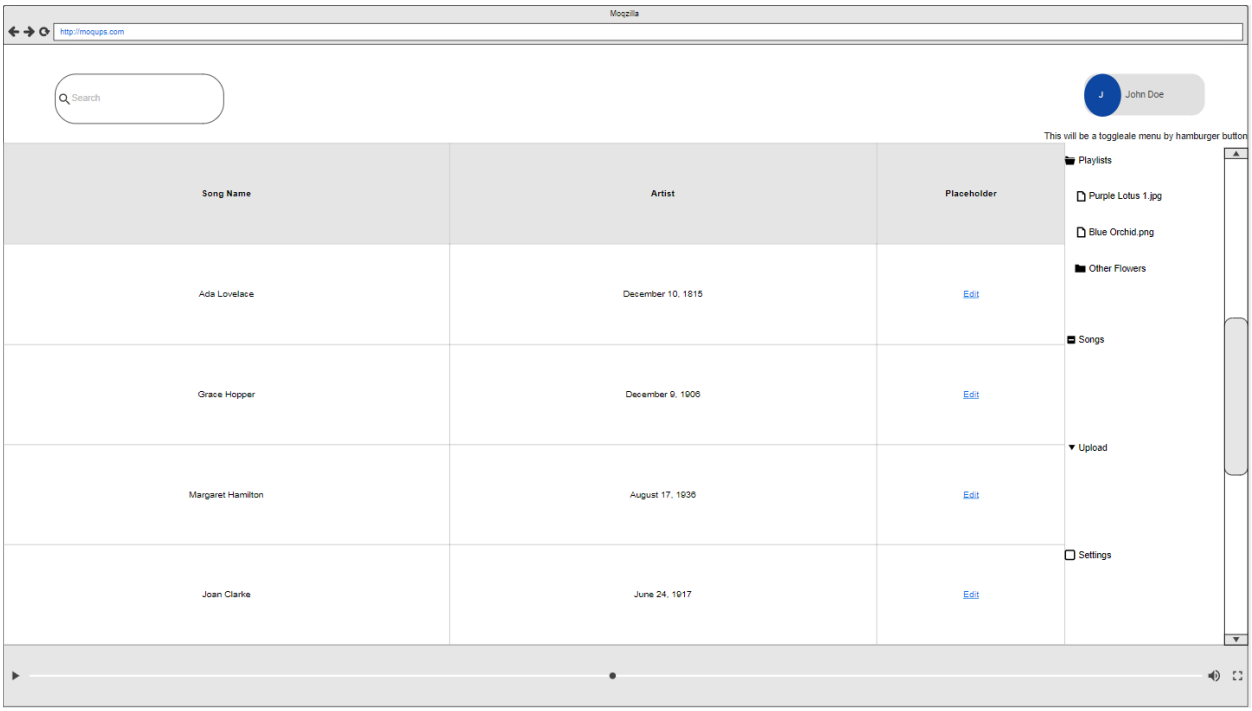
The use cases of shuffle, loop and playlist deletion were omitted due to the simplicity of their diagrams. They are extremely similar with the play/pause diagram.

Design Considerations

The language being used for this project is JavaScript with React.js. React.js is a component-based UI interface library. As such, most of the project planning is being done in the same manner. The impact of this design paradigm can be most heavily seen in the UI mockups, React.js is intended for single page applications, which is why so much of the UI is consistent between mockups. Instead of changing page, the state of the webpage will change. This will also make having an always on audio player significantly easier than it would be with multiple pages which require reloading.

