Sistema colaborativo para el etiquetado de datos

Collaborative system for data labeling

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de fin de grado

Grado en ingeniería informática

Facultad de informática

Curso 2021/2022

Autores:

Gemma González Palacios (Grado en Ingeniería informática)

Adrián Salvador Crespo (Grado en Ingeniería informática)

Directores:

Enrique Martín Martín

Adrián Riesco Rodríguez

Resumen

Actualmente, al utilizar técnicas de aprendizaje automático es necesario pasar por una etapa de etiquetado manual de datos para entrenar al software. Este proceso de etiquetado puede resultar muy tedioso y ocupar una gran cantidad de tiempo porque en muchas ocasiones debe realizarse de manera manual.

Este proyecto consiste en implementar un sistema colaborativo para etiquetar datos de forma más eficiente y fiable. La estrategia que hemos decidido para desarrollar este sistema es implementar un servicio de CAPTCHA. Este tipo de servicios suele funcionar presentando un reto que normalmente consiste en seleccionar los elementos dentro de un conjunto de datos que cumplen una condición determinada, por lo que las respuestas que envían los usuarios que realizan estos retos pueden servir para clasificar esos elementos que han seleccionado. Sin darse cuenta, los usuarios a los que el reto haya verificado como reales, están realizando un proceso de etiquetado de datos.

Palabras clave

Etiquetado de datos, CAPTCHA, Python, JavaScript, Bootstrap, Django.

Abstract

Currently, when using machine learning techniques, it is necessary to go through a stage of manual data labeling to train the software. This data labeling process can be very tedious and take a lot of time because often it is done manually.

This project consists of implementing a collaborative system to label data more efficiently and reliably. The strategy we have decided to develop this system is to implement a CAPTCHA service. This type of service usually works by presenting a challenge that generally consists of selecting the elements within a set of data that meet a certain condition, so the answers sent by users who perform these challenges can be used to classify those elements they have selected. Without being aware, users who have been verified as real by the challenge are performing a data labeling process.

Key words

Data labeling, CAPTCHA, Python, JavaScript, Bootsrap, Django.

Índice de contenidos

1 Introducción

En esta sección explicaremos la motivación que nos ha llevado a realizar este proyecto, los objetivos que nos proponemos alcanzar y la planificación para el desarrollo del proyecto.

1.1 Motivación

En la actualidad el uso de aplicaciones que se apoyan en inteligencia artificial está muy extendido, desde programas que se emplean para reconocimiento facial, hasta herramientas de dibujo que ayudan a completar un diseño desde un boceto o una descripción. Pero para que estas aplicaciones funcionen correctamente necesitan información preparada para entrenar sus funcionalidades.

La creación de estos conjuntos de datos etiquetados puede ser un proceso lento y tedioso; además, estos conjuntos que se generan muchas veces han sido construidos por una sola persona, lo que puede provocar que en el proceso de etiquetado influyan sesgos personales y no sea imparcial. Por ello surge la necesidad de crear herramientas con las que otras personas puedan clasificar datos de forma rápida, sencilla y revisable, es decir que varias personas den etiquetas al mismo conjunto de datos.

Hoy en día, la mayoría de las aplicaciones web nos piden comprobar que somos seres humanos los que estamos utilizando su servicio. Para diferenciar entre un usuario real y un programa automatizado se emplean servicios de CAPTCHA que mediante un sencillo reto realizan esa comprobación. Uno de los retos más empleados es indicar en un conjunto de datos, cuales cumplen cierta condición, es decir, estos retos pueden servir de ayuda para clasificar información, por lo que hemos desarrollado un servicio de CAPTCHA que recoge la información proporcionada en los retos para etiquetar datos.

1.2 Objetivos

El principal objetivo del proyecto es crear un servicio de CAPTCHA que con la información que reciba en respuesta a los retos que proponga ayude a etiquetar conjuntos de datos y genere un fichero con la información deseada.

En concreto, las metas de este proyecto son los siguientes:

1. Investigar sobre el uso de servicios de CAPTCHA para el etiquetado de datos.
2. Desarrollar un servicio de CAPTCHA fácil de integrar en una aplicación web.
3. Crear un servicio funcional y escalable para que pueda ser sencillo añadir nuevos tipos de datos para etiquetar.

4. Conseguir que el proceso para añadir colecciones de datos y recuperar la información etiquetada sea lo más sencillo posible.

1.3 Plan de trabajo

Para el desarrollo del trabajo hemos realizado una planificación que nos permita cubrir las necesidades de un proyecto de estas características, análisis del proyecto, investigación de servicios de CAPTCHA, investigación de herramientas para desarrollar nuestro servicio de CAPTCHA, implementación, testeo y escritura de la memoria.

Estás tareas consisten en lo siguiente:

- Análisis del proyecto: Estudio de las diferentes formas para producir conjuntos de datos etiquetados.
- Investigación de servicios de CAPTCHA: Aprendizaje de cómo funcionan otros servicios de CAPTCHA.
- Investigación de herramientas para desarrollar nuestro servicio de CAPTCHA: Análisis y elección de las posibles herramientas que podríamos utilizar para el desarrollo del servicio de CAPTCHA.
- Implementación: Fase de desarrollo del servicio de CAPTCHA y del componente utilizable para otras aplicaciones web.
- Testeo: Comprobación del correcto funcionamiento del servicio y corrección de errores.
- Escritura de la memoria: Creación del documento que describe el proceso de desarrollo del trabajo y sus resultados.

En la figura 1.1 se indica en forma de diagrama de Gantt, cómo se ha repartido el trabajo a lo largo del tiempo.

Figura 1.1 - Diagrama de Gantt de la planificación del proyecto

1.4 Licencia y código

El proyecto cuenta con la licencia Apache versión 2.0.

Todos los recursos del proyecto están disponibles en el siguiente enlace:

https://github.com/AdriSC/tfg

2 Introduction

In this section we will explain the motivation that has made us to do this project, the objectives we want to achieve and the planning for the development of the project.

2.1 Motivation

Today, the use of applications that rely on artificial intelligence is widespread. From programs used for facial recognition, to drawing tools that help to complete a design from a sketch or a description. But for these artificial intelligences to work properly, they need information prepared to train their functionalities.

But the creation of these labeled datasets can be a slow and tedious process, and the datasets that are generated have often been made by a single person, which can lead to personal biases influencing the labeling process and making it biased. This is why there is a need to create tools with which other people can classify data quickly, easily and reliably, that is, several people can label the same dataset.

Nowadays, most web applications ask us to verify that we are human beings using their service. To differentiate between a real user and an automated software, CAPTCHA services are used to perform this check by means of a simple challenge. One of the most used challenges is to indicate in a set of data, which meet a certain condition. These challenges can help to classify information, so we have developed a CAPTCHA service that collects the information provided in the challenges to tag data.

2.2 Objectives

The main goal of the project is to create a CAPTCHA service that, with the information it receives in response to the challenges it provides, helps to label datasets and generates a file with the requested information.

In particular, the goals of this project are the following:

1. To investigate the use of CAPTCHA services for data labeling.
2. To develop a CAPTCHA service that is easy to integrate into a web application.
3. To create a functional and scalable service so that it can be easy to add new data types for labeling.
4. Make the process of adding data collections and retrieving labeled information as simple as possible.

2.3 Workplan

For the development of the work, we have made a planning that allows us to cover the needs of a project of these characteristics, analysis of the project, research of CAPTCHA services, research of tools to develop our CAPTCHA service, implementation, testing and writing of the project report.

These tasks consist of the following:

1. Project analysis: Study of the different ways to generate labeled datasets.
2. CAPTCHA service research: Learning how other CAPTCHA services work.
3. Research of tools to develop our CAPTCHA service: Analysis and choice of possible tools we could use for the development of the CAPTCHA service.
4. Implementation: Development phase of the CAPTCHA service and the component which can be used for other web applications.
5. Testing: Checking the correct functioning of the service and correcting errors.
6. Writing the project report: Creation of the documentation describing the development process of the work and its results.

The illustration 2.1 shows, in the form of a Gantt chart, how the work has been distributed over time.

Figura 2.1 - Gantt chart of the workplan

2.4 License and code

The project is licensed under Apache license version 2.0.

All the project resources can be accessed through the following link:

https://github.com/AdriSC/tfg

3 Preliminares

En esta sección presentaremos varios temas. En primer lugar, qué es un CAPTCHA y qué beneficios nos puede aportar, los resultados de la investigación de otros servicios de CAPTCHA y las tecnologías que hemos elegido para implementar el proyecto.

3.1 CAPTCHA

Un CAPTCHA (término acuñado en el año 2000 por Luis Von Ahn), es una prueba de Turing automatizada para diferenciar a un ser humano de una máquina. Es un sistema de verificación utilizado por toda clase de páginas web, con el objetivo de comprobar que aquello que esté interactuando con la página sea realmente una persona dispuesta a utilizar el sitio de la manera para la que fue diseñado. El peligro predominante del cual protegen los CAPTCHAs son los *bots* de spam, los cuales se encargan de llevar a cabo la misma acción repetitivamente. Entre sus actividades se encuentran las siguientes:

1. Enviar correos electrónicos en masa con la intención de encontrar cuentas activas a las que enviar publicidad fraudulenta.
2. Crear cuentas de correo electrónico con el objetivo de utilizarlas para obtener el mayor número de pruebas gratuitas posible, derivando en su venta a un precio inferior al que ofrece la empresa que oferta dichos productos [1].
3. Añadir reseñas positivas sobre negocios/productos en masa para que la valoración global aumente y las reseñas reales (probablemente negativas) queden ocultas al primer *scroll* del usuario.

Desde su introducción, han surgido distintas variaciones de este sistema entre las que se encuentran [2]:

- De texto: este tipo fue el primer introducido y se trata de mostrar una cadena de caracteres y pedir la interpretación de estos. Para añadir dificultad estos caracteres se muestran en maneras como con una rotación o tamaño diferentes, o distorsionados. También pueden estar solapados con elementos gráficos como colores, líneas, arcos o puntos. Con el objetivo de confundir a los *bots* con reconocimientos de texto deficientes.
- De audio: Desarrollado como alternativa para aquellas personas con discapacidad visual. Se trata de un audio con los caracteres a introducir. La dificultad se encuentra en que se introduce en la grabación ruidos de fondo, con un volumen superior al de los caracteres.
- De imagen: Fueron desarrollados para sustituir a los de textos. Algunos muestran una fotografía fragmentada en cuadrículas, pidiendo identificar en ella donde se encuentra un determinado objeto (ej. semáforos, barcos). Otros muestran en cada cuadrícula una foto distinta y se debe identificar aquellas en las que se

encuentre el objeto mencionado. En algunos casos, el objeto no se encuentra en ninguna de las cuadrículas. Este tipo es más fácil de identificar para humanos que el de texto, así como para los *bots*.

- De problemas matemáticos o con palabras: Se trata de mecanismos que requieren la solución a un problema matemático como "2+7" ó "9-2". Otra variante es la de completar una frase a la que le falta una palabra. Este tipo es accesible para las personas con discapacidad visual también. Sin embargo, también son más fáciles de resolver para los *bots*, lo que puede derivar en una brecha de seguridad.
- *Login* con redes sociales: hay páginas que, para facilitar el inicio de sesión, piden a los usuarios acceder con la cuenta de una red social como Facebook o Google.
- Sin CAPTCHA reCAPTCHA: Esta clase únicamente requiere hacer clic en una casilla con la frase "No soy un robot". Se registran los movimientos del ratón y se decide si éstos corresponden a los de un humano o no para pasar la prueba. Es difícil no pasar la prueba y en este caso se deriva a un CAPTCHA de imagen.

