Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2019

The Underachievers
Synthify

Dominic Young
Kolten Sturgill
Mary Huerta
Mitchel Smith
Endy Pluviose
Minh-Quan Nguyen

Revision History

Revision	Date	Author(s)	Description
0.1	2.08.2019	DY	document creation
1.0	2.26.2019	DY	document release

Contents

List of Figures

List of Tables

1 Introduction

Synthify will get playlists from a user from each platform to bring them all together and have the playlists unified under one service. The service will auto-update after a certain amount of time has passed to make sure it's consistent with any playlist changes that the user may have done on the respective platform. The "look and feel" of Synthify is similar to Spotify's dashboard. Users should expect to listen to music from each platform they have signed up for.

2 System Overview

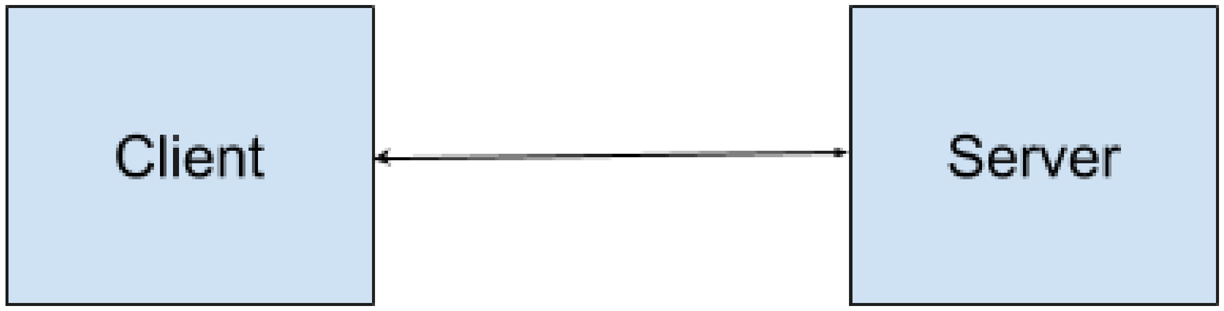

Figure 1: Breakdown of the Application Parts

2.1 Server Description

This layer will contains the resulting data and pulls from 3rd parties which will direct the data to the Client. The persistence frameworks will contain information such as the users log in credentials, previous queries to said music services, and play lists created by the user.

2.2 Client Description

The client will contain redux with will send data to the major viewable interfaces. These components will contain song and playlist information sent down from the server. The user can search for playlists.

3 Client

The client is broken down to different components. The main component for the client will be the Redux state interface, which will keep track of states happening in the application. Other components include logging in as an existing user, registering as a new user, searching through your content once you have connected your accounts from different music platforms, the media player itself, user settings, playlists, oauth, and finally the interface, that will bring it together with the server.

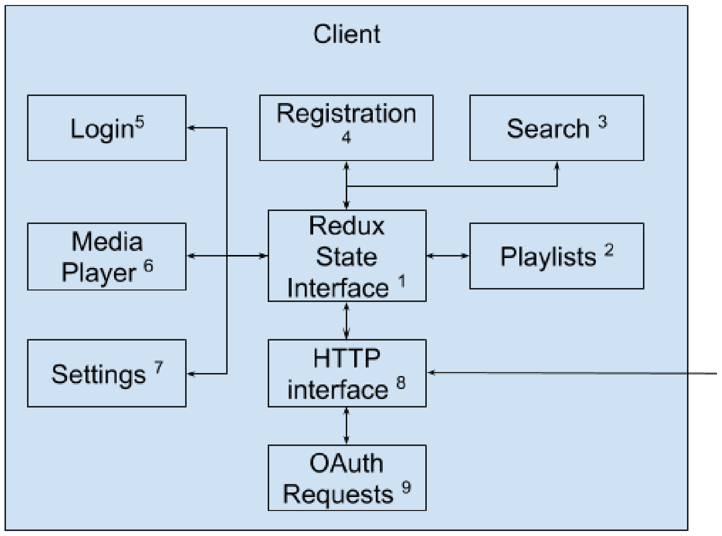

Figure 2: Client Subsystem

3.1 Layer Hardware

This layer will require no hardware

3.2 Layer Operating System

Chrome 42 and FireFox 39

3.3 Layer Software Dependencies

We'll be using React-Redux v6.0.1, React.js 16.8.0-alpha, The Fetch API from Mozilla, Material-UI v3.9.0, React-Router 4.3

3.4 Redux

This subsystem allows the Client Layer to interact with the HTTP Interface. It will give information to all the different subsystems.

