

Speech Recording Web Service and Application

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the Degree of
Engineer in the domain Technology and Telecommunication Systems
Study program: Telecommunications and Information Technologies

Thesis advisor(s):

Prof. Corneliu BURILEANU, Ph. D
Associate Prof. Horia CUCU, Ph.D

Student:

Cristian *MANOLACHE*

Bucharest
2017

APPENDIX 1

Annex 1bis

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department: Telecommunications

Department Director's Approval *:

Conf. Eduard POPOVICI, Ph D.

DIPLOMA THESIS
of student MANOLACHE R. Cristian, 441G



1. Thesis title: Speech Recording Web Service and Application

2. The student's original contribution will consist of (not including the documentation part):

The purpose of this thesis is to replace the current speech recording web application of the Speech and Dialogue (Speed) Research Laboratory with a new one implemented using new technologies due to security issues. The purpose of this application is to extend an existing database of recordings necessary to train acoustic models in the Romanian language.

The current web application is using Java applets which soon will no longer be supported by web browsers because of their security problems. The new database interface is made using endpoints (Java REST) and, as an addition, endpoints for admin users have been introduced. In the old version, the username and password were sent for each request; in the new one, after authentication, the user will receive a JWT (Java Web Token) containing the user ID, an expiration time and signed using a secret. This token will allow the user to access the database's resources. In order to allow requests from another domain outside the domain from which the resource originated, CORS (Cross-origin resource sharing) has been implemented. All these new features will be implemented from scratch using Java 2 Enterprise Edition.

3. The project is based on knowledge mainly from the following 3-4 courses:
Object-Oriented Programming, Data Structures and Algorithms, Computer Programming

4. The Intellectual Property upon the project belongs to: *Speed Research Laboratory*

5. The research is performed at the following location: *Speed Research Laboratory*



6. The hardware part of the project stays in the property of: *Speed Research Laboratory*

7. The thesis project was issued at the date: 28.07.2016

Thesis Advisor:

Prof. Corneliu BURILEANU, Ph. D.

Lect. Horia CUCU, Ph. D.

STUDENT:

Cristian MANOLACHE



STATEMENT OF ACADEMIC HONESTY

I hereby declare that the thesis "Speech Recording Web Service and Application", submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer of Science in the domain Technology and Telecommunication Systems, study program Telecommunications and Information Technologies, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, 04.07.2017

Cristian Manolache



(Student's signature)

TABLE OF CONTENTS

Table of Contents	7
List of Figures	9
List of Tables	11
Chapter 1 Introduction	15
1.1 Thesis Motivation.....	15
1.2 Main Objective.....	16
1.3 Specific Objectives.....	17
Chapter 2 Software Technologies for web apps, web services and databases.....	19
2.1 Java.....	19
2.1.1 Introduction.....	19
2.1.2 Java EE.....	20
2.2 PostgreSQL	20
2.2.1 Stored Procedures	20
2.2.2 JDBC – Java API for database interaction.....	21
2.3 Development Tools	21
2.3.1 Netbeans IDE	21
2.3.2 Web Server.....	21
2.3.3 Subversioning systems.....	22
Chapter 3 Web services	25
3.1 RESTful web services	25
3.1.1 Jersey – Java API for REST services.....	25
3.2 Web app authentication in web services	26
3.2.1 OAuth.....	26
3.2.2 JWT.....	27
3.3 Web services interaction with web apps	28

3.3.1	Introduction.....	28
3.3.2	JSON.....	29
3.4	CORS filter	29
Chapter 4	Application Description	33
4.1	General Description	33
4.2	Database Organization	36
4.3	Database stored procedures.....	39
4.4	Application Back-end: Endpoints	40
4.5	Application Back-end: Class architecture.....	42
4.6	Dataflow example	44
4.7	Implementation on a VM	46
4.8	Development methodology	46
Chapter 5	Conclusions.....	47
5.1	General Conclusions	47
5.2	Personal Contributions.....	47
5.3	Future Work	48
References	49	

LIST OF FIGURES

Figure 2.1 Web Server	22
Figure 2.2 Git working tree.....	23
Figure 3.1 Browser-Server interaction using JWT	27
Figure 3.2 Interaction between service provider and service consumer	28
Figure 3.3 CORS filter	30
Figure 3.4 Cross-site scripting attack.....	31
Figure 4.1 Application flowchart	34
Figure 4.2 Creating a new speaker.....	35
Figure 4.3 Recording a phrase	35
Figure 4.4 Creating a new user	36
Figure 4.5 Application architecture	36
Figure 4.6 Relations between tables in the database.....	38
Figure 4.7 Dataflow example.....	45

LIST OF TABLES

Table 3.1 JAX-RS annotations	26
------------------------------------	----

List of acronyms:

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

ASR – Automatic Speech Recognition

CORS - Cross-origin resource sharing

DBMS – Database Management Systems

DOM – Document Object Model

DVCS - Distributed Version Control System

EE – Enterprise Edition

EJB - Enterprise Java Beans

GUI - Graphical User Interface

HMAC - Hash-based message authentication code

HTTP - Hypertext Transfer Protocol

IDE – Integrated Development Environment

JAR – Java archive

JDBC – Java Database Connectivity

JNDI - Java Naming and Directory Interface

JPA - Java Persistence API

JS – JavaScript

JSON - JavaScript Object Notation

JSP - JavaServer Pages

JSR - Java Specification Request

JTA - Java Transaction API

JVM – Java virtual machine

JWT - JSON Web Token

NPAPI - Netscape Plugin Application Programming Interface

ODBC - Open Database Connectivity

OS – Operating System

REST - RESTful Web Services

RMI - Remote Method Invocation

SE – Standard Edition

SNR – Signal-to-Noise Ratio

SQL - Structured Query Language

TCP - Transmission Control Protocol

URI - Uniform Resource Identifiers

URL - Uniform Resource Locator

VCS - Version Control System

VM – Virtual Machine

WAR - Web application Archive

WORA – Write Once, Run Anywhere

XML - eXtensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 THESIS MOTIVATION

In the last decade, there has been a growing demand of speech recognition technology. This technology implies a machine or a program to recognize words in a spoken language and transform them in a machine-readable format. During the past years, more and more speech recognition applications have been developed. These applications have utility in various domains such as in car systems, where a simple voice command may be used to initiate a phone call, change radio stations or in-home automation (smart home), where lighting, air conditioning, security can be controlled through the human voice. Other examples include domains like medical documentation, aerospace, automatic translation, mobile telephony, robotics, speech-to-text transcriber, etc. Some of the most notable speech recognition systems would be Microsoft's Cortana and Apple's Siri.

Speech recognition, also known as ASR (Automatic Speech Recognition) is more and more popular because it is a very fast and natural way of communication. The recognition is made based on the extraction of some voice parameters from the voice signal and uses an acoustic model, a phonetic model, and a language model. The acoustic model represents the relationship between an audio signal and phonemes. Phonemes are the fundamental sound units in a spoken language. The phonetic model makes the bond between the acoustic model and the language model. A language model's purpose is to estimate the probability of a sequence of words to be a valid sentence of the respective language.

As I mentioned before, a very important part of an ASR consists in the acoustic model, which usually is built upon a set of recorded audio clips and the respective annotated text. The acoustic model also requires the existence of a phonetic dictionary which specifies the manner in which the words from transcribed text are pronounced.

At the current moment there are no other free Romanian annotated speech databases. Our goal is to extend the database in Romanian language, to acquire data from as many users as possible, in order to preform complex machine learning techniques and improve the current results regarding speech based applications.

The Speed laboratory is one of the leading speech research groups in Romania and is currently concerned with scientific research in all areas of Spoken Language Technology, including:

- Automatic Speech Recognition and Text-to-Speech synthesis
- Speaker Recognition
- Spoken Term Detection, Spoken Document Indexing/Retrieval

In order to conduct research in domains like ASR or Speaker Recognition, a large annotated speech database in the Romanian language is required.

1.2 MAIN OBJECTIVE

In this context, the main objective of this thesis is to develop a new Speech Recorder application used for creating and extending a database of voice recordings necessary for training acoustic models in Romanian language. Due to security issues, the old application randered obsolete, thus, we wish to implement a new one, more reliable and mentainable, adjusted to new techonologies.

The application above mentioned consists in three parts:

- Front-end
- Back-end
- Database

The front-end part represents represents the client interface from which he can send various HTTP(Hypertext Transfer Protocol) requests (such as GET or POST) to the web service. This part is made in JS(Javascript) which is a programming language of the web. JS runs on the client side and is used to design web pages as well as program their behaviour. All modern web browsers support it without plug-ins. The old application used Java applets which are no longer supported by web browsers due to security issues. The Java plug-in for web browsers relies on the cross platform plugin architecture NPAPI(Netscape Plugin Application Programming Interface), which has been supported by all major web browsers for over a decade.

The back-end part represents the web service in which requests are handled and proccessed by interacting with the database. The backend is made in Java which will be presented in the following chapter.

The database was created by importing the information from the old database and making various changes to it, including table modifications and creating new stored procedures or modifying existing ones in order to interact with the back-end and provide new functionalities.

1.3 SPECIFIC OBJECTIVES

This project was developed in collaboration with my colleagues from Speed laboratory and it was a team effort. My part in this project was to develop the back-end and set-up the database.

