



Politechnika Wrocławska

Wydział Informatyki i Zarządzania

kierunek studiów: Informatyka

Praca dyplomowa - inżynierska

Karaoke application for smartwatch

Krzysztof Desput

słowa kluczowe:

Smartwatch

Karaoke

Android Wear

krótkie streszczenie:

The goal of thesis is project and implementation of karaoke application for smartwatch. Project is based on analysis of existing solutions and introduction to technologies used to develop the application. The implementation part contains the description of problems solved during the creation and possible ways of future development.

opiekun pracy	dr inż. Marek Kopel
dyplomowej	<i>Tytuł/stopień naukowy/imię i nazwisko</i>	<i>ocena</i>	<i>podpis</i>

Do celów archiwalnych pracę dyplomową zakwalifikowano do: *

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

* niepotrzebne skreślić

pieczęćka wydziałowa

Wrocław 2018

To the people I met during my education

Abstracts

Wersja polska

Celem niniejszej pracy jest projekt oraz implementacja aplikacji mobilnej do wspomagania imprez typu karaoke. Aplikacja składa się z dwóch części - klienta działającego na platformie Android Wear oraz serwera napisanego w Node.js. Serwer zarządza bazami danych Elasticsearch i Redis oraz zwraca informacje na temat piosenek. Aplikacja mobilna ma za zadanie zebrać informacje używane do wyszukiwania piosenki oraz wyświetlić jej tekst użytkownikowi w przystępnej formie. Część serwerowa zawiera panel administratora, który może być używany do zbierania informacji na temat działania aplikacji.

W pracy zostaną przedstawione konkurencyjne rozwiązania oraz ich wady i zalety. Opisane zostaną technologie użyte do zrealizowania aplikacji oraz projekt, na którym bazowano się podczas tworzenia programu. W części implementacyjnej zaprezentowane zostaną technologie użyte podczas tworzenia aplikacji oraz problemy napotkane podczas jej realizacji. Na koniec zostaną przedstawione możliwe usprawnienia i kierunki rozwoju na przyszłość.

English version

The goal of this thesis is to create project and implementation of mobile application to support karaoke parties. The application consists of 2 parts - a client that works on Android Wear devices and a server, which is written in Node.js. The server manages Elasticsearch and Redis databases and returns the data about songs. The mobile application's goal is to collect the data used to search the song and to display its text to the user in the accessible form. The server part contains an admin panel, which can be used to collect the data about the application activity.

In the thesis, competitive solutions will be presented along with their advantages and disadvantages. Technologies used to develop the application and the project on which the application is based will be described. In the implementation part, the technologies used in the process of creating an application will be presented as well as problems met during the development. At the end, possible improvements and directions of development will be depicted.

Table of contents

1	Introduction	5
1.1.	Introduction to the topic	5
1.2.	Overview of existing solutions	7
1.3.	Goal of the thesis	10
2	Project.....	12
2.1.	Functional and non-functional requirements	12
2.2.	Use case diagram	12
2.3.	Use case scenarios	13
2.4.	Architecture schema	16
2.5.	Class diagram	17
2.6.	User interface prototype	21
3	Format of imported files with lyrics.....	26
4	Implementation.....	27
4.1.	Technologies used to develop the solution.....	27
4.2.	Libraries, classes and interfaces used during the development	28
4.3.	Design patterns implemented in the application.....	29
4.4.	Problems met during the implementation.....	31
5	User interface of the complete application.....	34
6	Application tests and verification.....	37
7	Possible ways of future development.....	40
8	Summary	41
9	References	42
10	List of figures	44

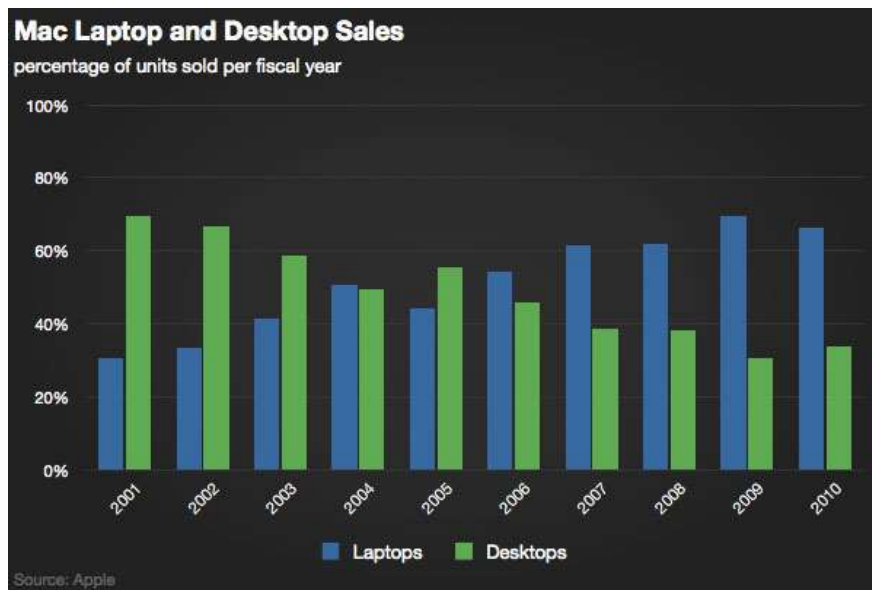


Figure 1.2 Mac laptop and desktop sales [12]

Since the release of first iPhone made by Apple and devices with Android, a lot of people replaced their mobile phones with smartphones. These devices made it possible to be online without any breaks. They took the responsibilities of computers very quickly, and nowadays most of popular models are faster than high-level computers manufactured 10-15 years ago. To allow people to be connected to latest messages without having to make any simple gesture like taking the smartphone out of their pockets, the engineers invented the latest fashionable device - smartwatches. Smartwatches are computers closed in a watch housing. They are equipped with wireless controllers that let them connect to the Internet with or without the usage of smartphone – they communicate via WiFi, Bluetooth or NFC. The global sales of smartwatches have been increasing for the last few years [Figure 1.3].

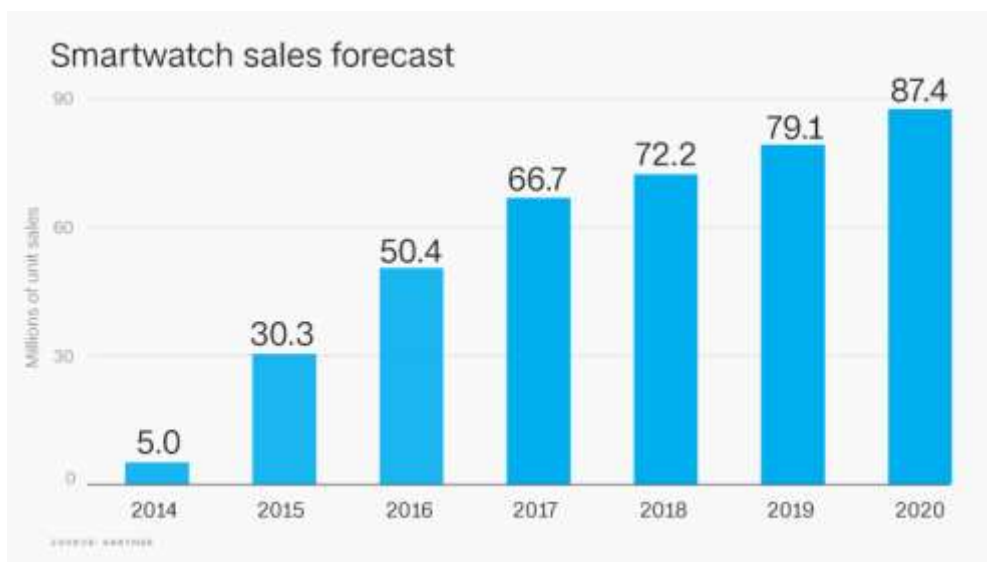


Figure 1.3 Smartwatch sales forecast [28]

What is a bad news for the users of wearables is that not all the developers took the new market seriously. Still there are not many applications available for smartwatches. However, it creates an opportunity for beginning programmers who want to create his own applications for mobile devices. As a developer who want to publish his application to a market like Google Play or Apple AppStore it is hard to get the satisfactory amount of downloads. It can be worth

to target the new type of devices and use them to overtake the competition and acquire a large share in the application market.

One of my hobbies is taking part in karaoke parties. Karaoke is a form of party that comes from Japan. During this party, the music is played, usually without the vocal part and the lyrics are displayed to the singers. It might be very interesting to create karaoke application for wearables. This will allow me to develop my first software for wearables, additionally in an interesting topic.

1.2. Overview of existing solutions

During the market overview several applications have been installed on smartwatch and smartphone to check their advantages and disadvantages. The most interesting of them are listed below:

1.2.1. Musixmatch

Probably the most popular software used for karaoke on mobile devices is Musixmatch. It offers more than 14 million lyrics in over 50 languages and the application advertises as the “the world’s largest lyrics catalog” [16]. The application provides the ability to recognize the music using the microphone and to show lyrics for songs played by other music players [Figure 1.4]. This feature is called FloatingLyrics and is really useful function for karaoke fans. Unfortunately, Musixmatch cannot recognize songs by lyrics sung by the user. The biggest disadvantage of Musixmatch is that it ended support for Android Wear devices. It was possible to run the application on Android Wear in the past, but now it is not available in Google Play Store for wearables. Therefore, it leaves a huge niche in the market.

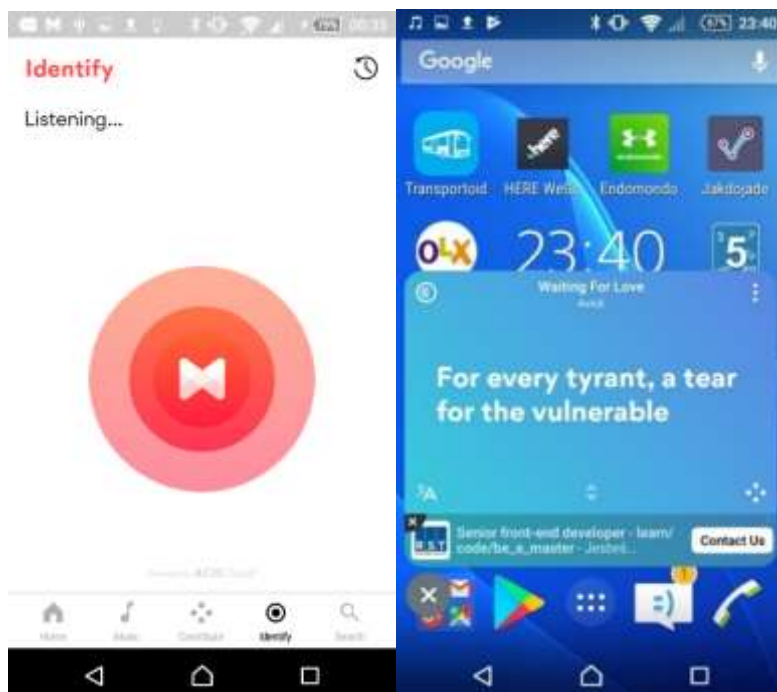


Figure 1.4 Musixmatch application

1.2.2. SoundHound

This application can recognize songs from humming or singing without any complications. There are also lyrics provided for the identified song and the ability to buy the

tune from several music shops [Figure 1.5]. User can see the history of the melodies detected [22]. When it comes to lyrics, they are served for the tracks and are dynamically changing as the music goes. The only weakness of this application is that it does not offer a client for wearables.



