



UNIVERSITAT DE
BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

**Facultat de Matemàtiques i Informàtica
Universitat de Barcelona**

**RECORDING CONTEXTS:
CONTEXTUALIZED AUDIOS
GATHERING TOOL**

Jose Manuel Cano Calvente

Director: Simone Balocco
Realitzat a: Departament de
Matemàtiques i
Informàtica

Barcelona, 27 de juny de 2019

Abstract

The following project called “Recording Contexts: Contextualized Audios Gathering Tool” is a web application for the gathering of contextualized audio recordings.

The main goal of the application is to collect audio recordings from different users through repetition and context-complete exercises previously created inside the application.

Recording Contexts has been developed to help the Phonetics Laboratory of the University of Barcelona expand their research on acoustic data. Furthermore, it provides a valuation to those users that carry out the exercises thanks to the execution of a Python process that compares their answer with a previously recorded audio example.

The application has been developed using the PHP framework Symfony4 to create the back-end and a group of front-end technologies: CSS, HTML and JavaScript. Moreover, the best methodologies and state-of-the-art web technologies have been applied, always thinking in terms of scalability and modularity so the application can adapt to future necessities and improvements.

This document outlines the process of design, analysis, planning and architecture of the proposed application.

Resum

El projecte documentat a continuació anomenat "Recording Contexts: Contextualized Audios Gathering Tool" és una aplicació web destinada a la recollida de gravacions d'àudio contextualitzades.

La principal funció de l'aplicació és la recollida de gravacions d'àudio de diferents usuaris mitjançant exercicis de repetició i de completat de contextos prèviament creats en l'aplicació.

Recording Contexts ha sigut desenvolupada per ajudar al Laboratori de Fonètica de la Universitat de Barcelona a expandir la seva investigació sobre dades acústiques. Proporciona a més una valoració als usuaris que realitzin els exercicis gràcies a l'execució d'un procés implementat en Python que compara la seva resposta amb un àudio d'exemple gravat prèviament.

L'aplicació ha sigut desenvolupada fent servir el framework PHP Symfony4 per crear el "back-end" i un conjunt de tecnologies "front-end" com son CSS, HTML y JavaScript. A més, s'han aplicat les metodologies i tècniques de desenvolupament web més modernes, pensant sempre en termes d'escalabilitat i modularitat perquè l'aplicació es pugui adaptar a les necessitats futures.

En el present document es detalla el procés de disseny, anàlisi, planificació i arquitectura de l'aplicació.

Resumen

El proyecto documentado a continuación llamado “Recording Contexts: Contextualized Audios Gathering Tool” es una aplicación web destinada a la recolección de grabaciones de audio contextualizadas.

La principal función de la aplicación es la recogida de grabaciones de audio de diferentes usuarios a través de ejercicios de repetición y de completado de contextos previamente creados en la aplicación.

Recording Contexts ha sido desarrollada para ayudar al Laboratorio de Fonética de la Universidad de Barcelona a expandir su investigación de datos acústicos. Proporciona además una valoración a los usuarios que realicen los ejercicios gracias a la ejecución de un proceso implementado en Python que compara su respuesta con un audio de ejemplo grabado previamente.

La aplicación ha sido desarrollada utilizando el framework PHP Symfony4 para el crear el “back-end” y un conjunto de tecnologías “front-end” como son CSS, HTML y JavaScript. Además, se han aplicado las metodologías y técnicas de desarrollo web más modernas, pensando siempre en términos de escalabilidad y modularidad para que la aplicación se pueda adaptar a las necesidades futuras.

En el presente documento se detalla el proceso de diseño, análisis, planificación y arquitectura de la aplicación.

Index

1. INTRODUCTION	1
2. PLANNING	3
3. REQUEST ANALYSIS.....	6
3.1 FUNCTIONAL AND NON-FUNCTIONAL REQUESTS	6
3.1.1 Functionalities.....	6
3.1.1.1 Admin Interface	6
3.1.2 Non-functional requests	9
4. TECHNOLOGY STACK.....	10
4.1 APACHE	10
4.2 POSTGRESQL.....	10
4.3 PHP	11
4.4 COMPOSER.....	11
4.5 HTML, CSS & JAVASCRIPT.	11
4.5.1 Bootstrap Material Design & SCSS	12
4.5.2 JQuery.....	12
4.5.3 TinyMCE.....	13
4.5.4 ChartJS	13
4.5.5 Custom Recording JavaScript.....	14
4.6 GITHUB	15
5. SYMFONY4	16
5.1 WHAT IS A FRAMEWORK?	16
5.2 WHY SYMFONY?	17
5.3 COMPONENTS.....	17
Doctrine	24
6. DATA MODEL.....	28
6.1 CLASS DIAGRAM.....	28
6.2 DATABASE SCHEMA	30
7. RECORDING CONTEXTS	31
7.1 PROJECT STRUCTURE	31
7.2 APPLICATION STRUCTURE.....	33
7.2.1 Controllers	33
7.2.2 Entities	33
7.3 PITCH COMPARE.....	34
8. VALIDATION AND PERFORMANCE TESTS	36
8.1 PHPUNIT.....	36
9. DEPLOYMENT	37
9.1 SERVER REQUIREMENTS.....	37
9.2 HOSTING OPTIONS.....	38
9.2.1 Custom Server.....	38
9.2.2 OpenShift.....	39
9.2.3 Amazon Web Services.....	39
9.3 AWS SETUP	39

10. CONCLUSIONS.....	42
10.1 FUTURE IMPROVEMENTS.....	42
10.1.1 Minify CSS & JavaScript content	42
10.1.2 Use OpenShift hosting	42
10.1.3 Own server.....	43
10.1.4 Web Service to compare audios.	43
11. BIBLIOGRAPHY	45
12. ANNEXES	47
12.1 USER MANUAL	47
12.1.1 Log in as Administrator.....	47
12.1.2 Admin/Superadmin Interface	48
12.1.3 Experiment creation/edition.....	48
12.1.4 Record/upload an audio example.....	50
12.1.5 Edit Landing Page.	50
12.1.6 Admins management	52
12.1.7 Answer an experiment.....	53
12.2 USE CASES.....	57
12.2.1 ADMIN USE CASES	57
12.2.2 SUPERADMIN USE CASES	62
12.2.3 USER USE CASES	64

1. Introduction

During the last decades, phonetics (which is the branch of linguistics that studies the sounds of speech) has largely benefited from the technological innovation implied by the so-called Digital Revolution [4], insofar as data collection (mostly recording of audio samples) and of data analysis (basically acoustic analysis of soundwaves) since the Nineties are carried out by means of electronic devices (recorders in the first case, software for acoustic analysis in the second case).

However, phonetics has not benefited from the so-called Internet Revolution [5], insofar as the digital techniques of data collection and analysis are still largely implemented offline. Recordings, for example, are usually carried out in laboratories located in research centers or in universities, while data analysis is done by means of software locally installed on personal computers.

To the best of our knowledge, to date, there are only a few tools that allow the **collection of acoustic data** for phonetic purposes on the internet, and there is no online tool capable of carrying out acoustic data analysis (aimed either at basic research or at applied research). Among the tools that allow the collection of acoustic data for phonetic purposes, one can mention Lingua Libre [6], which is an audio recording tool as well as a sound library designed by Wikimedians to improve several Wikimedia projects (such as Wiktionaries, Wikipedias, Wikimedia Commons and Wikidata). LinguaLibre is an open audio recording platform and web application to ease recording of wordlists into audio files. It was designed to ease the creation of consistent datasets of audio files and allows users to create their own recording campaigns. LinguaLibre is probably the most flexible online audio recording platform that exists, but it has important limitations as far as data elicitation techniques are concerned. It has been conceived, in fact, to record speakers *reading* a word or a short text. This means that it can be used only to obtain audio samples of non-spontaneous speech. More spontaneous acoustic data are obtained in phonetics by means of other techniques, like Discourse Completion Tasks [3], which are more complex and not compatible with the structure of LinguaLibre. For this reason, one of the objectives of this project is creating a platform that can allow greater flexibility as far as data elicitation is concerned.

The **online analysis of acoustic data** (specifically applied to language teaching, which is a common field of applied research in phonetics) is at a similar stage. Listen and repeat exercises have been used since 1980 in foreign language teaching. In the 20th century, these exercises were conducted in the classroom and the teacher gave feedback to the students stating whether their pronunciation was correct or not. As the internet has become more popular for second language teaching, this kind of exercises has been implemented in webpages and apps. However, without a teacher for giving feedback, the student has lost the opportunity of correcting the wrong pronunciation. This shortcoming has been solved for general pronunciation exercises (learning to produce a specific sound like “rr” in Spanish) by using general speech recognition systems. In other words, if Siri cannot understand you, you are saying it wrong. However, speech recognition systems (with meritorious exceptions like Google talk), do not recognize the intonation of a sentence as a linguistic feature. For example, Siri or Cortana cannot differentiate between “Llueve.” and “¿Llueve?”. And therefore, such systems cannot be used to test listen and repeat exercises that aim at teaching intonation. The second objective of this project is, thus, to create a webpage that solves this problem by implementing a system that analyses (i.e that compares) the audio recorded by the student and the audio used as a model.

For doing so it follows the following approach:

1. Preprocessing: Trims the recordings.
2. Extract the F0 of both signals.

The main acoustic correlation of intonation is the fundamental frequency of the speech signal. In this approach, F0 is computed by using the RAPT algorithm [1].

3. Compares the pitch contours using DTW [2].
4. Interpolates the result.

F0 is not continuous in speech, it only appears when vocal folds are vibrating, however, the human brain is able to reconstruct this deficient signal in what we call “pitch” or “melody”. In order to imitate the perceived pitch, F0 signal must be interpolated in those samples of the signal where there is no F0.

2. Planning

The project development was planned in 3 main phases:

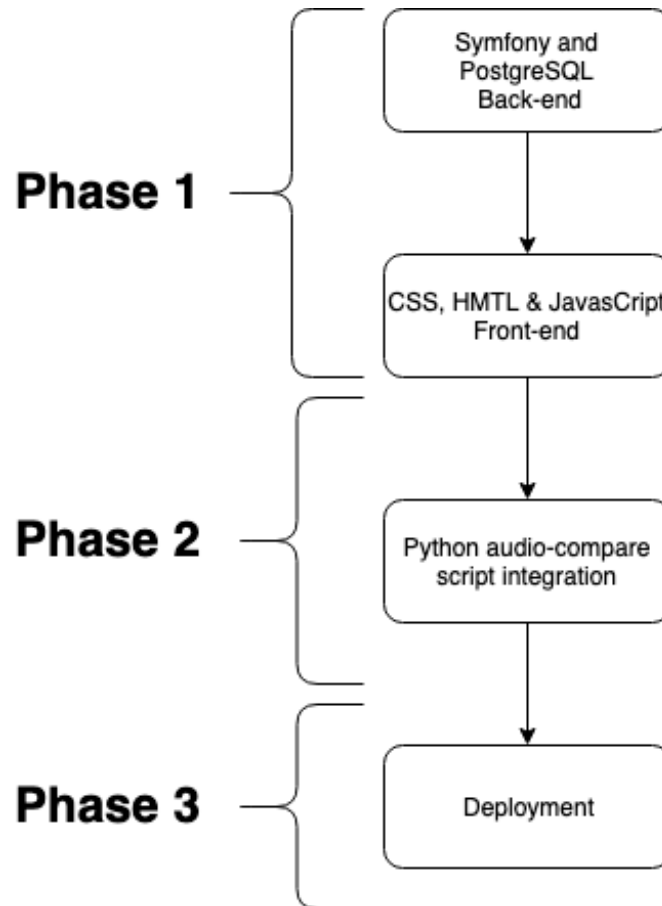


Figure 1 Block diagram with project's phases.

The first phase was focused on developing the 2 user interfaces, one for administrators and one for final users. This included the creation of experiments, the administrator system and the recording of audio through the web browser.

Phase 2 consisted of the analysis in real time of audio recordings uploaded by final users when they answered the context of an experiment. The python script that is used to make the comparison of audio recordings was provided by the Phonetics Laboratory.

The last phase was the deployment of the application to a server and the documentation of the project. A study of several options to make the deployment was carried out in order to choose the option that fitted best to our requirements.

Breakdown of milestones and tasks:

Phase 1:

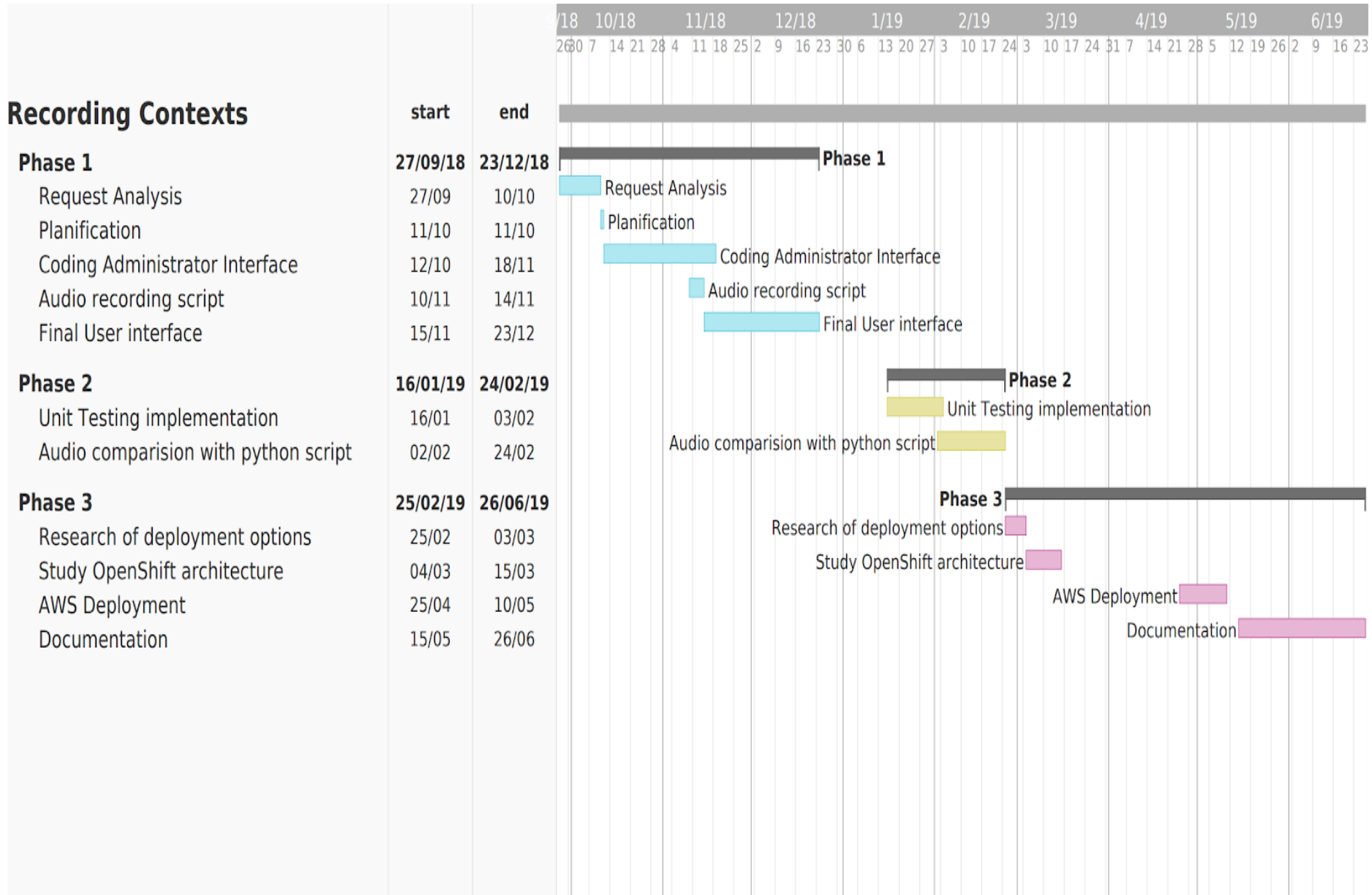
- Request analysis and functionalities specification (27/09/2018 - 10/10/2018)
- Planification (11/10/2018)
- Test of python script launching from Symfony Controller (12/10/2018 - 18/10-2018)
- Coding Administrator interface (19/10/2018 - 9/11/2018)
- Audio recording script (10/11/2018 - 14/11/2018)
- Final User interface (15/11/2018 - 23/12/2018)

Phase 2:

- Unit Testing implementation (16/01/2019 - 03/02/2019)
- Audio comparision with python script (04/02/2019- 24/02/2019)

Phase 3:

- Research of options to make the deployment (25/02/2019 - 03/03/2019)
- Study OpenShift server architecture (04/03/2019 - 15/03/2019)
- AWS Deployment (25/04/2019 - 10/05/2019)
- Documentation (15/05/2019 - 26/06/2019)



3. Request Analysis

The main functionality requested at the beginning of this project was to have a web application to gather data (audio recordings in .wav or .aiff format) from people who live in different locations through, what the lab calls “Experiments”.

These experiments contain “contexts” that can be classified into 2 types:

- Repetition of pre-recorded sentences.
- Discourse Completion Tests (DTC) [7].

From this requirement, it could be established that the web application should handle 2 types of users:

- **Administrators:** Registered users with a username and a password, with the ability to create Experiments. From now on Admins.
- **Final Users:** Not registered users that are invited to complete experiments created by admins. From now on Users.

3.1 Functional and non-functional requests

3.1.1 Functionalities

After the first meeting with the Phonetics Laboratory, we defined the main functionalities that the web application should have. Later on, these functionalities were tuned and redefined and some others came up as the project progressed.

