Proyecto Final de Ciclo

DESARROLLO DE APLICACIONES WEB

Modalidad bilingüe

I.E.S. CLARA DEL REY

Elena Pellín Carcelén — autora
2017 - 2019 — promoción

Tabla de contenido

Resumen

El objetivo del proyecto es desarrollar una aplicación web utilizando tecnologías muy punteras, como son Vue.js basado en componentes y Node.js utilizando javascript también del lado servidor. Además del empleo de metodologías y técnicas muy presentes en este sector como son Scrum como marco de trabajo y Design Thinking como guía para diseñar el producto resaltando la figura del usuario final.

Para ello he optado por la elaboración de una aplicación web en la que se exponen temas de debate donde los usuarios pueden votar si o no. Votaciones que irán acompañadas de una argumentación que a su vez podrá ser votada por el resto de usuarios. La aplicación es totalmente responsive y cumple con las normas de accesibilidad según W3C.

Palabras clave: Vue, Node, MongoDB, responsive, API, metodologías ágiles, experiencia de usuario.

Abstract

The aim of the project is to make a web application using very advanced technologies, such as Vue.js based on components and Node.js using javascript also on the server side. In addition to the use of methodologies and techniques very present in this sector such as Scrum as a framework and Design Thinking as a guide to design the product highlighting the figure of the end user.

In order to do so, we have chosen a web application in which topics of debate are presented where users can vote yes or no. Votes that will be accompanied by an argumentation that in turn will be able to be voted by the rest of users. The application is fully responsive and complies with accessibility standards according to W3C.

Keywordds: Vue, Node, MongoDB, responsive, API, agile methodologies, user experience.

Título

La idea de **shakeit** viene de la traducción al inglés de la famosa web de menéame https://www.meneame.net/ que permite que la gente debata sobre una idea, añadiendo comentarios y votando en contra o a favor. En este caso nuestro valor añadido es el uso de la tecnología moderna y puntera tanto en front (vue.js) como en back (nodejs y mongoDB). Además, en cuanto a la estética que se optado por un diseño simple haciendo uso del concepto de **mobile first**.

Introducción al proyecto

Esta es la documentación correspondiente al proyecto de fin de ciclo del Grado Superior de Desarrollo de Aplicaciones Web de la modalidad bilingüe del I.E.S. Clara del Rey, de la promoción 2017- 2019. El proyecto ha sido realizado entre los meses de abril y junio de 2018, siendo la coordinadora del mismo Rosa María Lanchas.

Se ha decidido hacer una aplicación dinámica que permita la interacción con el usuario de modo que pueda expresar su opinión sobre determinados debates que sean su interés. Para ello se ha utilizado el marco de trabajo Scrum, ideal para entornos de trabajo complejos en los que las necesidades del usuario pueden cambiar en cortos periodos de tiempo.

En los siguientes apartados se detallan las metodologías y tecnologías utilizadas, así como las funcionalidades que ofrece esta aplicación web.

El proyecto consiste en la elaboración de una aplicación web en la que se exponen temas de debate en los que los usuarios pueden votar si o no. Votaciones que irán acompañadas de una argumentación que a su vez podrá ser votada por el resto de usuarios. De modo que los argumentos más votados para ambas respuestas queden en lo alto de la página. Los debates podrán estar activos hasta una fecha tope si el administrador lo considera. La aplicación estará diseñada para que el usuario pueda interaccionar desde diferentes dispositivos, gracias a su comportamiento responsive (válida para iOS y android).

Objetivos

El objetivo principal del proyecto ha sido desarrollar una aplicación web con tecnologías muy punteras y demandadas por el mercado laboral actual. Disfrutando de todo el proceso de creación y aprendizaje, emulando un entorno de trabajo

profesional en el que supuestamente formo parte de un equipo. De ahí la elección del uso de ciertas metodologías y tecnologías que se desarrollan más adelante.

Para ello se ha optado por la elaboración de una aplicación web en la que se exponen temas de debate donde los usuarios pueden votar si o no. Votaciones que irán acompañadas de una argumentación que a su vez podrá ser votada por el resto de usuarios.

Además, la aplicación está diseñada para que el usuario pueda interaccionar desde diferentes dispositivos, gracias a su comportamiento responsive (válida para iOS y android), y cumpliendo con las normas de accesibilidad según W3C.

Cabe destacar que la elaboración de este trabajo se ha llevado a cabo compaginando la Formación en Centros de Trabajo y el trabajo en una consultora de analítica digital. Este entorno ha propiciado el descubrimiento de ciertas tecnologías y métodos de trabajo que han hecho posible el desarrollo del proyecto de la manera que se expone a continuación.

Herramientas y tecnologías utilizadas

Se han utilizado diversas herramientas y tecnologías que se han considerado las más idóneas para la realización de este proyecto.Se detallan a continuación:

Design thinking

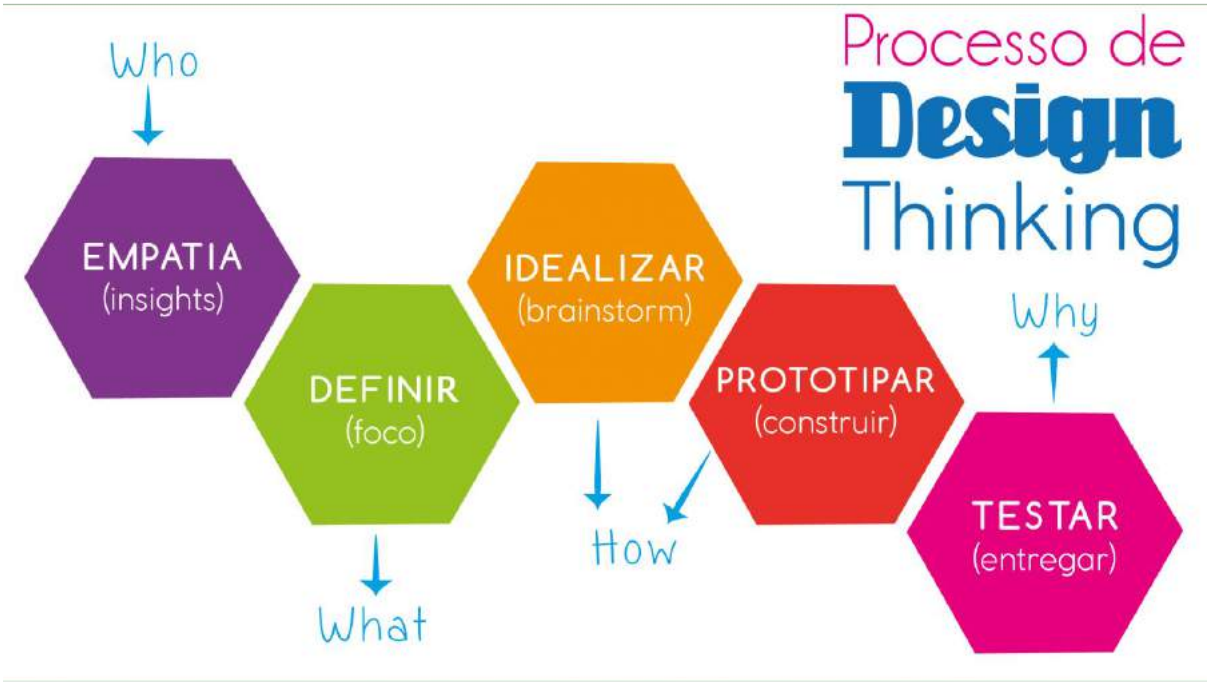
El Design Thinking es una metodología de creación de productos y servicios centrada en satisfacer de la mejor manera posible las necesidades de los usuarios haciéndolos parte activa del proceso de creación.

Ha sido desarrollado por diferentes investigadores, aunque principalmente se le atribuye a Tim Brown, profesor de la escuela de ingeniería de la Universidad de Stanford y CEO de la consultora IDEO, quien dio un gran impulso al Design Thinking en 2008.

De forma resumida, el proceso de desarrollo del producto o servicio abarca cinco fases, que se detallan a continuación:

- **Descubrimiento**: fase en la que se define el reto o problema que se intenta solucionar. Es necesario hacer una investigación y aprender de la observación de distintos conjuntos de individuos.

Caso de uso: No existe un espacio público en el que poder compartir opiniones y debatir sobre las metodologías ágiles.



- **Empatía**: se produce una evolución de los descubrimientos obtenidos en la fase anterior. Se organizan y analizan para obtener unos puntos de vista que contribuyan al desarrollo del producto o servicio. Se aprende de los individuos de manera directa, realizando encuestas y estudios de la web.

  Caso de uso: los "agilistas" *necesitan* las redes sociales *porque* quieren compartir sus opiniones y experiencias.

- **Ideación**: se realiza lluvia de ideas. Cualquier idea es buena, por muy descabellada que parezca. Se fomenta un entorno creativo que anima a la consecución de soluciones innovadoras.

  Caso de uso: Desarrollo de una aplicación web, que pueda ser accesible de manera cómoda desde cualquier dispositivo y permita la interacción entre multitud de usuarios de manera simultánea.

- **Prototipo**: se representa gráficamente el producto o servicio de manera muy rudimentaria. Consiste en crear un prototipo sin invertir mucho tiempo ni esfuerzo, pero que pueda aportar valor al usuario y pueda ser evaluado.

  Caso de uso: Imagen de la derecha.

- **Iteración**: el usuario interactúa con el prototipo de forma que se observa cómo interactúa este con el prototipo y se obtiene una retroalimentación que permite introducir mejoras para ir perfeccionando el producto hasta satisfacer de manera plena las necesidades del usuario final.

Caso de uso: el usuario no usa algunas funcionalidades dependiendo del dispositivo del dispositivo y echa en falta otras.

Scrum

Scrum es un marco de trabajo para el desarrollo de productos en entornos cambiantes y complejos creado por Jeff Sutherland y Ken Schwaber en 1995. Se basa en el empirismo, animando a la toma de decisiones y acciones basadas en la experiencia y no en suposiciones.

Los pilares básicos de Scrum son la transparencia, la inspección y la adaptación. Estos necesitan de manera indispensable que exista un entorno de confianza. Como valores principales figuran el compromiso, coraje, foco, apertura y respeto. Los miembros del equipo tienen coraje para hacer bien las cosas y para trabajar en los problemas difíciles. Todos se enfocan hacia las metas del equipo. Estos y las partes interesadas acuerdan estar abiertos a todo el trabajo y a los desafíos que se les puedan presentar. Los miembros del equipo se respetan entre sí para ser personas capaces e independientes.

El marco de trabajo Scrum establece una serie de roles, eventos en los que no se va a profundizar en este proyecto, que marcan unas pautas y herramientas para conseguir ejecutarlo correctamente. Sin embargo, Scrum dispone de tres artefactos que sí resulta útil describir un poco más. En este caso artefacto hace referencia a elementos físicos que aparecen gracias a la aplicación de Scrum. Estos artefactos son el product backlog, el sprint backlog y el incremento.