3.2 Beneficios adicionales que se han obtenido de los CAPTCHA

En la actualidad, la gran mayoría de aplicaciones web utiliza un servicio de CAPTCHA, aunque algunos servicios opten por emplear CAPTCHAs invisibles, los CAPTCHA visuales o de imagen siguen siendo uno de los más utilizados.

Unido a esto, dentro del campo de la inteligencia artificial se está avanzando bastante, llegando a tener hoy en día programas que pueden realizar funciones como:

- El reconocimiento de algún tipo de imágenes, desde la identificación de emociones o personas analizando el rostro humano, hasta aplicaciones que usan inteligencia artificial para analizar muestras de compuestos químicos y otros materiales.
- Comprender y procesar mensajes de texto y archivos de sonido en lenguaje natural.

Por esta razón, los servicios de CAPTCHA se convierten en buenas herramientas para el etiquetado de datos. Si al hacer el reto del CAPTCHA incluyes conjuntos de datos etiquetados junto a conjuntos de datos sin etiquetar puedes utilizar el primero para verificar si un usuario es legítimo o no y, tras la comprobación, aprovechar las respuestas dadas a los conjuntos para poder validar las etiquetas de los conjuntos de datos que tienen y etiquetar nuevos datos con los conjuntos que no tienen.

3.3 Análisis de otros servicios de CAPTCHA

Antes de comenzar con el desarrollo de nuestro CAPTCHA, estudiamos una serie de CAPTCHAs conocidos. Leímos la documentación, revisamos el código y realizamos pruebas de funcionalidad. A continuación, se detalla la información obtenida de cada uno.

3.3.1 ReCaptcha

Este es el servicio de CAPTCHA desarrollado por Google [3].

Hay varias formas de utilizar este servicio. El método principal para integrar el reCaptcha en una aplicación web es incluir la llamada al *script*. Google recomienda que esta llamada se realiza de forma asíncrona para minimizar el impacto de la carga, ya que no afectará su capacidad de reconocer usuarios ilegítimos.

También existe la posibilidad más personalizable de crear tu propio *script* para solicitar un CAPTCHA, haciendo uso de la biblioteca que Google ofrece para reCaptcha.

Primera Versión

Inicialmente el reto que mostraba para verificar al usuario era un CAPTCHA de texto. Este reto consistía en mostrar al usuario dos palabras una de las palabras era conocida para el sistema, la otra, era una palabra imposible de reconocer para un sistema óptico de lectura de caracteres (sistema OCR).

Más tarde comenzó a sustituir el sistema de palabras por imágenes que contenían fragmentos de texto, como calles o números de portales, que obtenían del servicio Google Street View.

Segunda y tercera versión

A partir de estas versiones el componente que mostraba el reto empezaba mostrando un botón con la etiqueta “No soy un robot”, al hacer clic, mostraba un conjunto de imágenes y se debían seleccionar aquellas que cumplían la misma condición.

Actualmente, el servicio de reCaptcha se ha dirigido a ofrecer un CAPTCHA que sea casi invisible para el usuario. Sigue mostrando el botón descrito anteriormente, solo que ahora el botón inicia un proceso por el cual reCaptcha analiza como el usuario interactúa con la aplicación web para decidir si es legítimo o no.

El *script* de reCaptcha funciona de la siguiente manera:

1. Se carga junto con la página web que incluye el *script* el botón para iniciar la comprobación
2. Al hacer clic, el *script* envía al servidor la información que ha recabado sobre la interacción del usuario con la aplicación web
3. El servidor analiza la información recibida y envía la respuesta de la verificación a la aplicación web.

3.3.2 hCaptcha

Muy parecido al reCAPTCHA de Google. Incluye todas las funcionalidades de sus versiones 2 y 3. El matiz que presenta es la posibilidad de modificar la dificultad de los retos, según el nivel de seguridad que el administrador de la página web considere necesario. Existen tres niveles de dificultad básicos: fácil, medio y difícil. También se ofrece la opción auto, con la cual el CAPTCHA [4] mostrará retos de cualquier dificultad. Las licencias de empresa tienen acceso a otros dos modos:

- Pasivo o Captcha invisible: no se muestra un reto, ya que el Captcha se encarga de ejecutar con JavaScript el reto internamente.
- 99% pasivo: por lo general, no muestra un reto. Sin embargo, si se detecta un caso de alto de riesgo de presencia de *bot*, se mostrará un reto de los básicos para reforzar la seguridad.

Al haber obtenido únicamente licencias básicas, no pudimos probar los modos ofrecidos a licencias de empresa.

Para poner en marcha esta Captcha son necesarios un dominio web y una clave proporcionada por el propio hCaptcha. Se inserta la clave en el código HTML de la página web en un *div* de clase hCaptcha, más la llamada al *script* JavaScript de hCaptcha, el cual insertará el contenido de dicho *div*.

El flujo de llamadas es el siguiente:

1. El usuario carga la página web.
2. La página carga el código JavaScript de hCaptcha.
3. El usuario marca la casilla del formulario, lo que internamente se traduce como una petición a la API de hCaptcha para devolver un reto o un código de comprobación.
4. La API responde a la petición y el usuario ya tiene a disposición el reto para resolverlo.
5. La API envía a la página el código de comprobación.
6. Al completar el reto satisfactoriamente, la página le devuelve a su propio servidor el código.
7. El servidor contacta con el servidor de la API de hCaptcha (hCaptcha *Siteverify*) para comprobar la veracidad del código.

8. hCaptcha *Siteverify* confirma la veracidad del código y se lo comunica al servidor de la página.
9. El servidor permite al usuario proceder con el formulario.

Dada su implementación dirigida al desarrollador, es posible la personalización del comportamiento de hCaptcha. Empezando por los parámetros de inserción del *script* de la API en el código HTML. Es posible modificar el idioma, renderizado automático o no o definir qué función ejecutar al acabar la carga, entre otros.

hCaptcha también pone a disposición una variedad de atributos HTML para modificar el comportamiento y la apariencia del Captcha.

Con JavaScript, es posible crear un objeto hCaptcha, el cual ofrece una serie de funciones, las cuales ayudan al control del comportamiento; útil en el caso del modo invisible.

En cuanto a funcionalidad es igual a reCAPTCHA. Se diferencia en el estilo ligeramente diferente y en la posibilidad de ver dos fotos más del objeto a encontrar. La dificultad añadida se encuentra en que a los objetos o animales se les añade otros matices, los cuales hay que tener en cuenta también (ej. caballo con las patas blancas, león con los ojos cerrados).

3.4 Tecnologías utilizadas en el proyecto

Actualmente contamos con multitud de tecnologías válidas para desarrollar un servidor que procese las peticiones de los retos. De las posibles opciones estás son las tecnologías y herramientas que hemos empleado en el desarrollo del proyecto:

- Tecnologías básicas:
  - HTML.
  - CSS.
  - JSON.
  - HTTP.
  - DOM.
- Lenguajes de programación:
  - Python.
  - JavaScript.
- Tecnologías del *backend*
  - Django.
  - SQLite.
- Tecnologías del *frontend*
  - Django Widget Tweaks.
  - Bootstrap.

- Herramientas para la gestión del proyecto
    - Git.
    - Visual Studio Code.

3.4.1 Tecnologías básicas

HTML

El Lenguaje de marcado de hipertextos o HTML [5] por sus siglas en inglés es la herramienta básica para definir el contenido de una página web.

Este estándar que está controlado y desarrollado por el *World Wide Web Consortium* (W3C) define cómo debe estar estructurado el código de una página web.

El contenido de un documento HTML está formado por varios tipos de elementos que determinan cómo se muestra el contenido del documento. Estos elementos están marcados por etiquetas, una al comienzo para indicar el inicio del elemento y otra al final, con el carácter "/" para marcar el cierre, de la siguiente manera: <nombre del elemento HTML>Contenido del elemento</nombre del elemento HTML>.

Generalmente un documento HTML está estructurado de la siguiente manera: el inicio y el final del documento se indican con la etiqueta *<html>* y en su interior la etiqueta *<head>* marca la cabecera del documento donde se puede incluir información auxiliar, como el título del documento o enlaces a otros recursos y la etiqueta *<body>* donde se define el contenido visible del documento, como se puede ver en la figura 3.1.

Figura 3.1 – Un documento HTML siempre estará estructurado entre cabecera y contenido

Dentro de la etiqueta inicial de un elemento se pueden incluir diferentes atributos para modificar la representación del contenido de este. Estos atributos están formados por una clave a la que se le asigna un valor con el carácter "=", de la siguiente forma: clave = "valor".

Uno de estos atributos es *style.* Con él podemos utilizar CSS para modificar como se muestra el contenido del elemento del HTML. *Style* también puede utilizarse como etiqueta dentro de la cabecera del documento.

En nuestro proyecto, utilizamos la versión más reciente, HTML5, para el desarrollo de las páginas web de los usuarios y el componente que muestra los retos.

CSS

CSS [6], en ingles de Hojas de estilo en cascada es otro estándar controlado por el W3C. Esta herramienta se utiliza junto a HTML para definir los estilos con los que se muestra el contenido de la página web.

Como hemos dicho en el apartado anterior, en un documento HTML se puede incluir CSS mediante el atributo o la etiqueta *style*, pero esto hace que la definición del documento, especialmente dentro de una etiqueta sea muy recargada. La recomendación principal es definir el estilo de los elementos de la página web en un fichero CSS separado del documento HTML y enlazarlo a este en la cabecera.

Dentro del documento CSS se pueden especificar los estilos o bien para todos los elementos de un mismo tipo, o bien para elementos concretos que deberán tener un atributo u otro elemento que pueda identificarlos. La especificación del estilo sigue esta estructura, primero se identifica el elemento y entre los caracteres "{" y "}", se incluyen las propiedades a modificar, como se muestra en la figura 3.2.

```
.window {
  width: 100%;
  height: 100%;
}
```

Figura 3.2 - Amplía al 100% el ancho y el alto del elemento window

Para el proyecto, hemos empleado esta herramienta en su última versión CSS3 para darle un aspecto más agradable a las páginas de los usuarios y al componente de los retos.

JSON

La Notación de objeto de JavaScript o JSON [7] es un formato de texto para representar datos de forma sencilla. Este formato viene de la notación de objetos de JavaScript, pero debido a su auge frente a otras herramientas como XML ha pasado a ser considerado como un formato independiente del lenguaje del que procede.

Un objeto JSON se delimita entre los caracteres "{" y "}". Dentro del objeto, la información de cada elemento que lo forma se representa mediante a un nombre que lo identifica, el carácter ":" y el contenido que tiene asignado, desde tipos de datos simples como números enteros o cadenas de caracteres, hasta información más compleja como listas u otros objetos JSON. Si un objeto contiene varios elementos se utiliza una coma para separarlos, como se puede ver en la figura 3.3.

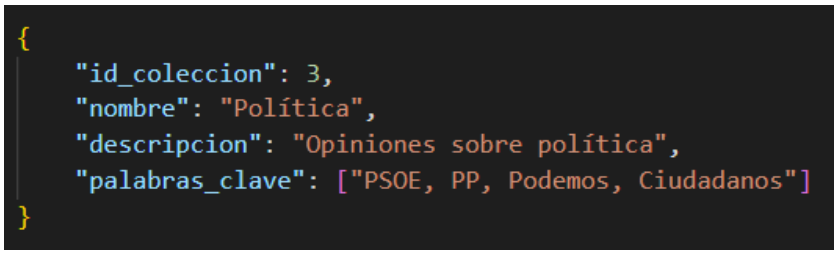

Figura 3.3 - Objeto con la información de una de las colecciones de textos

En el proyecto, utilizamos el formato JSON para definir los mensajes que intercambiaran el componente del reto y las distintas vistas de usuario con el servidor.