3.1.1.1 Admin Interface

A set of pages to allow Admins create, edit and manage Experiments. When creating or editing an Experiment, admins should be able to choose the type for each context.

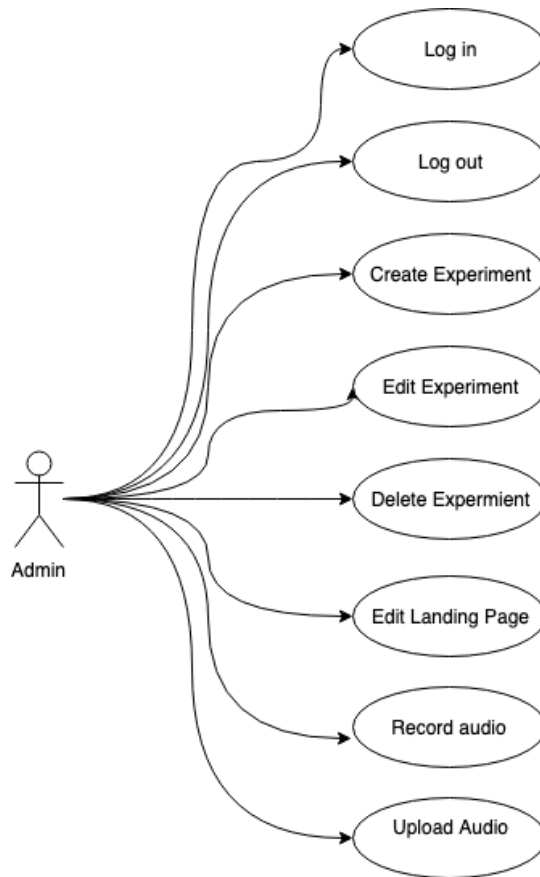


Figure 2: Administrator use case diagram.

It was defined that the application should have the possibility to handle several Administrators, for that reason a Superadmin profile was mandatory. This is a unique user inside the application and has the power to create and edit Admins, as well as edit the Terms of Use of the platform. Being a super-class of the base Admin, it can still access all the functionalities shown in the figure above.

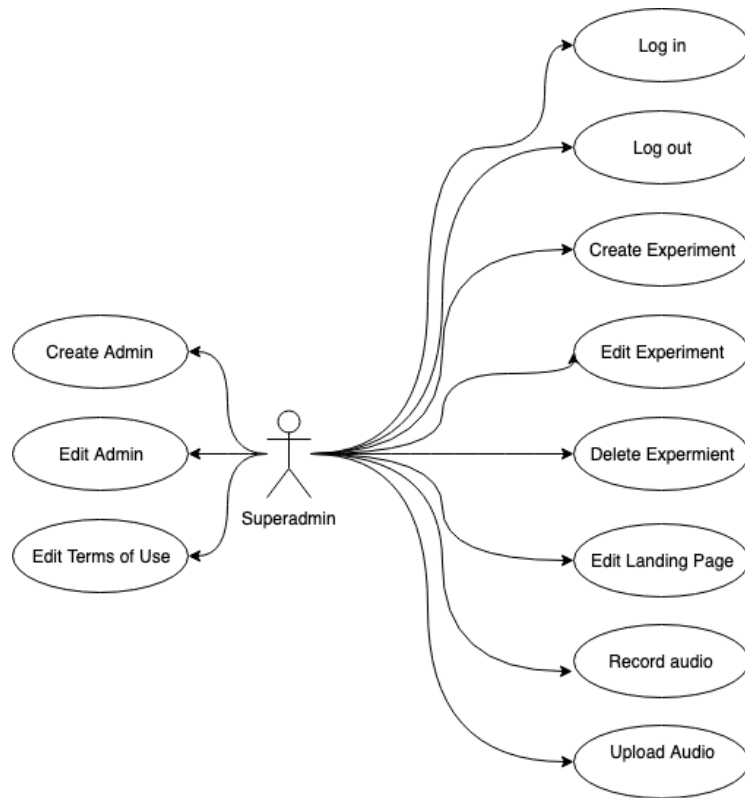


Figure 3: Superadmin use case diagram.

3.1.1.2 User Interface

The entry point of final users to Recording Contexts. They would receive a link to an experiment and at the very beginning of this page, an explanation of what they are going to be examined for should appear.

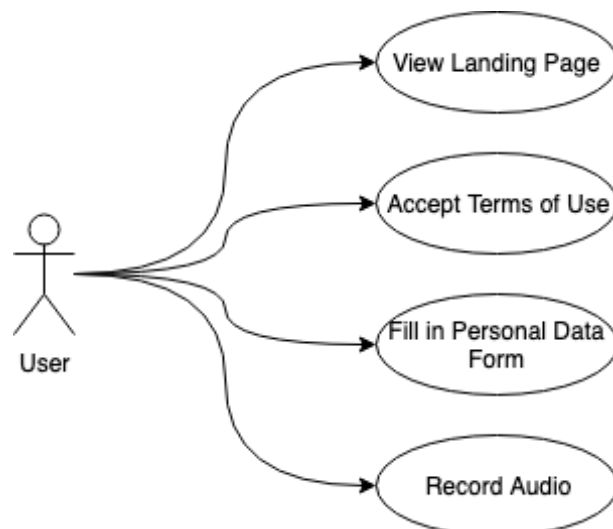


Figure 4: User use case diagram.

3.1.2 Non-functional requests

3.1.2.1 Accessibility

Due to the disparity of ways in which the internet can be accessed nowadays with the rise of smartphones and tablets, both admins and users are not forced to use a computer in order to access Recording Contexts. This obligates the developer to offer content that adapts to different screen sizes, therefore it is necessary to implement a responsive design.

3.1.2.2 Speed

Interaction between user and application should be smooth and even more so if input from users is requested.

3.1.2.3 Scalability

Being the first iteration of a project with a lot of possibilities to be extended in the future, the application must be scalable, not only in functionalities but also in the server-side requests handling if it is expected to have a lot of traffic.

4. Technology Stack

Recording context has been developed under a MAPP architecture. This is formed by the stack of MacOSX, Apache, PostgreSQL and PHP. Later on, it was deployed in a LAPP server (Linux, Apache, PostgreSQL and PHP). Thanks to MacOS being itself a UNIX OS, developing a web application which will be later deployed in a Linux server has not been any problem at all.

4.1 Apache

It is an open-source HTTP server [8] for modern operating systems including UNIX and Windows. Its goal is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. It runs on 63% of all web servers in the world [9] and many well-known web pages like paypal.com, bbc.com are using it as their main server. It is fast, reliable, secure and can be highly customized to meet the needs of many different environments by using extensions and modules for security, caching, URL rewriting, password authentication, and more.

Despite calling it a web server, Apache is not a physical server, but rather a software that runs on a server. Its job is to establish a connection between a server and the browsers of website visitors like Firefox, Google Chrome or Safari while delivering files back and forth between them in a client-server structure.

When a visitor wants to load a page, the browser sends a request to the server and Apache returns a response with all the requested files like text or images. The server and the client communicate through the HTTP protocol and Apache is responsible for the smooth and secure communication between the two machines.

4.2 PostgreSQL

PostgreSQL is a powerful, open source object-relational database that uses and extends the SQL language that runs on all major operating systems. It has earned a strong reputation for its proven architecture, reliability, data integrity, extensibility and the dedication of the open source community behind the software to consistently deliver innovative solutions.

It comes with many features aimed to help developers build applications, administrators to protect data integrity and manage data no matter how big or small the dataset. It is highly extensible, developers can, for example, define their own data types or build custom functions.

4.3 PHP

Is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It can be used on all major operating systems, including Linux, Windows, MacOs and many others, and offers support for most of the web servers today like Apache.

PHP is mainly focused on server-side scripting, so it can collect form data, generate dynamic page content, or send and receive cookies and it is not limited just to output HTML. It can output images, PDF files and even Flash movies which can be auto-generated and served as dynamic content.

One of the strongest and most significant features of PHP is its support for a wide range of databases. It provides database-specific extensions or abstraction layers like PDO (PHP Data Objects) which define a lightweight, consistent interface for accessing databases. For example, Recording Contexts uses PHP's pdo-pgsql to access the PostgreSQL database.

4.4 Composer

Composer [11] is a tool for dependency management in PHP. It allows developers to declare the libraries their project depends on and it manages (install/update) them. However, it is not a package manager in the same sense as Yum or Apt for Linux. It does deal with “packages” or libraries but it manages them on a per-project basis, installing them in a directory inside each project.

4.5 HTML, CSS & Javascript.

Nowadays there are many platforms which are able to interact with Web applications from different devices and all of them have different characteristics, which makes Web development a new challenge if it has to be developed from the ground. Thankfully, many developers foresaw how the system was shifting and developed modern open-source

HTML, CSS and JavaScript frameworks to help the developer community build web applications.

The aim of these libraries is to:

- Offer a base to develop responsive web pages which adapt to many common devices screen sizes, like smartphones or tablets.
- Fix rendering problems between different web browsers.
- Provide an accessible representation of HTML forms for any device.
- Reduce costs and speed up the creation of web pages applying generic solutions.
- Allow developers to extend this framework to solve their needs by offering helpful documentation and strong APIs.

Recording Contexts frontend has been built using some of these modern frameworks.

4.5.1 Bootstrap Material Design & SCSS

Bootstrap Material Design is the CSS framework chosen to create the frontend of Recording Contexts. It is an extension of the popular Bootstrap CSS framework, done by FezVrasta [12], which offers an overhaul of all functionalities provided by the original version 4 of the named library and adapts them to the standards of Google's Material Design [13]. This is a visual language that synthesizes the classic principles of good design with the innovation of technology and science that was introduced by Google in 2014. It uses more grid-based layouts, responsive animations and transitions, padding and depth effects such as lighting and shadows.

It is developed with SCSS which with a few small exceptions, is a superset of CSS. This means essentially that all valid CSS is valid SCSS as well. It provides an easy syntax that extends on traditional functionalities and allows developers to customize the style of web pages using variables, nested rules, mixins and functions which are later compiled into CSS stylesheets.

4.5.2 JQuery

A fast, small and feature-rich JavaScript library [14] that offers many functionalities to help developers with frontend development. It simplifies HTML document traversal and manipulation, event handling, animation and Ajax with an easy-to-use API that works across multiple web browsers. Despite being overcome in popularity by modern

Javascript libraries like Angular or React [15], it is still the foundation of many popular JavaScript libraries that depend on its functionalities.

4.5.3 TinyMCE

TinyMCE [16] is a powerful and flexible rich text editor that can be embedded in any web application and converts HTML textarea fields into editor instances. It integrates easily with JavaScript libraries like jQuery.

4.5.4 ChartJS

A simple and clean, yet powerful open-source JavaScript library [17] that allows front-end developers to deliver HTML based JavaScript charts. It offers 8 different chart types to visualize data, each of them animated and customizable. Charts are rendered in HTML5 Canvas which offers great performance across all modern browsers. It is aimed to be used in modern web development, for any given device screen size as it redraws charts on window resize, offering a responsive behavior.

Recording Contexts uses this library to show users the result of a comparison between their audio answers and the original answer when they perform a Repetition Experiment.

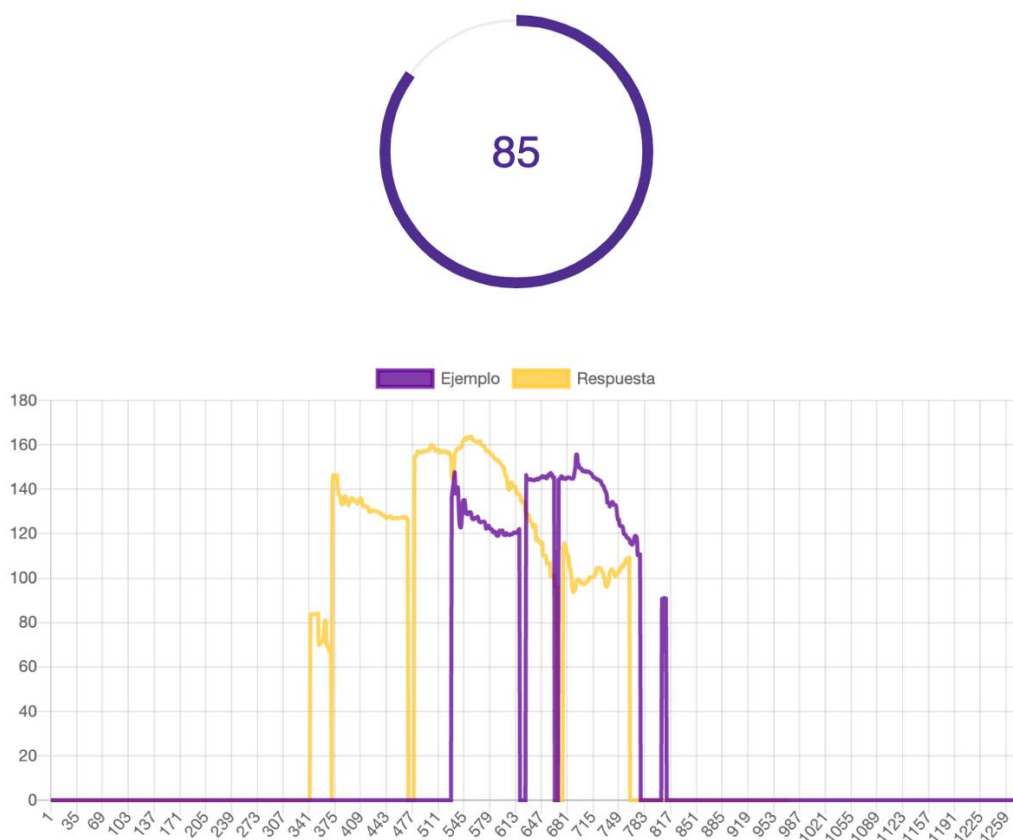


Figure 5: Chart created with ChartJs to illustrate pitch comparison between two recorded audios.

4.5.5 Custom Recording JavaScript

In order to allow Administrators and Users to record audios and upload them to the application, some custom JavaScript code had to be written. The file containing all the functionalities to make these recordings happen is called recorder.js. It uses the open-source library developed by Matt Diamond [18], which implements a custom object called Recorder to handle all interaction with JavaScript MediaDevices [19].

First, using JQuery's HTML Dom traversal and Event Handling, events are hooked onto all start and stop recording buttons in the page, these track the user's clicks on them, and start or stop the recording respectively.

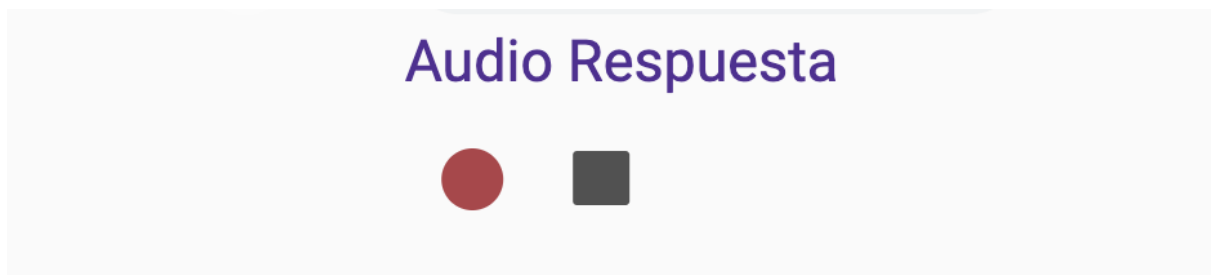


Figure 6: Custom buttons to start and stop the audio recording.

To record audio from users, JavaScript's function `MediaDevices.getUserMedia()` is used. This prompts first the user for permission to use a media input which produces a `MediaStream` that contains the requested type of media. The stream can include, for example, a video track (produced by either a hardware or virtual video source such as a camera, video recording device, screen sharing device, and so forth), an audio track (similarly produced by a physical or virtual audio source like a microphone), and possibly other track types.

Once the recording is stopped, an HTML5 `<audio>` tag with the recorded audio blob as a source is added to the web page. This creates audio controls to give users the ability to listen and download the audio if they want. Finally, an `XMLHttpRequest` is sent to the server in order to save the generated audio.

Audio Respuesta

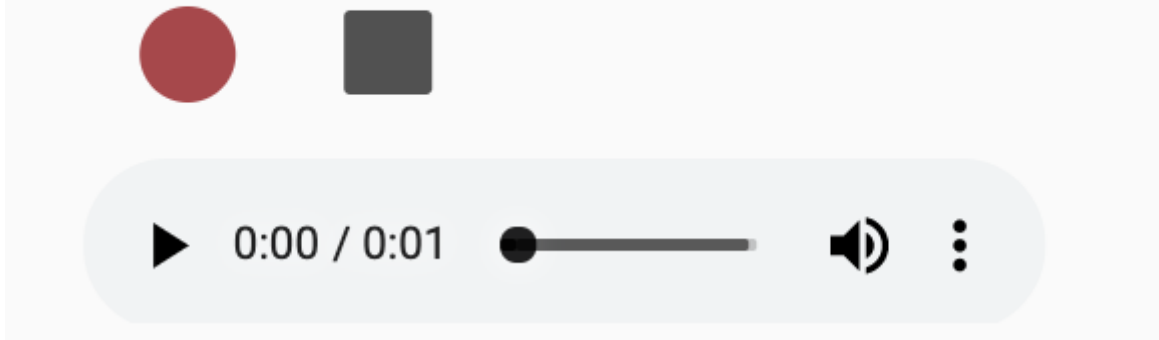


Figure 7: Example the audio controls generated when audio is recorded.