- The back-end of the new web application will be designed using REST architecture.
- Implementation of endpoints.
- Creation of SQL scripts in order to update the existing database.
- Development of stored procedures, which will be called by the web service in order to interact with the database (about 15-20 procedures of 10-15 SQL lines).
- Implementation of an authentication system than relies on JWT(JSON Web Token).
- A CORS(Cross-origin resource sharing) filter will be used as a security measure to allow the server to define a set of origins which are allowed to read information using a web browser.
- Implementation on a VM(Virtual Machine) and testing.

CHAPTER 2

SOFTWARE TECHNOLOGIES FOR WEB APPS, WEB SERVICES AND DATABASES

2.1 JAVA

2.1.1 *Introduction*

Java is a computer programming language and at the moment one of the most powerful and important ones. It was created by Sun Microsystems (which was later acquired by Oracle), the project was initiated in 1991 and the first implementation, Java 1.0, was released in 1995. Java has the following important characteristics:

- High-level
- Object-oriented
- Portable
- Open-source

A high-level programming language is more or less independent from a certain type of computer and is closer to the human language. On the opposite side we have low-level programming languages such as assembly languages which are closer to machine language.

Through object-oriented, we can say that the information obtained after running a program is not obtained just by applying a set of algorithms on the input data but through interactions

between entities called objects. As comparison, the procedural programming languages such as C and Pascal represent a set of instructions which form an algorithm.

Java is platform independent which means that any program written in this language can be executed regardless of the operating system (Windows, Linux). This is possible because the programs written in Java are not executed by the operating system but by a virtual machine called Java Virtual Machine, which is available for any operating system. JVM takes the Java files (.class files) and makes the operating system understand them. Because JVM is available for any operating system we have the "write once, run anywhere" (WORA) concept and thus the portability characteristic. Unfortunately, this characteristic comes with a downside, in the sense of increased execution time due to the necessity of the JVM to make the interpretation. Thus Java will not be used in applications such as video games where real-time precision is required but oriented more to business applications.

By open-source, we understand that anyone can make contributions or improvements to improve the programming language so it may be used by others.

Java is one of the most popular programming languages and it's very used in client-server web applications.

2.1.2 Java EE

Java EE is a computing platform used for development and deployment of enterprise software (such as web services). The Java EE platform is built on top of the Java SE platform and it provides an API and runtime environment for developing and running large-scale, multi-tiered, and secure network applications. Thus, we may say that Java EE is a collection of API, services and protocols that are meant for developing enterprise applications. In the following list, we will name a few of these technologies : JDBC, JAX-RS (API for RESTful Web Services), JPA, RMI, EJB, JTA, JSP, JNDI. In most situations, when developing a web service, only a few of these technologies are used.

2.2 POSTGRESQL

"PostgreSQL is a powerful, open source object-relational database system. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages)." ACID stands for Atomicity, Consistency, Isolation, Durability and represents a set of properties of database transactions. The main function of PostgreSQL as a database server is to securely store data and return data in response to requests from software applications. PostgreSQL is available on most operating systems such as Linux and Windows, it is also free and open source.

2.2.1 Stored Procedures

A stored procedure is a set of statements that can create, retrieve, modify or delete data in a database, somewhat similar to a function. They are called stored because they are stored in a relational database management system allowing them to be used by other programs.

Advantages:

A stored procedure can help us in the matter of security. For example, some data may be accessed only by admin users; the stored procedure will check if the user that is trying to access the resource has admin rights and returns the proper response depending on the case. The network traffic between clients and servers can be reduced by using stored procedures due to the fact that the code lines are not transmitted, but only the call of execution is sent over the network.

Disadvantages:

Most storage procedure languages are vendor specific, meaning that if for example we change database vendors, we will most likely have to change the rewrite the code for the stored procedures.

2.2.2 JDBC – Java API for database interaction

Java Database Connectivity (JDBC) is an API for the Java programming language , which allows a client to perform various operations on a database. JDBC is a part of Java SE. The main features of the JDBC API are the following:

- Establish a connection with a database
- Send SQL statements
- Process the results

We can divide JDBC into four components:

- The API : used to execute SQL statements;
- The Driver Manager: the class which is considered the backbone of the JDBC due to the fact that it defines objects which connect Java applications to a JDBC driver;
- The Test Suite: this component will help in determining whether the JDBC drivers will run the program or not. This is done by testing the features of the API.
- ODBC Bridge: it provides JDBC access via ODBC drivers. ODBC is an API for accessing DBMS;

2.3 DEVELOPMENT TOOLS

2.3.1 Netbeans IDE

Netbeans IDE is an application development environment which is free and open-source and available for most operating systems, such as Windows, Mac and Linux. An IDE represents more than a text editor, making it more friendly to programmers. Netbeans IDE includes the following features: autoindent, automatic placement of brackets, formatting, highlighting. It also gives tips and warnings regarding the code and can generate code like inserting getters and setter for example. The projects are displayed in an orderly fashion, thus keeping a clear overview. This is very helpful in the case of large application with many files. Netbeans IDE also comes with a debugger; putting a breakpoint in the code or stepping one line code at a time are just some of the features it offers.

2.3.2 Web Server

When we refer to web servers we may say that a web server is a computer specifically dedicated to serve web pages to clients or that a web server is a program which through the use of HTTP serves files which form web pages at the request of users. In order to host a web site, one must have a web server.

2.3.2.1 Apache Tomcat

Apache Tomcat is a free and open source web server. „This software is an implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies”. When developing a web application, in order to make it accessible to the rest of the world, it needs to be deployed on a web server.

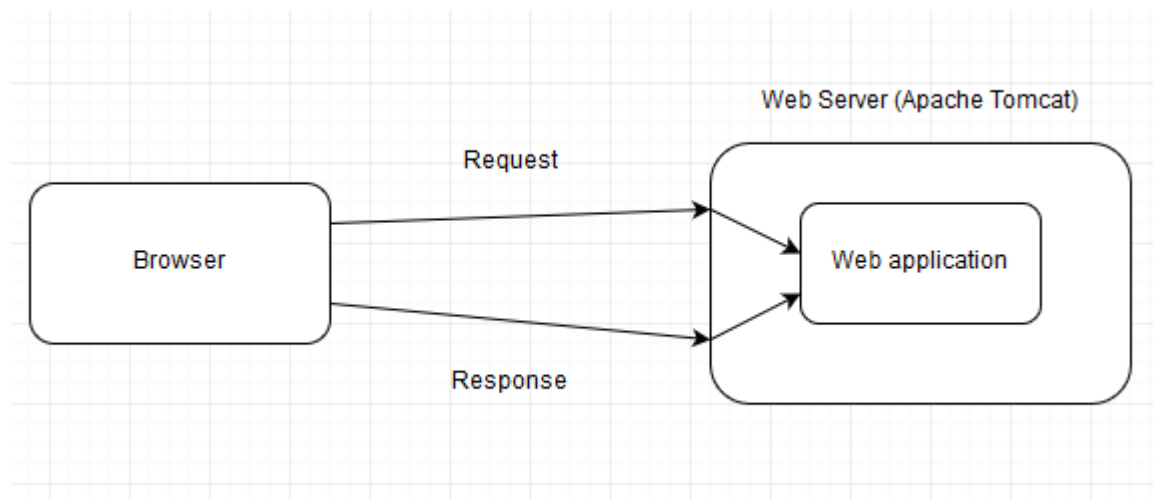


Figure 2.1 Web Server

The user sends from his browser a HTTP request over a TCP connection which is parsed into an object. The response object is send back by Tomcat as a HTTP response via the same TCP connection.

2.3.3 Subversioning systems

When we talk about subversioning, we think about VCS (Version Control System) which „is a software that helps software developers to work together and maintain a complete history of their work”. There are 3 major goals that this kind of system has to achieve:

1. To allow developers to work simultaneously;
2. Making sure that the changes made by one developer do not overwrite another's;
3. Keeping every version of the program and anything else that has been updated such as documentation.

2.3.3.1 Git

Git is a VCS . Git has a distributed architecture, making it a DVCS (Distributed Version Control System), meaning that „every developer's working copy of the code is also a repository that can contain the full history of all changes”, compared to the old VCS where the repository could be found in only one single place. The files of a project can be found in three different stages:

- Modified (or unstaged): the files have been modified on your local repository, but have not yet been committed.
- Staged: these files have been marked that they are ready to go into the next commit
- Committed: the files have been safely stored in the git repository.

