IABRAILLE

Proyecto Final de Desarrollo de Aplicaciones Web

JUDITH BARRAJÓN ALCARAZ
JESUÏTES EL CLOT
DAW2

Índice

1. Resumen inicial

IABraille es una solución web con integración en dispositivos móviles que ayuda a las personas con visión reducida a aprender Braille en catalán de manera sencilla. Gracias a la cámara del móvil, la aplicación se encarga de escanear modelos del abecedario en braille y traducirlos al alfabeto latín. Además, también dispone de una librería en la que se pueden consultar todas las letras y su equivalente en Braille.

IABraille és una solució web amb integració en dispositius mòbils que ajuda les persones amb visió reduïda a aprendre Braille en català de manera senzilla. Gràcies a la càmera del mòbil, l'aplicació s'encarrega d'escanejar models de l'abecedari a braille i traduir-los a l'alfabet llatí. A més, també disposa d'una llibreria on es poden consultar totes les lletres i el seu equivalent a Braille.

IABraille is a web solution with integration into mobile devices that helps people with vision difficulties to learn Braille in Catalan in a simple way. Through the mobile camera, the application scan models written in Braille alphabet and translate them into Latin alphabet. In addition, it also has a library where you can search for all the letters and their equivalent in Braille.

2. Estudio de viabilidad

2.1. Establecimiento del alcance del sistema

La aplicación utiliza un cliente sencillo para que personas de cualquier edad entiendan su funcionamiento, con un diseño responsivo adaptado a posibles dificultades visuales y colores cálidos que resultan agradables a la vista. En su desarrollo, podemos ver que su principal motor es el *FrontEnd*, ya que incorpora la API que permite el escaneo de las letras. Además, realiza peticiones a un *BackEnd* tanto para solicitar las fichas con el alfabeto en Braille como para crear un servidor seguro que permita el acceso a la cámara, tal y como demanda el motor de escaneo de *Teachable Machine*.

2.2. Estudio de la situación actual

En el mercado ya existen páginas web o aplicaciones nativas que ayuden a aprender Braille, pero acostumbran a ser páginas simples y que no ofrecen una interacción por parte del usuario. También, los idiomas que trabajan se limitan a lo más populares como inglés o español. Por lo tanto, hemos considerado interesante presentar una aplicación capaz de hacer al usuario partícipe mediante el escaneo de los caracteres, además de ofrecer el catalán como idioma principal de ésta.

2.3. Definición de los requisitos del sistema

Por un lado, si el usuario final desea utilizar la aplicación no requiere de sistemas o complementos complejos, sino de conexión a internet y un dispositivo Android. Es importante que éste último se cumpla ya que ciertas funcionalidades como el escaneo de caracteres no están preparados para productos del entorno Apple. También, a la hora de conectar con el servidor web, al ser un servidor seguro con un certificado autofirmado, necesita una solicitud de seguridad aceptada por el usuario.

Por otro lado, en cuanto al entorno de desarrollo, es necesario tener instalado las siguientes tecnologías: *Vite*, como plataforma de construcción de proyectos *FrontEnd* como *React*, *Vue* o *Angular*; *NodeJs*, para instalar paquetes mediante NPM además de para el uso del *BackEnd* en *ExpressJs*; *MongoDB*, como base de datos no SQL que contiene la información de cada una de las letras o signos del alfabeto latín.

2.4. Selección de la solución, estudio de las alternativas de solución y valoración de las alternativas.

Uno de los primeros pasos a llevar a cabo cuando ya disponemos de una idea para nuestro proyecto, es evaluar las posibles tecnologías que encajan con él. Para poder llevar a cabo IABraille, tuvimos en cuenta aspectos de FrontEnd, BackEnd y Bases de Datos que pudieran encajar en el modelo estándar actual y con posibles actualizaciones de la aplicación en el futuro.

El primer paso fue escoger la tecnología encargada de la parte cliente. Para poder realizar aplicaciones web rápidas y sencillas que permitan ofrecer al usuario una interacción cómoda y agradable, pensamos que la mejor opción *FrontEnd* son los SPA (Simple Page Aplicación). Los SPA permiten al navegador cargar todo el contenido en solo una única página, es decir, los componentes son dinámicos y se muestran en base la interacción del usuario. En términos más tecnológicos: existe un único documento html y son las "vistas" las encargadas de moverse por el documento cuando ésta es llamada. De esta forma, permite que las transiciones sean rápidas y las comunicaciones de cliente-servidor más fluidas. Los frameworks más conocidos son ReactJs, VueJs y Angular, cada uno desarrollado por una compañía diferente.

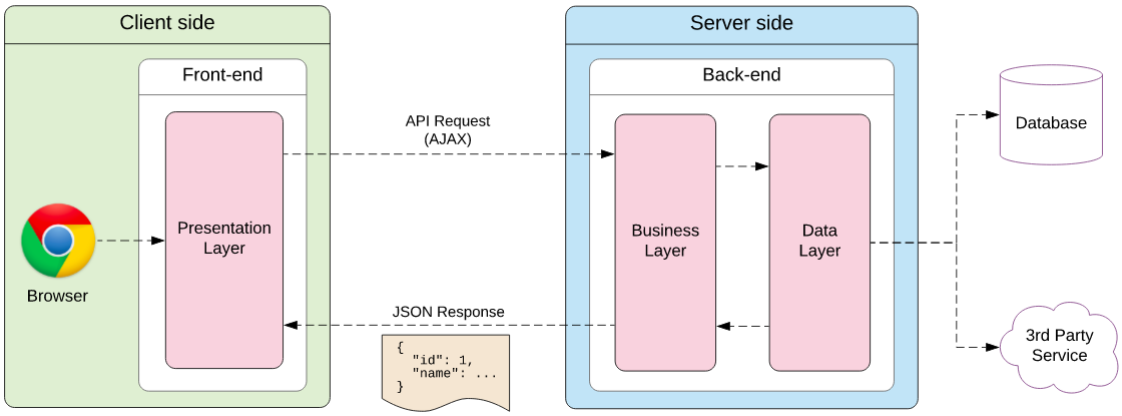

Imagen 1 - Arquitectura SPA[1]

[1] https://elmprogramming.com/what-is-a-single-page-app.html

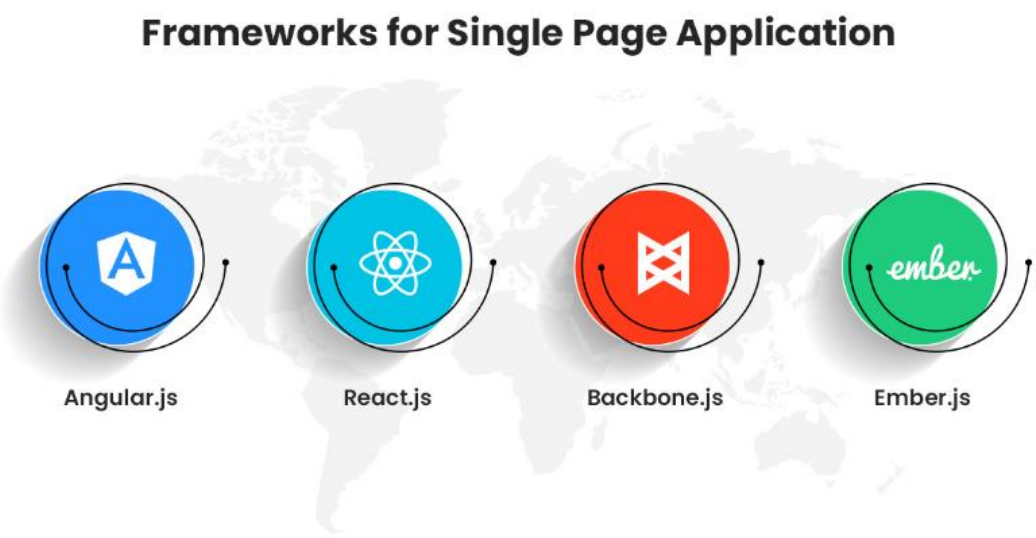

Imagen 2 - Principales Frameworks[2]

Existen otro tipo de soluciones FrontEnd enriquecidas para el uso nativo en dispositivos Smart. Éste son el caso de ReactNative y Flutter. La principal idea que teníamos era usar ReactNative, ya que es una tecnología que conocíamos y encajaba con los ideales de nuestro proyecto. No obstante, para poder desarrollar más cómodamente con esta tecnología nos veíamos sujetos a usar ExpoJs. ExpoJs ofrece un entorno de trabajo más cómodo para el desarrollador a la hora de trabajar con ReactNative sin necesidad de tener otras dependencias o software como Android Studio o XCode. Sin embargo, aunque parecía la opción más venidera, no era compatible con el motor de *machine learning* utilizado para IABraille: Teachable Machine. De esta forma, valoramos usar Flutter a pesar de que era una tecnología nueva y de la que no teníamos experiencia, pero nos encontramos con el mismo problema que con ReactNative, la librería de Teachable Machine llevaba meses sin estar actualizada y no era compatible con las versiones actuales de Flutter. Es por esa razón que se decidió que IABraille estaría desarrollada como un SPA usando ReactJS con un diseño responsivo adaptado a dispositivos móviles.