HTTP

El Protocolo de transferencia de hipertexto o HTTP [8] por sus siglas en inglés es la herramienta de comunicación que emplea el servidor para enviar el contenido de las páginas web y el componente del reto. HTTP es un protocolo sin estado, es decir, no almacena información sobre conexiones previas que funciona en la capa de aplicación, sobre TCP/IP.

Este protocolo define como estarán formados los mensajes de petición y respuesta entre servidor y cliente. HTTP cuenta con múltiples tipos de mensaje, estos son algunos de ellos:

- HTTP CONNECT: El método CONNECT sirve para dar comienzo a la conexión bidireccional con el punto de destino del mensaje. Un uso fundamental para este método es comprobar si un proxy nos permite acceder a un recurso bajo ciertas condiciones.
- HTTP DELETE: Este es un método muy sencillo cuya funcionalidad es borrar el recurso indicado.
- HTTP OPTIONS: Este tipo de mensaje se utiliza para preguntar por los métodos HTTP que admite la dirección del recurso solicitado.
- HTTP GET: Este método solicita al receptor una representación del recurso que se especifica en la URL del mensaje. La solicitud puede incluir parámetros que se añadirán a la URL.
- HTTP HEAD: Este tipo de mensaje se utiliza para enviar una petición similar a HTTP GET, pero en este caso, la contestación a este mensaje solo incluirá la cabecera que tendría la respuesta correspondiente a esa solicitud GET, sin el contenido del recurso solicitado.
- HTTP POST: Este es el método principal para enviar datos al servidor. En vez de emplear la URL como el método GET, en los mensajes HTTP POST se indica tipo del contenido que tiene la solicitud en la cabecera *Content-Type* y se incluye en el cuerpo de la petición.

- HTTP PUT: Este método se utiliza para crear un nuevo recurso o reemplazar otro existente en la dirección indicada en el mensaje y con el contenido incluido en este.

En nuestro proyecto emplearemos la versión del protocolo HTTP/1.1. Principalmente, los tipos de mensaje que más utilizaremos son GET y POST.

DOM

Developer.mozilla.org define el DOM como "una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a *script*s o lenguajes de programación." [9]

Igual que HTML prácticamente carece de utilidad sin JavaScript, JavaScript carece de utilidad sin DOM, pues sin esta representación de los objetos del documento HTML en JavaScript, no sería posible modificar las componentes de la página web para que cumplan con las funcionalidades asignadas para ellas. Con esta interfaz es posible crear elementos, recuperarlos y modificar su contenido. Se hace uso del atributo "id" de las etiquetas HTML, en el caso de referirse a un único elemento del documento. En otros casos se recurre al atributo "*class*" o a la búsqueda según tipo de etiqueta (*div*, *p*, *label*, etc.), lo que devolvería un array de elementos. En nuestro código recurrimos únicamente a llamadas por identificador, no sin reconocer la utilidad de los otros métodos mencionados.

3.4.2 Lenguajes de programación

Python

Python [10] es un lenguaje de programación de alto nivel cuya filosofía principal de diseño es la legibilidad del código. El uso de este lenguaje está ya bastante extendido llegando a ser utilizado en aplicaciones web tan conocidas como Dropbox, Netflix o Reddit. Este lenguaje, al igual que Java, es un lenguaje interpretado cuyo paradigma principal era la orientación a objetos, pero al ser un lenguaje multiparadigma, soporta en menor medida la programación funcional.

En el proyecto Python ha sido el lenguaje escogido para programar el funcionamiento del servidor debido a que es un lenguaje que ofrece una gran versatilidad de uso manteniendo su sencillez.

JavaScript

Lenguaje utilizado en el desarrollo del *frontend*, cuyo desarrollo se explicará en próximas secciones.

JavaScript [11] está definido como un lenguaje de alto nivel, interpretado y orientado a objetos, aunque también permite técnicas de programación funcionales e imperativas. Es un lenguaje de *scripts* basado en prototipos, que se ejecuta en el lado del cliente y se puede utilizar para programar cómo reacciona una página web ante unos eventos determinados. Aunque su principal uso sea en aplicaciones web, también puede tener usos en otros entornos a parte del navegador.

En este proyecto ha sido clave para la unión del *backend* y el *frontend*; así como la comunicación entre el widget y la página en la que esté incrustado. Su uso más útil ha sido el de manipulación de JSON. Gracias a DOM, es posible manipular los objetos HTML, para dotarlos de contenido, y/o ocultarlos cuando es necesario. Este lenguaje es el nexo más significativo entre los componentes, pues ayuda a procesar la información enviada por el servidor y la devuelta a este. Cuando el servidor envía datos, éstos se pierden si no existe ningún mecanismo que se encargue de su tratamiento. Al igual que una página HTML no puede ser dinámica sin recibir datos durante su uso. El lenguaje HTML, por sí solo, permite con gran limitación insertar contenido en una página. Sin embargo, esto no es suficiente, ya que es incapaz de dotarla de ninguna funcionalidad. Y por ende, de ninguna utilidad.

3.4.3 Tecnologías para el *Backend*

Django

Django [12] es un entorno escrito en Python para desarrollo web. Inicialmente fue diseñado para aplicaciones web de noticias de la World Company de Lawrence, Kansas, pero en julio de 2005 fue distribuida al público bajo una licencia de software libre.

Este entorno permite la implementación rápida de aplicaciones web fácilmente mantenibles y seguras. Django se encarga de la mayoría de las complicaciones que surgen a la hora de desarrollar una aplicación web, lo que permite centrarse en la implementación del comportamiento de la aplicación. Además de eso ofrece una biblioteca para simplificar la implementación de las vistas del cliente. Dentro del entorno a este formato de vistas se les llama plantillas.

Hemos elegido este entorno para el desarrollo del backend ya que el objetivo principal de Django es simplificar el desarrollo de sitios web complejos, haciendo hincapié en que los componentes que van a componer la aplicación web sean fáciles de ampliar e interconectar y puedan reutilizarse, lo que se adecua muy bien a nuestras necesidades porque agiliza el proceso para añadir nuevos tipos de datos que se quieran etiquetar.

SQLite

SQLite [13] es una biblioteca desarrollada en C que proporciona un ligero y sencillo gestor de bases de datos relacional.

En vez de ejecutarse como un proceso independiente con el cual el programa principal del servidor debe comunicarse, SQLite se une al proceso del programa principal pasando a formar parte de este. Debido a esto, el contenido de una base de datos implementada con este gestor, se puede almacenar en un solo fichero.

Este es uno de los gestores que ofrece Django para manejar la base de datos de tu servidor, lo hemos escogido ya que facilita la implementación del prototipo de una base de datos para un proyecto como este. Gracias a Django, en caso de que el proyecto crezca más y sea necesario otro gestor capaz de manejar bases de datos más grandes de manera más eficiente, se puede hacer la migración a otro sistema sin necesidad de hacer grandes cambios en el código del servidor.

3.4.4 Tecnologías del *Frontend*

Django Widget Tweaks

Django proporciona plantillas de formularios. Sin embargo, lo único que se muestra son los campos y un botón de enviar, en un fondo blanco. Un diseño que no da más que la impresión de encontrarse ante una página anticuada (figura 3.4).

Figura 3.4 - Formulario por defecto, sin personalización

Esto no es deseable a la hora de vender un producto, ya que no invita al usuario a elegirlo ni a volver a usarlo.

Para evitar este problema, se ha introducido el paquete Django Widget Tweaks [14], el cual hará posible la personalización de los formularios. Con muchos o pocos cambios en el código, dependiendo de la complejidad que requiera el nuevo diseño.

Cuando se invoca a una función para renderizar un formulario, Django lo que hace es devolverle a la plantilla HTML un único objeto, el cual incluye todos los campos que se hayan configurado para el formulario. No poder separar los campos, no imposibilita la personalización, ya que se podría modificar su estilo o realizar otros cambios como cambiar el color de fondo.

Esencialmente lo que hace Django Widget Tweaks es dar acceso a los distintos campos del formulario, para poder insertarlos en el código según sea conveniente. Devuelve un elemento HTML, cuyos atributos es posible modificar (*class*, *placeholder*, etc.). Dando resultados como el de la figura 3.5.

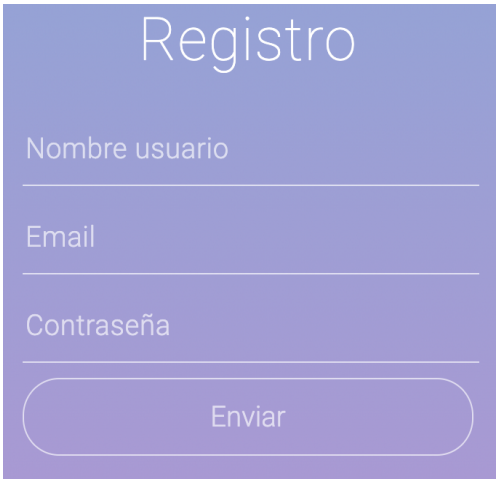

Figura 3.5 - Diseño más agradable para los usuarios

Bootstrap

Debido a la necesidad de un diseño agradable, mencionada anteriormente, se ha pensado en cómo alcanzar este objetivo. Al tratarse de una página HTML, lo primero en lo que se piensa es en CSS, lenguaje usado para la definición de estilos de HTML, así como de estructuración de los elementos. Con la práctica, crear páginas usables y con una apariencia agradable con CSS, es un logro conseguible. Si bien es uno que escapa a los intereses de este equipo. Lo que ha llevado a buscar una alternativa que facilite este proceso. De entre todas las opciones, se eligió el *entorno* Bootstrap [15].

La inclusión de este *entorno* en el proyecto se ha llevado a cabo vía CDN, ya que requiere menos pasos. Únicamente hay que insertar dos líneas de código en el fichero HTML a manipular. En el momento en que se empezó a usar dicha herramienta, la versión más reciente que había disponible era la 5, conocida por la eliminación de la posibilidad del uso de jQuery en su manejo. Esta fue evidentemente la versión elegida para el proyecto.

Cuando se dibuja el JSON recibido con los textos y las opciones, las opciones son asignadas cada una a un botón, el cual forma parte de un grupo de botones o en el lenguaje de Bootstrap btn-group, nombre del id del *div* en el que deben insertarse estos botones. Estos grupos de botones lo que facilitan es el posicionamiento de unos respecto a otros (horizontal, vertical), así como la separación entre ellos. También se puede establecer su estilo como grupo en lugar de tener que asignarle uno a cada uno por separado, y otras características como su aspecto al posicionar el puntero sobre el botón.

Bootstrap también fue utilizado para el diseño de la interfaz de usuario. Esta vez sacando provecho a las tablas y a su conocida característica de ordenación de elementos por cuadrículas.

3.4.5 Herramientas para la gestión del proyecto

Git

Git [16] es un software para control de versiones de código abierto, publicado en 2005 por Linus Torvalds. Es una herramienta muy potente que crea un repositorio que hace un seguimiento de todos los cambios en los archivos del proyecto, guarda un registro de las modificaciones que se han confirmado, además permite crear distintas versiones auxiliares o ramas del proyecto para que cada miembro del equipo pueda trabajar en sus cambios de forma individual y de forma sencilla realizar la unión de estos.

Git proporciona una serie de instrucciones u órdenes para manejar el contenido y las versiones del proyecto, algunas de estas instrucciones son:

- Git init: Esta orden sirve para crear un subdirectorio en el proyecto llamado *.git* con todos los ficheros necesarios para la gestión del repositorio.
- Git fetch: Esta instrucción se usa para descargar todos los cambios que se han aplicado al repositorio.
- Git merge [nombre de la rama]: Con esta instrucción puedes dirigir los cambios a la rama indicada en el parámetro nombre de la rama.
- Git pull: Esta orden combina los efectos de las instrucciones *fetch* y *merge* en un mismo comando.
- Git commit -m [mensaje]: Esta instrucción confirma los cambios realizados en el proyecto. Con la opción -m podemos añadir un mensaje para describir brevemente los cambios realizados.
- Git push [nombre rama]: Este comando sube al repositorio la rama indicada en el parámetro nombre rama.