Figure 1.5 SoundHound application

1.2.3. Shazam

Shazam is the world's most popular software when it comes to music recognition. It was downloaded more than 100 million times [21]. It offers the ability to show track information, to buy it on different platforms and to watch the video for the song. Regarding the lyrics, it offers a database from Musixmatch, but the lyrics are not shown dynamically - the user has to swipe the screen to keep them to date with the music [Figure 1.6].



Figure 1.6 Shazam for Android smartphones

Shazam offers a client for smartwatches, but it has very poor functionality. It just provides the ability to recognize the song and it shows its artist and title. Lyrics are unavailable in wearable version [Figure 1.7].

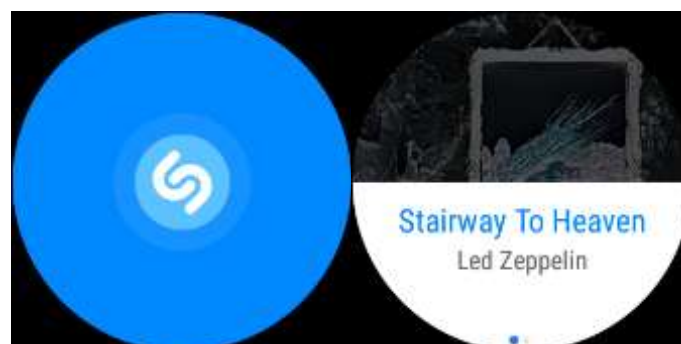


Figure 1.7 Shazam for Android Wear smartwatches

1.2.4. LyricsMania

This application has got Android Wear client that can be used to recognize the songs and to show their lyrics [14]. The fact is that the song recognition is bugged, and it has not recognized any songs during the tests. The only function that worked was showing the lyrics for songs played in LyricsMania for smartphones. Unfortunately, the lyrics have to be moved manually by the user. This application was installed more than 1 million times, so it gives a hope that the project created in this thesis will become much more popular.

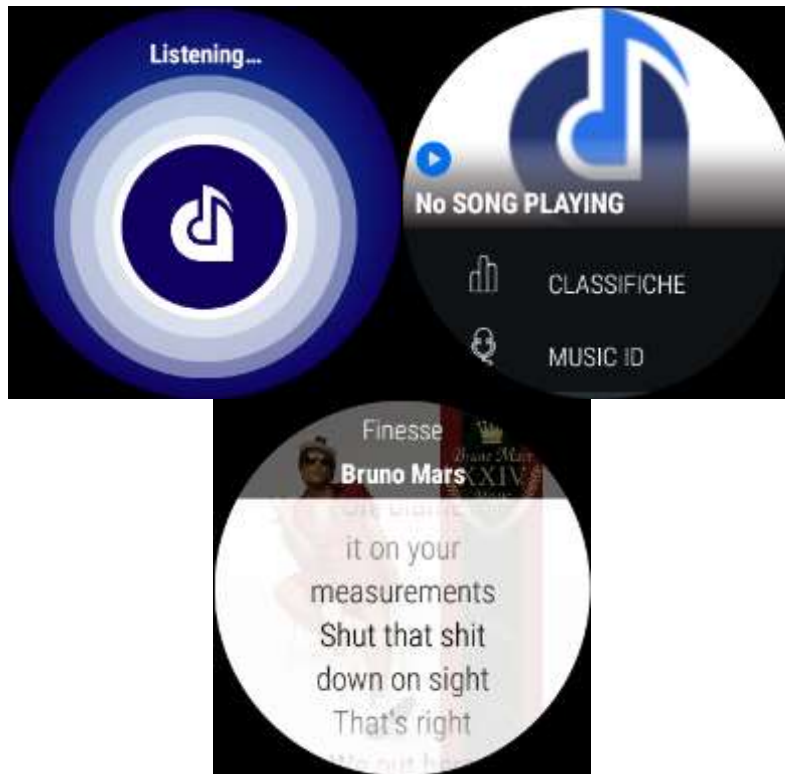


Figure 1.8 LyricsMania application

Summarizing, there are many great applications that can be used for karaoke, but the offer for wearables is not satisfactory. Any new application, designed mostly for wearables can easily get a lot of users and begin expansion on other platforms like smartphones or tablets.

1.3. Goal of the thesis

The goal of the thesis is to create project and implementation of application that will show the song's lyrics during the karaoke parties. The system will be based on client-server model. Client will be the application for smartwatch – it will transform the user's speech into text using Google speech recognition and then send it to the server. Server will create a response with found songs and their lyrics. Client application will present the lyrics on the screen in the way that it will be possible to sing the song with the correct tempo. Together with the client application, the admin panel accessible by the browser will be developed. It will allow the application owner to view the statistics of the application that can be used to improve the song database in the future. The schema of the application flow is shown in Figure 1.9.

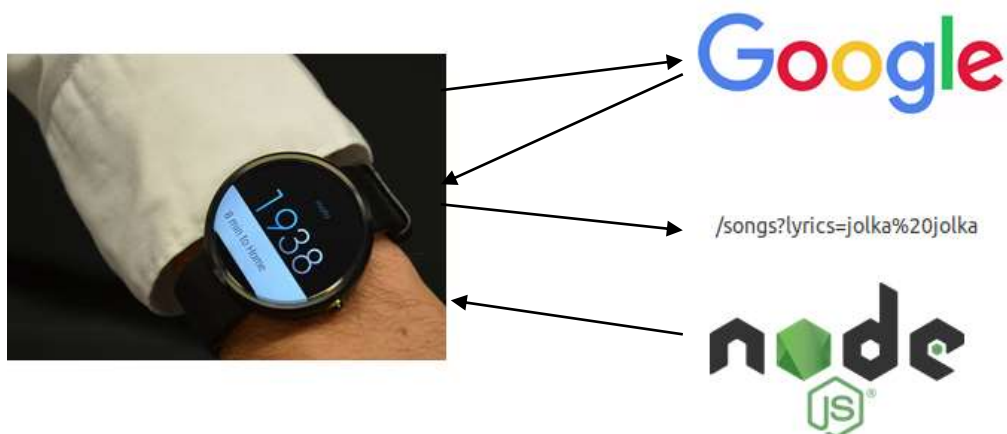


Figure 1.9 Schema of the application flow

To develop the solution following devices and software will be used:

1. Computer Asus X554L

Operating system: Linux Ubuntu 16.04 Xenial Xerus

Development environment: Google Android Studio 3.0.0

Source code editor: Visual Studio Code 1.18

2. Smartwatch LG G Watch R

Operating system: Google Android Wear 2.6

3. Virtual SSD Cloud Server hosted by Scaleway in Amsterdam

Operating system: Linux Ubuntu 16.04 Xenial Xerus

The thesis consists of 10 chapters. In the first chapter, the introduction to the topic of the thesis is presented with the analysis of existing solutions. The second chapter is the project part. It contains functional and non-functional requirements of developed application together with all the use cases covered by it. Additionally, the class diagrams of both client and server application are shown. This phase includes also prototypes of user interface. In the next chapter, the format of lyrics files used by the application is unveiled. The fourth chapter is the implementation part with description of technologies used during the development. In this section, also the most interesting problems met during the production are depicted. In the succeeding chapter, user interface of complete application is presented. The sixth part contains description of the tests executed on the product and the issues met during the usage. In the possible ways of future development, the improvements that can be developed in the future are described. The thesis ends with the summary and the list of references and figures.

2 Project

2.1. Functional and non-functional requirements

2.1.1. Client application requirements

Functional requirements:

- It should transform user's speech to text
- It should send recognized text to server
- It should obtain response with songs from the server
- It should show the list of found songs when there are multiple songs found by the server
- It should show song's lyrics in a proper time
- It should provide the ability to report incorrectly recognized songs

Non-functional requirements:

- It should be able to work on Android Wear 2.0 and higher
- It should be a stand-alone application (able to work without equivalent application installed on smartphone)
- It should have an intuitive and simple graphic interface
- It should be able to work on both rounded and square smartwatches
- It should be able to work in Ambient Mode (remaining in the foreground while saving the battery)

2.1.2. Server application requirements

Functional requirements:

- It should respond with lyrics for searched songs
- It should require password to view the admin panel
- It should show the list of lyrics for which the song was not found
- It should show the lyrics for which the song was wrongly recognized
- It should show the list of recognized songs
- It should give the ability to log off
- It should import songs from .txt files

Non-functional requirements:

- It should be runnable on computer with Linux system and 64 bit CPU
- It should be easy to scale in the future
- It should respond in less than 300ms
- It should provide a responsive admin panel

2.2. Use case diagram

There have been identified 3 actors using the system:

- Administrator – actor, who is able to connect to remote server with his public SSH key and knows the password to the admin panel
- Guest – actor, who has entered the admin panel website
- Smartwatch application User – actor, who has an application installed on his smartwatch

The use cases of interacting with the application are shown in Figure 2.1:

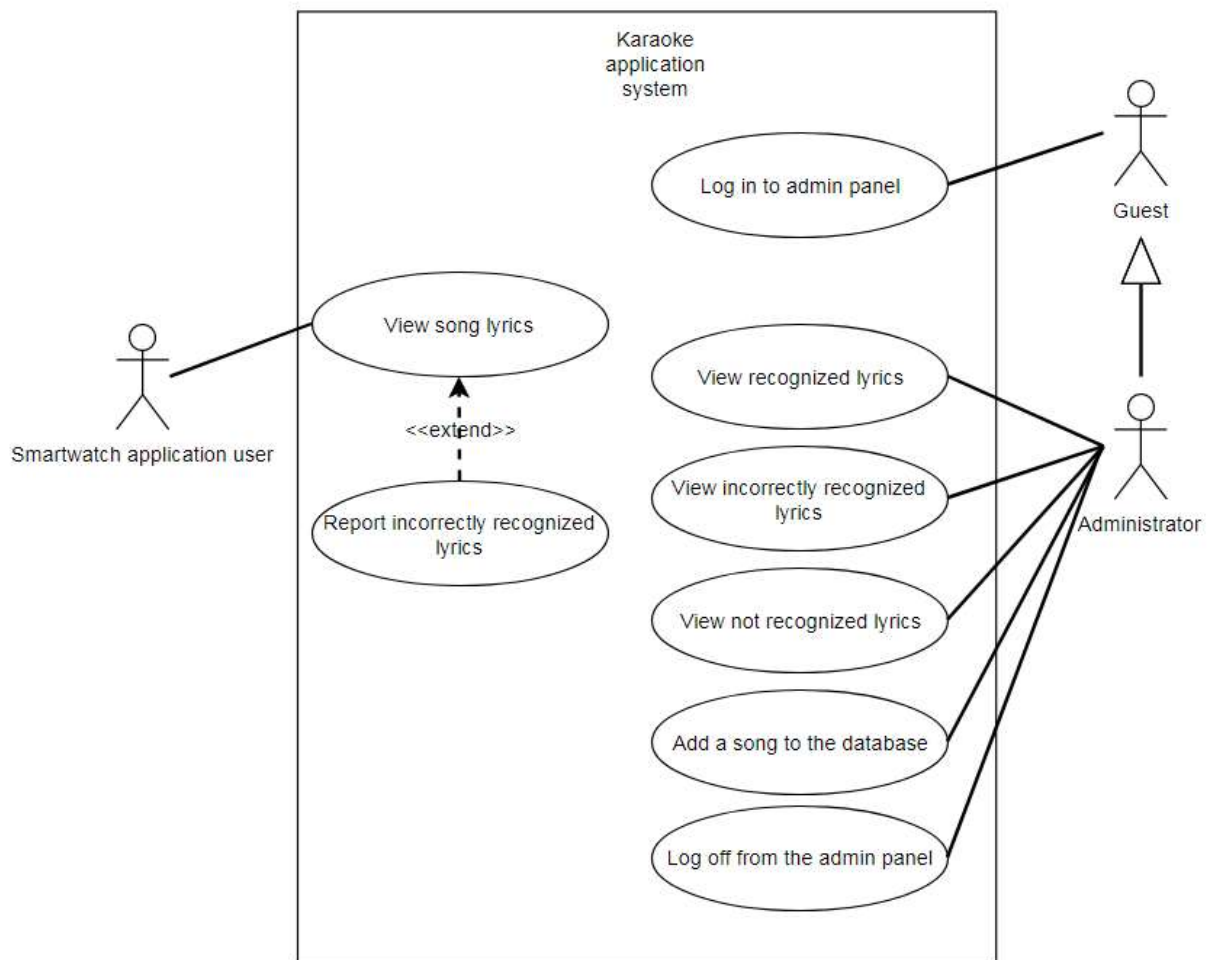


Figure 2.1 Use case diagram

2.3. Use case scenarios

Use case: View song lyrics

Description: It is the most important use case for the application. It allows the smartwatch application user to see the lyrics of the song.

Actor: Smartwatch application user

Preconditions: Actor has the application opened on his smartwatch.

Success scenario:

1. Actor taps the button to start listening.
2. System displays an icon encouraging to start singing with the text informing about the listening started.
3. Actor sings a piece of the song and stops singing.
4. System shows information about looking for the song in the server's database, after finding a song, the actor is moved to the activity with lyrics.

Alternative path 1:

The application does not have proper permissions and user grants them.

- 2a. System displays a question about giving it a proper permissions.
3. Actor taps the button to grant permissions to the application.
4. System displays an icon encouraging to start singing with the text informing about the listening started.
5. Actor sings a piece of the song and stops singing.

6. System shows information about looking for the song in the server's database, after finding a song, the actor is moved to the activity with lyrics.

Alternative path 2:

The application does not have proper permissions and user does not grant them.

- 2a. System displays a question about giving it a proper permissions.
3. Actor taps the button to not grant permissions to the application.
4. System displays an error.

Alternative path 3:

The speech cannot be transformed to text.

- 4a. System cannot transform speech to a text, error message is displayed.

Alternative path 4:

The application cannot connect to the server.

- 4a. System displays an information about the connection problem.

Alternative path 5:

No lyrics have been found

- 4a. System shows information about looking for the song in the server's database, after a while displays an information about no songs found.

Alternative path 6:

More than one song matches the searched lyrics, user selects the one he wants

- 4a. System shows information about looking for the song in the server's database, after a while displays a question with artists and titles of found songs.
5. Users taps the song for which he wanted to see the lyrics.
6. System displays the activity with lyrics for selected song.

Alternative path 7:

More than one song matches the searched lyrics, user does not select any.

- 4a. System shows information about looking for the song in the server's database, after a while displays a question with artists and titles of found songs.
5. Users does not tap any song and swipes right to go back to speech recognition.
6. System displays the start activity.

Use case: Report incorrectly recognized lyrics

Description: Sometimes user may want to report incorrectly recognized lyrics. This will allow the administrator to see the proper information in the admin panel and fix it in the future.

Actor: Smartwatch application user

Preconditions: Actor has the application opened on his smartwatch, and he is on the activity with the lyrics displayed, but has entered it less than 5 seconds before starting the use case scenario.

Success scenario:

1. Actor swipes right on his smartwatch in order to go back to the activity to detect the lyrics.
2. System displays the question about the correctness of the last song detected.
3. Actor taps the button with the answer that song was incorrect.
4. System displays the thank you message and displays the activity for detecting the songs.

Alternative path: Actor admits that the song was correct

- 3a. Actor taps the button with the answer that song was correct.
4. System displays the thank you message and shows the activity for detecting the songs.

Postconditions: Information about incorrect lyrics is sent to the server

Use case: Log in to admin panel

Description: Authorization in order to gain the access to the admin panel.

Actor: Guest

Preconditions: Actor is logged off from the admin panel.

Success scenario:

1. Actor enters the address of admin panel by his web browser.
2. System displays the page with a form to enter the password.
3. Actor inputs the password and clicks the button with “Login” text.
4. System displays the admin panel.

Use case diagram: Password is not correct

- 4a. System displays the information about wrong password on the page with form to enter the password.

Postconditions: Actor is logged in to the admin panel.

Use case: View recognized lyrics

Description: To analyze the preferences of the users, admin would like to see the list of recognized lyrics.

Actor: Administrator

Preconditions: Actor is logged in to the admin panel.

Success scenario:

1. Actor selects “Recognized lyrics” from the menu.
2. System displays the page with recognized lyrics.

Use case: View incorrectly recognized lyrics

Description: It shows the list of incorrectly recognized lyrics reported by the users.

Actor: Administrator

Preconditions: Actor is logged in to the admin panel.

Success scenario:

1. Actor selects “Incorrectly recognized lyrics” from the menu.
2. System displays the page with incorrectly recognized lyrics.

Postconditions:

Use case: View not recognized lyrics

Description: In order to improve the application in the future, the list of lyrics sent by users, for which no songs were found will be shown to the administrator.

Actor: Administrator

Preconditions: Actor is logged in to the admin panel.

Success scenario:

1. Actor selects “Not recognized lyrics” from the menu.
2. System displays the page with not recognized lyrics.

Postconditions:

Use case: Add a song to the database

Description: In this use case, the files with lyrics will be parsed by the application and added to the database.

Actor: Administrator

Preconditions: Actor has opened a shell and moved text files with lyrics to the proper folder.

Success scenario:

1. Actor enters the server shell via SSH and runs the proper script to import songs from files.
2. The information about success is displayed together with the command prompt.

Postconditions: The song data is stored in the database of the system

Use case: Log off from the admin panel

Description: When administrator does not want to stay logged in to the admin panel, he can just log off.

Actor: Administrator

Preconditions: Actor is logged in to the admin panel.

Success scenario:

1. Actor selects “Log off” from the menu.
2. System moves the actor to the page for logging in and unauthorized the actor’s cookie.
The success information is displayed.

Postconditions: Actor is logged off from the admin panel.

2.4. Architecture schema

The project will be based on project-server architecture. To communicate with the server, the wearable application will use HTTP requests. Therefore, the API available for the application should be planned before the implementation. What is more, there will be 3 services working on the server machine, but they can be divided in the future as the traffic will grow. The architecture for the beginning will look as in Figure 2.2.

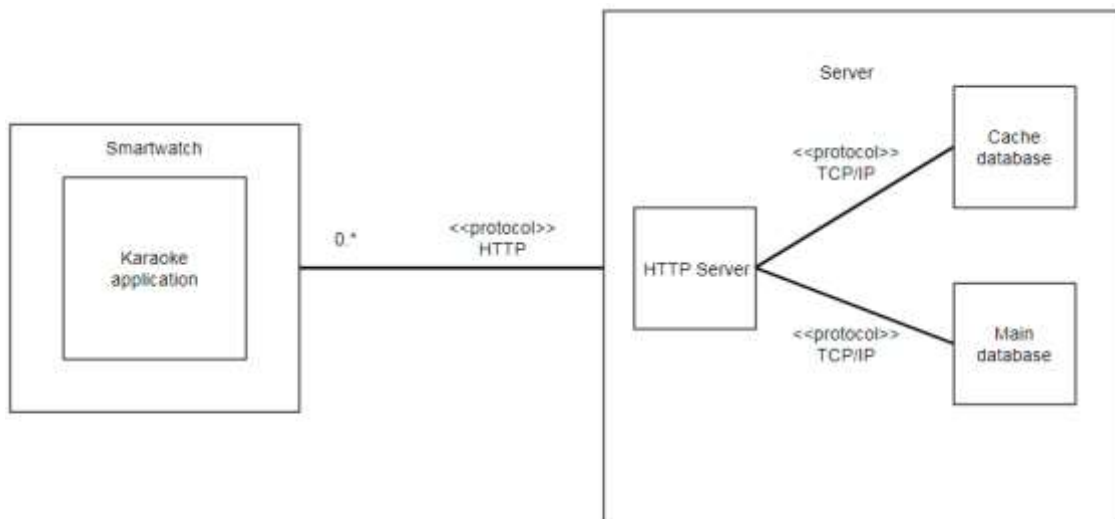


Figure 2.2 Architecture schema

The server will be able to handle requests from multiple clients and it will have only port 80 opened, the ports used internally between HTTP server module and databases will be blocked for external traffic. In order to isolate the components on server machine, all of them will be running as separate Docker containers. This will also make it easy to move them to separate instances in the future. The containers will communicate by TCP/IP protocol.