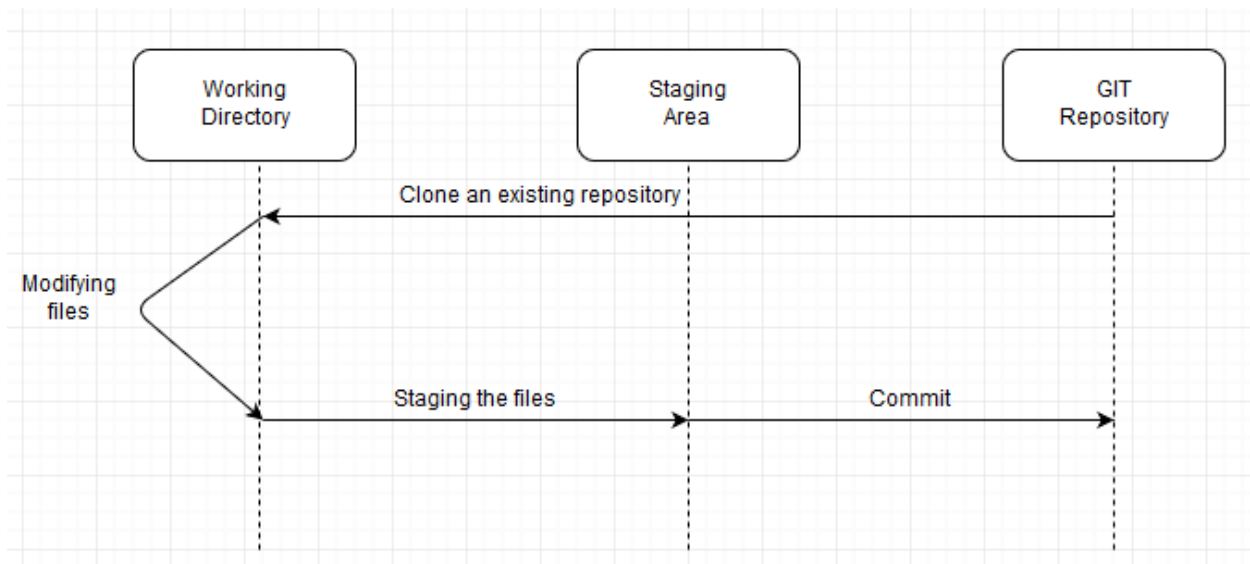


Figure 2.2 Git working tree

Another feature of Git is called branching. Branching is a way to diverge from the main line of development and to continue to work without interfering with that line. The default branch in Git is master. If for example we decide to create another branch called testing, we could make commit to that branch and the master branch will remain unchanged, thus allowing us to experiment on that branch while having the master branch as backup if something goes wrong.

CHAPTER 3

WEB SERVICES

3.1 RESTFUL WEB SERVICES

REST is an architectural style in which everything (data and functionality) is a resource which can be accessed through web links (URIs). RESTful web services are used very much in developing APIs for web-based applications.

RESTful applications have the following features:

- Resources are identified by URIs;
- Resources may be accessed through HTTP methods: GET (retrieves the resource; Note: a resource may not be changed via a GET request), POST (creates a new resource), PUT (updates a resource), DELETE (removes a resource);
- Resources can be accessed in many formats such as HTML, XML, JSON etc.
- Resource interaction is stateless.

3.1.1 Jersey – Java API for REST services

The REST support for Java is made via JSR (Java Specification Request) 311. „This specification is called JAX-RS (The Java API for RESTful Web Services). JAX-RS uses annotations to define the REST relevance of Java classes”.

„Jersey is the reference implementation for the JSR 311 specification. The Jersey implementation provides a library to implement Restful webservices in a Java servlet container and a client library to communicate with a RESTful webservice”.

The following table presents a list with some of the most important JAX-RS annotations:

Table 3.1 JAX-RS annotations

Annotation	Description
<code>@PATH(your_path)</code>	Sets the path to the resource to <code>base_URL/your_path</code>
<code>@POST</code>	The method will respond to a POST request.
<code>@GET</code>	The method will respond to a GET request.
<code>@PUT</code>	The method will respond to a PUT request.
<code>@DELETE</code>	The method will respond to a DELETE request.
<code>@Produces(MediaType. [types])</code>	This method defines which MIME type is returned as response to the accessed method.
<code>@Consumes(MediaType. [types])</code>	This method defines which MIME type is expected to be received by the method.
<code>@PathParam</code>	This method is used to take parameters from the URL and insert them into the respective method.

3.2 WEB APP AUTHENTICATION IN WEB SERVICES

One of the most important things that we must take into consideration when creating a web application is security. Through authentication we understand the verification process of the user. There are various ways to determine if a user is who he/she claims to be, the most common being by providing a username and password. Authentication is not to be confused with authorization or access control which represents whether a user has the rights to access specific resource.

3.2.1 Oauth

The OAuth 2.0 specification defines a delegation protocol that is useful for conveying authorization decisions across a network of web-enabled applications and APIs. OAuth is mostly used in providing mechanisms for user authentication but has other application as well. In some cases, OAuth is mistakenly considered an authentication protocol. This confusion comes from the fact that OAuth is used inside of authentication protocols.

JWT (RFC 7519) is an extension for OAuth.

3.2.2 JWT

„JWT is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object”. We can verify this information due to the fact that it is digitally signed using a secret with the HMAC algorithm. Because of its compact size, JWT can be sent through a URL, HTTP header or POST parameter. The payload of the JWT has all the necessary information about the user, thus we can say it's self-contained. The most common use of JWT is in Authentication. The user logs in with his or her user and password, if the credentials provided are correct, the user will receive a token which will be sent together with each subsequent request, thus allowing the user access to services and resources. The figure bellow shows this process:

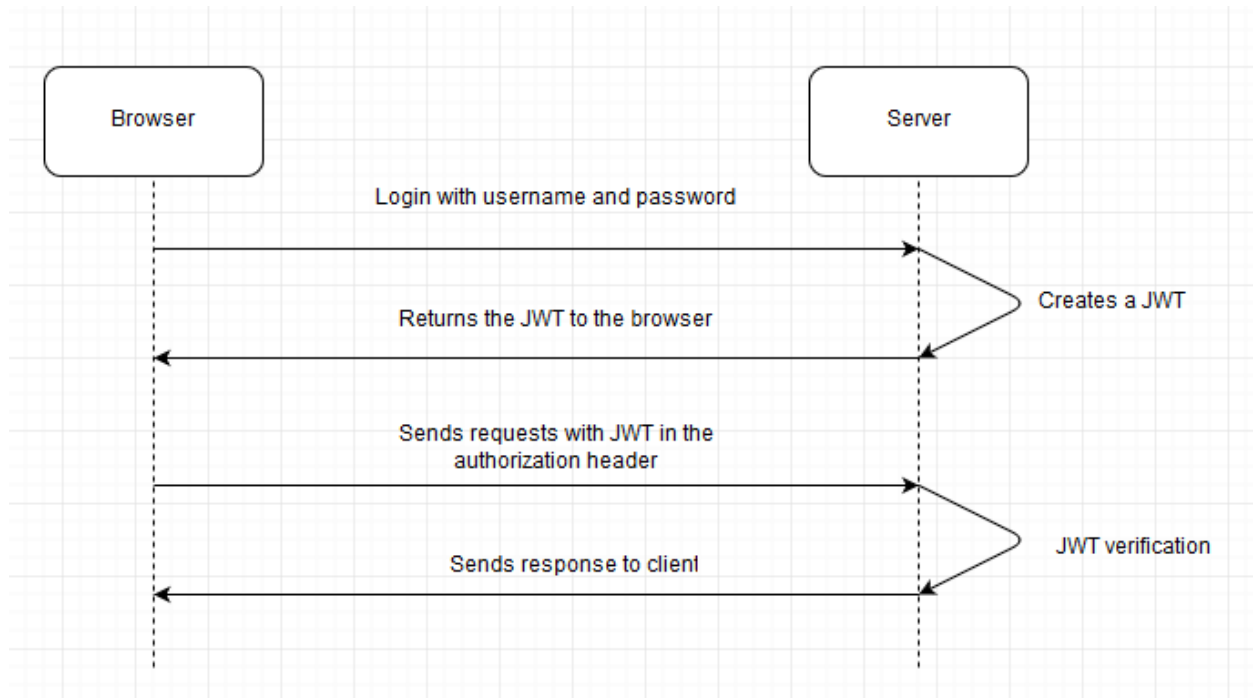


Figure 3.1 Browser-Server interaction using JWT

Regarding the structure of the JWT, it consists of three parts:

- Header
- Payload
- Signature

The header in turn is made from two parts: the hashing algorithm used and the type of the token. The header could look like this:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

The payload is the second part of the token and contains the claims. Claims represent statements about a certain entity, usually a user, and additional metadata. Payload example:

```
{
  "id": "14",
  "name": "Mike Hammond",
  "admin": true
}
```

Both the header and the payload are Base64Url encoded.

The signature is the third and final part of the JWT. In order to make the signature, we require the following parts: the encoded header, the encoded payload and a secret. Then we use the algorithm in the header. For example:

```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    secret)
```

The signature is used to verify that whoever sends a request together with the JWT is who it claims to be.

The end product will be formed from three base64 strings separated by dots. This format is compact and thus easily passed in HTML and HTTP environments.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE0IiwibmFtZSI6Ikp1pa2UgSGFtbW9uZCIsImFkbWwluIjoiY0cnVlQ.PxbLS3HYCwCElYYBrxheri3vBvUttt41Gn4ZzUg9dVk

3.3 WEB SERVICES INTERACTION WITH WEB APPS

3.3.1 Introduction

A web service is a software component which can be found on the internet via a simple find mechanism. In order to make itself available, it uses a standardized XML messaging system. Another key feature of a web service is that it shouldn't be tied to any programming language or operating system, thus being able to exchange data between applications, respectively systems.

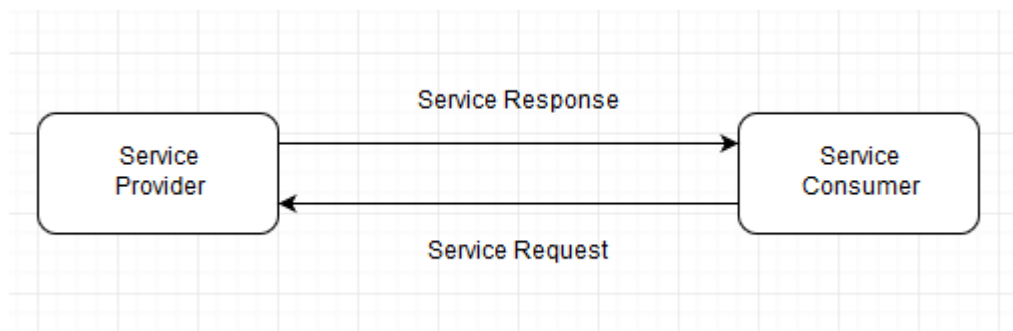


Figure 3.2 Interaction between service provider and service consumer

XML is a text-based markup language. A markup language is a set of tags placed in a text so that we can demarcate certain components of the document.