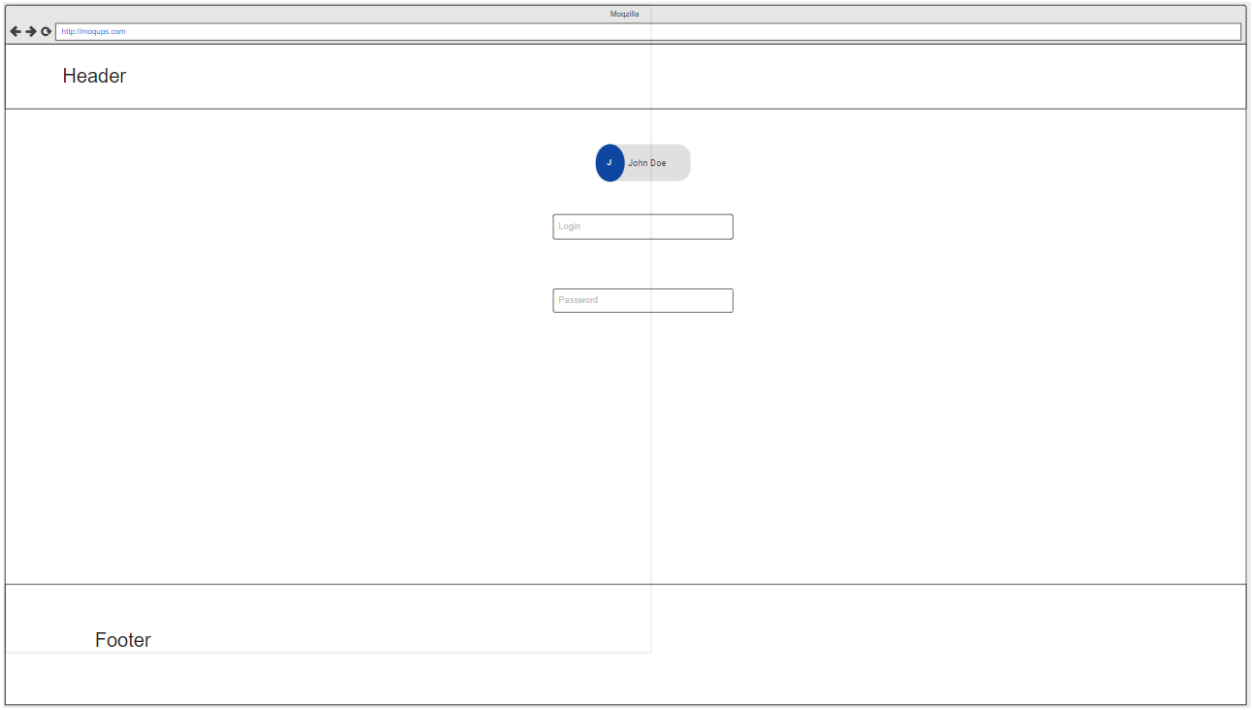
UI Mockups

Song listing page/main landing page:



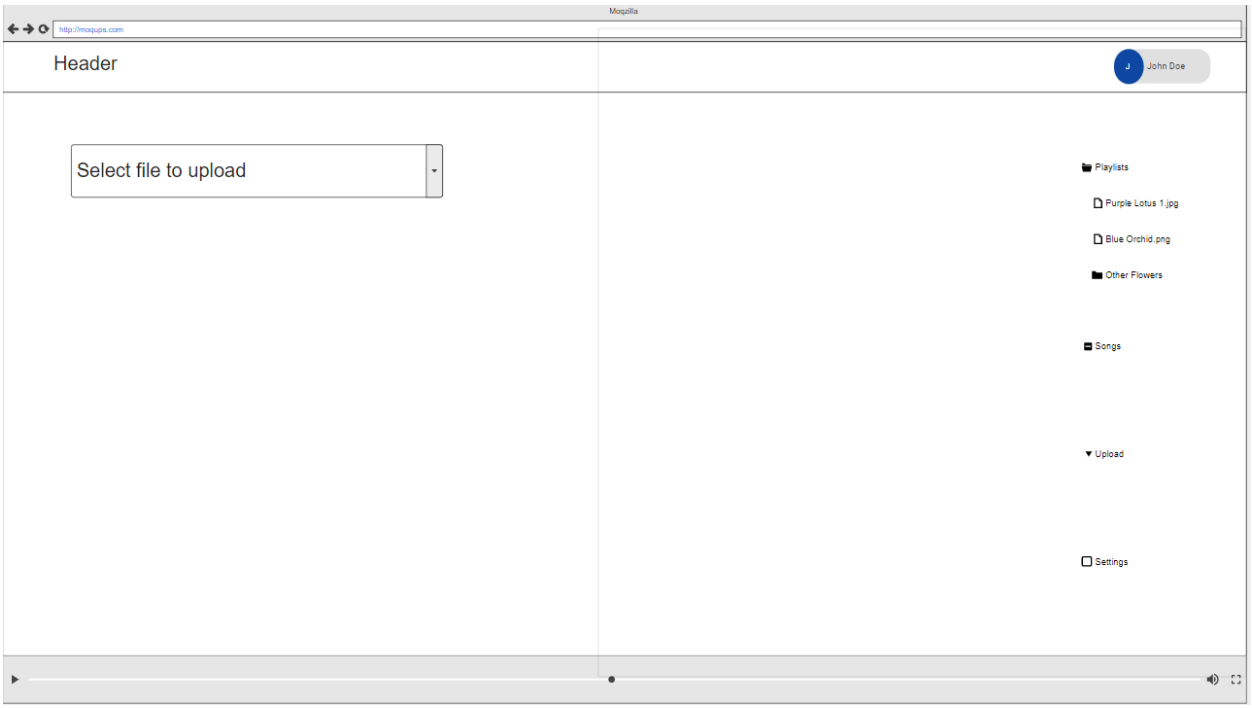
The UI in this mockup will be duplicated across much of the application. The only change will be that on the playlist page, it will be a list of playlists. The search results page will look the same, simply with songs filtered by the search text. When on an individual playlist page, it will display the songs in that playlist. When it is a listing of songs users will be able to select which song, they wish to play by clicking on the text in the placeholder column (this will be changed in the final product of course). When it is a listing of playlists, user's will be able to click on the playlist to browse the selected playlist. Most pages will also have a playback bar as the page footer, the audio will continue playing when moving between pages, so the playback control will stay fixed between pages.

Login page:



User's must authenticate themselves with webserver before gaining access to the song database.

Upload page:



Users will be able to select the files which they wish to upload to the database from the Web UI.

Sources cited

[1] Google. 2021. Chrome browser system requirements – Google Chrome Enterprise Help. Retrieved from https://support.google.com/chrome/a/answer/7100626?hl=en on 9/12/21.

[2] MongoDB. 2021. Welcome to the MongoDB Documentation – MongoDB Documentation. Retrived from https://docs.mongodb.com on 9/12/21.

[3] Heroku. 2021. Documentation | Heroku Dev Center. Retrieved from https://devcenter.heroku.com/categories/reference on 9/12/21.