- **Product backlog**: es un listado de tareas, funcionalidades, y demás aspectos que son necesarios llevar a cabo para la elaboración del producto. Listado que está ordenado según las prioridades de los elementos que lo componen y que permite tener una visión global e ir avanzando evitando posibles bloqueos.
- **Sprint backlog**: el proceso de creación de un producto puede durar mucho tiempo. En Scrum este proceso se divide en pequeñas etapas llamadas sprints que permiten reducir la complejidad y poder adaptarse a los cambios. El sprint backlog es un listado que se obtiene a partir del product backlog, pero que a diferencia de este último hace referencia solo a la duración de un sprint y no de todo el proyecto.

- **Incremento**: es el resultado del Sprint. Es una pieza de Software, acorde con los elementos seleccionados del Sprint Backlog que aporta un valor de negocio al producto que se está desarrollando.

Product backlog inicial de Shake it:

- hola mundo: vuejs - nodejs - mongoDB
- crear un debate
- modificar un debate
- poder realizar un voto a favor de un debate
- poder realizar un voto en contra de un debate
- añadir un argumento
- modificar un argumento
- poder hacer un voto en contra de un argumento
- poder hacer un voto a favor de un argumento
- Añadir un header con el logo de la aplicacion
- Añadir un footer de la aplicacion con las redes sociales

Git

Git es un sistema de control de versiones diseñado por Linus Torvalds. Git tiene tres estados principales en los que se pueden encontrar los archivos: confirmado, modificado y preparado. Confirmado significa que los datos están almacenados de manera segura en la base de datos local. Modificado significa que se ha modificado el archivo pero todavía no se ha confirmado a la base de datos. Preparado significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.

Esta herramienta permite que varias personas trabajen en el mismo proyecto a la vez gestionando los posibles conflictos que pueden surgir a nivel de código. Además, permite recuperar versiones anteriores y tener una copia de seguridad online de los archivos, que protege al desarrollador de pérdidas de datos por avería o fallo del ordenador desde el que se trabaja.

El repositorio de este proyecto se puede consultar en la URL: https://github.com/epellin/Shakeit-v2

Sourcetree

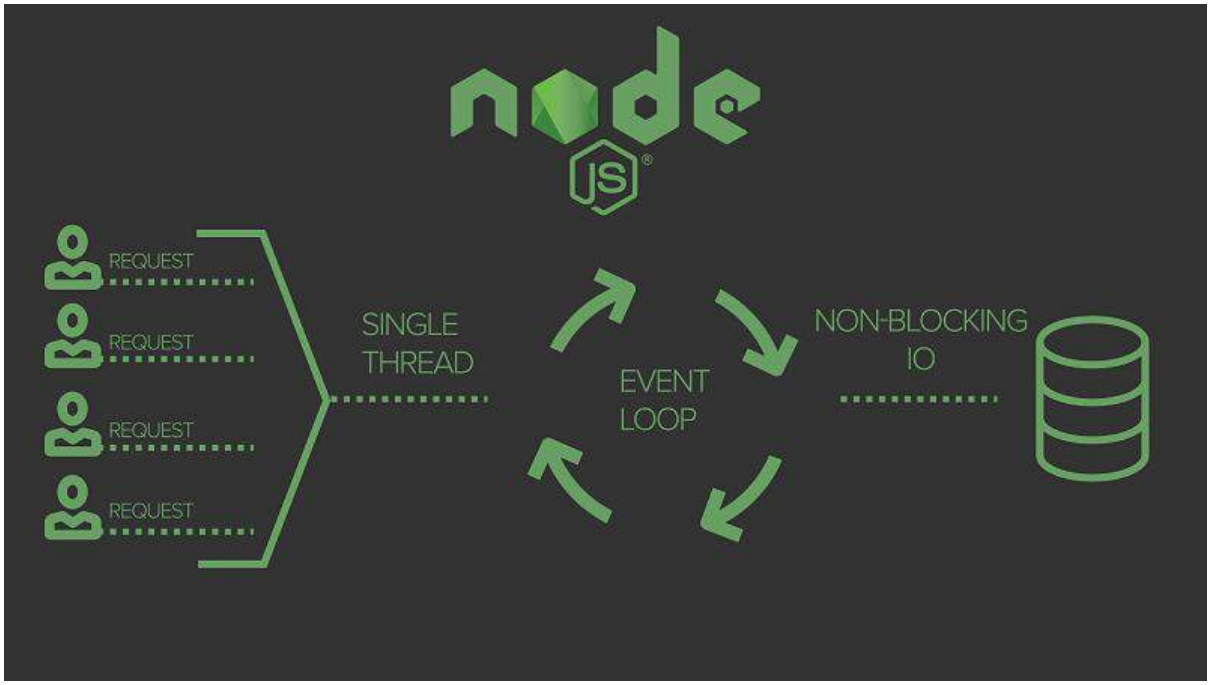
Sourcetree es una herramienta que simplifica la forma de interactuar con los repositorios de Git. Consiste en una interfaz gráfica que permite realizar acciones con git evitando el uso del terminal para ejecutar los comandos.

Node.js

Node.js es una librería y entorno Javascript del lado del servidor, basado en eventos. Funciona de manera asíncrona. Node ejecuta javascript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. Esto permite a Node proporcionar un entorno de ejecución del lado del servidor que compila y ejecuta javascript a velocidades increíbles. El aumento de velocidad es importante debido a que V8 compila Javascript en código de máquina nativo, en lugar de interpretarlo o ejecutarlo como bytecode. NodeJS es un código abierto de JavaScript. Nació en el 2009, y se ejecuta en Mac OS X, Windows y Linux.

El objetivo principal de Node es la escalabilidad. La gran mayoría de tecnologías que trabajan desde el lado del servidor accionan las peticiones de forma aislada y mediante hilos independientes. Por eso, cuando la cantidad de solicitudes aumenta, el consumo de los recursos también lo hace y todo se ralentiza. La cantidad de solicitudes así como los procesos entrantes y salientes se convierten en uno de los factores limitantes y NodeJS trata de optimizarlo.

NodeJS se diseñó para poder soportar una gran cantidad de conexiones simultáneas a un servidor, empleando un único hilo y un bucle de eventos asíncrono. De modo que las nuevas peticiones son tratadas como eventos en este bucle.



Esto permite que NodeJS sea capaz de gestionar múltiples conexiones y peticiones de forma muy eficiente, lo que lo hace apropiado para desarrollo y aplicaciones con un gran número de conexiones simultáneas. Por el contrario, en general, NodeJS no resulta adecuado en aplicaciones que requieran un número reducido de conexiones con un gran consumo de recursos (aplicaciones de cálculo, acceso intensivo a datos, etc).

Por estos motivos se ha considerado que Node es idóneo para Shake it, ya que la aplicación no requiere un gran consumo de recursos, pero podría darse el caso de que tuviera muchas peticiones simultáneas si hubiera un elevado número de usuarios interactuando con la aplicación a la vez.

Vue.js

Vue es un marco progresivo para construir interfaces de usuario. Fue creado como proyecto personal por Evan You, antiguo desarrollador de Google, en un intento de simplificar el funcionamiento de AngularJS.

A diferencia de otros marcos monolíticos, Vue está diseñado desde cero para ser adoptado de forma incremental. La biblioteca principal se centra únicamente en la capa de vista, y es fácil de recoger e integrar con otras bibliotecas o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de alimentar sofisticadas aplicaciones de una sola página (SPA) cuando se utiliza en combinación con herramientas modernas y bibliotecas de soporte.

Vue se caracteriza principalmente por ser :

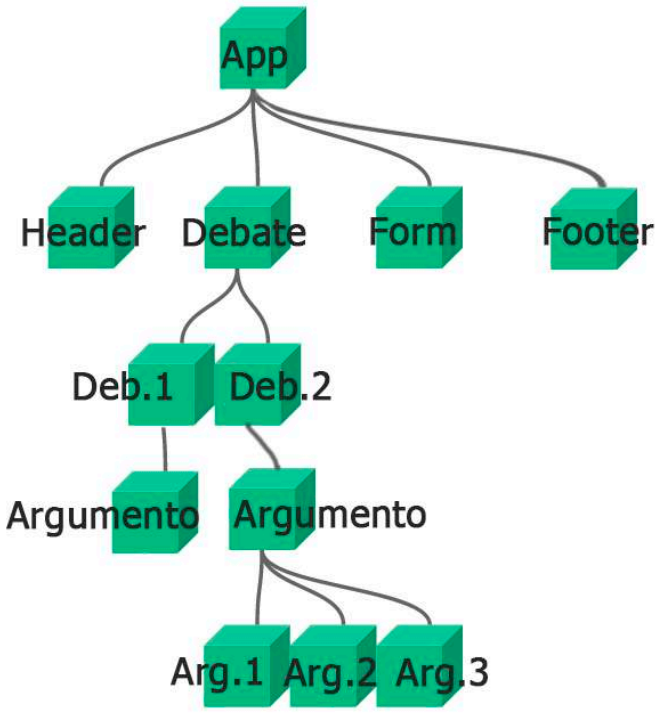
- **Versátil**: Su núcleo es bastante pequeño y se escala a través de plugins, con lo cual puede parecer que Vue es una librería muy parecida a React, una librería que cumple un propósito.
- **Reactivo**: VueJS sabe comunicarse muy bien por medio de eventos asíncronos.
- **Optimizado**: Su core ocupa 74KB.

El core (núcleo) principal de Vue está formado por una librería encargada de renderizar vistas en el navegador. Su forma de organizar el código es por medio de pequeños componentes que contienen todo el HTML, CSS y JavaScript necesario para funcionar como pieza independiente. Estas piezas se van incorporando en un árbol jerárquico de componentes hasta formar nuestra aplicación. Usar esta librería es tan fácil como importar el script en nuestra página HTML.

Por todo esto se ha considerado que Vue es la mejor opción para desarrollar Shake it, ya que su fórmula basada en componentes, simplifica mucho la cantidad de

líneas de código necesarias para desarrollarla. Una vez bien desarrollada la aplicación, ella sola va añadiendo más contenido si aumentan los datos en la base de datos.

La composición de componentes para este proyecto es la siguiente:



- **PellinMenu**: contiene la parte de la cabecera.
- **DebatePost**: que renderiza los debates que están almacenados en la base de datos, y que a su vez contiene el componente hijo de DebateArgumentPost.
- **DebateArgumentPost**: renderiza los argumentos de cada debate, y está incluido en el componente DebatePost, de ahí que se denomine componente hijo.
- **ArgumentForm**: se utiliza para añadir argumentos a la base de datos.
- **PellinFooter**: contiene la parte del pié de la página web.

HTML

HTML5 es un lenguaje de marcas (Hyper Text Markup Language) usado para estructurar y presentar el contenido para la web. Es la quinta revisión del estándar que fue creado en 1990. Hace unos años la W3C lo recomendó para transformarse en el estándar a utilizar para los proyectos de ahí en adelante.