Regarding the API available for the karaoke application, here is a list of designed endpoints and formats of their responses:

- Endpoint with song lyrics

Request:

Method: GET

Address: `http://server_address/songs?lyrics=searchedlyrics`

Parameters:

- lyrics: Lyrics of the fragment song by the user, for which the song is being searched

Responses:

- 200:

The response is the information about song found by the server. It is returned in JSON format:

```
{
  "title":title_of_the_song,
  "artist":artist_of_the_song,
  "creator":creator_of_lyrics,
  "genre":genre_of_the_song,
  "language":language_of_the_song,
  "bpm":bpm_of_the_song,
  "firstSyllablePosition":position_of_the_first_searched_syllable,
  "lyrics":[
    {
      "note":type_of_syllable,
      "position":position_of_the_syllable,
      "length":length_of_the_syllable,
      "pitch":bpm_of_the_song,
      "text":text_of_the_syllable
    }
  ]
}
```

When there are multiple songs found by the server, they are returned as an array of objects.

- 204:

The response 204(no content) means that there were no songs found for searched text.

- Endpoint for reporting incorrect lyrics

Request:

Method: POST

Address:

http://server_address/incorrect_lyrics?lyrics=searchedlyrics&artist=foundArtist&title=foundTitle

Parameters:

- lyrics: Lyrics of the fragment song by the user, for which the song was found
- artist: Artist of the found song
- title: Title of the found song

Response:

- 200:

The response code 200 means that the information about incorrect lyrics has been received by the server.

2.5. Class diagram

To develop the solution faster, the classes and relations between them have been planned before the implementation. However, a few changes to the diagrams occurred during the implementation. Diagrams were drawn in Draw.io application, using hints contained on [11]. To improve the clarity, only the most important fields and methods are shown.

The client application will have 2 classes for activities – *DetectSongActivity* and *DisplayTextActivity*. They will both extend the *WearableActivity* class – the base activity class that is designed for wearables. One of the most interesting thing is the usage of singleton – the implementation is called *AppSingleton*. The main purpose of this pattern is to make sure that there is only one object of selected class. To display only one line at once, the song will consist of lines created by the private *generateLyrics* method in *Song* class. Class diagram of the client application is presented in Figure 2.3.

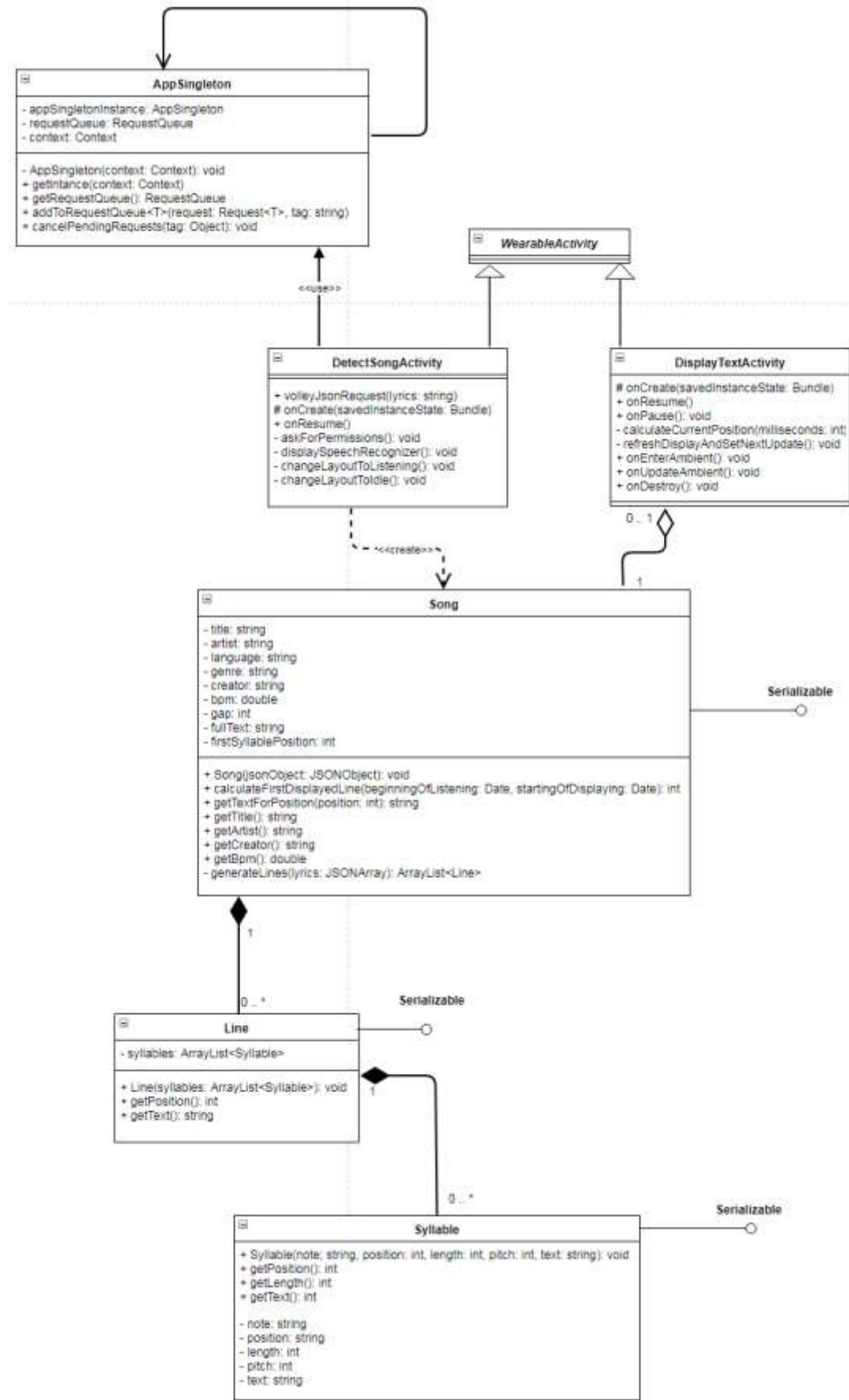


Figure 2.3 Class diagram of the client application

As on the Figure 2.4, the server application is designed to use the Dependency Injection pattern. It is a common pattern in web applications, that makes it easy to write tests for developed software. All the objects are created by the Container object and from this moment they are called services. They can be obtained only from the container and they can be replaced when it is needed.

Together with other objects, the container creates also an instance of Router class. It's main role is to provide API (its role is to be a controller in MVC pattern). To handle API requests, the Router object uses services obtained from Container object.

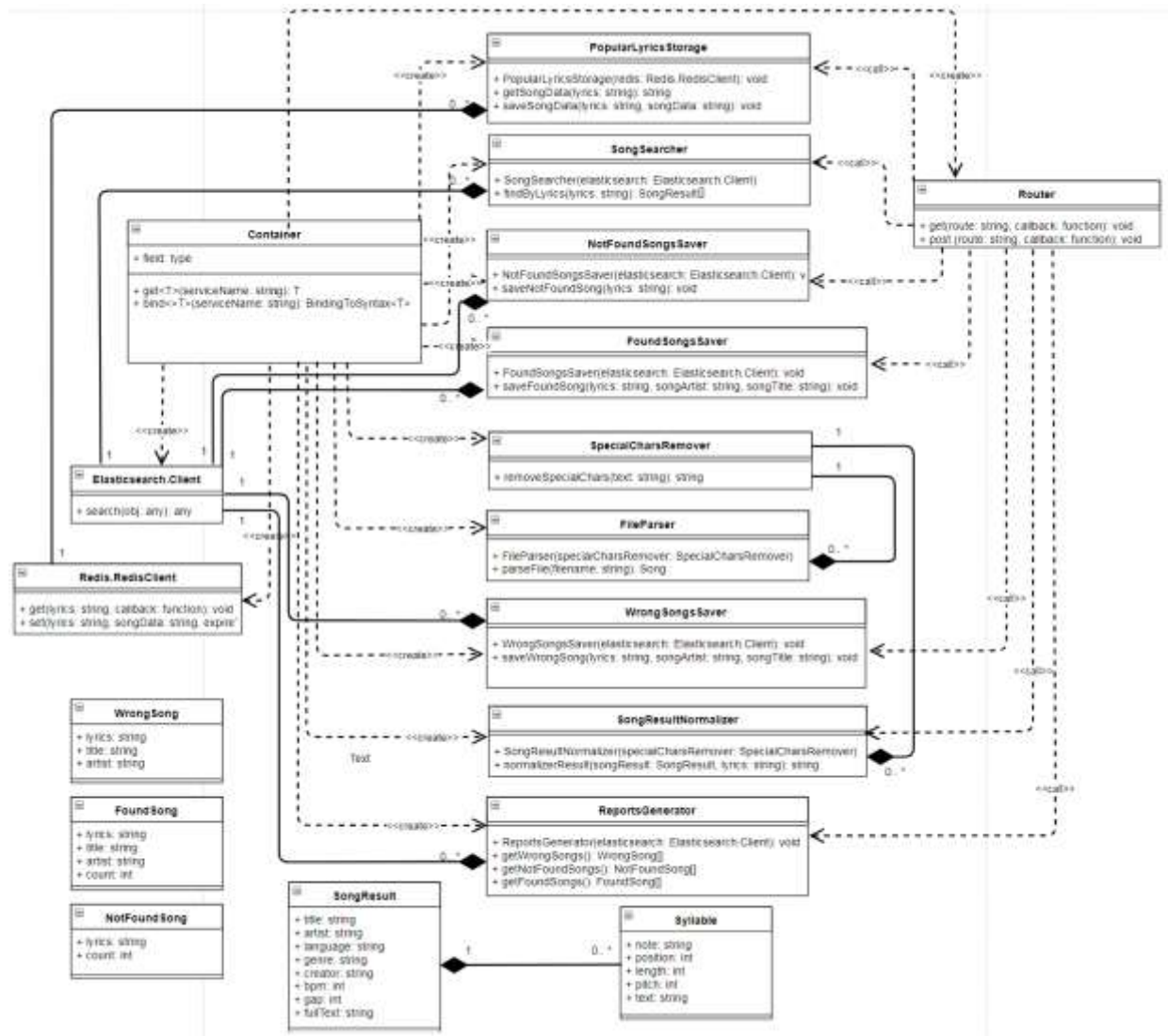


Figure 2.4 Class diagram of the server application

2.6. User interface prototype

2.6.1. Client application for smartwatch

For the client application, all the mocks were created using Balsamiq Mockups application [4]. The basic elements that create Android applications are activities – activity is a simple screen that is being displayed. The client application will consist of 2 activities, but there will be some modifications occurring during the activities lifetime that will make the flow more complicated. All the layout text will be displayed in Polish, because the application will be released on Polish market at the beginning.