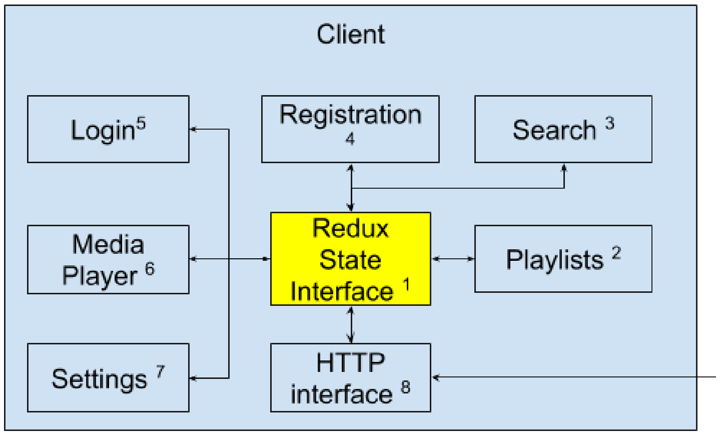

Figure 3: Redux subsystem

3.4.1 Subsystem Operating System

We'll be using Chrome 42 or Firefox 39

3.4.2 Subsystem Software Dependencies

We will be using React-Redux v6.0.1

3.4.3 Subsystem Programming Languages

We will be using JavaScript ES6

3.4.4 Subsystem Data Structures

We will have reducers and actions in Redux. The root reducer will contain the auth reducer, playlist reducer, and search reducer.

3.4.5 Subsystem Data Processing

We'll be using the fetch API to get data from the server in the form of JSON. It will return a JSON Ojbect back and compare it to the whole redux state and change the state where there are differences.

3.5 Playlist

This subsystem will contain all the information on a users playlists and display it to them.

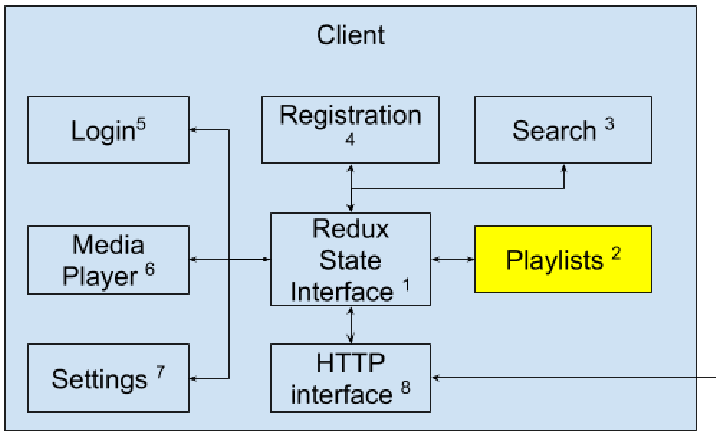

Figure 4: Playlist subsystem

3.5.1 Subsystem Hardware

No hardware is used for this layer. The playlist component will be a component of the home page which will be hosted on Heroku.

3.5.2 Subsystem Operating System

No OS is required in order for the playlist component to work properly.

3.5.3 Subsystem Software Dependencies

We will be using React.js 16.8.0-alpha.1 for our framework and will be using the Fetch API from Mozilla.

3.5.4 Subsystem Programming Languages

We will be using JavaScript ES6

3.5.5 Subsystem Data Structures

The data structure of the playlist is going to be an object that contains a playlist name as the key with its value being an object of song names and artists. This structure may change depending on what various music services API returns.

3.5.6 Subsystem Data Processing

When a use chooses a playlist that they would like to view, the Front End will take that choice in order to display the proper playlist by comparing it to the stored music data.

3.6 Search

This subsection will allow users to search for playlists on their music accounts through Synthify.