Con HTML5, los navegadores como Firefox, Chrome, Explorer, Safari y más pueden saber cómo mostrar una determinada página web, saber dónde están los elementos, dónde poner las imágenes, dónde ubicar el texto. La diferencia principal es el nivel de sofisticación del código que podremos construir usando HTML5.

Se incorporan una serie de elementos y atributos nuevos, a través de las nuevas etiquetas semánticas, como son header, footer, nav, article, section, audio, video, embed y canvas.

A continuación se muestra un fragmento de código del proyecto correspondiente al componente de la cabecera de la web:

```
<template>
  <header class="fluid-container w-100 m-0">
    <div class="row">
      <section id="identificacion" class="col-10 col-sm-6 col-md-4 py-4 text-left">
        <img id='logo' src="../assets/images/logo.png" alt='logo shake it. Dos bocadillos de conversacion contrapuestos.'/>
        <h2>{{web_info.current_page}}</h2>
      </section>
      <nav class="col-2 d-md-none offset-sm-4 w-100">
          <a id='menu_movil' href="#" ><img src="../assets/images/hamburguesa.png" alt='icono menu desplegable para móviles' class="w-100 pr-1 pt-4
          pt-sm-2"/></a>
        <section id='nav_oculto' class='col-2 offset-10 px-0'>
          <ul>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/home.png" alt='submenu Home'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/proyectos.png" alt='submenu Proyectos'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/persona.png" alt='submenu Sobre mi'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/contacto.png" alt='submenu Contactame'/></li>
          </ul>
        </section>
      </nav>
      <nav id='menu_tablet' class="col-12 col-md-6 d-none d-md-inline-block pt-4 text-right offset-1">
        <a v-for="section in sections" :key="section.id" :href="section.link" :title="section.name" class="row m-0 p-2">{{section.name}}</a>
      </nav>
    </div>
  </header>
</template>
```

CSS

CSS (Cascading Style Sheets) u hojas de estilo en cascada es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML. La información de estilo puede ser adjuntada como un documento separado o en el mismo documento HTML. En este caso en el que se utiliza Vue.js, la documentación oficial recomienda utilizar un archivo por componente, y en ese archivo incluir la parte HTML, javascript y CSS específica de dicho componente y así evitar dependencias.

CSS3 funciona mediante módulos. Algunos de los más comunes son colors, fonts, backgrounds, borders, etc. Los módulos son categorías en las que se pueden dividir las modificaciones que se hacen del aspecto del sitio web.

Se ha procurado reducir al máximo las líneas de código de CSS gracias a la utilización de Bootstrap. El principal uso que se ha hecho del CSS en este proyecto reside en la asignación de colores y tipos de letra a los distintos elementos que lo

componen. Así como el uso de animaciones y media queries para garantizar un comportamiento responsive de la aplicación, allí donde Bootstrap no llega.

A continuación se muestra una porción de código del estilo correspondiente al componente hijo de argumentos de debate:

```
/*Animaciones */
@-webkit-keyframes heartBeat {
0% { -webkit-transform: scale(1); transform: scale(1);}
14% { -webkit-transform: scale(1.3); transform: scale(1.3);}
28% { -webkit-transform: scale(1); transform: scale(1);}
42% {-webkit-transform: scale(1.3); transform: scale(1.3);}
70% {-webkit-transform: scale(1); transform: scale(1);}
}

@keyframes heartBeat {
  0% { -webkit-transform: scale(1); transform: scale(1);}
  14% { -webkit-transform: scale(1.3); transform: scale(1.3);}
  28% { -webkit-transform: scale(1); transform: scale(1);}
  42% { -webkit-transform: scale(1.3); transform: scale(1.3);}
  70% { -webkit-transform: scale(1); transform: scale(1);}
}

/*Media query*/
@media screen and (min-width: 768px) {
  #likeBox { border-right: .5em solid #ffffff;}
  #dislikeBox { border-left: .5em solid #ffffff;}
}
@media screen and (min-width: 1200px) {
  .argument { width: 85%;}
}
```

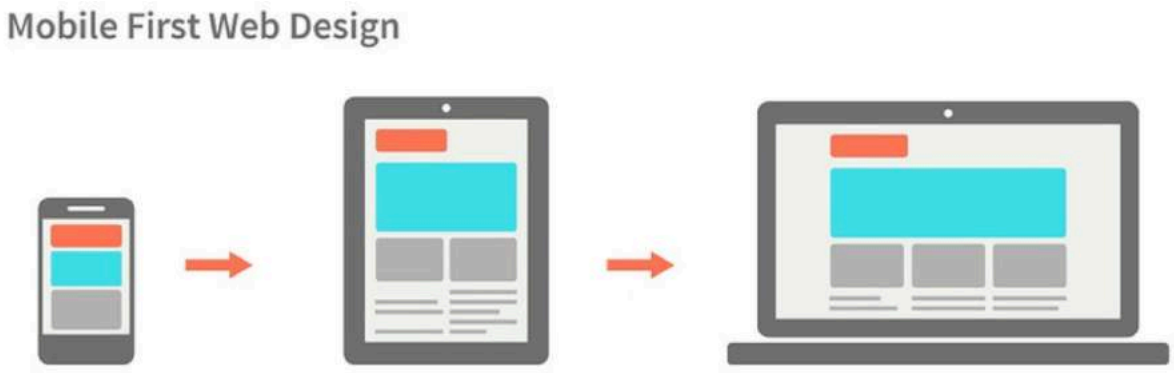
Bootstrap

Bootstrap es un framework específico inicialmente creado como una solución interna de twitter, que posteriormente fue puesto a disposición del público en 2011 como proyecto Open Source.

La finalidad de este framework es facilitar el desarrollo de aplicaciones web responsive a través de un sistema de rejillas, en el que imaginariamente la pantalla se divide en 12 columnas.

Bootstrap ofrece las herramientas para que un sitio web se vea bien en toda clase de dispositivos, ahorrando así el trabajo de tener que rediseñar un sitio web para

cada tipo de dispositivo. Apoya la modalidad de diseño Mobile First que consiste en comenzar un proyecto de diseño por la pantalla más pequeña e ir adaptándolo posteriormente a las más grandes.



Para poder utilizar Bootstrap en este proyecto ha bastado con incluir los enlaces a la CDN en el head del fichero index.html y utilizar las clases, que este framework pone a nuestra disposición, de manera conveniente. En la siguiente imagen se muestra un fragmento de código del componente correspondiente al footer en el que se observa cómo se adaptan los diferentes elementos que lo componen según el dispositivo que utilice el usuario. Por ejemplo, los enlaces a las redes sociales del footer aparecen en un orden distinto para dispositivos móviles que para el resto.

```
<footer class="fluid-container w-100 px-3 mt-5">
  <div class="row">
  <section class="col-10 col-md-3 col-lg-3 order-1 pt-4 order-1">
    <h3 class="row m-0">{{web_info.title}}</h3>
    <p class="row m-0">{{web_info.current_page}}</p>
  </section>
  <nav class="col-12 col-sm-9 col-md-6 col-lg-6 order-3 order-sm-2 pt-4 text-center text-md-right">
    <a v-for="section in sections" :key="section.id" :href="section.link" class="row m-auto p-2" :title="section.name">{{section.name}}</a>
  </nav>
  <section id="rrss" class="d-flex col-2 col-sm-3 col-md-3 col-lg-2 order-2 order-sm-3 offset-lg-1 py-4 justify-content-end ">
    <a href="https://twitter.com/E_Pellin" title='Haz click para ir al twitter de epellin' class="col-auto my-0 mx-1 p-1"><img class="img-fluid" src="
    ../assets/images/twitter_white.png" alt="icono twitter"></a>
    <a href="https://www.linkedin.com/in/epellin/" title='Haz click para ir al linkedin de epellin' class="col-auto my-0 mx-1 p-1"><img class="img-fluid
    " src="../assets/images/linkedin_white.png" alt="icono Linkedin"></a>
  </section>
  </div>
</footer>
```

MongoDB

MongoDB es un sistema de base de datos NoSQL opensource, escrito en C++, escalable y de alto rendimiento. Se desarrolló en 2007 por 10gen Inc.

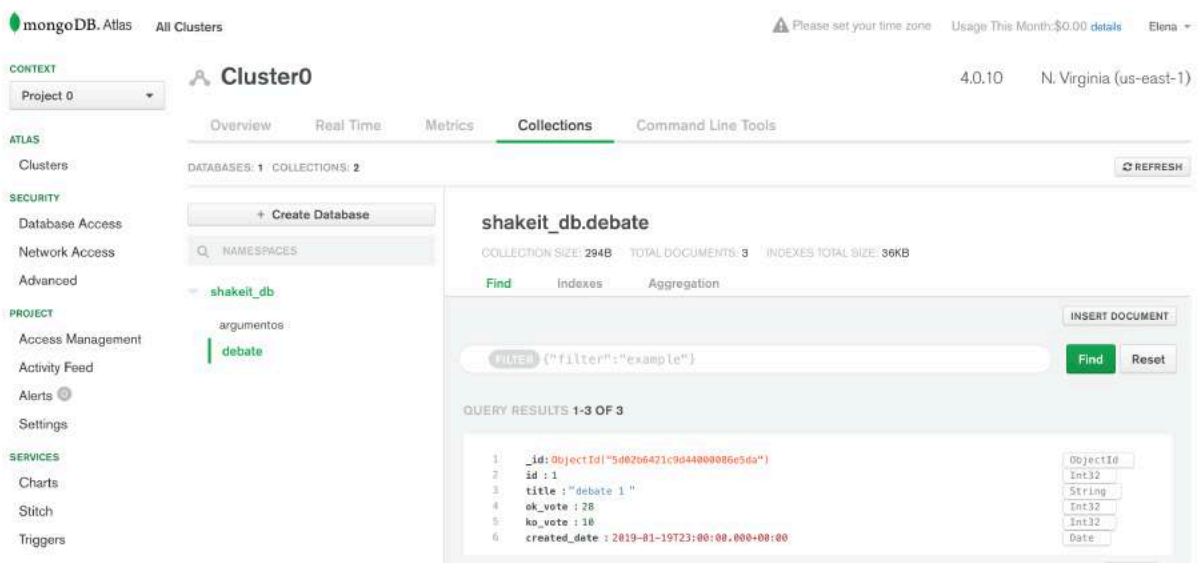
En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. Además, los documentos nos permiten nuevas estructuras como arrays o subdocumentos que permitirán que de

una sola consulta se recupere toda la información y evite así la necesidad de ejecutar consultas de tipo join.

Las principales características de MongoDB son su alto rendimiento, alta disponibilidad y escalado automático.