Here, we have an example:

```
<message>
    <text>Hello world!<text>
</message>
```

Thus, we can say that XML is merely information wrapped in tags. Due to the fact that it does not perform any algorithms or computational operations, XML cannot be considered a programming language. XML was designed to store and transport data.

3.3.2 JSON

JSON is a minimal, readable data format, used as an alternative to XML to store data and transmit data between web server and web application. JSON is text and we can convert any JSON into a JavaScript object and vice-versa. JSON is made from two parts: keys and values. JSON is a collection of key-value pairs. The key is always a string while the value can be a string, a number, a boolean or even an array. Example:

```
{"name":"Mike", "age":22, "city":"London"}
```

If we are to compare JSON with XML, we can say that regarding similarities, both JSON and XML are human readable, hierarchical (meaning that we have values within values), they can be parsed by most programming languages and they can be fetched using a HTTP request. We can also note some differences such as that JSON does not use end tags, is shorter, quicker to read and write and can use arrays. The biggest difference between XML and JSON is the way they are being parsed. XML is parsed using an XML parser, while JSON is parsed with a JavaScript function. The advantages JSON has over XML is that JSON is more easily parsed than XML and that JSON is parsed into a ready to use JavaScript object.

3.4 CORS FILTER

CORS is a way for web browsers to access resources on other domains than the origin. Web browsers implement a security concept called same-origin policy that prevents Javascript code from making requests from a different domain. CORS allows the Javascript code to consume a REST API from a different domain, thus relaxing the same origin policy. In order to better understand CORS, we will use an example:

A HTML page served from domain-a will request an image from domain-b (different from the origin which is domain-a).

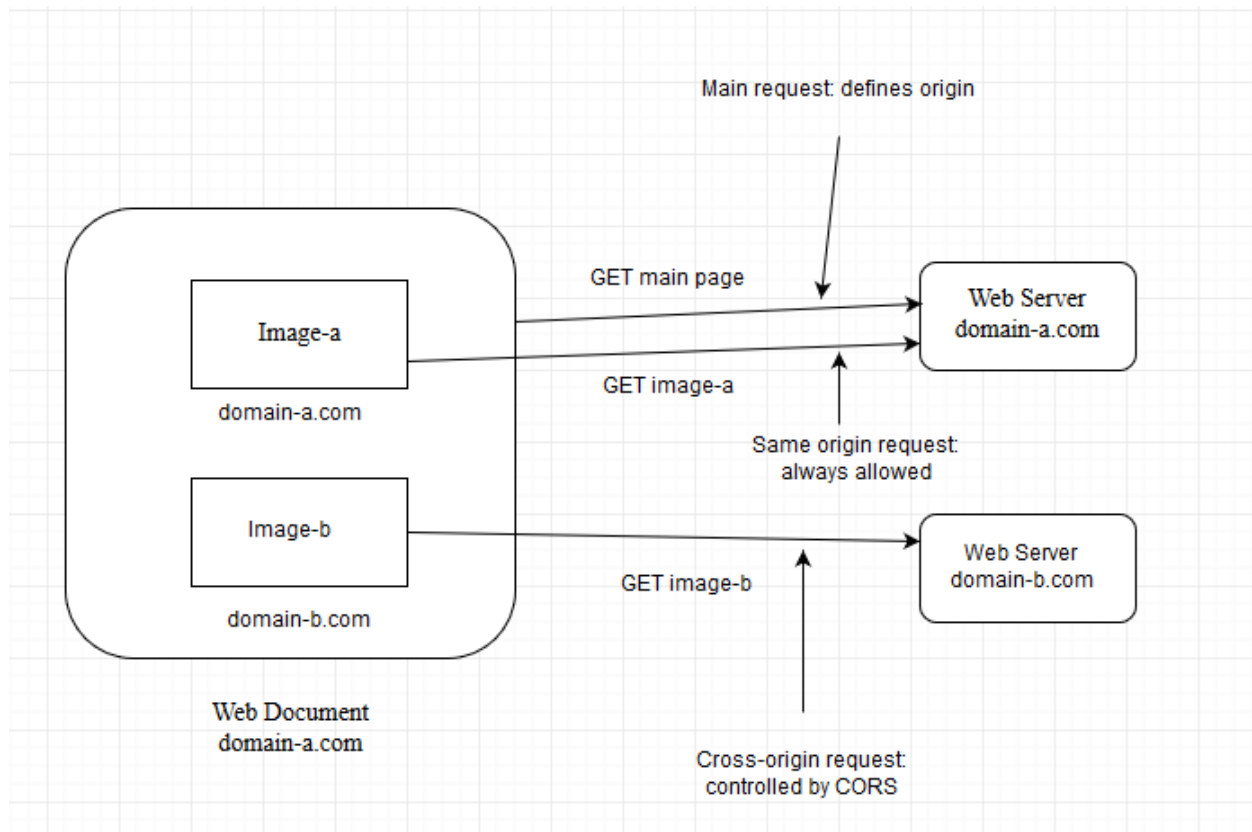


Figure 3.3 CORS filter

The same-origin policy defines what messages one site may sent to another. For example, this policy allows GET requests, but denies POST, PUT and DELETE methods. The purpose of the same-origin policy is to allow users to access some untrusted sites through trusted ones without risking any interference from the former.

A type of vulnerability used to bypass the same-origin policy is Cross-site scripting. The Cross-site scripting represents an injection attack on the client-side. A user viewing a vulnerable site will have a script injected into his web page. These scripts usually have malicious purpose. „In order for an XSS attack to take place the vulnerable website needs to directly include user input in its pages. An attacker can then insert a string that will be used within the web page and treated as code by the victim’s browser”.

The figure bellow shows a simple Cross-site scripting attack:

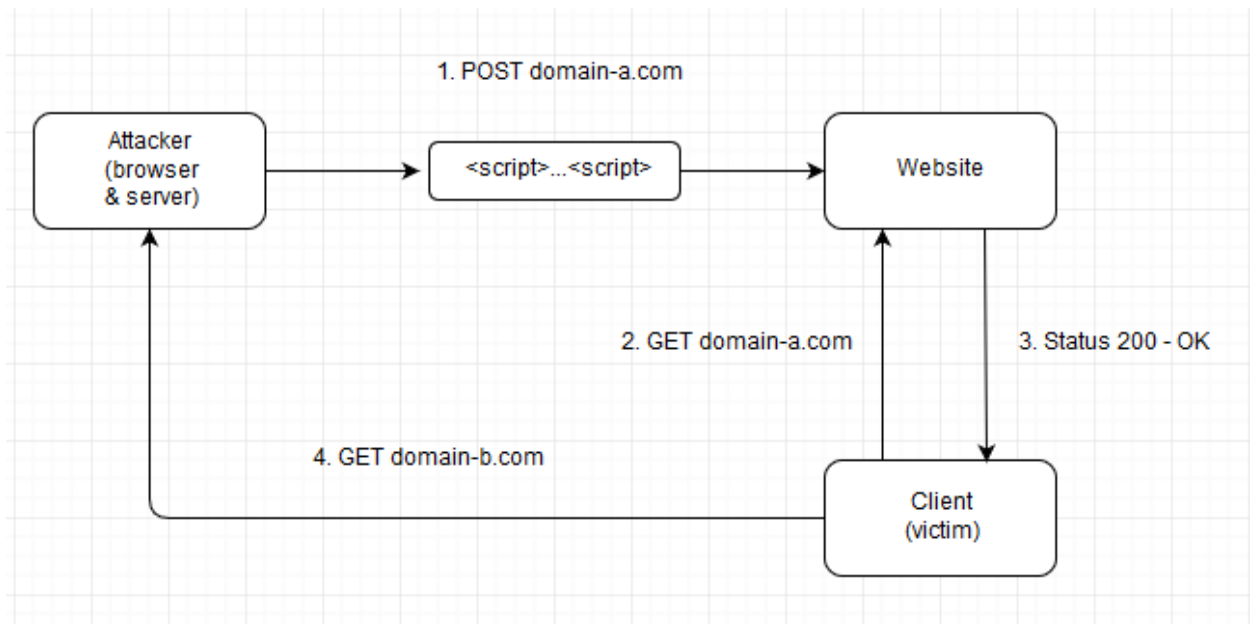


Figure 3.4 Cross-site scripting attack

1. The attacker will inject a script by submitting a vulnerable form in the database of the website.
2. The client requests a web page from the website.
3. The client's browser will receive the page with the attacker's payload as part of the HTML body.
4. The malicious script in the HTML body will be executed by the client's browser.

These malicious scripts can do a lot of harm. Some examples would be to read and make random modifications to the browser's DOM or send requests with random content to random destinations. Because the script has access to the same resources as the web page, it will have access to the cookies which often store tokens. Thus, the attacker could impersonate the user.

