IABraille

Proyecto Final de Desarrollo de Aplicaciones Web

judith barrajón alcataz

Jesuïtes el Clot

DAW2

Índice

[1. Resumen inicial 2](#_Toc104070512)

[2. Estudio de viabilidad 2](#_Toc104070513)

[2.1. Establecimiento del alcance del sistema 2](#_Toc104070514)

[2.2. Estudio de la situación actual 2](#_Toc104070515)

[2.3. Definición de los requisitos del sistema 2](#_Toc104070516)

[2.4. Selección de la solución, estudio de las alternativas de solución y valoración de las alternativas. 3](#_Toc104070517)

# 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.

# 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.

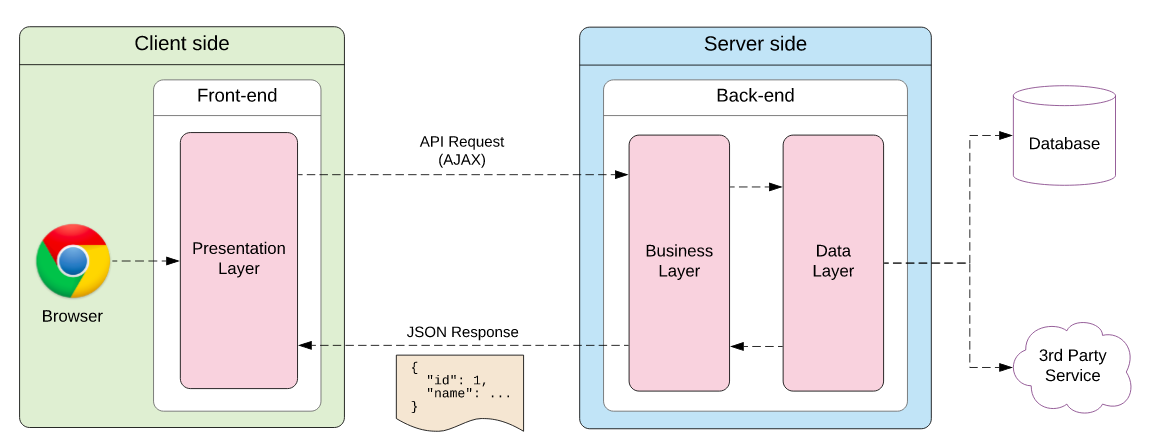


Imagen - Arquitectura SPA[[1]](#footnote-1)

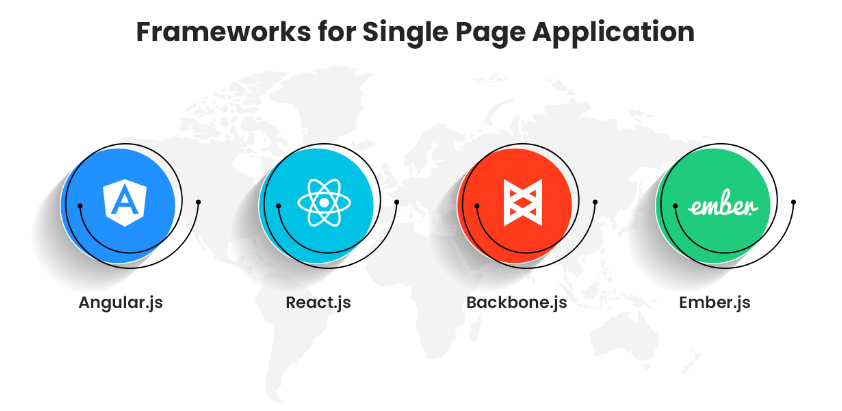


Imagen - Principales Frameworks[[2]](#footnote-2)

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.



Imagen - Soluciones Nativas[[3]](#footnote-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]](#footnote-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 que se planteó fue el uso de IndexedDB, no obstante, se descartó ya que no queríamos sobrecargar el cliente.



Imagen - 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.