When user enters a wearable application, he should see a big circle with the microphone icon [Figure 2.5].

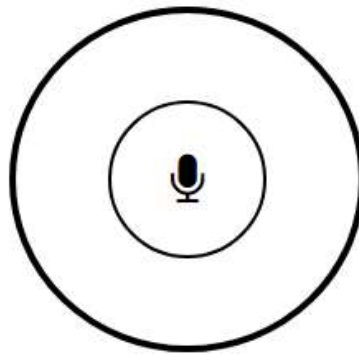


Figure 2.5 Prototype of activity for detecting song before starting listening

After the icon is tapped, the application starts listening and the layout changes. There will be an animation with the text encouraging the user to start singing [Figure 2.6].

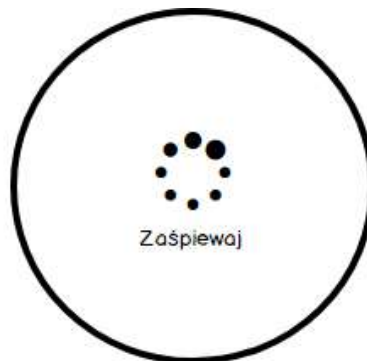


Figure 2.6 Prototype of activity for detecting song during the listening

After the listening is finished, the application sends the request to the server. While waiting for the response, the proper message will be displayed to not discourage the user when he is waiting [Figure 2.7].



Figure 2.7 Prototype of the message displayed while waiting for a response

When the response is received from the server, it is possible to get the information with more than one song that matches the lyrics recorded by the user, in this case the list with titles of the songs will be displayed. User selects the song by tapping the title he wanted to sing. When he selects the song, or there is only one song found for lyrics used for searching, the user is moved to activity with the lyrics [Figure 2.8].



Figure 2.8 Prototype of the view after receiving the response from the server

Sometimes, the user decides that the found song is not the song he wanted. In this case, when he goes back to the previous activity before the end of 5 seconds, the question will be displayed to ask whether the song is correct or not [Figure 2.9].



Figure 2.9 Prototype of reporting incorrect song recognition

2.6.2. Admin panel

The second part of the application is the admin panel. It should be available from the browser. All the mockups regarding the panel were created using the application called Moqups

[23]. When a guests enters a website for logging in, he should see the form to input the password [Figure 2.10].

A screenshot of a web browser window titled "Mozilla". The address bar shows "http://smartkaraoke.com/admin". The main content area displays the "SmartKaraoke" logo at the top. Below the logo is a centered box containing the text "Enter the password to see the admin panel". Inside this box is a text input field labeled "Password" and a "Login" button below it.

Figure 2.10 Prototype of the form to log in to the admin panel

When the entered password is not correct, the proper message will be displayed saying that the input was incorrect [Figure 2.11].

A screenshot of a web browser window titled "Mozilla". The address bar shows "http://smartkaraoke.com/admin". A red error message banner at the top of the content area reads "The entered password is not correct:". Below the banner is the "SmartKaraoke" logo. Underneath the logo is a centered box with the text "Enter the password to see the admin panel", a "Password" input field, and a "Login" button.

Figure 2.11 Prototype of the view with message about incorrect password

After a successful authorization, the user will be moved to the admin panel. It has a menu on the top with the tabs to change displayed data. The default tab after logging in is a tab with "Recognized". On this page, the table is rendered with the information of all recognized

songs and the lyrics used to find them. They are grouped and counted to make it easy to check the popularity of the songs [Figure 2.12].



▼ Lyrics	▼ Artist	▼ Song	▼ Count
I'm saying that	The Cranberries	Zombie	33
Ok let's go	Big Shaq	Man's not hot	20
Do you hear me	Adele	Hello	10
Like fire	Alice Cooper	Poison	1

Figure 2.12 Prototype of the page with recognized songs

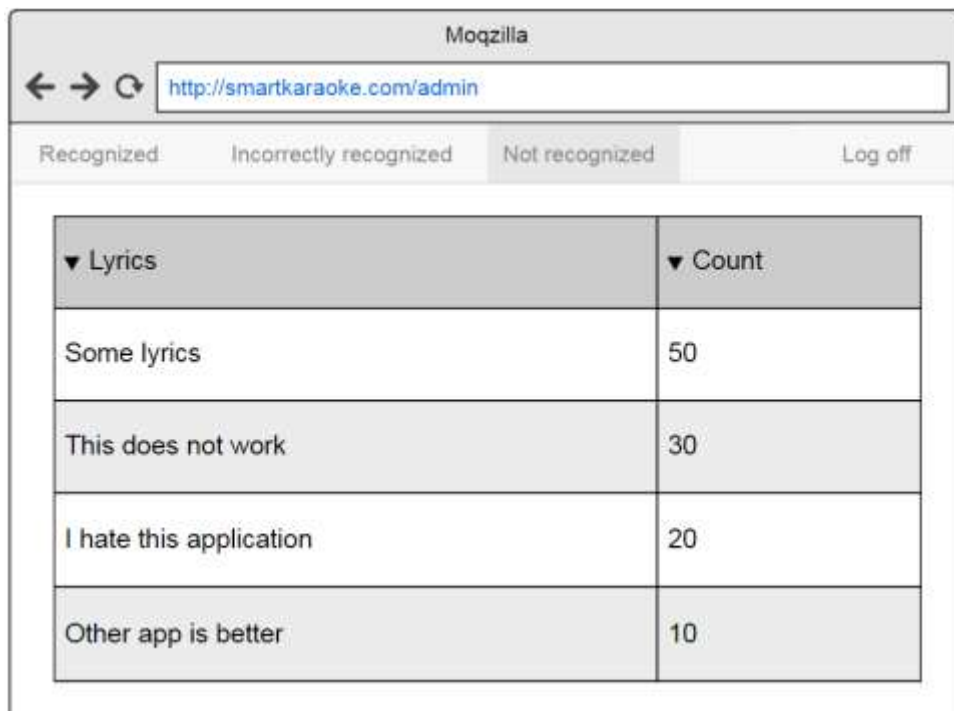
When user selects, the tab with “Incorrectly recognized” lyrics, he is moved to the page with another table. It is filled with lyrics reported by users to be matched to wrong songs. The lyrics are not grouped, they are just displayed in chronological order [Figure 2.13].



▼ Lyrics	▼ Artist	▼ Song
I'm saying that	The Cranberries	Zombie
Ok let's go	Big Shaq	Man's not hot
Do you hear me	Adele	Hello
Like fire	Alice Cooper	Poison

Figure 2.13 Prototype of the page with incorrectly recognized songs

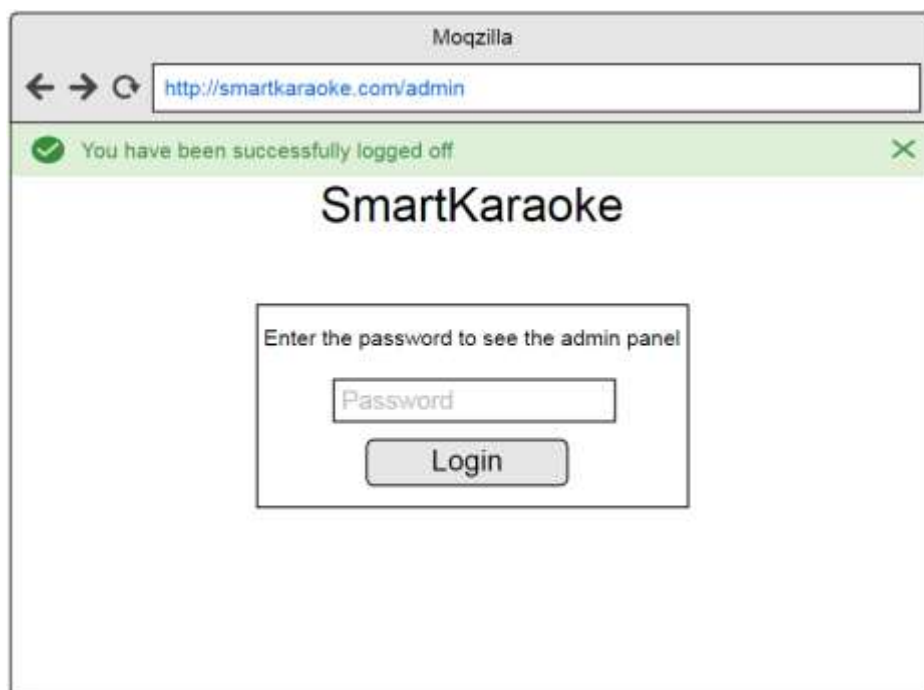
After selecting “Not recognized” tab, the lyrics with no matching song are presented to the administrator [Figure 2.14]. They are ordered by the number of occurrences, so it is easy to check which new songs will be desirable by the users.



▼ Lyrics	▼ Count
Some lyrics	50
This does not work	30
I hate this application	20
Other app is better	10

Figure 2.14 Prototype of the page with lyrics that have not been recognized

On the top right corner, there is a button with the text “Log off”. After clicking on it, the user is moved to the login page with the information about logging off from the admin panel [Figure 2.15].



Moqzilla

← → ↻ http://smartkaraoke.com/admin

Recognized Incorrectly recognized **Not recognized** Log off

✓ You have been successfully logged off ✕

SmartKaraoke

Enter the password to see the admin panel

Password

Login

Figure 2.15 Prototype of the page visible after logging off the admin panel

3 Format of imported files with lyrics

For the purpose of the developed application, the existing file format for lyrics will be used. The format is already used in Ultrastar – a popular karaoke application for Windows. The application has a wide community that creates text files with lyrics for many trendy tunes. The text files are well documented, so they can be used to fill the database of developed application. However, it is important to get all the required permissions from the text owners and the people who created the files if we want to share our application outside.

