UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

RECORDING CONTEXTS 2

Autor: Ester Segura Guijarro

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

Barcelona, Juny 20, 2021

Resum

El projecte actual se centra en la millora d'una aplicació web desenvolupada anteriorment que permet la creació de formularis, amb la finalitat de poder recopilar àudios gràcies al sistema de gravació implementat.

L'objectiu principal és posar en funcionament i desplegar una pàgina web afegint noves implementacions a una eina en desús. El segon objectiu tracta del desenvolupament d'un sistema de geo-localització a partir dels resultats obtinguts mitjançant els formularis.

L'anterior projecte havia estat desenvolupat fent ús del framework PHP anomenat Symfony4, i altres tecnologies com HTML, CSS i JavaScript. A causa dels problemes de compatibilitat que van sorgir a l'hora d'implementar totes les millores, es va haver de substituir Symfony4 per un nou framework.

Per tant l'actual projecte s'ha creat utilitzant el framework Django, que permet crear aplicacions web a partir del llenguatge de programació Python. Ha estat desplegat a la plataforma Heroku, i disposa de Google Cloud Storage per poder gestionar els fitxers generats en la resposta dels formularis.

Com a resultat del canvi de framework els objectius s'han reduït, ja que l'aplicació havia de ser creada de nou. Per tant la finalitat del projecte resideix en poder proporcionar una aplicació web allotjada al núvol que permeti obtenir una col·lecció d'àudios per poder ser analitzats.

Resumen

El proyecto actual se centra en la mejora de una aplicación web desarrollada anteriormente que permite la creación de formularios, con la finalidad de poder recopilar audios gracias al sistema de grabación implementado.

El objetivo principal es poner en funcionamiento y desplegar una página web añadiendo nuevas implementaciones a una herramienta en desuso. El segundo objetivo trata sobre el desarrollo de un sistema de geo-localización a partir de los resultados obtenidos mediante los formularios.

El anterior proyecto había sido desarrollado haciendo uso del framework PHP llamado Symfony4, y otras tecnologías como HTML, CSS y JavaScript. A causa de los problemas de compatibilidad que surgieron a la hora de implementar todas las mejoras, se tuvo que substituir Symfony4 por un nuevo framework.

Por lo tanto el proyecto actual se ha creado utilizando el framework Django, que permite crear aplicaciones web a partir del lenguaje de programación Python. Ha sido desplegado en la plataforma Heroku, y dispone de Google Cloud Storage para poder gestionar los ficheros generados en la respuesta de los formularios.

Como resultado del cambio de framework los objetivos se vieron reducidos ya que la aplicación debía ser creada de nuevo. Por lo tanto la finalidad del proyecto reside en poder proporcionar una aplicación web alojada en la nube que permita obtener una colección de audios para poder ser analizados.

Abstract

The current project focuses on improving a previously developed web application that allows the creation of forms, in order to collect audios thanks to the implemented recording system.

The main objective is to get a web page up and running by adding new implementations to a deprecated tool. The second goal is the development of a geo-location system based on the results obtained using the forms.

The previous project had been developed using the PHP framework called Symfony4, and other technologies such as HTML, CSS and JavaScript. Due to the compatibility problems that arose when implementing all the improvements, Symfony4 had to be replaced by a new framework.

Therefore the current project has been created using Django framework, which allows us to create web applications using Python. It has been deployed on Heroku platform, and uses Google Cloud Storage to manage the files generated in the forms response.

As a result of the framework change, the objectives were reduced since the application had to be created again. Consequently, the purpose of the project resides in providing a cloud hosted web application that allows us to obtain a collection of audios to be analyzed.

Index

1 Introducció

La fonètica és la ciència encarregada d'estudiar els sons de la parla. Donat el seu objecte d'estudi, és necessari l'obtenció de dades constituïdes de mostres de parla a través d'una gravació digitalitzada, preferiblement en formats sense compressió com poden ser wav o aiff. Aquests àudios són utilitzats per diferents motius:

- Entendre com funciona la parla.
- Caracteritzar la llengua de grups determinats, com pot ser la llengua dels joves o les característiques presents a les veus dels pacients de Parkinson.
- Entrenar sistemes automàtics de reconeixement de veu o síntesi.

Idealment aquestes mostres són recollides amb micròfons i gravadores d'alta qualitat, i s'enregistren en cabines anecoiques o insonoritzades. La realitat és que avui en dia els estudis requereixen un major nombre de dades, provocant que la recollida de dades en línia s'hagi convertit en una alternativa habitual. Això s'ha fet més patent amb la situació sanitària actual, ja que les cabines que permeten gravar de forma presencial només es poden fer servir una vegada al dia, i algunes universitats encara no han aprovat protocols per ser utilitzades. A causa d'aquest context, la recollida de dades en línia esdevé l'opció preferida per fonetistes de tot el món que busquen la manera més eficaç d'enregistrar les seves dades.

Aquest projecte presenta una eina que permet al Laboratori de Fonètica de la Universitat de Barcelona la creació de formularis que estan formats per diferents contextos. Això fa possible la recollida de dades en línia, ja que cada context proporciona una informació que serveix perquè l'usuari pugui enregistrar els seus propis àudios sense haver de fer-ho presencialment.

1.1 Objectius

L'objectiu principal és posar en funcionament, actualitzar i desplegar una eina en desús desenvolupada anteriorment en un treball de fi de grau. Inicialment havia de canviar-se el framework Symfony per Laravel i desplegar l'aplicació implementada, a més de portar a terme un objectiu secundari que consistia en la creació d'un sistema de geo-localització.

El framework proposat i les versions utilitzades generaven moltes dependències que no podien ser resoltes. Això va comportar una modificació dels objectius, ja que es va decidir implementar de nou una aplicació en Django per poder ser desplegada al núvol amb èxit. Per tant l'objectiu final del projecte consisteix en el desenvolupament d'una aplicació web amb les funcionalitats requerides que permeti ser desplegada i accessible per tothom.

1.2 Estructura de la memòria

La memòria comença amb una l'explicació de dues planificacions, una corresponent als objectius inicials proposats, i l'altra que correspon als objectius un cop es pren la decisió del canvi de framework a Django.

Seguidament, a l'apartat d'antecedents, es parla de l'estat en el qual estava el projecte anterior, explicant les tecnologies que s'havien utilitzat i les funcionalitats implementades. A més s'explica el problema trobat i perquè es pren la decisió d'utilitzar Django.

Un cop vist tot això, es presenten els requisits de l'aplicació i quines tecnologies seran utilitzades per complir l'objectiu del projecte. A més es parla de com s'ha estructurat tant el codi com la base de dades utilitzada, i quin ha estat el flux seguit a l'hora de fer el desplegament de l'aplicació.

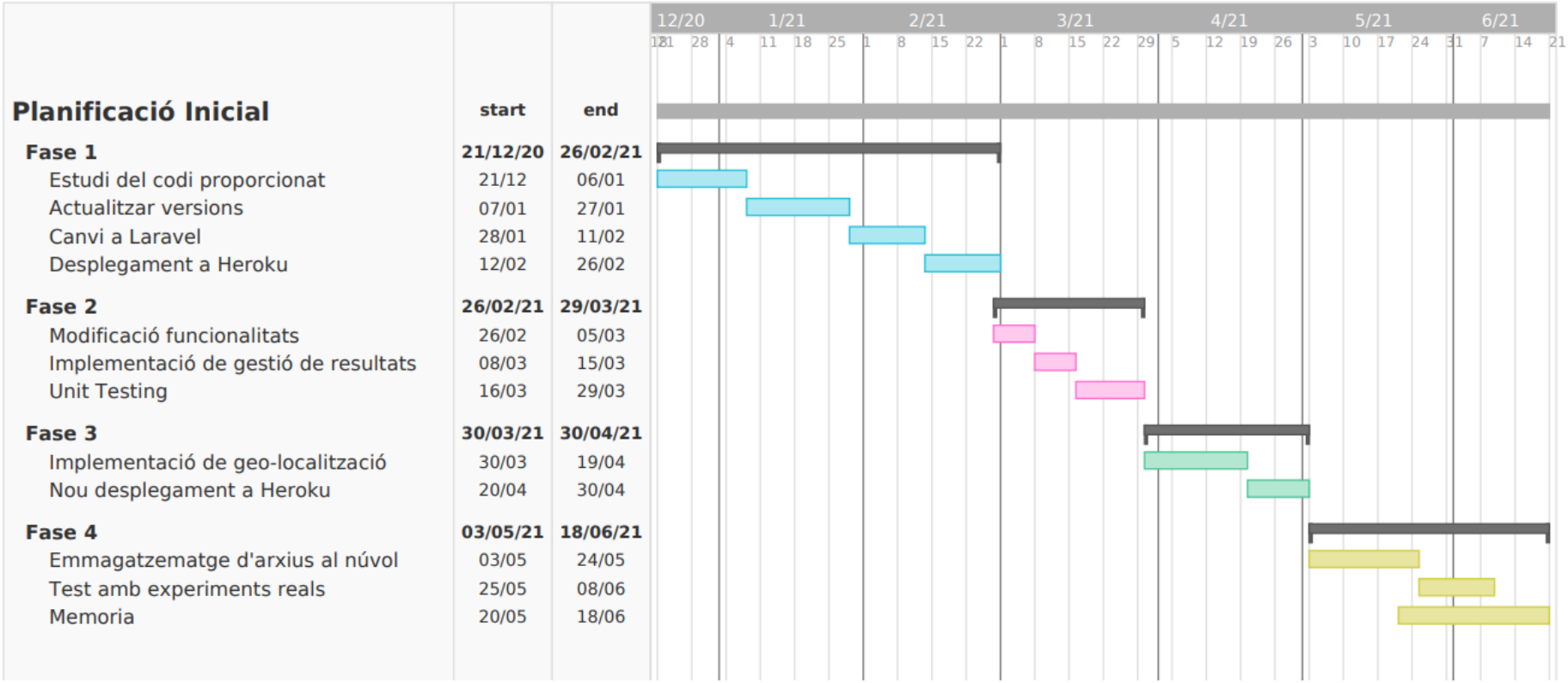
Finalment es mostren les proves realitzades i els resultats obtinguts, i es fa una valoració de com ha transcorregut la implementació del projecte, i quines millores es poden fer de cara al futur.

2 Planificació inicial

Aquesta planificació correspon a la proposta inicial del projecte i es divideix en 4 fases diferenciades:

- **FASE 1:** Definida per l'estudi del codi proporcionat i l'actualització de versions. A més en aquesta fase entra el canvi de framework de Symfony4 a Laravel. I finalment un desplegament per comprovar que l'objectiu principal és assolible.
- **FASE 2:** Aquí s'han de realitzar totes les modificacions de funcionalitats necessàries que el Laboratori de Fonètica requereix. A més s'ha de definir un sistema de retorn de resultats en format zip, que contigui els àudios gravats pels usuaris i un excel amb les dades recollides.
- **FASE 3:** L'objectiu secundari està definit en aquesta fase. És a dir, en aquesta etapa s'ha de desenvolupar el sistema de geo-localització requerit. A més el codi ha de ser desplegat amb la nova versió de l'aplicació web.
- **FASE 4:** Marcada per la recerca de solucions davant el problema de gestió de fitxers present a Heroku, del qual parlarem més endavant. Aquí és on s'implementa el sistema d'emmagatzematge al núvol. També es porta a terme l'última fase de testing amb experiments reals, juntament amb el desenvolupament de la memòria.

Degut a la complicació del desplegament a partir del codi proporcionat, es va decidir canviar d'estratègia i utilitzar un framework nou. Això dóna pas a una nova planificació vista al següent apartat.



Imatge 1: Primera planificació

3 Planificació final

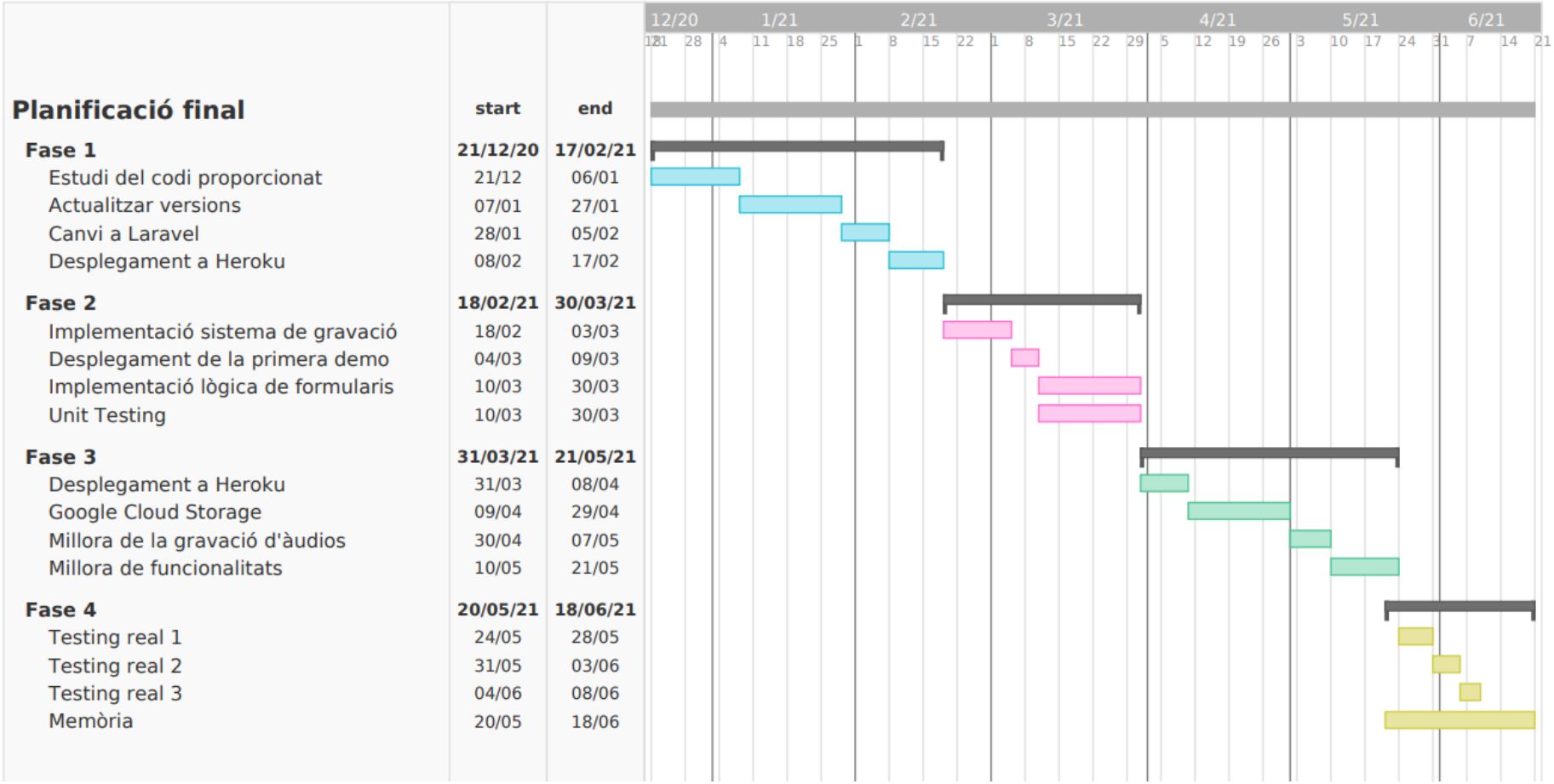
El temps que s'havia d'invertir a crear una nova aplicació web a partir d'un altre framework va fer que la planificació canvies. És a dir l'objectiu secundari no podia realitzar-se ja que el projecte havia de centrar-se en assolir l'objectiu principal, i per això es necessitava més temps que el planejat inicialment. Per tant va sorgir una nova planificació, que també consta de 4 fases diferenciades:

- **FASE 1:** És semblant a la primera fase de la planificació inicial. Se centra a estudiar el codi proporcionat i actualitzar els components que el formen. A més en aquesta etapa es canvia a Laravel i s'intenta fer un desplegament a Heroku sense èxit. Això dóna pas a una nova fase que vindrà marcada pel canvi de framework a Django.

- **FASE 2:** Aquí es defineix el sistema de gravació d'àudios que serà utilitzat per la generació dels fitxers. Per poder comprovar que l'objectiu principal és assolible es crea una demo de Django amb les funcionalitats essencials del projecte, i seguidament es fa un desplegament. Un cop fet això, es procedeix a la creació de la lògica de formularis juntament amb el testing unitari.

- **FASE 3:** Aquesta etapa està marcada pel Cloud Computing. Es fa el desplegament de l'aplicació funcional a Heroku i a més es realitza la recerca per veure com es pot gestionar la problemàtica que presenta Heroku amb els fitxers estàtics. Finalment s'acaba optant pel sistema d'emmagatzematge Google Cloud Storage. Tot això fa que s'hagin de fer millores en el sistema de gravació i en algunes funcionalitats ja implementades com són la gestió i el guardat d'àudios.

- **FASE 4:** Finalment en aquesta última etapa es realitzen una sèrie de proves amb experiments reals per comprovar la funcionalitat de l'aplicació desplegada. Totes aquestes proves fan que algunes de les funcions hagin de ser canviades i adaptades als requeriments del Laboratori de Fonètica. Tot això es fa de manera conjunta amb el desenvolupament de la memòria.

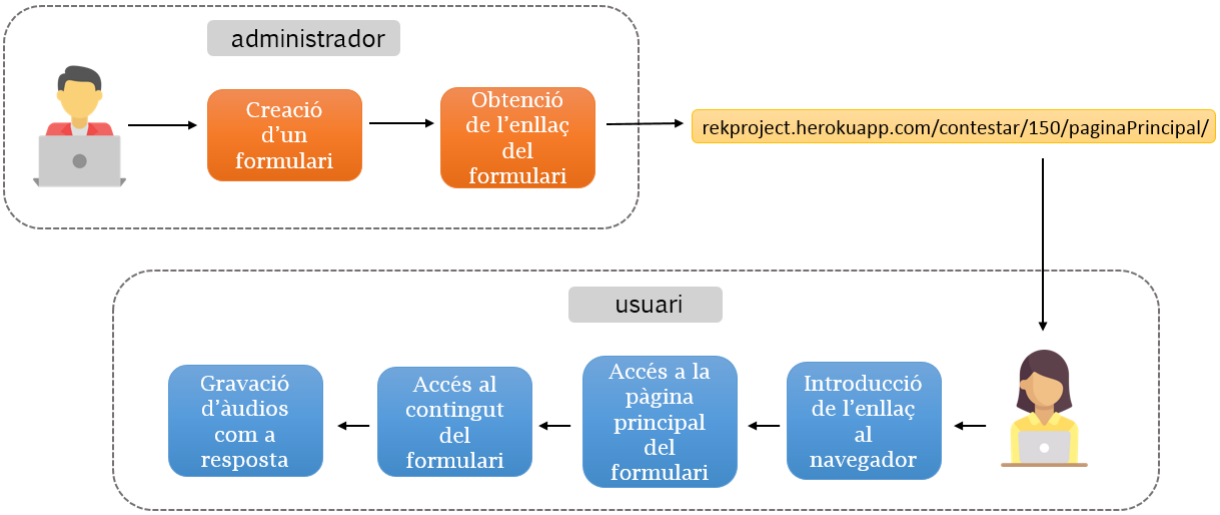


Imatge 2: Planificació final

4 Antecedents

4.1 Estat del projecte

El projecte implementat anteriorment, i a partir del qual haig de realitzar el meu treball, disposa d'un sistema de gestió de comptes que permet la creació de diversos administradors. Cada un d'ells pot generar formularis que contenen diferents contextos explicatius. Un cop creat el formulari, la pàgina proporciona un enllaç a l'administrador que serveix perquè diferents persones accedeixin al contingut sense necessitat de tenir cap sessió iniciada. Quan l'usuari introdueix l'enllaç al navegador, es mostra una pàgina principal que explica quina és la motivació del formulari. Seguidament s'accedeix al contingut per poder respondre cada un dels contextos a través d'àudios. Aquest procés es pot veure il·lustrat a la següent imatge:



Imatge 3: Procés de creació i resposta de formularis

Tot i així l'aplicació web desenvolupada anteriorment no és utilitzada pel Laboratori de Fonètica, ja que requereix certes millores per poder ser emprada a l'hora de realitzar investigacions.

El principal problema resideix en el fet que l'aplicació només és funcional si s'està executant en un servidor web local. El desplegament al núvol que es va fer no permet que els usuaris puguin accedir al formulari a través de l'enllaç proporcionat per l'administrador. Això fa que sigui impossible que els investigadors obtinguin resultats. A més no s'ha implementat cap sistema de recuperació de resultats en forma d'Excel, que és com ho requereixen els membres del Laboratori de Fonètica.