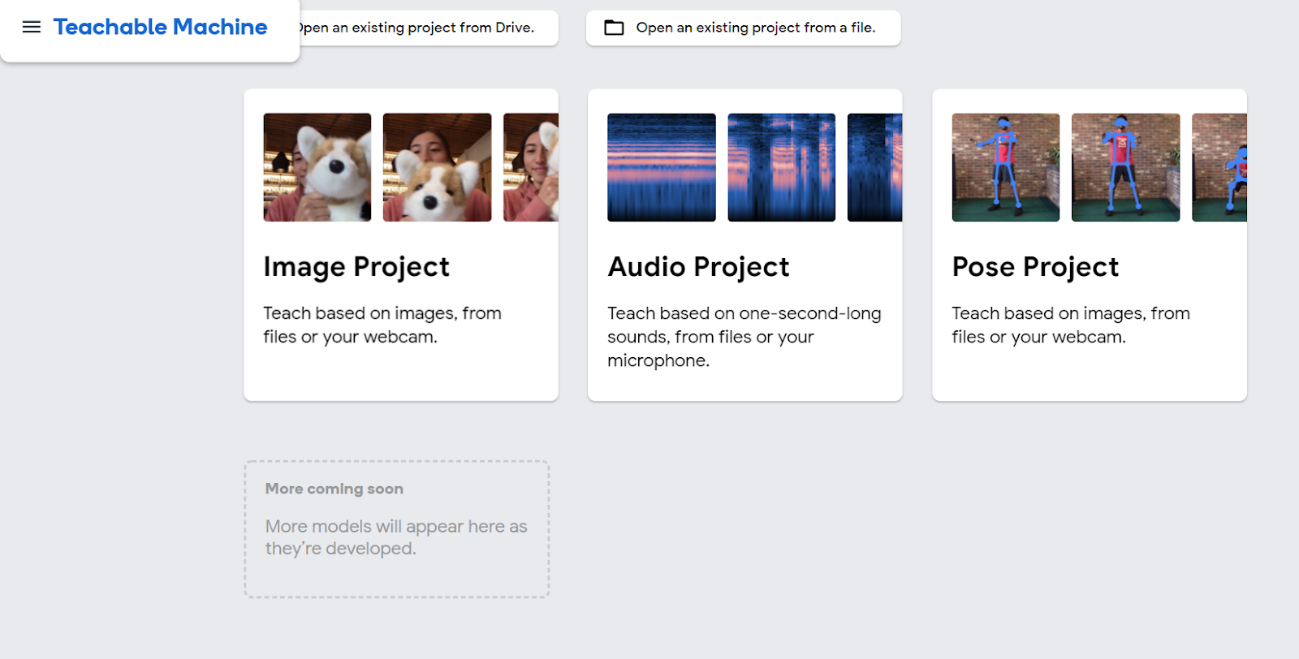


Imagen - Captura Teachable Machine

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 - Funcionalidades de las PWA[[5]](#footnote-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.

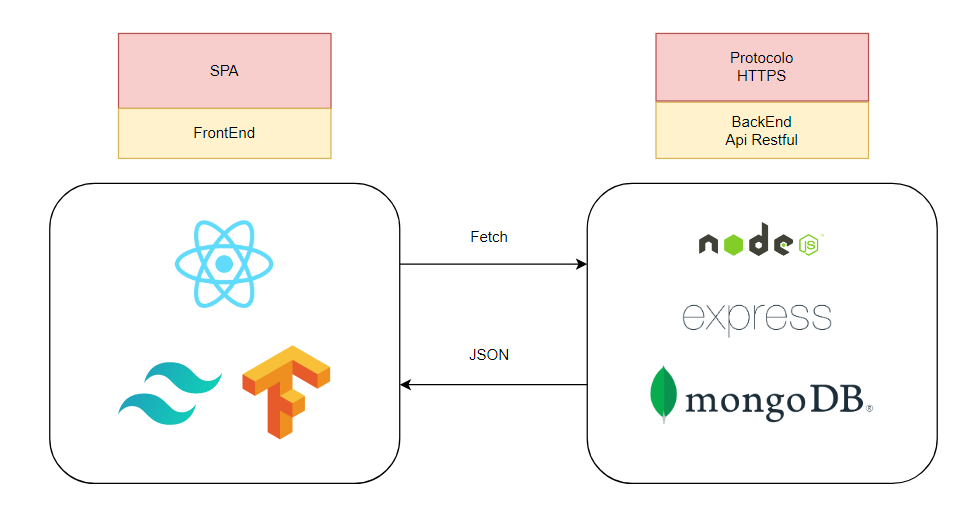
Más adelante entraremos en profundidad en cómo hemos desarrollado estas ideas, y de por qué no han resultado como se esperaba.