[2]https://www.angularjsindia.com/blog/top-single-page-application-frameworks-to-consider-for-web-apps/

Imagen 3 - Soluciones Nativas[3]

Una vez decidida la parte del lado cliente, escoger la tecnología del lado servidor resultó ser un trabajo más sencillo. Debido a que la idea de utilizar un SPA o de crear una aplicación nativa estuvo muy presente y, desde un inicio, se descartó utilizar como solución un MVC (Modelo-Vista-Controlador)[4], era necesario pensar en un servidor simple que acarreara con pocas funciones, ya que los SPA no ofrecen ese tipo de recurso. En nuestro caso, la mayor parte de la responsabilidad de la aplicación recae en la experiencia de usuario (UX), es decir, el FronEnd, por lo que el servidor únicamente de solicita para una conexión segura (https) y enviar respuestas a las peticiones del cliente. Pensamos que NodeJs era innecesario ya que el proyecto implementaría una API Restful, y ExpressJs es un buen candidato como Web Server. Además, éste se programa con menos líneas de código y es compatible con otras funciones que podrían aplicarse en un futuro.

Lo mismo ocurrió a la hora de escoger base de datos. Cualquier solución podría estar bien, ya que ésta únicamente lee los datos de un JSON y los almacena. Pensamos que, al tratarse un JSON, MongoDB resuelve con más facilidad la lectura de datos semiestructurados y se almacena como un objeto de igual modo. Otra posible opción

[3] https://tekify.vn/flutter-react-nativie-which-framework-to-choose/
[4] Los modelos-vista-controlador son soluciones que contienen el "todo", es decir, es una estructura de tres capas que implementa la interfaz de usuario, los datos y su lógica en un mismo espacio. Ofrece un mejor mantenimiento y división del trabajo.

que se planteó fue el uso de IndexedDB, no obstante, se descartó ya que no queríamos sobrecargar el cliente.

Imagen 4 - Logo MongoDB

Por último, toda la aplicación de IABraille gira en torno a su principal motor de escaneo, que se ejecuta gracias a Teachable Machine. Ésta es una tecnología de Inteligencia Artificial que trabaja con tensorflow, una potente herramienta de machine learning. Para poder comprender las tareas que realiza Teachable Machine, es necesario entender qué es el machine learning. Machine Learning es una forma de estudio de datos masivos para uso en el campo de la Inteligencia Artificial (IA) que estudia patrones y genera predicciones de éstos. Como el objetivo de IABraille es hacer que el usuario interactúe utilizando la cámara para escanear código Braille y traducirlo al catalán, es necesario implementar este tipo de tecnología que devuelve un modelo entrenado con la traducción.

Para poder desarrollar en Machine Learning, el mejor lenguaje de programación es Python. Sin embargo, existen soluciones que generan modelos entrenados adaptados a diferentes tipos de código. Éste es el caso de Teachable Machine, una herramienta que proporciona Google fácil de utilizar y que ofrece soporte tanto para imagen y grabación de video, como para sonido o posturas corporales.

Estos modelos que entrena Google devuelven código que puedes adaptar a tu proyecto además de ficheros donde se encuentran los modelos entrenados. Los códigos que dispone son en Tensorflow.js (javascript), Keras (framework de Python), y TensorflowLite (para móviles Android). Particularmente, en IABraille hemos usado la api de Tensorflow.js.

Existen otras empresas encargadas de crear modelos entrenados como la de Google: Amazon Machine Learning o MLJAR. No obstante, la documentación que ofrecen es más tosca en comparación a la sencillez que propone Google.

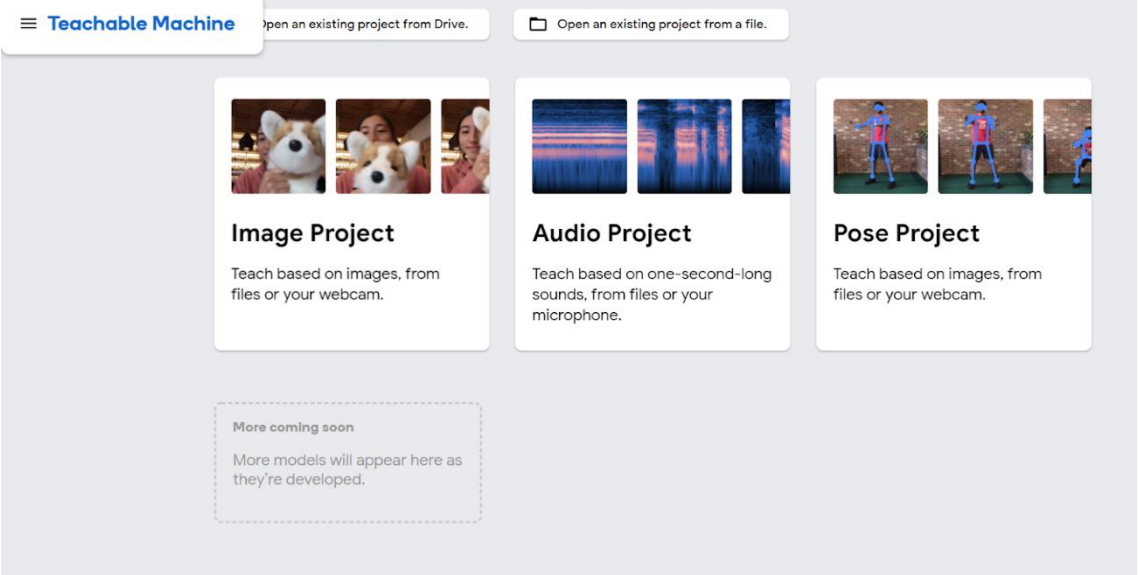

Imagen 5 - Captura Teachable Machine

Otras opciones que se han valorado para poder proporcionar un aspecto nativo en la aplicación era crear una PWA (Progressive Web App) y un WebView con React Native.

Por una parte, las PWA son aplicaciones web que permiten un uso nativo sin llegar a ser aplicaciones propiamente nativas. La diferencia que ofrecen a las apps móviles es que no requieren de la tienda online (Google Play, App Store) para descargarse, sino que el usuario puede "instalarlas" desde el navegador en la pantalla de inicio haciendo como un acceso directo a la página web. Otra de las ventajas es que el usuario puede usarla sin conexión y además recibir notificaciones push.

Imagen 6 - Funcionalidades de las PWA[5]

Por otra parte, un WebView en React Native es una aplicación nativa que redirige a una página web, por lo que es instalable en los dispositivos, pero alberga la conexión y el aspecto de una página web.

Más adelante entraremos en profundidad en cómo hemos desarrollado estas ideas, y de por qué no han resultado como se esperaba.

[5] https://medium.com/react-adventure/introduccion-a-las-pwa-e578100b9015

3. Análisis del sistema

3.1. Definición del sistema

La aplicación se define de la siguiente forma:

- **FrontEnd**: SPA con el framework de ReactJs. Para poder llevar a cabo las funcionalidades de machine learning se usa la herramienta de TensorFlow mediante el Teachable Machine de Google. Además, para ayudar a hacer el diseño responsive se hace uso de la librería de Tailwind.
- **BackEnd**: ExpressJs con una conexión a MongoDB. Protocolo web HTTPS con certificado auto-firmado. Api Rest
- **Comunicación cliente-servidor**: mediante Fetch Api obtenemos los recursos de forma asíncrona.

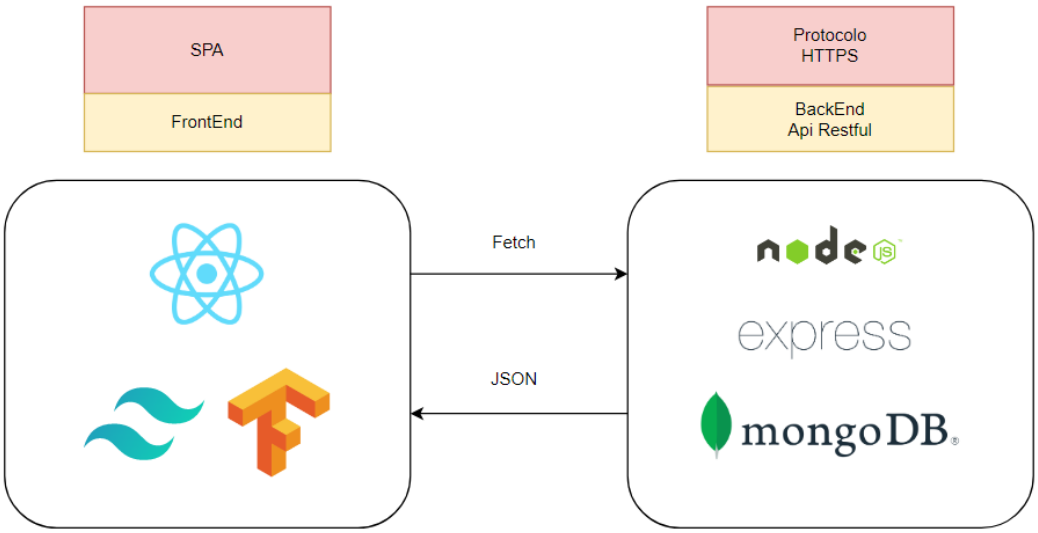

Imagen 7 - Definición del sistema

3.2. Establecimiento de los requisitos

En el siguiente diagrama de casos de uso se explica el papel que tiene el usuario en el uso de la aplicación:

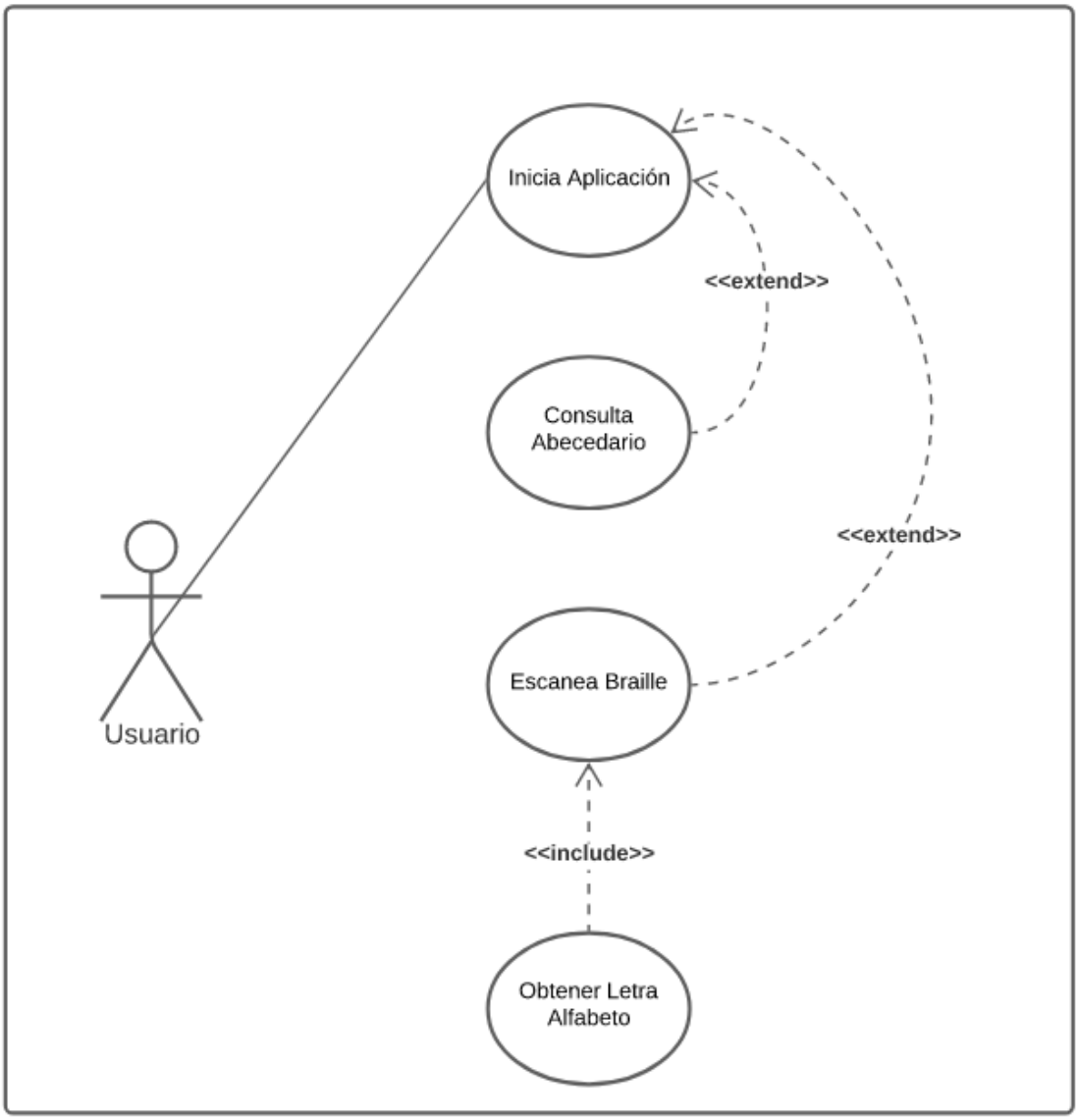

Imagen 8 - Diagrama casos de uso

3.3. Definición de la interfaz de usuario

El diseño de la aplicación se forma con el siguiente patrón:

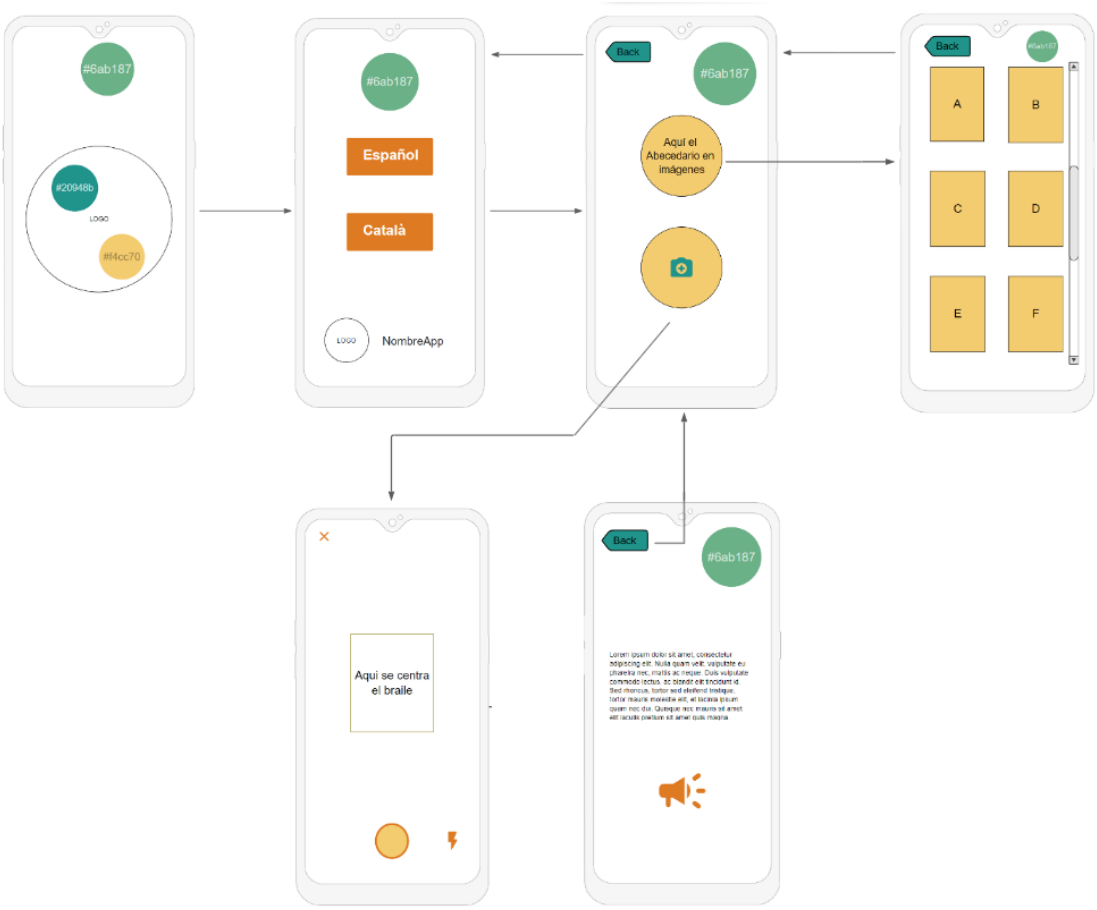

Imagen 9 - MockUp Inicial de la IABraille

3.4. Especificación del plan de pruebas

La aplicación dispone de un backend que hace consultas a una base de datos, por lo tanto, es importante comprobar que las peticiones de éste se cumplen.

La parte más complicada y delicada el proyecto la lleva a cabo el escáner de letras. En éste tendremos que comprobar si realmente está traduciendo la escritura en Braille a la letra correspondiente.

También, será necesario un test de usabilidad para comprobar que la aplicación es accesible en diferentes usuarios.