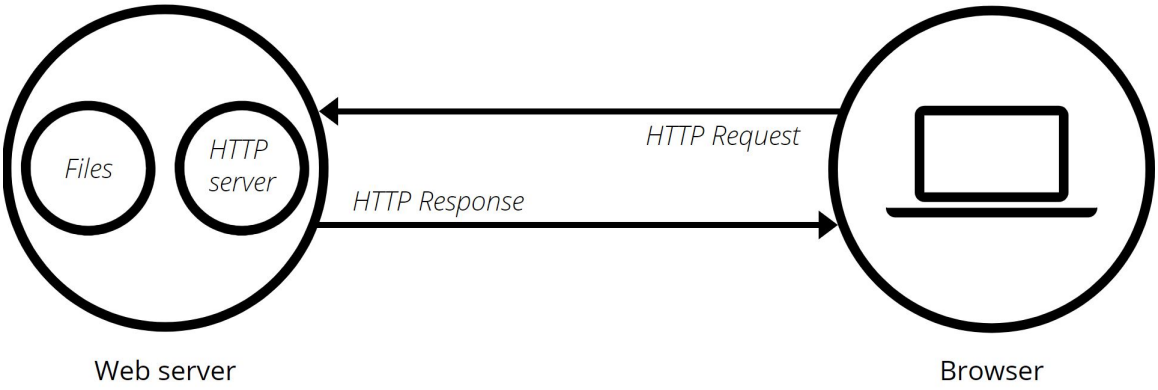
Per tant, el projecte que es va implementar compleix certs requisits

indispensables, però està en desús a causa del funcionament de l'aplicació web en un servidor allotjat al núvol.

4.2 Tecnologies utilitzades

4.2.1 Servidor web i base de dades

Un servidor web és el que s'encarrega de distribuir les pàgines als usuaris en el moment que són requerides. Quan una aplicació web s'està executant en un servidor, i el navegador requereix un fitxer que es troba al servidor web utilitzat, el mateix navegador envia una sol·licitud via HTTP. En el moment que això li arriba al servidor, accepta la sol·licitud i a través d'HTTP envia el fitxer necessari al navegador. Dintre d'aquest procés es poden produir errors, com per exemple que el servidor no sigui capaç de trobar el fitxer demanat. Això donaria com a resposta un error 404 que seria enviat al navegador.



Imatge 4: Funcionament d'un servidor

El servidor utilitzat en el projecte que es va desenvolupar és Apache [2], un servidor web HTTP de codi obert i multiplataforma que funciona tant a Unix com a Windows. Tot i que li diem servidor web, Apache en realitat és un software que s'executa en un servidor, que té com a objectiu extreure el contingut demanat en cada sol·licitud que l'usuari fa i enviar-lo a la web corresponent.

Apache disposa de mòduls que permeten activar o desactivar funcionalitats, això fa que sigui personalitzable. Aquests mòduls, entre altres, poden ser d'emmagatzematge a la memòria cau o de seguretat.

Perquè l'aplicació web realitzi les funcions necessàries, s'ha de gestionar de forma correcta la connexió amb una base de dades, la recuperació de la informació, i la inserció de noves dades. Per poder realitzar tota aquesta gestió, el projecte utilitza PostgreSQL, una base de dades relacional de codi obert. L'avantatge que

proporciona és que ofereix una gran escalabilitat, és a dir pot adaptar-se a les capacitats del hardware de forma òptima.

4.2.2 Tecnologies Front-end

En relació a la part de l'aplicació amb la que els usuaris poden interactuar, es van utilitzat diverses tecnologies.

Com a base s'hi va utilitzar HTML, que permet definir l'estructura que tindrà la web. Per poder fer més presentable tot el que l'usuari veu, es va fer servir CSS, un llenguatge d'estil que a través de diferents fulles dóna l'aparença a la pàgina que visualitzem. A més perquè l'aplicació web sigui responsive, es va utilitzar Bootstrap Material Design, un framework que correspon a una extensió de Bootstrap CSS.

Per poder crear una web amb més dinamisme també es va fer servir Javascript, un llenguatge de programació que permet la gestió d'esdeveniments i la modificació dels elements que formen l'estructura de la pàgina. A més, el sistema de gravació implementat va ser creat utilitzant el llenguatge Javascript, que permet l'ús d'un objecte anomenat Recorder utilitzant una llibreria creada per Matt Diamond [5].

4.2.3 Cloud computing

Per fer el desplegament al núvol es va optar per utilitzar Amazon Web Services (AWS) [6], una plataforma que permet pujar el codi en una infraestructura proporcionada i així poder tenir una aplicació web accesible per tothom.

AWS ofereix un ampli servei de monitoratge de la web. A més inclou diferents aplicacions per poder configurar el projecte. En aquest cas es va utilitzar S3 Bucket per emmagatzemar els fitxers estàtics al núvol.

4.3 Symfony

Respecte al Back-end, el projecte que es va desenvolupar està programat en el llenguatge PHP, i el framework utilitzat és Symfony.

4.3.1 Què és un framework?

Un framework és una estructura de programari que permet desenvolupar aplicacions software. Conté diferents classes i funcions que poden ser utilitzades a l'hora de crear un programa en una plataforma concreta. Els frameworks poden gestionar les connexions a la base de dades, processar els inputs i a més

proporcionar una estructura definida per a la implementació.

Per tant un dels avantatges d'utilitzar un framework a l'hora de desenvolupar aplicacions és una millor optimització del temps, ja que al contenir classes i diferents funcions que es poden utilitzar de manera repetida, evita al programador haver d'implementar diferents cops un codi semblant. És a dir, permet que el desenvolupament se centri en la construcció de codi de més alt nivell, gestionant de manera interna la programació a més baix nivell. A més la majoria de frameworks proporcionen un esquelet detallat, per tant el programador tampoc necessita invertir temps a crear l'estructura necessària per a poder desenvolupar el projecte.

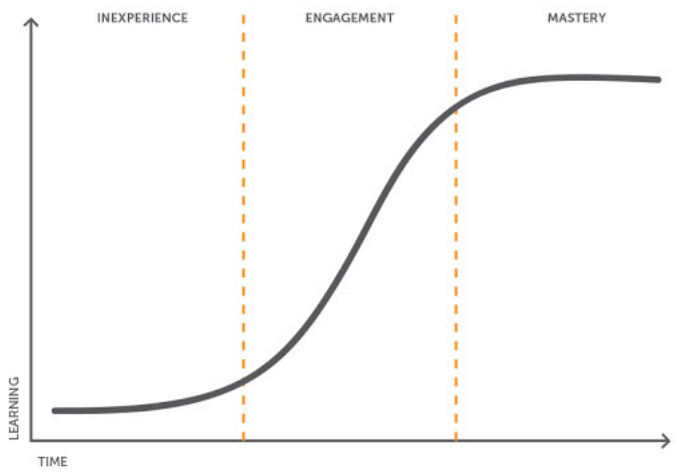
4.3.2 Com escollir un framework?

Actualment existeixen molts frameworks, per tant a l'hora d'escollir un s'han de tenir en compte diferents factors.

Corba d'aprenentatge

A l'hora d'escollir un framework que mai hem tractat, una de les principals coses que hem de valorar és la seva corba d'aprenentatge. Aquest terme fa referència al progrés d'aprenentatge que es produirà respecte al temps.

En la imatge de més avall podem observar un exemple de corba d'aprenentatge. L'eix horitzontal indica el transcurs del temps, i l'eix vertical és el progrés que s'ha de fer per assolir els coneixements. Com menys temps triguem a avançar en la formació per saber com s'utilitza i com està estructurat el framework, millor serà la corba d'aprenentatge.



Imatge 5: Corba d'aprenentatge de diferents Frameworks

Documentació

És important que el framework consti de documentació suficient ja que facilitarà la feina a l'hora de desenvolupar. Una bona documentació estalviarà temps quan s'hagin de buscar possibles solucions a un error concret. A més, si hi ha més informació sobre un framework, la corba d'aprenentatge serà millor ja que disposem de més documents per poder aprendre a utilitzar-lo.

Aquest punt va lligat a la documentació que els propis creadors del framework aporten, però també té a veure amb la quantitat de gent que l'utilitza. Si el nombre de persones que fan ús d'ell és més elevat, més fàcil serà trobar informació, idees d'implementació, o fins i tot llibreries exclusives creades de forma externa al framework.

Escalabilitat, rendiment i velocitat

Quan el nivell de tràfic d'una web augmenta, és important escollir un framework que sigui capaç d'adaptar-se a aquestes condicions mantenint la seva funcionalitat.

En el cas d'una aplicació web és difícil saber quin serà el nombre d'usuaris que accedeixin als serveis que proporciona. El framework ha de poder recuperar els arxius en el temps més gran possible sense que això es vegi afectat pel tràfic. A més la connexió a la base de dades i l'ús dels recursos hardware que tenim disponibles tampoc ha de veure's afectat. Per tant, busquem un framework que mantingui una relació el més lineal possible entre el nombre de connexions i el temps emprat en proveir els arxius necessaris.

Seguretat

Una aplicació web pot ser potencialment vulnerable i està exposada a una gran quantitat de persones. És per això que la seguretat és un factor molt important a tenir en compte. Per tant, necessitem que un framework disposi de funcions de seguretat que ajudi a la nostra aplicació web a protegir-se d'atacs com poden ser scripts que són inserits al client, o intents d'executar determinades accions a través de les credencials d'un altre usuari.

Hosting

Perquè la nostra aplicació web sigui accessible per tothom, hem de poder desplegar-la al núvol. En cas que no tinguem un servidor propi on executar el projecte, s'han de buscar diferents alternatives en plataformes que permetin pujar la nostra aplicació. Per tant el framework ha de ser compatible amb algunes de les plataformes d'allotjament i poder proporcionar una fàcil configuració.

4.3.3 Com funciona Symfony?

Symfony és un framework PHP desenvolupat per crear aplicacions web que està format per un conjunt de components PHP reutilitzables. Tots els components són llibreries independents, però tot això en conjunt permet a Symfony ser capaç de realitzar les funcions necessàries a l'hora de crear una web. Per poder integrar tots aquests components dins de l'aplicació, existeixen els bundles.

Symfony està basat en el patró d'arquitectura de software Model-Vista-Controlador. Això permet separar les dades amb el que el client veu a la interfície. El Model és qui conté totes les dades de l'aplicació i és l'encarregat de controlar la seva persistència. En canvi la Vista correspon a la informació que el Model envia al client, i compon la interfície d'usuari. La lògica de connexió i el que s'encarrega de controlar tot el flux d'informació entre el Model i la Vista, en aquest cas és el Controlador.

4.4 Laravel i desplegament

L'objectiu principal del projecte és poder solucionar els errors que la versió anterior de l'aplicació web tenia, i a més fer un desplegament efectiu al núvol. Per poder fer això vam decidir allotjar el projecte a la plataforma de cloud computing anomenada Heroku (veure secció 6.4.1). A causa de la poca compatibilitat de Heroku amb Symfony, es va decidir canviar al framework Laravel.

Laravel és un framework que permet desenvolupar aplicacions web amb PHP utilitzant l'arquitectura MVC. Utilitza diferents components d'altres frameworks, i una gran part del codi l'extreu de Symfony. Per tant es va escollir Laravel, ja que fa servir moltes de les llibreries que també utilitzava Symfony, i a més és compatible amb la plataforma Heroku.

4.5 Limitacions trobades

Per poder desplegar l'aplicació al núvol primer havia d'actualitzar la versió de PHP, ja que hi havia algunes llibreries obsoletes. Per tant la versió a la qual vaig passar va ser a la 7.2.24.

Seguidament havia de canviar el framework a Laravel, però hi havia una gran quantitat de dependències que no es podien resoldre i això no permetia que pogués avançar en la realització de l'objectiu principal. Per tant es va prendre la decisió de canviar a un altre framework i, aprofitant parts del codi que hi havia de l'anterior projecte, crear una nova aplicació web sense errors de compatibilitat. Vaig decidir que el nou framework a utilitzar seria Django (veure secció 7) degut a l'experiència que he obtingut arran de desenvolupar diversos projectes amb aquest

framework.

Django s'utilitza per desenvolupar aplicacions web a través del llenguatge de programació Python. La utilització de Django proporcionava una llista d'avantatges al projecte:

- **Corba d'aprenentatge:** Django té una corba d'aprenentatge relativament curta, sobretot si el desenvolupador ha tractat amb altres frameworks semblants com Flask.
- **Documentació:** És un framework molt utilitzat, això suposa una gran quantitat d'informació disponible per diferents desenvolupadors. A més Django disposa d'una pàgina pròpia que proporciona als usuaris documentació i guies d'implementació.
- **Escalabilitat i rapidesa:** És realment escalable i a més permet l'eliminació o inclús la instal·lació de diferents plugins per tal d'afegir noves funcionalitats a l'aplicació creada.
- **Seguretat:** Django ofereix de manera predeterminada diferents funcions de seguretat que fan que la web implementada disposi d'un sistema de seguretat.
- **Hosting:** El framework és totalment compatible amb Heroku, a més la mateixa plataforma d'allotjament disposa d'una guia per poder desplegar aplicacions web amb Django [25].

Tot i els avantatges d'aquest framework, per poder canviar d'una manera definitiva de Symfony a Django, aquest últim framework havia de proporcionar dues funcionalitats bàsiques per poder acomplir el projecte amb èxit:

- **Creació de formularis:** La creació de formularis és una funcionalitat bàsica de l'aplicació, per tant el nou framework havia de proporcionar una eina per la seva creació. Django té la seva pròpia implementació dels formularis que facilita aquesta funcionalitat.
- **Gravació de veu i recuperació d'arxius:** En el projecte la gravació es fa amb Javascript, per tant no hi havia cap problema amb el canvi al nou framework, ja que era feina del Front-End. I respecte a la recuperació d'arxius, Django proporciona un sistema de gestió de fitxers amb emmagatzemament propi.

Per tant un cop comprovat que els requisits per aquest projecte eren totalment compatibles amb Django, es va decidir implementar una nova aplicació web amb aquest framework.

5 Anàlisi de requisits

El principal requisit que l'aplicació web ha de proporcionar és la creació de formularis, que seran útils al Laboratori de Fonètica per portar a terme les investigacions. Per això és necessari disposar d'un sistema de gravació d'àudios que permeti la creació dels formularis i posteriorment la resposta per part dels usuaris. Aquests àudios formen part del que anomenem contextos, on l'investigador exposa una situació que l'usuari ha de respondre. Per tant els formularis es compondran d'aquesta estructura:

- **Pàgina Principal** que es mostra a l'inici del formulari i serveix a l'administrador per exposar el motiu de l'experiment.
- **Dades personals** que varien segons l'interès de l'investigador. Els tres tipus de dades que es poden demanar són:
  - Text: La resposta de l'usuari serà una string
  - Data: L'usuari ha d'introduir una data en format DD/MM/YYYY.
  - Resposta de Si/No: Se li farà una pregunta a l'usuari i a través d'un Radio Button haurà de marcar l'opció que ell cregui corresponent.
- **Contextos** que fan referència als apartats on l'usuari ha de respondre a través d'àudios. Els diferents tipus de context que l'administrador pot utilitzar són:
  - Text: Se li proporcionarà una frase a l'usuari i a través d'això ha de ser capaç de respondre.
  - Àudio: L'administrador gravarà un àudio i l'usuari ha de poder repetir el que l'investigador ha dit.
  - Imatge: El context es compondrà d'una imatge il·lustrativa que permetrà a l'usuari respondre.

Respecte a els àudios que es gravaran als contextos, es poden classificar en dos tipus:

- Repetició de paraules o frases
- Proposta de situacions

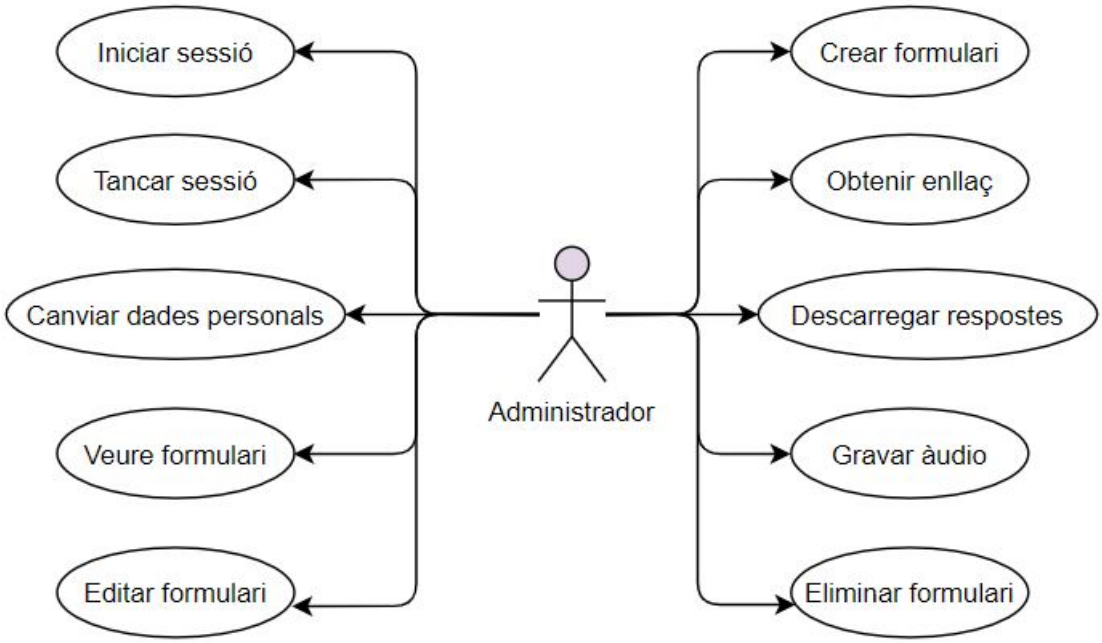
5.1 Requisits funcionals

A partir dels requisits exposats anteriorment podem extreure dos tipus d'usuaris:

- **Administrador:** Usuari registrat que podrà crear els formularis corresponents per dur a terme investigacions.
- **Usuari no registrat:** No tindran cap sessió per poder accedir a la creació de formularis. Seran els que responguin els experiments creats a través de l'enllaç proporcionat per l'administrador.

5.1.1 Interfície d'administrador

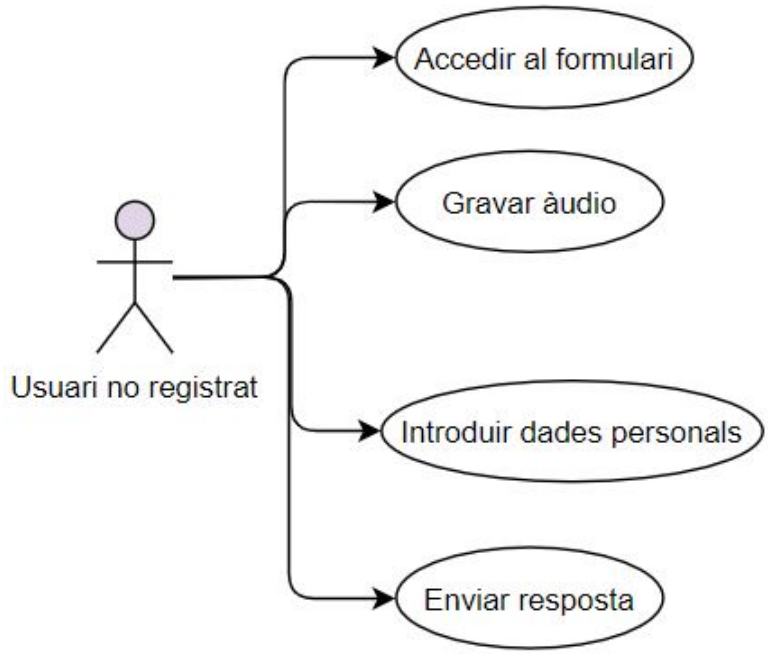
L'aplicació web només consta d'un administrador que té els privilegis de superuser. És l'encarregat de crear els formularis i realitzar diferents operacions sobre ells per a l'obtenció dels resultats. A la següent figura es mostren els diferents casos disponibles per l'administrador, explicats a la secció 13.1.



Imatge 6: Casos d'ús de l'administrador

5.1.2 Interfície d'usuari no registrat

En el cas de l'usuari que respon mitjançant l'enllaç proporcionat, no té cap accés a la creació ni la manipulació dels formularis. Simplement respon a partir d'àudios que ha de gravar segons el context descrit per l'investigador. A continuació es mostren els casos d'ús explicats a la secció 13.1.



Imatge 7: Casos d'ús de l'usuari sense registrar

5.2 Requisits no funcionals

5.2.1 Disponibilitat

L'aplicació web ha d'estar disponible la major part del temps. És a dir, qualsevol persona ha de poder realitzar les funcions específiques de la pàgina segons el seu rol d'usuari en qualsevol moment. A més la disponibilitat de la pàgina no ha de dependre de la localització geogràfica de l'usuari que l'utilitza, ja que la motivació del projecte resideix en que persones de diferents localitats puguin respondre mitjançant la gravació d'àudios.

5.2.2 Accessibilitat

El dispositiu sobre el qual s'està utilitzant la web no ha de ser un factor decisiu a l'hora de poder manipular o respondre formularis. És a dir, la pàgina ha de poder ser accessible des de qualsevol dispositiu, sigui un mòvil o un ordinador. Aquest punt determina que l'aplicació web ha de disposar d'un disseny adaptatiu per poder oferir una correcta visualització als usuaris.