4.6 GitHub

GitHub is a very popular website and cloud-based service that helps developers store and manage their code, as well as track and control changes made to it. GitHub makes use of git the most used open-source version control system. It offers Code Reviewing and Project Management tools, and has a ton of Integrations that further expand its functionalities. It has primarily been used to host all the code and help to track versions of it.

5. Symfony4

Symfony is the PHP framework chosen to create the backend of Recording Contexts. It is one of the 2 options given by my tutor when the project was first outlined, the other one was Laravel.

5.1 What is a Framework?

A framework, or software framework, is a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform. For example, a framework may include predefined classes and functions that can be used to process input, manage hardware devices, and interact with system software. This streamlines the development process since programmers don't need to reinvent the wheel each time they develop a new application.

Nowadays there are many popular PHP frameworks: Zend Framework, CakePHP, Laravel, Symfony, CodeIgniter, etc. Each of them has its pros and cons and deciding between them is a matter of preferences and needs.

Laravel, for instance, is a relatively new framework that can handle complex web applications securely, at a considerably faster pace than other frameworks. It has a soft learning curve and it is a great framework for new developers that build a web application for the first time. It was developed using several Symfony Components but keeping it more simple, with a syntax which is easy to learn, read and maintain.

On the other hand, Symfony as a PHP framework is well suited for large-scale or complex enterprise level projects. For both novice and advanced users, Symfony offers complete flexibility. There is a lot of documentation and community support which make it easy to learn. It also offers a powerful debug toolbar which presents all the information needed for easy debugging. Although it may seem a more complex framework to approach for beginners, it allows developers to easily scale their web applications into whatever they want to achieve.

5.2 Why Symfony?

I decided to use Symfony due to my previous experience with the framework as I had been developing with it for 1 year. Despite having advanced knowledge, this had been acquired using version 2, which is now discontinued by the developers who maintain the framework. I decided then, it would be a good opportunity to use this project to recycle what I already knew using the most recent version. Symfony 4 introduces several changes in its core that make working with it a whole new experience.

Symfony is made of several components, more than 50 [20]. These are independent PHP libraries which provide functionalities already implemented to developers, which are ready to be used the moment they are installed. This installation is in fact very simple thanks to “Flex” tool which version 4 introduces. Paired with Composer, developers can install components without having to worry about dependencies because they are configured automatically on installation.

Like many other web frameworks, Symfony works under the Model-View-Controller paradigm (MVC from now on). By default, it delegates the Model and View control to third-party libraries, and excels at defining Controller responsibilities and processes. The most used library to manage Model interaction is Doctrine an Object Relational Mapper, and to handle View interaction it uses Twig, a PHP template engine. Both libraries are explained on its own later.

5.3 Components

Below, the most important components that have been used for the development of Recording Contexts are listed and detailed.

Form

Provides tools to easily create, process and reuse HTML forms. These are easy to use thanks to the perfect integration with Symfony's proposed programming paradigm, Model-View-Controller (from now on MVC). This allows a fluid collection of data from the View to the Model with ease.

```

public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder->add( child: 'titulo', type: TextType::class,
        array (
            'required' => true,
            'constraints' => [new NotBlank(['message' => 'Un formulario necesita un titulo.'])],
        )
    );

    $builder->add( child: 'secciones', type: CollectionType::class,
        array (
            'allow_add' => true,
            'allow_delete' => true,
            'by_reference' => false,
            'prototype_name' => '__seccion__',
            'entry_type' => SeccionType::class
        )
    );
}

```

Figure 8: Example of a FormType used in Recording Contexts.

Several of its benefits are:

- **Request Handling:** It manipulates the Request when an HTML form is submitted to the controller.
- **CSRF Protection:** A defense against Cross-Site-Request-Forgery attacks.
- **Templates:** It integrates perfectly with the template engine Twig, which allows HTML recycling when the form is shown.
- **Translation:** Supports translation of error messages, field labels and many other texts related to forms.
- **Validation:** It integrates with the Validator component to generate error messages when the data submitted by the users is evaluated.

Asset

Manages URL generation and versioning of web assets such as CSS stylesheets, JavaScript files and image files. Thanks to its integration with Twig developers can easily access assets from HTML templates and serve them to final users.

Security

Provides a security system for web applications developed with the framework. To list some of its many upgrades, there are utilities to create HTTP basic authentication, an interactive form to handle login inside the app, and allows developers to create their own authentications strategy. Moreover, a system to authorize final users by roles is also available.

```
access_control:  
  - { path: ^/admin, roles: ROLE_ADMIN }  
  # - { path: ^/profile, roles: ROLE_USER }
```

Figure 9: A piece of security.yaml configuration in Recording Context.

In the above image, an example of access control to a determined URL is shown. This restricts access to all URL's starting by "/admin" to those users that have role "Admin".

Webpack Encore

A powerful API to integrate the bundler Webpack.js [21] inside web applications, providing by that a system to build Javascript modules, pre-processing CSS and compiling and minifying assets.

Webpack compiles all CSS and JS into two main files app.js and app.css which are then included in the base HTML of the web app.

Validator

This tool allows developers to define validation rules for Entities and Classes, and their attributes.

Flex

A plugin that modifies the behavior of Composer's require, update, and remove commands, and automatically installs or removes dependencies of components and third-party libraries installed through the PHP package manager.

Process

This component allows developers to launch console commands in a sub-process, in a second plane. Moreover, it allows callbacks that inform about the result of that process. It is used in Recording Contexts to launch the Python script that compares the audios when a user records an answer to a context, returning a JSON with the result of that comparison:

```
$process = new Process( commandline: 'python3 uploads/audios/pitchCompare.py uploads/audios/' . $audioOriginal . ' uploads/audios/' . $audioRespuesta);
$process->run();
if (!$process->isSuccessful()) {
    throw new ProcessFailedException($process);
}

$array = json_decode($process->getOutput(), assoc: true);
return new JsonResponse(['message' => "OK",
    'dataOriginal' => $array['f01'],
    'dataRespuesta' => $array['f02'],
    'distancia' => $array['distancia']], status: 200);
```

Figure 10: Example of a Process execution in a controller.

Configuration

Setting up a web application with Symfony is fairly easy thanks to its console, a component that helps developers build, maintain and monitor projects developed with the framework. All parameters and state variables can be managed with YAML, XML or PHP configuration files which are later converted into PHP files and cached in Symfony's cache component to increase system speed and performance.

Prior to version 4, all configuration parameters and state variables were held in 2 YAML files, but since the introduction of Symfony4 Flex, each component has its own configuration file. This allows web applications to be decoupled and dependent-less, giving developers facilities to plug-in and out components with ease, and drives the framework to a micro-service oriented system.

```
doctrine:
  dbal:
    # configure these for your database server
    driver: 'pdo_pgsql'
    charset: utf8
    default_table_options:
      charset: utf8mb4
      collate: utf8mb4_unicode_ci

    url: '%env(resolve:DATABASE_URL)%'

  orm:
    auto_generate_proxy_classes: '%kernel.debug%'
    entity_managers:
      default:
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true
        mappings:
          App:
            is_bundle: false
            type: annotation
            dir: '%kernel.project_dir%/src/Entity'
            prefix: 'App\Entity'
            alias: App
```

Figure 11: : Doctrine configuration file in Recording Contexts.

Twig

Twig [22] is a PHP template engine which isn't part of Symfony's component package, but despite being a third-party library, it comes installed by default with the framework. It was created by the same developer that made Symfony, Fabien Potencier.

A template is a regular text file that can generate any text-based format (HTML, XML, etc). It contains variables or expressions, which get replaced with values when the template is evaluated by the PHP interpreter, and tags, which control the template's logic.

```
<div class="card p-1 mb-2 formulario">
  <div class="card-body">
    <h6 class="card-title text-dark">{{ formulario.titulo | capitalize}}</h6>
    <p class="card-text ml-2"><i class="fas fa-chalkboard"></i> Contextos: {{ formulario.secciones | length}}</p>
    <p class="card-text ml-2"><i class="far fa-comment"></i> Respuestas: {{ random(100) }}</p>
  </div>
  <div class="card-body">
    <a href="{{ path('ver_form', {'id': formulario.id}) }}" class="card-link ">VER</a>
    <a href="{{ path('edit_form', {'id': formulario.id}) }}" class="card-link editar-formulario">EDITAR</a>
    <a href="#" style="..." data-id="{{ formulario.id }}" class="delete card-link eliminar-formulario">ELIMINAR</a>
    <a href="#" data-id="{{ formulario.id }}" class="card-link text-secondary-dark url-generator"><i class="fas fa-link"></i> URL</a>
  </div>
</div>
```

Figure 12: Example of a template in Recording Contexts.

The most powerful part of Twig is its template inheritance. This allows developers to build a base “skeleton” template that contains all the common elements of their web and defines blocks that child templates can override.

```
<!DOCTYPE html>
<html>
  <head>
    {% block head %}
      <link rel="stylesheet" href="style.css" />
      <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
  </head>
  <body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
      {% block footer %}
        &copy; Copyright 2011 by <a href="http://domain.invalid/">you</a>.
      {% endblock %}
    </div>
  </body>
</html>
```

Figure 13: base.html twig file example.

In the image above, a template which defines an HTML skeleton document. It sets the names of the different “blocks” that form the skeleton using “tags”. These are reserved words used between “{% %}”, and indicate the start and the end of such blocks. Later, children templates that inherit from this base can override them.

```

{% extends "base.html" %}

{% block title %}Index{% endblock %}
{% block head %}
    {{ parent() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>

    <p class="important">
        Welcome to my awesome homepage.
    </p>
{% endblock %}

```

Figure 14: A children template that inherits from base.html.

The extends tag is the key here. It tells the template engine that this template inherits from another. When the template engine evaluates this template, it first locates the parent and then overrides any blocks that were originally defined in it with the values of the child template. Since the child template does not define the footer block in this example, the value from the parent template is used instead.

When rendering templates from the PHP controller, developers can pass values as variables to be used later in the template.

```

if(is_null($terms)){
    throw new \InvalidArgumentException();
}
return $this->render( view: 'formulario/formularioLanding',
    array(
        'formulario' => $formulario,
        'terms' => $terms,
    ));

```

Figure 15: Example of variables passed to a twig template. In this example, both \$formulario and \$terms are Entities that the template engine can access.

```

<div class="row justify-content-center mt-3">
  <div class="col col-lg-8 align-self-center">
    <div class="card">
      <div class="card-body" id="cuerpo-landing">
        <h2>{{ formulario.admin.paginaPrincipal.titulo }}</h2>
        {{ formulario.admin.paginaPrincipal.cuerpo | raw }}
      </div>
      <div class="card-footer align="right">
        <button type="button" class="btn btn-primary" data-toggle="modal" data-target="#termsModal">EMPEZAR</button>
      </div>
    </div>
  </div>
</div>

```

Figure 16: Usage of variables passed from the controller in the Twig template.

Doctrine

Doctrine [23] is an Object-Relational Mapper (ORM) for PHP 7.1+ that provides transparent persistence for PHP objects. It separates business logic from the persistence in a relational database management system.

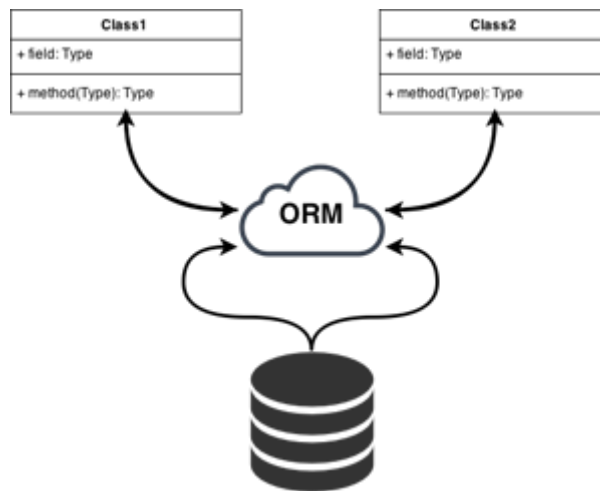


Figure 17: ORM concept in a diagram.

Doctrine gives programmers the ability to focus on the object-oriented business logic and worry about persistence only as a secondary problem. However, it does not mean that persistence is downplayed by Doctrine, but a strong belief that there are considerable benefits for object-oriented programming if persistence and entities are kept apart is what impulses the ORM.

It supports several types of Databases, which can be SQL like PostgreSQL or Non-SQL like MongoDB.

Entities

Entities are PHP Objects that can be identified over many requests by a unique identifier or primary key. There is one entity for each element of our data model.

The union between Entities and Doctrine can be done in two ways: using PHP Annotations or a configuration file. Recording Contexts entities have been set up with the first method because it is the easiest and the fastest. PHP Annotations help define how each field of the entity is going to be stored in the database, like the name of the column, the data type, and the properties of such data type.

```
/**
 * @ORM\Entity(repositoryClass="App\Repository\AdminRepository")
 */
class Admin implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $username;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];
```

Figure 18: Entity Admin in Recording Contexts. It demonstrates how database configuration for this entity is done through PHP Annotations.

Entity Manager

It's a class that provides access points to the complete lifecycle management for entities, and transforms them from and back to persistence.

Entity Repositories

Conceptually, a Repository encapsulates the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer.

In Doctrine, every Entity must declare a repository which by default offers a bunch of convenience methods that can be used to query for instances of that Entity. Developers can later override those methods or implement new ones as they find necessary.

Default functions in a EntityRepository:

- findAll(): returns all records of an Entity as an array.
- find(): returns one record matching the primary key passed as a parameter.
- findBy(): returns records matching the query parameters passed to the function. These are the ones used in the clause WHERE of a SQL query.
- findOneBy(): returns one record matching the query parameters passed to the function.

```
/**
 * @param Request $request
 * @Route("/admins/list", name="view_admins")
 * @IsGranted("ROLE_SUPERADMIN")
 */
public function viewAdmins(Request $request){

    $em = $this->getDoctrine()->getManager();

    $admins = $em->getRepository( className: 'Admin')->findAll();

    return $this->render( view: 'admin/viewAdmins', [
        'admin' => $this->getUser(),
        'admins' => $admins,
        'activeTab' => 'admins'
    ]);
}
```

Figure 19: Function in AdminController that returns an HTML page with all Admins. It uses the EntityManager to find all Admins through the AdminRepository.

Doctrine offers developers its own query language called Doctrine Query Language (DQL), which in essence provides powerful querying capabilities over the object model. A common mistake is to mistake DQL for being just some form of SQL and therefore trying to use table names and column names or join arbitrary tables together in a query. DQL is a query language for the object model, not for the relational schema that is defined from it. Therefore when querying with it, developers do not have to worry about how tables or columns are named. Doctrine is responsible for transforming all DQL queries to SQL and perform the collection of records.

```

public function findAllOrdered()
{
    $dql = 'SELECT admin FROM App\Entity\Admin admin ORDER BY admin.name DESC';
    $query = $this->getEntityManager()->createQuery($dql);
    return $query->execute();
}

```

Figure 20: : Example of a DQL query to obtain all Admin entities ordered by their name.

Furthermore, the ORM provides an API called QueryBuilder designed for conditionally constructing a DQL query in several steps. It provides a set of classes and methods that is able to programmatically build queries with a fluent API.

```

public function findAllOrdered()
{
    $qb = $this->createQueryBuilder( alias: 'admin');
    $qb->select( select: 'admin')
        ->from( from: 'App:Admin', alias: 'a')
        ->where( predicates: 'a.id = :identifier')
        ->orderBy( sort: 'a.firstName', order: 'DESC')
        ->setParameter( key: 'identifier', value: 100); // Sets :identifier to 100

    return $qb->getQuery()->getResult();
}

```

Figure 21: Example of a query built using the QueryBuilder.

6. Data Model

6.1 Class Diagram

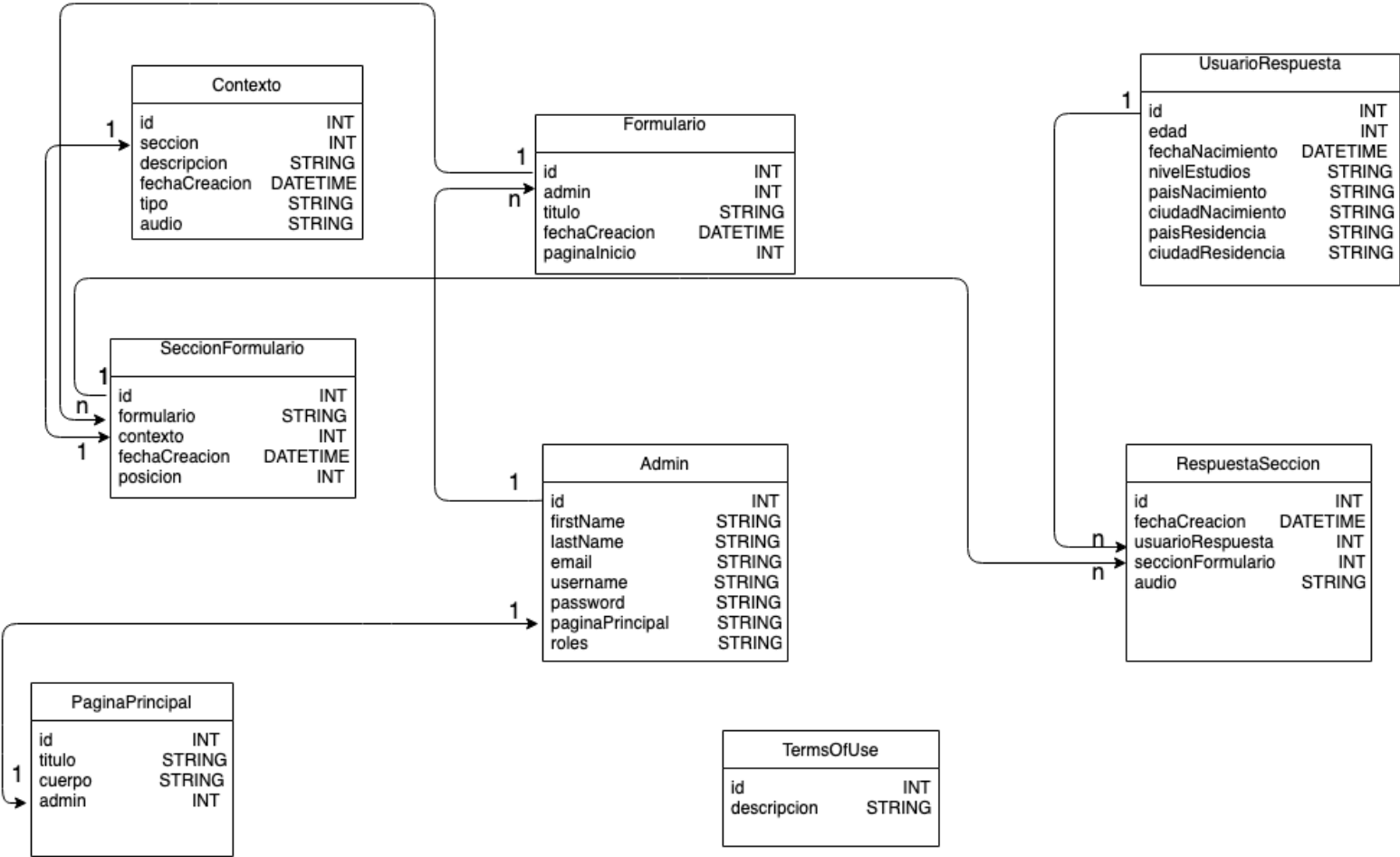


Figure 22: Class diagram of Recording Contexts.