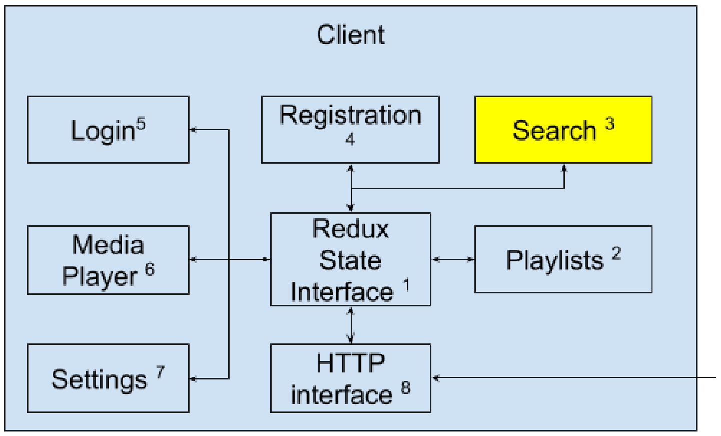

Figure 5: Search subsystem

3.6.1 Subsystem Hardware

No specific hardware will be used for searching.

3.6.2 Subsystem Operating System

No OS required. This will run through supported browsers such as Chrome 42 and Firefox 39.

3.6.3 Subsystem Software Dependencies

We will be using React.js 16.8.0-alpha, React-Router 4.3, Material-UI v3.9.0

3.6.4 Subsystem Programming Languages

JavaScript ES6 will be used.

3.6.5 Subsystem Data Structures

A fetch request will be made to the server. It will return an array of hashmaps that contains information such as artist name, song title, cover art and more.

3.7 Login

This subsystem is a graphical user interface that allows a user to log into Synthify

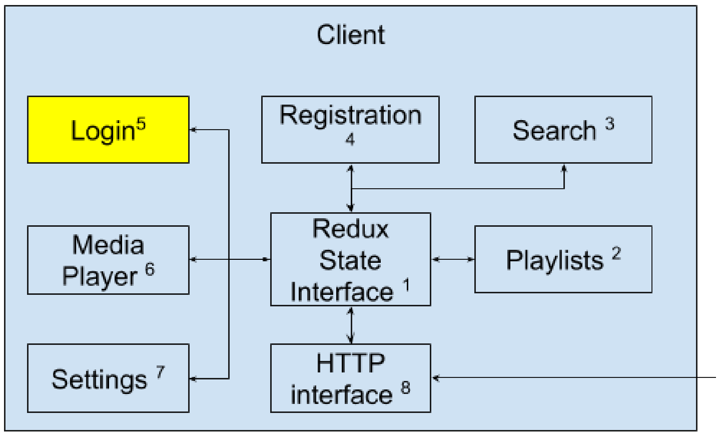

Figure 6: Login subsystem

3.7.1 Subsystem Hardware

There is no specific hardware required for the login feature.

3.7.2 Subsystem Operating System

No operating systems but we will be using FireFox 42 and Chrome 39

3.7.3 Subsystem Software Dependencies

We are using React.js 16.8.0-alpha, React-Router 4.3, Material-UI v3.9.0,

3.7.4 Subsystem Programming Languages

JavaScript ES6 will be used.

3.7.5 Subsystem Data Structures

We will be using a Json Object with the user's email and password information.

3.7.6 Subsystem Data Processing

When user submit their credential we will make a database call to search the database and validate the credential.

3.8 Media Player

This subsystem is apart of the graphical interface that allows the user to play/stop a song streaming from a third party music service.