Al haber trabajado principalmente de forma remota, era necesario el uso de un repositorio Git en línea en el que poder gestionar el proyecto. Junto a esto, hemos elegido GitHub como plataforma para alojar el proyecto ya que su licencia para estudiantes tiene muchas ventajas.

Visual Studio Code

Visual Studio Code es un editor de código fuente gratuito desarrollado por Microsoft que ofrece muchas herramientas para el programador, soporte para depurar código,

integración con Git para facilitar el control de versiones y múltiples extensiones para el lenguaje con el que se está codificando.

Hemos elegido este editor de código porque estas herramientas, por ejemplo, la extensión para poder ver el contenido del archivo que forma la base de datos [17], nos han ayudado mucho a simplificar el flujo de trabajo.

4 Arquitectura del proyecto

Para este proyecto nos centraremos en diseñar una versión inicial de nuestro servicio de CAPTCHA cuyo reto muestre una serie de textos y un conjunto de opciones para cada texto que dependerán del contenido de este.

El proceso de etiquetado consiste fundamentalmente en asignar una de estas opciones como etiqueta de un texto para que una inteligencia artificial pueda aprender a reconocer el valor, el sentido, o las intenciones de un mensaje de texto en lenguaje natural.

4.1 Visión global del diseño del proyecto

La arquitectura de nuestro proyecto está divida en dos bloques, el cliente, que está formado por el componente de los retos y las vistas para los usuarios y el servidor que envía los retos al componente y almacena toda la información de las colecciones de datos etiquetados y usuarios.

El esquema de la figura 4.1 representa esta estructura

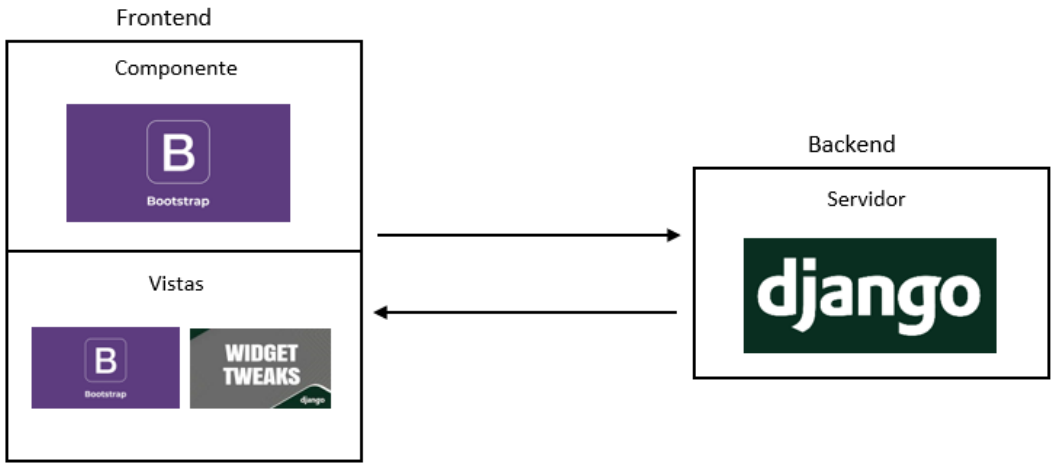

Figura 4.1 - Representación de la arquitectura del proyecto

Ambos bloques se comunican mediante mensajes en formato JSON para servir y comprobar los retos que se presentan al usuario. El componente es un pequeño *script* desarrollado en JavaScript que se puede añadir a una aplicación web como campo de un formulario, inicialmente se muestra un botón para dar comienzo al reto al hacer clic, el componente solicita al servidor un conjunto de textos con los parámetros indicados en la configuración del *script*, estos parámetros son:

- Palabras clave: El usuario podrá proporcionar un conjunto de palabras clave para que el componente solo muestre textos de las colecciones relacionadas con esas palabras.

- Palabras a evitar: Al igual que con las palabras clave, también podrá incluir un conjunto de palabras en caso de que quiera evitar que se muestren textos de colecciones relacionadas con ellas.
- Número de textos: La cantidad de textos que muestre el componente puede personalizarse, si no se indica nada, se mostrarán tres textos.
- Información del cliente: Para evitar solicitudes espurias el cliente del nuestro servicio tendrá que proporcionar su nombre de usuario y su clave de API.

El servidor responde a esta solicitud con un mensaje que incluye el texto para los retos con el formato mostrado en la figura 4.2.

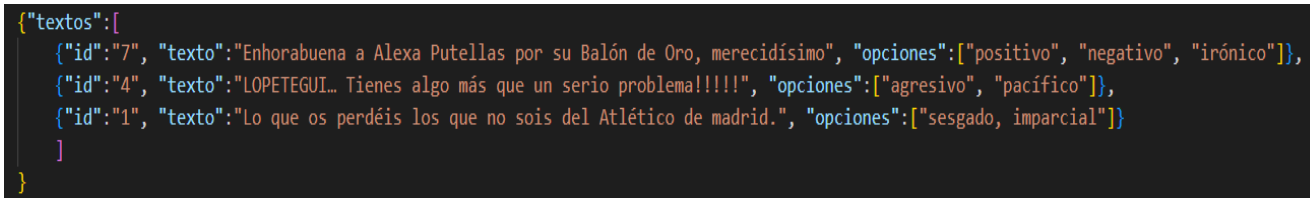

Figura 4.2 - Formato del mensaje con los textos para el reto

Con esta información el componente muestra los retos uno por uno y va registrando las opciones que ha elegido el usuario, al finalizar, envía estas respuestas al servidor con un mensaje en el formato mostrado en la figura 4.3.

Figura 4.3 - Formato del mensaje con las respuestas

Entonces el servidor comprueba que las respuestas proporcionadas para textos ya etiquetados coincidan con la información en la base de datos, si son iguales, entonces se considera un usuario legítimo y añade la información necesaria a las colecciones de datos, en caso contrario, los datos etiquetados no se actualizan.

Después, el servidor envía el resultado de la comprobación con un mensaje con los siguientes formatos dependiendo de si es un usuario legitimo (figura 4.4).

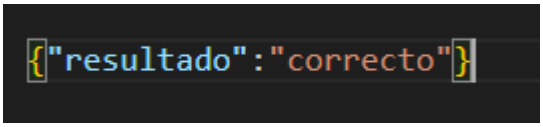

Figura 4.4 – Mensaje si se ha superado el reto

O puede ser un programa informático fingiendo ser un usuario legítimo (figura 4.5).

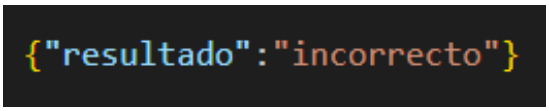

Figura 4.5 - Mensaje si se ha fallado el reto

Por último, hay dos vistas para los clientes del servicio de CAPTCHA y para los administradores que quieran añadir o extraer colecciones de la base de datos.

4.2 Vista de administrador

La vista de administrador consiste en una pantalla hecha a partir de la vista de administrador que proporciona Django.

En esta vista, el usuario que tenga este rol puede acceder al contenido de la base de datos, concretamente a las tablas que contienen toda la información relacionada con los textos, es decir, colecciones, textos y opciones (figura 4.6). Así como ver y modificar la información y permisos de los distintos usuarios, habiendo también la posibilidad de ordenarlos en grupos. Dentro de la vista de la tabla de colecciones, los administradores tienen a su disposición un botón para subir colecciones desde un archivo de texto (figura 4.7), lo que lleva a un simple menú de selección de archivo para enviar (figura 4.8).

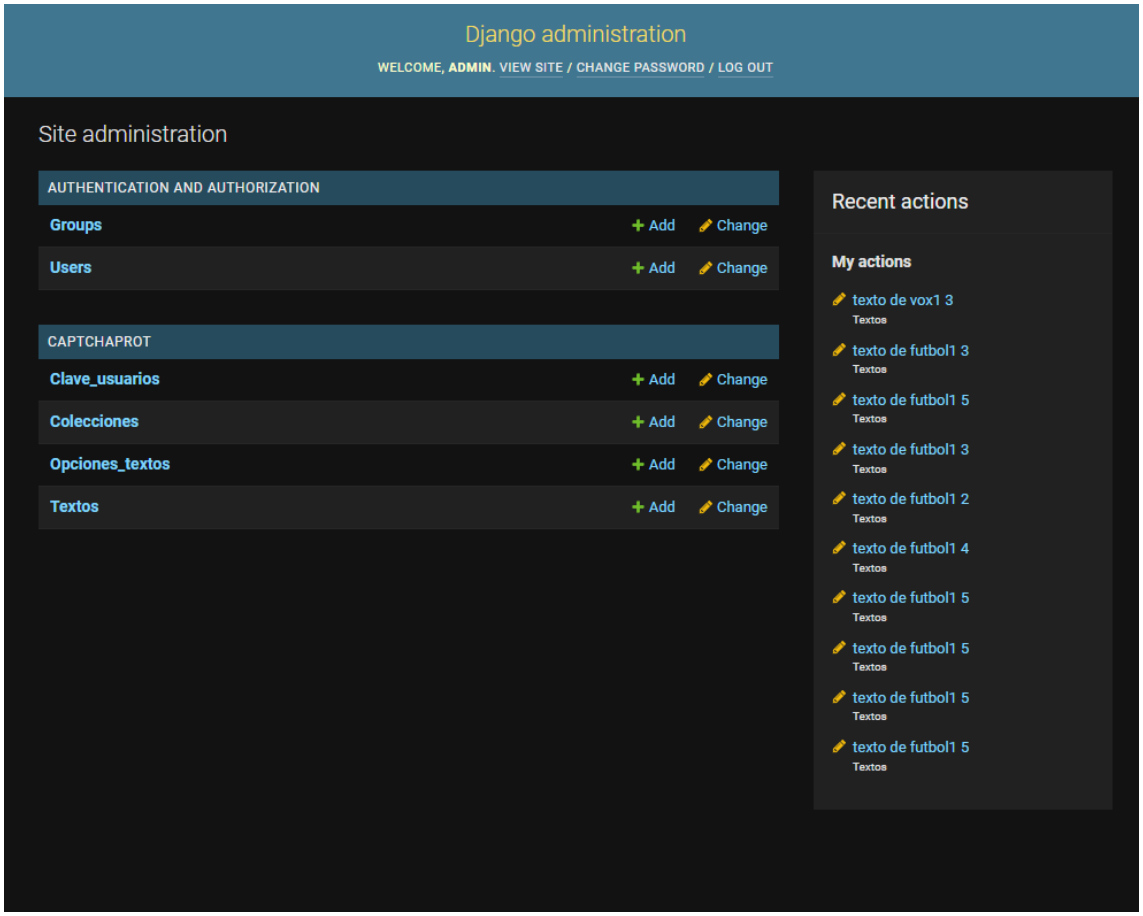

Figura 4.6 – Menú principal de administrador

Figura 4.7 – Se selecciona "Subir archivo CSV"

Figura 4.8 – Menú para subir archivos

4.3 Vista del cliente

Esta es una pantalla más sencilla, diseñada con el sistema de vistas de Django junto con Bootstrap para añadir estilos a la página. Esta vista (figura 4.9) muestra la siguiente información al usuario:

- Clave de usuario: Esta es la clave que un usuario del servicio de CAPTCHA necesita para solicitar un reto al servidor. Junto con esta información se encuentra un botón que permite al usuario renovar su clave, en caso de que se halla visto comprometida.
- Información sobre las colecciones: En esta parte de la pantalla se muestran los nombres y las palabras clave de las colecciones junto con dos botones, uno para ver el contenido de las colecciones (figura 4.10) y otro para descargarlas.
  La información sobre las colecciones se solicita al servidor que la envía en un mensaje JSON con el formato mostrado en la figura.

