BRAINBOOST

Memoria

Proyecto final de ciclo

SAN PABLO, RAPOSO, SANTIAGO FRANCISCO
MENDOZA SAUCEDO, JUAN CARLOS
TUMANOV KONDRATYEVA, YEVGEN

11 DE JUNIO DE 2023
CIFP CARLOS III - CARTAGENA
2º CURSO
DESARROLLO DE APLICACIONES WEB

Contenido

BrainBoost:

1.- Introducción al proyecto.

El proyecto consiste en una aplicación enfocada a ayudar a los estudiantes a memorizar conceptos, fórmulas, vocabulario en idiomas, entre otros, de manera divertida y amena. **Ofrece dos niveles de dificultad (fácil y difícil), y cuenta con dos modalidades**: práctica y desafío.

- **El modo práctica** permite al usuario estudiar y practicar las preguntas.
  - Se plantean preguntas, y el usuario las va respondiendo. Si falla una pregunta, puede volver a contestarla, hasta dar con la respuesta correcta.
- **El modo desafío** pone a prueba sus habilidades y memoria.
  - Se plantean preguntas, y el usuario las va respondiendo. Si falla una pregunta, no podrá volver a contestarla (si no es iniciando un intento nuevo del test).

Las dificultades presentan las siguientes características:

- **La dificultad fácil**: se proporciona la nota instantáneamente, nada más responder a la pregunta.
- **La dificultad difícil**: la nota se da al final, cuando se envían las respuestas al servidor.

Tanto el modo práctica, como el modo desafío, se basan en un **sistema de preguntas y respuestas almacenadas en la BB.DD que se formularán en orden aleatorio**, en el que por cada pregunta acertada, sumas una puntuación (en la escala del 0 al 10). Los fallos no restan, el objetivo no era plantearlo como un examen, sino como un "juego" para hacer del aprendizaje algo mucho más entretenido.

El repositorio GitHub del proyecto es este:
https://github.com/yevgentumanov/BrainBoostLaravel

1.1.- Hitos.

Aunque nos hubiese encantado implementar todas las características que se nos iban ocurriendo (entre otras, la creación de tests propios o el compartir dichos tests con amigos), hoy podemos presentar un proyecto totalmente funcional en todo lo que hemos implementado, entregándole así al usuario final un producto que se ve terminado y que cumple con **todos los hitos del anteproyecto**:

- **Hito 1**: Diseño y creación de la base de datos que almacenará las preguntas y respuestas, así como también los usuarios registrados en la aplicación.
- **Hito 2**: Crear categorías de test e incluir preguntas y respuestas.
- **Hito 3**: Programación de la lógica de la aplicación, centrándose en el funcionamiento de las preguntas y respuestas y su corrección.
- **Hito 4**: Diseño de la interfaz web, utilizando Bootstrap y CSS.

- **Hito 5**: Implementación de la página de login y registro.
- **Hito 6**: Integración de la base de datos con la lógica de la aplicación, completando todas sus funcionalidades.
- **Hito 7**: Implementación de la funcionalidad de estadísticas y seguimiento de progreso del usuario.
- **Hito 8**: Pruebas y depuración de errores. Optimización y pruebas de rendimiento de la aplicación.
- **Hito 9**: Lanzamiento y presentación de la aplicación.

Además de estos hitos, hemos logrado otros, como, por ejemplo:

- **Hemos hecho el despliegue (con SSL) de la aplicación en Internet**. El proyecto es accesible mediante la siguiente dirección en Internet:

  https://brainboost.es

- **Se ha implementado un sistema de verificación de usuarios**, por el que enviamos un email para verificar las cuentas creadas.
- **Se ha implementado la autenticación con Google Auth** (disponible a través de la dirección en Internet).

2.- Tecnologías empleadas.

Para el desarrollo de nuestra aplicación web, hemos tenido múltiples debates:

Al principio, pensamos en realizar todo el proyecto usando PHP y JavaScript puros.

1. **Ventajas**: son los lenguajes que hemos dado durante el curso, y, por ende, tenemos experiencia utilizándolos.
2. **Desventajas**:
    a. **En el backend**: se torna compleja la gestión de rutas, la creación de las APIs (algo que podía convertirse en ardua tarea), la generación de contenidos dinámicos con PHP puro, etc.
    b. **En el frontend**: la manipulación del DOM "a mano", y mantener los datos que el usuario ve por pantalla sincronizados con los datos de la aplicación, podía volverse un tanto complicado, al no tener una característica de "reactividad".