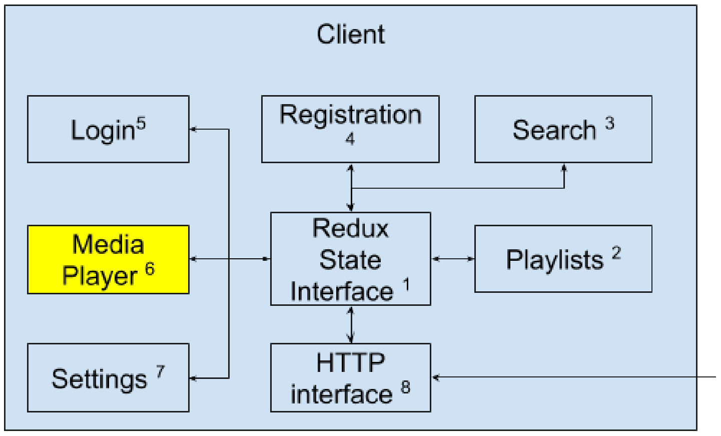

Figure 7: Media player subsystem

3.8.1 Subsystem Hardware

No hardware is used for this layer. The media player will be a component of the home page which will be hosted on Heroku.

3.8.2 Subsystem Operating System

No OS required. The media player will run through supported browsers such as Chrome 42 and Firefox 39.

3.8.3 Subsystem Software Dependencies

React.js 16.8.0-alpha.1 is used for the framework and the Fetch API from Mozilla will be used to retrieve song information.

3.8.4 Subsystem Programming Languages

JavaScript ES6 will be used.

3.8.5 Subsystem Data Structures

A JSON object containing music information will be used to display song information and to stream audio.

3.8.6 Subsystem Data Processing

No algorithms used.

3.9 Settings

The settings subsystem allows users to changed their user account information. A user can add more music accounts (SoundCloud, Spotify, etc), logout, and change the theme.

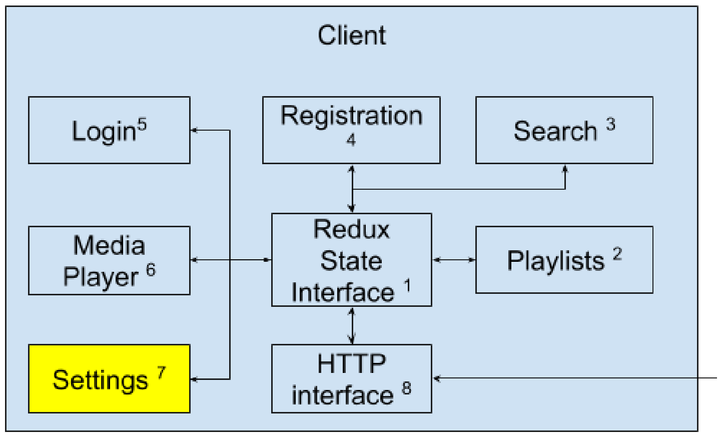

Figure 8: Settings subsystem

3.9.1 Subsystem Hardware

No hardware is used for this layer. The setting component will be a component of the home page which will be hosted on Heroku.

3.9.2 Subsystem Operating System

No OS is required in order for the playlist component to work properly.

3.9.3 Subsystem Software Dependencies

We will be using React.js 16.8.0-alpha.1 for our framework and will be using the Fetch API from Mozilla.

3.9.4 Subsystem Programming Languages

We will be using JavaScript ES6

3.9.5 Subsystem Data Structures

A list of different configuration of Synthify that the user can use to customize their experience.

3.10 HTTP Interface

The client side HTTP interface will use the built in Fetch JavaScript functionality to send and receive resources from and to the server.

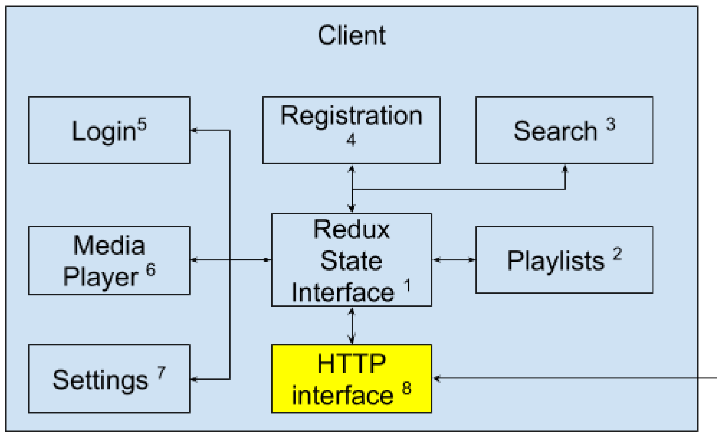

Figure 9: Client-side HTTP interface subsystem

3.10.1 Subsystem Operating System

We need the Fetch API from Mozilla and will be using Chrome 42 or Firefox 39

3.10.2 Subsystem Software Dependencies

We will be using React.js 16.8.0-alpha.1 for our framework and will be using the Fetch API from Mozilla.