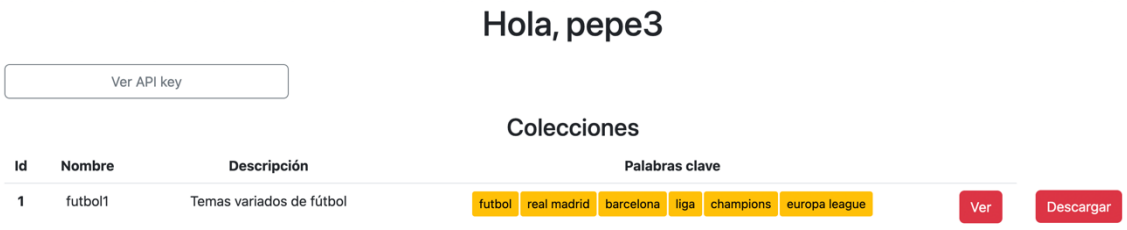

Figura 4.9 – Menú principal de cliente


Colección #1

Retos etiquetados

Texto	Elección	Fiabilidad
¡Qué partido tan alucinante el de anoche!	Alegría	45.45 %
Ese futbolista no se merecía el premio, debió ir a otro.	Decepción	85.50 %
El comportamiento de los expectadores fue inaceptable.	Enfado	91.43 %
El arbitraje del partido ha sido una vergüenza, deberían despedir al arbitro.	Indignación	90.05 %

Retos sin etiquetar

Texto	Fiabilidad
Nada mejor que una cerveza fría para ver el partido.	50.27 %
Todo el mundo se merece un contrato digno en este deporte.	34.44 %
Los sueldos de los futbolistas son desórbitados.	8.67 %
La táctica del equipo fue muy mala, no me extraña que perdieran.	51.00 %

Figura 4.10 – Vista de una colección

5 *Backend*

Como hemos mencionado anteriormente, para la implementación del servidor utilizamos el entorno Django, desarrollado en Python. Django nos permite desplegar rápidamente un servidor funcional y listo para personalizar.

5.1 Modelos (Base de datos)

Para almacenar la información, Django proporciona una capa de herramientas y métodos para simplificar la implementación de la base de datos, esta es la biblioteca de Modelos. Con ella definiremos los modelos como clases, con sus atributos correspondientes para representar la información y el entorno se encarga de crear las tablas, con los atributos formando las columnas en la base de datos. Dentro del modelo también podemos añadir los métodos que necesitemos para manipular las tablas de la base de datos.

Ya que los campos de una base de datos relacional tienen que estar tipados, también tenemos que asignar un tipo a los atributos de los modelos, para que el entorno pueda definir esos campos. La biblioteca de Django para los modelos proporciona múltiples tipos, estos son los que usamos principalmente:

- BigAutoField: Define un campo para un numero entero que se incrementa automáticamente al añadir una fila a la tabla. En concreto hacemos uso de este tipo para definir el campo de id.
- ForeignKey: Este tipo se usa para definir un campo que indica una relación varios-a-uno entre dos tablas.
- OneToOneField: Define en la base de datos un campo que, similar al campo ForeignKey, indica una relación uno-a-uno entre dos tablas.
- CharField: Este tipo equivale a un campo VARCHAR en la base de datos. En este atributo es necesario indicar el valor del atributo *max_length* para especificar el número máximo de caracteres.
- BigIntegerField: Define un campo en la base de datos para números enteros muy grandes.
- DecimalField: Equivale a un campo en la base de datos para números decimales. En este atributo es obligatorio especificar el número máximo de dígitos y el número máximo de decimales.

Los modelos que forman la base de datos de nuestro proyecto son los siguientes:

- Colección:
  Una colección es la forma en la que los textos están organizados en nuestro proyecto. Con este modelo definimos su tabla correspondiente en la base de datos. Estos son los atributos que tiene el modelo:

Atributo	Tipo
id	BigAutoField
nombre	CharField
descripción	CharField
palabras_clave	CharField

- Id: Atributo para definir el identificador de cada colección.
- Nombre: Atributo que define el campo para el nombre de la colección.
- Descripción: Atributo que define el campo para una pequeña descripción sobre los textos de la colección.
- Palabras_clave: Atributo que define un campo en la base de datos que almacenara una cadena de caracteres de palabras clave de los textos de la colección, separadas por comas.

- Texto:
  Para este proyecto nuestro servicio de CAPTCHA etiqueta textos. Este es el modelo que define la tabla que almacenará toda la información con respecto a los textos, excepto las opciones correspondientes a cada uno.
  Los atributos de este modelo son:

Atributo	Tipo
id	BigAutoField
texto	CharField
coleccion	ForeignKey
cuenta_respuestas	BigIntegerField
eleccion	CharField
umbral_eleccion	DecimalField
fiabilidad_eleccion	DecimalField

- Id: Atributo para definir el identificador de cada texto.
- Texto: Atributo que define el campo para el texto.
- Coleccion: Atributo que define el campo para relacionar cada texto con su colección correspondiente.
- Cuenta_respuestas: Atributo para definir un campo que almacena una cuenta de cuantas respuestas se han dado a un texto.
- Elección: Atributo que define un campo para almacenar la elección elegida de forma mayoritaria por los usuarios
- Umbral_eleccion: Este atributo representa un porcentaje que deberá superar una opción de un texto para ser considerada la elección de ese texto.

- Fiabilidad_eleccion: El porcentaje de respuestas de la opción considerada como elección del texto.

Para trabajar con la información almacenada en la tabla correspondiente, en este modelo definimos los siguientes métodos:

- Comprueba (Opción): Método *booleano* que comprueba si la opción elegida para el texto mostrado en el reto coincide con la elección escogida para el texto. Si son las mismas, este método devuelve *true*, o *false* en caso contrario.
- Actualiza_eleccion: Método que calcula la fiabilidad de cada elección y la compara con el umbral, en caso de la fiabilidad sea mayor o igual, actualiza la elección correspondiente al texto.

- Opciones_texto:
  Como cada texto puede tener un número variable de opciones necesitamos un modelo para definir una tabla que almacene las opciones y las relacione con su texto correspondiente.
  Estos son los atributos del modelo:

Atributo	**Tipo**
texto	ForeignKey
opcion	CharField
cuenta	BigIntegerField

  - Texto: Atributo que define el campo para relacionar cada opción con su texto correspondiente.
  - Opcion: Atributo para definir un campo que almacena la opción.
  - cuenta: Atributo para definir un campo que almacena una cuenta de cuantas respuestas se han dado a una opción en concreto.

  La biblioteca de modelos ofrece un sistema de reglas en caso de que se quieran añadir restricciones a la información que se puede almacenar en una tabla. Para este modelo tendremos que definir la regla *unique_together* entre los atributos texto y opción para restringir la posibilidad de que pueda haber opciones repetidas para un texto.

  En este modelo también necesitamos un método auxiliar para trabajar con la información de la tabla. Este método es:

  - Actualiza_cuenta: Método que añade uno a la cuenta de la opción en caso de que haya sido elegida en un reto.

- Clave_usuario:

Este modelo definirá una tabla complementaria a la tabla de usuarios que Django define al inicializar el servidor. Esta tabla almacena las claves de los clientes que utilizan el servicio de CAPTCHA y las relaciona con el usuario correspondiente. Los atributos del modelo son:

Atributo	**Tipo**
usuario	OneToOneField
clave	CharField

- Usuario: Atributo que define el campo para relacionar cada clave con su usuario correspondiente.
- Clave: Atributo para definir un campo que almacena la clave. Esta clave está formada por una cadena de 32 caracteres alfanuméricos aleatorios.

Dentro de este modelo definimos los siguientes métodos para trabajar con la información contenida en la tabla:

- Renueva_clave: Método que sustituye la clave almacenada en la tabla por otra nueva, en caso de que el cliente así lo decida.
- Comprueba_clave: Método *booleano* que comprueba que la clave que proporciona el cliente del servicio de Captcha al solicitar un reto es igual a la clave almacenada en la tabla. Si coinciden la respuesta del método será true, o false en caso contrario.

5.2 Interfaz para las peticiones del componente del reto

En esta sección explicaremos la implementación de la interfaz que se encarga de procesar las peticiones que le envía el componente. Esta interfaz está definida el archivo *api.py.*

Como hemos dicho anteriormente, el proceso del reto se compone de tres partes junto con la lectura de las opciones que realiza el componente.

Primero se hace un filtro, si el usuario ha definido un conjunto con las palabras clave y las palabras a evitar, de las colecciones válidas para buscar textos para el reto.

Después se selecciona un número de textos para enviar al componente del reto, algunos de los textos tienen ya una etiqueta definida y otros no. El número total de textos es mínimo de tres por defecto, pero el cliente puede indicar otro número en su petición.

A continuación, el servidor recibe las opciones elegidas por el usuario. Con los textos que ya estaban etiquetados, comprueba si se trata de un usuario real o no. Para todos los textos mostrados se actualizará su información en caso de un usuario legítimo.

Los métodos implementados en la interfaz para realizar este proceso son:

- Filtrar_colecciones (palabras clave, palabras a evitar, lista de colecciones): Este método *booleano* realiza tres tareas, primero filtra las colecciones posibles con las palabras clave. A continuación de este conjunto elimina las colecciones que contengan alguna palabra a evitar. Finalmente, comprueba que la lista definitiva de colecciones no está vacía devolviendo *true,* o *false* si lo está.
- Selecciona_textos (retos etiquetados, retos sin etiquetar, número de textos): Este método recibe los textos de la lista definitiva de colecciones, separados en dos listas según tengan etiqueta o no y la cantidad de textos que el cliente quiere mostrar en el reto. En primer lugar, hace la *div*isión entera de este número entre dos, una de las mitades, más uno, es la cantidad de textos con etiqueta y el resto son los textos que no están etiquetados. A continuación, selecciona la cantidad correspondiente de textos con y sin etiquetar. Por último, los devuelve agrupados en una lista.
- Reto: Este es el método que envía los textos al componente. Primero, llama a filtrar_colecciones para generar una lista de colecciones validas, si la lista no está vacía separa los textos en dos conjuntos dependiendo de si tienen etiqueta o no. Con estos conjuntos llama a selecciona_retos que devolverá la lista de textos para enviar al componente. Con este conjunto, creará el mensaje en el formato JSON correspondiente. Si la lista definitiva de colecciones está vacía, creará el mensaje en el formato JSON correspondiente. El componente hace la solicitud POST a este método a través de la URL <<*host*>>/tagcaptcha/reto
- Comprobación: Este método recibe el conjunto de respuestas de un usuario a los textos mostrados en el componente. Después, comprueba que las opciones elegidas para los textos del conjunto que tienen etiqueta son iguales con las elecciones almacenada en la base de datos, si coinciden se considera un usuario legítimo, se actualizan las opciones de todos los textos mostrados y las etiquetas haciendo el cálculo de los umbrales de cada opción. Si no coinciden el usuario ha fallado el reto y no se actualiza nada. Finalmente, envía el resultado de la comprobación. El componente hace una solicitud POST a este método con la URL <<*host*>>/tagcaptcha/comprobación

5.3 Interfaz para las peticiones de las vistas del cliente

En este apartado, explicaremos el conjunto de métodos que se encarga de responder a las peticiones que se hacen desde las vistas del cliente.

Dependiendo del tipo de usuario las vistas tienen funcionalidades distintas.

Actualmente, la funcionalidad particular que tienen disponible los clientes del servicio de CAPTCHA es renovar su clave de usuario.