CHAPTER 4

APPLICATION DESCRIPTION

4.1 GENERAL DESCRIPTION

The main purpose of this application is to enable the end user to record and store his/her speech utterances. These speech utterances are mainly used to create large, annotated speech corpora which can be used to develop acoustic models for various spoken language technology applications such as: speech recognition, speaker recognition and speech synthesis.

In this application we encounter two types of users: normal users and admin users.

Depending on the type, the users have different functionalities: besides the main feature of recording which is available to both types of users, normal users can manage their speakers list, by creating or update existing speakers while admin users can access a list of all speakers in the database as well as a list of all users. Admin users can also delete a speaker/user besides creating/updating one.

An user represents the account that one or more persons use to access the web site. A speaker represents the actual person that makes the recording. One user may have one or more speakers. For example, for internship students there will be only one user but for each student there will be assigned a speaker.

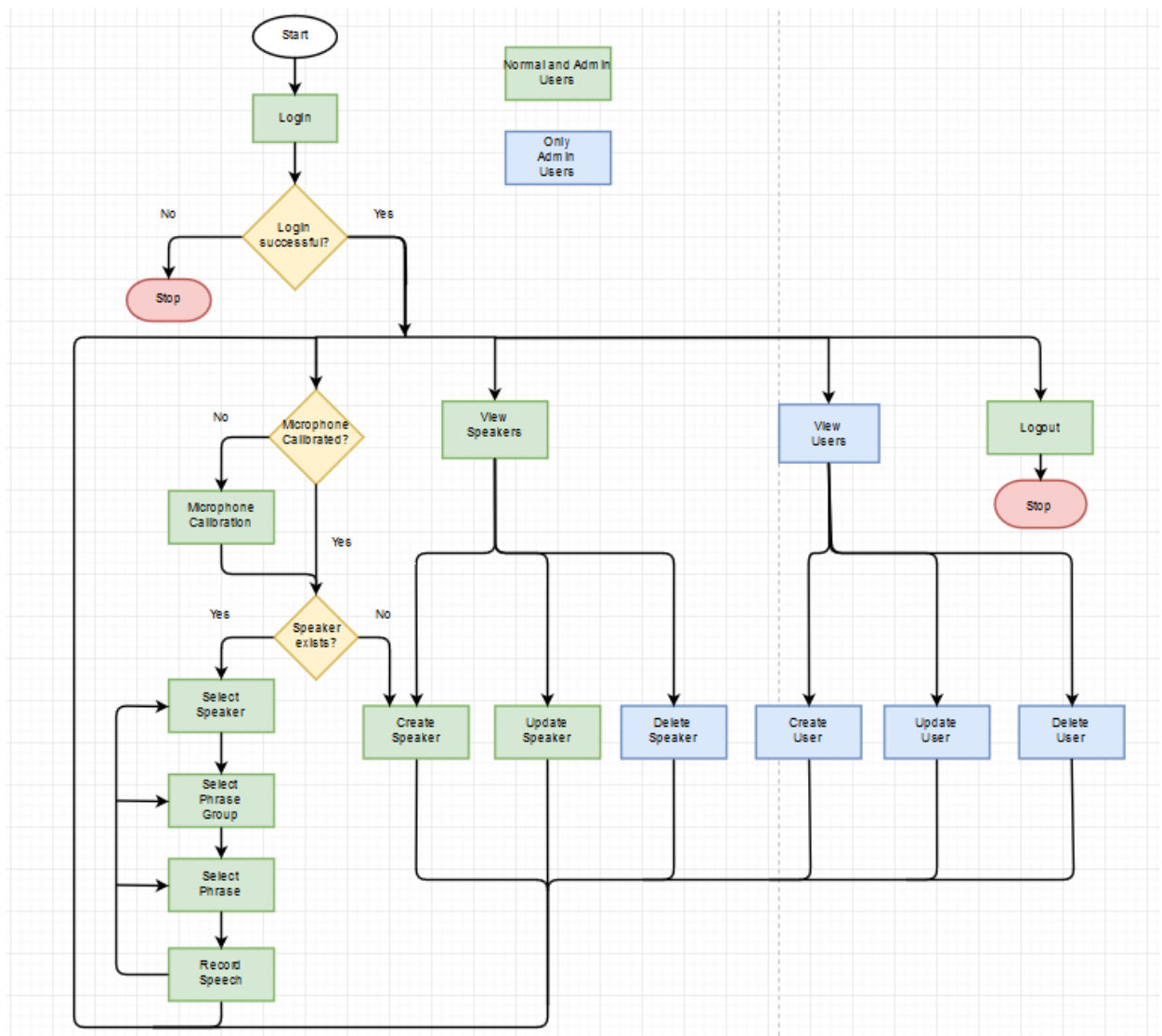


Figure 4.1 Application flowchart

Use cases:

- Record speech (login, microphone calibration, create speaker (if necessary), recording);
- Create/Update speaker (login, view speakers, create/update speaker);
- Create/Update user (login, view users, create/update user);
- Delete speaker (login, view speakers, delete speaker);
- Delete user (login, view users, delete user);

Following up, the steps will be presented:

1. Login: firstly, to use the application, a user must enter his or her credentials in the log in menu. In the upper right corner, the user can find 3-4 tabs depending on the type of user: Users (only for admin users), Speakers, Record and Logout.
2. View speakers in the database: by selecting the Speakers tab, a list with speakers can be found. For normal users, only the speakers belonging to that user may be seen, while admin users have access a list with all the speakers in the database, also knowing to which user they belong to.
3. Create a new speaker: in the Speakers tab, one may add a new speaker by introducing the following: first name, last name, email, mother language, gender, age, other speech characteristics.

First Name	Last Name
<input type="text"/>	<input type="text"/>
Email	Mother Tongue
<input type="text"/>	<input type="text"/>
Gender	Age
<input type="text"/>	<input type="text"/>
Other Speech characteristics	
<input type="text"/>	
<input type="button" value="SAVE CHANGES"/>	<input type="button" value="CANCEL"/>

Figure 4.2 Creating a new speaker

- Update a speaker: a speaker's information can be edited by selecting the edit option next to the respective speaker in the speaker tab.
- Record a phrase group: If the log in is successful, the user must perform two recordings, one without speech in order to record the background noise and one with speech. This is necessary to determine if the environmental conditions are suitable for recording, in contrast, we may encounter cases such as: low SNR, unplugged microphone, uncalibrated microphone and so on. After selecting the Recording tab, a user must firstly select his speaker from the speaker list and the phrase group for which he wishes to perform the recordings. Once the phrase group has been selected, a sentence or a row of numbers will appear on the screen, this being the first phrase of the phrase group. The user is free to browse through the list of phrases by using the arrow buttons or by simply introducing the desired phrase number from the respective group. The phrases will also show a recorded status, stating whether the phrase was recorded or not by the current speaker. Phrases with recorded status will have a playback option. Problems may also occur during the recording, for example a recording started too soon or ended too late, in which case the application informs the user through an error message of the problem and the user will have to make the recording again.

Cristian Manolache

▼

misc #1

▼

⏮

1

⏭

▶

🎤

recorded

/ 400

Specialiștii consideră că o bursă puternică este în măsură să atragă investiții străine considerabile.

Figure 4.3 Recording a phrase

The following use cases are exclusive to admin users:

- View users in the database: in the Users tab, we can find a list of all existing users, with some details: first name, last name, email, username, admin user status, ability to login status.

7. Create a new user: also in the Users tab, we can create a new user. The window for creating a new user is different from the one for creating a new speaker as can be seen in the image bellow:

Figure 4.4 Creating a new user

As it can be seen, we have the basic fields: first name, last name, username, email and password as well as 4 abilities:

- Login: which can enable (if checked) the user to login into his account;
 - Listen: allows all the speakers belonging to that user to use the playback option;
 - Modify: this field gives the ability to modify an existing recording;
 - Admin: states whether the user has admin rights or not.
8. Update an user: in the list of users, an admin has an actions column with two actions for each user. One of the actions consists in modifying an existing user.
 9. Delete an user: the second action in the actions column in the users tab is the deletion action which removes the user from the database.
 10. Delete a speaker: in the Speakers tab, an admin user can delete any speaker whether the speaker belongs to the user or not.

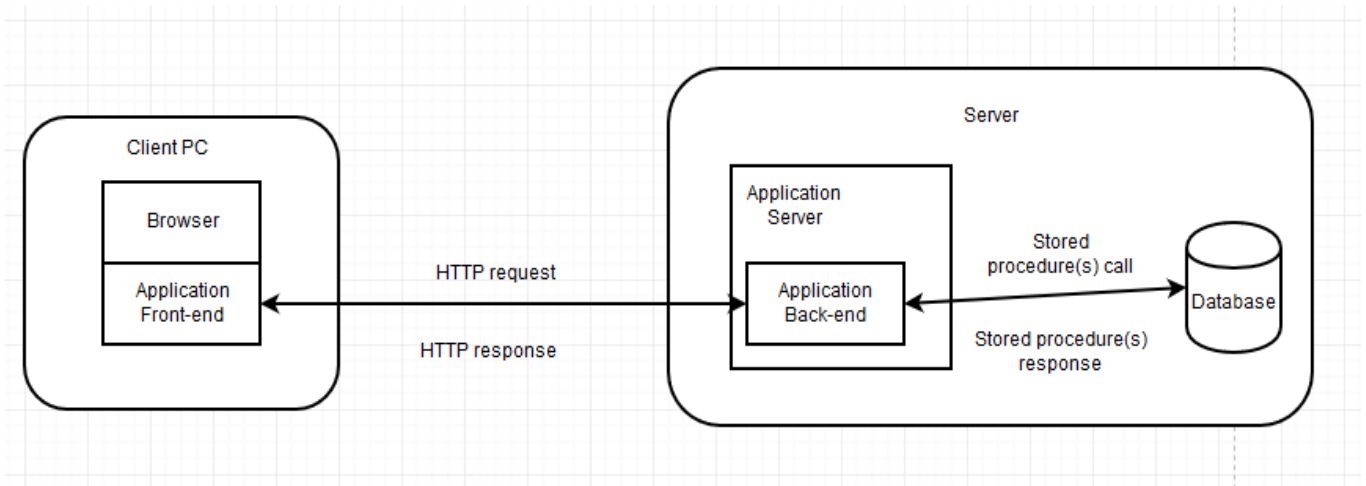


Figure 4.5 Application architecture

4.2 DATABASE ORGANIZATION

The database is created in pgAdmin and deployed on a VM. The access to the database is made using the JDBC API. The database consists of 6 tables:

1. audio_clips: stores data about the audio clips, including the audio clips themselves;
2. miscellaneous: stores the speaker agreement;
3. phrase_groups: stores data regarding phrase groups;
4. phrases: stores the phrases and information about them;
5. speakers: stores information about the speakers;

6. users: stores information about the users.