3.10.3 Subsystem Programming Languages

A description of any programming languages used is JavaScript ES6

3.10.4 Subsystem Data Structures

The first data structure is the a resource request with fetch(url-from-server) The second data structure is the Response from the server which will be a JSON (key-pair values) object with the authorization, music,and specific playlists requested

4 Server

The server is broken down to being a database, controllers for the endpoints, a cache for preserving past results (so that we don't make unnecessary requests each time), a queue to keep track of the requests going out and what order they should be returned in, and finally the interface, that will bring it together with the client.

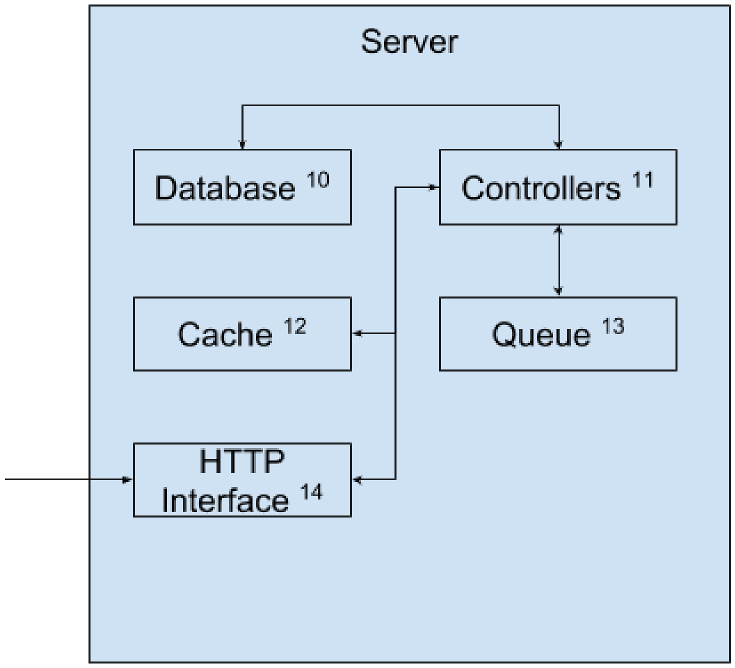

Figure 10: Server Subsystem

4.1 Database

The database will be used to keep track of user info which includes name, email, passwords, connections [to different services], and user preferences.