5.2.3 Usabilitat

A causa de la varietat d'usuaris que respondran als formularis, la pàgina ha de poder proporcionar un disseny comprensible per totes les persones que responguin. A més obliga a la web a presentar una estructura on tota la informació estigui en un

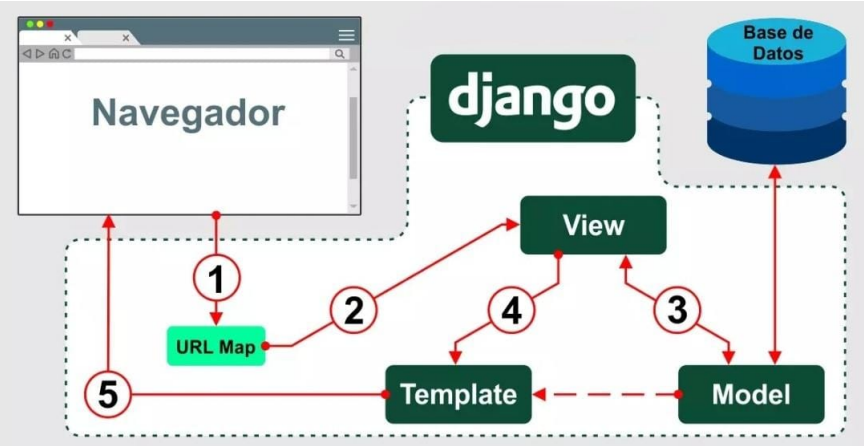
idioma internacional. És per això que la página de resposta està escrita en anglès, ja que és la pàgina a la que accediran els usuaris. Tot i així els investigadors poden escriure els contextos i les dades personals en l'idioma que vulguin tenint en compte a qui va adreçat el formulari.

5.2.4 Escalabilitat i rapidesa

L'aplicació web ha de mantenir un funcionament correcte independentment del nombre de persones que l'estiguin utilitzant. A més ha de proporcionar una estructura adient per la implementació de futures versions sense que això afecti el seu rendiment.

6 Tecnologies utilitzades

El projecte està desenvolupat seguint l'arquitectura Model-Vista-Controlador. Aquesta arquitectura segueix un flux concret que s'il·lustra a la següent imatge:



Imatge 8: Arquitectura

1. El navegador envia una petició que a partir dels protocols arriba al nostre servidor.
2. Aquesta petició un cop arriba crida a Django, que processa l'URL i crida al controlador anomenat View.
3. Un cop al controlador, aquest interactua amb el model per realitzar una operació concreta, com pot ser obtenir un valor de la base de dades.
4. Després que el model hagi realitzat les operacions necessàries, el controlador crida als Templates, que són els que formen la nostra vista.
5. Finalment la vista renderitza la resposta a la sol·licitud que el navegador havia generat.

Per tant el controlador en aquest cas és la View, ja que és qui decideix quines dades seran presentades a l'usuari. I en canvi la forma en la que es presenta ve definit pels Templates. És per això que Django també es pot considerar un framework MTV (Model-Template-View).

6.1 Tecnologies Front-end

Pel que respecta a la part visible per l'usuari, l'estructura semàntica s'ha creat a partir d'HTML. A través d'un sistema d'etiquetes, HTML s'encarrega de definir el contingut de les pàgines que veiem, i això ens permet generar diferents elements com poden ser textos o imatges.

Per poder complementar HTML i oferir un disseny diferent de l'estàndard s'utilitza el conegut llenguatge de fulles d'estil en cascada CSS. És utilitzat ja que HTML no proporciona una eina per poder canviar l'aparença de la pàgina més enllà de modificar l'estructura dels elements. És per això que és útil l'aplicació de CSS per poder implementar un nou disseny a la nostra web a través de fulles d'estil.

El dinamisme necessari per al funcionament eficient de la pàgina web és creat a partir de JavaScript, que permet la gestió d'esdeveniments i la creació d'efectes. És a dir quan un usuari prem un botó, aquest genera un esdeveniment que és capturat per JavaScript, qui s'encarrega de gestionar les operacions a fer.

6.1.1 Bootstrap

Per complementar el disseny de la pàgina, utilitzo el framework Bootstrap, que permet crear pàgines adaptables independentment del dispositiu i de la mida de la pantalla on es visualitza. Conté un sistema de quadrícules que permet generar aplicacions completament responsives. Aquest framework està compost de dos tipus d'arxius, un CSS que s'encarrega de l'estilització dels elements, i un altre JS, que defineix el comportament dels elements a l'interactuar amb ells.

L'avantatge d'utilitzar aquest framework és que podem estalviar-nos molt temps d'escriure fulles d'estil, ja que Bootstrap conté una gran quantitat de components que podem incloure a la nostra web de manera fàcil. Aquests mateixos components, com poden ser les barres de navegació, faciliten la interacció amb l'usuari.

6.1.2 JQuery

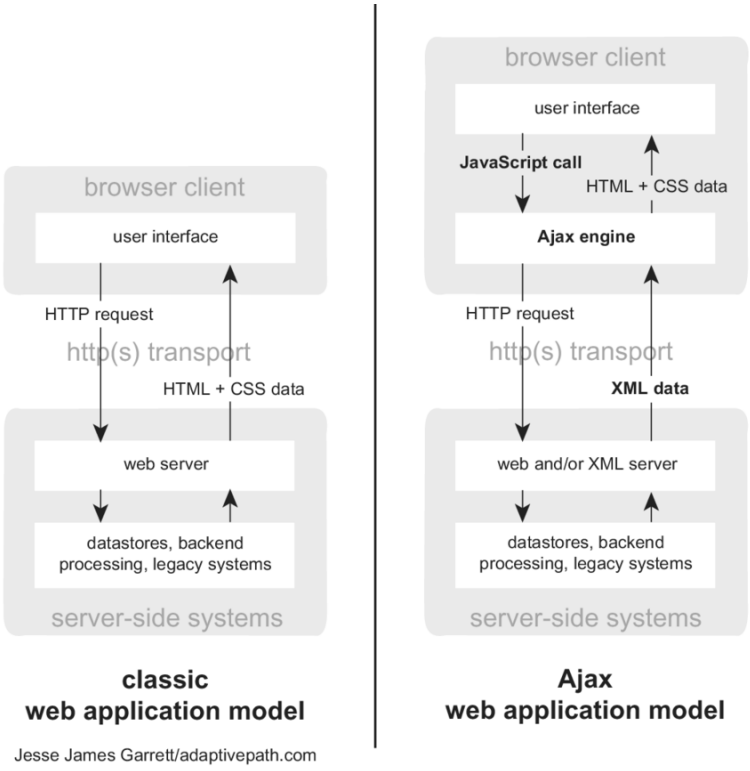
Ja que pel desenvolupament de Recording Contexts es fa molt ús de JavaScript, és necessari la utilització d'alguna llibreria que simplifiqui l'escriptura de codi. Per poder fer això faig servir la llibreria de JavaScript anomenada JQuery. Ens permet manipular els elements que formen l'estructura principal de la pàgina igual que fèiem amb JavaScript, però d'una forma més ràpida.

6.1.3 Ajax

Per cada resposta a un formulari, és necessari enviar fitxers i dades que seran emmagatzemades a la base de dades. Per no haver d'esperar que una petició finalitzi de manera correcta utilitzo AJAX, un conjunt de tècniques que permeten a una pàgina web realitzar sol·licituds al servidor en segon pla de forma asíncrona.

El principal objectiu d'AJAX és reduir el temps a l'hora de realitzar peticions, ja que no fa falta esperar resposta. Això ho fa a través d'un protocol dissenyat per efectuar una transmissió ràpida actuant en segon pla. Aquesta implementació evita

haver de recarregar tota la pàgina cada cop que la petició s'ha completat. Per tant també dota de més dinamisme a la pàgina perquè, combinat amb JavaScript, quan arriben les dades de la sol·licitud podem manipular-les de forma directa.



Imatge 9: Petició amb AJAX

A l'anterior figura podem veure la comparació entre la comunicació que s'estableix normalment, en front la comunicació que AJAX permet establir. Si ens fixem en el model classic veiem que un cop el navegador ha enviat la sol·licitud, aquesta és rebuda pel servidor i la processa. Després de que pasi una breu estona de temps, el servidor envía la resposta i el navegador es recarrega amb la nova informació.

En canvi en el model que AJAX fa servir, el client ja disposa d'una pàgina web carregada. En segon plà s'avisa al servidor de que es necessita un paquet de dades, i aquest processa la petició. Un cop ha elaborat la resposta, el procès és molt més ràpid ja que no fa falta recarregar la pàgina, simplement s'ha d'enviar el paquet de dades que s'havia sol·licitat. El format de les dades enviades pot variar i entre altres pot ser en format XML o JSON.

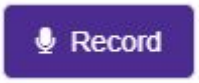
Això pot suposar un avantatge al funcionament de Recording contexts ja que ens permet enviar els àudios de manera asíncrona i sense haver d'esperar que arribi la resposta de les altres peticions.

6.1.4 Gravació d'àudio amb Javascript

La creació de formularis no tindria sentit sense la resposta dels usuaris, per això és necessari implementar un sistema que permeti gravar àudios per poder ser guardats i posteriorment estudiats pels investigadors.

Aquest sistema de gravació d'àudios ha estat implementat a partir d'una llibreria desenvolupada per Matt Diamond [5] i proporcionada per Stephino [16], la mateixa que s'utilitzava a l'anterior projecte. Gràcies a aquesta llibreria és possible la inicialització d'un objecte al qual anomeno myRecorder, i que permet accedir a la Web Audio API proporcionada per JavaScript encarregada de processar l'àudio a les aplicacions web.

En el moment que accedim a la pàgina de resposta de qualsevol formulari, el botó que permet gravar es troba en el següent estat:



Imatge 10: Botó per gravar

Quan l'usuari prem el botó, un nou recorder s'inicialitza amb un identificador específic i seguidament s'executa la funció start que inicia una nova gravació. En aquest moment, la gravadora sol·licita permisos al navegador per poder utilitzar el dispositiu d'entrada de veu predefinit. En cas que l'usuari accepti aquests permisos, comença a gravar. En cas contrari l'aplicació web no serà capaç d'accedir als serveis de gravació.

Si la gravació s'ha iniciat de manera correcta, l'estil del botó canvia i passa a visualitzar-se de la següent manera:

Imatge 11: Botó per parar la gravació

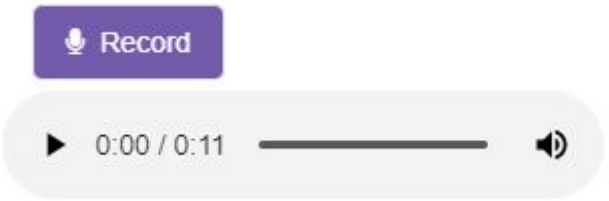
Si el botó es troba en estat de gravació, quan el tornem a prémer es crida a la funció stop del Recorder. Això fa que la gravadora s'aturi i s'exporti l'objecte a format d'àudio WAV generant un BLOB. Aquest tipus d'objecte fa referència a Binary Large Object i és molt utilitzat per guardar fitxers multimèdia. Cada un dels àudios gravats està definit per una sèrie de paràmetres:

- **Nombre de canals:** Existeix la possibilitat de guardar l'àudio en Mono (un canal) o en Stereo (dos o més canals). En aquest cas com que el nostre sistema

d'emmagatzematge només disposa de 5 GB és necessari intentar que la mida de l'àudio sigui el més petita possible, per tant es guarda en Mono.

- **Sample width:** Defineix el nombre de bits per sample. El valor ha de ser 1 o 2, i en aquest cas és dos.
- **Frame rate:** Representa la freqüència de frames i depèn de la gravació, és a dir deu de ser la mateixa que la que s'utilitza al gravar l'àudio.
- **Nombre de frames:** No és definida a l'hora de crear l'àudio, ja que depèn de la duració i del framerate.

Tot aquest procès es realitza al Back-End un cop enviem l'àudio. Respecte al Front-End, un cop l'hem gravat s'afegeix un element àudio que conté el BLOB resultant de la gravació al mateix contenidor on estava posicionat el botó.



Imatge 12: Resultat de la gravació

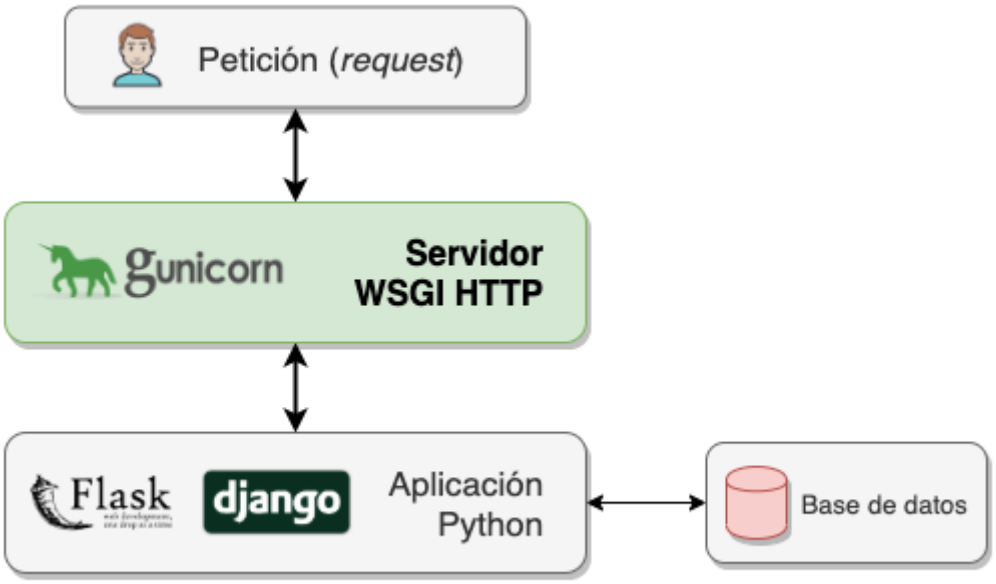
Com es pot veure, un cop es para la gravació el botó retorna a l'estil inicial. A més la implementació permet gravar un nou àudio en cas que no vulguem enviar l'àudio actual. Això faria que l'àudio que es mostra en pantalla desapareixes i fos substituït pel nou.

6.2 Gunicorn

El servidor que Django proporciona és útil durant l'etapa de desenvolupament, però no té la potència necessària ni compleix els requisits per ser utilitzat al desplegament de l'aplicació. A causa d'això es necessita un altre servidor capaç de gestionar la lògica que hi ha darrere d'una web. Per això fem servir Gunicorn, un servidor WSGI que només serveix en aplicacions que s'executen en un sistema UNIX.

WSGI (Web Server Gateway Interface) va ser implementat degut a la manca de servidors existents per gestionar aplicacions desenvolupades amb Python. El motiu de la utilització de WSGI resideix en dos factors importants:

- **Escalabilitat:** Permet gestionar un gran nombre de peticions.
- **Flexibilitat:** És possible el canvi de diferents servidors que utilitzen WSGI de manera ràpida i sense haver de modificar gran part del codi.



Imatge 13: Connexió de Gunicorn amb Django

Com es pot veure a la imatge, Gunicorn s'encarrega d'administrar totes les operacions entre el client i l'aplicació desenvolupada, gestionant diferents instàncies de la web i ocupant-se de que no es produeixi cap error.

6.3 PostgreSQL

És essencial la utilització d'una base de dades que guardi quins són els àudios corresponents a cada formulari, i a més quines són les dades personals de l'usuari que el respon. Per això en aquest projecte s'utilitza el sistema de gestió de bases de dades relacional anomenat PostgreSQL. És un sistema multiplataforma i amb una fàcil accessibilitat, el que fa que sigui molt utilitzat i reconegut.

S'utilitza PostgreSQL perquè és compatible amb el sistema de gestió de base de dades present a la plataforma de desplegament Heroku que explicaré al següent apartat, i a més presenta una sèrie d'avantatges:

- La majoria de funcionalitats que PostgreSQL aporta estan implementades seguint l'estàndard SQL, per tant es poden realitzar i incloure consultes d'altres sistemes de bases de dades.
- Segons els recursos hardware dels quals disposem, és possible configurar PostgreSQL per ajustar els recursos consumits i així optimitzar el funcionament.
- Segueix les característiques ACID d'una base de dades:

Atomicity: Es realitzen transaccions completes.

Consistency: Només s'executen transaccions que no afectin la integritat.

Isolation: Una operació no pot afecta a les altres.

Durability: Encara que el sistema falli, quan es fa una operació no es pot desfer.

6.4 Cloud Computing

El terme 'Computació al núvol' fa referència als serveis de computació que s'ofereixen a través de la xarxa. Això permet allotjar la teva pròpia aplicació sense haver-te de preocupar pel manteniment que hi ha darrere de la infraestructura on s'emmagatzema. Podem trobar tres models de Cloud Computing:

- **Infraestructura com a servei (IaaS):** És la categoria més bàsica de les tres, i es caracteritza per subministrar sistemes d'emmagatzematge de dades, servidors o màquines virtuals entre altres.
- **Plataforma com a servei (PaaS):** En aquesta categoria entren els serveis que proveeixen un environment per poder desenvolupar i gestionar aplicacions. Per fer això, els entorns proporcionats contenen una sèrie d'APIs o diferents components que permeten la integració i l'execució d'un software.
- **Software com a servei (SaaS):** És el tipus més utilitzat, i el formen les aplicacions a les quals podem accedir mitjançant el nostre navegador. La gestió de la mateixa aplicació es realitza a la infraestructura del proveïdor, i l'usuari no té cap control sobre ella més enllà de petites configuracions que se li permeten realitzar.

6.4.1 Heroku

L'objectiu principal d'aquest projecte és desplegar al núvol l'aplicació web creada i que així pugui ser accessible pels investigadors i per tots els usuaris que responguin als formularis. Per poder complir aquest objectiu utilitzem Heroku, una plataforma com a servei (PaaS) que s'encarrega de proporcionar una infraestructura perquè els desenvolupadors puguin allotjar i executar les seves aplicacions al núvol. Com ja he explicat anteriorment, en tractar-se d'una PaaS, com a usuaris d'aquest servei no ens hem de preocupar d'administrar el servidor, la seguretat o la infraestructura on s'allotja.

L'elecció d'aquest servei ve donada principalment per tres factors. El primer d'ells és la gratuïtat, ja que Heroku proporciona una opció de desplegament totalment gratuïta amb certes limitacions que no són determinants a l'hora de fer servir Recording Contexts. A més aquesta plataforma disposa d'una gran quantitat d'informació a la xarxa, i també aporta una àmplia configuració

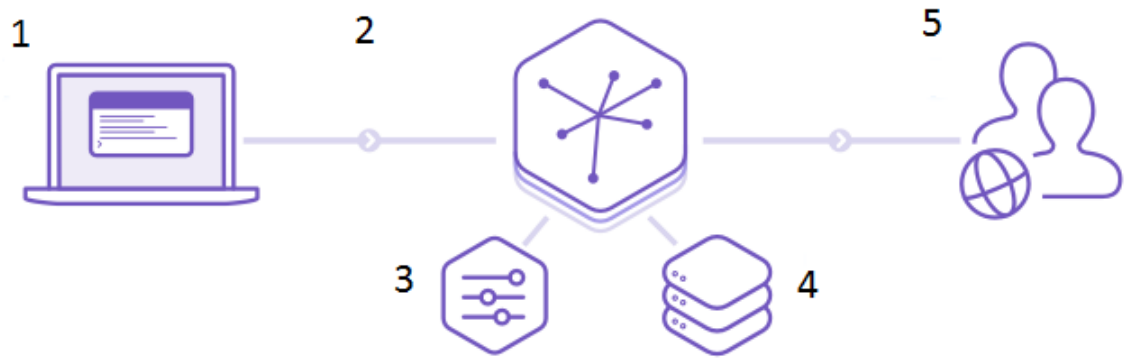
d'opcions a l'hora del desplegament. I finalment està el factor de l'experiència. Heroku és una plataforma que havia utilitzat abans, per tant resulta més ràpid el desenvolupament i el desplegament de l'aplicació.

Per poder executar l'aplicació, Heroku ho fa a través d'un sistema de contenidors anomenats dynos. Cada un d'aquests contenidors és una virtualització a nivell de sistema operatiu per Linux. Els processos que es realitzaran en aquest contenidor són definits a un arxiu Procfile situat al directori arrel del nostre projecte. Totes les comandes efectuades en un dyno no afecten a un altra, ja que són contenidors independents que estan aïllats entre si. Això implica un avantatge, ja que en cas que un dyno falli, els altres no es veuran afectats.

Heroku proporciona diferents opcions per configurar la base de dades utilitzada, entre les que està Heroku PostgreSQL. És una base de dades com a servei (DBaaS), és a dir una base de dades situada en un entorn virtual i gestionada pel proveïdor del servei. Per la utilització d'una DBaaS és obligatori disposar d'una infraestructura o una plataforma com a servei. En aquest cas la nostra PaaS és Heroku, i aquesta DBaaS està basada en PostgreSQL.