El método que se encarga de realizar la renovación son los siguientes:

- Generar_clave: Este método simplemente crea una cadena aleatoria de treinta y dos caracteres alfanuméricos.
- Renovar_clave: Recibe la petición del botón para generar una clave nueva. Después, crea la nueva clave llamando al método generar_clave y busca en la tabla correspondiente de la base de datos al usuario y para sustituir la clave antigua por la recién creada.

Los administradores del servicio de CAPTCHA podrán añadir y descargar colecciones de textos de la base de datos.

Para la carga de colecciones implementamos los siguientes métodos:

- Leer_documento_cargado (datos de la petición): Este método se encarga de procesar la información recibida en la petición y escribirla en un archivo de texto temporal, como el que se muestra en la figura 5.1.
  El formato de este archivo es similar a un archivo de valores separados por comas (*CSV*) [18], es decir, los elementos de cada fila estarán separados por comas. Sin embargo, no todas las filas tendrán el mismo número de columnas ya que la primera línea tendrá solo dos elementos que son el nombre y la descripción de la colección y la segunda línea una lista de palabras clave. A partir de la tercera fila comienzan los textos de la colección, el primer elemento de la fila es el contenido del texto, a continuación, la lista de opciones entre corchetes y por último el umbral de aceptación para la opción que salga elegida como etiqueta.
  Además, como los textos pueden contener comas y también las utilizamos como separadores en la lista de palabras clave y las listas de opciones empleamos el punto y coma como delimitador.
- Cargar_coleccion_bbdd: La funcionalidad de este método es leer el archivo temporal con la información de la colección y la almacena en la tabla correspondiente de la base de datos.
- Subir_retos: Este método recibe la petición con la información de la colección que se desea almacenar en la base de datos. El contenido de la solicitud se procesa con la función leer_documento_cargado creando un archivo temporal que procesará cargar_coleccion_bbdd. Una vez ha terminado la subida a la base de datos, elimina el archivo temporal. La petición se realiza mediante una solicitud POST a la URL <<*host*>>/tagcaptcha/subir_retos.
  El formato que tendrá el archivo para subir la colección es el siguiente:

```
Fútbol;Opiniones de fútbol sobre jugadores y equipos.
futbol,Real Madrid,Barcelona,liga,Balón de Oro,europa league
Lo que os perdéis los que no sois del Atlético de madrid;[sesgado, imparcial];75
LOPETEGUI… Tienes algo más que un serio problema!!!!!;[agresivo, pacífico];70
Enhorabuena a Alexa Putellas por su Balón de Oro, merecidísimo;[positivo, negativo, irónico];90
El @CelticFC ha salido como una moto.;[positivo, negativo];85
```

Figura 5.1 - Formato del archivo para cargar colecciones

Para descargar de la base de datos la información de una colección solo es necesario el siguiente método:

- Descargar_csv: Está función se encarga de buscar en la base de datos la información con los textos de una colección que recibe como parámetro en una petición GET a la URL <<*host*>>/tagcaptcha/subir_retos?id_coleccion=<<id>> y descargarlos en un archivo como el que se muestra en la figura 5.2.
  Al igual que en el archivo de carga el formato de este archivo también es similar al formato *CSV.* La primera fila tiene exactamente la misma estructura, nombre y descripción de la colección. La siguiente fila, que será la misma para todos los archivos que se descarguen, contiene los nombres de los elementos que estructuran las entradas de los textos para que el documento se muestre de forma organizada en una tabla. Y a partir de la tercera fila comenzarán los registros de los textos.

```
Fútbol;Opiniones de fútbol sobre jugadores y equipos.
Texto;Opción elegida;Aceptacion
El @CelticFC ha salido como una moto.;positivo;86.17
LOPETEGUI… Tienes algo más que un serio problema!!!!!;agresivo;81.33
Enhorabuena a Alexa Putellas por su Balón de Oro, merecidísimo;positivo;93.05
Lo que os perdéis los que no sois del Atlético de madrid;sesgado;79.41
```

Figura 5.2 - Formato del archivo descargado con la información de la colección

6 *Frontend*

En este apartado se explicará cómo el comportamiento del CAPTCHA por parte de frontend, la vista para los usuarios y su implementación.

6.1 Inicio del reto

El reto se carga por defecto al cargar la página en la que esté incrustado el widget, ya que el *script* al cargarse la página hace la petición al servidor para que le devuelva el JSON con el reto. Cuando se comprueba que la petición ha tenido éxito (código 200), se invoca una función (crearBoton), la cual crea el botón y lo incluye en el HTML con el uso DOM. A este botón se le añade un *eventListener* de tipo clic, el cual se encarga de llamar a la función que muestra los textos del reto. Esta función (mostrarMensaje) sigue los siguientes pasos:

1. Oculta los elementos de pantalla principal.
2. Cambia el tamaño del elemento de salida y le asigna como contenido HTML el texto de la primera pregunta del reto.
3. Entra en un bucle para imprimir los textos, el cual hace lo siguiente:
    a. Crea un *div* para el grupo de botones si no está ya creado.
    b. Crea un nuevo botón y un nodo de texto al que se le asigna como valor el texto de la opción. Se añade el nodo al botón. Su atributo *value* se pone al índice de la opción, para después añadir dicho índice al JSON con las respuestas, si se pulsa dicho botón (se elige esa opción).
    c. Al botón se le añade un *eventListener* de tipo clic, el cual:
        - Si todavía quedan textos en el reto, aumenta el contador y vuelve e invoca a mostrarMensaje recursivamente.
        - Si no, se ocultan todos los elementos y se invoca la función que envía el JSON con las opciones elegidas al servidor.
        - En ambos casos, se produce una llamada a la función que se encarga de incluir las opciones elegidas en el JSON.
    d. Se añade el botón al grupo y el grupo al *div*.

Para darle a entender al usuario que se trata de un CAPTCHA, se añade encima del botón el título “¿Eres un robot?” (figura 6.1).

Figura 6.1 - Lo primero que el usuario verá al cargarse el formulario

6.2 Almacenamiento de respuestas

Al hacer clic en el botón "No", al usuario se presenta la primera pregunta del reto. Lo que se puede observar es el texto de la pregunta y debajo de él una serie de botones con las posibles respuestas (figura 6.2).

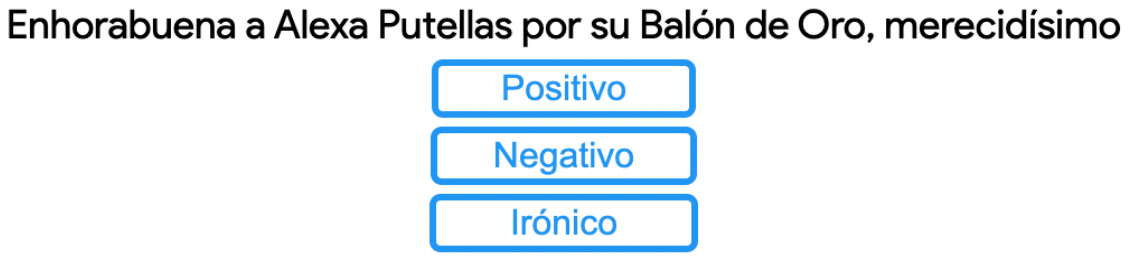

Figura 6.2 - El CAPTCHA muestra una pregunta

Con el objetivo de ayudar al usuario a tener claro, qué respuesta está siendo seleccionada, con CSS se configura la propiedad hover del elemento div que contiene estos botones, para que al posicionar el cursor sobre una de las respuestas, el botón cambie de color, resaltando así sobre los demás (figura 6.3).

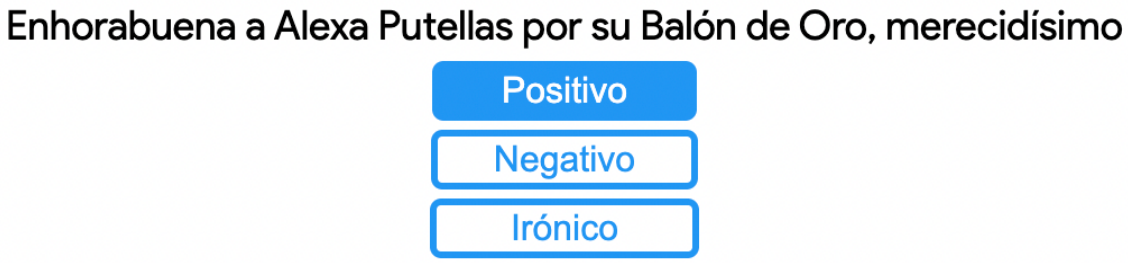

Figura 6.3 - Respuesta resaltada

Pulsar uno de estos botones llevará a la siguiente y el proceso continuará hasta que se hayan contestado todas.

Inicialmente la variable que guardará las respuestas del reto, tiene como valor un objeto, cuyo único campo es un array vacío de nombre "respuestas". Los campos que se almacenan son los siguientes:

1. Identificador de la pregunta (id).

2. Opción elegida (respuesta).

La función encargada de incluir la opción seleccionada en el JSON, sigue únicamente tres pasos:

1. Crea una cadena de caracteres con el identificador de la respuesta y su índice en el array de opciones.
2. Se convierte esta cadena de caracteres a un objeto JSON.
3. Se añade este nuevo elemento al final del JSON de respuestas.

6.3 Envío de respuestas

Como se ha mencionado, cuando se comprueba que ya no quedan más textos que mostrar, se llama a una función que sigue los siguientes pasos:

1. Se inicializa la petición POST.
2. Se establece como la función a ejecutar cuando la petición devuelva una respuesta (mostrarExito).
3. Se establece el tipo de contenido a enviar a tipo JSON.
4. Se convierte el JSON de las respuestas a una cadena de caracteres.
5. Se hace la HTTP Request con el JSON como cuerpo de la petición.

6.4 Recepción del veredicto

En resumen, mostrarExito consulta el veredicto del servidor y muestra un resultado acordemente. Estos son los pasos que sigue:

1. Convierte la cadena de caracteres recibida a JSON.
2. Comprueba el valor del campo resultado:
    a. Si es igual a “correcto” (se trata de un humano), se envía un mensaje a la página contenedora del widget para que vuelva a habilitar el botón de enviar el formulario. Cerrándose el CAPTCHA al final (figura 6.4).

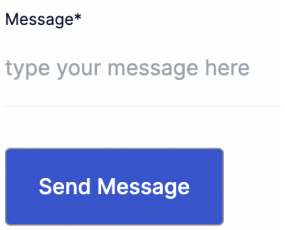

Figura 6.4 - Botón habilitado si se ha superado el reto

    b. En otro caso (se trata de un *bot*), se muestra el mensaje “Inténtalo de nuevo”, junto a un botón de reintentar (figura 6.5), el cual invocará a

mostrarMensaje para comenzar de nuevo por la primera pregunta de otro reto.

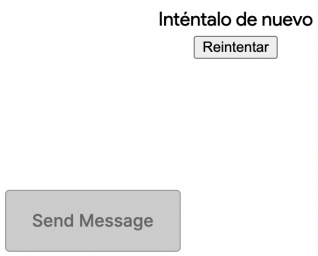

Figura 6.5 - Botón deshabilitado si se ha fallado el reto

6.5 Vista de usuario

Como se ha explicado anteriormente, Django ofrece plantillas, las cuales requieren de una posterior personalización para cumplir los requisitos de funcionalidad. Este es el caso de la plantilla de administración para el usuario básico que se descarga nuestro CAPTCHA. El elemento más importante de esta interfaz son las tablas que muestran la lista de textos (etiquetados y sin etiquetar). Para esto, Bootstrap ofrece una manera más simple de estructurar una tabla y darle estilo. En el elemento de etiqueta table, se debe asignar al atributo class el valor table y complementariamente otros valores para dotar a la tabla de otras características, por ejemplo*: table-dark* para activar el modo oscuro, *table-striped* para que las filas tengan colores intercalados, etcétera. En nuestras tablas utilizamos el valor *table-hover* para que la fila quede resaltada cuando se posiciona el cursor sobre ella. También existe la posibilidad de resaltar filas y columnas por defecto con el valor *table-active*, no usado debido a la falta de necesidad.