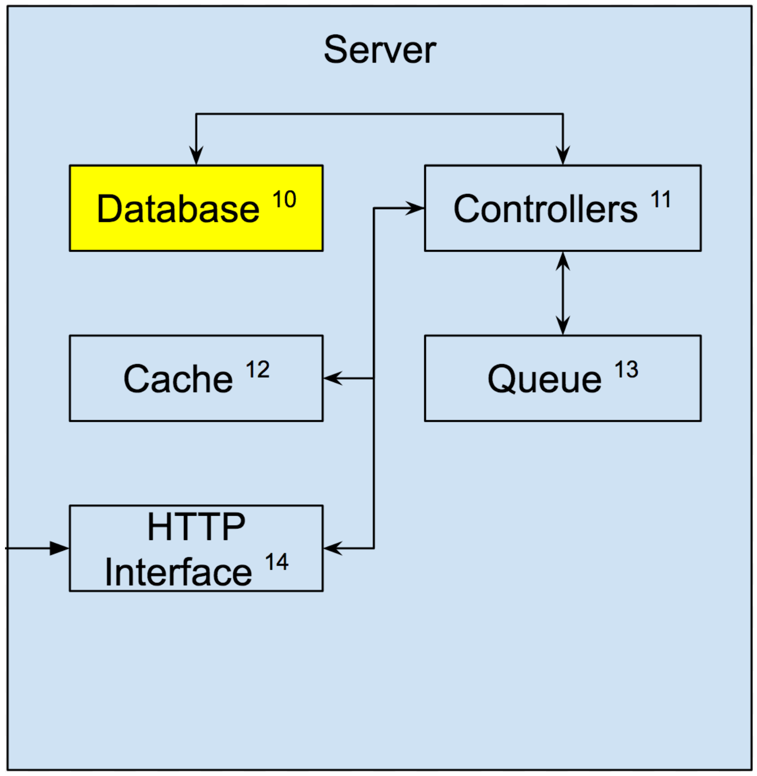

Figure 11: Database subsystem

4.1.1 Subsystem Hardware

This project is primarily software, but we will be hosting our live site using Heroku, so whatever hardware they are using for their servers is our hardware.

4.1.2 Subsystem Operating System

This project is primarily software, but we will be hosting our live site using Heroku, so whatever OS they are using for their servers (usually UNIX systems such as macOS, Linux) is our OS.

4.1.3 Subsystem Software Dependencies

Libraries

- PostgreSQL	
- jwt	"github.com/dgrijalva/jwt-go"
- bcrypt	"golang.org/x/crypto/bcrypt"
- pq (postgres driver)	"github.com/lib/pq"

Frameworks

- spotify	"github.com/zmb3/spotify"

4.1.4 Subsystem Programming Languages

The database interaction models are written in Go, but we are using PostgreSQL under the hood.

4.1.5 Subsystem Data Structures

We have a User object that will represent a user. Information includes name, email, and password. We also keep track of other attributes on our own in the back-end (when the user was created, the last time any attribute of a user was updated, and access tokens for accessing the API for each platform.)

4.1.6 Subsystem Data Processing

Once a user has signed in to the service, their information from their connected services will be used to fetch information from each respective service. Stored access tokens will be sent from the database and server, to the client as cookies (which are basically key,vaue pairs). So Hash maps will be the primary data structure

4.2 Controller

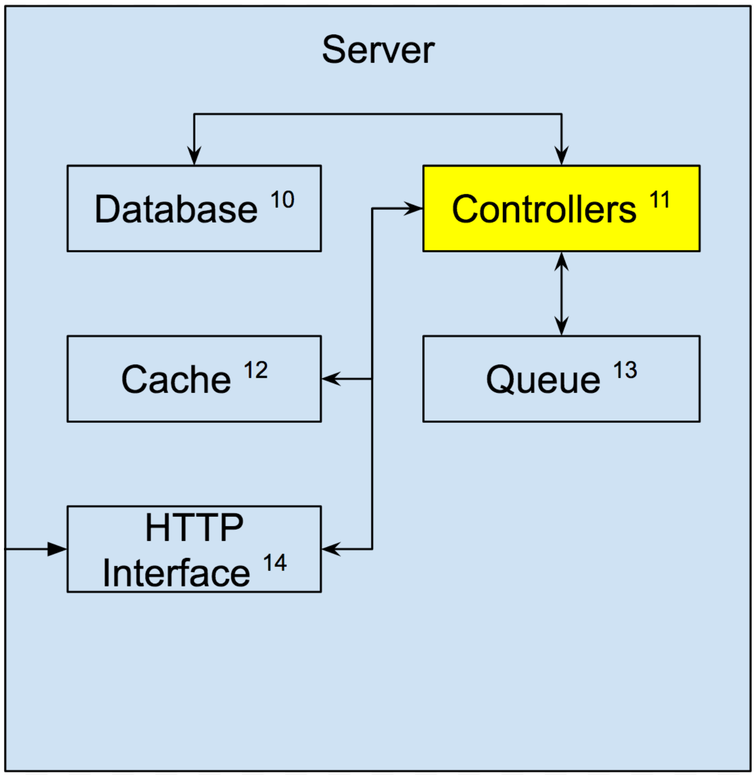

Figure 12:

4.2.1 Subsystem Operating System

This project is primarily software, but we will be hosting our live site using Heroku, so whatever OS they are using for their servers (usually UNIX systems such as macOS, Linux) is our OS.

4.2.2 Subsystem Software Dependencies

Libraries
- Gorilla Mux

4.2.3 Subsystem Programming Languages

The programming language to be used will be Go. Go is a modern and great programming language to work with and will satisfy our needs to provide an implementation of controller methods.