Although PHP is not a typed language, those that define types for all variables and attributes in a class, it is necessary to define a type for each attribute in a class in order to maintain a correlation between database values and code. For that reason, Symfony uses PHP Annotations to declare the type of variables using inline comments. Annotations can be used to declare simple variable types like integers or strings, but also class variables:

```
/**
 * @return null|string
 */
public function getDescripcion(): ?string
{
    return $this->descripcion;
}
```

Figure 23: Example of PHP Annotation to declare a type String in a variable.

```
/**
 * @return Admin|null
 */
public function getAdmin(): ?Admin
{
    return $this->admin;
}
```

Figure 24: Example of PHP Annotation to declare a Class type variable.

6.2 Database Schema

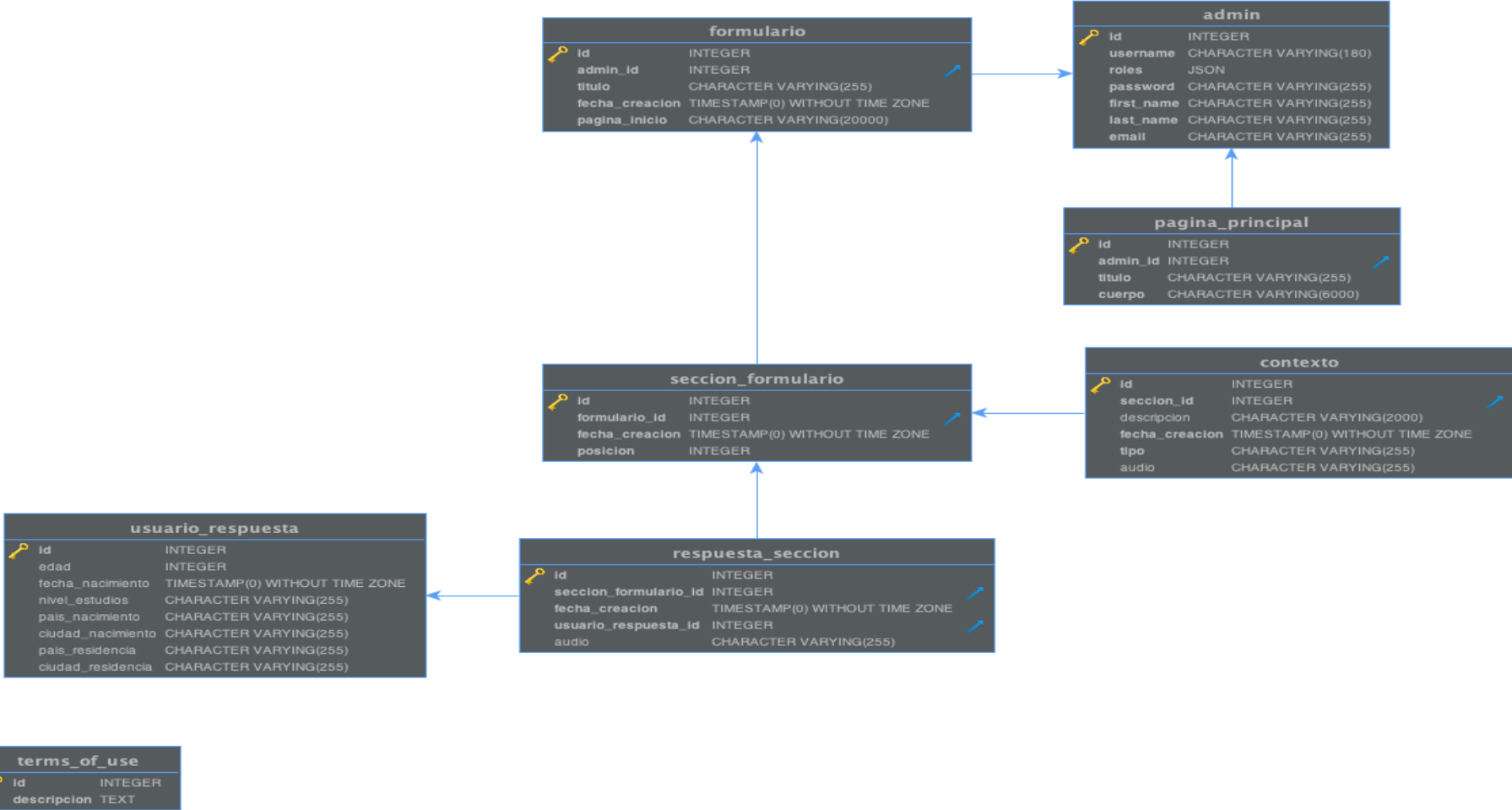


Figure 25: Definition of data model.

7. Recording Contexts

7.1 Project structure

After the update to version 4, Symfony completely changed the way projects are structured. In its previous versions, a project was mainly conformed of Bundles, which are a collection of files and folders organized in a specific structure. They could be packaged specifically to an application such as AdminBundle (admin section), BlogBundle (site's blog), etc. But they could also be shared between applications, to do so they had to be modelled as generic bundles with decoupled dependencies so they could be easily copied from one application to another. Therefore for a Bundle to be shareable it had to contain all the logic of a small application. The typical structure of a Bundle was:

- **Controller:** Contains all the Controller classes.
- **Entity:** Model definition for the bundle, contains Entities and Repositories.
- **Form:** Form classes.
- **Resources:** Usually divided into 3 sub-directories. Contains configurations files (config directory), view templates (view directory), and pure resources files like JavaScripts, stylesheets, images, etc (public directory).
- **Tests:** Unit tests classes.

Whenever a Bundle was installed (a component) developers had to enable and configure it manually. Moreover, if the bundle was no longer needed, removing it was even more difficult because it had to be decoupled from all configuration files it was referenced in.

Now in Symfony 4 and with the introduction of Flex, the framework has dropped the Bundle structure developers were used to. Components can be easily installed just requiring them via Composer, and they are automatically configured. Furthermore, versión 4 changed the way projects are structured, focusing on scalability and a micro-services-oriented applications development.

The new project structure is a Unix-like directory structure with fewer sub-directories:

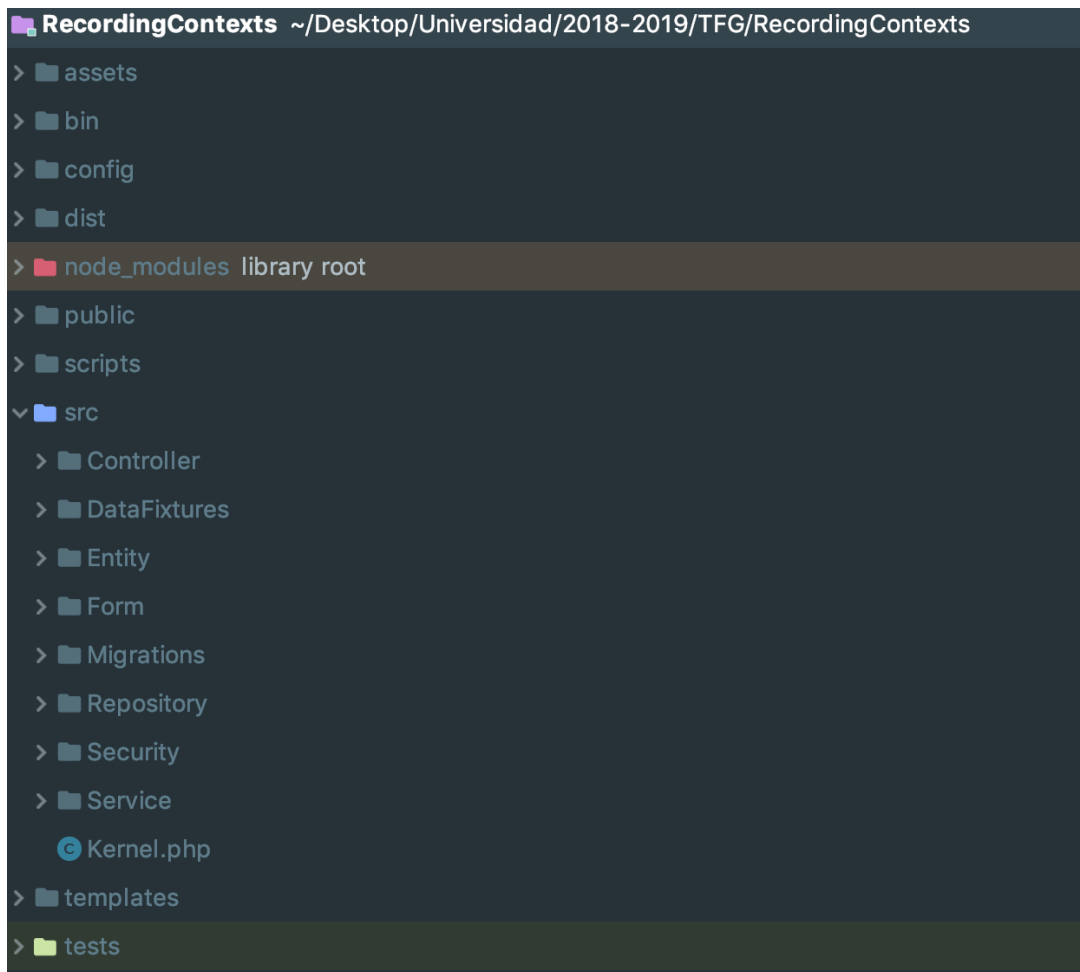


Figure 26: Recording Contexts project structure.

- **Assets:** Contains CSS and JavaScript files that will be later compiled using WebPack, and exported to the “public” directory so they can be referenced in the HTML templates.
- **Config:** All configuration files for the project are located here. If a installed component needs a configuration file, it is placed here upon requiring it through Composer.
- **Node Modules:** Contains javascript libraries that are to be used in the project.
- **Public:** Contains files that are accessible through the web application, this includes JavaScripts, stylesheets, images, etc.
- **Src:** Where the backend of the application lives. It contains sub-directories:
 - **Controller:** Controller classes.
 - **Entity:** Entities classes.
 - **Repository:** Repository classes.
 - **Form:** Form classes.

- **Migrations:** PHP classes that help keep a history track of changes done to the database. They can be later executed to create a copy of the current structure of the database or roll it back to a previous state.
- **Service:** Contains classes that provide a “service”. They can be understood as classes that are specialized in accomplishing a specific task like uploading a file, send a mail, etc. They can be accessed from any Controller if they are required.
- **Templates:** Contains the views.
- **Tests:** Unit test classes.

7.2 Application Structure

7.2.1 Controllers

There are 7 controllers in Recording Contexts:

- **AdminController:** Holds all the logic related to the Admin interface except the one in charge of creating, editing and viewing in detail experiments. Only Admins can access routes in this controller.
- **FormulariosController:** Manages experiments logic. Only Admins can access routes in this controller
- **SecurityController:** Holds the logic related to admin authentication (login and logout).
- **ContestarController:** Manages the pages that Users access to fulfil an experiment.
- **RespuestasController:** Handles the upload of audio recordings by Users, when they answer an experiment’s context.
- **ImagePostController:** Handles the upload of images through the TinyMCE plugin.
- **DefaultController:** Manages the pages of Terms of Use, About Us and the default page.

7.2.2 Entities

As presented in the Data Model, there are 8 Entities in Recording Contexts and each of them has its own repository:

- **Admin:** Represents administrator users and makes use of the **UserInterface** interface provided by the Security component to handle authentication and access permissions to certain controllers.
- **UsuarioRespuesta:** Represents final users who do not require authentication to access the application.
- **Formulario:** Represents experiments, each of them is owned by an administrator and have a collection of contexts.
- **Seccion:** An abstraction of the position of a context within an experiment.
Context: Represents contexts.
- **RespuestaSeccion:** Represents answers to contexts made by final users.
- **TermsOfUse:** This entity was created to give the Superadmin the ability to edit the Terms of Use and Privacy of RecordingContexts within the application.

7.3 Pitch Compare

Comparison of repetition-contexts audio files is done using a Python script provided by the Phonetics Laboratory.

To launch the script, Symfony's Process component is used. It allows executing a command-line process from a controller. This command-line is a call to the python script which receives both paths to the files to compare as parameters. When it finishes, it returns a JSON (JavaScript Object Notation) tuple containing the values of both audio curves and the distance between them. Finally, the controller takes back control of the process and returns the tuple to the view.

Invocation of the comparison process is done by an AJAX call when the user ends recording his answer. Let us see a diagram:

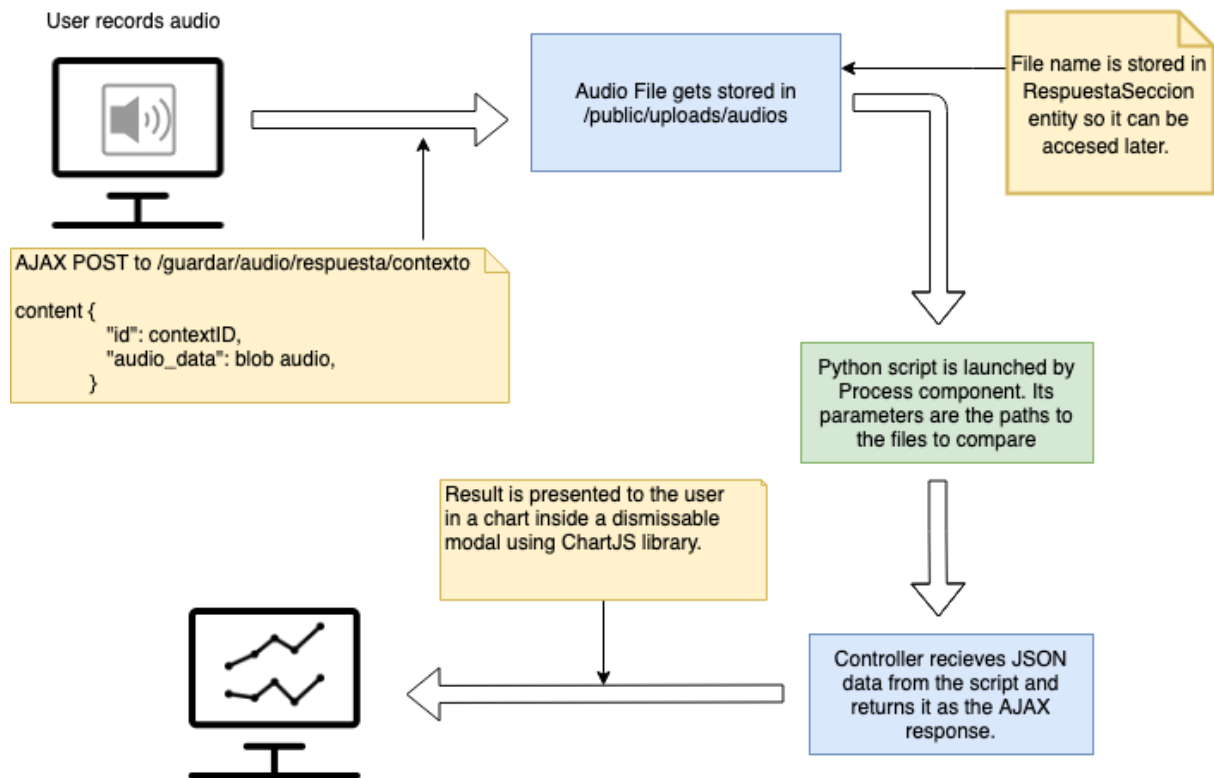


Figure 27: Diagram illustrating the process to compare 2 audio files.

Although a good implementation has been achieved to be able to carry out the comparison of 2 audio files through the integration of PHP with Python, it is not by far the most efficient and elegant way of doing it.

If the scalability of the web application is taken into account, it is not a good practice to have the same server responsible for serving the website to the user and also for processing the audio file comparison. If the application grows in users there will come a time when the computing capacity of the server will be affected by the number of requests performed by those users, therefore, no computing capacity will be left to the python script.