The fields for each table and their types are presented bellow:

- audio_clips :
 - speaker_id (integer): contains the id of the speaker who made the recording;
 - phrase_group_id (integer): contains the id of the phrase group from which the recording belongs;
 - phrase_id (integer): contains the id of the phrase which was uttered to make the recording;
 - created_date (date): contains the date at which the recording was made;
 - wav_byte_array (bytea): contains the recording wav file as a sequence of bytes.
- miscellaneous:
 - speaker_agreement (text): contains the list of terms every speaker has to agree to in order to make recordings.
- phrase_grops:
 - phrase_group_id (integer): contains the unique identifier for each phrase group;
 - group_name (character varying): contains the name of the respective phrase group;
 - group_size (smallint): contains the number of phrases found in the phrase group;
 - description (character varying): contains a short description of the phrase group, showing to which domain they are related to such as phrases from newspapers and magazines.
- phrases:
 - phrase_group_id (integer): contains the id of the phrase group from which the phrase belongs to;
 - phrase_id (integer): contains the unique identifier of each phrase;
 - phrase_text (character varying): contains the text of the phrase.
- speakers:
 - speaker_id (integer): contains the unique identifier for each speaker;
 - user_id (integer): contains the id of the user from which the speaker belongs to;
 - first_name (character varying): contains the first name of the speaker;
 - last_name (character varying): contains the last name of the speaker;
 - mother_language (character varying): contains the native language of the speaker;
 - email (character varying): contains the speakers's email address;
 - gender (character varying): contains the speaker's gender;
 - age (smallint): contains the age of the speaker;

- comments (text): contains comments regarding the speaker's voice;
- created_date (date): contains the date at which the speaker was created.
- users:
 - user_id (integer): contains the unique identifier of the user;
 - first_name (character varying): contains the first name of the user;
 - last_name (character varying): contains the last name of the user;
 - user_name (character varying): contains the user name;
 - email (character varying): contains the user's email;
 - password (character varying): contains the password to the user's account;
 - can_login (boolean): contains the status regarding the ability to login;
 - can_listen (boolean): contains the status regarding the ability to listen to recordings;
 - can_modify (boolean): contains the status regarding the ability to modify (replace) an existing recording;
 - is_admin (boolean): contains the status regarding whether or not the user has admin rights.

The relation between these tables is presented in the following figure:

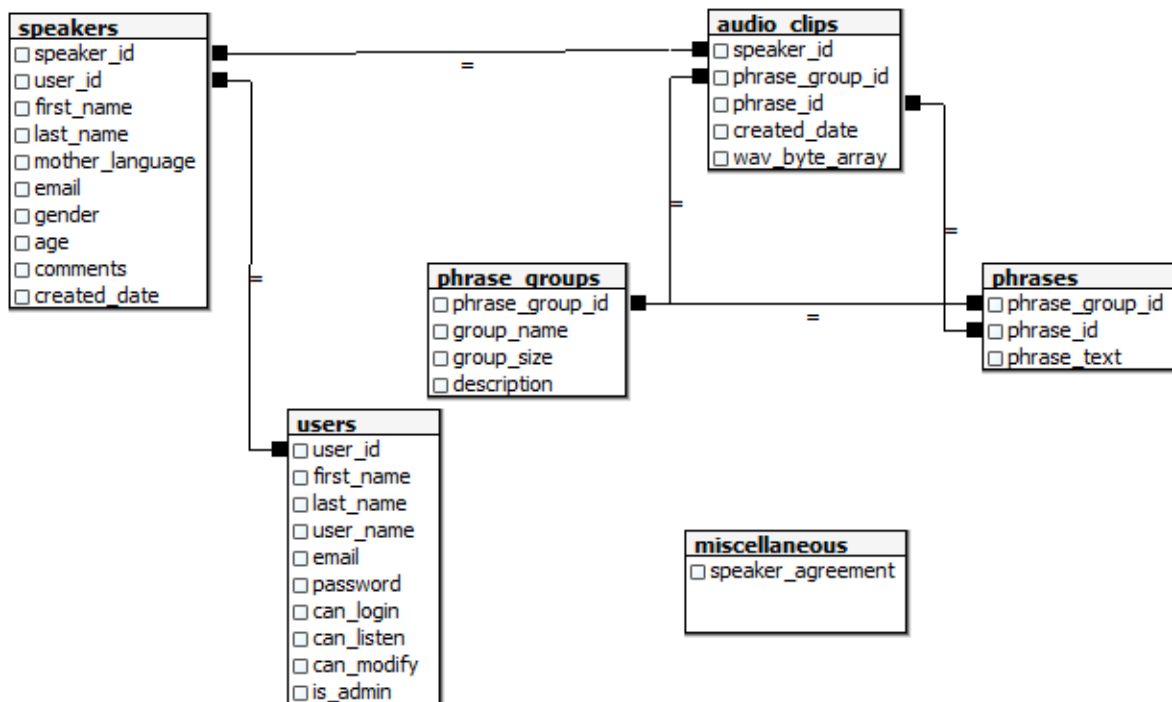


Figure 4.6 Relations between tables in the database

4.3 DATABASE STORED PROCEDURES

The stored procedures are implemented by executing a SQL script. The 18 stored procedures can then be found in the pgAdmin database in the Functions list.

It can be seen that some stored procedures require one or more inputs. The types of these inputs are the ones in brackets. The stored procedures will have various return types depending on their functionality. All the stored procedures are called from the server using the `Connection.prepareCall` function. Afterwards, using `CallableStatement` we may set the input and output parameters.

The following list presents each function:

- `check_recording_status`:
 - Input: speaker id, phrase group id;
 - Verifies which phrases from a certain phrase group have been recorded for a speaker and returns a list of the ids of the phrases which have been recorded;
- `delete_speaker_for_userid`:
 - Input: user id, speaker id;
 - Deletes a speaker from the speakers table if the user requesting this action has admin rights;
- `delete_user`:
 - Input: the id of the user who requests this action, the id of the user to be deleted;
 - Deletes a user from the users table if the user requesting this action has admin rights;
- `get_all_phrase_groups`:
 - Input: none;
 - Returns a list of all the phrase groups including information about them;
- `get_all_speakers`:
 - Input: none;
 - Returns a list of all the speakers including information about them;
- `get_all_users`:
 - Input: user id;
 - Returns a list of all the users and information about them if the user requesting this action has admin rights;
- `get_audio_clip_wav_data`:
 - Input: user id, speaker id, phrase group id, phrase id;
 - Checks if the respective user has the right to listen to his/her recordings and returns the audio clip corresponding to the speaker, phrase group and phrase as a sequence of bytes;
- `get_phrase_groups`:
 - Input: user id;
 - Returns a list of phrase groups and their information ordered by name;
- `get_phrases_for_phrase_group`:
 - Input: phrase group id;
 - Returns a list of all phrases which belong to the phrase group whose id is given as input;
- `get_speaker_agreement`:
 - Input: none;
 - Returns the speaker agreement;
- `get_speakers_for_user_id`:

- Input: user id;
- Returns a list of speakers including information about them which belong to a certain user;
- insert_speaker_for_user_id:
 - Input: user id, first name, last name, mother language, email, gender, age, comments;
 - Creates a new speaker in the speakers table if there is no other already existing speaker with the same first name and last name;
- insert_user:
 - Input: user id, first name, last name, user name, email, password, can login, can listen, can modify, is admin;
 - Checks if the user requesting this action has admin rights and creates a new user in the users table if there is no other existing user with the same user name or email;
- login:
 - Input: user name, password;
 - If the credentials are correct and if the user has the right to login, it returns the respective users information;
- modify_speaker_for_user_id:
 - Input: user id, speaker id, first name, last name, mother language, email, gender, age, comments;
 - Modifies an existing speaker if the new first name and last name are not the same with the ones of another speaker;
- modify_user:
 - Input: id of the user making the request, id of the user to be modified, first name, last name, user name, email, password, can login, can listen, can modify, is admin;
 - If the user requesting this action has admin rights, it modifies an existing user if the new user name and email are not the same with the ones of another user;
- put_user_agreement:
 - Input: user id, agreement;
 - If the user has admin rights, it updates the speaker agreement;
- upload_audio_clip:
 - Input: user id, speaker id, phrase group id, phrase id, wav bytes;
 - Inserts the wav file as a sequence of bytes and the associated speaker id, phrase group id and phrase id in the audio_clips table. If the recording already exists, the procedure first verifies if the user has the right to modify an existing audio clip and only then proceeds to the replacement of the old recording;