4.2.4 Subsystem Data Structures

The primary data structures that will be returned from the controller will be hash maps and arrays. Internally, the controller will use structs (similar to objects in Object Oriented Programming languages) and arrays.

4.2.5 Subsystem Data Processing

To maintain authentication, we use a strategy where we read in each requests header to verify that the user has valid and non-expired token. A token in this context will either allow the user to access the requested resources or direct them to re-authorize. Afterwards, we validate the user has sent in a valid JSON object, and convert it to a map in Go.

4.3 Cache

The cache will be utilized so that we do not have to make unnecessary requests to the connections when we are grabbing a user content on the respective service

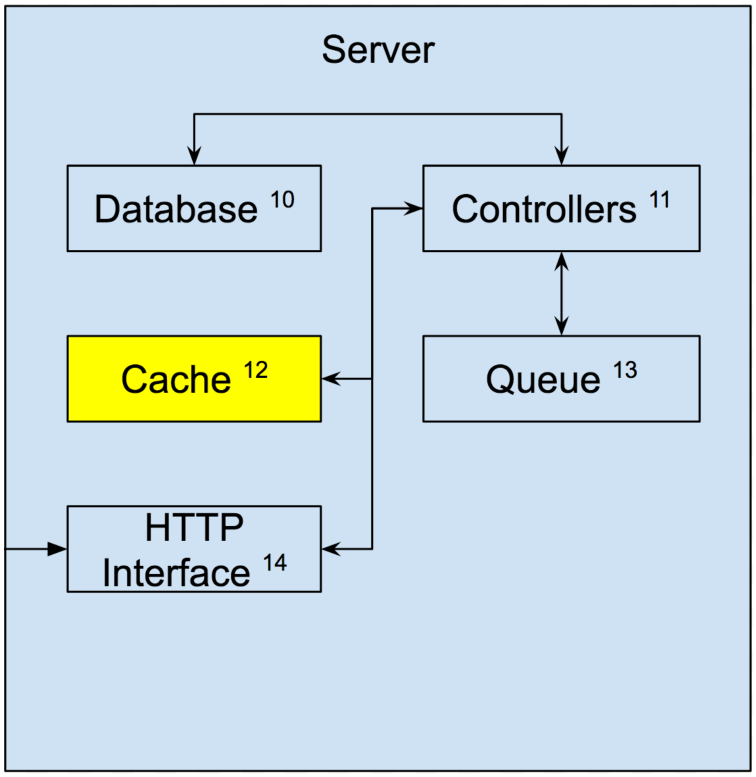

Figure 13: Cache subsystem

4.3.1 Subsystem Operating System

N/A

4.3.2 Subsystem Software Dependencies

- Redis, "https://github.com/go-redis/redis"

4.3.3 Subsystem Programming Languages

Go

4.3.4 Subsystem Data Structures

Redis has many data structure built in that will provide us faster access to previously requested data, such as key-pair value maps with the option to expire, and weighted/sorted lists.

4.3.5 Subsystem Data Processing

In the event that our service via the Controller(s) goes out to one of the music services, we want to store that result in Redis in a key-pair value that will expire in 3-4 minutes later for later access.

4.4 Queue

The queue will handle keeping track of requests made and their responses in the order that they are made.