# 3. Análisis del sistema

## 3.1. Definición del sistema

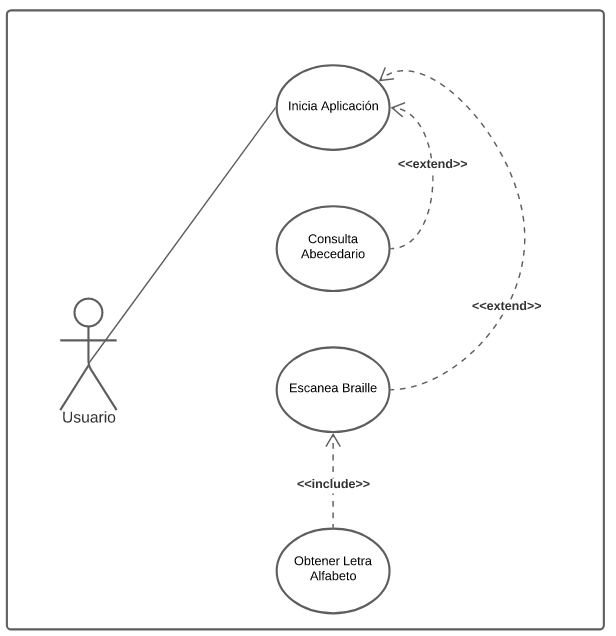
La aplicación se define de la siguiente forma:

* **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.



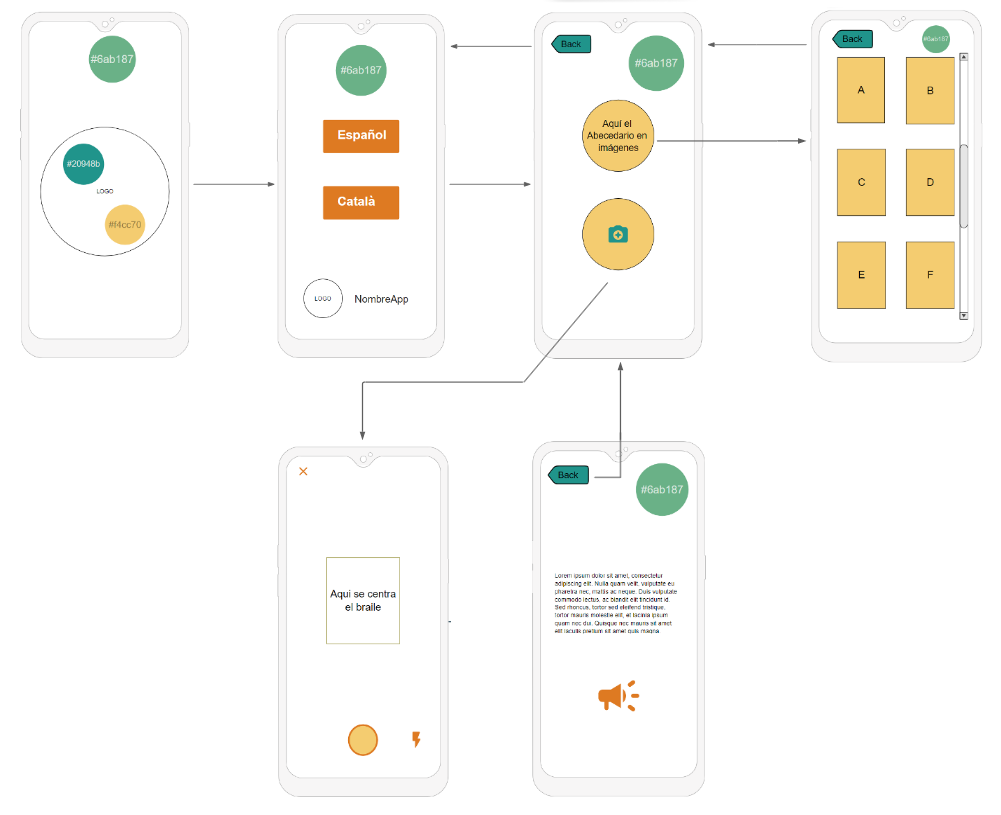
## 3.2. Establecimiento de los requisitos