A més Heroku proporciona una sèrie de components anomenats add-ons que subministren diferents funcionalitats a la nostra aplicació. Algunes d'aquestes funcionalitats són l'administració d'emmagatzematge de les dades, o el monitoratge de l'aplicació. El nombre d'interaccions i la manera en la qual ho fa depèn del add-on utilitzat.

A la següent imatge podem veure un diagrama del funcionament de Heroku:



Imatge 14: Diagrama del funcionament de Heroku

1. El desenvolupador realitza un desplegament de la seva aplicació.
2. Aquesta aplicació és executada als dynos corresponents.
3. Mitjançant un panell podem configurar i gestionar l'aplicació desplegada.

4. El dyno es connecta amb la base de dades a través d'un o més add-on.

5. L'usuari realitza una petició i accedeix a l'aplicació.

6.4.2 Google Cloud Storage

Cada dia els dynos són reiniciats mínim un cop. Aquest procés anomenat Cycling fa que cada contenidor torni a l'estat de l'últim projecte desplegat a Heroku. A més hi ha altres situacions en les quals el dyno es reinicia:

- Es fa un nou desplegament.
- Es canvia la configuració del projecte.
- Es canvien els add-ons.
- S'executa la comanda 'heroku restart'.

Aquest procés no fa que la base de dades s'esborri, ja que està separada del dyno. Però els fitxers estàtics nous s'emmagatzemen al contenidor, és a dir que cada cop que pugem un nou àudio a l'aplicació, aquest s'esborra al final del dia. Això obliga als investigadors a haver de descarregar tots els fitxers nous abans que el dyno es reinicii.

Per resoldre aquest inconvenient, hi ha la possibilitat d'emmagatzemar els fitxers estàtics a un servei dedicat exclusivament a l'allotjament d'aquests arxius. Existeixen una gran varietat de tecnologies que permeten fer això, però la majoria requereixen pagar una quota mensual per ús. Per aquest fet es va haver de buscar la millor alternativa amb relació al preu. A l'hora de buscar hi havia tres requisits importants:

- Emmagatzematge disponible.
- Límit d'accés per mes. És a dir, quantes sol·licituds es poden fer en un període de trenta dies.
- Preu segons la duració.

Entre totes les opcions que hi havia vaig escollir les tres que oferien un millor servei tenint en compte les característiques esmentades. Les opcions es mostren a la següent taula:

TECNOLOGIA	EMMAGATZEMATGE	LIMIT D'ACCÈS PER MES	PREU - DURACIÓ
Amazon S3	5GB	• 20000 sol·licituds GET • 2000 sol·licituds PUT o POST • 15 GB de transferencia de dades de sortida	Gratis durant 12 mesos
Google Cloud Storage	5GB	• 50000 sol·licituds GET • 5000 sol·licituds PUT o POST	Gratis sempre mentre no es passi el límit mensual
DropBox	2GB	Sense límit	Gratis sempre

Imatge 15: Comparació de les diferents opcions d'emmagatzematge

A causa de la gratuïtat indefinida sempre que no es passi del límit mensual, es va decidir optar per Google Cloud Storage. Es va descartar DropBox ja que hi havia molt poca documentació. L'opció escollida és una IaaS molt semblant a la que apareix a la taula anomenada Amazon S3. Permet emmagatzemar i accedir a arxius allotjats a través d'una API proporcionada. Tots aquests fitxers s'organitzen en directoris anomenats buckets, que contenen una clau per poder accedir. A més aquesta eina disposa d'un monitoratge que permet supervisar l'estat del nostre bucket.

6.5 GitHub

Per poder arribar a la versió final del projecte, s'han hagut de passar per diferents etapes. Per això és necessari l'ús de GitHub, un control de versions que permet als desenvolupadors allotjar el seu codi. Gràcies a aquesta eina és possible administrar el projecte i tenir versions allotjades de manera segura al núvol.

A més GitHub és essencial a l'hora de poder fer un desplegament de la nostra aplicació, ja que Heroku dóna la possibilitat d'obtenir el codi des de un repositori creat en aquesta plataforma de control de versions.

7 Django

Django és un framework d'alt nivell que serveix per desenvolupar aplicacions web utilitzant Python. La seva arquitectura compleix el patró Model-Vista-Controlador vist en apartats anteriors.

La principal característica de Django és el seu Object-Relational Mapping (ORM). Aquest terme fa referència a la forma de convertir la informació entre la base de dades i el sistema que utilitza el llenguatge amb què estem programant. Django proporciona un ORM que es caracteritza per la facilitat a l'hora de realitzar operacions a la base de dades.

El framework conté una classe anomenada models que permet crear subclasses que defineixen objectes. El que definim al Model és el que determinarà com és la base de dades. Com a exemple veurem la següent imatge:

```python
from django.db import models

class Usuari(models.Model):
    nom = models.CharField(max_length=30)
    cognom = models.CharField(max_length=30)
```

Imatge 16: Model Usuari

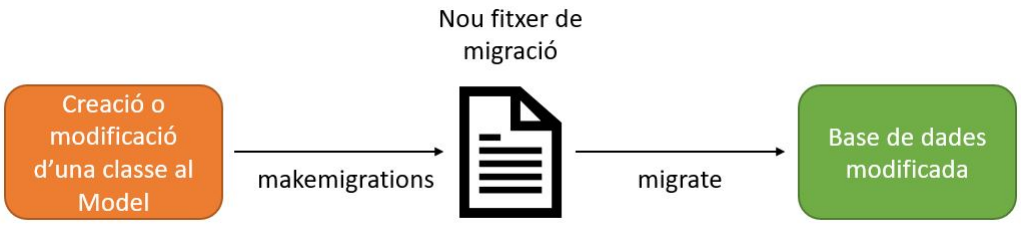
Veiem que s'ha definit una classe Usuari. Aquesta classe farà referència a la taula de la base de dades que es vol crear, i els atributs nom i cognom seran els camps que contindrà la taula. És a dir que la classe Usuari equival a crear una taula amb sintaxi PostgreSQL d'aquesta manera:

```sql
CREATE TABLE rek_usuari (
    "id" serial NOT NULL PRIMARY KEY,
    "nom" varchar(30) NOT NULL,
    "cognom" varchar(30) NOT NULL
)
```

Imatge 17: Creació taula a partir del model

El nom de la taula és definit pel framework, però pot ser canviat. I en el cas del id és creat de forma automàtica. En aquest exemple es mostren els atributs en format varchar, però hi ha una varietat de tipus d'atributs que podem definir a les classes. Per tant totes les classes que implementem a Models, aniran a parar a la base de dades en forma de taula.

Per poder indicar a la base de dades les taules que volem crear o modificar, es fan servir les migracions. Com es pot veure a la imatge d'avall, quan es modifica o es crea una classe al Model, podem executar una comanda específica anomenada makemigrations que genera un fitxer amb tots els canvis fets o amb la informació d'una nova classe. Per poder notificar aquests canvis a la base dades és necessari executar la comanda migrate, que s'encarrega d'aplicar les migracions fetes anteriorment que es troben al fitxer.



Imatge 18: Migracions

El fitxer generat a partir de la comanda makemigrations segueix aquesta estructura:

```
class Migrations(migrations.Migration):

    dependencies = [('migrations', '0001_initial')]

    opertations = [
        migrations.DeleteModel('Persona'),
        migrations.AddField('Usuari', 'edat', models.IntegerField(default=0)),
    ]
```

Imatge 19: Fitxer generat

Està compost d'una classe anomenada Migrations. I l'estructura està dividida en dos:

- **dependencies:** És la llista de migracions anteriors de les qual depèn l'actual.
- **operations:** Són totes les operacions que s'han de realitzar a la base de dades. En aquest cas veiem que s'esborra un model anomenat Persona, és a dir que una taula és esborrada.

Un cop modificada la base de dades, Django proporciona una API per instanciar els models i realitzar operacions com pot ser la modificació d'una ocurrència a la base de dades, o la creació d'una nova. A continuació mostro un exemple de com seria realitzar una query a un model anomenat Usuari que conté el camp 'nom':

```
Usuari.objects.get(nom__contains='Lennon')
```

Imatge 20: Query a classe Entry

Primer s'ha d'escriure el model referenciat seguit d'un objects que permet iterar per tots els objectes creats a la taula de la base de dades. I finalment s'escriu un get perquè la consulta retorni l'objecte que conté Lennon a la columna nom. Per tant la semàntica contains fa referència al LIKE que es realitza a una consulta SQL i que indica que es busca un element que contingui el paràmetre especificat. Traduït a SQL seria així:

```
SELECT ... WHERE nom LIKE '%Lennon%';
```

Imatge 21: Equivalència de la query

Per tant, l'ORM de Django facilita la feina a l'hora de crear noves taules a la base de dades o de modificar altres ja existents. A més l'API que proporciona simplifica les consultes i la manipulació de la informació.

7.1 URL Mapping

A l'iniciar un projecte amb Django, hi ha dos fitxers essencials que s'han de crear:

- **settings.py:** Representa la configuració del projecte. Aquest fitxer conté una variable anomenada ROOT_URL que indica quin és l'arxiu on estan definides les URLs del projecte.
- **urls.py:** És on es troben instanciades totes les possibles URLs que conté el nostre projecte. És a dir, la variable ROOT_URL de settings.py apuntarà a urls.py . A la següent imatge es mostra un exemple d'aquest arxiu.

```
urlpatterns = [
    path('login/', views.login, name='login'),
    path('nuevoFormulario/', views.nuevoform, name='nuevoform'),
    path('eliminarForm/', views.eliminarForm, name='eliminarForm'),
    url(r'editarFormulario/(?P<formulario>.*)/$', views.editForm, name='editForm'),
    url(r'contestar/(?P<formulario>.*)/paginaPrincipal/$', views.viewFormPaginaPrincipal, name='viewFormPaginaPrincipal'),
]
```

Imatge 22: urls.py

Es pot veure que normalment una URL conté tres parts:

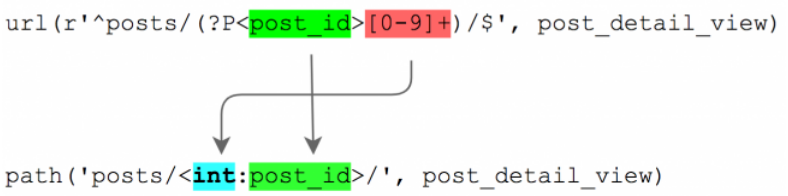
- **Patró:** Serveix per indicar la URL que s'ha de mapejar.

- **Funció:** És la funció que s'ha d'executar en el moment que es vol accedir a l'URL. Per exemple al primer path de l'anterior imatge, en cas que l'URL mapejada sigui 'login/', s'executarà la funció login de l'arxiu views.py, on es realitzen les operacions necessàries per iniciar sessió.
- **Nom:** És una string que s'associa a l'URL concreta i serveix per cridar-la a través d'un nom en cas que sigui necessari.

A més, com es veu a l'exemple, Django proporciona dues formes de definir el patró:

- **url():** Utilitza una expressió regular per veure si l'URL coincideix.
- **path():** Va sorgir per poder simplificar la creació de patrons a partir d'una expressió regular. Utilitza diferents paràmetres segons el tipus de dada que volem.

A continuació veiem com és l'equivalència entre url() i path():



Imatge 23: Conversió de url a path

Un cop definides les URLs, quan es realitza una petició se segueix el següent algorisme per buscar la funció a executar i retornar la resposta adient:

1. A partir de la variable ROOT_URL definida a settings.py Django defineix el mòdul URLconf, que indica els patrons que s'hauran de comparar i apunta al fitxer que conté la llista de les URLs.
2. El framework accedeix al mòdul definit a URLconf i busca la variable urlpatterns, que és on es troben definides totes les URL.
3. Es va iterant per tota la llista de URLs i es comprova que alguna coincideixi amb el patró de l'URL sol·licitada.
4. En cas que coincideixi, Django executa la funció definida a l'URL coincident. A aquesta funció li passa una instància del HttpRequest, i es retorna un HttpResponse.
5. Si cap URL definida a l'arxiu coincideix amb la sol·licitada, llavors es genera un error.

7.2 Sistema de fitxers

Django proporciona un sistema de gestió de fitxers que permet realitzar diferents operacions com per exemple la lectura d'arxius.

El framework fa una separació segons el tipus de fitxers que l'aplicació conté:

- **STATIC:** En aquest tipus entren els fitxers que el desenvolupador proporciona. Un exemple serien els fitxers JavaScript i CSS.
- **MEDIA:** Aquests arxius són els que generen els usuaris. Per tant, els àudios gravats pels usuaris corresponen a aquest tipus.

La gestió d'aquests fitxers és portada a terme per Django i es fa mitjançant la inicialització d'aquestes quatre variables:

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR,'media')
```

Imatge 24: Variables pels fitxers

STATIC_URL defineix quin és el directori on es troben els fitxers estàtics que volem recuperar. Per poder recuperar-los es fa a través d'una comanda anomenada collectstatic, que obté els arxius de l'URL especificada i els copia a STATIC_ROOT. En aquest cas el BASE_DIR especificat a STATIC_ROOT indica el directori del projecte.

En cas dels arxius de tipus media funciona igual, MEDIA_URL representa l'URL des d'on són utilitzats els fitxers multimèdia. En canvi MEDIA_ROOT fa referència al directori on s'han de guardar.

Un cop definides aquestes variables, STATICFILES_STORAGE serà la classe encarregada de gestionar els fitxers estàtics. En canvi els fitxers media seran gestionats per DEFAULT_FILE_STORAGE.

7.3 Django Template Language

La majoria d'aplicacions web necessiten una estructura Front-end que sigui capaç de gestionar dades dinàmiques. És a dir, molts cops una web requereix mostrar una sèrie de dades contingudes en una llista on cada una d'elles conté valors diferents. Per això Django proposa templates que disposen d'una part

estàtica proporcionada per l'HTML, però a més ofereixen funcions per poder tractar dades dinàmiques, i això és anomenat Django Template Language (DTL).

Els templates són fitxers de text que utilitzen una estructura HTML. A l'hora de mostrar una pàgina, primer es carrega el template requerit i es compila la presentació. I després es renderitza substituint els paràmetres necessaris per les dades corresponents. L'API de Django defineix una sintaxi concreta a l'hora d'utilitzar variables, etiquetes o filtres.

En el cas que tinguem una variable anomenada blog, es representarà com {{ blog }}. A més les variables poden contenir diferents estructures:

- **Atributs:** {{ blog.titol }} En aquest cas es mostrarà el títol del blog.
- **Diccionaris:** {{ blog.key }} En cas que la nostra variable sigui un diccionari, podem obtenir les claus o els ítems necessaris.
- **Llistes:** {{ blog.0 }} Si la variable és una llista, podem accedir als valors escrivint l'índex on es troba la dada que ens interessa.

També podem definir etiquetes que seran representades com {% ... %}, i les funcions que es poden fer servir són:

- **Condicionals:** {% if %} ... {% endif %}
- **Iteratius:** {% for i in x %} ... {% endfor %}
- **Herencia:** {% extends "base.html" %}

I per últim podem crear filtres que es representaran com les variables, però utilitzant el caràcter | com si fos una canonada. Alguns exemples són:

- **Format de la data:** {{ birth_date|date:" D M Y" }}
- **Valors predeterminats:** {{ name|default:"Maria" }}
- **Canviar string a minúscules:** {{ name|lower }}

A la següent imatge es pot veure un exemple amb algunes de les funcions esmentades.

```
{% for form in forms_list %}
    <h1> {{forms.title|lower }} </h1>
    <p> Num contextos: {{ form.num_contextos }} </p>
    <p> Num respostes: {{ form.num_respostes }} </p>
{% endfor %}
```

Imatge 25: Template que mostra formularis

En aquest exemple des del Back-End s'ha enviat una llista de formularis amb diferents atributs com el títol, el nombre de contextos i el nombre de respostes. Es defineix un bucle que itera per tots els formularis imprimint cada un dels atributs, i en el cas del títol el mostra en minúscula.

7.4 Autentificació d'usuari

Per poder administrar els usuaris, Django proporciona el seu sistema d'autentificació que permet la creació de comptes amb permisos, i la gestió de diferents grups.

Els usuaris a Django són objectes que es troben al sistema d'autentificació. Només existeix una classe d'usuari i conté els següents atributs:

- username
- password
- email
- first_name
- last_name

La contrasenya és guardada amb un hash. A més aquesta classe pot ser modificada per afegir un usuari personalitzat amb els atributs que a nosaltres ens interessin. També tenim la possibilitat de crear un superuser que servirà per gestionar l'administració de l'aplicació, ja que tindrà tots els permisos.

Als altres comptes els hi podem donar diferents permisos a l'hora de crear-los o des del panell d'administrador del qual parlaré més endavant. Aquests permisos permeten realitzar les següents operacions: veure, afegir, canviar o esborrar, i es realitzen sobre els objectes de la base de dades. Per no haver de donar permisos un per un, existeix la possibilitat de crear un grup que tingui uns permisos concrets. En el moment que un usuari sigui afegit a aquest grup, els seus permisos passaran a ser els indicats al grup.

Per poder iniciar sessió, Django proporciona un mètode login():

```python
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username, password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

Imatge 26: Inici de sessió

A la imatge podem veure que s'envia un POST que conté un username i una password. Amb aquestes variables es verifiquen les credencials a través de la funció authenticate(). En cas que les dades introduïdes siguin vàlides, el mètode retorna un objecte User i es realitza el login(), que permet que el ID de l'usuari es guardi a la sessió. A més també proporciona una funció logout() per poder tancar sessió.

Django també proporciona un panell d'administrador al que només poden accedir els usuaris amb permisos:

Imatge 27: Panell d'administrador

Aquesta interfície mostra les dades que hi ha a cada model creat al projecte. A la secció de l'esquerre es veuen tots els models, incloent-hi els usuaris i els grups.

Aquest panell permet modificar o afegir noves dades a la taula corresponent. A la part de la dreta es poden veure les últimes operacions realitzades sobre la base de dades.

Els models que volem veure en aquesta interfície vénen definits pel fitxer admin.py a través del mètode admin.site.register().

```
from django.contrib import admin
from rek.models import Formulario, Seccion, Respuesta

# Register your models here.
admin.site.register(Formulario)
admin.site.register(Seccion)
admin.site.register(Respuesta)
```

Imatge 28: Registre de models al panell d'administrador

Per poder accedir a la interfície és necessari que a l'URL principal afegim '/admin/'. Per exemple si estem executant el codi en local, al navegador haurem d'introduir: http://127.0.0.1:8000/admin/. A la secció 13.2.2 dels annexos es pot trobar el manual d'ús d'aquest panell.

7.5 Seguretat

Django proporciona diferents funcionalitats per garantir la seguretat a l'hora d'utilitzar l'aplicació web. Algunes d'elles són:

- **Cross site scripting (XSS):** Aquests atacs permeten inserir scripts dins del navegador de l'usuari a través de l'aplicació web. Per evitar això, els templates de Django protegeixen la pàgina de la gran majoria d'atacs XSS.
- **Encriptació del tràfic:** Es pot habilitar l'opció d'utilitzar SSL/HTTPS per poder encriptar les dades enviades entre el navegador i l'aplicació.
- **Validació del HOST:** Gràcies a una llista que el desenvolupador proporciona, Django pot validar el header del host des d'on s'està intentant executar l'aplicació.

7.6 Configuració

És necessari que el nostre projecte contingui un arxiu anomenat settings.py que servirà per definir la configuració de Django. En aquest fitxer es defineixen diferents variables:

- **SECRET_KEY:** Utilitzada per fer els hash necessaris.

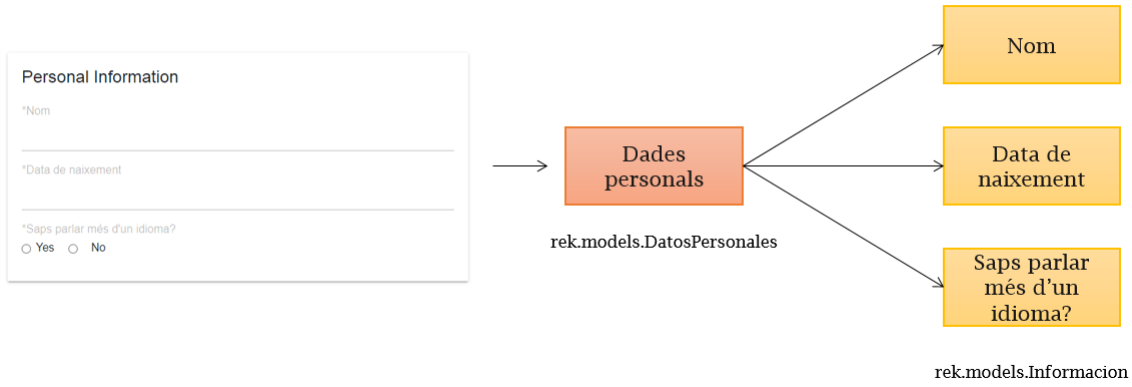
- **DEBUG:** L'estat d'aquesta variable varia entre True i False, i en cas que estigui True ens servirà per mostrar per pantalla els detalls dels errors.
- **ALLOWED_HOSTS:** Com ja he explicat a l'apartat anterior, ens serveix per indicar quins són els hosts des d'on podem executar la web.
- **ROOT_URLCONF:** Permet l'URL Mapping indicant el mòdul on es troben les URLs del projecte.
- **TEMPLATES:** Defineix quina és la lògica que segueixen els templates i en quin directori es troben.
- **DATABASES:** És una llista de les bases de dades que són utilitzades a l'aplicació.
- **Variables de fitxers:** Són les variables vistes a l'apartat de la gestió dels fitxers com STATIC_URL o STATIC_ROOT (veure secció 7.2).