The sample text file in Ultrastar format is presented below:

```
#TITLE: Title of the song
#ARTIST: Artist behind the song
#MP3: file.mp3
#GAP: 1000
#BPM: 120
: 0 2 12 Tee
: 2 2 12 nage
: 6 6 12 dreams
- 12
: 12 2 9 in
: 14 2 7 a
: 16 3 12 tee
: 20 3 12 nage
: 24 4 16 cir
: 28 3 14 cus
- 32
```

Excluding a few lines with # at the beginning, each line of the file represents one note or line break. The first column can have one of 4 values [24]:

- : – meaning that the note is regular note
- * – meaning that the note is golden note, it gives more points in Ultrastar application when user sings this note properly
- F – meaning that the syllable is just a freestyle syllable
- - – meaning that this is a line break

The second column represents when the syllable appears in the song, the metric is one beat. The next column states the length of the note with beats as a metric. In the fourth column, the code for the pitch is stored. It is used to grade the user's vocal skills in Ultrastar application. The last column includes the text to be sung at that note.

4 Implementation

4.1. Technologies used to develop the solution

4.1.1. Java

Java is the basic language when it comes to Android software development. Most of tutorials and books about Android concern it. It is also possible to write native Android applications in C++, but Java code is more pleasant to write in author's opinion. It is also achievable to develop Android Wear applications in C# - there is a platform called Xamarin [3] which offers writing applications for Android, iOS and Windows Phone. It would be an interesting option, but after research it was found that there would be several problems when implementing speech recognition with Xamarin. Therefore, Java was chosen as the language for wearable client.

4.1.2. Node.js

When choosing the technology for the backend of the application, one of the most important factors taken into consideration was the speed. Node.js uses a non-blocking input/output model that gives it really powerful acceleration [17, 18]. It is built on JavaScript engine made by Google. It has a HTTP server, so it does not need additional software like NGINX or Apache. There are some a little bit faster alternatives like Go and Elixir, but they would require learning new language. JavaScript and its modifications are simple and comfortable to use.

4.1.3. TypeScript

One of the biggest drawbacks of using JavaScript in huge projects is that it does not check types. With the increasing popularity of JavaScript libraries and frameworks, Microsoft developed the language called TypeScript. It can be compiled to JavaScript or just be executed with Node using a package called ts-node.

4.1.4. EJS

When developing the admin panel, it is required to serve the views to the user. The panel is really small component, so it was created using EJS (Embedded JavaScript templates). EJS allows the usage of data passed to the view in HTML templates. It can be replaced in the future by a really separated front-end module. This can be reached with Vue.js or React as Angular is too massive for the small components like the admin panel.

4.1.5. Elasticsearch

During the development of the application, it was not sure how the model will look like. There were many concepts about the structure of the data. Furthermore, the important factor while selecting the database was to select the one that offers quick searches as this will be its most common usage. After the research of existing databases, the engine called Elasticsearch [10] was chosen. It is a non-relational database, designed especially for searches. The communication with Elasticsearch takes place by REST queries and the model changes dynamically as the new data is inserted.

4.1.6. Redis

To make the server responses even faster, the additional database for cache was added to the project. The most popular cache databases are Redis [19] and Memcached. Both are very fast, but Redis is said to be better when it comes to memory usage [29].

4.1.7. Docker

Docker is a tool that is recently gaining popularity across the servers around the world. Docker offers a feature called containerization [13]. The applications are ran in separate docker containers with separated memory and disk space. The separate containers can be easily moved to other servers as the load of the machine will rise. The number of services working on the server in this project (3 services) caused it interesting to try out Docker.

4.1.8. Docker-compose

Docker compose is a tool for defining and running Docker containers [8]. It can run the containers from the file created by the developer. In the file, the volumes attached to container can be defined together with commands ran during the container initialization.

4.1.9. Bootstrap

Bootstrap is a popular CSS framework developed by Twitter programmers [25]. It is used to make the admin panel responsive, so it will look elegant on devices with small screens like tablets and smartphones.

4.2. Libraries, classes and interfaces used during the development

Following libraries, classes and interfaces have been used during the application development:

4.2.1. Client application

- **android.speech.RecognitionListener** – The interface is used for receiving the notifications from SpeechRecognizer class [1].
- **android.speech.SpeechRecognizer** – It is the class that provides access to speech recognition service [2].
- **java.util.concurrent.TimeUnit** – It is an enumerator that represents the given unit of time. It is used to set the interval for ambient mode in Android Wear.
- **java.io.Serializable** – The interface that must be implemented by the class in order to serialize it's state.
- **java.util.ArrayList** – The class implementing List interface in Java. Used to store lines and syllables in songs.
- **java.util.Comparator** – Interface that has to be implemented in order to sort the object of the class. Sorting is used to have the lines of lyrics organized.
- **com.android.volley.Request** – Class used to create HTTP requests.
- **com.android.volley.RequestQueue** – A class that manages threads for running the network operations.
- **com.android.volley.toolbox.Volley** – Class used to create the RequestQueue
- **com.android.volley.toolbox.JSONObjectRequest** – A request used for obtaining a JSON response body. It is used to get JSON from the server.
- **org.json.JSONArray** – Class that represents array obtained from JSON

- **org.json.JSONObject** - Class that represents object obtained from JSON

4.2.2. Server application

- **body-parser** – This module parses the body of HTTP requests to make it able to use it when handling the requests.
- **elasticsearch** – It is a module used to communicate with Elasticsearch database.
- **express** – Express is a framework for handling HTTP requests.
- **iconv-lite** – It is used for converting character encodings. This module was needed to convert files with lyrics that were saved using different encodings.
- **inversify** – This module is needed for injecting the dependencies for Dependency Injection pattern.
- **moment** – It allows to manipulate and parse dates.
- **passport** – Passport is used for authentication.
- **redis** – This module is used to communicate with Redis database.
- **reflect-metadata** – This module was required by Inversify.
- **chai, mocha** – They are test frameworks used to execute tests on server application.
- **sinon** – Sinon provides mocks that can be used during the tests
- **supertest** – This library was used to test endpoints of the server
- **ts-node** – Execution environment for TypeScript files for node.js
- **typescript** – Installing this dependency allows the usage of TypeScript in node.js applications

4.3. Design patterns implemented in the application

During the development of the project, followed design patterns were used:

4.3.1. Singleton

Singleton is said by some people to be an anti-pattern [30], but it is an useful design pattern that guards the class. It prevents the system from creating more than one class instance. It was used in the client application – there can be only one request queue for HTTP requests and that is why this pattern is implemented.

4.3.2. Cache-aside pattern

This pattern was used in the server application to organize the flow when looking for the data in the databases. The basic action is based on trying to read from cache. When the value is not present, the database is queried, and then if the value was present in the database, it is stored in cache. The pattern has not been implemented for updating the data, because it will be very hard to find all the lyrics (keys) for which the new song will be better than the existing one. Instead of this, the expiration of keys in cache has been set to 24 hours. It means that after adding a new song, it will took 24 hours to update the cache data about it.

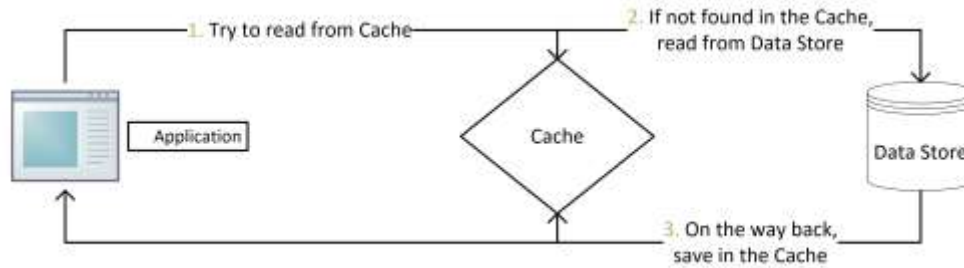


Figure 4.1 Reading from cache-aside pattern [7]

4.3.3. Dependency Injection and Builder

The main idea of the Dependency Injection pattern is to remove the dependencies between the components and to use them as plug-ins [15]. This makes it easy to swap elements and to use mocks for testing purposes. It was implemented in the server application with the usage of Inversify module and Container class. All the components are available from the container by using proper method with a component name as a parameter. Dependency Injection patterns requires also a Builder to be implemented. It creates the instances of injected classes [Figure 2.4].

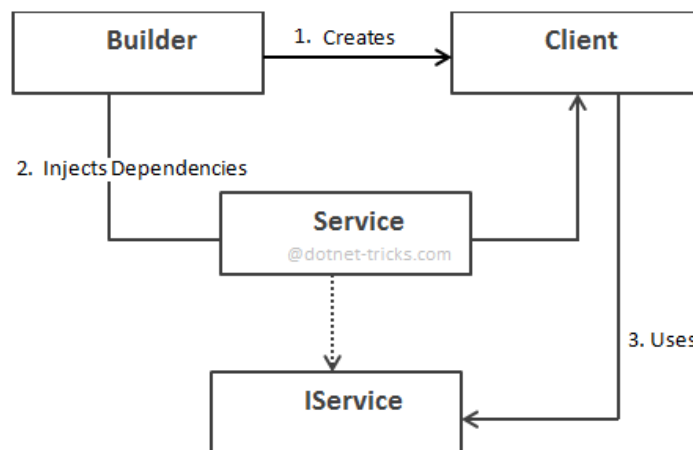


Figure 4.2 Schema of Dependency Injection pattern [6]

4.3.4. Model-View-Controller

To make the server application easy to maintain, the MVC pattern was implemented. MVC means Model-View-Controller. Model are classes responsible for the data, controller was implemented by the Router class which provides the API and views are EJS templates for the admin panel.

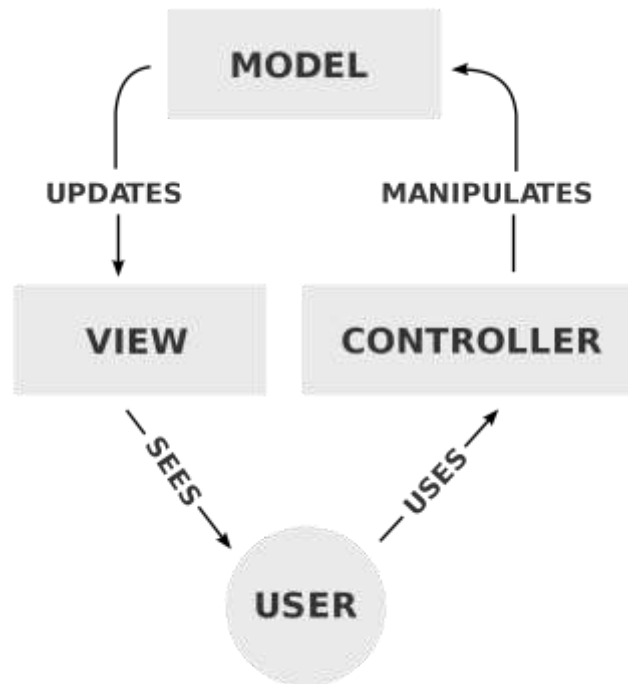


Figure 4.3 Schema of Model-View-Controller pattern [27]

4.4. Problems met during the implementation

4.4.1. Song object passed to the new activity was seen as empty

Description of the problem:

Song objects are created by DetectSongActivity after a successful recognition by the server. They are passed to DisplayTextActivity in a bundle added to the intent. All the objects passed to DisplayTextActivity had empty fields, so it was impossible to process the songs and display the lyrics.