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



## 3.3. Definición de la interfaz de usuario

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



## 3.4. Especificación del plan de pruebas

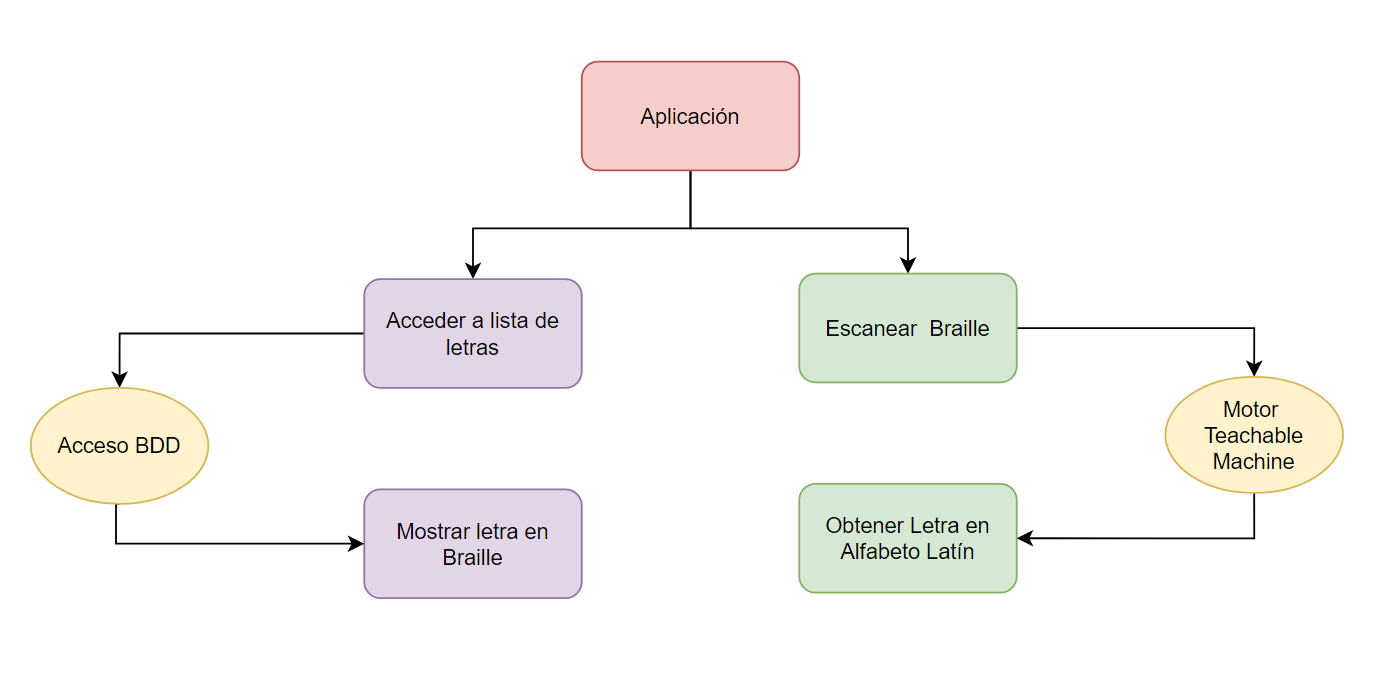
La aplicación dispone de un backend que hace consultas a una base de datos, por lo tanto, es importante comprobar que las peticiones de éste se cumplen.

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.



1. https://elmprogramming.com/what-is-a-single-page-app.html [↑](#footnote-ref-1)
2. https://www.angularjsindia.com/blog/top-single-page-application-frameworks-to-consider-for-web-apps/ [↑](#footnote-ref-2)
3. https://tekify.vn/flutter-react-nativie-which-framework-to-choose/ [↑](#footnote-ref-3)
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. [↑](#footnote-ref-4)
5. https://medium.com/react-adventure/introduccion-a-las-pwa-e578100b9015 [↑](#footnote-ref-5)