8 Model de dades

8.1 Diagrama de classes

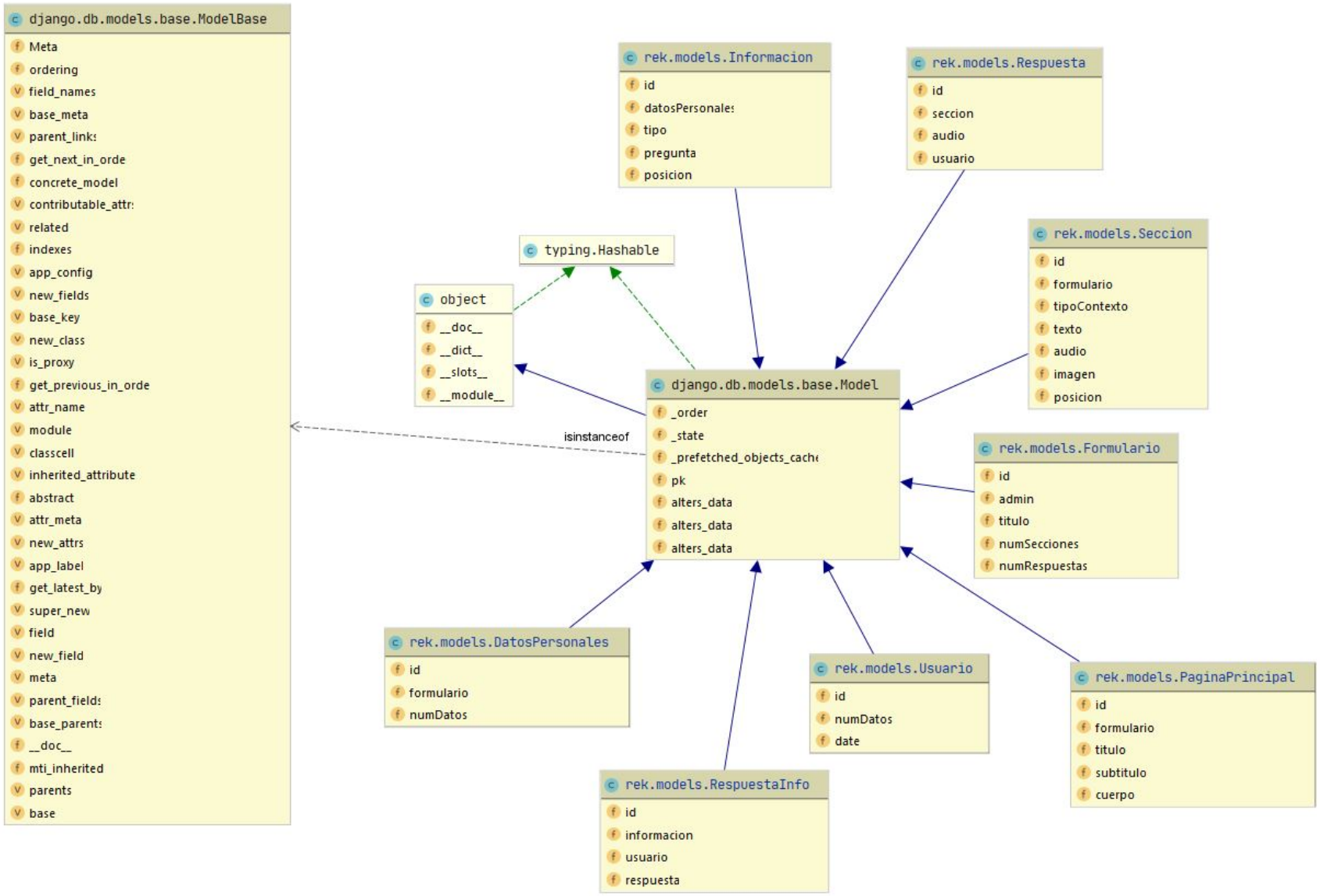
Cada model creat a l'aplicació és una subclasse de Model, que a l'hora és una instància de la classe ModelBase. És per això que al diagrama mostrat a la següent pàgina es pot veure una relació entre aquestes classes. A més s'han definit diferents models per poder crear relacions dinàmiques entre els objectes creats.

Per exemple un formulari conté dades personals, però no sabem el nombre de dades representades al formulari, per tant s'ha de crear un contenidor principal que contindrá la ID del bloc i a l'hora anar creant una serie d'objecte que seran les dades que volem. Aquest cas està il·lustrat a la següent imatge:



Imatge 29: Exemple de dades personals

A l'esquerra es pot veure un formulari on es demanen tres camps a omplir. El conjunt d'aquestes dades ve donat pel model "DatosPersonales", i com no sabem el número de dades que l'investigador afegirà en aquest contenidor d'informació, creem un nou objecte per cada un dels camps. És a dir que cada un d'ells s'instanciarà com un objecte del model "Informacion". Llavors al final tenim un sol objecte de dades personals, i tres que indiquen la informació demanada.



Imatge 30: Python class diagram

8.2 Base de dades

A l'hora de crear els models Django permet establir relacions entre ells. Això serveix per poder definir connexions i associar un model amb altres. El framework aporta les tres relacions bàsiques que podem trobar a una base de dades relacional:

Many-to-one

Defineix que un model pot estar associat a diferents objectes d'un altre model. Si tornem a l'exemple de l'apartat anterior, veiem que un formulari de dades personals pot tenir diferents instàncies d'objectes que representen els camps demanats per l'investigador. En canvi, un camp d'informació no pot estar a un altre formulari, ja que són únics per cadascun.

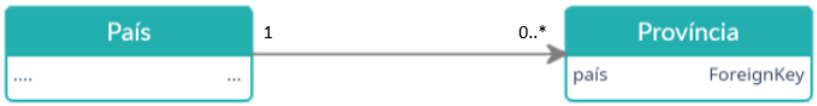
Per poder indicar que hi ha una relació Many-to-one entre dos models es fa de la següent forma:

```python
from django.db import models

class Pais(models.Model):
    # ...
    pass

class Provincia(models.Model):
    pais = models.ForeignKey(Pais, on_delete=models.CASCADE)
    # ...
```

Imatge 31: Many-to-one relationship

Un país pot tenir moltes províncies, en canvi una mateixa província no pot estar en països diferents. Per tant al model de País no hem de fer cap canvi, però al model de Província, que és la que pot presentar diferents instàncies, definim una ForeignKey que fa referència al País.

A l'hora de ser representat a un diagrama, es faria de la següent manera:



Imatge 32: Many-to-one a un diagrama

Many-to-many

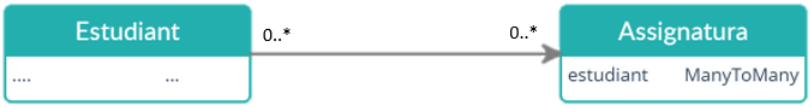
En aquest tipus de relació, diferents objectes poden estar associats a diferents objectes d'un altre model. A Recording Contexts no hi ha cap situació en la que es

produeixi aquest tipus de relació. Com a exemple posaré un estudiant i les diferents asignatures que hi pot haver en un curs escolar.

```python
from django.db import models

class Estudiant(models.Model):
    # ...
    pass

class Assignatura(models.Model):
    # ...
    estudiant = models.ManyToManyField(Estudiant)
```

Imatge 33: Many-to-many relationship

S'estableix una relació Many-to-many ja que una assignatura pot ser cursada per més d'un estudiant, i un estudiant pot cursar més d'una assignatura. És igual en quin dels dos models posem la relació, però només ha de definir-se en un. La representació en un diagrama sería així:



Imatge 34: Many-to-many a un diagrama

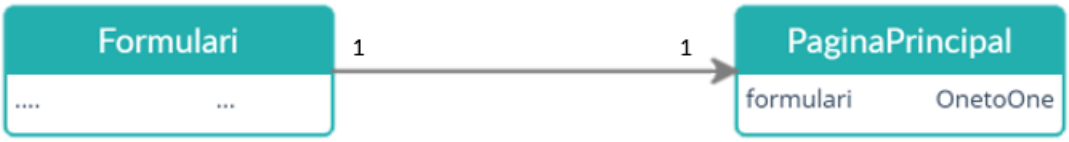
One-to-one

És la relació que s'estableix quan un objecte d'un model només pot fer referència a un altre. A Recording Contexts trobem dos casos en els que es dona aquesta relació. Un d'ells és la relació entre un formulari i la pàgina principal que explica a l'usuari de que tractarà el formulari.

```python
from django.db import models

class Formulari(models.Model):
    # ...
    pass

class PaginaPrincipal(models.Model):
    # ...
    formulari = models.OneToOneField(Formulari)
```

Imatge 35: One-to-one relationship

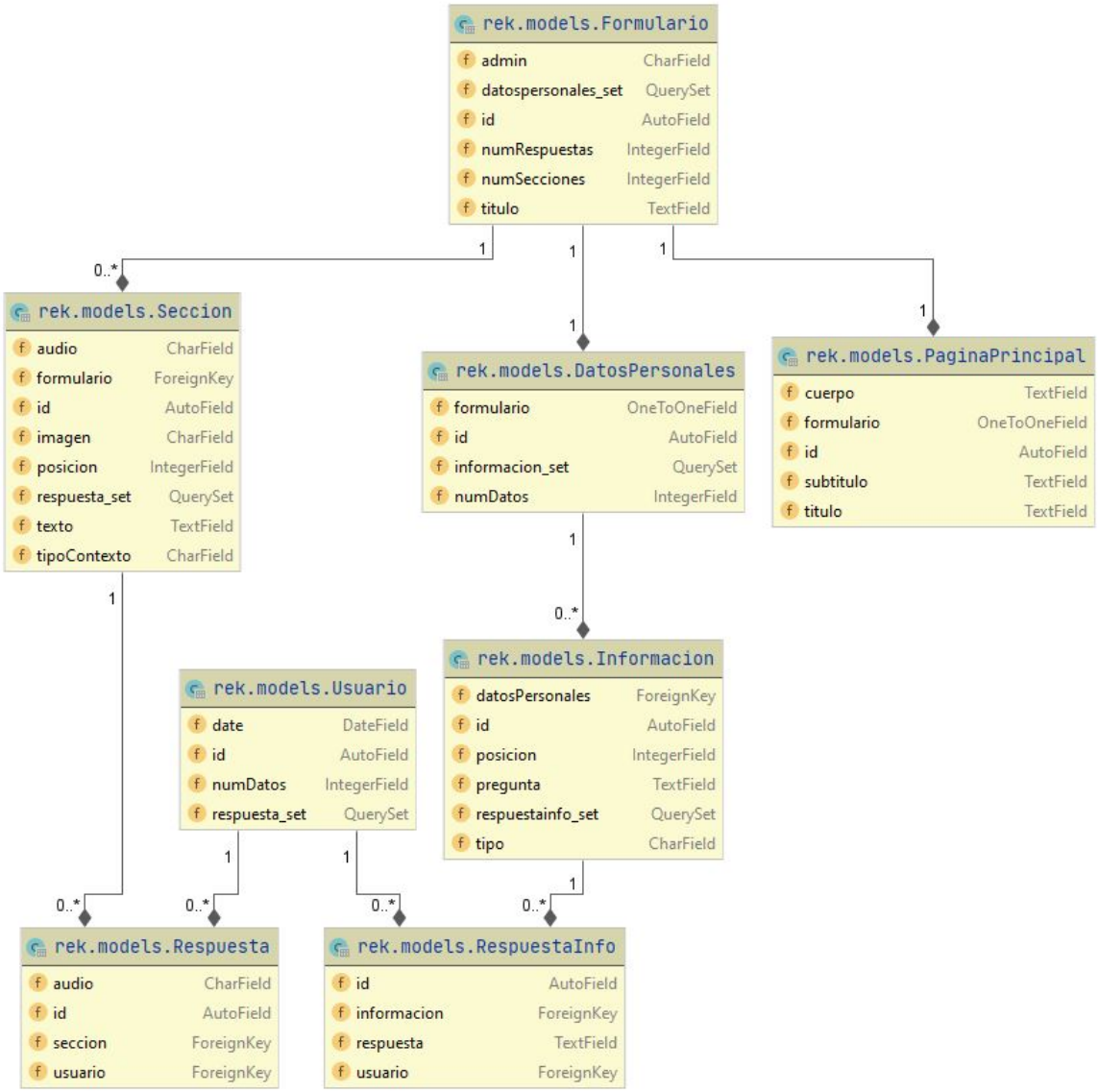
I la representació al diagrama seria de la següent forma:



Imatge 36: One-to-one a un diagrama

Diagrama de dependència dels models

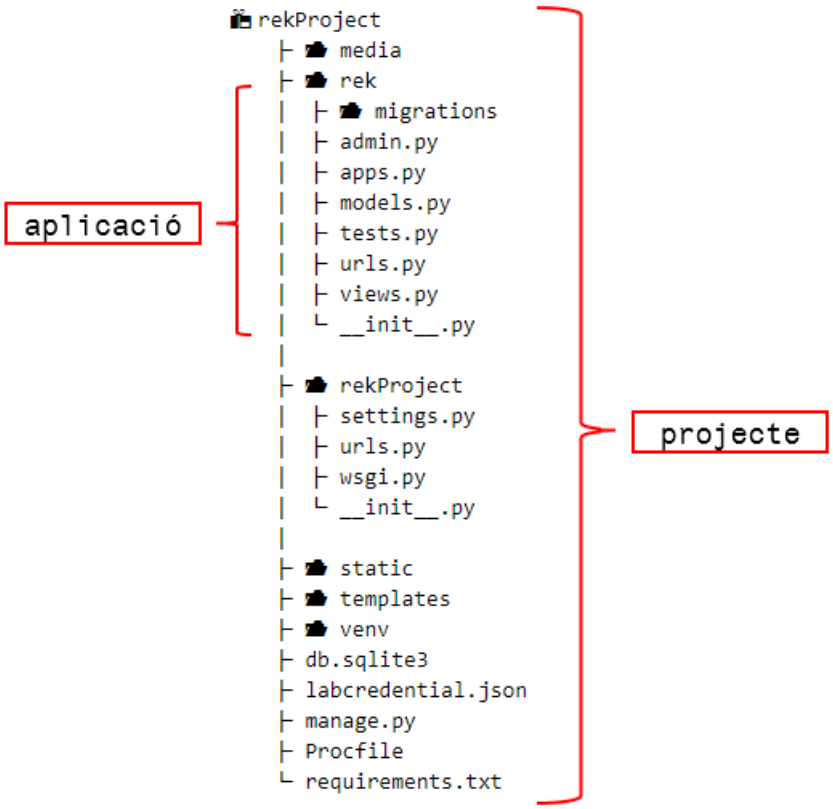
Totes les relacions entre els models que conté l'aplicació estàn definides de la següent manera:



Imatge 37: Model Dependency Diagram

9 Estructura

En un projecte de Django l'estructura es pot dividir en dos parts, una és el projecte i l'altre l'aplicació. A la següent imatge es pot veure l'estructura de Recording Contexts.



Imatge 38: Estructura del projecte

El projecte defineix l'aplicació i totes les parts que la formen, en canvi l'aplicació és un módul que es troba dins del projecte i que permet implementar les funcionalitats necessàries. A Recording Contexts tenim un projecte anomenat rekProject, i una aplicació anomenada rek.

9.1 Projecte

Per poder crear un projecte primer s'ha d'executar la comanda "django-admin startproject rekproject". Això ens crearà de forma automàtica una estructura predeterminada que conté els següents fitxers:

- **settings.py:** Com ja vaig explicar en apartats anteriors, aquest fitxer representa la configuració del projecte. Defineix diferents aspectes com poden ser el middleware o la base de dades.

- **_init_.py:** Un package de Python és un directori que conté un arxiu _init_.py, és a dir al nostre projecte afegim un fitxer _init_.py buit per indicar que el directori del projecte ha de considerar-se com un paquet.

- **manage.py:** És un fitxer que permet executar tasques específiques de Django a través d'una sèrie de comandes. Algunes d'aquestes comandes són:

  – runserver: És utilitzat per poder executar el servidor que Django proporciona. Si no especifiquem el host ni el port, el servidor començarà a escoltar en "127.0.0.1:8000", és a dir al port 8000 del localhost.

  – makemigration i migrate: Aquestes comandes ja han estat comentades en apartats anteriors, i serveixen per aplicar a la base de dades els canvis que hi ha hagut al model de la nostra aplicació.

- **wsgi.py:** Serveix per poder desplegar l'aplicació a diferents servidors, ja que determina la connexió entre el framework i el servidor WSGI. Durant el desenvolupament lo normal és utilitzar el servidor proporcionat per Django, això fa que aquest arxiu només es faci servir durant el desplegament o si s'està fent servir un docker.

- **url.py:** Aquest fitxer també ha sortit en apartats anteriors, i és utilitzat per definir totes les URLs que conté el projecte.

Per tant els fitxers anteriors són els que ens trobem a l'hora d'iniciar un projecte, però durant el desenvolupament fa falta crear directoris i altres tipus de fitxers per definir nova informació:

- **media:** És el directori que conté els fitxers que l'usuari carregarà.

- **static:** Conté tots els fitxers estàtics com poden ser els CSS i els JS.

- **templates:** És la carpeta on es troben tots els arxius HTML que serviran per donar estructura al Front-End.

- **venv:** Defineix l'entorn virtual que conté tots els paquets de Python necessaris per poder executar i desenvolupar l'aplicació.

- **db.sqlite3:** És la base de dades inicial creada per Django. Aquesta només es farà servir durant el procés de desenvolupament. En el moment de desplegar l'aplicació a Heroku és necessari canviar la base de dades perquè estigui al núvol.

9.2 Aplicació

Un cop el projecte ha estat creat, s'ha de definir una aplicació per poder implementar les funcionalitats que volem que tingui la nostra web. A través de

l'execució d'una comanda amb manage.py, el framework genera l'estructura bàsica que una aplicació ha de tenir, això fa que ens estalviem la feina de crear nosaltres mateixos els directoris necessaris.

Imatge 39: Creació de l'aplicació rek

A la imatge anterior podem veure l'estructura generada per l'aplicació de Recording Contexts, on cada un dels components té una finalitat específica:

- **migrations:** Aquest directori conté tots els fitxers generats a partir de la comanda makemigrations. És a dir, en aquesta carpeta és on es definiran els canvis a la base de dades.
- **admin.py:** Serveix per indicar quins seran els models que volem mostrar a la interfície d'administrador.
- **apps.py:** Permet configurar l'aplicació. Actualment, a Recording Contexts només es defineix el nom de l'aplicació.
- **models.py:** És el fitxer on es defineixen els models, que són una representació del que contindrà la base de dades.
- **test.py:** Aquí és on es desenvolupen els tests necessaris per comprovar el correcte funcionament del projecte.
- **urls.py:** La funció és la mateixa que al fitxer url.py del projecte, es defineixen les diferents URLs de l'aplicació.
- **_init_.py:** El propòsit d'aquest arxiu és com el que ens hem trobat al projecte, indica que aquest directori s'ha de tractar com un paquet de Python.
- **views.py:** Es defineixen diferents view functions, que s'encarreguen de retornar una resposta a una sol·licitud determinada. Aquí podem veure un exemple de view:

```python
from rek.models import Formulario

def adminprofile(request):
    forms = Formulario.objects.filter(admin=request.user.username)
    return render(request, 'adminProfile.html', {"forms": forms})
```

Imatge 40: View function

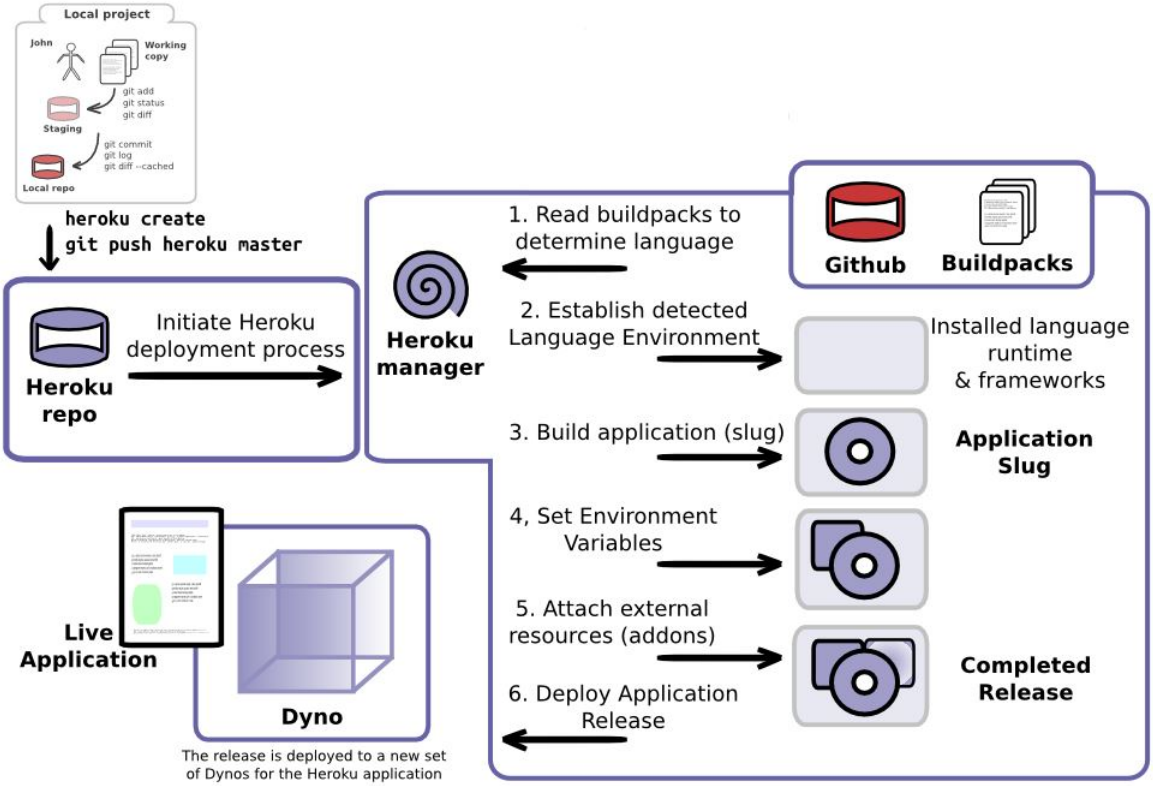
La funció definida s'anomena adminprofile, i l'objectiu és extreure tots els formularis de la base de dades on l'administrador sigui el mateix que l'usuari amb sessió iniciada.