4.4 APPLICATION BACK-END: ENDPOINTS

A web service endpoint is a web address (URL) where the clients can gain a specific service. Thus, endpoints are a very important part in the process of developing a web service. The following list presents the application endpoints:

POST login

- Allows or denies a user access to the web site
- Parameters: username and password
- Response: token or error message

Endpoints for normal users :

- **GET** speakers
 - Returns the list of existing speakers for a particular user
 - Parameters: none
 - Response: [{id, first_name, last_name, mother_language, email, gender, age, comments}]
- **POST** speakers
 - Creates a new speaker
 - Parameters: [{first_name, last_name, mother_language, email, gender, age, comments}]
 - Response: none
- **PUT** speakers
 - Updates the information for an existing speaker
 - Parameters: [{speaker_id, first_name, last_name, mother_language, email, gender, age, comments}]
 - Response: none
- **GET** phrase-groups
 - Returns the list of existing phrase-groups
 - Parameters: none
 - Response: [{id, name, size, description}]
- **GET** speakers/{speaker_id}/phrase-groups/{phrase_group_id}/phrases
 - Returns the phrases in a specific phrase-group and the recording status for each phrase for a specific speaker
 - Parameters: none
 - Response: [{id, text, recording_status}]
- **GET** speakers/{speaker_id}/phrase-groups/{phrase_group_id}/phrase/{phrase_id}/audio
 - Returns an audio recording
 - Parameters: none
 - Response: {audio_clip_base64}
- **PUT** speakers/{speaker_id}/phrase-groups/{phrase_group_id}/phrase/{phrase_id}/audio
 - Creates a new audio recording or updates the existing one
 - Parameters: {audio_clip}
 - Response: none
- **GET** user-agreement
 - Returns the user agreement
 - Parameters: none
 - Response: html user agreement
- **POST** audio-verification
 - Verifies the audio signal from the microphone
 - Parameters: {silence_audio_clip, test_audio_clip}
 - Response: OK or error message
- **GET** noise
 - Checks if the received token is associated with a noise power
 - Parameters: none
 - Response: none

Endpoints for admins:

- **GET** users
 - Returns the list of existing users
 - Parameters: none

- Response: [{id, first_name, last_name, user_name, email, password, can_login, can_listen, can_modify}]
- **POST** users
 - Creates a new user
 - Parameters: {first_name, last_name, user_name, email, password, can_login, can_listen, can_modify}
 - Response: none
- **PUT** users/{user_id}
 - Updates information regarding an existing user
 - Parameters: {first_name, last_name, user_name, email, password, can_login, can_listen, can_modify}
 - Response: none
- **DELETE** users/{user_id}
 - Deletes an existing user
 - Parameters: none
 - Response: none
- **DELETE** users/{user_id}/speakers/{speaker_id}
 - Deletes an existing speaker
 - Parameters: none
 - Response: none
- **PUT** user-agreement
 - Updates the user agreement
 - Parameters: {agreement_text}
 - Response: none

4.5 APPLICATION BACK-END: CLASS ARCHITECTURE

The architecture of the web service is divided into four packages:

- Model
- Serializers
- Service
- Web

In the „model” package we have the classes which will be used to create the objects necessary for the interaction with the web application(frontend) as well as with the database. The following classes can be found in the model package: AudioClips, AudioClipsPK, NoiseData, PhraseGroups, Phrases, PhrasesPK, Speakers, SpeakersPK, UserAuth, Users. The classes AudioClips, PhraseGroups, Phrases, Speakers and Users are the ones which correspond with the fields in the tables from the database. The classes AudioClipsPK, PhrasesPK and SpeakersPK contain the connections between the previous classes. For example, an AudioClips object will contain an AudioClipsPK which contains the fields speakerId, phraseGroupId and phraseId thus giving us the information about which speaker made the recording, from which phrase group and which phrase. Similarly, SpeakersPK found in a Speakers object contains the variables speakerId and userId showing to which user the speaker belongs to. PhrasesPK found in Phrases contains phraseGroupId and phraseId thus tying a phrase to a phrase group. The UserAuth class contains the variables userName and password and is used in the login process. The NoiseData class is used during the initial audio verification and during recordings. It contains the variables messageId, message and SNR, offering full information about the cause of error.

In the „serializers” package we can find the PhrasesSerializer and SpeakerSerializer classes. These classes are used to rewrite the Phrases, respectively Speakers JSONs sent to the client. The rewriting consists in sending only certain fields from the JSON and not all of them. Thus, these classes could be called JSON custom serializers.

The „service” package contains the following classes: AudioVerification, CORSFilter, CORSHeaders, DatabaseInterface, FilterReg, Finals and SpeedService. The most important classes in this package are DatabaseInterface and SpeedService. DatabaseInterface contains the functions necessary for the interaction with the database such as performing the connection with the database and making calls on the stored procedures. SpeedService is an intermediary class, it makes the connection between the DatabaseInterface class and the classes in the „web” package, also performing various operations such as adding the JSON custom serializer. SpeedService is also a Singleton class which means that only one instance of this class will exist at any time. The SpeedService class loads the properties from the configuration file in which we can find the following fields: environment, databaseURL, username, password, token_secret, token_exp_time, minimumSNRAllowed and cutTimeInMilliseconds. Another important function of the SpeedService class is creating and verifying tokens. Using the secret from the configuration file, the logging function in the SpeedService class will sign the following claims: userID, exp (the time at which the token expires; this field is the sum of the current time in milliseconds and the token expiration time in the configuration file) and is Admin. The SpeedService class also contains a HashMap called „noisePowers” which contains token -background noise power pairs. Each time a user sends a recording, the background noise associated with his token will be used for the audio verification. In the case of an expired token, the user will have to log in again to receive a new token and make another background noise recording which will be associated with it. The main functions of the AudioVerification class is to compute the noise powers and to process the audio signal, among other tasks such as to cut the front and back ends of the recordings necessary to eliminate the clicking sound at the beginning and at the end of the recording. The function of processing the audio signal will return a NoiseData object. Then the SpeedService class will take that object and extract the messageId in order to determine whether the audio clip has been properly recorded or if there was some irregularity. Thus the NoiseData object will update with a message depending on the messageId. Both messageId and message variables are extracted from the Finals class. Lastly, the CORSFilter, FilterReg and CorsHeaders classes are used to implement and register the CORS filter.

The „web” package includes the classes: WEBAudioClips, WEBPhraseGroups, WEBSpeakers and WEBUsers. These are the classes in which we can find the endpoints. The functions found in these classes will also have to call the JWTVerify function in SpeedService in order to check if the token used to access that specific resource is valid, otherwise it returns an error message. The only function which makes an exception is the login function which checks if the credentials received are found in the database and returns a token if that is the case. The following list shows the endpoints found in each class.

- WEBAudioClips
 - PUT audio
 - GET audio
 - PUT audio-verification
 - GET noise
- WEBPhraseGroups
 - GET phrase-groups
 - GET phrases

- WEBSpeakers
 - GET speakers
 - POST speakers
 - PUT speakers
 - DELETE speakers
- WEBUsers
 - POST login
 - GET users
 - POST users
 - PUT users
 - DELETE users
 - GET user_agreement
 - PUT user_agreement

4.6 DATAFLOW EXAMPLE

In this subchapter, a simple example of the interaction between the 3 components, namely front-end, back-end and database will be presented. The case of creating a new speaker will be shown:

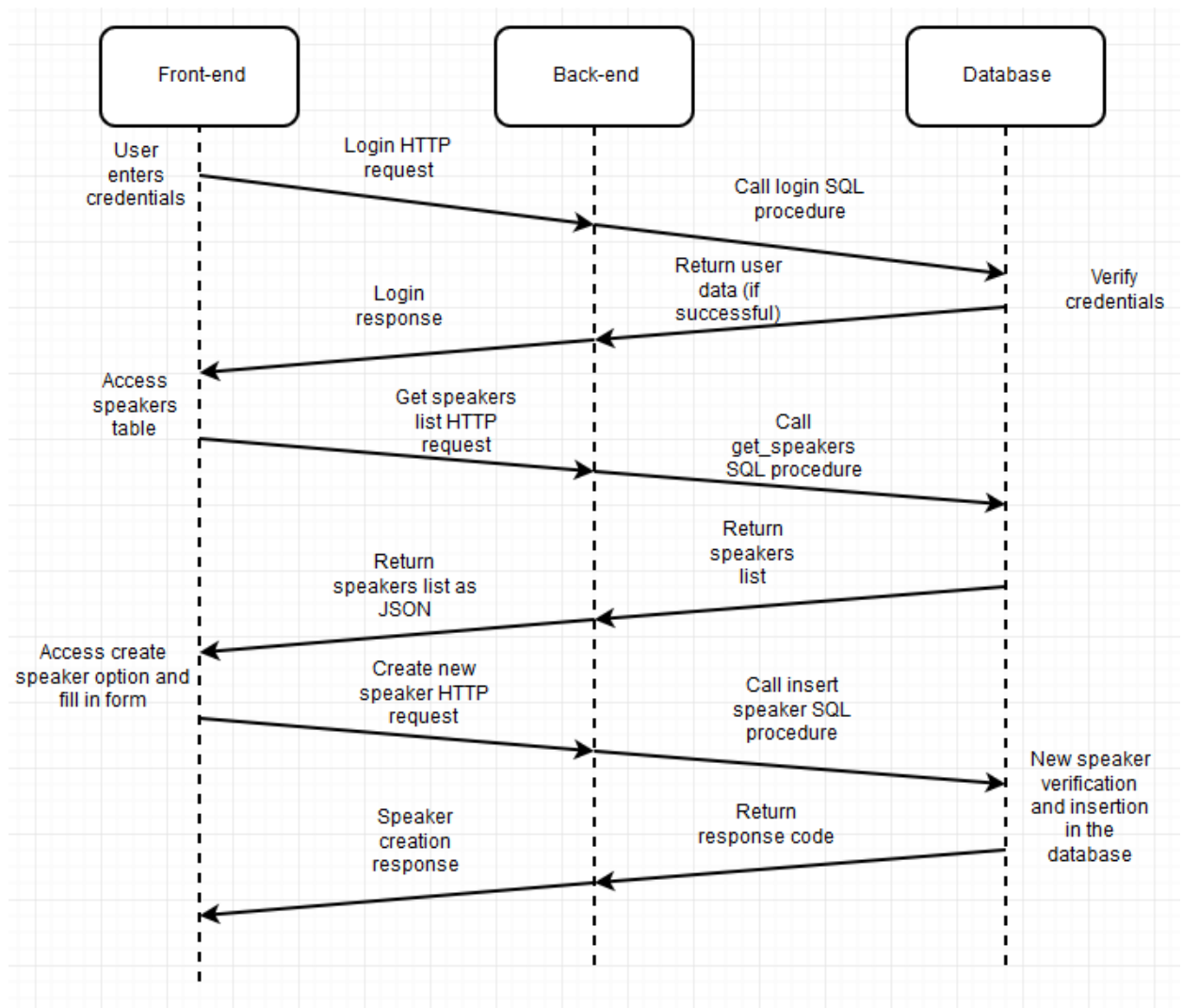


Figure 4.7 Dataflow example

The upper figure is explained through the following steps:

- After accessing the login page, the user enters his credentials and sends the data through the login endpoint via a HTTP POST request. The back-end will then send this information further by calling the login stored procedure of the database.
- After verifying if the username and password match any user in the users table, the database will return, if successful, the data for the respective user which will be used by the back-end in creating the JWT. The user id and admin rights will be stored in the claims of the JWT which will be sent back to the user and any subsequent requests from this user will be made using this JWT. If there were no matches in the database, the user will be prompted with an error message stating that either the username or password introduced, or both were wrong.
- The user now being able to access the menu of the application will now proceed to the speakers tab, from where the GET speakers endpoint will be accessed and a HTTP GET request will be sent. The back-end will call the `get_speaker_for_user_id` procedure.
- The database will return the speakers list for the respective user to the back-end which will send it back to the user as a list of JSON objects.
- Once in the speakers tab, the user can access the create speaker window. After completing the form, the information about the speaker will be sent via a HTTP POST request. The back-end will send the data further by calling the `insert_speaker_for_user_id` stored procedure.

- Upon receiving the data, the database will check if there are other speakers which have the same first name and last name with the received speaker. If that is not the case, the speaker will be introduced in the speakers table in the database and a response code will be sent back to the back-end stating whether the operation was successful, there is an existing speaker with the same first name and last name or some other error has occurred. Depending on the case, the user will be sent back to the updated speakers list in the case of a successful event or will receive an error message.

4.7 IMPLEMENTATION ON A VM

A VM is a software which allows one to emulate a OS and imitate dedicated hardware on a host OS. This software is just like another program. For example, we can run a Linux OS on a Windows OS.

In order to implement the web service on a VM, there are a couple of steps that must be taken. Firstly, we need to install the programs necessary to run the service, meaning Java, Apache Tomcat, Git and Maven. Apache Maven represents a tool used for building and managing any Java-based project. The Apache Tomcat server will require some configuration processes such as: setting execution rights, port configuration, setting user and password for the control panel. Git will be used to clone the repository on the VM and Maven will build the backend and generate the WAR file which will be used by the Apache server. A WAR file is actually a JAR file which distributes resources such as JSP, Java Classes XML files, static web pages in order to build up the web application. In short it is a single file which has all of the applications files bundled inside. A JAR file represents a single file in which we have archived many Java classes and associated metadata and resources. The WAR file will be installed in Apache Tomcat in order to deploy the backend. The backend can be deployed either through the Manager App of the Apache Tomcat GUI or through the command line. In the Manager App, one must simply browse for the file and enter the Context Path (the app will be found at {base URL}+Context Path). The alternative consists in moving the WAR file in Apache Tomcats webapp folder.

4.8 DEVELOPMENT METHODOLOGY

In the beginning of development, the technologies which were to be used were chosen such as the programming language and the IDE. Afterwards, the endpoints were drawn out, which during development had encountered some slight changes. Once most of the work on endpoints was achieved, the important matter of security and authentication was discussed. Thus, JWT and CORS were implemented. Once the application was beginning to take shape, several tests on recordings were made. These included speaking too loud or not at all to see if the application returned an error message, thus making sure the application behaved as intended.

Just like any real-life application, this web service might encounter some bugs, glitches or there might be room for some improvements regarding some functionalities. Thus, a production version was installed on a new VM, different from the one used for development. The development version will often use different parameters for making various tests. In this regard, the configuration file has the environment property which can be equal to “dev” or to “prod”, making the switch between the two environments more easily, instead of modifying each field.

CHAPTER 5

CONCLUSIONS

5.1 GENERAL CONCLUSIONS

The Speed Recording Web application is currently online on the Speed Laboratory site. During development, there have been some issues that required solving. One of the most notable would be: during the elimination of the front and back ends of the audio clips, the header of the file was also removed thus making the file unusable. This matter was solved by keeping the original header of the file and reattaching it after performing the cutting operation.

This thesis presents the successive steps which were employed by the author in order to create the web service and set up the database. After describing the motivation and main objective in Chapter 1 and the theoretical aspects regarding software technologies, development tools used and security measures in Chapter 2 and 3, Chapter 4 presents the application description, class and stored procedures implementation and functionality, implementation and development.

5.2 PERSONAL CONTRIBUTIONS

The personal contributions of the author of this thesis can be found in Chapter 4 and can be summarized as follows:

- a) Creation and implementation of new stored procedures to provide new features;
- b) Updating existing procedures to facilitate the interaction between the back-end and the database;

- c) Creation and execution of scripts that modify the database (update the users table with a `is_admin` column, generate a drop list for old procedures, create new procedures);
- d) Creation of back-end service based on REST architecture;
- e) Implementation of endpoints;
- f) Creation of an authentication system based on JWT;
- g) Implementation of CORS filter;
- h) Deployment of the back-end on a VM.

5.3 FUTURE WORK

Regarding future improvements, the primary concern would be to implement a logging system to keep track of some minor glitches that seem to have occurred in the service from time to time to solve the issue that causes them. Another update consists in implementing new endpoints which allows admin users to create new phrase groups and the phrases which belong to those phrase groups or update existing ones by editing phrases in a phrase group, adding new ones or deleting them. Also, a new feature that we would like to include in our application would be a way to download wav files from the database. For example, download all the recordings of a certain speaker or all the recordings of a certain phrase or phrase group.

REFERENCES

- Java ([https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language)))
- Java EE (https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition)
- Java SE and EE comparison (<http://docs.oracle.com/javase/6/firstcup/doc/gkhoy.html>)
- PostgreSQL (<https://www.postgresql.org/about/>)
- JDBC (<http://www.oracle.com/technetwork/java/javase/jdbc/index.html>)
- JDBC (<https://docs.oracle.com/javase/tutorial/jdbc/overview/>)
- Netbeans (<https://netbeans.org/features/index.html>)
- VCS (https://www.tutorialspoint.com/svn/svn_basic_concepts.htm)
- Git (<https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>)
- Git (<https://www.atlassian.com/git/tutorials/what-is-git>)
- Apache Tomcat (<http://searchmicroservices.techtarget.com/definition/Apache>)
- Apache Tomcat (<http://tomcat.apache.org/>)
- REST (<https://www.tutorialspoint.com/restful/>)
- XML (https://www.tutorialspoint.com/xml/xml_overview.htm)
- JSON (<http://www.json.org/>)
- JSON (<https://developers.squarespace.com/what-is-json/>)
- CORS (https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)
- CORS (<https://spring.io/understanding/CORS>)
- Cross-site scripting (<https://www.acunetix.com/websitesecurity/cross-site-scripting/>)
- Cross-site scripting (https://en.wikipedia.org/wiki/Cross-site_scripting)
- Same origin policy (https://www.w3.org/Security/wiki/Same_Origin_Policy)
- JWT (<https://jwt.io/introduction/>)
- Oauth (<https://oauth.net/articles/authentication/>)
- Oauth and JWT (<http://www.seedbox.com/en/blog/2015/06/05/oauth-2-vs-json-web-tokens-comment-secrifier-un-api/>)
- Horia Cucu, AVR Laboratory Guide

Eric Freeman & Elisabeth Robson, “Head First Design Patterns”, 2004, O'Reilly Media

Kathy Sierra, Bert Bates, “SCJP Sun Certified Programmer for Java 6 Study Guide”, 2008,
McGraw Hill Professional