Las tablas se *div*iden en las etiquetas *thead* (columnas) y *tbody* (filas). El número de columnas no varía por lo que se insertan directamente en el HTML. En cuanto a las filas, se le asigna un identificador a la etiqueta *tbody* para posteriormente añadir su contenido con ayuda de un *script*, el cual se encarga de llamar a la URL que devuelve los textos. Con el atributo *length* del array puede saber cuántos textos se han devuelto para así incluir en el HTML el número de filas correspondiente (elementos *tr* con nodos *td* agregados a él).

Una peculiaridad de Bootstrap, la cual lo dota de practicidad, es su sistema de columnado. Para poder aprovecharlo, se requiere crear un elemento, cuyo atributo class contenga el número de columnas a ocupar, la notación sería: col - [nº de columnas a ocupar por el elemento]. Cada fila está compuesta de 12 columnas, siempre, y el comportamiento de las columnas según la configuración dependerá de esto. Por ejemplo, si se desea que un *div* ocupe 3 columnas, su valor class deberá ser col-3. Una fila puede tener dos *div* de tamaño 3 y 4. El resultado será que queden 5 columnas en blanco (figura 6.6).

Figura 6.6 - El área sin texto son las cinco columnas restantes

Si a esta fila se le añade un *div* con un tamaño superior a 5 columnas (ej. 10 columnas), dicho elemento quedará desplazado por completo a la siguiente fila (figura 6.7).

Figura 6.7 - Se puede apreciar que en la nueva fila quedan dos columnas vacías

Adicionalmente, dependiendo del tamaño de ventana, se puede establecer cuántas columnas ocupará una celda. Por ejemplo, el valor col-sm-4 quiere decir que cuando el tamaño de ventana es inferior a 768 píxeles y superior a 576, la celda ocupará 4 columnas, cuando por defecto, este número podría ser 3 (col-3) para el resto de los casos.

En nuestro proyecto, hemos usado esta propiedad para repartir de manera proporcional el espacio que debían de ocupar los botones de ver y renovar la clave de usuario. Se decidió que los botones ocuparan 3 columnas cada uno y la clave 6 (figura 6.8).

Figura 6.8 - El tamaño que ocupa cada elemento se corresponde con el asignado

El resto de uso de Bootstrap fue dedicado al cambio de estilo de los elementos y cambio en su comportamiento estético (ej. cuando se posicione el cursor sobre un botón, su color cambia).

7 Conclusiones

En esta sección haremos una valoración con respecto a los objetivos que hemos podido cumplir, también explicaremos que conocimientos hemos ido adquiriendo durante la realización del proyecto. Por último, hablaremos de futuras mejoras que podrían ser implementadas.

7.1 Aprendizaje y objetivos cumplidos

Al iniciar este proyecto consideramos una serie de objetivos. Inicialmente, las primeras metas eran entender como funcionaban los servicios de CAPTCHA, en este proceso comprendimos que la forma más fácil para la persona que desarrolla una aplicación web es insertar un *script* en el código de la página para mostrar una ventana con el reto y que devuelva un mensaje sencillo sobre el resultado. También nos dimos cuenta de que por un lado hay servicios de CAPTCHA que intentan ser invisibles al usuario y por otro hay servicios que gracias al tipo de reto que presentan pueden aprovecharse para clasificar datos. Por todo esto, consideramos el primer objetivo cumplido.

Una vez terminado este proceso de investigación empezamos con el desarrollo de nuestro servicio. Principalmente nos propusimos crear un servicio de CAPTCHA que, aunque inicialmente etiquete datos sencillos como textos, pueda ser fácilmente escalable a otros tipos de datos. Con respecto a este objetivo consideramos que gracias a las herramientas que hemos elegido para implementar el servidor, añadir modelos para colecciones de otros tipos de datos, por ejemplo, imágenes, es una tarea sencilla.

Como resultado de este proceso, hemos aprendido estrategias para implementar las funcionalidades de un servidor pensando en la posibilidad de futras mejoras o añadidos y como desarrollar una interfaz para gestionar las peticiones de un componente que puede usarse desde muchas aplicaciones web.

También se ha logrado incluir un CAPTCHA de nuestro diseño a una página web. Teniendo éste un comportamiento similar al de reCAPTCHA, lo que denota que se cumplen las funciones básicas, ya que los demás CAPTCHAs estudiados, tenían características parecidas. Se abre una pequeña ventana, se selecciona una casilla/se pulsa un botón, se muestra un reto, se resuelve el reto y se procede con el formulario/acción de la página.

Durante el desarrollo de este proyecto, los conocimientos de JavaScript han sido ampliados. Ya que, al principio de éste, se trataban de conocimientos básicos. Se ha aprendido a insertar y modificar elementos con DOM, así como a configurar y hacer peticiones HTTP. Lo que ayuda a cumplir el objetivo tercer objetivo. Se ha aprendido desde cero a trabajar con el entorno Bootstrap 5: botones, tablas, columnas, etcétera. Ahora se tienen conocimientos de cómo se comunica un widget con la página con la que está incrustado. Por último, se ha aprendido cómo funcionan las plantillas y formularios de Django

y lo que ocurre cuando se hace una petición al servidor. Cumpliendo esto con el cuarto objetivo, ya que esto facilita el acceso y la edición de los datos del servidor.

7.2 Trabajo futuro

En futuras actualizaciones, se podrían crear otros tipos de diseño para que los usuarios tengan más opciones a la hora de incorporar nuestro CAPTCHA en sus páginas web. Podría hacerse uso de un modal, como se pensó inicialmente, con el objetivo de hacer su presencia dentro de la página más llamativa, pudiendo elegir una posición determinada para ello.

Otra mejora sería poder ofrecer una API en JavaScript, para que el usuario pueda personalizar el comportamiento y la apariencia del widget. Lo que podría lograrse implementando una función render, cuyos parámetros podrían ser modo (oscuro/claro), color de fondo, mostrar el reto en un modal, entre otros. En el caso de que el cliente no quiera una carga automática del widget y quiera esperarse a algún evento, con dicha función también se le podría facilitar esto. Adicionalmente, se le podría añadir a la función un parámetro opcional de *reset*, con el cual, renderizaría el widget con un botón de *reset* para que el usuario pueda volver a empezar el reto desde la primera pregunta si así lo desea.

Ofrecer accesibilidad a las personas con discapacidad visual es esencial y una mejora absolutamente necesaria para que nuestro CAPTCHA no se convierta en un impedimento, al no poder un usuario invidente enviar un formulario o realizar una acción por no poder pasar el reto al ser incapaz de leer las preguntas. Esta problemática podría derivar en una denegación injusta de servicios. Aparte de que infringiría la Ley 51/2003, de 2 de diciembre, de igualdad de oportunidades, no discriminación y accesibilidad universal de las personas con discapacidad. Para llegar a este objetivo, primero habría que hacer pruebas para asegurarse de que los sistemas de lectura de los navegadores más populares (VoiceOver, Chromevox, NVDA) puedan leer el contenido del CAPTCHA. Y si no se diera el caso, descubrir cómo hacer esto posible (cambiando/añadiendo código, empleando una nueva tecnología, etcétera). Una vez asegurado que el contenido del CAPTCHA puede ser leído por los navegadores, una posible mejora sería añadir un botón que permitiera volver a leer el contenido del widget únicamente: un ícono de accesibilidad con texto alternativo como “Repetir pregunta”. La persona en cuestión podría volver a escuchar la pregunta con dicho comando y elegir una respuesta. El texto alternativo de los botones empezaría por “Opción [nº opción]” para separar claramente unas respuestas de otras y para facilitar la contestación.

Por último, también nos gustaría crear modelos para más tipos de colecciones de datos y estudiar métodos para datos más complejos como archivos de sonido.

8 Conclusions

In this section we will make an assessment regarding the objectives that we have been able to meet, we will also explain what knowledge we have acquired during the development of the project. Finally, we will talk about future improvements that could be implemented.

8.1 Learning and objectives met

Before starting this project, we considered a series of objectives. Initially, the first goals were to understand how the CAPTCHA services worked, in this process we understood that the easiest way for the web application developer is to insert a script in the code of the page to show a window with the challenge and return a simple message about the result. We also realized that on one hand there are CAPTCHA services that try to be invisible to the user and on the other there are services that, thanks to the type of challenge they present, can be used to classify data. For all of this, we consider that the first objective has been accomplished.

Once this research process was finished, we began with the development of our service. Mainly we set out to create a CAPTCHA service that, although it initially labels simple data such as text, can be easily scalable to other types of data. Regarding this goal, we consider that thanks to the tools with which we have chosen to implement the server, adding models for collections of other types of data, for example, images, is a simple task.

As a result of this process, we have learned strategies to implement the functionalities of a server considering the possibility of future improvements or additions and how to develop an interface to manage the requests of a component that can be used from many web applications.

It has also been possible to include a CAPTCHA of our design to a web page. Having a behavior similar to that of reCAPTCHA, which denotes that the basic functions are fulfilled, since the other CAPTCHAs, which we analysed, had similar characteristics. A small window opens, a checkbox is selected/button pressed, a challenge is displayed, the challenge is solved, and the form/page action proceeds.

During the development of this project, the knowledge of JavaScript has been expanded. Since, at the beginning of it, it was about basic knowledge. We have learned to insert and modify elements with DOM, as well as to configure and make HTTP requests. Which helps to accomplish objective number 3. We have learned from scratch to work with the Bootstrap 5 framework: buttons, tables, columns, etc. We now have an understanding

of how a widget communicates with the page it's embedded to. Finally, we have learned how Django templates and forms work and what happens when you make a request to the server. Which accomplishes the last objective, because this allows easy access and modification of the data of the server.

8.2 Future work

In future updates, other types of design could be created so that users have more options when incorporating our CAPTCHA into their web pages. A modal could be used, as was initially thought, with the aim of making its presence within the page more striking, being able to choose a certain position for it.

Another improvement would be to be able to offer an API in JavaScript, so that the user can customize the behavior and appearance of the widget. What could be achieved by implementing a render function, whose parameters could be mode (dark/light), background color, show the challenge in a modal, among others. In the event that the client does not want an automatic loading of the widget and wants to wait for an event, this function could also be facilitated with this function. Additionally, an optional reset parameter could be added to the function, with which the widget would be rendered with a reset button so that the user can restart the challenge from the first question if they wish.

Offering accessibility to people with visual disabilities is essential and an absolutely necessary improvement so that our CAPTCHA does not become an impediment, as a blind user cannot send a form or perform an action due to not being able to pass the challenge due to being unable to read the questions. This problem could lead to an unfair denial of services. Apart from the fact that it would violate Law 51/2003, of December 2, on equal opportunities, non-discrimination and universal accessibility for people with disabilities. To reach this goal, it would first be necessary to test to ensure that the most popular browsers' reading systems (VoiceOver, Chromevox, NVDA) can read the CAPTCHA content. And if that's not the case, figure out how to make this possible (changing/adding code, using new technology, etc.). Having ensured that the content of the CAPTCHA can be read by browsers, a possible improvement would be to add a button that would allow re-reading the content of the widget only: an accessibility icon with alternative text such as “Repeat question”. The person in question could listen to the question again with said command and choose an answer. The alternative text of the buttons would begin with “Option [option number]” to clearly separate some answers from others and to facilitate the answer.