4. Diseño del sistema

4.1. Arquitectura de información, guía de estilos, usabilidad, accesibilidad.

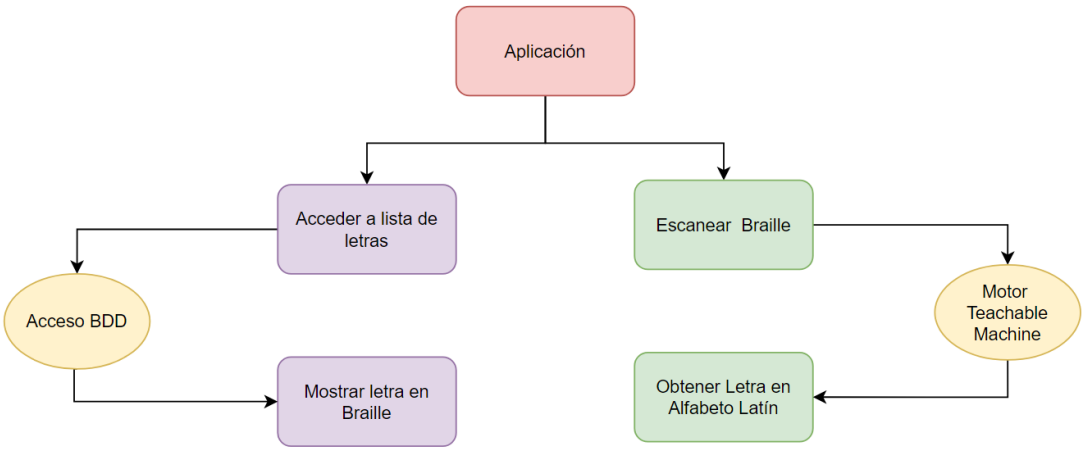

Imagen 10 - Arquitectura de la información

Guía de estilos

- **Colores:**

La aplicación se rige por la combinación de cuatro colores principales:

	Mar	#20948b	RGB: 32, 148, 139
	Naranja	#de7a22	RGB: 222, 122, 34
	Lago	#6ab187	RGB: 106, 177, 135
	Arena	#f4cc70	RGB: 244, 204, 112

Los colores fuertes son los encargados de señalar marcos, iconos o letras, mientras que los colores claros suelen usarse como relleno o colores de fondo.

Imagen 11 - Botón para Acceder a la biblioteca de letras

En ocasiones se hace uso de colores complementarios como es el caso del morado, que es usado en las fichas de Braille:

Morado #46439b RGB: 70, 67, 155

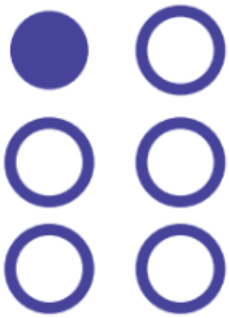

Imagen 12 - Ficha de la letra "a" en Braille

También el blanco y en ocasiones el negro se utiliza para complementar las pocas partes de texto que contiene la aplicación.

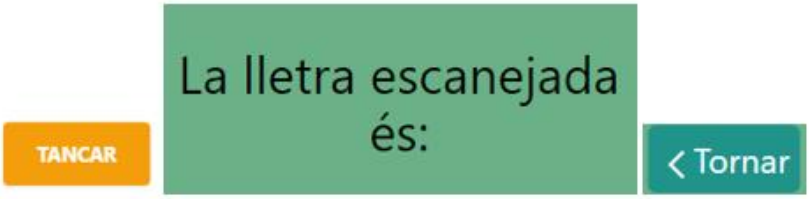

Imagen 13 - Ejemplo de colores en texto

- ## Tipografía:

La tipografía utilizada es Roboto, ya que se trata de una escritura ligera y que combina muy bien con las aplicaciones con aspecto nativo.

- ## Logo:

El logo es una parte importante a la hora de crear aplicaciones que puedan ser nativas ya que es la cara de la empresa y la propia app. En nuestro caso hemos querido combinar los colores que representan a IABraille con una adaptación del lenguaje Braille para escribir la palabra “Braille”.

Imagen 14 - Logo completo de IABraille

Imagen 15 - Parte del Logo que tiene escrito "Braille"

Además de simular la escritura en Braille, la “I” recuerda a los comandos de ascensor que también disponen de escritura en Braille.

- ## Diseño responsivo:

La aplicación responde a diferentes tamaños dependiendo de la pantalla en la que se esté usando:

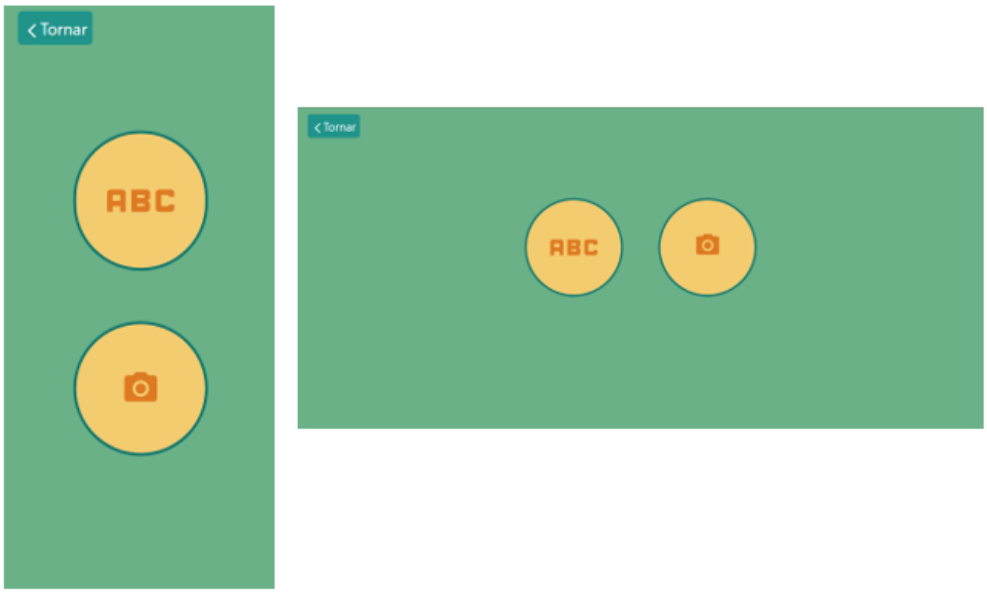

Imagen 16 - Equivalencia Responsive 1

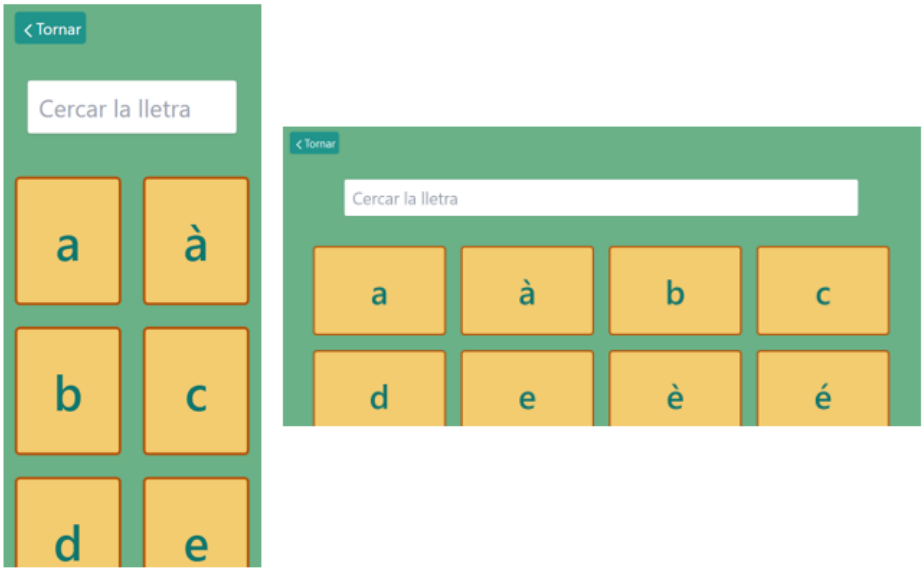

Imagen 17 - Equivalencia Responsive 2

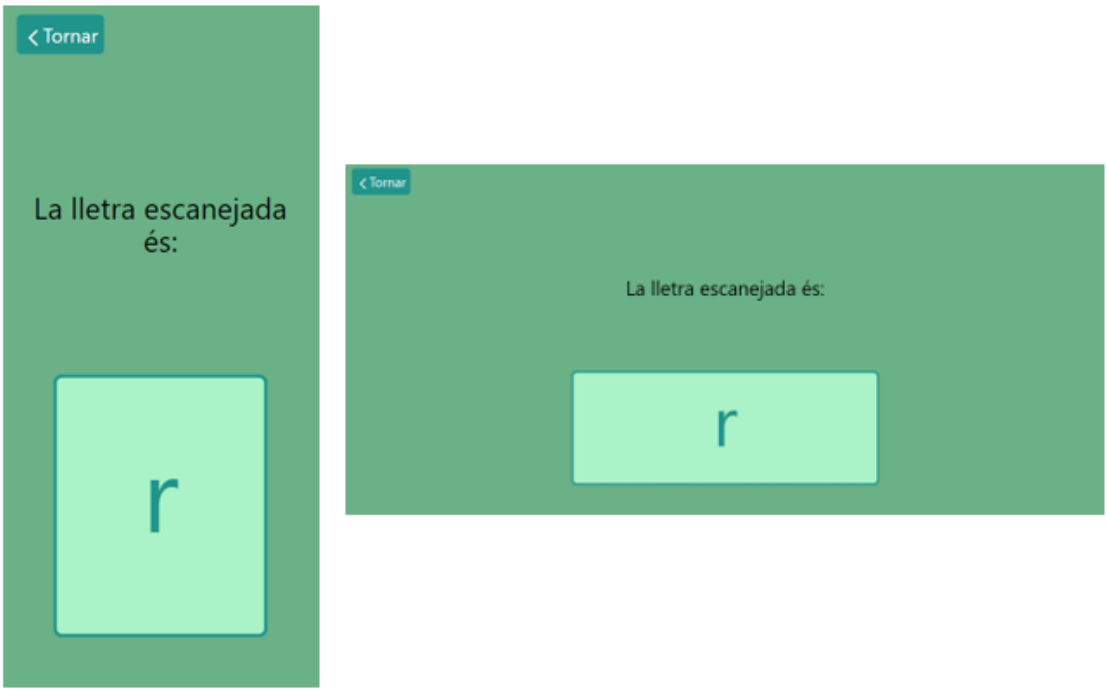

Imagen 18 - Equivalencia Responsive 3

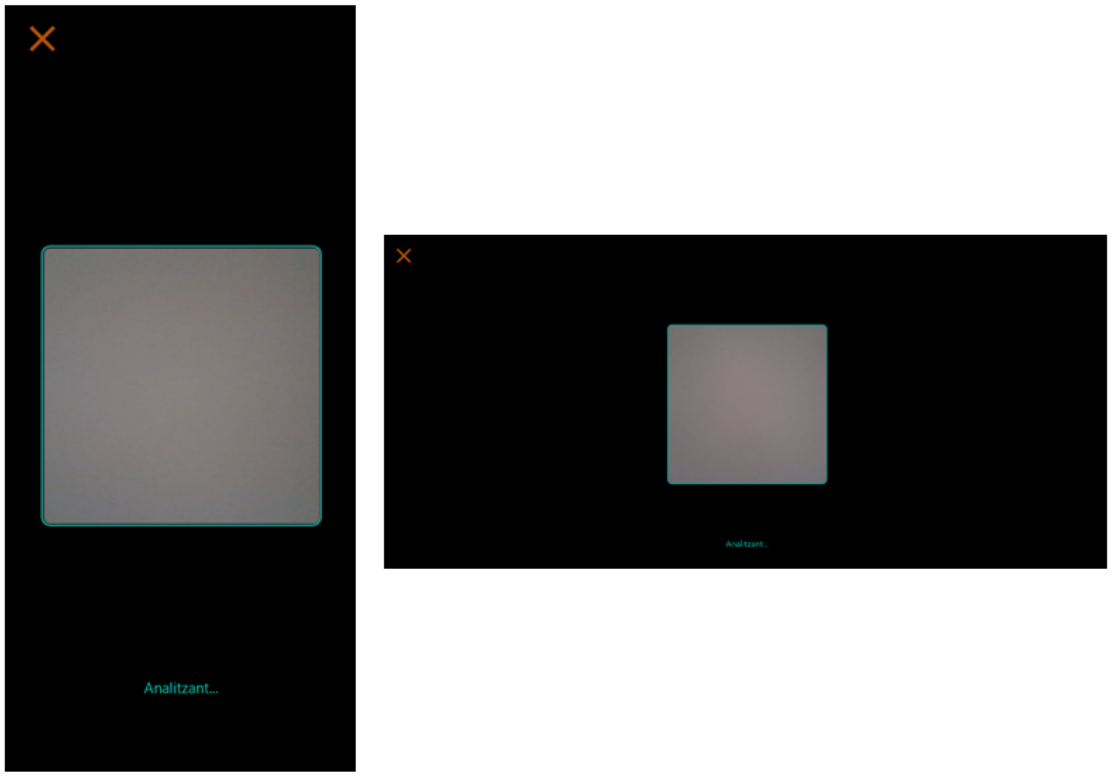

Imagen 19 - Equivalencia Responsive 4

Accesibilidad y usabilidad

Los colores y el tamaño de las letras están escogidos de manera que las personas con dificultades visuales puedan sentirse cómodas a la hora de usar la aplicación. También IABraille pretende que personas de cualquier edad puedan utilizarla fácilmente, por lo que no se encuentra sobrecargada de contenido ni de conceptos complicados de comprender.

El tamaño de las letras es grande para que los usuarios no tengan que hacer un sobreesfuerzo visual. Los colores escogidos son cálidos y tienen simetría entre ellos. Asimismo, se trata de colores pastel y no de aquellos que irradian mucho brillo. Además, se defino como una paleta dinámica y animada que encaja con el concepto de la aplicación.

Todas las imágenes disponen de un título alternativo, para que éstas puedan ser leídas con transcripción de voz.

En futuras actualizaciones, como comentaremos más en adelante, la aplicación dispondrá de su propia transcripción de letras.

4.2. Arquitectura Web

4.2.1. Definición de niveles de arquitectura

El proyecto se ha desarrollado como una aplicación de dos capas: un SPA con React en la parte cliente y un servidor conectado a una base de datos con ExpressJs y MongoDB.

La estructura del cliente con React se ha llevado a cabo mediante Vite, una herramienta de “buildeo” de aplicaciones para diferentes soluciones en el FrontEnd. Generado el proyecto base, usando ReactJs, hemos implantado diferentes componentes (también conocidos como clases), que se encargarán de mostrar las vistas de la aplicación. La propia solución nos permite tener un dinamismo ligero, sin tropiezos en el despliegue, que gracias a herramientas como hooks o props nos ha facilitado enviar datos entre vistas y así tener el código mejor estructurado.

Para poder movernos entre las diferentes vistas, hemos usado una Api de Routing que hemos implementado en el componente principal del proyecto. De esta forma, cualquier enlace que creemos queda ligado a la raíz.

El motor de tensorflow mediante Teachable machine, como ya hemos informado antes, lo requiere también el FrontEnd. Nos conectamos con los servicios de Google gracias a un CDN que llamamos en el documento principal de la aplicación: index.html. El modelo entrenado se genera directamente en la propia página de Teachable Machine. Más adelante explicaremos cómo funciona su plataforma.

La conexión con el servidor la realizamos mediante la api Fetch, que se solicita cuando el cliente entra en la ruta de abecedario.

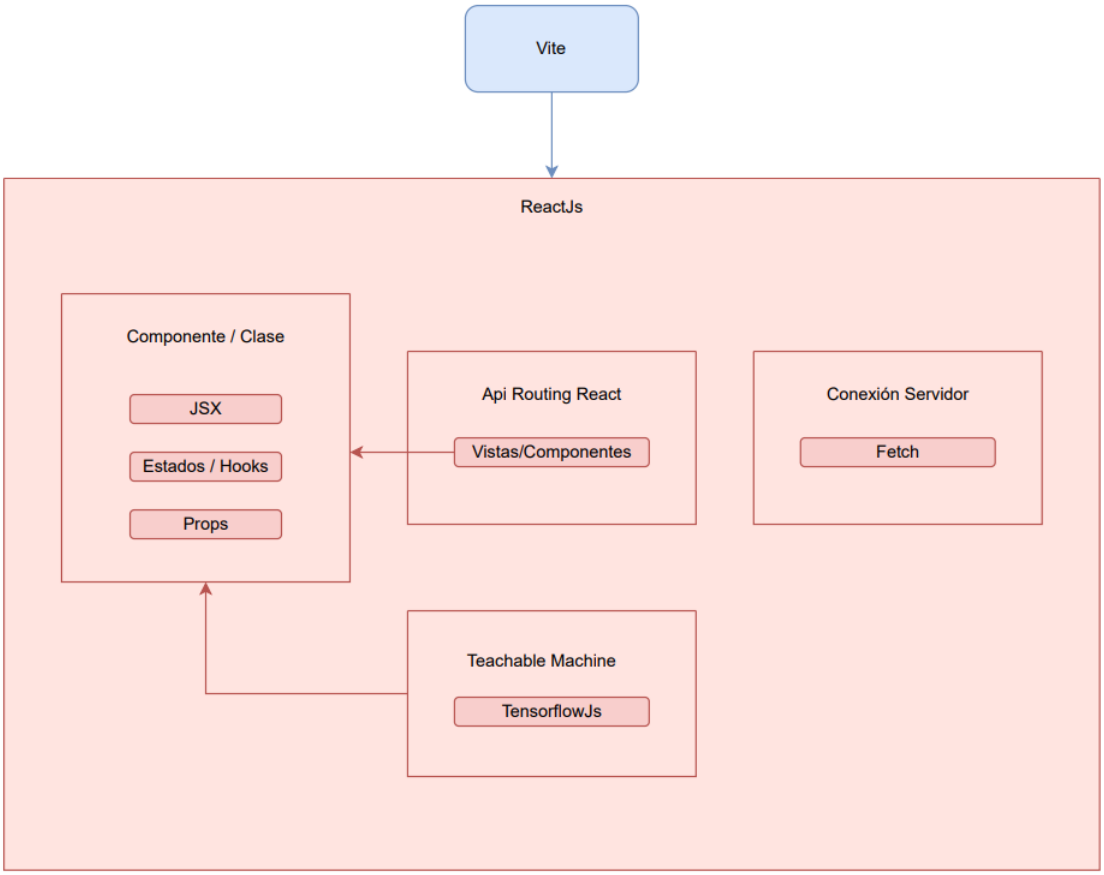

Imagen 20 - Esquema del desarrollo del Cliente

El desarrollo del servidor ha resultado más sencillo que el propio en el cliente. Simplemente hemos desplegado tres archivos divididos en servidor, Rutas y DataBase, cada uno con una clara función.