1. Importem el model Formulario, que és el que fa referència a la taula on es troba la informació sobre els formularis.
2. La funció rep com a paràmetre un request enviat des del navegador.
3. Realitzem les operacions necessàries per poder obtenir els formularis. En aquest cas utilitzem l'API proporcionada per Django i filtrem entre tots els formularis per poder trobar els que tenen com a administrador l'usuari enviat a través de la sol·licitud.
4. Finalment enviem la HttpResponse a través d'un render, que rep com a paràmetres el request, el nom del fitxer HTML que volem utilitzar i el context que no és més que un diccionari amb les dades extretes.

10 Cloud Computing

10.1 Desplegament amb Heroku

El desplegament del projecte s'ha realitzat mitjançant Heroku, i per poder fer-ho s'ha seguit el procés il·lustrat a la imatge que apareix a continuació.



Imatge 41: Procès del desplegament

Abans de fer un desplegament a Heroku, necessitem que el nostre codi estigui a un controlador de versions com GitHub. És a dir les implementacions que fem en local han de passar-se a un repositori. Seguidament s'ha de crear una nova aplicació de Heroku amb la comanda "heroku create rekproject". Un cop fet això, des de la consola de comandes o des de la interfície de Heroku, s'ha de fer un push del codi implementat a GitHub, i a partir d'aquí comença el procés de desplegament:

1. Es determina quin llenguatge de programació es fa servir.

2. Es crea un enviroment en funció d'alguns fitxers especificats al projecte. En el cas de Django és possible crear un arxiu anomenat requirements.txt a través de la comanda "pip freeze > requirements.txt". Aquest fitxer serveix per poder especificar quins són els paquets que el projecte està utilitzant i quines

versions són necessàries, inclòs el framework. Per tant l'enviroment creat per Heroku contindrà tots els paquets que vénen determinats a requirements.txt.

3. Seguidament l'aplicació és compilada a un Slug, que és una còpia optimitzada de l'aplicació per ser distribuïda al dyno.

4. Un cop l'aplicació està compilada, s'inicialitzen les variables de l'entorn i s'afegeixen els add-ons i les seves configuracions. L'únic add-on utilitzat a Recording Contexts és el que gestiona i permet la connexió a una base de dades PostgreSQL.

5. Finalment es crea una Release, ja que que cada cop que fem un canvi i realitzem un desplegament es crea una. Això permet tornar a versions anteriors en cas que la nova versió no sigui del tot eficient.

La Release creada va a parar al Dyno, que serà el contenidor on s'executi l'aplicació a partir del Procfile present al projecte, ja que és el fitxer que determina quina és la comanda que executarà l'aplicació.

10.2 Emmagatzematge a Google Cloud Storage

Com que Heroku no proporciona un emmagatzematge de fitxers adient, Recording Contexts utilitza Google Cloud Storage per gestionar els diferents arxius que són pujats a l'aplicació. Per això primer hem de crear un bucket, que serà qui contingui tots els fitxers. En aquest cas el nom asignat al bucket és lab-fonetica-rek.

Per poder enllaçar l'aplicació amb el bucket creat és necessària la llibreria django-storages, que realitza l'autenticació mitjançant les credencials associades a Google Cloud Storage. Per això, a través de la interfície proporcionada per Google, s'han de baixar les credencials en format JSON, i crear una variable d'entorn del projecte que apunti al fitxer descarregat.

```
from google.oauth2 import service_account

GS_CREDENTIALS = service_account.Credentials.from_service_account_file(
                            os.path.join(BASE_DIR, 'labcredential.json')
                    )
```

Imatge 42: Credencials

Un cop indicades les credencials, s'ha de canviar el sistema que utilitza el projecte per proporcionar els fitxers estàtics.

```
DEFAULT_FILE_STORAGE = 'storages.backends.gcloud.GoogleCloudStorage'
GS_BUCKET_NAME = 'lab-fonetica-rek'
STATIC_URL = "https://storage.googleapis.com/{}/".format(GS_BUCKET_NAME)

STATICFILES_DIRS = (os.path.join(BASE_DIR, "static"), )
STATICFILES_STORAGE = 'storages.backends.gcloud.GoogleCloudStorage'
```

Imatge 43: Canvi a Google Cloud Storage

Per indicar que el projecte utilitza Google Cloud Storage per proporcionar els fitxers estàtics i els fitxers de tipus media en comptes del sistema de gestió de fitxers predeterminat, es canvien DEFAULT_FILE_STORAGE i STATICFILES_STORAGE. Seguidament s'inicialitza el nom del bucket utilitzat a partir de GS_BUCKET_NAME, i per obtenir la STATIC_URL es concatena l'URL principal de Google Storage amb el nom definit. I per últim, a través de STATICFILES_DIRS, s'indica quin és el directori que es copiarà al bucket un cop s'executi la comanda collectstatic.

Quan ja s'ha definit el sistema de fitxers, cada cop que es faci un desplegament a Heroku s'executarà la comanda collectstatic i es farà un update de tots els fitxers del bucket en funció dels arxius que hi ha al directori local. Tot i així hi ha fitxers que no es troben a cap directori, ja que són els usuaris qui els creen a través de les funcions proporcionades per l'aplicació. Per tant són necessàries diferents operacions per poder pujar arxius, esborrar-los o simplement obtenir-los. Aquí podem veure un exemple de descàrrega d'un àudio:

```
storage_client = storage.Client()
bucket = storage_client.bucket('lab-fonetica-rek')
blob = bucket.blob(audio)
blob.download_to_filename('media/' + audio + '.wav')
```

Imatge 44: Descarrega d'un àudio

Primer es defineix el nom del bucket i seguidament agafem l'àudio a partir del seu nom, que en aquest cas es troba inicialitzat a la variable 'audio'. Finalment a través del mètode download_from_filename descarreguem el fitxer al directori especificat.

En cas que vulguem eliminar un àudio, el procés seria el mateix, però l'última comanda canviaria i passaria a ser blob.delete(). I si l'operació que volguéssim fer és pujar un arxiu, la comanda final seria upload_from_filename.

11 Testing

11.1 Proves unitàries

Django utilitza la llibreria de Python anomenada unittest per poder fer tots els tests unitaris necessaris. Són creats a l'arxiu test.py i cada un d'ells representa una subclasse de TestCase, que a l'hora és una subclasse de unittest.TestCase.

A l'executar els tests es crea una base de dades que és on es guardaran els objectes inicialitzats, i un cop acabi el procés es destruirà:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.......
----------------------------------------------------------------------
Ran 7 tests in 0.017s

OK
Destroying test database for alias 'default'...
```

Imatge 45: Execució de diferents tests

Perquè un test sigui executable, a cada classe hem de definir una funció que comenci per la paraula test. A continuació veiem un exemple:

```
class FormTest(TestCase):

    def test_form(self):
        Formulario(admin='rek', titulo='formulario 1', numSecciones=5, numRespuestas=0).save()
        Formulario(admin='rek', titulo='formulario 2', numSecciones=2, numRespuestas=1).save()

        forms = Formulario.objects.all()
        self.assertEquals(forms.count(), 2)

        form_db_1 = forms[0]
        self.assertEquals(form_db_1.titulo, 'formulario 1')
        self.assertEquals(form_db_1.numSecciones, 5)

