Sportifry

Software Architecture Document

Version 1.0

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 27/11/2021 | 1.0 | -components/classes structures and relationships have been included already  - implementation and deployment views are not specified | All members |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[**Introduction**](#_gjdgxs) **5**

[1.1. Purpose:](#_mvavbgtf6n70) 5

[1.2. Scope:](#_8dx4g6dm3od8) 5

[1.3. Definitions, acronyms and abbreviations](#_hvbqerbcj09y) 5

[1.4. References:](#_xn0gppc4x9c4) 5

[1.5. General view:](#_63r8rz1gb6sx) 5

[**Architectural Goals and Constraints**](#_30j0zll) **6**

[**Use-Case Model**](#_1fob9te) **7**

[**Logical View**](#_phmh24xs91jx) **8**

[Component: GUI](#_qmukl3f1433) 9

[Component: Model](#_3bbidrelqepy) 9

[User](#_dhi9n4970n3z) 10

[Admin](#_eq01nic80lit) 10

[Musician](#_f6jugku1lajf) 10

[Guest](#_h7wbnaa7vhw1) 11

[Music](#_tj4rlxlqoa3q) 11

[Chatbox](#_hhhpdjsfv0by) 11

[Todolist](#_k167k91zb5t2) 11

[Component: Controller](#_6lapck6bf8k0) 11

[Login](#_n4j5evn586yr) 12

[SignUp](#_gndci4uzr5qj) 12

[ViewDetailSong](#_8ug4iwjx7eo) 13

[playMusic](#_d4el89k2joxm) 13

[SearchMusic](#_ihzho46gqqjr) 14

[AddMusic](#_cpj36zwxnbyi) 14

[RemoveMusic](#_s23zefg4z3tf) 15

[CreateTodolist](#_85jaml9z8zc9) 15

[ReadTodolist](#_djuyk75m80hc) 16

[UpdateTodolist](#_9k9rh6fndhpl) 16

[DeleteTodolist](#_vkfn3z4wdam) 17

[DeleteAccount](#_8rnzo7m8xzyu) 17

[Comment](#_dnn0jniubtd1) 18

[Chat](#_yei0wc8383o) 18

[Component: View](#_8jcdyexyey9l) 19

[Component: APIs](#_6ebeayrl1tva) 19

[Component: Firebase](#_xaxfmgdak9v) 19

[**Deployment**](#_3dy6vkm) **20**

[**Implementation View**](#_1t3h5sf) **20**

Software Architecture Document

# Introduction

Sportifry provides a platform for people to listen to music from a variety of music sources while organizing and tracking their daily tasks, housework and study or discussing with other people about new music releases on other platforms to catch up with the increasing tendency of “music - study with me”.

This document defines the goals of this web application architecture, including a brief description of use-cases of the application, an illustration of the structures of this application’s components, classes and relationships between them, and a general implementation. In addition, the basis that accounts for the web application architecture are also mentioned in this document, from conceptual ideas to brief implementation.

## 1.1. Purpose:

This document provides an overview of the application architecture, including the components diagram and

class diagram with different approaches with many significant decisions have been made that have impacts

on the application’s architecture.

## 1.2. Scope:

The scope of this document is to elucidate the architecture of the Sportifry Application, including architecturally significant aspects that are fundamental to the development and implementation of the application.

## 1.3. Definitions, acronyms and abbreviations

MVC stands for Model-View-Controller architecture.

HTTP stands for HyperText Transfer Protocol

API stands for Application Programming Interface

## 1.4. References:

Use-cases Specification Document: [Use-case Specification](https://docs.google.com/document/d/1JQWaPJqWHhU6hO5i0MjR4RLGX4deoG87jau3dOUjqX8/edit?usp=sharing)

## 1.5. General view:

The document provides an analysis of the architecture of Sportifry and contains the following sections:

***Section 2***: describes the software requirements and objectives of the architecture construction.

***Section 3:*** describes the use-case model.

***Section 4:*** describes the architecture with components, classes and relationships between them.

***Section 5:*** describes the deployment.

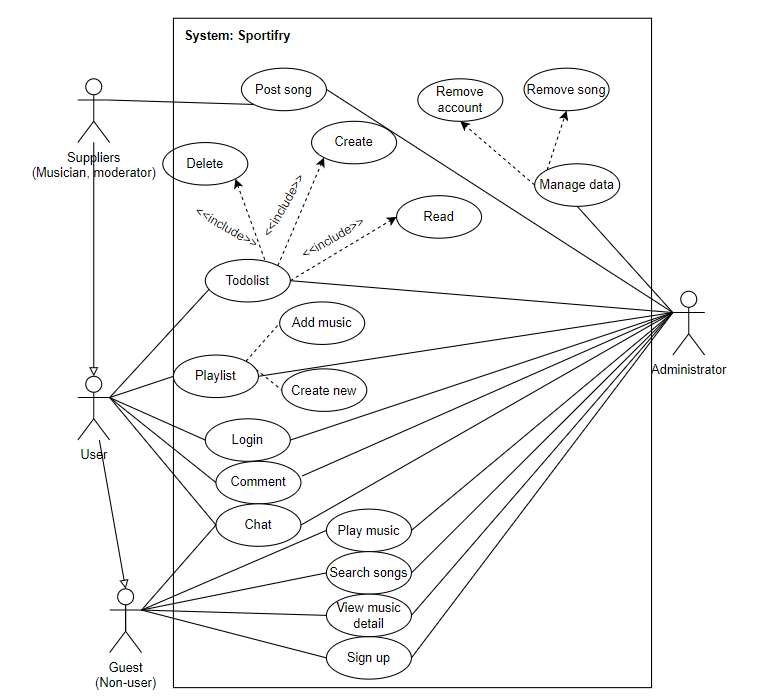
***Section 6:*** describes the implementation view.

# Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system is meant as a proof of concept for a more complete project prediction system to be built in the future. Therefore one of the primary stakeholders in this document and the system as a whole are future architects and designers, not necessarily users as is normally the case. As a result, one goal of this document is to be useful to future architects and designers.
2. One of the primary goals of the system architecture is to minimize the impact of these changes by minimizing the amount of code that would need to be modified to implement them. The architecture seeks to do this through the use of modularization and information hiding to isolate components that are likely to change from the rest of the system.
3. The System must ensure complete protection of data from unauthorized access. All remote accesses are subject to user identification and password control.
4. All performance and loading requirements, as stipulated in the Vision Document, must be taken into consideration as the architecture is being developed.

# Use-Case Model

**

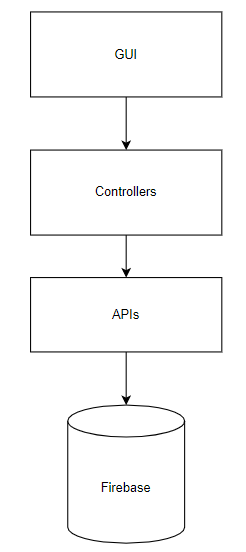
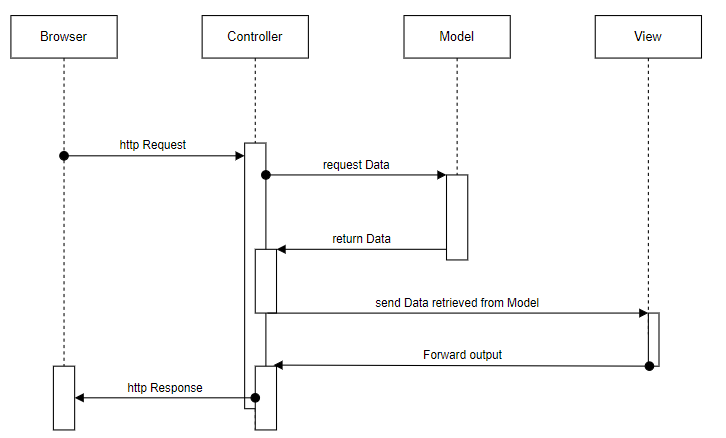
# 

# Logical View

The system will be designed according to MVC model:

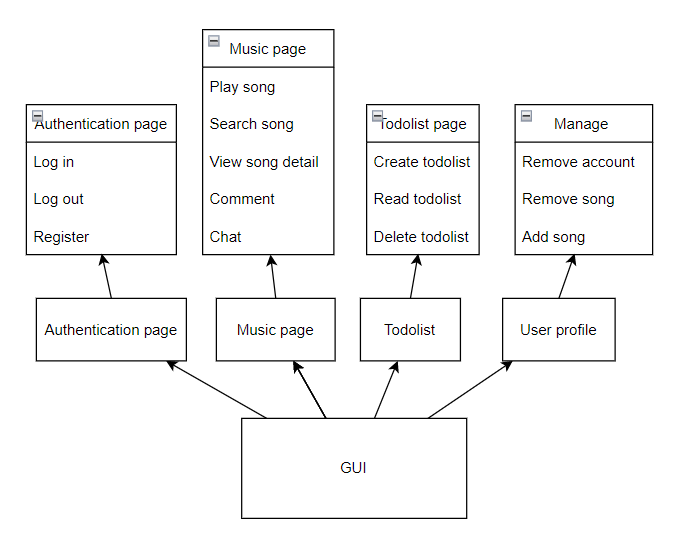
* Model component
* View component
* Controller component

The user will send a request from the Controller, then the controller will send a request to get data from Model, Model will access the database and send data back to Controller. Controller will handle and send to View, View will handle data to display.



## 

## Component: GUI



In GUI component, we design all have 4 main pages:

* Authentication page: this page contains buttons and input boxes for users to perform functions: login, log-out, register.
* Music page: This is the main display page that users focus on during use. This page will display all the interfaces related to listening to music such as: play a song, search a song, view detail and the interface of additional support functions to increase the experience of human-to-human interaction
* Todolist page:is the interface that performs the todolist function, functions with buttons to connect specific functions, depending on the specific function, it can also display the input content area (eg creating todolist)
* Manage page: This page has a simple design, in this page shows content related to managing songs and users

## Component: Model, Controller :

### **User**

1. Attribute:

* ID: to identify the user
* username: user’s name, this name is also the display name of the user
* password: password of the user
* role: identify user type (admin, musician, user)
* playlist: contains a list of playlists the person has listened to
* todolist: list of todo lists which user has saved

1. Function:

* SignIn
* comment
* chat

### **Admin**

1. Attribute:

* This class is inherited from the user.

1. Function:

* createMusic
* deleteMusic
* deleteAccount

### **Musician**

1. Attribute:

* This class is inherited from the user.

1. Function:

* createMusic

### **Guest**

1. Attribute:
2. Function:

* SignIn
* SignUp

### **Music**

1. Attribute:

* ID: to identify the song
* title: the song’s name
* thumbnail: reduced-size image, represent the song
* source: url path
* author: who writes that song
* lyrics: the lyrics of that song
* The list of comment about that song
* isPlaying: boolean value, check if that song is playing or not

1. Function:

* controlMusic
* searchMusic
* createPlaylist

### **Chatbox**

1. Attribute:

* message: list of message which has sent before
* id\_users: id of 2 users who participant that chatbox
* name\_users: stores usernames, this also use to show in chat box

1. Function:

* sendMessage
* readMessage

### **Todolist**

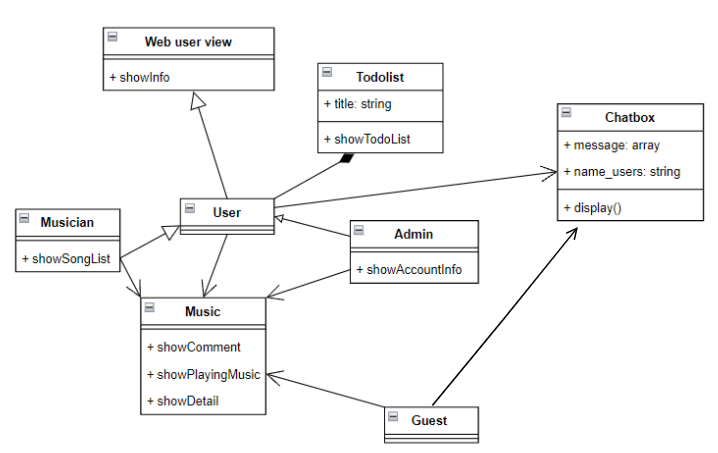
1. Attribute:

* id: to identify the todolist in database
* title: user set this name, requirement attribute
* content: content of todolist

1. Function:

* create
* read
* delete

## Component: View



## Component: APIs

Use react's api to simplify some data extraction and processing operations, and at the same time make the construction process simpler and less time consuming.