A way to approach the solution to this problem is the implementation of an external web service dedicated exclusively to the comparison of audio files. This would separate both processes and not compromise the general operation of the application. A proposal for the implementation of this web service and the integration with the current version of the application can be read in chapter 10.

8. Validation and Performance tests

8.1 PHPUnit

To ensure all functionalities of the web page worked correctly Unit Testing was implemented. The most used testing framework for PHP is PHPUnit [24] an open-source library developed by Sebastian Bergmann. To test not just functionalities but also Entities and Repositories interaction with the database a mocking of such database had to be done. To do so I decided to use an instance of SQLite3 database that would be always created at the beginning of the tests and populated with Data Fixtures generated by Doctrine's Fixtures Bundle.

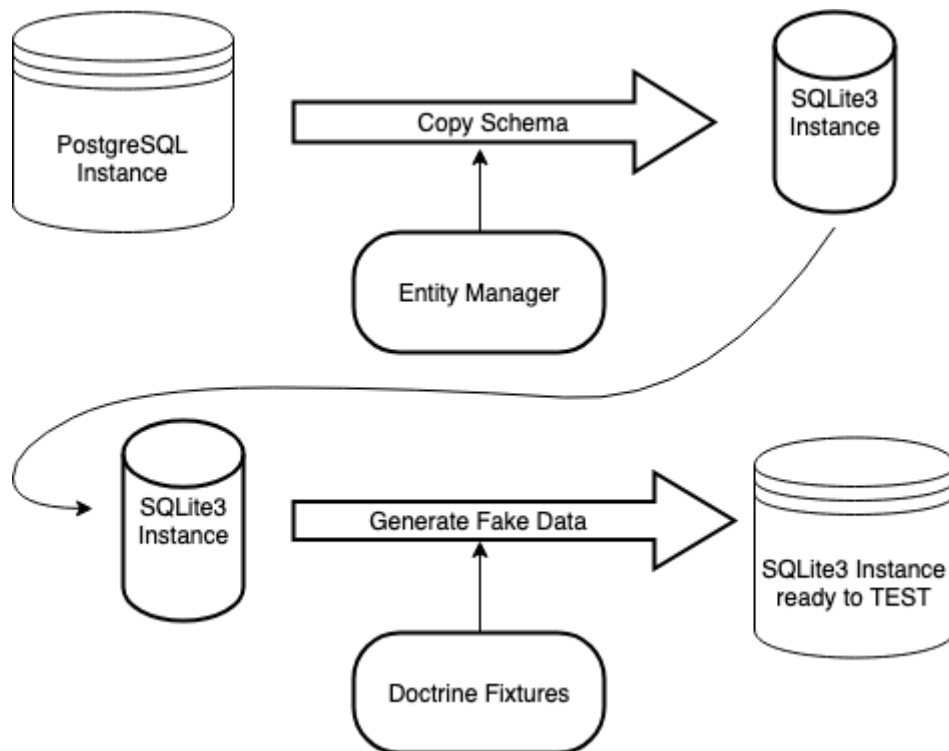


Figure 28: Diagram of database change to test.

As I did not follow a Test Driven Development and the implementation of PHP Unit was done when the project was already well advanced I decided to focus on implementing Functional Tests for all the Controller Classes.

9. Deployment

Due to the application not being just a Bachelor's Degree Final Project but a web developed so that the Phonetics Laboratory can use it as its main tool to obtain audio recordings for its study, it is necessary that the platform is deployed in an online server.

For this, we studied different hosting options and compared what each of them offered, in terms of server performance, price and difficulty in maintaining the said server.

9.1 Server Requirements

In terms of server capabilities, there were 2 main objectives that the server should cover in a comfortable way: computing capacity and storage.

The first requirement is necessary for the calculation of the comparison between audios performed by the Python script launched from the application when a repetition exercise (when final users upload an audio recording imitating a previous one uploaded by the administrator) is executed. Here is an example of memory usage of the said script launched in a local environment, comparing two small audio recordings:

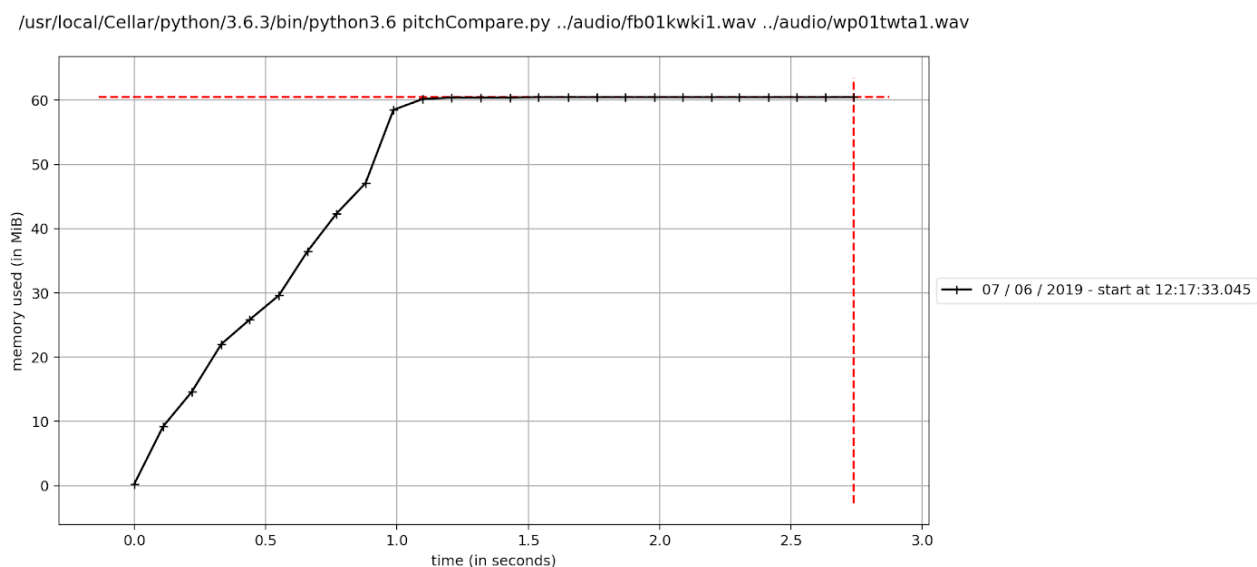


Figure 29: Graphic of memory usage by pitch compare python script.

As can be seen in figure 1, time execution for small audios is pretty fast and it does not consume a lot of memory (60 MiB in this example). We can say with total certainty that the experiments that will be carried out in Recording Contexts will use short audios, of no more than 10 seconds of duration, therefore, the minimum computing capacity that a server has to offer to make the comparison of such recordings can be around 200 MiB.

Memory and CPU consumption for hosting the application is also to be taken into account when deciding over a server option. Summarizing what has been said, we decided that the server should have at least 500 MiB of RAM memory and 1 core.

The second objective, storage, is pretty straightforward. Recording Contexts is expected to collect a lot of audio recordings and although each of these recordings is expected to have a size of no more than 10Mb, the sum of all of them can produce a storage problem if the server is short of storage capacity.

9.2 Hosting Options

As the final responsibility to maintain the application and pay for the hosting is entirely on the Phonetics Laboratory, they made the decision to use only services that offer a free plan. This is because of the uncertainty of being able to maintain funding for the project and therefore the cost of this should be as low as possible.

9.2.1 Custom Server

The first option proposed by the Lab was to host the application themselves in a private server installed in the laboratory. This could be the best option if not for the heavy restrictions in terms of security that the University of Barcelona imposes in order to have a server with a connection to the internet that can serve as an endpoint.

After a first meeting with the Phonetic Laboratory staff, It was considered that for the time being this option would be complex to implement, time-consuming, and hence discarded for the current TFG. Therefore, a future plan to move the whole application to a private server is taken into account as an improvement. A small explanation about the steps that should be done to make this possible is explained in section 10.1.

9.2.2 OpenShift

The second option was to use OpenShift Hosting Services [25]. They offer a free plan with good server specifications, that could fit our requirements, but lack storage capability and an easy to use configuration interface. Moreover, all applications deployed to OpenShift servers are containerized which Recording Contexts is not. Containerization [26] involves bundling an application together with all of its related configuration files, libraries and dependencies required for it to run in an efficient and bug-free way across different computing environments.

OpenShift provides base skeletons for containerized applications in several languages, but for PHP it does not provide a skeleton for a web application developed under Symfony. It could be possible though to containerize your own application and upload it to an OpenShift server, but as I had no previous experience in creating container-based applications I decided not to invest time in learning to do so when there existed other options.

As a future improvement for this project I have detailed the steps that should be taken in order to port Recording Contexts to a free OpenShift server, it can be read in section 10.1.

9.2.3 Amazon Web Services

As the third and final option, we considered Amazon's hosting services [27]. These are the most used cloud computing services in the world, trusted by thousands of companies some of them with big names like Netflix, Airbnb or Unilever. They offer a free tier with good server specs for 12 months and have an easy-to-use web-console user interface and plenty of tutorials to help developers push their projects into the cloud. For that reason, we decided to go on with AWS and deploy Recording Contexts in one of their servers.

9.3 AWS Setup

The easiest way to deploy a web application to AWS is to use Elastic Beanstalk (EB). You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring.

Creating an EB application is simple, you just have to give it a name, select the programming language your web backend is built with and, optionally, make a direct upload of your code.

Furthermore, you can tweak the default configuration of the server such as:

- Add environment variables.
- Select the type of EC2 instance the app is created with.
- Storage size.
- Security profiles.
- Add a database.

EB applications mainly consist of 3 AWS services that are automatically launched and seamlessly connected between them:

EC2 Instance

A virtual server designed to run applications that are created from Amazon Machine Images [28]. These are templates that contain the software configuration (operating system, application server and applications) required to launch an instance. Our EC2 instance has the following configuration:

- Amazon's Linux 64bit v2.8.9
- PHP 7.2
- Apache 2.4.39
- Python 3.6

RDS Database

Amazon's Relational Database Service (RDS) provides 6 six familiar database engines to choose from, including Amazon Aurora, MySQL, MariaDB, Oracle, Microsoft SQL Server, and PostgreSQL. A DB instance is an isolated database environment in the cloud that can contain multiple user-created databases that can be accessed by using the same tools and applications that developers would use with a stand-alone database instance. Our instance has one virtual CPU and 1GB of RAM and runs PostgreSQL v10.6.

S3 Bucket

Amazon Simple Storage Service (Amazon S3) is a scalable, high-speed, web-based cloud storage service designed for online backup and archiving of data and applications. It consists of Buckets that are logical units of storage used to store objects which consist of data and metadata that describes the data.

Elastic Beanstalk automatically creates an S3 Bucket to store objects that are required for the proper operation of our application like application versions, source bundles, log files saved configurations, etc.

10. Conclusions

As a general conclusion we can state that Recording Contexts has satisfied all the requirements and functionalities that were defined at the beginning of the project. It is a stable web application that will help the Laboratory of Phonetics expand their investigation on analysis of acoustic data. It offers them a system to gather a greater number of audio data and at the same time give feedback to those users who complete the experiments.

To do so, the platform has been developed under the modern standard of web applications, using state-of-the-art technologies to provide a system with a lot of potential and that can grow both in terms of scalability and functionality in the future.

Personally, the goal of the project was to prove myself I could apply all the knowledge I have gained during my studies into developing a bigger project, while also learning to satisfy the requirements imposed by an external client.

10.1 Future improvements

Although having satisfied all needs, there is still a lot of room for improvements regarding functionalities and scalability of the platform.

10.1.1 Minify CSS & JavaScript content

Activate the unification and minification of CSS and JavaScript [29]. This allows developers to merge all CSS and JavaScript files into one single file each, eliminating white spaces and jumping lines. Having minified all files reduces the number of requests that the web browser must make to the web server and also lowers the download size of such files.

10.1.2 Use OpenShift hosting

As mentioned in chapter 8, OpenShift offered good server specifications that totally fit the needs of the application as it is now. However, it could not be used as it did not provide native tools to deploy with ease a web application developed with Symfony. What

it does provide is a base skeleton for launching web applications developed with Laravel, another PHP framework which was created using Symfony's components. Therefore, a future improvement would be porting Recording Contexts to Laravel and then host it in an OpenShift server.

10.1.3 Own server

As mentioned in chapter 8 the best option to host the web application was to use a private server hosted in the Phonetics Laboratory. Provided that permission to make it possible is given by the University of Barcelona these are the steps that should be followed to create the server:

1. Install LAPP software: Linux as the operating system, Apache for the web server, PostgreSQL as the database engine and PHP to run the backend.
2. Install Python 3.6 to run the comparison script.
3. Configure Apache.
4. Set permissions to the public folder where the application will live, so it is accessible for every user that makes a request to the server.
5. Use ngrok to expose the local server to the public internet over a secure tunnel.
6. Install the application on the public folder.
7. Create the database and point the backend to it.

10.1.4 Web Service to compare audios.

It was previously mentioned in chapter 6, using the same server that runs the web application to make the comparison of audio recordings is not a good practice. A possible way to solve this problem is to create a web service in another server using python 3.6 and Django [30].

Django is a python framework to create web applications. In this case, we would not need to create a web application but a RESTful web service [31] that would run the comparison script and return the result as a JSON tuple. Currently, all audio files are stored locally when they are uploaded to Recording Contexts, so it is impossible for an external web service to access them. To solve this problem, it would be necessary to create an AWS S3 bucket to upload the audio files after they are uploaded to Recording Contexts. Later when a request to compare 2 files is made to the python web service, it just has to retrieve them from the S3 bucket, process them and return the JSON response.

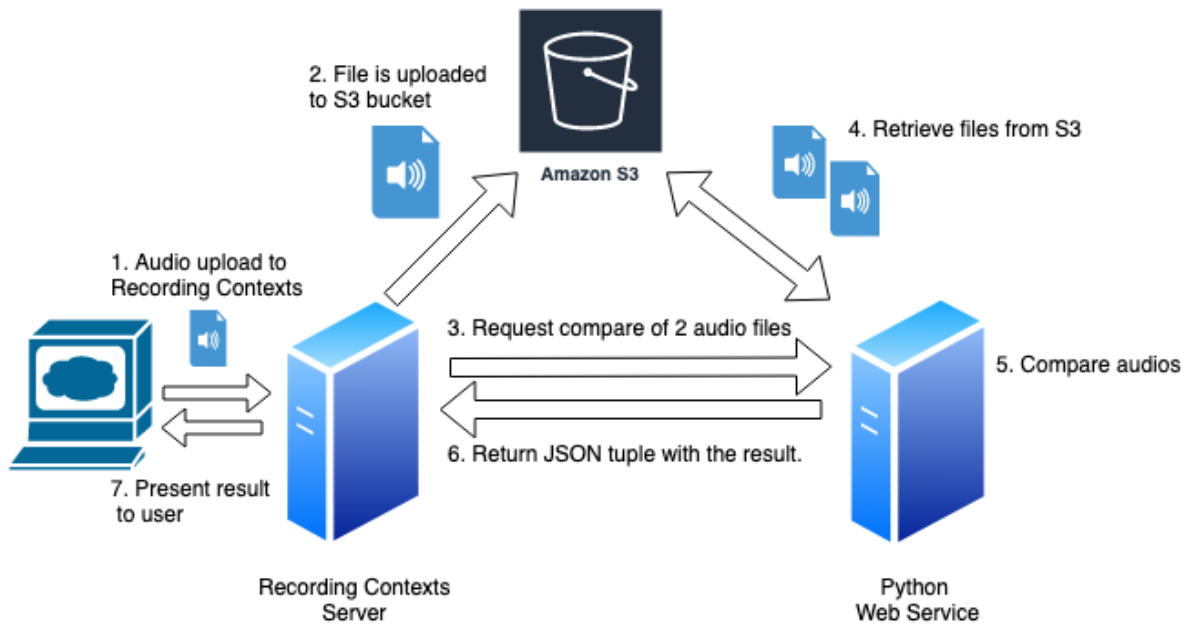


Figure 30 Diagram of interaction between Recording Contexts server and an hypothetical Python web service when an audio file is uploaded

This solution would not only lighten Recording Contexts server’s workload but also free storage space as it would not need to keep all audio files in the local filesystem.