Solution:

Implementation of Serializable interface by Song class. Only classes implementing this interface can be passed to other activities.

4.4.2. Special Polish characters were not imported correctly to the database

Description of the problem:

Text files made by Ultrastar users are saved using different encodings:

- iso-8859-2
- win1251
- win1250

After importing them to the database, special Polish characters like “ą”, “ę” and “ś” were replaced by various Unicode symbols.

Solution:

The usage of iconv-lite module. It is used to convert the character encodings. In the application, all lyrics are converted using the 3 encodings listed above, and therefore the special Polish characters after each conversion are counted. The conversion with the most of them is saved to the database.

4.4.3. Lyrics from the files contain special characters that are not present in the recognized speech

Description of the problem:

Lyrics that are obtained from Ultrastar files contain characters like commas, dots and question marks. They are not present in the text created from recognized speech, so it creates a problem during song recognition.

Solution:

Each song stored in the database has a field called fullText. In this field, the full lyrics of the song are stored, but all the unnecessary special characters are removed before by SpecialCharsRemover.

4.4.4. Making HTTP requests from the client application did not work

Description of the problem:

At the beginning, there were plans to use HttpURLConnection class that is one of the basic classes provided in Java. After the implementation, the requests were not sent.

Solution:

The usage of Volley and RequestQueue.

4.4.5. Elasticsearch container could not start on the local machine because of lack of memory

Description of the problem:

During the development, all the containers of the application were working at the same time on local machine. After some time, the container for Elasticsearch database did not want to start.

Solution:

Problem with starting the container was caused by the lack of memory. To avoid it, environment variable has been passed to the container to reduce the memory usage to 512MB:

ES_JAVA_OPTS: -Xms512m -Xmx512m

4.4.6. Lyrics displayed to the user when opening the activity should match the moment in which he started speech recognition

Description of the problem:

When opening the activity that displays lyrics, the lyrics should match the time when they start to appear.

Solution:

The current time is stored when the user starts to sing. After receiving a response from the server, this collected time value is sent together with the Song object. Basing on this time and the song's tempo there is a calculation fired to find the line that matches the moment of starting displaying the lyrics. The selected line is the line which should be sung if the user does not stop singing after the listening finishes.

4.4.7. By default, using speech recognition intent provided by Google forces the screen to show the original view of speech recognizer

Description of the problem:

In the Android documentation, there is an example how to use the speech recognition. After implementing it the way it is described, the user sees the default view of Google speech recognition.

Solution:

In order to use the custom view, the implementation of SpeechRecognizer class has been developed and used in the activity.

4.4.8. Changes made in the source files are not seen by the docker container

Description of the problem:

During the development of the application, a lot of files have been changed multiple times. The changes were not seen by Docker containers. They had to be built again to examine the difference.

Solution:

Attaching volumes to docker containers. Volumes are the folders seen in Docker file system that can be changed from the outside. The volume is also used to persist the Elasticsearch data after shutting down the container. Attaching volumes can be done in docker-compose file:

```
volumes:  
  ./server:/var/app
```

4.4.9. The lyrics have to change dynamically in DisplayTextActivity

Description of the problem:

The lyrics displayed by DisplayTextActivity should change dynamically. It means that they should refresh and user does not have to move them manually.

Solution:

The usage of CountdownTimer class and onTick() listener. The timer ticks every 200 milliseconds and changes the displayed line of lyrics.

4.4.10. Application does not work properly in Ambient mode

Description of the problem:

In order to improve the battery life of the wearables, the ambient mode is starting after the lack of user activity. When the application enters the ambient mode, the lyrics do not change until waking up from this mode.

Solution:

The implementation of onEnterAmbient(), onUpdateAmbient(), and onExitAmbient() methods. When the device is in Ambient mode, there is an alarm ran every second and it forces the screen to update.

5 User interface of the complete application

To present the look of complete application, there were some screenshots taken. Screenshots of client application were made on rounded smartwatch, therefore the layout is inside a circle. When the smartwatch is squared, the layout fulfills the square. The screenshots are shown below:

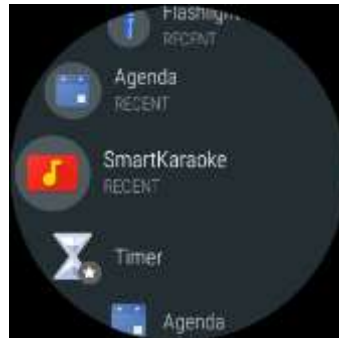


Figure 5.1 The client application icon displayed in Android Wear launcher



Figure 5.2 The starting activity of the application



Figure 5.3 Animation visible during singing

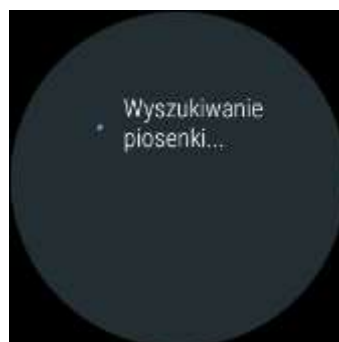


Figure 5.4 Text and animation displayed while waiting for server response



Figure 5.5 View of the activity displaying the lyrics

SmartKaraoke

Enter the password to see the admin panel

Login

Figure 5.6 View of the login page to the admin panel

<div> <div>Recognized</div> <div>Incorrectly recognized</div> <div>Not recognized</div> <div>Log off</div> </div>			
Lyrics	Artist	Song	Count
ofiaruję mojej dziewczynie	Róże Europy	Jedwab	14
jolka jolka	Budka suflera	Jolka Jolka	9
jak jedwab	Róże Europy	Jedwab	6
powiedz czemu	Ich Troje	Powiedz	3
z ciebie jest	Virgin	Dżaga	1

Figure 5.7 View of the page with list of recognized songs

Recognized	Incorrectly recognized	Not recognized	Log off
------------	-------------------------------	----------------	---------

Lyrics	Artist	Song
ofiaruję	Róże Europy	Jedwab
przeżyj to sam	Lombard	Przeżyj to sam
jak jedwab	Róże Europy	Jedwab
przeżyj to sam	Lombard	Przeżyj to sam
życie pędzi	Szymon Wydra & Carpe Diem	Poemat
było ciepłe lato	Maryla Rodowicz	Agnieszka

Figure 5.8 View of the page with incorrectly recognized songs

Recognized	Incorrectly recognized	Not recognized	Log off
------------	------------------------	-----------------------	---------

Lyrics	Count
baba koguta	3
not hot	1
zółty ananas	1

Figure 5.9 View of the page with not recognized lyrics

6 Application tests and verification

To make sure that the server API will work properly, it was built using TDD (Test Driven Development). It means that for all the classes and endpoints used by the smartwatch application, the tests were created before development. There are 2 types of tests created for the server application:

- Unit tests – It is the method of testing software by testing its basic elements – classes and functions.
- Functional tests – It is a type of black-box testing. It means that only the functionalities of the application are being examined without checking the internal structures.

The executed tests for the application with the descriptions are listed below:

Unit test 1

Class: *FileParser*

Name: It should parse the song correctly

Description: The demo file is located in the location given to the *FileParser* object. It is checked if the song returned by *parseFile()* method has all the proper attributes.

Unit test 2

Class: *PopularLyricsStorage*

Name: It should get data from Redis

Description: Redis client is mocked and the function *getSongData()* is executed. It is verified if the returned data is the same as the data in Redis.

Unit test 3

Class: *PopularLyricsStorage*

Name: It should save data to Redis

Description: Redis client is mocked and the function *saveSongData()* is executed. It is verified if the method *set()* of Redis client was executed with the proper attributes.

Unit test 4

Class: *SongResultNormalizer*

Name: It should normalize song data when searched lyrics are in the beginning

Description: Instance of *SpecialCharsRemover* class is mocked and the function *normalizeResult()* is executed. It is verified if the returned song data is correct.

Unit test 5

Class: *SongResultNormalizer*

Name: It should normalize song data when searched lyrics are not in the beginning

Description: The scenario is the same as in Unit Test 4, but the lyrics that have to be located in the song are now placed inside the song. This test case was invented in order to check if the location of searched lyrics that is returned by the server is correct.

Unit test 6

Class: *SongResultNormalizer*

Name: It should normalize song data when there are multiple songs found

Description: The scenario is the same as in Unit Test 4, but the function *normalizeResult()* is executed with the data containing 2 songs. It is checked if the function returns 2 songs as a result.

Unit test 7

Class: *SongSearcher*

Name: It should return a song when it is present in Elasticsearch

Description: Elasticsearch client is mocked and it returns a single song for a query with proper arguments. It is checked if the song returned by *findByLyrics()* is correct.

Unit test 8

Class: *SongSearcher*

Name: It should empty array when the song has not been found

Description: Elasticsearch client is mocked and it returns response saying that no songs were found for a query with proper arguments. It is checked if *findByLyrics()* function returns null.

Unit test 9

Class: *SongSearcher*

Name: It should return multiple songs when multiple songs have been found

Description: Elasticsearch client is mocked and it returns response saying there are more than 1 songs found for given lyrics and they have the same score. It is checked if *findByLyrics()* function returns both songs.

Unit test 10

Class: *SpecialCharsRemover*

Name: It should remove special chars from the string

Description: Function *removeSpecialChars()* is called with the string containing special characters as an argument. It is checked if the string was properly transformed.

Unit test 11

Class: *SpecialCharsRemover*

Name: It should return empty string when there is no string given

Description: Some syllables of the songs do not have the field with lyrics. Therefore, it is important to check if the function will return empty string for these cases.

Unit test 12

Class: *NotFoundSongsSaver*

Name: It should save not found song in Elasticsearch

Description: Elasticsearch client is mocked and the function *saveNotFoundSong()* is called. It is checked, if the mock function to save data was executed with proper arguments.