El primero será el encargado de llamar al protocolo HTTPS. Es imprescindible que éste sea HTTPS ya que es seguro y es algo que se demanda a la hora de crear PWA (Progressive Web Aplication) y, sobre todo, para el uso de Teachable Machine. Este requiere un acceso a la cámara, por lo que un protocolo que no sea seguro no funciona.

Server también contiene el acceso a las rutas, que serán verificadas por el CORS. También, para que el protocolo sea seguro ha sido necesario crear un certificado autofirmado. Por último, para poder realizar provisionalmente una PWA medianamente funcional, se requiere de Service Worker para que funcione en segundo plano, aunque la aplicación no se esté activa. Este código del SW permitirá a la PWA funcionar, a pesar de que no exista conexión con la página web. Sin embargo, el resultado que hemos obtenido de esta solución no era el esperado, así que hemos priorizado otros desarrollos.

El archivo de Rutas contiene las peticiones a la base de datos para obtener los datos que ésta contiene.

Para finalizar, tenemos el fichero de DataBase que hará el llamamiento a mongodb, y recogerá una información que previamente habrá sido solicitada por el cliente gracias a la api de Fetch.

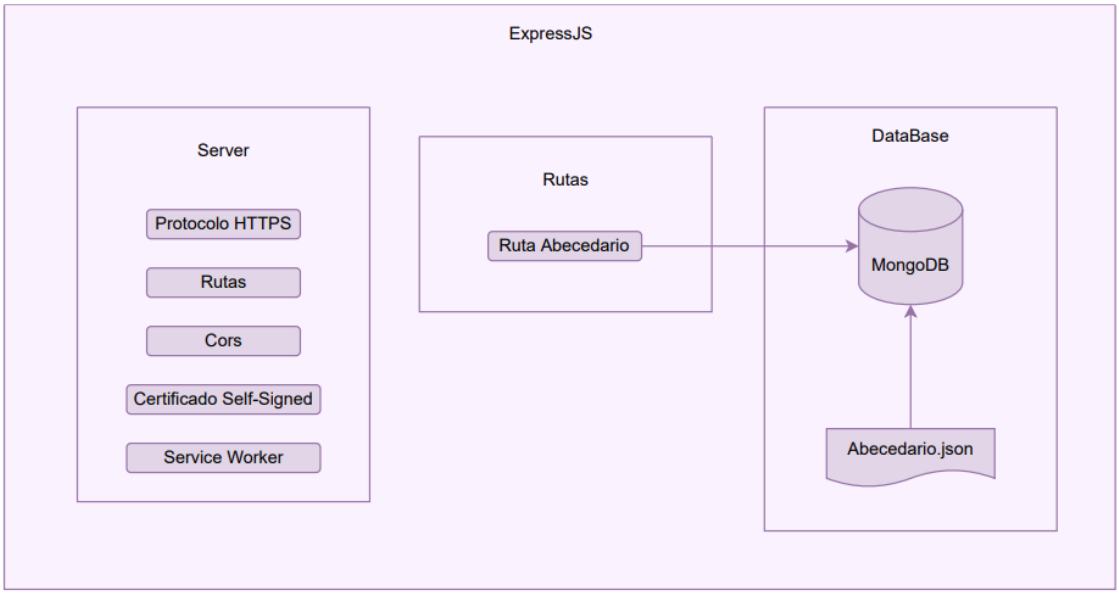

Imagen 21 - Esquema del desarrollo del Servidor

4.2.3. Especificación, normas de diseño y construcción

Durante el desarrollo de IABraille nos hemos encontrado con diferentes dificultades y numerosas limitaciones, algunas las hemos podido afrontar y otras han requerido de un "workaround" para poder seguir adelante con la idea.

En base al resultado final, comentaremos las limitaciones que éste contempla:

- **La necesidad de un protocolo seguro HTTPS:** Esto se debe a dos razones: para poder desarrollar con la tecnología de tensorflow; y para poder crear una PWA de la aplicación. Por la parte de tensorflow, este requisito se debe a que Google accede a la cámara del dispositivo del usuario, por lo que si no se trata de un sitio seguro la petición con los servicios no se realiza. Por parte de la PWA, es necesario porque usa un service worker, para asegurar que el contenido no pueda ser manipulado.
- **Flag WebRTC:** es una opción que suele estar activada en cualquier navegador por defecto, pero en case de que no lo esté es necesario activarla para que el Techeable Machine funcione. WebRTC se encarga de comunicar a tiempo real con el audio y cámara del dispositivo.

- **Certificado autofimardo:** para poder acceder a un protocolo seguro es necesario un certificado. Puede ser autofirmado, que es la opción que hemos utilizado. Sin embargo, el problema con el que nos hemos encontrado es que solo es posible desarrollar una PWA en un ordenador, desde dispositivos requiere otro tipo de certificados.
- **Modelos básicos:** los modelos entrenados con teachable machine, requieren muestras con buena iluminación y estándar: ni fotos muy oscuras ni muy iluminadas. También una buena calidad de imagen y un número de muestras por encima del cien imágenes.

4.2.4. Identificación de los subsistemas

En el caso de IABraille, todas las herramientas instaladas son necesarias para el próspero funcionamiento de ésta. No obstante, debido a la falta de tiempo dejamos una de las soluciones a mitad de desarrollo, por lo que se puede prescindir de ésta temporalmente. Estamos haciendo referencia al uso de la PWA.

Si prescindimos de su uso, la aplicación funciona de igual modo desde cualquier dispositivo que disponga de un navegador. Lo que nos ofrece la PWA es poder utilizar esa aplicación de forma offline haciendo un uso de ésta como si se tratara de una aplicación nativa. No obstante, no pudimos aplicar su función en nuestro dispositivo Android de prueba debido a problemas con el certificado que usábamos y a la falta de tiempo.

Por lo tanto, podríamos decir que se puede prescindir del archivo js que contiene el registro, el service worker y el manifest.json.

4.3. Revisión de casos de uso

4.3.1. Elección de alternativas de componentes y licencias adecuadas

Como ya se ha comentado anteriormente, existen otras opciones que también habrían sido correctas para el desarrollo de IABraille.

Por ejemplo, podríamos haber utilizado cualquier otro framework de javascript como Vue o Angular. También, en lugar de usar una base de datos local como mongodb podríamos haber usado su versión en la nube Mongo Atlas, o la api de IndexedDB.

En el caso del uso que quisiéramos darle a una aplicación Mongodb Atlas ofrece recursos de pago personalizables:

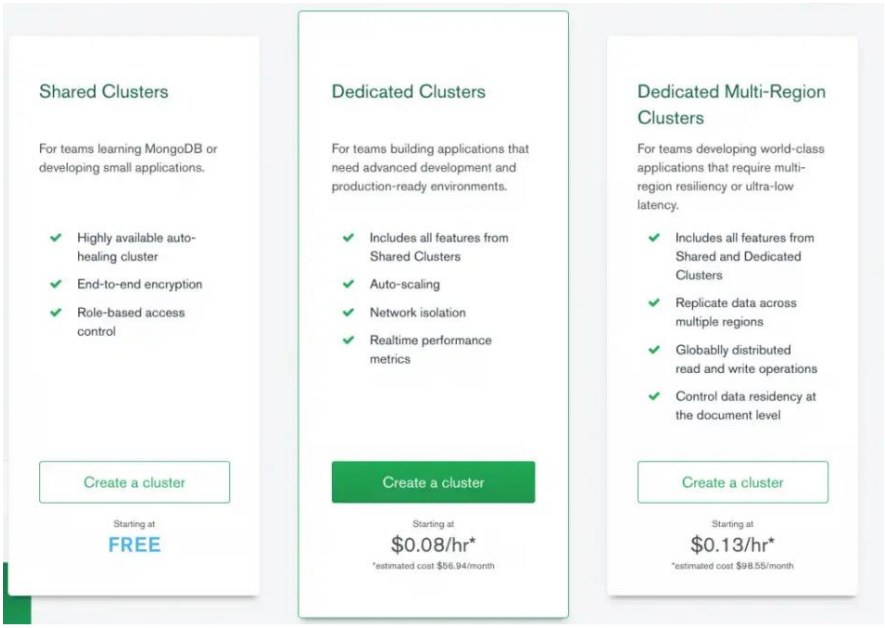

Imagen 22 - Precio servicio Mongo Atlas

En cuanto a las opciones que nos ofrecen en backend, cualquier plataforma nos sirve siempre y cuando se trate de un servidor con protocolo https. Como web service nosotros hemos usado la Api Restful, pero GraphQL también es una buena opción, sobre todo a la hora de trabajar con React.