* addDoc()
* arrayUnion()
* arrayRemove()
* collection()
* deleteDoc()
* doc()
* getDoc()
* getDocs()
* getFirestore()
* onSnapshot()
* query()
* setDoc()
* updateDoc()
* where()

## Component: Firebase

Firebase is a simple, easy-to-use data management software that fully supports methods to connect to the system. so firebase is used as the database of the software.

Format database: there are 5 tables

1. User:

* id\_user: string
* user\_name: string
* password: string
* role: string
* playlist\_array: array
* todo\_array: array

1. Role:

* id\_role: string
* role\_name: string

1. Playlist:

* id\_playlist: string
* playlist\_name: string
* playlist\_image: image file
* song\_array: array

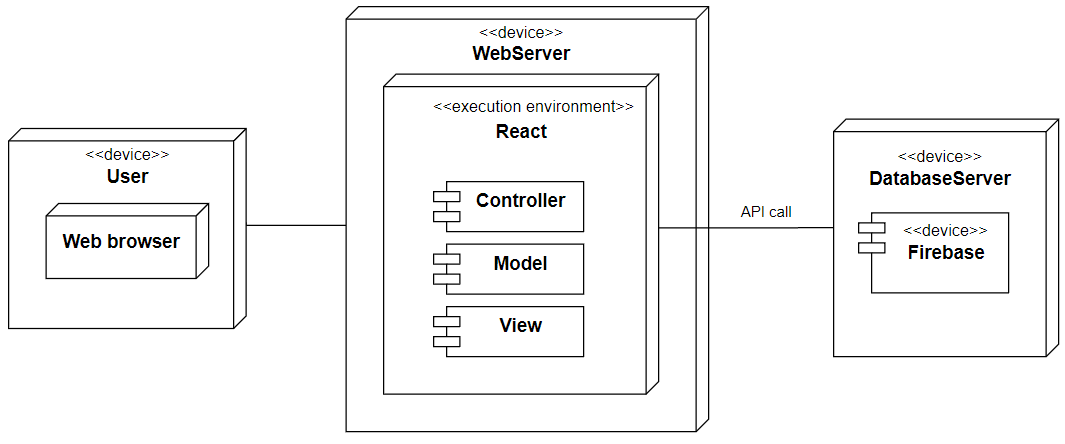
1. Song:

* id\_song: string
* song\_name: string
* singer: string
* link: string
* song\_image: image file
* lyrics: string
* comment\_array: array

1. Todolist:

* id\_todo: string
* title: string
* content: string

# Deployment



* On the user side will use a web browser to access and communicate with the server by HTTP protocol.
* On the web server use HTTP protocol to connect with users. Environment in the server is react.js. Use APIs to connect with Firebase in the database.
* We will use the cloud server of Firebase.

# Implementation View

This is the expected file structure of the application, which can be further modified to be more appropriate, each updated change will be saved in this document.

¦ package-lock.json

¦ package.json

¦ README.md

¦ tree.txt

+---node\_modules

¦//contains a few packages with modules for npm install to run the web

+---public

¦ favicon.png

¦ index.html

+---src

¦ App.css

¦ App.js

¦ index.js

+---components

¦ Footer.js

+---css

¦ ChatBubble.css

¦ CommentBubble.css

¦ Footer.css

¦ gg-button-o.css

¦ ManageSite.css

¦ MusicSite.css

¦ PlaylistCard.css

¦ SongCard.css

¦ TodoSite.css

+---Item

¦ Chatbox.js

¦ Footer.js

+---Manage

¦ Account.js

¦ Manage.js

¦ ManageMain.js

¦ ManageSite.js

¦ Song.js

+---Music

¦ MusicMain.js

¦ MusicNav.js

¦ MusicPlayer.js

¦ MusicPlaylist.js

¦ MusicSearch.js

¦ MusicSite.js

+---Todo

¦ TodoMain.js

¦ TodoSite.js

¦

+---util

Utilities.js