Unit test 13

Class: *FoundSongsSaver*

Name: It should save found song in Elasticsearch

Description: Elasticsearch client is mocked and the function *saveFoundSong()* is called. It is checked, if the mocked function to save data was executed with proper arguments.

Unit test 14

Class: *WrongSongsSaver*

Name: It should save incorrectly matched songs in Elasticsearch

Description: Elasticsearch client is mocked and the function *saveWrongSong()* is called. It is checked, if the mock function to save data was executed with proper arguments.

Functional test 1

Name: It should return 204 when there is no song available

Description: The endpoint for getting songs is called with the lyrics, for which there is no song in the database. It is checked, if the server returns 204.

Functional test 2

Name: It should return a song with correct lyrics when the song has been found

Description: The endpoint for getting songs is called with the lyrics, for which there is exactly one song in the database. It is checked, if the response is proper.

Functional test 3

Name: It should return multiple songs with correct lyrics when multiple songs have been found

Description: The endpoint for getting songs is called with the lyrics, for which there is more than one song in the database. It is checked, if the response is proper.

Functional test 4

Name: It save incorrect song recognitions reported by the user

Description: The endpoint for getting songs is called with the data of incorrectly recognized lyrics. It is checked if the server returns 200 and saves the data in the database.

The client application was created without the tests, but it was tested by a few people that received the .apk file. There were some issues that have been fixed after the tests:

- Speech recognition did not finish when closing the application
- The lyrics displayed for the recognized text did not appear in the proper time
- When connection for Google servers was unavailable, the application used to crash

Regarding the admin panel, it was tested manually by the author and no issues have been found so far.

7 Possible ways of future development

To help the application in reaching more users there are some improvements planned to help it in gaining real huge popularity:

- Recognizing the music – It means recognizing the music from the voice, not from singing the lyrics. It can be done by creating a database with attributes obtained from songs (e.g. tempo) to assign the recorded voice to one of them. It will be the most important feature to encourage users to change their karaoke application to this product.
- Storing recognized songs database for the user - Users like having their custom data in applications. The history of their searches can be stored on their smartwatch by using a small database like SQLite. It can be displayed in the addition activity. Selecting a song from the history can move the user directly to the lyrics
- Grading user's vocal skills - The format of text files used in Ultrastar application has a number indicating the pitch of the note. During the singing the most dominating sound pitch can be collected by the application and the user's singing can be rated.
- Adding a web client and application for other platforms - In order to increase the popularity of the application, the versions for other devices can be developed. The main emphasis should be focused on Android smartphones, iOS, Apple Watch and web version available from the browser.
- Monitoring for the server – To eliminate possible failures and avoid the downtime it will be good to add monitoring for Node.js service and the databases. It can be done by Telegraf to collect data and Grafana to provide dashboard accessible by the administrator.
- Ability to scroll the lyrics - Some users do not want to start listening from the moment selected by the application. It would be excellent for them to add the ability to move the lyrics to adjust them to the best moment.
- Additional languages support – In order to gain users from other countries than Poland, the interface can be translated to other languages. It can be easily implemented in Android Wear application by creating additional string resources.

8 Summary

To conclude, the process of development of the application that was shown in the thesis, has been successfully finished. After obtaining the required permissions to use lyrics, it can be published on Google Play store and reach first “real” users.

During the work on the thesis, the existing solutions have been analyzed with their advantages and disadvantages. It was noticed, that there are no karaoke applications for Android Wear that are worth installing. That is why creating the application was even more exciting. The created project containing use cases, class diagrams, architecture diagram and user interface mockups was very useful during the implementation.

The application created in this thesis is the first application for wearables created by the author. All the issues were caused by the lack of experience.

The created application will be named “SmartKaraoke”.

After creating the application, it is worth saying that starting the adventure with developing application for wearables is not difficult for people, who created some applications, (even amateur ones) for Android smartphones. Therefore, it is hard to understand why there are still not many products available for wearables on Google Play store.

9 References

- [1] Android Developers, `RecognitionListener`, [access January 23, 2018]
<https://developer.android.com/reference/android/speech/RecognitionListener.html>
- [2] Android Developers, `SpeechRecognizer`, [access January 23, 2018]
<https://developer.android.com/reference/android/speech/SpeechRecognizer.html>
- [3] Xamarin Inc., `Android Wear – Xamarin`, [access January 23, 2018]
<https://developer.xamarin.com/guides/android/wear/>
- [4] Balsamiq Studios, LLC, `Balsamiq, Rapid, effective and fun wireframing software`, [access January 23, 2018] <https://balsamiq.com/>
- [5] Calvio A., `Beginning Android Wearables`, Apress, 2015
- [6] Chauhan S., `Understanding Inversion of Control, Dependency Injection and Service Locator`, 2013, [access January 23, 2018]
<http://www.dotnettricks.com/learn/dependencyinjection/understanding-inversion-of-control-dependency-injection-and-service-locator>
- [7] Demicoli C., `Design Patterns: Cache-Aside Pattern`, [access January 23, 2018]
<https://blog.cdemi.io/design-patterns-cache-aside-pattern/>
- [8] Docker Inc., `Docker Compose | Docker Documentation`, [access January 23, 2018]
<https://docs.docker.com/compose/>
- [9] Daniel S.F., `Android Wearable Programming`, Birmingham, Packt Publishing, 2015
- [10] Elasticsearch BV, `Elastic Stack and Product Documentation`, [access January 23, 2018] <https://www.elastic.co/guide/index.html>
- [11] Fakhroutdinov K., `Dependency in UML`, [access January 23, 2018]
<https://www.uml-diagrams.org/dependency.html>
- [12] Jade C., `The Ongoing Decline of the Desktop Mac`, 2010, [access January 23, 2018]
<https://gigaom.com/2010/11/05/the-ongoing-decline-of-the-desktop-mac/>
- [13] Khare N., `Docker Cookbook`, Birmingham, Packt Publishing, 2015
- [14] Lyrics Mania, [access January 23, 2018] <https://www.lyricsmania.com/>
- [15] Miszczyszyn M., `Wzorce Projektowe: Dependency Injection`, [access January 23, 2018] <https://typeofweb.com/2016/07/07/wzorce-projektowe-dependency-injection/>
- [16] Musixmatch, `Musixmatch – Song Lyrics and Translations`, [access January 23, 2018]
<https://www.musixmatch.com/>
- [17] Node.js Foundation, `Node.js Docs`, [access January 23, 2018]
<https://nodejs.org/en/docs>

- [18] Powers S., Learning Node, Sebastopol, O'Reilly Media, Inc., 2016
- [19] Redislabs, Redis Documentation, [access January 23, 2018]
<https://redis.io/documentation>
- [20] Ruiz D. D. C., Goransson A., Professional Android Wearables, Indianapolis, John Wiley & Sons, Inc., 2015
- [21] Shazam Entertainment Ltd., Shazam – Music Discovery, Charts & Song Lyrics,
<https://www.shazam.com/>
- [22] SoundHound Inc., SoundHound – Home, [access January 23, 2018]
<https://soundhound.com>
- [23] S.C Evercoder Software S.R.L, Online Mockup, Wireframe & UI Prototyping Tool – Moqups, [Access January 23, 2018] <https://moqups.com/>
- [24] Ultraguide.net, .txt Files in More Depth, [access January 23, 2018]
<http://www.ultraguide.net/txtfiles.php>
- [25] W3Schools, Bootstrap 3 Tutorial, [access January 23, 2018]
<https://www.w3schools.com/bootstrap/>
- [26] Wikipedia, Transistor count, [access January 23, 2018]
https://en.wikipedia.org/wiki/Transistor_count
- [27] Wikipedia, Model-view-controller, [access January 23, 2018]
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [28] Zimmermann A., Opinion: Watchmakers get creative in face of smart threat, 2016 [access January 23, 2018] <http://money.cnn.com/2016/03/18/luxury/watches-smartwatches-industry/index.html>
- [29] Zulauf Carl, Memcached vs Redis: Direct Comparision, [access January 23, 2018]
<https://stackoverflow.com/questions/10558465/memcached-vs-redis>
- [30] Black J. et al., Why is Singleton considered an anti-pattern?, [access January 23, 2018] <https://stackoverflow.com/questions/12755539/why-is-singleton-considered-an-anti-pattern>

10 List of figures

Figure 1.1 Change in the number of transistors in processors between 1971 and 2011[26]	5
Figure 1.2 Mac laptop and desktop sales [12].....	6
Figure 1.3 Smartwatch sales forecast [28]	6
Figure 1.4 Musixmatch application.....	7
Figure 1.5 SoundHound application.....	8
Figure 1.6 Shazam for Android smartphones.....	9
Figure 1.7 Shazam for Android Wear smartwatches	9
Figure 1.8 LyricsMania application	10
Figure 1.9 Schema of the application flow	10
Figure 2.1 Use case diagram	13
Figure 2.2 Architecture schema	16
Figure 2.3 Class diagram of the client application.....	19
Figure 2.4 Class diagram of the server application.....	20
Figure 2.5 Prototype of activity for detecting song before starting listening.....	21
Figure 2.6 Prototype of activity for detecting song during the listening.....	21
Figure 2.7 Prototype of the message displayed while waiting for a response	22
Figure 2.8 Prototype of the view after receiving the response from the server	22
Figure 2.9 Prototype of reporting incorrect song recognition.....	22
Figure 2.10 Prototype of the form to log in to the admin panel	23
Figure 2.11 Prototype of the view with message about incorrect password	23
Figure 2.12 Prototype of the page with recognized songs	24
Figure 2.13 Prototype of the page with incorrectly recognized songs	24
Figure 2.14 Prototype of the page with lyrics that have not been recognized	25
Figure 2.15 Prototype of the page visible after logging off the admin panel.....	25
Figure 4.1 Reading from cache-aside pattern [7]	30
Figure 4.2 Schema of Dependency Injection pattern [6]	30
Figure 4.3 Schema of Model-View-Controller pattern [27]	31
Figure 5.1 The client application icon displayed in Android Wear launcher.....	34
Figure 5.2 The starting activity of the application	34
Figure 5.3 Animation visible during singing	34
Figure 5.4 Text and animation displayed while waiting for server response.....	34
Figure 5.5 View of the activity displaying the lyrics	35
Figure 5.6 View of the login page to the admin panel	35
Figure 5.7 View of the page with list of recognized songs	35
Figure 5.8 View of the page with incorrectly recognized songs.....	36
Figure 5.9 View of the page with not recognized lyrics	36