Otras licencias que cabe tener en cuenta son a la hora de hacer pública nuestra aplicación. Si quisiéramos que su uso fuera exclusivo nativo nos interesa saber los precios que existen en las tiendas de aplicaciones del mercado:

El coste por crear una cuenta de desarrollador para Android es de 25$, mientras que una cuenta de desarrollar para la App Store de Apple asciende a: 99$ anualmente si eres un único desarrollado y 299$ si buscas la licencia de empresa.

4.3.2. Especificaciones de desarrollo y testing

Anteriormente habíamos hablado de las verificaciones que se tenían que llevar a cabo para proliferar en el desarrollo de la aplicación. A continuación, se especifica los pasos que se han seguido:

- Uso de Postman para verificar que las peticiones funcionen en el servidor.
- Pruebas en diferentes usuarios para comprobar que es una aplicación usable y de fácil entendimiento.
- Trabajo con diferentes tipos de muestras para el modelo entrenado de tensorflow.

4.3.3. Requisitos de implantación

A continuación, se muestran las versiones instaladas para el desarrollo de la aplicación:

DEPENDENCIA	VERSIÓN
ReactJs	18.0.0
React Icons	4.3.1
Api Router React	6.3.0
Vite Despliegue Aplicaciones	2.9.5

TailwindCss	3.0.24
NodeJs	16.13.1
TensorflowJs	1.3.1
Teachable Machine	0.8
Mui Material	5.6.4
Mui Material Icons	5.6.2
MongoDB	5.0.7
ExpressJs	4.16.1

Todas estas versiones son instalables gracias al comando de NPM una vez se haya instalado NodeJs (a excepción de los CDN de Teachable Machine).

A parte de las versiones, es importante que el servidor utilice el protocolo https para asegurar el funcionamiento de Tensorflow y la PWA.

4.4. Análisis y diseño del paradigma orientado a objetos

Realmente, al usar React, no podríamos decir que trabajo en base al paradigma orientado a objetos, más bien trabaja bajo el prisma del paradigma orientado a componentes. Éstos pueden estar compuestos por lenguaje jsx, el cual tiene la capacidad de utilizar código javascript entre elementos de html.

Para poder explicar la estructura dentro del paradigma, presentamos un UML en base a los componentes utilizados:

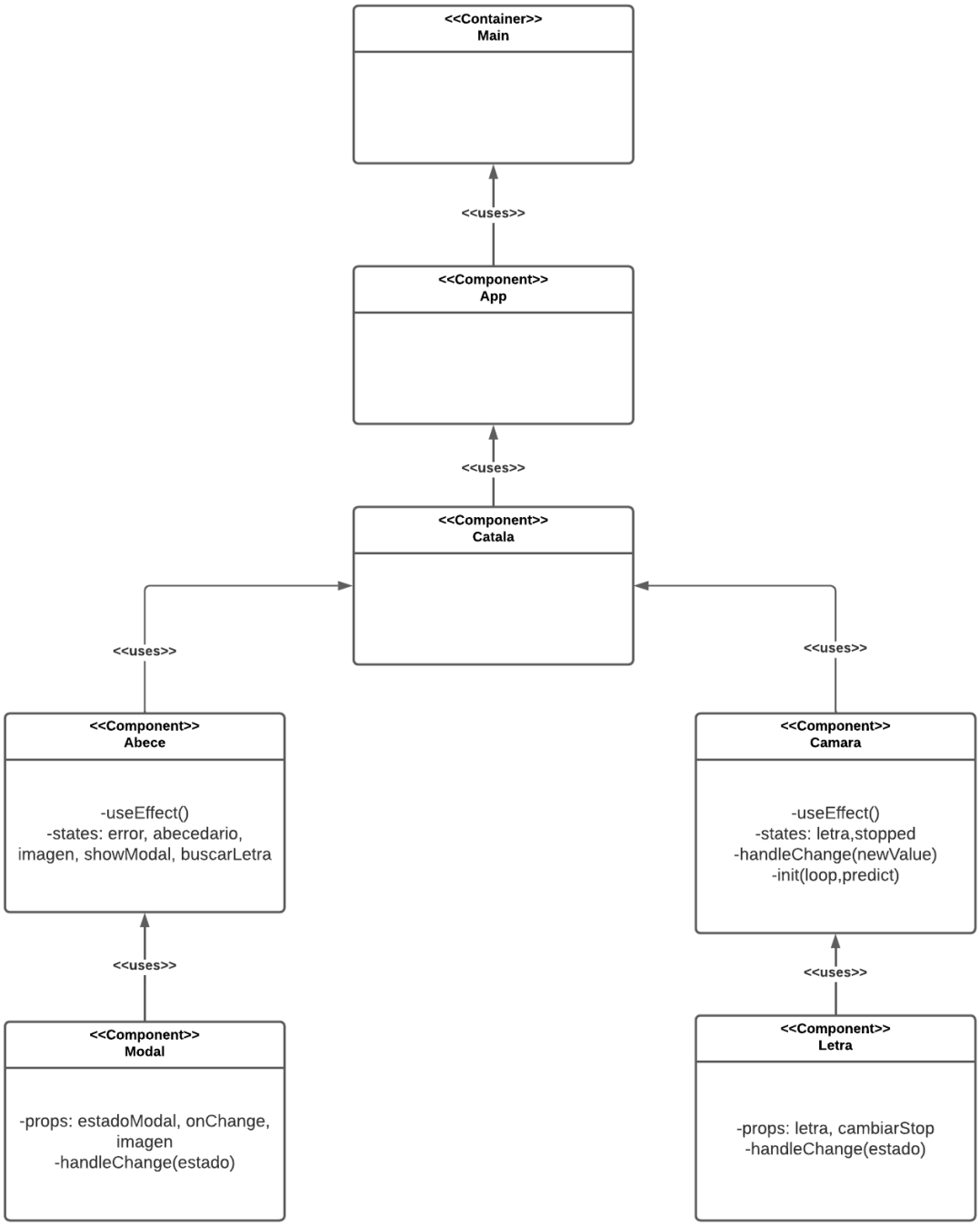

Imagen 23 - Uml de Componentes de React

4.5. Persistencia de datos: diseño base de datos

La base de datos con Mongo únicamente dispone de un documento “abecedario” con un id, un string de letra y un string la ruta de la fotografía. Éste se ha generador a partir de un archivo json.

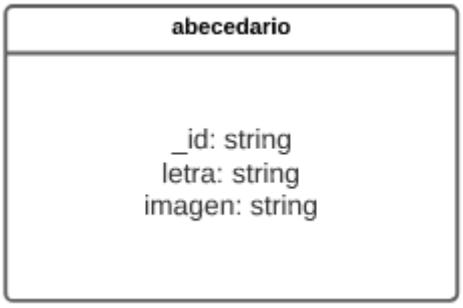

Imagen 24 - documento MongoDB

5. Desarrollo

5.1. Planificación de les actividades de desarrollo e integración del sistema

25 Descanso	26 Pruebas con TF y ReactNative	27 Pruebas con TF y ReactNative	28 Pruebas con TF y Flutter	29 Pruebas con TF y Flutter
2 Creación proyecto React con Vite	3 Información sobre PWA	4 Probar api cámara	5 Creación Servidor	6 Creación Backend
9 Desarrollo FrontEnd	10 Desarrollo FrontEnd	11 Desarrollo FrontEnd	12 Desarrollo FrontEnd	13 Creación modelos TF

16 Creación modelos TF	17 Set Interval – Cartulinas	18 PWA	19 Memoria	20 Memoria

5.2. Desarrollo

Ver Zip Adjuntado.

5.3. Documentación técnica del programa

Documentación en el código del proyecto. Más documentación de los lenguajes en página web oficial.

6. Implantación

6.1. Formación

Anteriormente ya se ha comentado que IABraille dispone de un diseño sencillo con el objetivo de buscar una correcta usabilidad. A continuación, explicaremos su funcionalidad en el siguiente manual de usuario:

- **Inicio**

Al iniciar IABraille, se encuentra el logo de la aplicación que es clickable y redirige a la siguiente vista.

Imagen 25 - Vista de Logo

- **Menú de selección**

Una vez clickado el logo, podemos ver dos opciones que serán nuestro menú de selección. La primera con un icono que da paso al abecedario y la segunda con un icono de una cámara que redirige a la función de escanear Braille.