Lastly, we would also like to create models for more types of data collections and study methods for more complex data such as sound files.

9 Contribuciones personales

Para finalizar, explicaremos cómo hemos contribuido cada uno de los integrantes a la realización del proyecto.

9.1 Gemma González Palacios

Investigación previa

Antes de empezar la implementación y ya sabiendo qué tecnologías tendría que utilizar, procedí a estudiar estas tecnologías, ampliando mis conocimientos en ellos o adquiriéndolos de cero cuando se dio el caso. La única tecnología en la que no tenía ningún conocimiento era el entorno Bootstrap, por lo que empecé visionando tutoriales [19] en los que se explicaban los conceptos básicos y algunos intermedios de esta tecnología. La longitud de estos vídeos oscilaba entre los 4 y los 16 minutos. El visionado de esos vídeos se produjo seguido, gracias a una lista de reproducción, la cual se encontraba a disposición de los usuarios en la página que hospedaba estos vídeos. Si bien con paradas entre unos vídeos y otros, en las que probé el código de ejemplo para explorar su funcionamiento y el mecanismo que lleva a él, tratando de entender todo lo que se mostraba para poder ponerlo personalmente en práctica cuando empezase a emplear este entorno en la implementación de mi parte del proyecto. Era importante visionar el curso entero, ya que conocer todo lo que se puede hacer con una herramienta facilita su uso en la práctica porque se sabe que se puede utilizar cierta característica para lograr cierto objetivo. Lo que ahorra mucho tiempo de investigación sobre cómo lograr un objetivo.

Fue necesario también, familiarizarme con el funcionamiento de otros CAPTCHAs, así como de conocer su implementación. Con el objetivo de tener una idea sólida de lo que son para que nuestro CAPTCHA, se comportara como uno y no como otro tipo de widget distinto, por eso es necesario analizar la competencia. En esta memoria sólo he escrito un análisis sobre hCaptcha. Sin embargo, también estudié el uso e implementación de recaptcha y otros CAPTCHAs básicos encontrados en GitHub.

Implementación

Una vez entendidos los requisitos iniciales del CAPTCHA, procedí a la creación del primer prototipo, compuesto de un fichero HTML con JavaScript incrustado en él. Este prototipo únicamente dibujaba el JSON recibido del servidor. La manera de elegir la respuesta era un grupo de radio *buttons* y era necesario pulsar un botón de siguiente para recibir proceder a la siguiente pregunta del reto. Todavía no estaba disponible el servidor Django para recibir los JSON, por lo que creé uno dentro del script como prueba para

comprobar que pudiera representar su contenido sin problemas. Una vez fue creado el servidor, fue posible representar los datos que enviaba.

Poco después al volverse el código script más voluminoso y por recomendación de los directores, trasladé el código JavaScript a un fichero externo a incluir en el HTML principal. También empecé a tener en cuenta el aspecto visual por lo que añadí Bootstrap con el objetivo de mejorarlo. Sin embargo, por problemas con CORS, no se cargaba su contenido y daba el formato limpio a los elementos. Quedando la página principal con un aspecto algo rudimentario, reminiscente de los días en los que la versión más actual de Windows era Windows XP.

Ya con un funcionamiento básico, el siguiente paso fue incrustar el CAPTCHA dentro de una página web con un formulario para simular el comportamiento que tendría en el caso de uso más conocido. Para ello, descargué el código de una página web de prueba que contenía un formulario de contacto. Extendí la plantilla de dicho formulario, añadiéndole otro campo y en dicho campo inserté un *iframe* que daría acceso al fichero HTML del CAPTCHA. En la primera versión, antes de que fuera posible recibir el veredicto del servidor de si se trata de un humano o no, modifiqué el código JavaScript de la página de ejemplo para que tuviera el botón de enviar el formulario deshabilitado hasta simplemente todas las preguntas se hubieran contestado. En versiones posteriores, para que el botón se habilite de nuevo, es necesario obtener un veredicto positivo del CAPTCHA.

Habiendo conseguido el comportamiento deseado en cuanto a la interacción entre el CAPTCHA y el formulario. Procedí al embellecimiento de la apariencia del CAPTCHA. Para entonces, los problemas con Bootstrap ya se habían solucionado, por lo que pude usarlo de apoyo. Por recomendación de los directores, cambié la manera en que se selecciona la respuesta de un grupo de radio *buttons* a un grupo de botones que requieren solamente de un clic para proceder a mostrar la siguiente pregunta, lo que aportó a la mejora de la usabilidad. Cambié el color de fondo a blanco y la fuente.

Modifiqué dos plantillas HTML de Django: *login*, registro. Creé una plantilla para la interfaz de usuario, cuyas funciones consisten en poder ver y renovar la clave de la API (*API key*) y ver la lista de retos (etiquetados y sin etiquetar). También consideré, que era de utilidad para el usuario poder ver el texto de los retos, su fiabilidad y la elección correcta en caso de los etiquetados. Por lo que habilité un botón, el cual abre un diálogo que muestra esta información, sin ser posible la modificación.

En cuanto a la interfaz de administrador, no consideré necesario crear una nueva, ya Django proporciona una bastante extensa. Únicamente la modifiqué para incorporar la funcionalidad de subir una colección desde un archivo CSV.

Memoria

Una vez estuvo terminado el código, llegó el momento de empezar a escribir esta memoria, para lo que se necesitó una preparación previa, ya que un documento de este calibre requiere conocimientos amplios sobre la materia sobre la que se escribe, así como un buen entendimiento del proceso de desarrollo. Repasé todos los pasos realizados hasta ese punto, recabé información sobre los temas a tratar y revisé el código. Durante la escritura, leí en varias ocasiones el código que había desarrollado, con el fin del cubrir la descripción de éste de la mejor manera posible. Así como el de mi compañero, para asegurarme de no incluir ninguna afirmación errónea sobre el proyecto.

9.2 Adrián Salvador Crespo

Análisis del proyecto

Para esta etapa inicial, mi tarea consistió principalmente en aprender sobre cómo funciona el etiquetado de datos para el entrenamiento de inteligencias artificiales.

También investigué sobre sistemas de etiquetados de datos en la actualidad, descubriendo que existían servicios de CAPTCHA que realizaban una función de etiquetado de datos

Investigación de servicios de CAPTCHA

Durante esta etapa comencé analizando otros servicios de CAPTCHA. Para ello primero estuve leyendo la documentación online y algunos artículos sobre las diferentes versiones que se habían publicado de los servicios que investigué también visioné algunas demos para ver como un desarrollador podía integrar estos servicios en su aplicación web y por último diseñe una página con HTML que contenía un formulario con el que pude utilizar yo mismo el componente de estos servicios y comprobar cómo era el funcionamiento al cargar la página, solicitar, recibir el reto y procesar las respuestas o la información que envía sobre el usuario.

Entre ellos cabe destacar reCAPTCHA, al ser uno de los servicios más utilizados, pero también investigué acerca de otros servicios, como FriendlyCAPTCHA.

El resultado de esta investigación fue aprender como la persona que desarrolla una aplicación web puede incluir de forma sencilla el componente para mostrar los retos y como ese componente responde a la aplicación.

Investigación de herramientas para desarrollar el servicio de CAPTCHA

En esta etapa del proyecto decidimos que yo me encargaría principalmente del backend. Una vez *div*idido el trabajo de esta manera investigué que tecnologías podría emplear para desarrollar el servidor. Actualmente hay múltiples opciones, hice algunas pruebas con Node.js y Apache, pero finalmente me decidí por Django.

Antes de empezar con el diseño del servicio, realice una serie de pruebas de cómo se realizaba el despliegue y la configuración inicial de un servidor con Django. Una vez hecho esto Django ofrece una consola de usuario con la que puedes interactuar con el servidor, con ella investigue cómo funcionaba el sistema de modelos.

Una vez decidido la tecnología para el servidor comencé a diseñar en papel la estructura que tendría la base de datos y que funcionalidades necesitaría para la comunicación con el cliente.

Implementación y testeo

Esta fue la etapa de desarrollo del servidor. Todo mi trabajo estuvo dedicado a la parte del backend, implementar los modelos y las interfaces de funciones para las peticiones del cliente.

El primer prototipo era una versión sencilla, consistía en una base de datos que almacenaba solo los textos con un número fijo de opciones por reto sin agrupar por colección y una versión inicial de la interfaz de funciones para el componente del reto. El siguiente paso fue añadir el modelo de colecciones, además de modificar los métodos que emplea el componente para añadir el filtrado por palabras clave y a evitar y, las funcionalidades de cargar y descargar colecciones. Después, continué con la gestión de usuarios, es decir, las funciones de registro, iniciar sesión y crear la clave de usuario.

La parte de testeo que principalmente se entrelazaba con la implementación consistió en realizar las comprobaciones necesarias de que las funcionalidades que estaba añadiendo se comportaban de forma correcta.

Por último, cuando la implementación ya estuvo terminada realicé más pruebas de casos de uso con la versión funcional del servicio de CAPTCHA y también aproveché esta fase para buscar en redes sociales textos reales que poder añadir a la base de datos

principalmente en comprobar el correcto funcionamiento del servidor y sus respuestas a las peticiones del cliente.

Escritura de la memoria

Finalmente, nos reunimos para definir la estructura que tendría la memoria y repartimos las diferentes partes para su redacción.

Mi trabajo en la memoria consistió principalmente en escribir los apartados de resumen. En el apartado de preliminares, redacté las secciones de *Beneficios adicionales de los CAPTCHA* y los resultados de mi investigación sobre el servicio de reCaptcha, aunque respecto a la información que obtuve sobre FriendlyCAPTCHA no consideré incluirla en la memoria ya que era un sistema muy similar a la versión tres de reCaptcha. Finalmente, para el apartado de arquitectura y el capítulo sobre el diseño del backend, me encargué de realizar la redacción y las capturas de pantalla necesarias para las figuras.

10 Bibliografía

[1] «What Is Fake Account Creation and how to prevent it»: https://www.perimeterx.com/solutions-by-threat/fake-account-creation/

[2] «What Does CAPTCHA Mean?»: https://www.imperva.com/learn/application-security/what-is-captcha/

[3] «reCAPTCHA»: https://www.google.com/recaptcha/about/

[4] «Developer Guide»: https://docs.hcaptcha.com

[5] «HTML: Lenguaje de etiquetas de hipertexto»: https://developer.mozilla.org/es/docs/Web/HTML

[6] «CSS»: https://developer.mozilla.org/es/docs/Web/CSS

[7] «Trabajando con JSON»: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

[8] «HTTP»: https://developer.mozilla.org/es/docs/Web/HTTP

[9] «Introducción - Referencia de la API Web»: https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction

[10] «About Pyhton»: https://www.python.org/about/

[11] «Acerca de JavaScript» https://developer.mozilla.org/es/docs/Web/JavaScript/About_JavaScript

[12] «The web framework for perfectionists with deadlines.»: https://www.djangoproject.com/

[13] «About SQLite»: https://www.sqlite.org/about.html.

[14] «Django Widget Tweaks»: https://pypi.org/project/django-widget-tweaks/

[15] «Get started with Bootstrap»: https://getbootstrap.com/docs/5.2/getting-started/introduction/

[16] «About Git»: https://git-scm.com/about

[17] «SQLite Viewer»: https://marketplace.visualstudio.com/items?itemName=qwtel.sqlite-viewer

[18] «Comma-separated values - RFC 4180 and MIME standards»:
https://en.wikipedia.org/wiki/Comma-separated_values#RFC_4180_and_MIME_standards

[19] T. N. Ninja, «Bootstrap 5 Tutorial»:
https://www.youtube.com/playlist?list=PL4cUxeGkcC9joIM91nLzd_qaH_AimmdAR