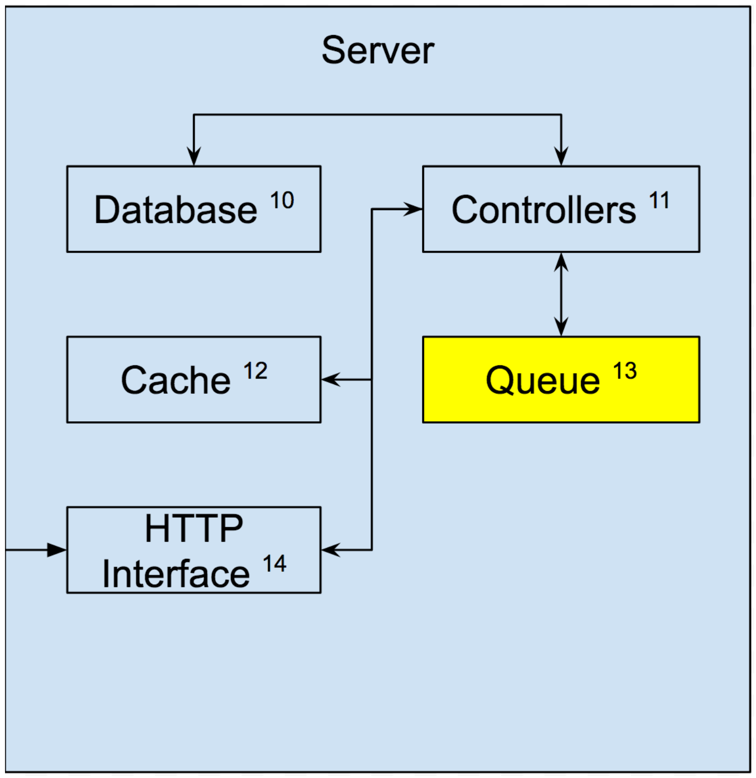

Figure 14: Queue subsystem

4.4.1 Subsystem Hardware

This project is primarily software, but we will be hosting our live site using Heroku, so whatever hardware they are using for their servers is our hardware.

4.4.2 Subsystem Operating System

This project is primarily software, but we will be hosting our live site using Heroku, so whatever OS they are using for their servers (usually UNIX systems such as macOS, Linux) is our OS.

4.4.3 Subsystem Software Dependencies

The queue will be implemented as a normal queue (First In First Out) in our chosen programming language, and as such, will not have any dependencies.

4.4.4 Subsystem Programming Languages

The programming language to be used will be Go. Go is a modern and great programming language to work with and will satisfy our needs to provide an implementation of a request queue.

4.4.5 Subsystem Data Structures

We will be utilizing a regular queue for our request queuing. The data structure used for the queuing will be arrays, and each request that comes through will be appended to our arrays. Then we will be popping the arrays from the front when we are ready to handle the next request.

4.4.6 Subsystem Data Processing

The data will be processed in the same way a queue processes data: "First In First Out".

4.5 HTTP Interface

This interface will be the library used for the server that will direct the incoming request to the controllers via a HTTP/TCP socket.

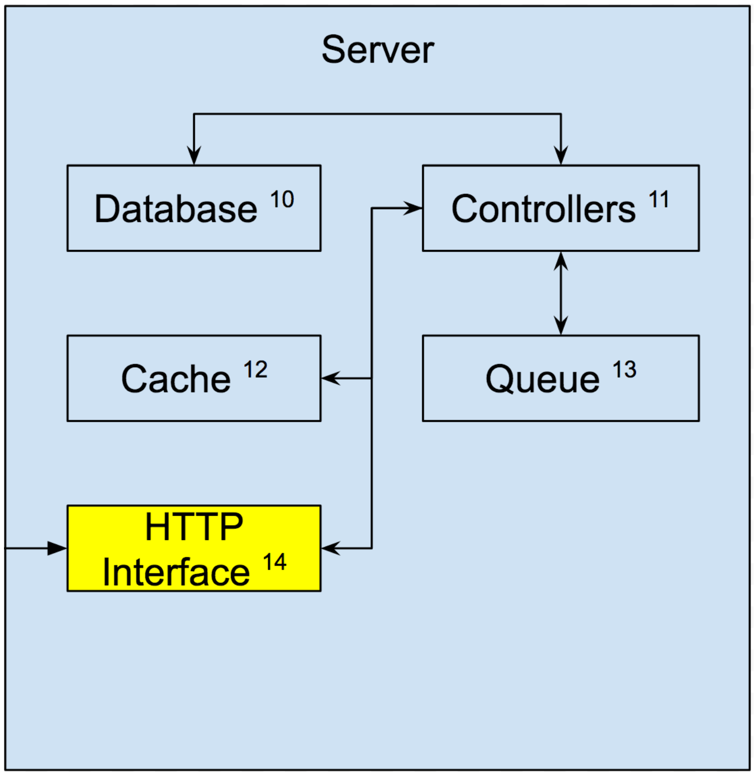

Figure 15: Server-side HTTP Interface subsystem

References