Inicialmente se iba a utilizar MySQL en vez de MongoDB para este proyecto, pero tras un periodo de investigación y pruebas se ha decidido utilizar MongoDB, porque está mejor adaptado a las necesidades de Vue y Node. Para pasar la información a la aplicación utilizando el framework Vue se utilizan ficheros JSON que facilita MongoDB a través de Node. Una vez obtenido el JSON con los datos Vue puede renderizar los mismos tantas veces como sea necesario gracias al sistema de componentes. A nivel de código parece que se ha escrito únicamente un debate, pero a nivel visual se observa que se han creado tantos debates como contiene el JSON.

En la siguiente imagen se muestra la interfaz de MongoDB en la que se pueden observar las colecciones que se han creado para este proyecto.



Bases de datos

Como hemos mencionado anteriormente, MongoDB es una base de datos NoSQL, que trabaja con colecciones.

En el caso de la colección de **debate** la información que se guarda aparece de la siguiente manera:

```
_id: ObjectId("5d02b6421c9d44000086e5da")
id: 1
title: "El Scrum Master es el antiguo Project Manager"
ok_vote: 30
ko_vote: 10
created_date: 2019-01-19T23:00:00.000+00:00
```

_id: es un identificador propio de mongoDB que genera para cada registro nuevo en el sistema.
id: es el identificador único del debate.
title: el título del debate.
ok_vote: número de votos a favor de un debate.
ko_vote: número de votos en contra de un debate.
created_date: fecha de creación del debate.

Por otro lado, la información que se guarda en la colección de argumentos se puede observar como sigue:

```
_id: ObjectId("5d002202a20c1755042ba9b1")
id: 1
title: "Un Scrum Master hace un poco de Project Management"
description: "al principio, el Scrum Master debe de hacer project manager ya que el ..."
ok_vote: 117
ko_vote: 2002
post_id: 1
created_date: 2018-12-31T23:00:00.000+00:00
type: "like"
```

_id: es un identificador propio de mongoDB. Genera un identificador nuevo para cada registro. en este caso el sistema, en este caso para argumentos.
id: es el identificador único del argumento.
title: el título del debate.
ok_vote: número de votos a favor de un argumento.
ko_vote: número de votos en contra de un argumento.
created_date: fecha de creación del argumento.
post_id: identificador del debate; así reflejamos la relación que hay entre un argumento y un debate
description: descripción del argumento.
type: el tipo del voto, si es a favor o en contra.

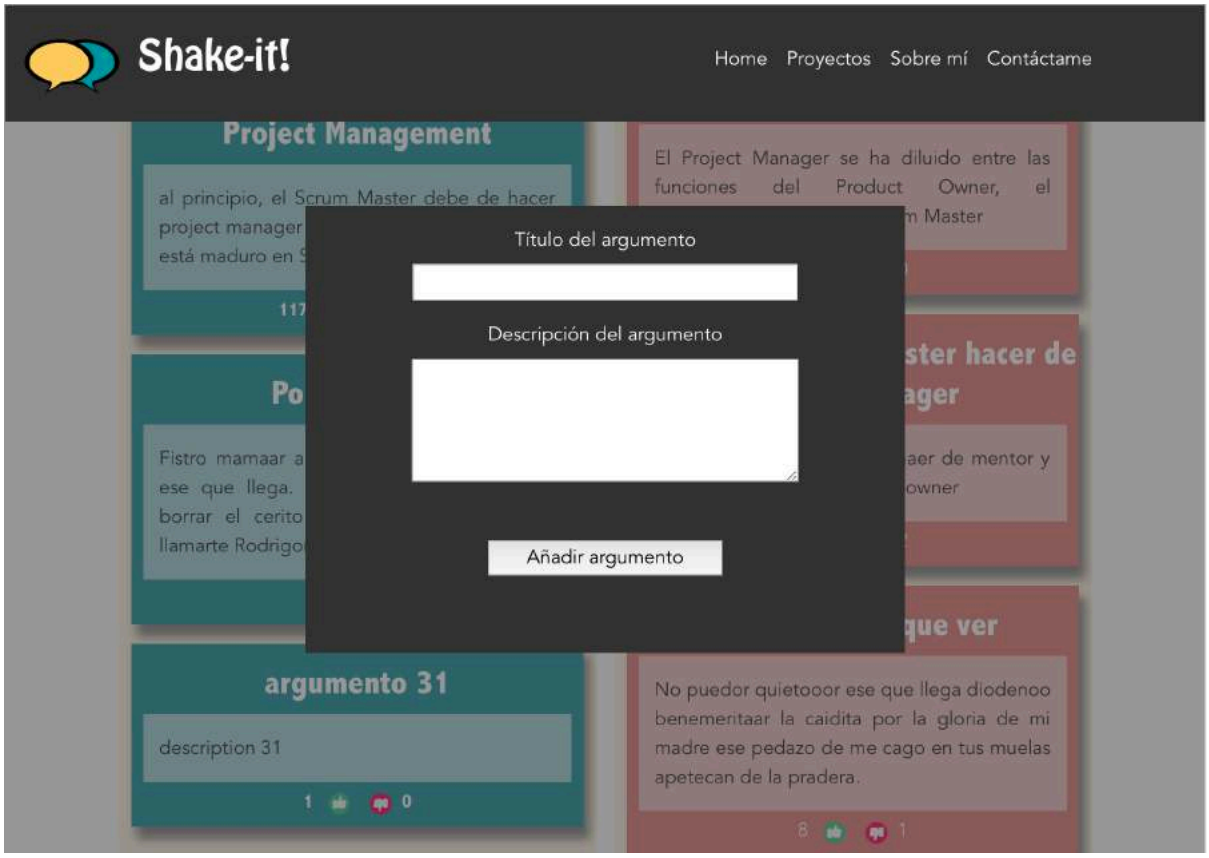
Prototipo

El prototipo obtenido a través del proceso de Design Thinking descrito en un apartado anterior se describe a continuación.

La **página principal** de Shake it será tal y como muestra la imagen. La cabecera con el menú de navegación sólo se quedará fija en pantalla para dispositivos grandes, en los que se considera que hay suficiente espacio como para que esta no resulte molesta al usuario. En la medida de lo posible, en todas las páginas se intentará hacer scroll, para visualizar el footer en la pantalla.

Cada debate está compuesto por una cabecera en la que figura el título y los votos a favor y en contra. Seguido de los argumentos que apoyan cada una de las dos posturas divididos en dos columnas. Cada argumento se representa como un elemento independiente que permite diferenciarlo del resto a simple vista.

Cuando se quiere añadir un argumento, se muestra un componente que a priori permanece oculto hasta que se haga click en el botón de Agregar argumento. El resultado es el siguiente:



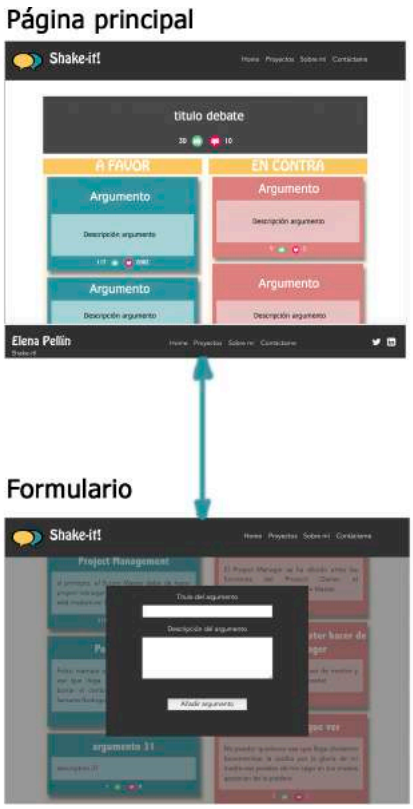
A la derecha se muestran unas imágenes de cómo se visualizan algunos elementos de las pantallas anteriores en diferentes dispositivos.

En dispositivos móviles, el navegador lateral se reduce a un botón tipo hamburguesa que sirve para desplegar el menú cuando se hace click sobre él.

Muchos elementos cambian su posición para pasar de una distribución horizontal a una vertical. La sección de argumentos pasa de estar en dos columnas a una sola, en la que figuran los argumentos en contra, seguidos a continuación por los argumentos a favor.

Mapa de navegación

En este proyecto nos hemos centrado en la parte correspondiente a Shake it, pero el objetivo es que forme parte de un proyecto más grande en el que se puedan consultar los distintos proyectos que he realizado. En estos momentos el proyecto Shake it realmente consta de una página que se puede considerar dos por los elementos ocultos que aparecen cuando el usuario interactúa con la aplicación.



De esta manera, se busca conseguir una navegación fluida entre páginas. Por ello se facilita el acceso a cualquiera de las 4 páginas principales (Home, Proyectos, Sobre mí y Contáctame) desde cualquier página de la web, gracias a los menús de navegación de la cabecera y el footer. Por otra parte, en una futura mejora se desarrollará una sección de administración desde la cual se podrán crear debates.

Guía de estilos

Uso de la guía

El objetivo de la guía de estilos es normalizar la estructura de los contenidos y el diseño de la página web de ***"Shake it"***, homogeneizando estilos y estructuras para el desarrollo de nuevas páginas y futuras actualizaciones.

Usabilidad

En esta web es muy importante que al usuario le resulte fácil navegar por ella y que el contenido esté organizado de forma coherente. El menú debe ser sencillo e intuitivo, de ahí que el navegador esté disponible tanto en la cabecera como en el footer, permitiendo navegar de forma sencilla y cómoda por la web. Debe evitarse

en la medida de lo posible que el usuario se sienta confuso o frustrado al navegar por nuestra web.

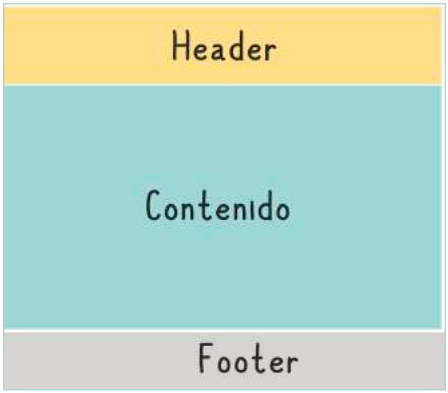
Accesibilidad

Una de las características cuya importancia se destaca en esta web es la accesibilidad. De modo que puedan acceder a la web y a sus contenidos todas las personas, independientemente de la capacidad que presenten (incluidas las tecnológicas).
Se recomienda el seguimiento de las pautas y técnicas desarrolladas por la Iniciativa de Accesibilidad Web (WAI) que se pueden encontrar en el siguiente link: https://www.w3.org/WAI/standards-guidelines/

Diseño gráfico

Estructura Web

La web se compone de tres elementos principales: la cabecera o header, una sección de contenido que ocupa el 100% del ancho de la página hasta un máximo de 1200px y un elemento de pié de página o footer situado en la parte inferior de la página como muestra el esquema.