11. Bibliography

- [1] Talkin, D. (1995). A robust algorithm for pitch tracking (RAPT). *Speech coding and synthesis*, 495, 518.
- [2] Stan Salvador, and Philip Chan. (2007) "FastDTW: Toward accurate dynamic time warping in linear time and space." *Intelligent Data Analysis* 11.5: 561-580.
- [3] Vanrell, M.M., Feldhausen, I., Astruc, LI. (2018). The Discourse Completion Task in Romance prosody research: status quo and outlook. In: Feldhausen, I., Fließbach, J., Vanrell, M.M. (Editors). *Methods in prosody: A view from Romance languages*. Berlin: Language Science Press, pp. 191-227.
- [4] Schoenherr, Steven E. (5 May 2004). "The Digital Revolution". Archived from the original on 7 October 2008.
- [5] "Internet Revolution." U*X*L Encyclopedia of U.S. History. [Online]. Available: <https://www.encyclopedia.com>.
- [6] "LinguaLibre". [Online]. Available: <https://lingualibre.fr/wiki/LinguaLibre>About>
- [7] Blum-Kulka, S. 1982. "Learning to Say What You Mean in a Second Language: A Study of Speech Act. Performance of Learners of Hebrew as a Second Language." *Applied Linguistics* 3: 29-59.
- [8] "Apache HTTP server project". [Online]. Available: <https://httpd.apache.org/>
- [9] "Datanyze: Apache market share". [Online]. Available: <https://www.datanyze.com/market-share/web-and-application-servers/Alexa%20top%201M/apache-http-server-market-share>
- [10] "PostgreSQL Documentation". [Online]. Available: <https://www.postgresql.org/docs/>
- [11] "Composer | Dependency Manager for PHP." [Online]. Available: <https://getcomposer.org/>.
- [12] "FezVrasta, Bootstrap Material Design". [Online]. Available: <https://fezvrasta.github.io/bootstrap-material-design/>
- [13] "How Google created a custom Material theme". [Online]. Available: <https://material.io/articles/how-google-created-a-custom-material-theme.html>
- [14] "jQuery documentation". [Online]. Available: <https://api.jquery.com/>
- [15] "Stack Overflow Trends Survey on JavaScript frameworks". [Online]. Available: <https://insights.stackoverflow.com/trends?tags=jquery%2Cangular%2Cangularjs%2Creactjs>
- [16] "TinyMCE documentation". [Online]. Available: <https://www.tiny.cloud/docs/>

- [17] “ChartJS Open source HTML charts”. [Online]. Available: <https://www.chartjs.org/>
- [18] “Matt Diamond RecorderJS library”. [Online]. Available: <https://github.com/mattdiamond/Recorderjs>
- [19] “MediaDevices - Web APIs | MDN”. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices>
- [20] “Symfony Components”. [Online]. Available: <https://symfony.com/components>
- [21] “WebpackJS concepts”. [Online]. Available: <https://webpack.js.org/concepts/>
- [22] “Twig - The flexible, fast and secure PHP template engine”. [Online]. Available: <https://twig.symfony.com/>
- [23] “Doctrine”. [Online]. Available: <https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html#what-is-doctrine>
- [24] “PHPUnit Manual”. [Online]. Available: <https://phpunit.readthedocs.io/en/8.2/>
- [25] “OpenShift: Container Application Platform by RedHat”. [Online]. Available: <https://www.openshift.com/>
- [26] “What is Containerization?”. [Online]. Available: <https://hackernoon.com/what-is-containerization-83ae53a709a6>
- [27] “Amazon Web Services (AWS) Cloud computing services”. [Online]. Available: <https://aws.amazon.com/>
- [28] “Amazon Machine Images”. [Online]. Available: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html>
- [29] “Optimizing Encoding and Transfer Size of Text-Based Assets”. [Online]. Available: <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#minification-preprocessing--context-specific-optimizations>
- [30] “Django Python Framework”. [Online]. Available: <https://www.djangoproject.com/>
- [31] “What are RESTful Web Services?”. [Online]. Available: <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>

12. Annexes

12.1 User manual

12.1.2 Log in as Administrator

To log into Recording Contexts a user must click the top-right button on the navigation bar.



Figure 31 Homepage.

He'll be redirected to the login page:

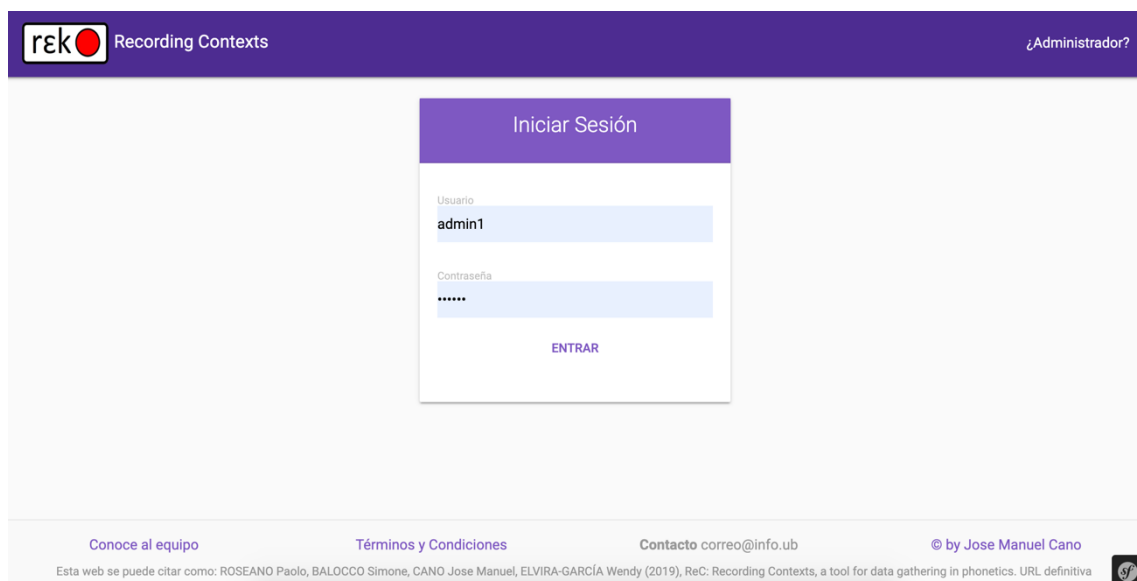


Figure 32 Log in form.

12.1.2 Admin/Superadmin Interface

Once an Admin/Superadmin logs in successfully, he is presented with the following page:

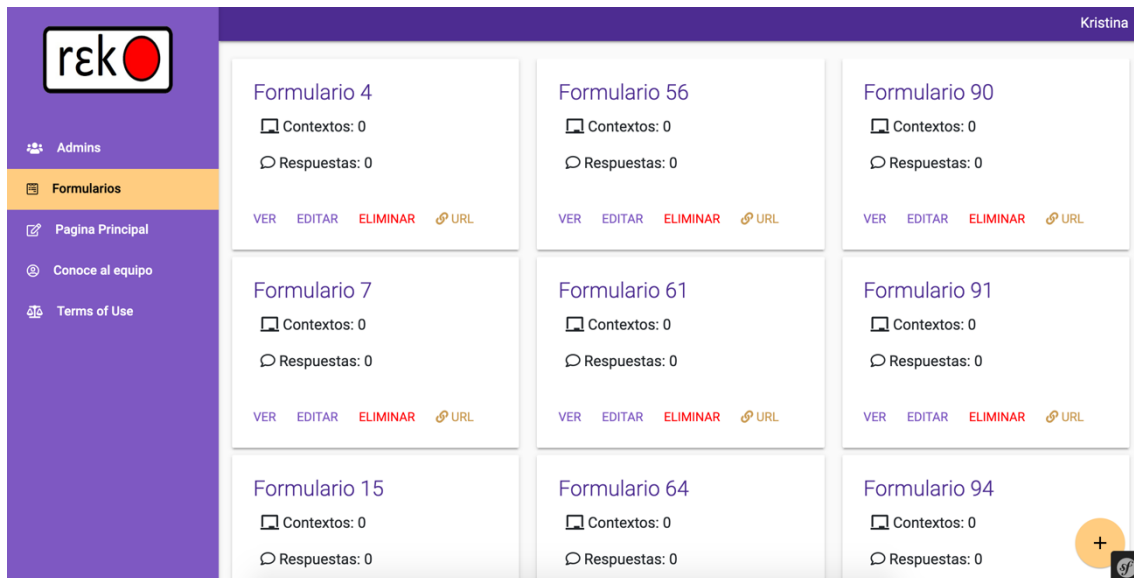


Figure 33 Superadmin interface homepage.

Admins can use the left navigation bar to navigate through the different sections of the Admin interface which has 4 or 5 options (depending on the type of user): Formularios, Pagina Principal, Conoce al equipo, Terms of Use, Admins (just Superadmin can access this section).

12.1.3 Experiment creation/edition

To create an Experiment, Admins have to click the floating button on the bottom-right of the page. To edit an Experiment, they have to click the option “EDITAR” in the card of the experiment they want to edit:

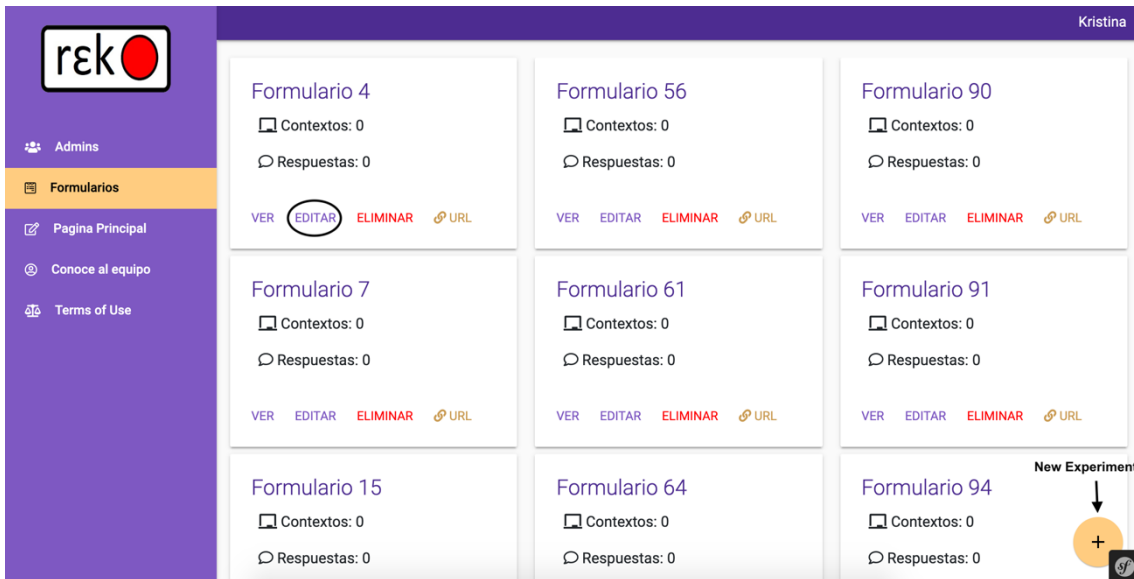


Figure 34 Superadmin homepage interface with annotations.

Either way they will be redirected to the creation page of an experiment, where they will be able to edit the title and manage the sections:

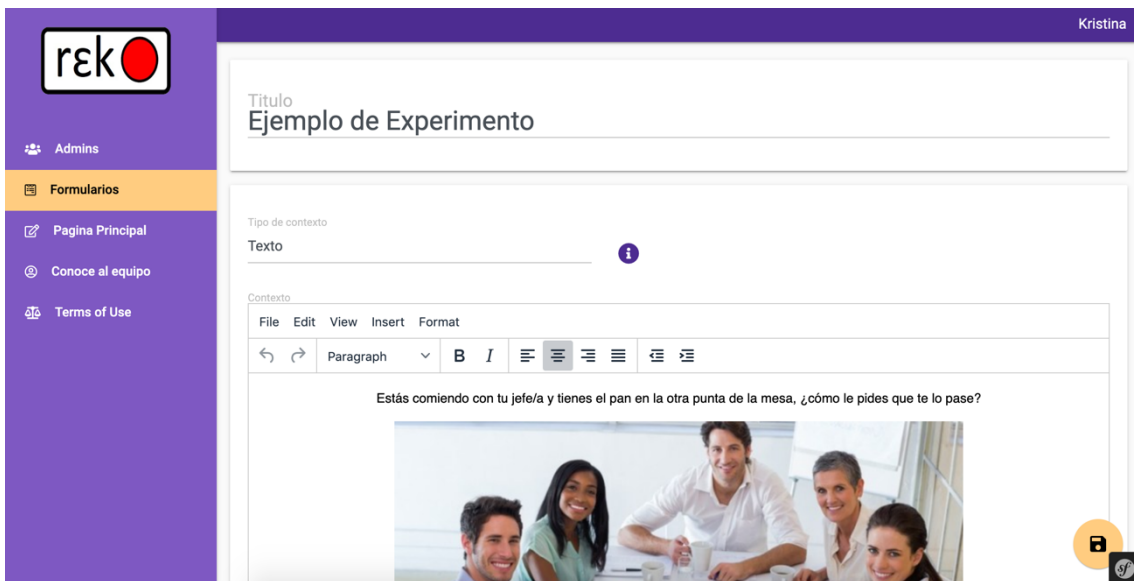


Figure 35 Experiment creation page.

12.1.4 Record/upload an audio example.

To record or upload an audio example and attach it to an experiment's context, click on "VER" option of an experiment card to be redirected to the detail page of the experiment:

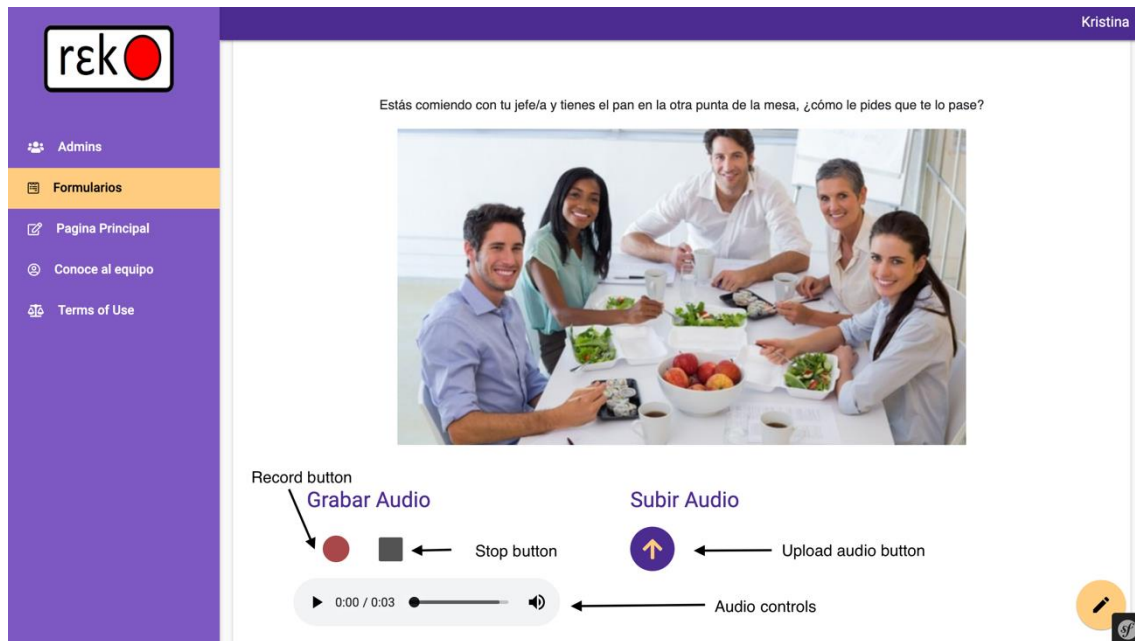


Figure 36 Experiment detail page.

To record an audio, click on record button at the bottom left of the context. Once the recording is finished click on stop button. To upload directly a pre-recorded audio file, click on upload button and a window to select the file will be prompted. Once a context has an audio, controls to play it will appear at the bottom of the record and stop buttons.

12.1.5 Edit Landing Page.

The Landing Page of an Admin is the first page final users see when they access Recording Contexts to answer an experiment. To edit it, click on the "Pagina Principal" option on the navigation bar and access the detail page of the landing page:

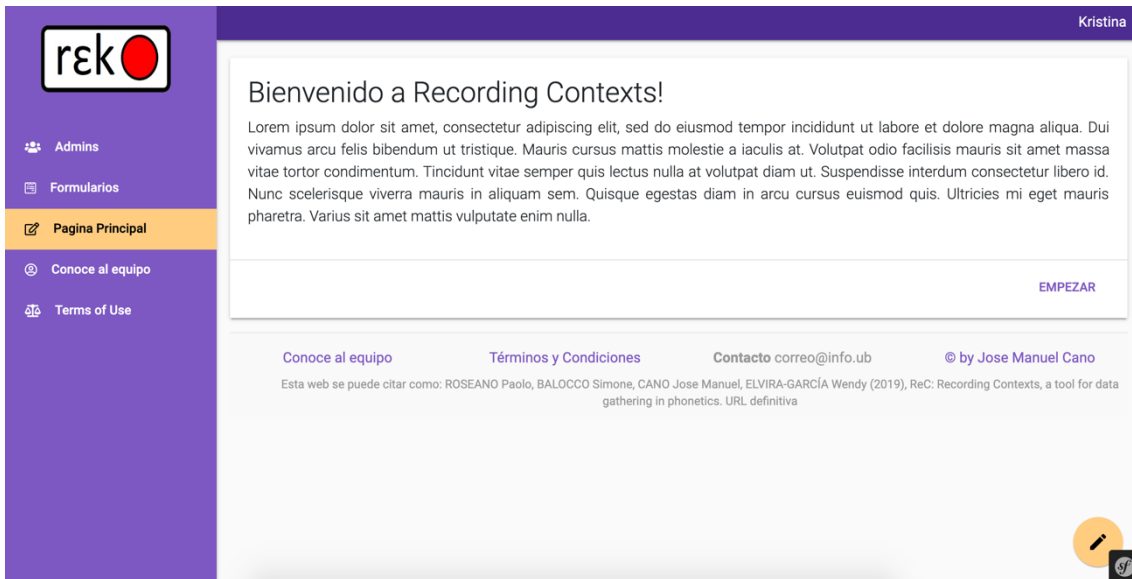


Figure 37 Landing Page detail page.