Hacerlo todo en PHP y JavaScript, como se puede observar, aunque es una opción que hemos barajado, nos pareció que podía complicarnos mucho el código, así que, decidimos usar frameworks tanto para el backend como para el frontend, para obtener un código mucho más limpio, lo que nos ha permitido centrarnos en tareas menos rudimentarias e ir al grano a implementar otras cosas, como la lógica de la aplicación (de una forma más sencilla), la maquetación y la manipulación de los datos en la BB.DD.

Por tanto, nuestro "stack" de tecnologías web es:

- **Laravel & Blade**, para el backend.
- **Vue & Vite**, para el frontend de los tests.
- **Bootstrap & Blade (Laravel)**, para diseño y maquetación.

En las siguientes subsecciones, explicamos las decisiones de por qué utilizar cada uno de ellos.

2.1.- ¿Por qué utilizar Laravel & Blade?

Pensamos en utilizar el framework Laravel para el backend. Este es el gran pilar del proyecto, sobre el que se sustenta todo lo demás.

¿Qué es Laravel?

Laravel es un framework de desarrollo web muy popular y de código abierto que tiene muchas ventajas para los desarrolladores. A continuación, se enumeran algunas de las ventajas más destacadas de Laravel:

- **La sintaxis limpia y expresiva de Laravel**, que facilita el desarrollo de aplicaciones web. Resulta sencillo escribir y mantener un código limpio y fácil de entender gracias a su enfoque en la legibilidad del código.
- **Arquitectura Modelo Vista Controlador (MVC)**: Laravel utiliza el patrón MVC, que facilita el desarrollo estructurado y la separación del código en partes bien diferenciadas. Esto mejora la legibilidad y el mantenimiento del proyecto al permitir una mejor organización del código y **una mayor modularidad**.

- **ORM integrado**: Eloquent es un ORM (Mapeo Relacional de objetos) proporcionado por Laravel que permite interactuar con la base de datos mediante una sintaxis orientada a objetos. La manipulación de datos y la realización de consultas a la base de datos se facilitan con Eloquent, lo que evita la necesidad de escribir sentencias SQL manualmente.
- **Sistema de enrutamiento poderoso**: Laravel proporciona un sistema de enrutamiento flexible y fácil de usar. Permite definir rutas y conectarlas a controladores y métodos específicos. Esto facilita el desarrollo de APIs RESTful y el manejo de solicitudes HTTP entrantes.
- **Seguridad integrada**: Laravel ofrece numerosas funciones y herramientas para mejorar la seguridad de las aplicaciones web. Proporciona métodos para proteger las rutas, validar y filtrar los datos de entrada, cifrar contraseñas, prevenir ataques de CSRF (Cross-Site Request Forgery) y XSS (Cross-Site Scripting), entre otros.
- **Cuenta con el sistema de plantillas Blade**.

Aunque Blade lo utilizamos como parte de la maquetación, creemos conveniente explicar en este apartado en qué consiste Blade y para qué más cosas lo hemos utilizado.

¿Qué es Blade?

Blade es un sistema de plantillas Laravel con múltiples características que facilitan el desarrollo de vistas en el framework. A continuación, se enumeran algunas de las ventajas más destacadas de Blade:

- **La sintaxis clara y concisa de Blade**, permite crear y manipular vistas de manera eficiente. La sintaxis de Blade es similar a la de PHP, pero incluye estructuras de control más pequeñas y directivas para trabajar con plantillas.
- **Herencia de plantillas**: Blade permite la herencia de plantillas, lo que significa que se puede definir una plantilla base y luego expandirla para vistas específicas. Esto facilita la creación de un diseño de aplicación compartible y la reutilización de secciones de código en varias páginas.
- **Directivas útiles**: Blade proporciona una variedad de directivas que simplifican las tareas comunes en las vistas. Por ejemplo, las directivas @if, @foreach, @include y @yield, para realizar comprobaciones condicionales, iterar conjuntos de datos, incluir subvistas y definir secciones de contenido. Estas instrucciones ayudan a mantener el código limpio y fácil de entender.

Estas bondades hacen que trabajar con vistas en Laravel sea más eficiente y productivo.

Como se puede comprobar, **Laravel nos solucionó la parte de la gestión de rutas, y, sobre todo, gracias a Blade** (su motor de plantillas), nos facilitó muchísimo el poder generar contenido dinámico desde el lado del servidor, con un código mucho más limpio y legible.

También, **gracias a sus modelos ORM Eloquent y los controladores, gestionamos los datos de la BB.DD con gran eficacia** y sin complejidades típicas que nos podríamos encontrar utilizando PHP puro con mysqli o PDO.