Cabecera

La cabecera ayudará a identificar la web y a homogeneizar el conjunto de páginas favoreciendo la coherencia del sitio web.
Para todos los dispositivos, en la parte izquierda se sitúa el logo y nombre de la página en la que nos encontramos, en este caso Shake it!. A la derecha se mostrará el menú de navegación de tipo “hamburguesa” para dispositivos móviles y en formato texto para dispositivos grandes.

Cuerpo de la página

Como se ha comentado en el apartado de Estructura web, el ancho del contenido ocupará el 100% de la página hasta un máximo de 1200px. El resto de elementos que aparecen en el contenido se definen en otros apartados más adelante.
El lenguaje utilizado es formal, pero cercano, utilizando la segunda persona del singular.

Pié de página

Se ubica en la parte inferior de la página y ocupa todo el ancho, con un color de fondo *Gris fondo*. Incluye el nombre de la web y página en la que nos encontramos, en este caso Shake it!. Además contiene en el centro un menú auxiliar que lleva a

las secciones principales de la web. En la parte derecha se ubican los enlaces a las principales redes sociales (twitter y linkedin).

Los espacios en blanco

Su uso permitirá que la web se vea equilibrada y limpia, compensando el peso visual del resto de elementos. Esto nos permite marcar los límites y diferenciar fácilmente las diferentes secciones.
El espacio en blanco a ambos lados del contenido será el mismo, de manera que este quede centrado.
El espacio en blanco entre la cabecera y el contenido será el mismo que entre este último y el pié de página.

Color

Los **colores principales** de la web:

Amarillo
Hex: #E08280
RGB: 224, 130, 128

Argumentos **en contra**

Amarillo
Hex: #FFCB65
RGB: 255, 203, 101

Argumentos

Turquesa
Hex: #2696A1
RGB: 38, 150, 161

Argumentos **a favor**

Gris oscuro
Hex: #333333
RGB: 51. 51. 51

Textos, cabecera y footer

Gris
Hex: #494949
RGB: 73. 73. 73

Cabecera **Debates**

Blanco
Hex: #FFFFFF
RGB: 255. 255. 255

Textos

Otros colores de la web:

Amarillo fondo
Hex: #FFF4E1
RGB: 255. 244. 225

Fondo auxiliar argumentos

Rosa fondo
Hex: #EDC4C4
RGB: 237. 196. 196

Fondo auxiliar de argumentos en contra

Turquesa fondo
Hex: #A8D5D9
RGB: 168. 213. 217

Fondo auxiliar de argumentos a favor

Verde Hex: #3AC389 RGB: 58. 195. 137	**Rojo** Hex: #FE0262 RGB: 254. 2. 98
OK / Votos a favor	Error / Voto en contra

El **color del texto** será principalmente en Gris oscuro y Blanco. El resto de colores se usará como norma general para el fondo de algunos elementos y en casos muy concretos donde se quiera resaltar el texto. Cada elemento tiene atribuido un color principal que lo identifica.

Tipografía

Se van a utilizar cuatro fuentes diferentes para la página. El contenido de la página y el texto de la cabecera se escriben con la fuente ***Avenir***. Los textos de los debates se escriben con la fuente ***Trubble***. Los títulos de tipo de argumentos y Shake it se escriben con fuente ***HoboStd***. Y para los títulos de los debates se utiliza la fuente ***Abadi***. Todas ellas contribuyen a darle un aspecto un poco más informal y amigable a la web.

Avenir: Hola mundo (16px)

Trubble: Hola mundo (16px)

HoboStd: Hola mundo (16px)

Abadi: Hola mundo (16px)

Se utilizarán como alternativas fuentes de tipo Sans Serif.
La unidad de medida utilizada para los textos es em (preferibles en accesibilidad), partiendo de un font-size de la página de 16px. Los tamaños de las etiquetas de texto serán las siguientes:

H1: 7em
H2: 3.3em
H3: 1.5em
p: 1em

Además, se utilizará un interlineado de 1,5 para que la lectura sea más agradable al usuario.

Iconografía

Los iconos utilizados en la web no deben de ser malinterpretados fácilmente por el usuario. Son sencillos, pequeños y escasos. Permiten obtener la información de manera clara y rápida sin necesidad de leer ningún texto.
Con el objetivo de mantener la coherencia con el resto de elementos de la web, los iconos se presentarán de manera general sobre un fondo de color y el icono tomará el color blanco. Se hace una excepción con el icono de usuario, cuando se ha iniciado sesión para que se integre mejor con el resto de botones del navegador lateral.
Algunos iconos no irán sobre fondo circular con el fin de ocupar menos espacio y no tomar mucho protagonismo. Cuando se quiera resaltar algún icono de manera justificada se coloreará con color que corresponda según la sección en la que se encuentre.

Funcionalidades y componentes

La aplicación consiste en una web de debates en la que los usuarios pueden votar a favor o en contra y añadir argumentos que justifiquen su decisión. A a su vez, éstos también pueden ser votados. Recordamos que la aplicación es totalmente responsive, por lo tanto se mostrará de forma diferente en función del tipo de dispositivo.

Los componentes de Vue.js se diferencian en tres partes, como se ha mencionado en el apartado de 'Herramientas y tecnologías utilizadas'. La parte del template, corresponde al código HTML del componente, la segunda parte es el script de javascript y finalmente el estilo css.

App

El componente App es el componente padre que contiene el resto de componentes. Este componente interactúa con la base de datos en todo momento para renderizar los datos y sus posibles modificaciones de manera instantánea.

HTML de App:

```
<template>
  <div id="app" class="small-container" >
    <pellin-menu :web_info="web_info" :sections="sections"/>
    <debate-post :posts="posts" :argumentss="deb_argumentss" @addVote:post="addVoteDebate"  @setVisibility="setVisibility($event)"/>
    <argument-form @add:argument="addArgument" :formClass="formClass"/>
    <!--addVote:argument es la funcion a la q llamo desde el componente. Y hace que se ejecute la funcion addVoteArgument de este archivo, con los
    parametros q le paso desde el componente. Recorro la lista de argumentos y el que coincida con el id lo sustituyo con el argumento actualizado q he
    modificado en el componente y he pasado como parametro.-->
    <pellin-footer :web_info="web_info" :sections="sections"/>
  </div>
</template>
```

Código JavaScript de App:

```
<script>
  import DebatePost from '@/components/DebatePost.vue';
  import ArgumentPost from '@/components/DebateArgumentPost.vue';
  import ArgumentForm from '@/components/ArgumentForm.vue';
  import PellinMenu from '@/components/PellinMenu.vue';
  import PellinFooter from '@/components/PellinFooter.vue';

  export default {
    name: "App",
    components: {
      DebatePost,
      ArgumentPost,
      ArgumentForm,
      PellinMenu,
      PellinFooter,
    },
    data() {
      return {
        formClass: '',
        posts: [],
        deb_argumentss: [],
        web_info: { ...
        },
        sections: [ ...
        ],
      }
    },
    mounted() {
      this.getPosts();
      this.getArguments();
    },
    methods:{
      async getPosts() {
        try {
          const response = await fetch('http://localhost:3000/debates')
          const data = await response.json()
          this.posts = data
        } catch (error) {
          console.error(error)
        }
      },
      async addPost(post) {
        try {
          const response = await fetch('http://localhost:3000/debates', {
            method: 'POST',
            body: JSON.stringify(post),
            headers: { 'Content-type': 'application/json; charset=UTF-8' },
          })
          const data = await response.json()
          this.posts = [...this.posts, data]
        } catch (error) {
          console.error(error)
        }
      },
      async addVoteDebate(id, updatedDebate) {
        try {
          const response = await fetch(`http://localhost:3000/debates/${id}`, {
            method: 'PUT',
            body: JSON.stringify(updatedDebate),
            headers: { 'Content-type': 'application/json; charset=UTF-8' },
          })
          const data = await response.json()
          this.posts = this.posts.map(post => (post.id === id ? data : post));
        } catch (error) {
          console.error(error)
        }
      },
      async getArguments() {
        try {
          const response = await fetch('http://localhost:3000/argumentos')
          const data = await response.json()
          this.deb_argumentss = data
        } catch (error) {
          console.error(error)
        }
      },
```

```
    async addArgument(argument) {
      const lastId = this.deb_argumentss.length > 0 ? this.deb_argumentss[this.deb_argumentss.length - 1].id : 0;
      const id = lastId + 1;
      const today = Date.now();
      argument.created_date = today;
      const newArgument = {...argument, id};
      try {
        const response = await fetch('http://localhost:3000/argumentos', {
          method: 'POST',
          body: JSON.stringify(newArgument),
          headers: { 'Content-type': 'application/json; charset=UTF-8' },
        })
        const data = await response.json()
        this.deb_argumentss = [...this.deb_argumentss, data]
        this.getArguments();
      } catch (error) {
        console.error(error)
      }
    },
  }
}
</script>
```

CSS de App:

```
<style>
  #app {
    font-family: 'Avenir', Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
    margin-top: 0;
    width: 100%;
    padding: 0;
  }

  .small-container {
    margin: auto;
  }
</style>
```

PellinMenu

El componente PellinMenu contiene el header de la página web, el cual se mostrará de manera diferente según el tamaño del dispositivo. Cabe destacar que los textos de las distintas secciones que hay en el menú de navegación se rellenan de forma dinámica en función de los datos que se han definido en el componente App. No están escrito de manera estática en el código HTML.