Here Admins can see an example of how their Landing Page will look like on an experiment. To edit it click on the bottom-right floating button:

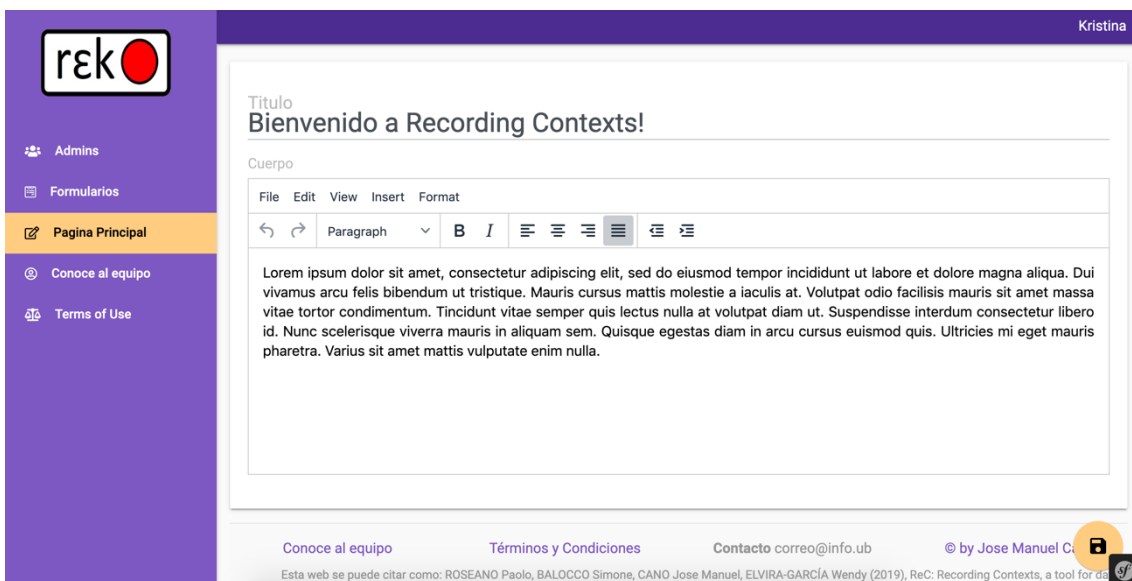


Figure 38 Landing Page edit page.

12.1.6 Admins management

Superadmins have an extra section to manage Admins:

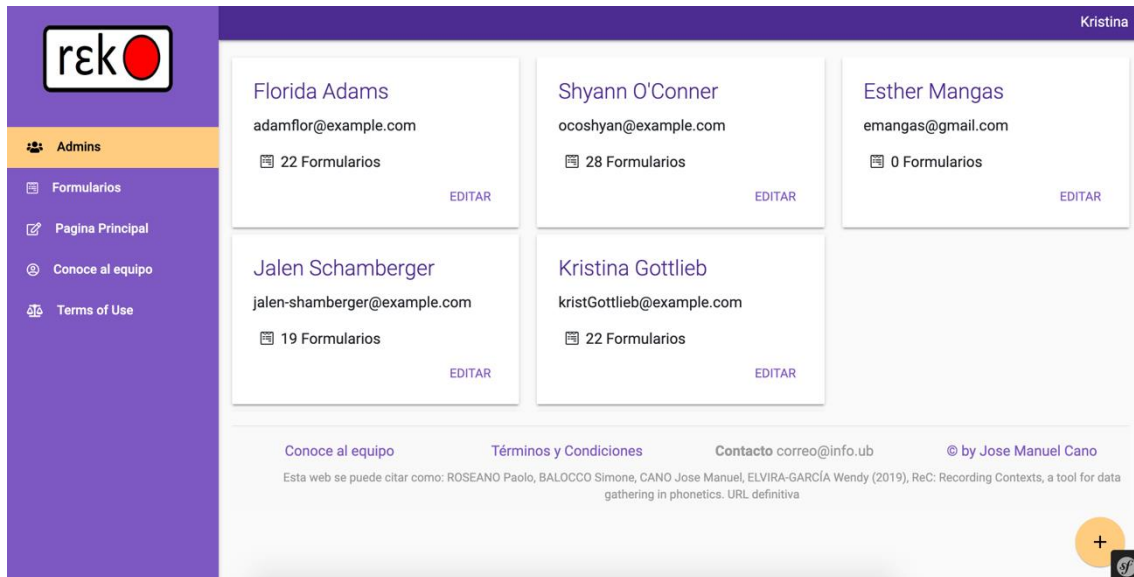


Figure 39 Admins management section.

Clicking on “EDITAR” button on any of the Admin cards or on the bottom-right floating button will launch a modal window to edit or create an Admin respectively.

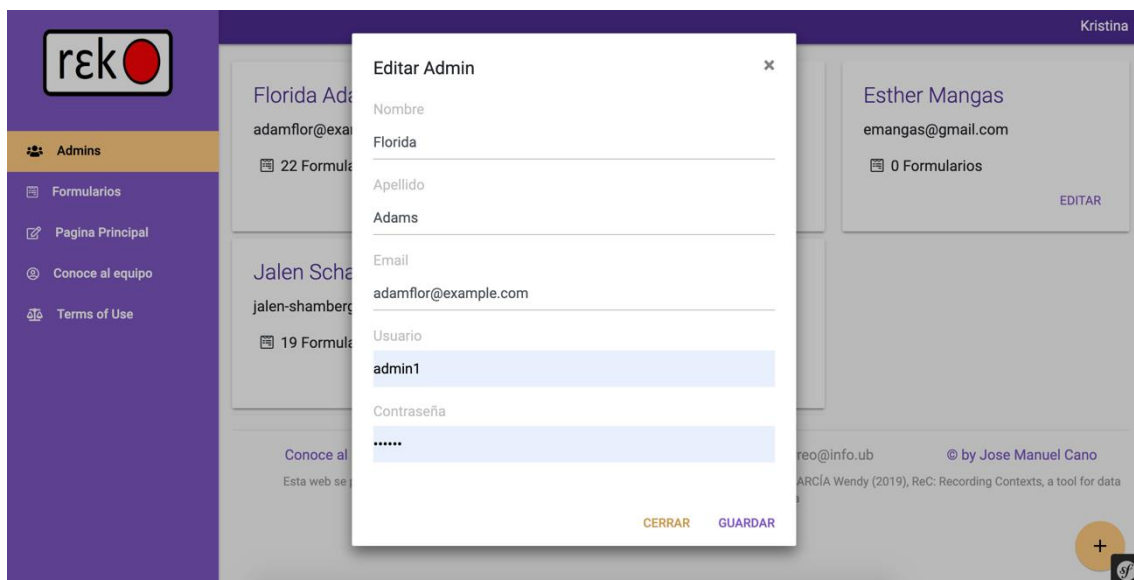


Figure 40 Modal window to create or edit an Admin.

12.1.7 Answer an experiment

In order to access an experiment and be able to answer it, users have to receive a link to it from an Admin. To generate this link Admins can use the option button “URL” in any experiment card to show a modal window that will contain the link.

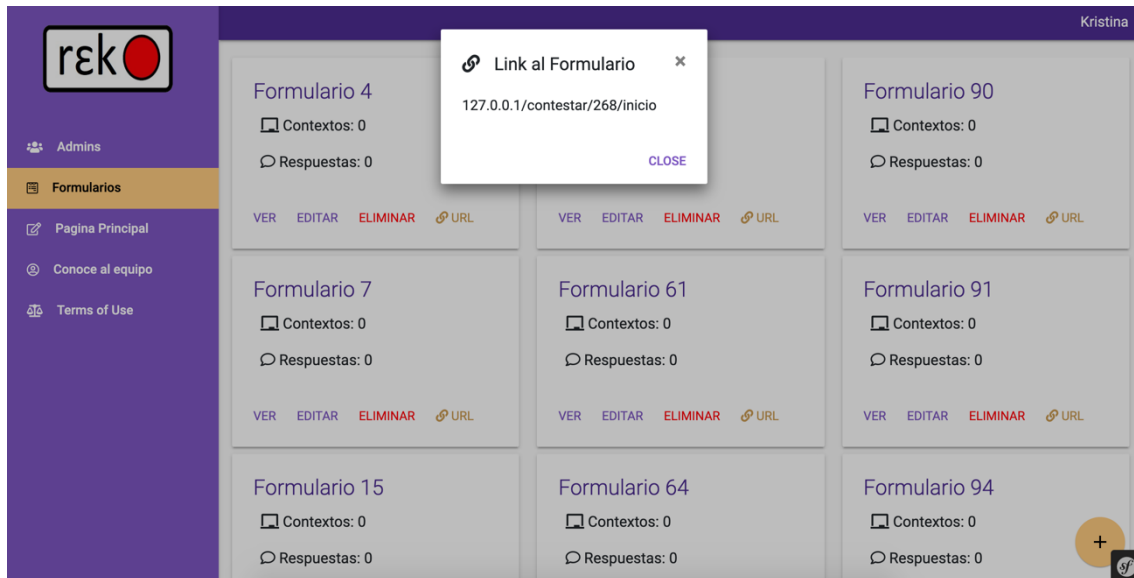


Figure 41 Modal window with a link to an experiment.

After receiving the link to the experiment, users can access Recording Contexts using it and they'll be presented the following page:

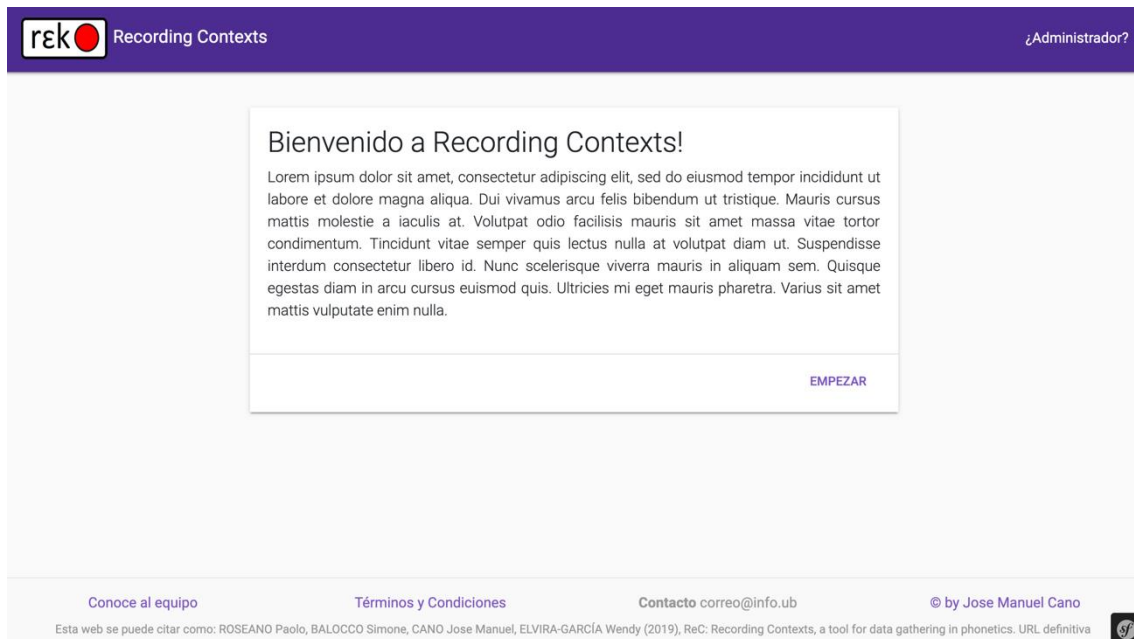


Figure 42 Landing page of an experiment in the final user interface.

The image above shows the Landing Page of the experiment the final user received as a link. After clicking on “EMPEZAR”, a modal window with the Terms of Use of Recording Contexts will appear:

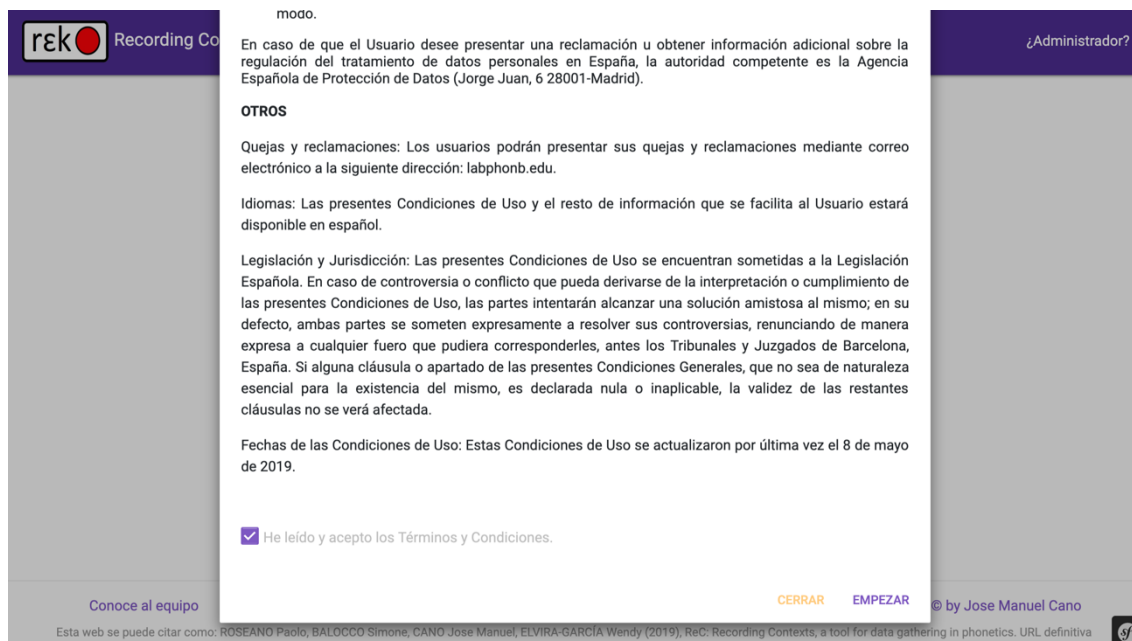


Figure 43 Terms of use modal window.

After accepting the Terms of Use and clicking “EMPEZAR” the experiment will begin.

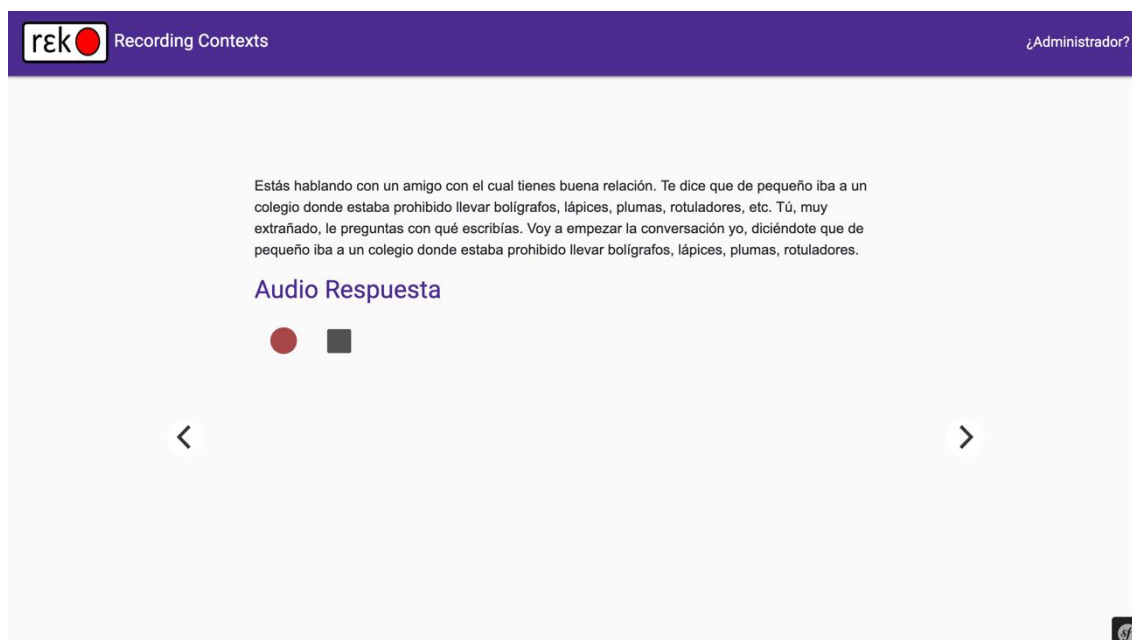


Figure 44 Example of a context when answering an experiment.

As can be seen in the image above, users can record an audio as an answer to each context using the same controls admins use.

If the context is a repetition exercise, the comparison process will be launched upon uploading the audio answer:

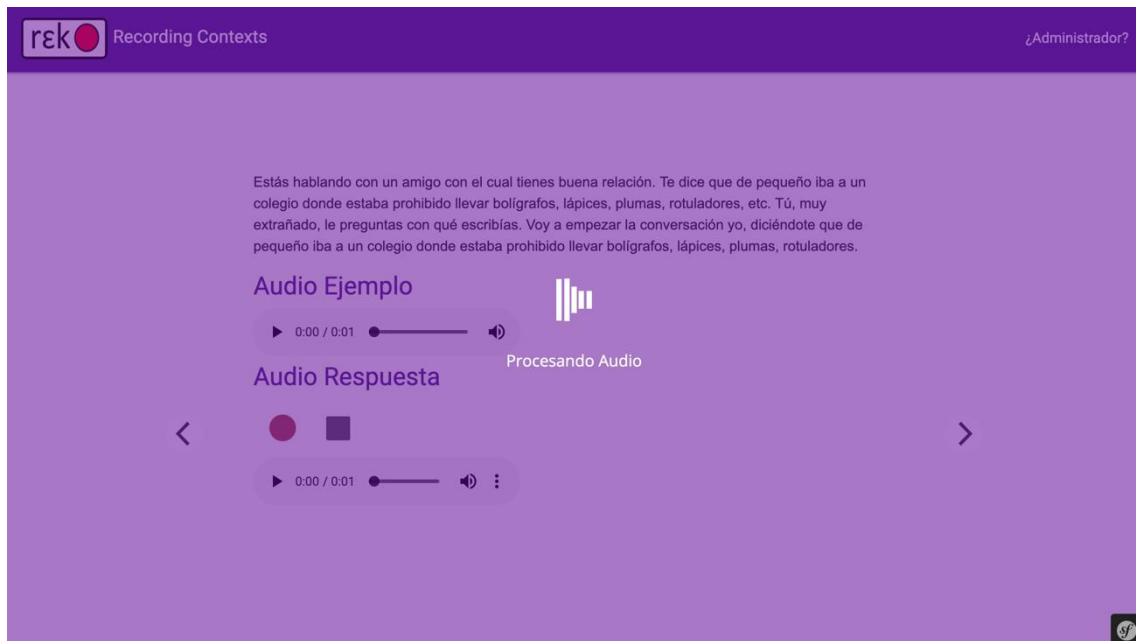


Figure 45 Example of audio comparison when answering a context.