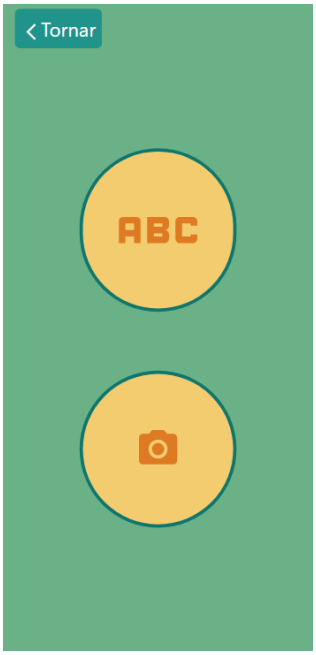

Imagen 26 - Vista Menú de Selección

- **Menú Abecedario**

Al acceder al menú de abecedario vemos una lista de todas las letras e signos que nos ofrece el alfabeto catalán. Al hacer lick en una de las fichas, aparece el equivalente en Braille de ésta. Podemos cerrar la ficha y salir del menú usando el botón de “Tancar” y “Tornar” respectivamente.

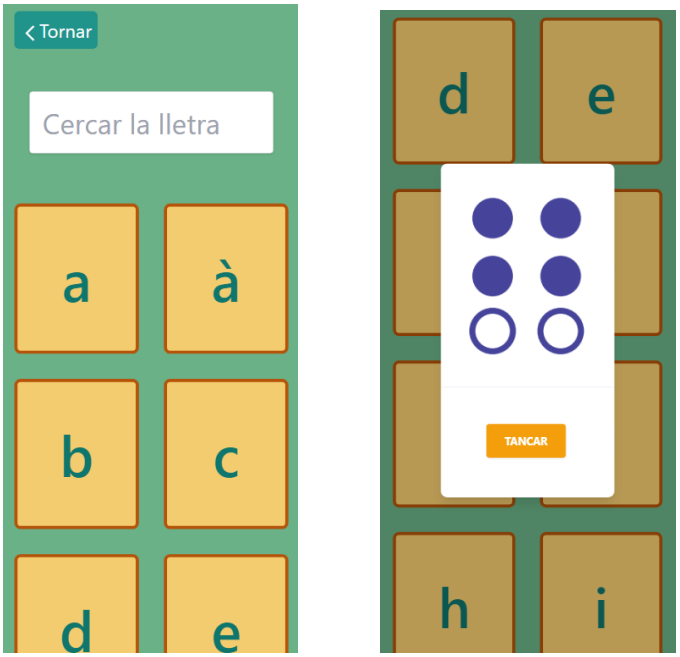

Imagen 27 - Menú de Abecedario

- **Menú Escáner Cámara**

Si se selecciona la opción de escáner, antes de mostrarlo la vista al completo, si es la primera vez que se accede, la aplicación solicitará permisos para poder acceder a la webcam o la cámara trasera del dispositivo.

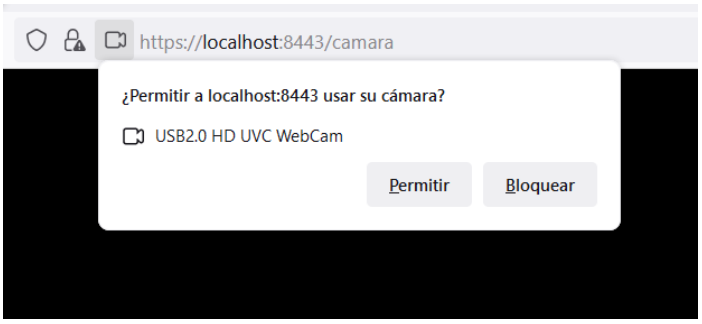

Imagen 28 - Permisos de acceso a la cámara

Al aceptar los permisos, aparecerá la vista de escáner con un recuadro que mostrará el visionado de la cámara. Tras 9 segundos de escaneo, aparecerá la traducción del braille al catalán.

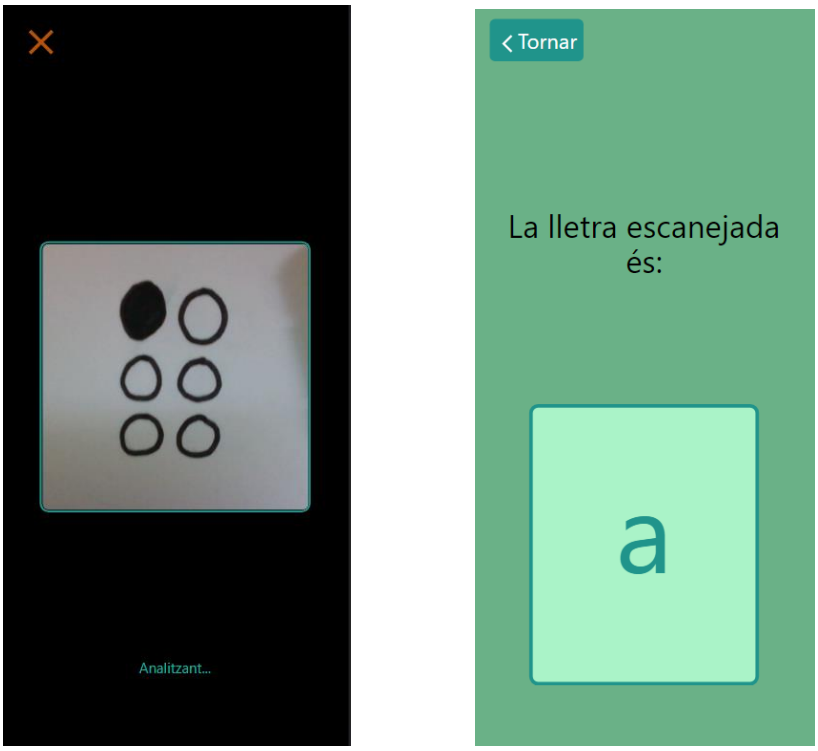

Imagen 29 - Vista Escáner y vista traducción

6.2. Implantación del sistema de pruebas

Tras realizar el testing de diferentes herramientas, encontramos algunas dificultades, algunas de ellas pudieron cumplirse mientras otras han terminado por abandonar el proyecto de IABraille.

Un ejemplo claro es el intento de integración como aplicación propiamente nativa. Se perdió mucho tiempo al inicio del proyecto intentando que la plataforma de Teachable Machine funcionara de forma nativa, así que realizamos pruebas con una tecnología Web y el resultado fue próspero. Al tratar con ese tipo de solución, pensamos que era buena idea intentar crear un WebView con React Native, pero al hacer pruebas encontramos que no era posible con el tipo de certificado que teníamos creado. Lo mismo ocurrió al intentar instalar la pwa en el Android.

No obstante, el mayor problema lo tuvimos a la hora de hacer pruebas con las muestras para el Techeable Machine. Probamos con diferentes modelos de braille, pero debido a la cámara del móvil, que no era de gran calidad, y que la aplicación no dispone de un foco de luz claro, la lectura de las muestras resultaba errónea. También intentamos aplicar modelos en base a fotografías en lugar de fotogramas de video, pero éste producía peores resultados.

Tratamos de mejorar los tiempos de análisis, pero descubrimos que se necesitaban más imágenes y que el modelo debía entrenarse más. Así que aislamos el obstáculo adjuntando fotogramas con una misma iluminación para todos, diferentes posiciones y focos.

6.3. Aceptación del sistema

IABraille se creó con el objetivo educativo de enseñar a personas de cualquier rango de edad a aprender Braille en catalán. Es el motivo por el que la aplicación dispone de un diseño sencillo y que con el material justo y necesario. En las pruebas que hemos realizado entre diversos usuarios han parecido entender su funcionamiento sin problema. En futuro mantenimiento, cuando la aplicación disponga de más funcionalidades se prevé hacer una remodelación, pero manteniendo su sencillez.

7. Mantenimiento y versiones futuras

El proyecto que hemos llevado a cabo consideramos que es una demo de una idea que puede abarcar más. Debido a la falta de tiempo hemos marcado los límites en sus dos funciones principales: escanear código y mostrar la transcripción al catalán. Consideramos el uso de otras APIS interesantes como transcriptor de voz o incorporar un motor de cámara a parte para poder hacer el foco y la luz más potente.

Sin embargo, nuestro punto de mejora se centrará en hacer que la aplicación pueda llegar a ser nativa e instalable a través de las diferentes plataformas y mercados existentes. También buscamos trabajar con modelos de casos reales como Braille en ascensores o en textos, en vez de muestras personales.

8. Bibliografía

8.1. Documentación oficial tecnologías

MongoDB

https://www.mongodb.com/docs/

Tailwind

https://tailwindcss.com/docs/installation

ReactJs

https://reactjs.org/docs/getting-started.html

TeachableMachine

https://teachablemachine.withgoogle.com/

Material Icons

https://mui.com/material-ui/material-icons/

ExpressJS

https://expressjs.com/

NodeJS

https://nodejs.org/es/docs/

github teachable Machine

https://github.com/googlecreativelab/teachablemachine-community/blob/master/libraries/image/README.md

8.2. Páginas web

¿Qué es una PWA?

https://digital55.com/que-es-pwa-ventajas-desventajas/

Express Api React Mongo

https://www.mongodb.com/languages/express-mongodb-rest-api-tutorial

Add Teachable Machine to nodejs Project

https://blog.sashido.io/how-to-load-a-teachable-machine-image-model-in-a-node-js-project/

React Native Webview

https://github.com/react-native-webview/react-native-webview