HTML de PellinMenu:

```
<template>
  <header class="fluid-container w-100 m-0">
    <div class="row">
      <section id="identificacion" class="col-10 col-sm-6 col-md-4 py-4 text-left">
        <img id='logo' src="../assets/images/logo.png" alt='logo shake it. Dos bocadillos de conversacion contrapuestos.'/>
        <h2>{{web_info.current_page}}</h2>
      </section>
      <nav class="col-2 d-md-none offset-sm-4 w-100">
          <a id='menu_movil' href="#" ><img src="../assets/images/hamburguesa.png" alt='icono menu desplegable para móviles' class="w-100 pr-1 pt-4
          pt-sm-2"/></a>
        <section id='nav_oculto' class='col-2 offset-10 px-0'>
          <ul>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/home.png" alt='submenu Home'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/proyectos.png" alt='submenu Proyectos'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/persona.png" alt='submenu Sobre mi'/></li>
            <li class='list-unstyled my-1 mx-0'><img src="../assets/images/contacto.png" alt='submenu Contactame'/></li>
          </ul>
        </section>
      </nav>
      <nav id='menu_tablet' class="col-12 col-md-6 d-none d-md-inline-block pt-4 text-right offset-1">
        <a v-for="section in sections" :key="section.id" :href="section.link" :title="section.name" class="row m-0 p-2">{{section.name}}</a>
      </nav>
    </div>
  </header>
</template>
```

Código JavaScript de PellinMenu:

```
<script>
export default {
  name: 'pellin-menu',
  props: {
    web_info: Object,
    sections: Array,
  },
}
window.addEventListener('DOMContentLoaded',CargandoseEstaPagina,false);
//Funcion que se ejecuta cargandose la pagina
  function CargandoseEstaPagina(e) {
    //Definimos variables
    var botonMenu = document.getElementById("menu_movil");
    var navOculto = document.getElementById("nav_oculto");

    //Llamada a la funcion
    botonMenu.addEventListener("click", function(e){toggleMe(navOculto);}, false);
  }
// Funcion para mostrar u ocultar submenu en dispositivos moviles
  function toggleMe(element){
      if(!element) return true;
      if(element.style.display == "none") {
          element.style.display = "block";
      } else {
          element.style.display = "none";
      }
      return true;
  }
</script>
```

CSS de PellinMenu:

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
  header { background-color: #333333; color: #ffffff; z-index: 10;}

  h2 { margin: 0; display: inline-block;}

  a { color: #ffffff; text-decoration: none; display: inline-block;}

  #menu_tablet a:hover { color: #2c2c2c; background-color: #ffffff;}

  #logo { width: 25%; height: auto; display: inline-block; margin: 0 1em;}

  #nav_oculto{ position: fixed; top: 3em; right: 1.1em; display: none; z-index: 10;}

  #nav_oculto img { position: relative; width: 30px; height: auto; z-index: 10;}

/*Media query*/
  @media screen and (min-width: 576px) {
    #nav_oculto img { width: 50px; right: 0; top: 3.2em;}
  }
  @media screen and (min-width: 992px) {
    header{ position: fixed; top:0;}
  }
</style>
```

DebatePost

El componente DebatePost representa un debate que a su vez contiene el componente ArgumentPost con los comentarios sobre ese debate. Además, este componente contiene los métodos necesarios para actualizar la base de datos en lo que se refiere a la votación de los argumentos.

HTML de DebatePost:

```
<template>
  <div id="debate-post" class=" fluid-container text-center">
    <article class="debate" v-for="post in posts" :key="post.id">
      <header class="debateHeader col-12">
        <h2>{{ post.title}}</h2>
        <p>{{ post.ok_vote}} <a @click="addingOk(post)"><img  class="icon mx-2" src="../assets/images/like.png" alt='
        icono para indicar voto a favor'></a><a @click="addingKo(post)"><img class="icon mx-2" src="../assets/images/
        dislike.png" alt='icono para indicar voto en contra'></a>{{ post.ko_vote}}</p>
      </header>
      <argument-post class="col-12"
        :argumentss="argumentss" :debateId="post.id" @addVote:argument="addVoteArgument"
      />
    </article>
  </div>
</template>
```

Código JavaScript de DebatePost:

```
<script>
  import ArgumentPost from '@/components/DebateArgumentPost.vue';

  export default {
    name: 'debate-post',
    props: {
      posts: Array,
      argumentss: Array,
    },
    components: {
      ArgumentPost,
    },
    data() {
      return {
        post: Object,
        debateId: Number,
      }
    },
    methods: {
      addingOk(post) {
        console.log('addingOk debate');
        post.ok_vote +=1;
        this.$emit('addVote:post', post.id, post)
      },
      addingKo(post) {
        console.log('addingKo debate');
        post.ko_vote +=1;
        this.$emit('addVote:post', post.id, post)
      },
      async addVoteArgument(id, updatedArgument) {
        console.log('addVoteArgument post');
        try {
          const response = await fetch(`http://localhost:3000/argumentos/${id}`, {
            method: 'PUT',
            body: JSON.stringify(updatedArgument),
            headers: { 'Content-type': 'application/json; charset=UTF-8' },
          })
          const data = await response.json()
          this.argumentss = this.argumentss.map(argument => (argument.id === id ? data : argument))
        } catch (error) {
          console.error(error);
        }
      },
    },
  }
</script>
```

CSS de DebatePost:

```
<style scoped>

  h2 { font-family: Trubble; }

  p { font-family: HoboStd; }

  #debate-post { width: 90%; display: inline-block;}

  .debateHeader {
    color: #ffffff;
    width: 90%;
    margin: 1.5em auto 0.5em auto;
    box-sizing: border-box;
```

```
/*Media query*/
  @media screen and (min-width: 768px) {
    .icon{ width: 1.8em;}
  }
  @media screen and (min-width: 992px) {
    #debate-post{ position: relative; top:7em;}
  }
  @media screen and (min-width: 1200px) {
    #debate-post{ width: 100%; max-width: 1200px;}
  }
</style>
```

ArgumentPost

El componente ArgumentPost representa un comentario, que gracias a sus propiedades se renderizará en el debate al que esté asociado. Para poder distinguir entre votos a favor y votos en contra se crean dos secciones casi idénticas en las que gracias a un bucle for y la definición de unas condiciones, los argumentos se representarán en la columna que le corresponda.

HTML de ArgumentPost:

```
<template>
  <div id="argument-post" class="m-auto">
    <div class="row">
      <p v-if="argumentss.length ==0">¡Sé el primero en escribir un argumento!</p>
      <section id='likeBox' class="col-12 col-md-6 my-1 px-0 pb-3 pb-sm-5">
        <h1 class="w-100">A FAVOR</h1>
        <article class="argument mb-3 mx-auto pt-3 pb-2" v-for="argument in argumentss" :key="argument.id"
        v-if=" argumentss.length >0 && debateId === argument.post_id && argument.type==='like'"
        v-bind:class="getClass(argument)">
          <h3 class="mx-auto">{{ argument.title}}</h3>
          <p class="mt-2 mb-1 mx-auto text-justify">{{ argument.description}}</p>
          <div id="votos">{{argument.ok_vote}} <a @click="addingOk(argument)"><img  class="icon mx-2" src="
          ../assets/images/like.png" alt="icono para indicar voto a favor"></a><a @click="addingKo(argument
            )"><img class="icon mx-2" src="../assets/images/dislike.png" alt="icono para indicar voto en
          contra"></a>{{ argument.ko_vote}}</div>
          <!--Llamo a addingOk funcion de este archivo cuando hago click. Añado un voto y llamo a la
          funcion addVote:argument de la App(HTML) y le paso dos argumento el id y el argumento en si.-->
        </article>
        <button class="heartBeat mt-4 px-4 py-1" @click="sendInfo('like')">Añade tu comentario</button>
      </section>
      <section id='dislikeBox' class="col-12 col-md-6 my-1 px-0 pb-3 pb-sm-5">
        <h1 class="w-100">EN CONTRA</h1>
        <article class="argument mb-3 mx-auto pt-3 pb-2" v-for="argument in argumentss" :key="argument.id"
        v-if="debateId === argument.post_id && argument.type==='dislike'" v-bind:class="getClass(argument)"
        >
          <h3 class="mx-auto">{{ argument.title}}</h3>
          <p class="mt-2 mb-1 mx-auto text-justify">{{ argument.description}}</p>
          <div>{{ argument.ok_vote}} <a @click="addingOk(argument)"><img  class="icon mx-2" src="../assets/
          images/like.png" alt="icono para indicar voto a favor"></a><a @click="addingKo(argument)"><img
          class="icon mx-2" src="../assets/images/dislike.png" alt="icono para indicar voto en contra"></a>
          {{ argument.ko_vote}}</div>
          <!--Llamo a addingOk funcion de este archivo cuando hago click. Añado un voto y llamo a la
          funcion addVote:argument de la App(HTML) y le paso dos argumento el id y el argumento en si.-->
        </article>
        <button class="heartBeat mt-4 px-4 py-1" @click="sendInfo('dislike')">Añade tu comentario</button>
      </section>
    </div>
  </div>
</template >
```

Código JavaScript de ArgumentPost:

```
<script>
  import {eventBus} from "../main";

  export default {
    name: 'argument-post',
    props: {
      argumentss: Array,
      debateId: Number,
    },
    data() {
      return {
        formClass: 'invisible',
      }
    },
    methods: {
      addingOk(argument) {
        console.log('addingOk argument');
        argument.ok_vote +=1;
        this.$emit('addVote:argument', argument.id, argument)
      },
      addingKo(argument) {
        console.log('addingKo argument');
        argument.ko_vote +=1;
        this.$emit('addVote:argument', argument.id, argument)
      },
      getClass(argument){
        console.log('getClass argument');
        return {
          'like': argument.type === 'like',
          'dislike': argument.type === 'dislike',
        }
      },
      sendInfo(type) {
        console.log('sendInfo argument');
        eventBus.$emit('setVisibility', 'visible', this.debateId, type)
      },
    },
  }
</script>
```

CSS de ArgumentPost:

```
<style scoped>
  #argument-post{ width: 90%; }

  #likeBox, #dislikeBox { background-color: rgba(255, 203, 101, 0.2); color: #ffffff; }

  .like, #likeBox button { background-color: #2696A1; border: 1px solid #2696A1;}

  .dislike, #dislikeBox button { background-color: #e08280; border: 1px solid #e08280;}

  .like>p { background-color: #a8d5d9; width: 95%;}

  .dislike>p { background-color: #edc4c4; width: 95%;}

  .argument { box-shadow: 5px 10px 8px #888888; width: 95%;}

  .argument p { color: #2c2c2c; padding: 1em 0.8em;}

  h1 { font-family: HoboStd; background-color: #FFCB65; width: 100%; font-size: 2em; }
```

```
h3, #votos { font-family: Abadi;}

.icon{ width: 5%; height: auto;}

button {
  color: #ffffff;
  border-radius: .5em;
  font-weight: bold;
  font-size: 1em;
}

#argument-post section button:hover {
  color: #2c2c2c;
  background-color: white;
  border: 1px solid #2c2c2c;
  box-shadow: 0em 0em .5em .5em #2c2c2c;
}

.heartBeat:hover {
-webkit-animation-name: heartBeat;
animation-name: heartBeat;
-webkit-animation-duration: 1.3s;
animation-duration: 1.3s;
-webkit-animation-timing-function: ease-in-out;
animation-timing-function: ease-in-out;
}

/*Animaciones */
@-webkit-keyframes heartBeat {
0% { -webkit-transform: scale(1); transform: scale(1);}
14% { -webkit-transform: scale(1.3); transform: scale(1.3);}
28% { -webkit-transform: scale(1); transform: scale(1);}
42% {-webkit-transform: scale(1.3); transform: scale(1.3);}
70% {-webkit-transform: scale(1); transform: scale(1);}
}

@keyframes heartBeat {
  0% { -webkit-transform: scale(1); transform: scale(1);}
  14% { -webkit-transform: scale(1.3); transform: scale(1.3);}
  28% { -webkit-transform: scale(1); transform: scale(1);}
  42% { -webkit-transform: scale(1.3); transform: scale(1.3);}
  70% { -webkit-transform: scale(1); transform: scale(1);}
}