        form_db_2 = forms[1]
        self.assertEquals(form_db_2.titulo, 'formulario 2')
        self.assertEquals(form_db_2.numRespuestas, 1)
```

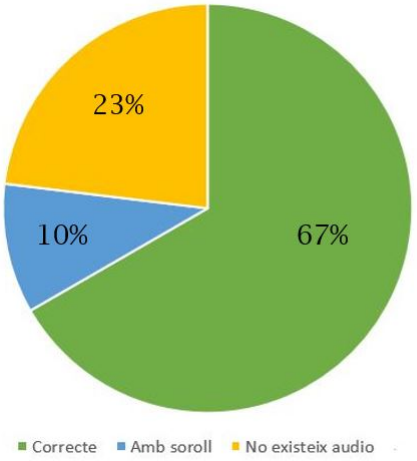
Imatge 46: Test del formulari

Es defineix el mètode test_form que comença amb la paraula test per indicar que és allà el que s'ha d'executar. Dins es creen dos formularis diferents que són guardats a la base de dades. Seguidament amb l'API de Django obtenim tots els formularis existents a la base de dades i fem un count() per veure si coincideix amb el número indicat. Finalment comprovem diferents paràmetres dels dos formularis guardats.

11.2 Proves d'investigadors

Es van realitzar tres iteracions de testing per part dels investigadors del Laboratori de Fonètica que consistien a crear un formulari i distribuir l'enllaç a diferents persones perquè poguessin respondre i així obtenir resultats. Les proves que mostraré han estat dividides en àudios i no en persones que han respost el formulari, ja que els errors apareixien a l'hora de guardar els àudios de manera individual.

Iteració 1



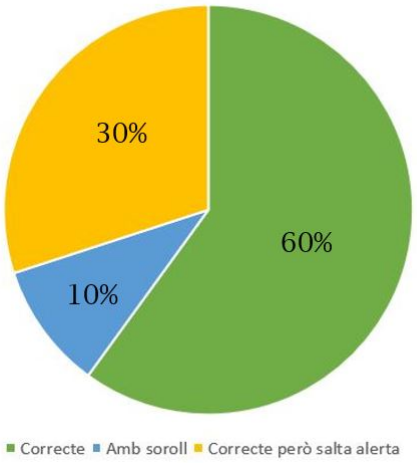
	Número àudios	%
Correcte	26	67 %
Amb soroll	4	10 %
No existeix audio	9	23 %

Imatge 47: Resultats del primer test

Com es pot veure a la imatge, hi va haver 26 àudios que es van guardar correctament. En canvi uns altres 4 es van guardar, però a l'hora de reproduir-los no s'escoltava a l'usuari parlant, sinó que se sentia molt soroll. Per una altra banda hi va haver 9 àudios que no van ser guardats.

D'aquest test podem treure diferents conclusions. Més de la meitat dels àudios van ser guardats amb èxit, enfront del 4% d'àudios que contenien soroll. A més es va poder detectar que el sistema de guardat a través d'AJAX estava causant problemes perquè s'executa de forma asíncrona. Això feia que l'usuari número 1 gravés tres àudios, però abans de guardar-ho la base de dades s'actualitzava fent que l'últim usuari fos l'usuari 2, provocant que l'àudio quedés enllaçat a l'usuari 2 en comptes de l'1. És per això que molts usuaris no tenien assignats cap àudio, fent que un 23% no es recuperessin de forma correcta.

Iteració 2

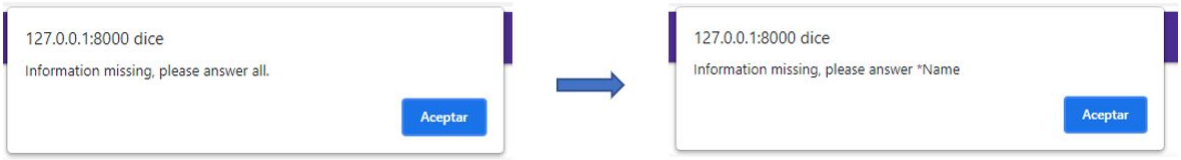


	Número àudios	%
Correcte	12	60 %
Amb soroll	2	10 %
Correcte però salta alerta	6	30 %

Imatge 48: Resultats del segon test

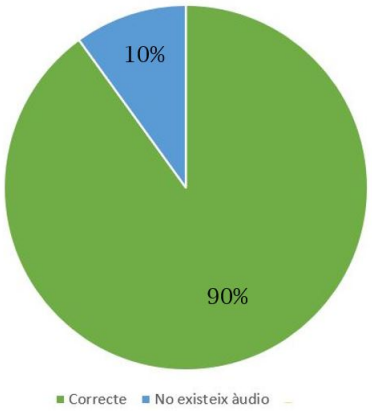
En aquest segon testing 12 dels 30 àudios van ser guardats i recuperats de forma correcta, enfront dels 2 que van ser guardats amb soroll. En canvi un 30% es van guardar de forma correcta però tot i així, a l'hora d'enviar els resultats, a l'usuari li apareixia en pantalla una alerta que indicava que l'àudio no ha pogut guardar-se.

Aquest no era l'únic problema amb les alertes, sinó que alguns usuaris no van poder enviar els seus àudios, ja que alguns paràmetres no estaven omplerts. Això és indicat a l'usuari a través d'alertes, però es va programar un avís tan genèric que no eren capaços de saber quin era el camp que faltava. Per tant es van modificar les alertes perquè, en cas que falti un camp, indiqui quin és el nom del camp que falta.



Imatge 49: Modificació de les alertes

Iteració 3



	Número àudios	%
Correcte	9	90 %
No existeix àudio	1	10 %

Imatge 50: Resultats del tercer test

En aquest cas no hi va haver àudios que presentessin soroll, però hi va haver un àudio que no va ser guardat amb èxit. Aquest error és atribuït a una caiguda a gran escala dels servidors que es va produir el mateix dia del testing i que va afectar Google, i en conseqüència al sistema d'emmagatzematge utilitzat.

Com a conseqüència de totes les proves realitzades juntament amb l'article publicat a la revista científica Journal of the Acoustical Society of America [24], s'ha conclòs que la qualitat i el soroll present a les gravacions dependrà de la forma en que l'usuari gravi i el micròfon que utilitzi. És per això que s'ha d'introduïr com una variable més als estudis realitzats.

12 Conclusions

Després del canvi de planificació que va patir el projecte a causa de la poca compatibilitat de les tecnologies que s'havien utilitzat amb Heroku, es pot dir que l'objectiu de l'aplicació ha estat assolit amb èxit. Malgrat les dificultats que anaven sorgint a mesura que el projecte avançava, s'ha pogut crear una aplicació que fos desplegada al núvol i que complís els requeriments essencials.

El resultat d'aquest projecte permet al Laboratori de Fonètica la recopilació d'àudios a través de formularis. A més es tracta d'una aplicació que ja ha passat per una sèrie de proves per tal de millorar-la i adaptar-la a les necessitats dels investigadors. Per tant s'ha desenvolupat un projecte que permet l'obtenció de resultats d'usuaris de diferents localitzacions geogràfiques.

A més m'ha servit per poder valorar i tenir en compte els riscos i els avantatges d'una decisió concreta en un projecte tecnològic. L'objectiu secundari que s'havia plantejat inicialment va haver de ser descartat per falta de temps a causa d'un problema de versions. Però la supressió d'aquest objectiu ha suposat l'existència d'una aplicació funcional al núvol que a més aporta millores i noves funcionalitats.

Durant els mesos que ha durat el desenvolupament no només he pogut aprendre noves coses, sinó que també he consolidat molts dels coneixements que havia après al llarg del grau universitari. Per tant, poder portar un projecte gran i haver de prendre diferents decisions durant aquest període de temps ha sigut una experiència molt positiva per mi.

12.1 Treballs futurs

Aquesta suposa la segona iteració d'un mateix projecte, però encara hi ha diferents funcionalitats que poden ser afegides en cas d'haver-hi una tercera.

Optimització del temps de càrrega

La utilització de Google Cloud Storage fa que el temps de càrrega sigui elevat quan es necessiten recuperar diferents arxius. Aquest treball futur es tracta de poder millorar el rendiment de la pàgina a l'hora de recuperar diferents fitxers, siguin HTML o àudios [26].

Aquest apartat també inclou la creació de dos sistemes de gestió de fitxers. Un dels dos sistemes seria el responsable de recuperar els fitxers HTML, CSS i JavaScript, que serien guardats a un repositori. I l'altre sistema utilitzaria Google Cloud Storage, i s'encarregaria de recuperar els fitxers i guardar els arxius pujats pels diferents usuaris. Això faria que no s'hagués d'utilitzar un bucket quan es vol accedir a fitxers estàtics que configuren l'estructura i la visualització de l'aplicació.

Sistema de geo-localització

Actualment els resultats són mostrats en una taula d'Excel, però és una idea interessant poder mostrar tots els resultats de manera gràfica en un mapa. La localització geogràfica afecta a la forma de parlar d'una persona, per tant seria una gran implementació poder veure de forma directa tots els resultats d'aquesta forma.

A més crec que es podrien afegir diferents mètriques i a partir d'aquí crear gràfiques segons el que l'investigador vol veure. I utilitzant la funcionalitat de l'Excel, es podria crear un sistema de filtratge que permeti a l'administrador obtenir la taula en funció de variables com la localització geogràfica.

Nou tipus de context

A l'hora de crear un formulari, l'aplicació actual proporciona tres tipus de context:

- Text
- Àudio
- Imatge

Es podrien ampliar tots aquests tipus afegint un de nou que utilitzés un vídeo com a context. A l'hora de desenvolupar-ho es podria aprofitar la implementació creada que permet pujar i guardar imatges.

Utilització d'un servidor propi

Heroku presenta diferents desavantatges que plantegen la possibilitat d'utilitzar un servidor propi:

- El procés de Cycling realitzat al dyno fa que Heroku no sigui capaç de proporcionar un sistema de gestió de fitxers adient. És per això que el projecte necessita utilitzar Google Cloud Storage.
- Quan l'aplicació web passa més de trenta minuts sense ser utilitzada, el dyno entra a un estat repòs que fa que la pàgina trigui més a carregar un cop que algú la torna a fer servir.

És per això que considero una bona opció la utilització d'un servei propi utilitzant la base de dades PostgreSQL i el servidor Apache [27].

Bibliografia

[1] Django Documentation [Online]. Recollit de https://docs.djangoproject.com/en/3.2/

[2] Apache HTTP Server Project [Online]. Recollit de https://httpd.apache.org/

[3] PostgreSQL: The World's Most Advanced Open Source Relational Database [Online]. Recollit de https://www.postgresql.org/

[4] Cursos GIS: ¿Cómo integramos los lenguajes HTML, CSS, y JavaScript? (2015) [Online]. Recollit de https://www.cursosgis.com/como-integramos-los-lenguajes-html-css-y-javascript/

[5] GitHub: Matt Diamond Recorder.js (2016) [Online]. Recollit de https://github.com/mattdiamond/Recorderjs

[6] Amazon Web Services: Cloud computing with AWS [Online]. Recollit de https://aws.amazon.com/what-is-aws/

[7] Symfony: What is Symfony [Online]. Recollit de https://symfony.com/what-is-symfony

[8] Symfony: Ten criteria for choosing the correct framework [Online]. Recollit de https://symfony.com/ten-criteria

[9] Laravel Configuration [Online]. Recollit de https://laravel.com/docs/8.x/configuration

[10] Functional vs Non Functional Requirements (2020) [Online]. Recollit de https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

[11] Patrón MVT: Modelo-Vista-Template [Online]. Recollit de https://docs.hektorprofe.net/django/web-personal/patron-mvt-modelo-vista-template/

[12] Django FAQ [Online]. Recollit de https://docs.djangoproject.com/en/3.2/faq/

[13] Bootstrap: Introduction [Online]. Recollit de https://getbootstrap.com/docs/5.0/getting-started/introduction/

[14] JQuery Fundamentals: A guide to the basics of jQuery [Online]. Recollit de http://jqfundamentals.com/

[15] What is AJAX and How Does It Work? (2021) [Online]. Recollit de https://www.hostinger.com/tutorials/what-is-ajax

[16] GitHub: Stephino Recorder.js (2020) [Online]. Recollit de https://stephino.github.io/dist/recorder.js

[17] Gunicorn - WSGI server [Online]. Recollit de https://docs.gunicorn.org/en/stable/

[18] ¿Qué es Cloud Computing? [Online]. Recollit de https://www.salesforce.com/mx/cloud-computing/

[19] Heroku Architecture [Online]. Recollit de https://devcenter.heroku.com/categories/heroku-architecture

[20] ¿Qué es Cloud Storage? [Online]. Recollit de https://cloud.google.com/storage/docs/introduction

[21] Django ORM Tutorial – The concept to master Django framework (2021) [Online]. Recollit de https://data-flair.training/blogs/django-orm-tutorial/

[22] Deployment Pipeline Approach [Online]. Recollit de https://practical.li/clojure-webapps/projects/banking-on-clojure/deployment-pipeline.html

[23] django-storages: Google Cloud Storage [Online]. Recollit de https://django-storages.readthedocs.io/en/latest/backends/gcloud.html

[24] Comparing acoustic analyses of speech data collected remotely (2021) [Online]. Recollit de https://arxiv.org/ftp/arxiv/papers/2103/2103.01059.pdf

[25] Deploying Python and Django Apps on Heroku (2020) [Online]. Recollit de https://devcenter.heroku.com/articles/deploying-python

[26] django-storages [Online]. Recollit de https://django-storages.readthedocs.io/en/latest/index.html

[27] How to use Django with Apache and mod_wsgi [Online]. Recollit de https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/modwsgi/

13 Annexos

13.1 Casos d'ús

Administrador

Cas d'ús	**UC1**
Objectiu	L'usuari vol iniciar sessió
Actor	Administrador
Precondicions	L'usuari es troba a la pantalla d'iniciar sessió.
Procés	1. L'usuari escriu el nom en el camp corresponent. 2. L'usuari escriu la contrasenya en el camp corresponent. 3. L'usuari prem el botó Entrar. 4. El sistema valida les credencials introduïdes. 5. El sistema redirigeix a la pàgina de formularis.
Excepció	Si les credencials no són correctes, tornarà a la pantalla d'inici de sessió amb un missatge d'error.

Cas d'ús	**UC2**
Objectiu	L'usuari vol tancar sessió
Actor	Administrador
Precondicions	L'usuari s'ha d'haver registrat correctament.
Procés	1. L'usuari es desplaça fins la part d'adalt a la dreta de la pantalla. 2. L'usuari prem el seu nom. 3. El sistema tanca la sessió. 4. El sistema mostra la pantalla principal.

Cas d'ús	**UC3**
Objectiu	L'usuari vol canviar les seves dades personals
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió
Procés	1. L'usuari es desplaça fins el menú de l'esquerre. 2. L'usuari prem el botó Cambiar datos. 3. L'usuari introdueix les dades que vol canviar. 4. L'usuari prem el botó de guardar. 5. El sistema guarda les dades. 6. El sistema redirigeix a la pantalla d'inici de sessió.
Excepció	Si l'usuari s'ha deixat algun camp obligatori, el sistema mostrarà una alerta.

Cas d'ús	**UC4**
Objectiu	L'usuari vol crear un formulari
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.

Procés	1. L'usuari prem el botó de + situat abaix a la dreta per crear un nou formulari. 2. El sistema redirigeix a l'usuari a la pàgina de creació de formularis. 3. L'usuari introdueix el text de la pàgina principal. 4. L'usuari introdueix les dades personals adients pel formulari. 5. L'usuari introdueix el títol del formulari. 6. L'usuari crea diferents contextos. 7. L'usuari guarda prement al botó de guardar situat abaix a la dreta. 8. El sistema guarda les dades. 9. El sistema redirigeix a la pantalla de *Formularios*.