Once the process finishes, a modal window will appear to show the user the results of the comparison. These results are conformed by a score and a line chart illustrating how similar the 2 audios are:

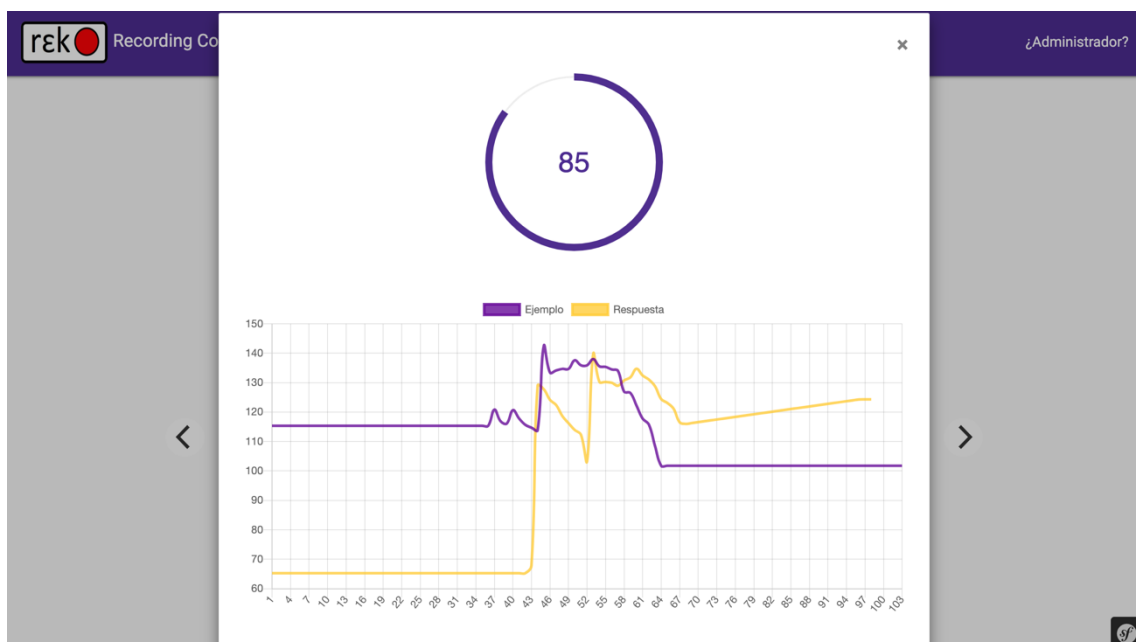


Figure 46 Results of a comparison process between 2 audios.

At the end of the experiment, users will be thanked for participating and asked to fill up a form with data about them. This form is optional and can be skipped if the user does not want to submit his personal data:

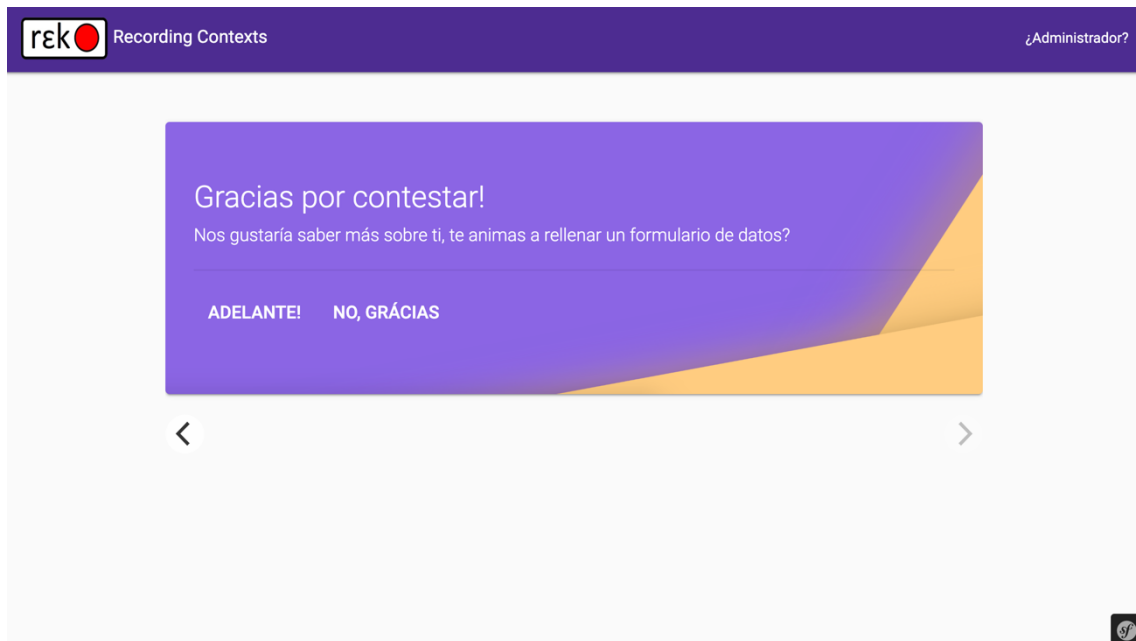


Figure 47 Final section of an experiment.

The screenshot shows a mobile application interface with a purple header bar containing the 'rek Recording Contexts' logo and a link to '¿Administrador?'. The main content area is a form titled 'Datos Personales'. It contains several input fields: 'Fecha de Nacimiento' with a calendar icon, 'Nivel de estudios' with the value 'Sin estudios', 'Pais de nacimiento', 'Ciudad de nacimiento', 'Pais de residencia actual', and 'Ciudad de residencia actual'. At the bottom of the form is a purple 'ENVIAR' button. A small 'sf' logo is in the bottom right corner.

Figure 48 Form to gather personal data about users at the end of an experiment.

12.2 Use cases

12.2.1 Admin use cases

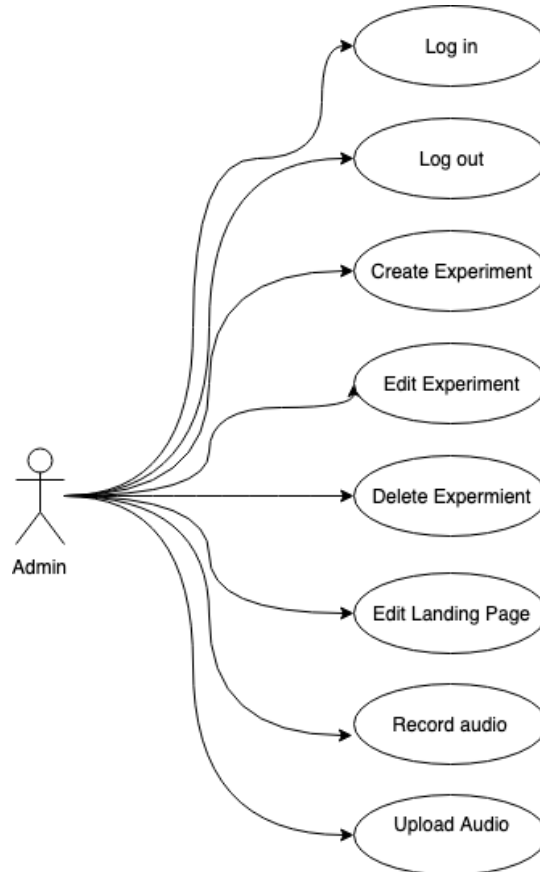


Figure n: Use case diagram of Admin.

UC1. Log in

Objective: Log into the application.

Actor: Admin

Procedure:

1. Admin: Visits Recording Contexts
2. System: Process request.
3. Admin: Clicks on Log In button.
4. System: Redirects Admin to log in form.
5. Admin: Fills the form with username and password and sends the form.

6. System: Validates data.
7. System: Authenticates admin.
8. System: Redirects Admin to his interface.

Extensions:

6. User did not submit correct data. Return to step 4 and show error message.

UC2. Log out

Objective: Log out of the application closing the current session.

Actor: Admin

Procedure:

1. Admin: Visits Recording Contexts
2. System: Process request.
3. Admin: Clicks on Log out button.
4. System: Process request and deletes user session.
5. System: Redirects Admin to homepage.

UC3. Create experiment

Objective: Register an experiment in the application so it can be answered later by a user.

Actor: Admin

<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks new experiment. 4. System: Redirects Admin to experiment creation page. 5. Admin: Names the experiment with a title, creates sections with contexts and sends the data 6. System: Validates data. 7. System: Redirects Admin to homepage interface.
<p>Extensions:</p> <ol style="list-style-type: none"> 6. Admin did not fill the title of the experiment, return to step 5 and show error in form.

UC4. Edit experiment
Objective: Change the content of an already created experiment.
Actor: Admin
<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks edit button of an experiment. 4. System: Redirects Admin to edit experiment page. 5. Admin: Makes any changes to the current experiment and clicks on save button. 6. System: Validates data. 7. System: Redirects Admin to homepage interface.
<p>Extensions:</p> <ol style="list-style-type: none"> 6. Admin did not fill the title of the experiment, return to step 5 and show error in form.

UC5. Delete experiment
Objective: Remove an experiment and its content from the application.
Actor: Admin
<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks delete button of an experiment. 4. System: Show confirmation modal. 5. Admin: Click accept button on modal. 6. System: Delete the experiment.

UC6. Edit landing page
Objective: Change the content of the landing page users will see when they first enter to answer an experiment.
Actor: Admin
<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks landing page section button on the lateral menu. 4. System: Redirects Admin to view landing page. 5. Admin: Clicks edit button. 6. System: Redirects Admin to edit landing page view. 7. Admin: Makes changes to the landing page and clicks on save button. 8. System: Validates data. 9. System: Redirect Admin to landing page view.

UC7. Record Audio
Objective: Record and upload an audio file and attach it to a context of an experiment.
Actor: Administrator
<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks view button of an experiment. 4. System: Redirects Admin to view experiment page. 5. Admin: Clicks record button of any context in the experiment. 6. Admin: Clicks stop button of the previous contexts. 7. System: Receives audio file and stores it.

UC8. Upload audio
Objective: Upload a pre-recorded audio file and attach it to any context of an experiment
Actor: Admin
<p>Procedure:</p> <ol style="list-style-type: none"> 1. Admin: Visits Recording Contexts 2. System: Process request. 3. Admin: Clicks view button of an experiment. 4. System: Redirects Admin to view experiment page. 5. Admin: Clicks upload button of any context in the experiment. 6. Admin: Select file to upload from local system. 7. System: Receives audio file and stores it.

12.2.2 Superadmin use cases

The Superadmin user shares the same use cases with a normal Admin, but has additional cases related to administrators management.

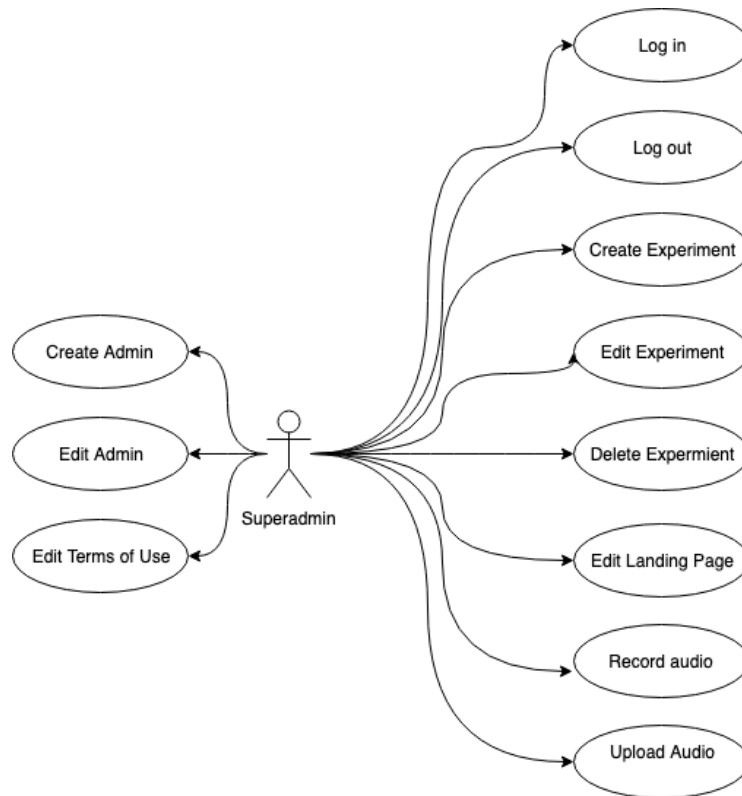


Figure n: Superadmin use case diagram.

UC9. Create Admin

Objective: Add a new Admin to the application.

Actor: Superadmin

Procedure:

1. Superadmin: Visits Recording Contexts
2. System: Process request.
3. Superadmin: Clicks admins section button on the lateral menu.
4. System: Redirects Superadmin to view admins page.
5. Superadmin: Clicks add button.

6. System: Show modal with a form to fill the data of an admin.
7. Superadmin: Fill data of an admin in the form and click on save button.
8. System: Validate data received.
9. System: Dismiss modal and show a successful message.

Extensions:

8. Data was not correct, return to step 6 and show an error message.

UC10. Edit Admin

Objective: Edit any property of an already registered Admin.

Actor: Superadmin

Procedure:

1. Superadmin: Visits Recording Contexts
2. System: Process request.
3. Superadmin: Clicks admins section button on the lateral menu.
4. System: Redirects Superadmin to view admins page.
5. Superadmin: Clicks edit button of an Admin.
6. System: Show modal with a form that contains the data of an Admin.
7. Superadmin: Edit the data and click on save button.
8. System: Validate data received.
9. System: Dismiss modal and show a successful message.

Extensions:

8. Data was not correct, return to step 6 and show an error message.

UC11. Edit Terms of Use

Objective: Change the text of the Terms of Use of the application.

Actor: Superadmin

Procedure:

10. Superadmin: Visits Recording Contexts
11. System: Process request.
12. Superadmin: Clicks Terms of Use section button on the lateral menu.
13. System: Redirects Superadmin to view the Terms of Use page.
14. Superadmin: Clicks edit button.
15. System: Redirect to edit page.
16. Superadmin: Make any changes to the Terms of Use and click on save button.
17. System: Validate data received.
18. System: Redirect to view the Terms of Use page.

12.2.3 User use cases

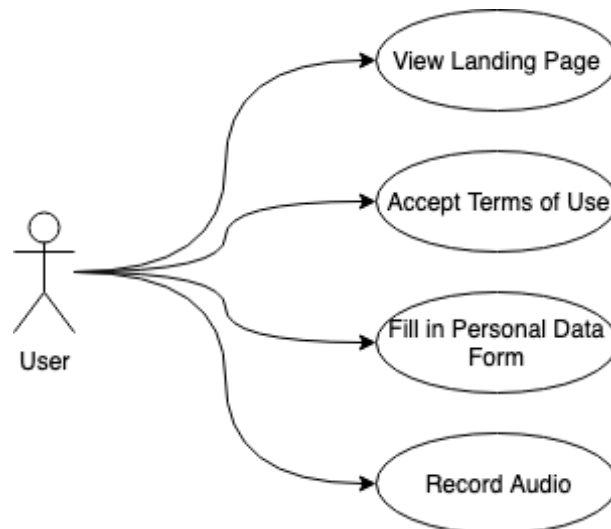


Figure n: User use case diagram.

UC12. View Landing Page

Objective: View the first page of an experiment with an explanation of what it consists of.

Actor: User

Procedure:

1. User: Visits Recording Contexts experiment through the link he receives from an Admin.
2. System: Process request and shows the Landing Page of the experiment.

UC13. Accept Terms of Use

Objective: Accept the Terms of Use of the application when the experiment begins.

Actor: User

Procedure:

1. User: Visits Recording Contexts experiment through the link he receives from an Admin.
2. System: Process request and shows the Landing Page of the experiment.
3. User: Clicks on start experiment button.
4. System: Show modal with the Terms of Use.
5. User: Clicks on accept button.
6. System: Redirects to the experiment page.

UC14. Record Audio

Objective: Record and upload an audio file and attach it as an answer to a context of an experiment.

Actor: User

Procedure:

1. See UC13.
2. User: Click on record button on any of the contexts of an experiment.
3. User: Click on stop button on any of the contexts of an experiment.
4. System: Recieves audio file and stores it.

Extensions:

5. System: If context is of type "Repetition", launch compare Process with the original audio of the context and the answer uploaded by the user.
6. System: Return result of the comparison to the User in a modal.

UC15. Fill in Personal Data form

Objective: Fill a form with personal data about the user at the end of an experiment.

Actor: User

Procedure:

1. See UC14.
2. User: Scroll to end of the experiment and click on the accept button of the explanation section.
3. System: Redirect to user personal data form page.
4. User: Fill the form and click on send button.
5. System: Process data and redirect to "About Us" page.