/*Media query*/
@media screen and (min-width: 768px) {
  #likeBox { border-right: .5em solid #ffffff;}
  #dislikeBox { border-left: .5em solid #ffffff;}
}
@media screen and (min-width: 1200px) {
  .argument { width: 85%;}
}
</style>
```

ArgumentForm

El componente ArgumentForm se utiliza para añadir argumentos al debate a partir de un click que haga el usuario. Este componente está al mismo nivel que los componentes PellinMenu, DebatePost y PellinFooter y consigue pasar la información entre componentes gracias a las propiedades y métodos que contiene. En función de dónde haga el click se manda directamente el identificador del debate al que responde y el tipo de argumento.

HTML de ArgumentForm:

```
<template>
  <div id="argument-form"  class="fluid-container text-center py-4" v-bind:class= "formClass">
    <!-- the submit event will no longer reload the page -->
    <form @submit.prevent="handleSubmitArgument" class="row d-flex justify-content-center">
      <div class="row">
        <label class="col-12 mx-auto text-center">Título del argumento</label><br/>
        <input class="col-10 mx-auto "
          :class="{ 'has-error': submitting && invalidArgumentTitle }"
          v-model="argument.title"
          type="text"
          @focus= "clearStatus"
          @keypress= "clearStatus"
        />
      </div>
      <div class="row">
        <label class="col-12 text-center pt-3">Descripción del argumento</label><br/>
        <textarea class="col-10 mx-auto"
          rows='4' cols='14'
          v-model="argument.description"
          :class="{ 'has-error': submitting && invalidDescription }"
          type="text"
          @focus= "clearStatus"
        />
      </div>
      <p v-if="error && submitting" class="error-message">
        ❗ Por favor, rellena los campos requeridos.
      </p>
      <p v-if="success" class="success-message">
        ✅ Debate añadido satisfactoriamente
      </p>
      <button class="col-10 col-sm-5 mx-auto my-3 my-md-5">Añadir argumento</button>
    </form>
  </div>
</template>
<!--Utilizamos v-model para que sea bidireccional la actualización de datos-->
```

Código JavaScript de ArgumentForm:

```
<script>
  import {eventBus} from "../main";

  export default {
    name: 'argument-form',
    data() {
      return {
        formClass:'invisible',
        post_id: Number,
        argType: String,
        submitting: false,
        error: false,
        success: false,
        argument: {
          title: '',
          description: '',
          type: '',
          post_id: 0,
          ko_vote: 0,
          ok_vote: 0,
        },
      }
    },
```

```
    created() {
      eventBus.$on('setVisibility', (visibility, debateId, type) => {
        console.log('created');
          this.formClass = visibility;
          this.post_id = debateId;
          this.argType = type;
      });
    },
    methods: {
      handleSubmitArgument(){
        this.submitting = true
        this.clearStatus()

        if (this.invalidArgumentTitle || this.invalidDescription) {
          this.error = true
          return
        }
        this.argument.type = this.argType;
        this.argument.post_id = this.post_id;
        console.log('handleSubmitArgument');
        this.$emit('add:argument', this.argument)

        this.argument = {
          title:'',
          description: '',
          type: '',
          post_id: 0,
          ko_vote: 0,
          ok_vote: 0,
        }
        this.error = false
        this.success = true
        this.submitting = false

        this.formClass = 'invisible';
      },
      clearStatus(){
        this.success = false
        this.error = false
      },
    },
    computed: {
      invalidArgumentTitle() {
        return this.argument.title === ''
      },
      invalidDescription() {
        return this.argument.description === ''
      },
    },

  }
  //Se recomienda usar add:post en vez de add-post en la d
</script>
```

CSS de ArgumentForm:

```
<style scoped>
  form {
    margin-bottom: 2rem;
    background-color: #333333;
    color: #ffffff;
    width: 75%;
    max-width: 630px;
    padding: 1em;
    margin: auto;
    position: fixed;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
  }
  .visible {
    position: fixed;
    top:0;
    width: 100%;
    height: 100%;
    display: block;
    background-color: rgba(51,51,51,0.5);
  }
  .invisible {display: none;}

  [class*="-message"] {font-weight: 500;}

  .error-message { color: #d33c40;}

  .success-message { color: #32a95d;}

  label { display: inline-block;}

  input, textarea { display: inline-block; clear:both;}

/*Media query*/
  @media screen and (min-width: 992px) {
    form{ width: 50%;}
  }
</style>
```

PellinFooter

El componente PellinFooter contiene el footer de la página web, que se mostrará de manera diferente según el tamaño del dispositivo. Al igual que en el header, los textos de las distintas secciones que hay en el menú de navegación se rellenan de forma dinámica en función de los datos que se han definido en el componente App. Además, cabe destacar que el orden de representación de los iconos de las redes sociales cambia en función del tamaño del dispositivo.

HTML de PellinFooter:

```
<template>
  <footer class="fluid-container w-100 px-3 mt-5">
    <div class="row">
    <section class="col-10 col-md-3 col-lg-3 order-1 pt-4 order-1">
      <h3 class="row m-0">{{web_info.title}}</h3>
      <p class="row m-0">{{web_info.current_page}}</p>
    </section>
    <nav class="col-12 col-sm-9 col-md-6 col-lg-6 order-3 order-sm-2 pt-4 text-center text-md-right">
      <a v-for="section in sections" :key="section.id" :href="section.link" class="row m-auto p-2" :title="
        section.name">{{section.name}}</a>
    </nav>
    <section id="rrss" class="d-flex col-2 col-sm-3 col-md-3 col-lg-2 order-2 order-sm-3 offset-lg-1 py-4
    justify-content-end ">
      <a href="https://twitter.com/E_Pellin" title='Haz click para ir al twitter de epellin' class="
      col-auto my-0 mx-1 p-1"><img class="img-fluid" src="../assets/images/twitter_white.png" alt="icono
      twitter"></a>
      <a href="https://www.linkedin.com/in/epellin/" title='Haz click para ir al linkedin de epellin' class
      ="col-auto my-0 mx-1 p-1"><img class="img-fluid" src="../assets/images/linkedin_white.png" alt="
      icono Linkedin"></a>
    </section>
    </div>
  </footer>
</template>
```

Código JavaScript de PellinFooter:

```
<script>
export default {
  name: 'pellin-footer',
  props: {
    web_info: Object,
    sections: Array,
  },
}
</script>
```

CSS de PellinFooter:

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
  footer { background-color: #333333; color: #ffffff;}

  h3 { font-family: HoboStd; }

  p { margin: 0;}

  a { color: #ffffff; text-decoration: none; display: inline-block;}

  a:hover { color: #FFCB65;}

  img { width: 21px; height: auto;}

/*Media query*/
  @media screen and (min-width: 992px) {
    footer{ position: relative; top:7em;}
  }
</style>
```

Descripción de la API Node.js

En este proyecto se desarrolló una API REST ("REST" significa Transferencia de Estado Representacional) bajo node.js. En pocas palabras, las API REST implican solicitudes y respuestas, no muy diferentes a las de una página web. Realiza una solicitud a un recurso almacenado en un servidor y el servidor responde con la información solicitada. El protocolo utilizado para transportar los datos es HTTP. A continuación se reflejaran los métodos (GET, POST y UPDATE) de la API describiendo cada uno de ellos.

- **Método GET - Debate**

URL:
/debates

Method:
GET

URL - Params:
Required: ninguno

Success Response:
Code: 200

```
{
   "_id": "5d02b6421c9d44000086e5da",
   "id": 1,
   "title": "debate 1",
   "ok_vote": 28,
   "ko_vote": 10,
   "created_date": "2019-01-19T23:00:00.000Z"
}
```

Code:

```
router.get('/debates', function (req, res,next) {
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
    res.setHeader("Content-Security-Policy", "default-src 'none'; font-src 'self' data:; style-src 'self'
    var response;

      MongoClient.connect(url, function(err, db) {
        if (err) throw err;
        var dbo = db.db("shakeit_db");
        dbo.collection("debate").find({}).toArray(function(err, result) {
            if (err) throw err;
            res.json(result);
        });
        });
});
```

Método UPDATE - Debate

URL:
/debates

Method:
PUT

URL - Params:

Required:

- Title
- ko_vote
- ok_vote

Success Response:
Code: 200

```
{
   "_id": "5d02b6421c9d44000086e5da",
   "id": 1,
   "title": "debate 1",
   "ok_vote": 28,
   "ko_vote": 10,
   "created_date": "2019-01-19T23:00:00.000Z"
}
```

Error Response:
Code: 500

```
{
"error" : "hubo un error a nivel de servidor."
}
```

Code:

```
//update
router.put('/debates/:id', function (req, res,next) {
   const {title, ok_vote, ko_vote, created_date} = req.body;
   const {id }= req.params;
   //var myid = { "id": Number.parseInt(id)};
   var myid = parseInt(id);
   if (title && (ok_vote ||  ko_vote) && created_date){
       var newvalues = { $set: {  ok_vote: ok_vote, ko_vote: ko_vote} };
       MongoClient.connect(url, function(err, db) {
           if (err) throw err;
           var dbo = db.db("shakeit_db");
           dbo.collection("debate").updateOne({"title" : title}, newvalues, function(err, res) {
               if (err) throw err;
           db.close();
           });
```

```
            res.json({
                "id": myid,
                "title": title,
                "ok_vote": ok_vote,
                "ko_vote": ko_vote,
                "created_date": created_date,
            });
        });
    }else{
        console.log("Some params is empty or wrong");
        res.status(500).json({"error" : "All the fields are required"});
    }
});
```

- **Método GET - Argumentos**

URL:
/argumentos

Method:
GET

URL - Params:
Required: Ninguno

Success Response:
Code: 200


```
[
  {
    "_id": "5d002202a20c1755042ba9b1",
    "id": 1,
    "title": "argumento Rojo",
    "description": "description 1",
    "ok_vote": 117,
    "ko_vote": 2002,
    "post_id": 1,
    "created_date": "2018-12-31T23:00:00.000Z",
    "type": "dislike"
  }
]
```

Code:

```
//routes
router.get('/argumentos', function (req, res,next) {
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
    res.setHeader("Content-Security-Policy", "default-src 'none'; font-src 'self' data:; style-src 'self'
    MongoClient.connect(url, function(err, db) {
        if (err) throw err;
        var dbo = db.db("shakeit_db");
        dbo.collection("argumentos").find({}).toArray(function(err, result) {
          if (err) throw err;
         res.json(result);
        });
      });

});
```

Método POST - Argumentos

URL:
/argumentos

Method:
POST

URL - Params:

Required:

- title
- description
- post_id
- type
- created_date
- ok_vote
- ko_vote

Success Response:
Code: 200

```
{
   "msg": "registro guardado correctamente."
}
```

Error Response:
Code: 500

```
{
"error" : "hubo un error a nivel de servidor”
}
```