Excepció	Si l'usuari s'ha deixat algun camp obligatori, el sistema mostrarà una alerta. Si el sistema no ha guardat un arxiu, es mostrarà un missatge d'error.

Cas d'ús	**UC5**
Objectiu	L'usuari vol veure un formulari
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.
Procés	1. L'usuari prem l'opció de *VEURE* al menú del formulari corresponent. 2. El sistema redirigeix a l'usuari a la pàgina de visualització del formulari.

Cas d'ús	**UC6**
Objectiu	L'usuari vol editar un formulari
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.
Procés	1. L'usuari prem l'opció d'*EDITAR* al menú del formulari corresponent. 2. El sistema redirigeix a l'usuari a la pàgina d'edició de formularis. 3. L'usuari afegeix, elimina o modifica l'apartat que vulgui. 4. L'usuari guarda prement al botó de guardar situat abaix a la dreta. 5. El sistema guarda les dades. 6. El sistema redirigeix a la pantalla de *Formularios*.
Excepció	Si l'usuari s'ha deixat algun camp obligatori, el sistema mostrarà una alerta. Si el sistema no ha guardat un arxiu, es mostrarà un missatge d'error.

Cas d'ús	**UC7**
Objectiu	L'usuari vol eliminar un formulari
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.
Procés	1. L'usuari prem l'opció d'*ELIMINAR* al menú del formulari corresponent. 2. El sistema borra totes les dades del formulari. 3. El sistema torna a carregar la pàgina de *Formularios*.

Cas d'ús	**UC8**
Objectiu	L'usuari vol obtenir l'enllaç del formulari
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.
Procés	1. L'usuari prem l'opció d'*URL* al menú del formulari corresponent. 2. El sistema mostra una alerta amb l'enllaç corresponent. 3. L'usuari copia l'enllaç.

Cas d'ús	**UC9**
Objectiu	L'usuari vol descarregar les respostes
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar al menú de *Formularios*.
Procés	1. L'usuari prem l'opció de *Descargar audios* al menú del formulari corresponent. 2. El sistema redirigeix a l'usuari a la pàgina de descarrega. 3. El sistema descarrega automàticament els fitxers en format zip. 4. El sistema redirigeix a l'usuari a la pàgina de *Formularios*.

Cas d'ús	**UC10**
Objectiu	L'usuari vol gravar un àudio
Actor	Administrador
Precondicions	L'usuari ha d'haver iniciat sessió i ha d'estar a la pantalla de creació o edició d'un formulari.

Procés	1. L'usuari prem el botó de *Record audio*. 2. El sistema demana permisos per poder utilitzar el micrófon. 3. L'usuari accepta els permisos. 4. L'usuari grava l'àudio. 5. L'usuari prem el botó d'Stop audio. 6. El sistema mostra l'àudio gravat.
Excepció	Si l'usuari no accepta els permisos, no es podrà gravar l'àudio.

Usuari no registrat

Cas d'ús	**UC11**
Objectiu	L'usuari vol accedir a un formulari
Actor	Usuari no registrat
Precondicions	L'usuari necessita tenir l'enllaç del formulari corresponent.
Procés	1. L'usuari introdueix l'enllaç al navegador. 2. El sistema mostra la pàgina principal del formulari. 3. L'usuari prem el botó d'*START*. 4. El sistema mostra el formulari.
Excepció	Si l'enllaç proporcionat és incorrecte, es mostrarà un error.

Cas d'ús	**UC12**
Objectiu	L'usuari vol introduïr les dades personals
Actor	Usuari no registrat
Precondicions	L'usuari necessita haver accedit al formulari.

Procés	1. L'usuari introdueix les seves dades als camps corresponents. 2. L'usuari prem el botó de *FINISH*. 3. El sistema valida les dades. 4. El sistema guarda les dades.
Excepció	Si algún camp no s'ha introduït adequadament, es mostrarà una alerta.

Cas d'ús	**UC13**
Objectiu	L'usuari vol enviar enviar una resposta
Actor	Usuari no registrat
Precondicions	L'usuari necessita haver accedit al formulari.
Procés	1. L'usuari introdueix tots els camps demanats. 2. L'usuari grava els àudios necessaris. 3. L'usuari prem el botó de *FINISH*. 4. El sistema valida les dades. 5. El sistema guarda les dades. 6. El sistema mostra una pantalla d'agraïment.
Excepció	Si algún camp no s'ha introduït adequadament, es mostrarà una alerta. Si un àudio no es guarda correctament, es mostrarà un error.

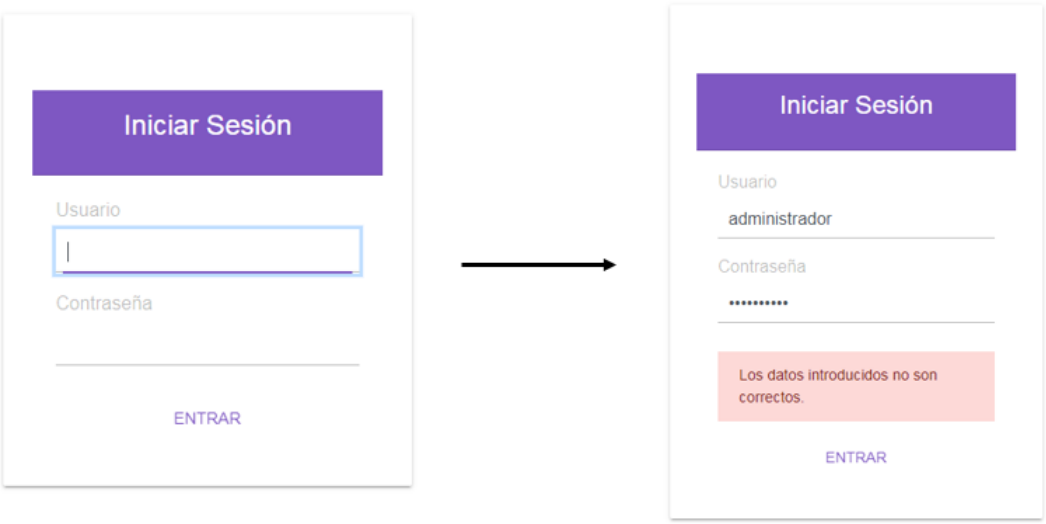
13.2 Manuals d'usuari

13.2.1 Recording Context

Per poder començar a utilitzar l'aplicació web i crear formularis és necessari iniciar sessió com a administrador. Per poder fer-ho s'hi ha de prémer a la part superior dreta de la pàgina inicial.

Imatge 51: Inici de sessió

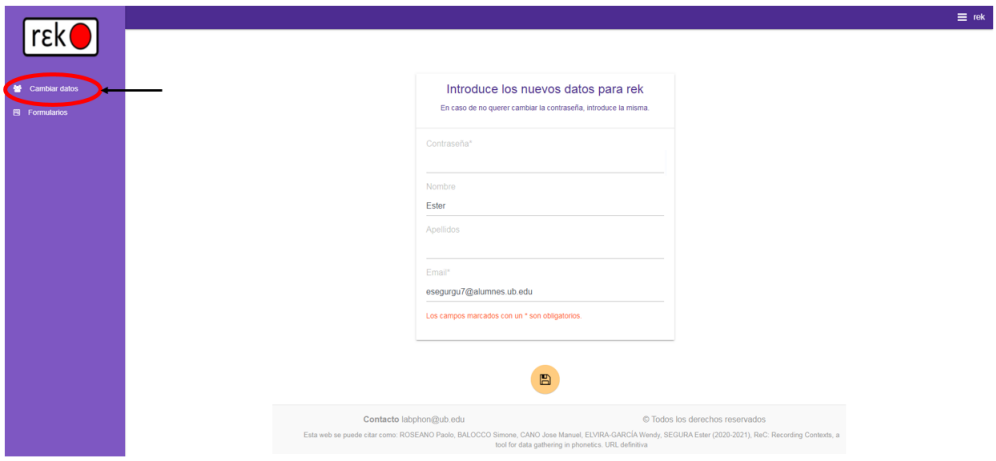
Llavors apareixerà una pantalla d'inici de sessió on haurem d'introduir les credencials adients per poder accedir. En cas de produir-se un error es mostrarà un missatge. Si l'inici de sessió s'efectua amb èxit,ens redigirà a la pàgina principal de l'administrador.



Imatge 52: Mostra de l'error al iniciar sessió

Canviar les dades personals de l'administrador

Les dades de l'administrador es poden modificar a través de l'opció "Cambiar datos" del menú que es troba a l'esquerra.



Imatge 53: Opció per canviar les dades

Aquest apartat permet canviar la contrasenya, el nom, el cognom i el mail de l'usuari. Per poder guardar els canvis és necessari introduir la contrasenya actual. En cas que la vulguem modificar llavors hem d'introduir la nova contrasenya. A més també és obligatori introduir el mail, ja que no pot quedar mai en blanc.

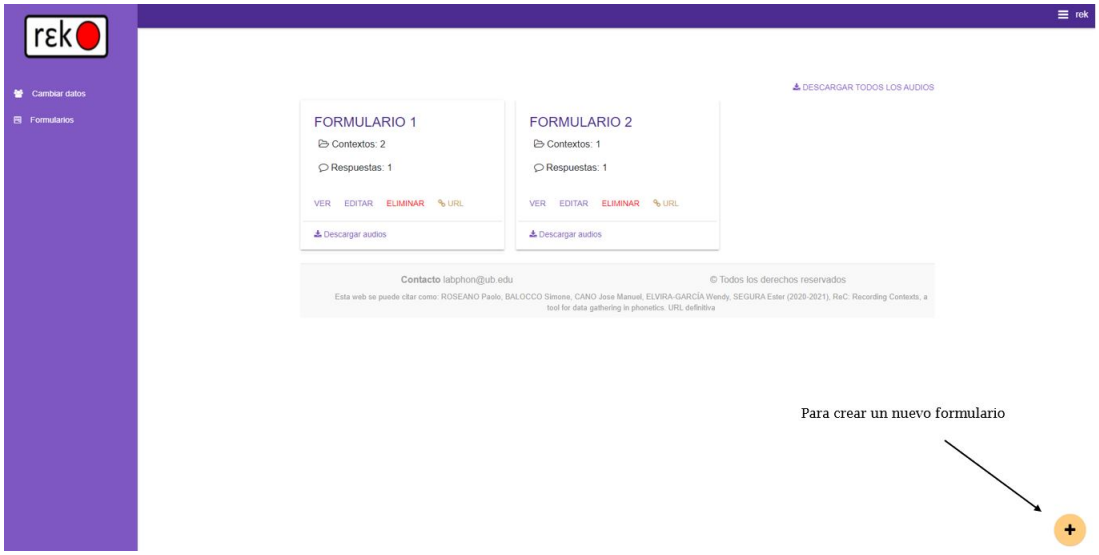
Un cop haguem canviat les dades, hem de prémer el botó de guardar situat a baix del formulari. Això farà que es tanqui la sessió que teníem iniciada, i ens redigirà a la pantalla d'inici de sessió.

Crear un formulari

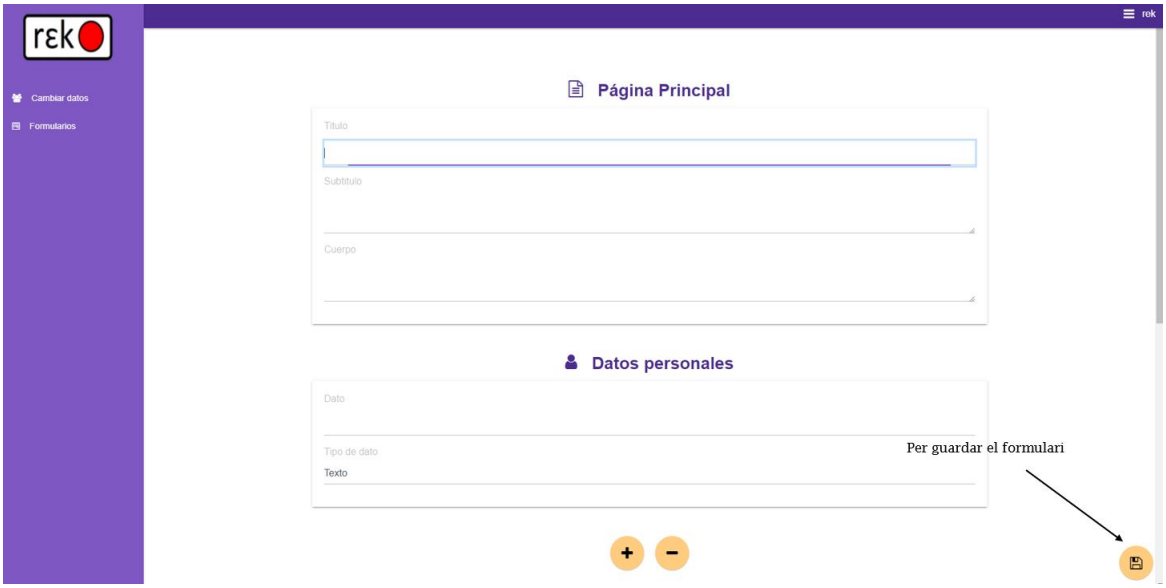
Per poder crear un nou formulari és necessari prémer l'opció del menú anomenada "Formularis". Allà es mostraran tots els formularis existents, i en cas que no hi hagi cap apareixerà un missatge indicant-ho. Un cop en aquesta pantalla, s'hi ha de prémer al botó amb una icona de +.

Això ens redigirà a la pantalla de creació, separada en tres apartats:

- Pàgina Principal
- Dades personals
- Contextos

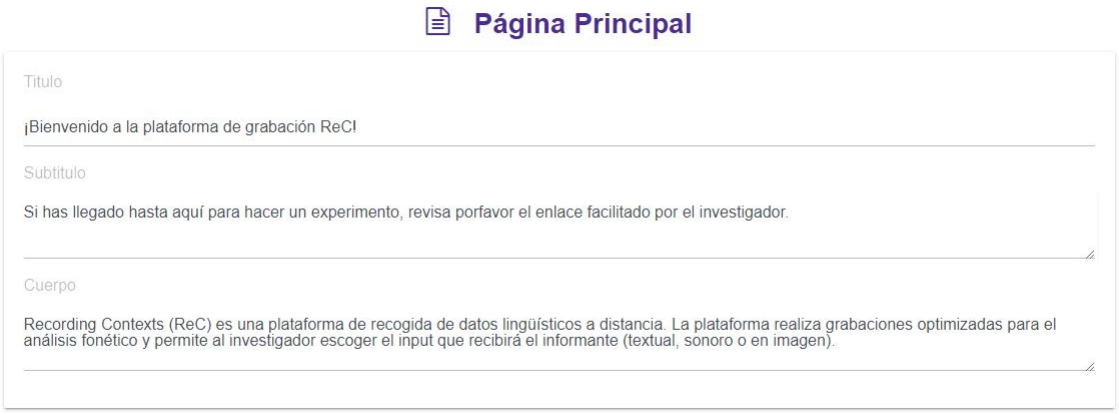


Imatge 54: Crear un nou formulari



Imatge 55: Pantalla de creació de formularis

Pàgina Principal: Escriurem el que volem que l'usuari vegi per primer cop a l'entrar a l'enllaç del formulari. Ha de ser un text explicatiu, i tenim la possibilitat d'afegir un subtítol i un cos que poden estar en blanc. En canvi el títol s'hi ha d'introduir de manera obligatòria.



Imatge 56: Pàgina Principal

Dades personals: Aquí escriurem les dades que volem conèixer de l'usuari que respongui. És obligatori introduir com a mínim una dada. S'ha implementat un sistema que valida la dada en funció del tipus:

- Text
- Data
- Resposta de sí o no

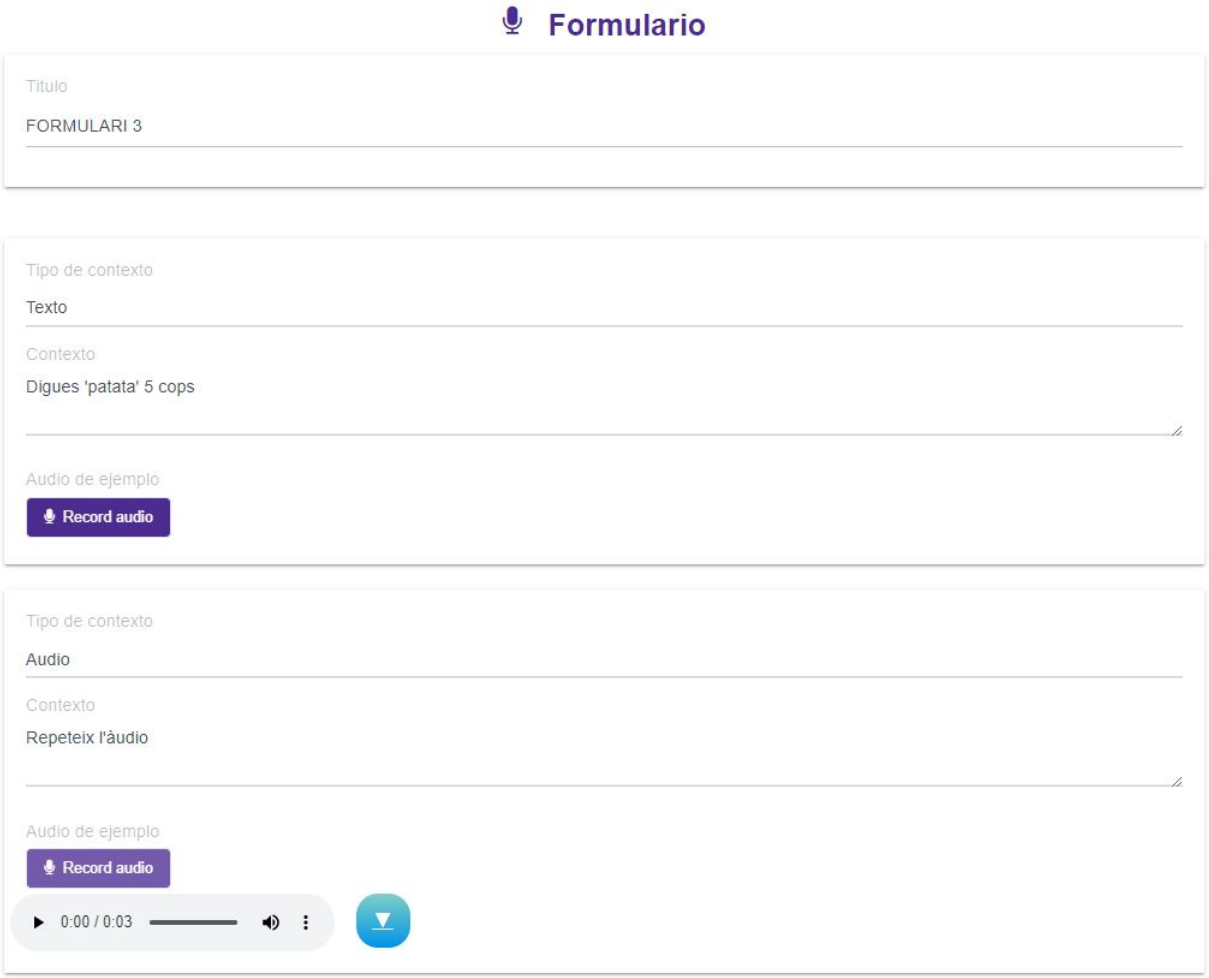
A través dels botons que es troben abaix es pot afegir una nova dada o borrar una anterior.

Imatge 57: Dades personals

Contextos: En aquest apartat és on es creen els contextos perquè l'usuari pugui respondre. A més és on es defineix el títol del formulari. Hi ha tres tipus de context:

- Text
- Audio
- Imatge

En qualsevol de les tres opcions està la possibilitat d'afegir un text perquè acompanyi al context. Per poder afegir un, o esborrar un anterior es fa de la mateixa manera que a les dades personals. Per gravar un àudio es fa prement el botó on posa "Record audio", i per parar-lo es fa prement-lo un altre cop.



Imatge 58: Creació de contextos

Un cop omplert tot, per a que el formulari quedi registrat s'ha de premer al botó de guardar.

Editar un formulari

Quan ens trobem a la pàgina de "Formularios" es mostren tots els que hem creat, i dins de cada un tenim diferents opcions. Per poder editar el formulari hem de prémer l'opció EDITAR.

Imatge 59: Editar formulari

El funcionament d'aquesta pàgina és el mateix que a la pàgina de creació de formularis. Podem afegir o esborrar antics contextos i dades personals. A més podem modificar el text de qualsevol apartat.

Imatge 60: Opcions per editar

Veure un formulari

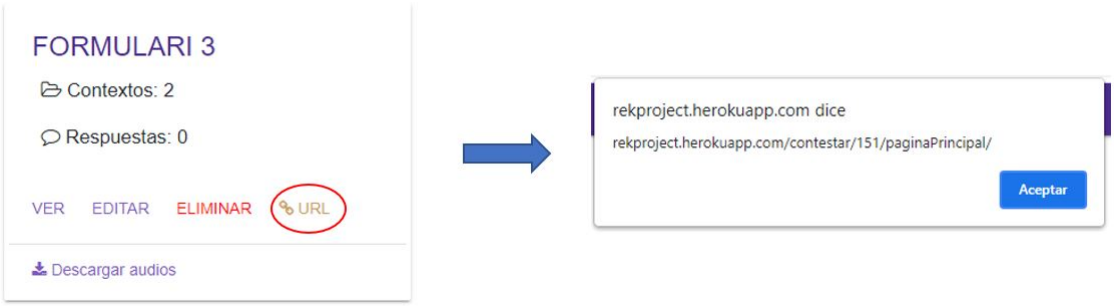
Per poder veure com ha quedat un formulari sense haver de fer-ho a través de l'enllaç s'ha de prémer a l'opció de VER.

Imatge 61: Veure formulari

Aquesta opció no ens donarà la possibilitat d'editar ni de respondre al formulari. Només és una previsualització de l'estat del formulari.

Accedir a un formulari a través de la URL

Per poder obtenir l'URL que serà utilitzada pels usuaris a l'hora de contestar el formulari, s'hi ha de prémer l'opció URL.



Imatge 62: URL del formulari

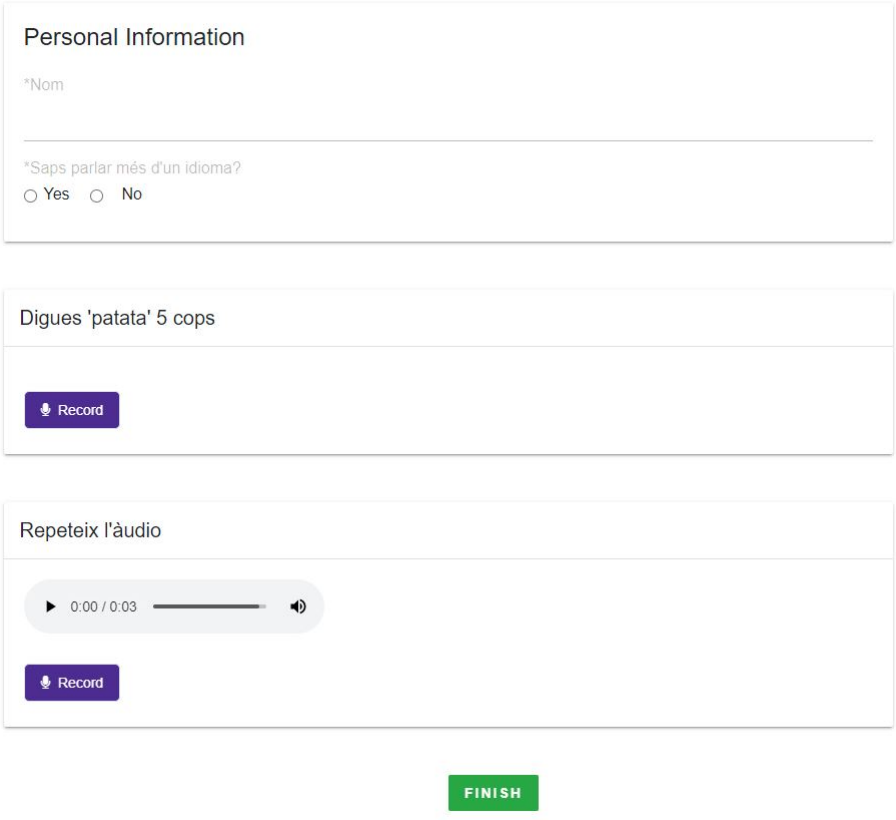
Això farà que es mostri una alerta indicant quina és l'URL que ha de ser distribuïda. Quan aquest enllaç sigui introduït al navegador, primer es mostrarà la pàgina principal definida i després el formulari per respondre.

Respondre un formulari

Per començar a la pàgina principal hem de prèmer el botó d'START.

Imatge 63: Pàgina Principal del formulari

Seguidament s'ens mostrarà el formulari, on s'haurà d'introduir una resposta per cada un dels camps, sigui de dades personals o de context. Finalment per poder enviar-ho s'hi haurà de prémer el botó verd on posa escrit "FINISH".



Imatge 64: Cos del formulari per respondre

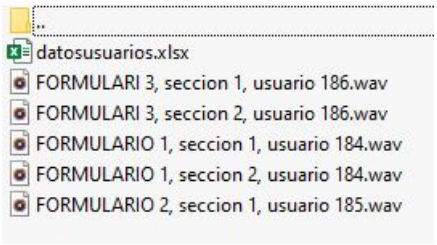
Descarregar les respostes

Quan volem descarregar les respostes dels usuaris tenim dos possibilitats:

1. Descarregar només les d'un formulari a través de l'opció "Descargar audios".
2. Descarregar les respostes de tots els àudios a través de l'opció "Descargar todos los audios".

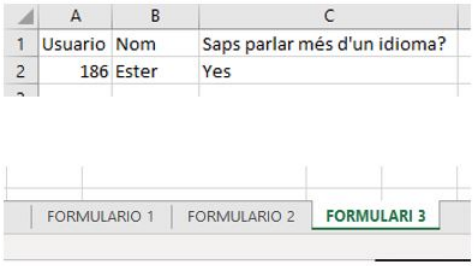
Imatge 65: Descàrrega d'àudios

En qualsevol dels casos s'ens descarregarà un zip on es troben tots els àudios gravats i un excel amb les respostes. Els noms dels àudios estan compostos del títol del formulari + secció + ID del usuari.



Imatge 66: Estructura del zip

En el cas de l'excel, les columnes representen totes les dades personals, i les files fan referència l'usuari indicant el seu ID. Si hem escollit l'opció de descarregar les respostes de tots els formularis, cada un d'ells estarà en una fulla diferent de l'excel.



Imatge 67: Estructura de l'excel

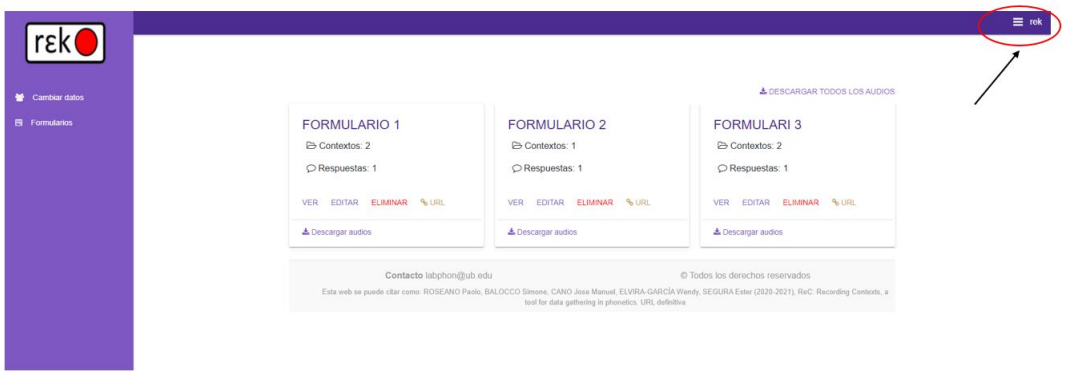
Eliminar un formulari

Per eliminar un formulari només és necessari prémer el botó d'ELIMINAR del formulari corresponent.

Imatge 68: Eliminar formulari

Tancar sessió

Finalment per poder tancar la sessió hem de prémer el nom de l'usuari que es mostra adalt a la dreta.



Imatge 69: Tancar sessió

13.2.2 Administració de Django

Per poder accedir afegim '/admin/' a l'enllaç principal de la pàgina web, i es mostrarà la següent pàgina que conté tots els models definits representant les taules de la base de dades.

Imatge 70: Pantalla principal

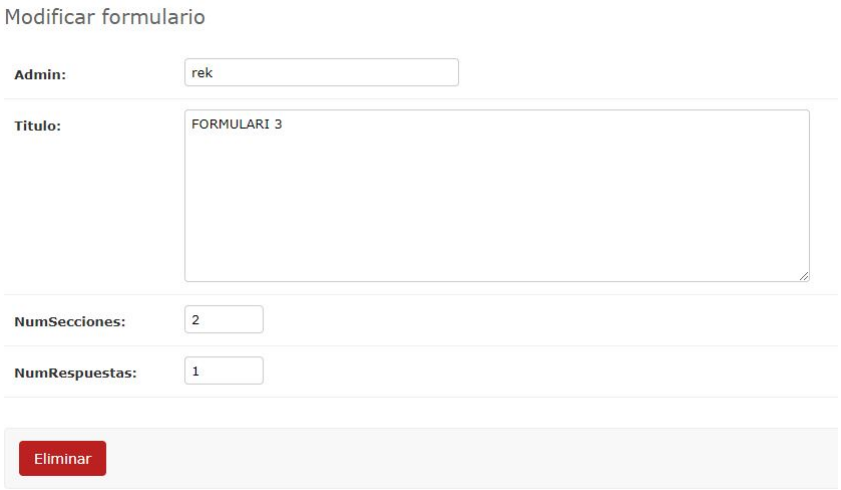
Es pot afegir o modificar qualsevol objecte de la taula a partir d'aquesta pàgina, però també podem prémer un concret per entrar a veure en detall tots els objectes creats. Per exemple si entrem a "Formularios" veurem això:

Imatge 71: Model "Formulario"

En el moment que seleccionem un objecte, tenim la possibilitat d'esborrar-lo de la taula.

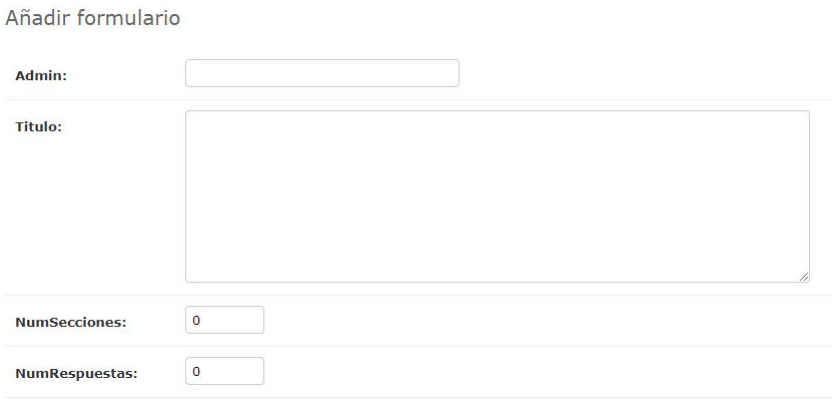
Imatge 72: Eliminar objecte

Si volem modificar un formulari hem de premer al titol del formulari i es redigirà a la pàgina d'edició.



Imatge 73: Modificar objecte

També tenim la possibilitat d'afegir un de nou prement al botó "Añadir Formulario". Es mostrarà una pantalla amb tots els camps que han de ser omplerts, ja que són els definits al model el Formulari.

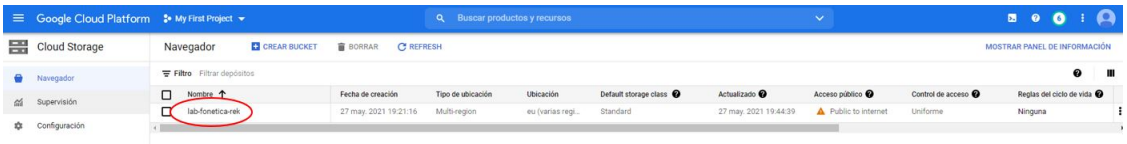


Imatge 74: Afegir objecte

Tots els canvis realitzats en aquesta interfície seran reflectits en la visualització de l'aplicació web. Si per exemple introduïm un nou formulari, es veurà a la pàgina de formularis de l'aplicació.

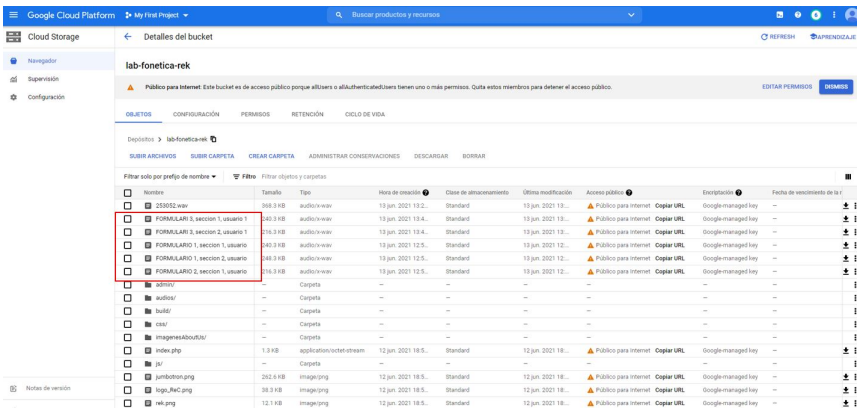
13.2.3 Google Cloud Storage

Per poder accedir és necessari entrar a la consola des de Google Cloud Platform. Un cop aquí busquem Cloud Storage al navegador, i apareixerà la següent pantalla:



Imatge 75: Pantalla principal de Google Cloud Storage

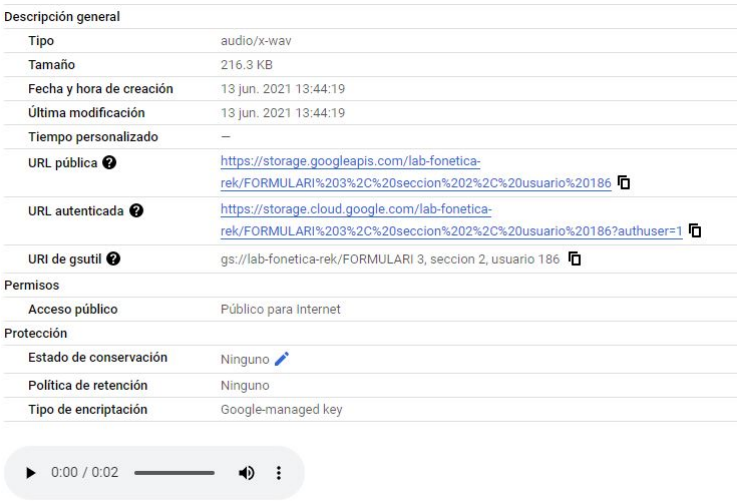
El que està marcat en vermell és el bucket on es troben tots els fitxers. Si premem apareixerà la pantalla que es mostra a continuació.



Imatge 76: Fitxers

Aquí estan guardats tant les imatges que són utilitzades a l'aplicació, com els fitxers CSS que apliquen l'estil a la pàgina.

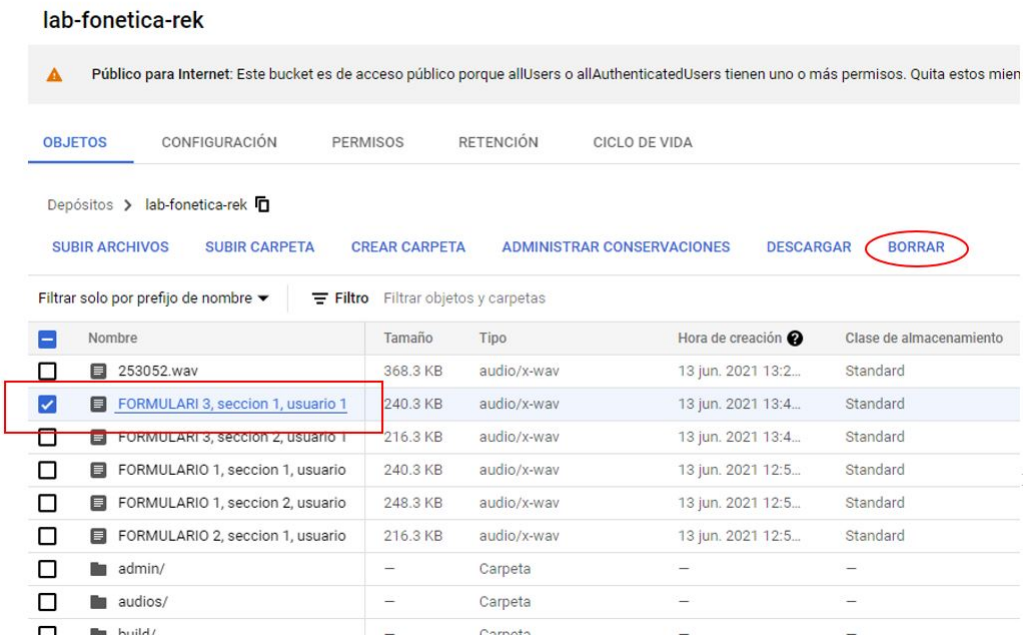
El que està marcat en vermell són els àudios que es descarreguen juntament amb l'excel quan premem l'opció de descarregar. És a dir, són les respostes dels usuaris. Si premem un concret veurem això:



Imatge 77: Detall de l'àudio

Ens permet veure els detalls de l'àudio, com poden ser les URLs d'accés o la data de publicació. A més podem escoltar l'àudio o descarregar-lo.

Per finalitzar, també és possible esborrar els fitxers que vulguem. Només s'hi haurien d'esborrar els àudios enviats pels usuaris, o les imatges i els àudios que formen part del formulari. Els demès arxius no haurien de ser esborrats, ja que canviaria el comportament de l'aplicació. I en cas que s'esborrin els àudios, a l'hora de mostrar el formulari no es podran recuperar i es produiran errors.



Imatge 78: Esborrar un àudio