Code:

```
//create argumento
router.post('/argumentos', function (req, res,next) {
    const {id, title, description, ok_vote, ko_vote, post_id, created_date, type} = req.body;
      if (id && title && description && created_date){
          const id_var = id +1;
          console.log("id: " + id_var);
          const newArgumento = {...req.body,id};
          console.log(newArgumento);
          var myobj = {id : id , title: title, description:description, ok_vote:ok_vote, ko_vote:ko_vote, post_id:post_id, created_date:created_date,
          console.log("myobj es: " + myobj);
          MongoClient.connect(url, function(err, db) {
              if (err) throw err;
              var dbo = db.db("shakeit_db");
              dbo.collection("argumentos").insertOne(myobj, function(err, res) {
                  if (err) throw err;
                  db.close();
              });
              res.json({"msg" : "registro guardado correctamente."});
          });
          /*res.json(myobj);*/
      }else{
          console.log("Some params is empty or wrong");
          res.send("Worng request");
      }

});
```

Método UPDATE - Argumentos

URL:
/argumentos

Method:
PUT

URL - Params:

Required:

- title
- ok_vote
- ko_vote

Success Response:
Code: 200


```
{
    "_id": "5d002202a20c1755042ba9b1",
    "id": 1,
    "title": "argumento Rojo",
    "description": "description 1",
    "ok_vote": 117,
    "ko_vote": 2002,
    "post_id": 1,
    "created_date": "2018-12-31T23:00:00.000Z",
    "type": "dislike"
}
```

Error Response:
Code: 500
{
"error" : "hubo un error a nivel de servidor"
}

Code:

```
//update
router.put('/argumentos/:id', function (req, res,next) {
   const {title, description, ok_vote, ko_vote, created_date, post_id, type} = req.body;
   const {id }= req.params;
   var myquery = { 'title': title};
   //if (ok_vote && ko_vote)
   if (title && (ok_vote ||  ko_vote) && created_date){
       var newvalues = { $set: { ok_vote: ok_vote, ko_vote: ko_vote} };
       MongoClient.connect(url, function(err, db) {
           if (err) throw err;
           var dbo = db.db("shakeit_db");
           dbo.collection("argumentos").updateOne(myquery, newvalues, function(err, res) {
               if (err) throw err;
           db.close();
           });

           res.json({
               "id": id,
               "title": title,
               "description": description,
               "ok_vote": ok_vote,
               "ko_vote": ko_vote,
               "post_id": post_id,
               "created_date": created_date,
               "type": type
           });
       });
   }else{
       console.log("Some params is empty or wrong");
       res.status(500).json({"error" : "All the fields are required"});
    }
});
```

Accesibilidad

En esta aplicación se da importancia a la accesibilidad. De modo que puedan acceder a la web y a sus contenidos todas las personas, independientemente de la discapacidad que presenten (incluidas las tecnológicas).
Se ha pasado un primer test de accesibilidad en el que se ha encontrado una serie de errores, los más relevantes de los cuales han sido solventados. Para llevarlo a cabo hemos seguido las pautas y técnicas desarrolladas por la Iniciativa de

Accesibilidad Web (WAI) que se pueden encontrar en el siguiente link: https://www.w3.org/WAI/standards-guidelines/

Algunas de las incidencias solucionadas son:

- Indicar el idioma de la página en la etiqueta html.
- Corregir la inexistencia de un h1 al inicio de la web.
- Añadir la el atributo alt a las etiquetas omg que no lo tienen.
- Añadir contenido a un enlace que no lo tenía.

Pruebas

La aplicación ha sido probada en varias ocasiones por usuarios potenciales relacionados con el tema de las metodologías ágiles. Como resultado se llevaron a cabo algunas acciones:

- Eliminar el botón genérico de añadir argumento y vincularlo al apartado específico al que se refiere.
- Dejar el header fijo solo para dispositivos grandes.
- Cambiar el color de fondo de los argumentos para favorecer el contraste.
- Oscurecer el fondo de la pantalla cuando aparece el formulario de añadir argumento para centrar la atención en el formulario.

Conclusiones

Futuras mejoras

Gracias a la técnica de inspección y adaptación que propone Scrum, a lo largo de la vida de este producto van a seguir surgiendo posibles mejoras que añadan valor al mismo. Se detallan a continuación las más relevantes a día de hoy:

- Encontrar el hosting adecuado para poder alojar la aplicación teniendo en cuenta que se trata de una aplicación que usa Vue.js y Node.js. El hosting en el que estaba previsto desplegar la aplicación no soporta Node correctamente.
- Crear página de administración desde la que poder añadir los debates. La API está preparada. Solo falta la parte de Vue.
- Ordenar los argumentos, de forma que aparezcan ordenados de más votados a menos.
- Implementación del efecto acordeón que incorpora jQuery UI o equivalente para que sólo se muestren los argumentos del debate activo y se oculten los del resto. Según la documentación analizada no es muy recomendable utilizar jQuery UI con Vue.js. Habría que investigar qué alternativa sería la idónea para esta situación.

Comentarios

La realización de esta aplicación ha sido un reto muy bonito. El objetivo principal era poner en práctica los conocimientos aprendidos durante todo el ciclo con el plus de utilizar unas tecnologías relativamente nuevas con alta demanda en el mercado laboral.

Tanto Vue como Node son tecnologías que se han visto durante el curso muy por encima y que he tenido la suerte de conocer más en detalle gracias a la Formación en Centros de trabajo. Ambas me han parecido fascinantes y con un potencial enorme.

He disfrutado de todo el proceso de desarrollo del proyecto y animo a todo aquel que no conozca Vue o Node a que lo haga. En el apartado de fuentes dejo algunos enlaces de tutoriales muy completos y bien explicados que facilitan enormemente el aprendizaje.

La realización de este proyecto me ha permitido conectar muchos conceptos que hasta ahora mantenía inconexos. Me ha parecido muy interesante todo el proceso y lo he disfrutado enormemente. Ha supuesto un gran esfuerzo y tiempo de investigación especialmente. Aún así tengo ganas de poder implementar las mejoras mencionadas anteriormente y seguir aprendiendo.

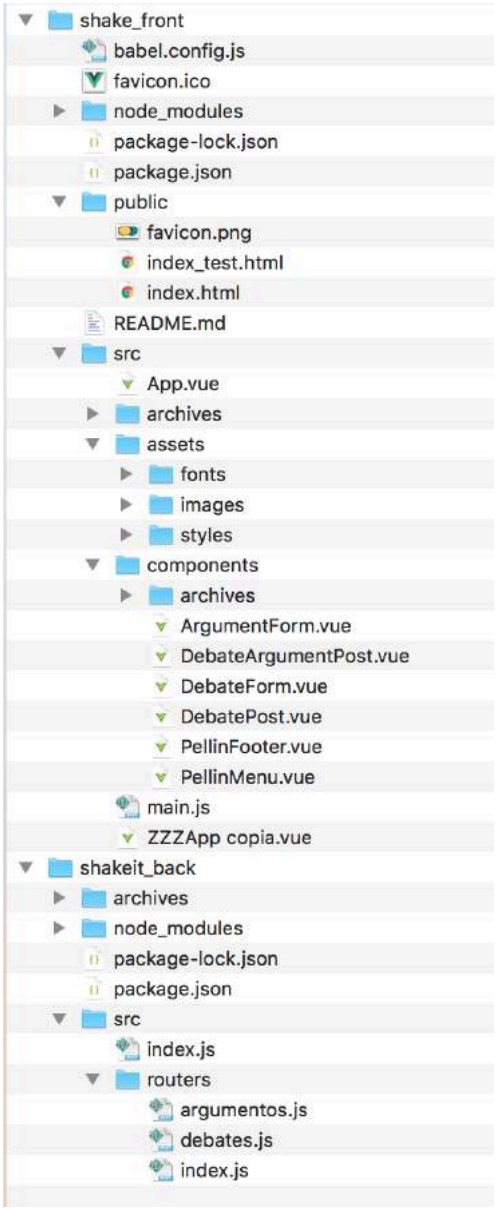
Apéndice

Fuentes

- Design Thinking: http://www.designthinking.es/inicio/
- Scrum: https://www.scrum.org/
- Git: https://git-scm.com/
- Sourcetree: https://www.sourcetreeapp.com/
- Tutorial básico Vue: https://scrimba.com/playlist/pZ45Hz
- Tutorial completo Vue: https://www.taniarascia.com/getting-started-with-vue/
- Vue: https://vuejs.org
- Tutorial Node: https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html
- Node: https://nodejs.org/es/
- HTML: https://www.w3schools.com/
- CSS: https://www.w3schools.com/
- Animaciones CSS: https://daneden.github.io/animate.css/
- jQuery: https://jqueryui.com/
- jQuery: https://daneden.github.io/animate.css/

- Bootstrap: https://getbootstrap.com
- MongoDB: https://cloud.mongodb.com
- Investigación: https://stackoverflow.com

Estructura de carpetas



Se ha dividido la aplicación en dos grupos principales, la parte front con Vue y la parte back con Node.

En ambos casos se incluye el directorio node_modules que permite ampliar las funcionalidades de Vue y Node.

En la parte front tenemos los directorios public y src. El primero contiene el fichero índex.html y el segundo contiene los recursos que se van a utilizar para hacer que

funcione la aplicación. Estos se dividen en assets y componentes. En este último incluimos un fichero por componente.

En la parte back se encuentra el directorio src con el fichero índex.js principal que hay que ejecutar para que funcione la aplicación en local y también está el directorio routers que contiene las rutas que se van a emplear. El primer fichero index.js llama al segundo para conectar con las rutas. En los ficheros argumentos.js y debates.js se desarrolla la conexión a la base de datos y las llamadas que se harán desde Vue.

Guía de implementación

A continuación se comenta en líneas generales los pasos necesarios para poder ejecutar la aplicación objeto de este proyecto en un ordenador en local.

En primer lugar hay que instalar Node. No tiene ninguna complejidad especial, sólo hay que seguir las especificaciones de la página oficial. Una vez instalado, desde la consola hay que ir al directorio src de la parte back y ejecutar el fichero index.js para que empiece a correr la aplicación.

Comando:
node index.js

En segundo lugar hay que instalar Vue en el ordenador. Vue proporciona un CLI oficial para el andamiaje rápido de aplicaciones de una sola página. Proporciona configuraciones de compilación que incluyen baterías para un flujo de trabajo de front end moderno.

Los comandos son:
npm install -g @vue/cli

Y si queremos crear un proyecto nuevo:
vue create my-project

En este caso, bastará con ubicarnos en el directorio que contiene la parte front y ejecutar el comando:
ppm run serve

Una vez tenemos Vue y Node instalados, hay que descargarse el repositorio con el proyecto, que se puede encontrar en el siguiente link:
https://github.com/epellin/Shakeit-v2

Finalmente podemos ejecutar el fichero index.html de la parte front, en nuestro navegador y todo funcionará perfectamente. Si se quisiera utilizar otra base de datos, simplemente hay que cambiar la URL de MongoDB a la que apuntan los ficheros de debates.js y argumentos.js por la que se quiera utilizar.