Las páginas se generan en base a plantillas y fragmentos de Blade que hemos creado. De esta forma, la maquetación de las páginas se convierte en un proceso mucho menos tedioso,

al poder generar contenido dinámico que llega directamente desde el controlador. **Es aquí donde reside la potencia de Blade**.

Las rutas de Laravel nos permiten definir exactamente a qué sitios puede acceder el usuario. Por otra parte, la seguridad integrada de Laravel permite que solo nosotros, podamos acceder a las APIs (gracias a su protección incluida de CORS), y protege la información de los usuarios para que solo ellos, puedan acceder a su información.

En resumen, podemos resumir las ventajas y desventajas de utilizar Laravel en:

1. **Ventajas**: poseíamos conocimientos previos del curso.
2. **Desventajas**: aún así, ha requerido de investigación en la documentación de Laravel, y tutoriales para otras cosas que mencionamos en la sección de dificultades.

2.2.- ¿Por qué utilizar Vue?

Dado que la realización de los tests requería de mucha **interactividad con el usuario**, JavaScript "puro" se tornaba como una opción poco eficiente. Es por eso por lo que decidimos hacer uso de un **framework** para el lado del cliente.

¿Qué es Vue?

Vue es un framework open source (software de código abierto) de JavaScript para crear interfaces de usuario interactivas creado por Evan You.
Debido a sus múltiples funciones, Vue.js es un framework de JavaScript muy popular en el desarrollo de aplicaciones web. **Algunas de las ventajas de Vue.js son**:

- **Aprendizaje fácil**: Vue.js es conocido por su curva de aprendizaje suave. Incluso los desarrolladores con menos experiencia en frameworks JavaScript pueden comenzar a trabajar con Vue.js rápidamente gracias a su sintaxis fácil de entender y su enfoque gradual.
- **Flexibilidad**: Vue.js es extremadamente flexible y se integra fácilmente en proyectos existentes. Además, puede aplicarlo a una sección específica de una página sin tener que reescribir todo el código actual.
- **Rendimiento**: Vue.js es conocido por trabajar rápidamente. Utiliza una arquitectura de renderizado virtual eficiente que realiza un seguimiento de los componentes que necesitan ser actualizados y reduce las actualizaciones del DOM. Esto significa una aplicación web más rápida y una experiencia de usuario mejorada.
- **Componentes reutilizables**: el enfoque de desarrollo basado en componentes de Vue.js, nos ha permitido separar por "módulos" cada bloque y crear unas interfaces de usuario, que de otra forma, hubiesen resultado en una ardua tarea.
- **La documentación detallada y bien organizada de Vue.js facilita el aprendizaje y el desarrollo**. Además, ofrece a los desarrolladores una amplia gama de recursos en línea, incluidos tutoriales, videos y ejemplos de código, para ayudarlos a familiarizarse con el framework.

Estos son solo algunos de los beneficios que Vue.js ofrece. En general, es un framework robusto, adaptable y fácil de usar que se ha convertido en una opción popular.

Como se puede apreciar, Vue nos solucionó gran parte del código de JavaScript. Es un framework que cumplía con nuestras necesidades de mantener los datos "reactivos", y a la

vez, con una curva de aprendizaje más llevadera en comparación con otras alternativas como Angular o React, dado el límite de tiempo.

Hemos decidido **utilizar Vue** para facilitar la realización de las vistas en el lado del cliente, en concreto, **para la parte de la realización de los tests, que es la que más interactividad requería**.

Los componentes de Vue (y la comunicación entre sí) han permitido que hayamos desarrollado una interfaz de usuario para los tests sencilla para el usuario, a la vez que simple para nosotros a la hora de desarrollarlo. De otra forma, si lo hubiésemos hecho con JavaScript "puro", esta habría sido una ardua tarea.

2.3.- ¿Por qué utilizar Blade & Bootstrap?

Ya hemos visto qué es Blade, y, además de todas sus ventajas ya mencionadas, permite maquetar dinámicamente gracias a sus directivas @if, @foreach, @include y @yield, entre otras.

Bootstrap nos ha permitido realizar la maquetación y diseño de las páginas, ahorrándonos grandes quebraderos de cabeza en cuanto al sistema de posicionamiento de los objetos en pantalla, gracias a su sistema de rejilla (o grid), basado en 12 columnas.

¿Qué es Bootstrap?

Bootstrap es un framework de desarrollo web muy popular. Tiene muchas ventajas que lo hacen atractivo para los desarrolladores y diseñadores web. Las siguientes son algunas de las ventajas más notables de Bootstrap:

- **Responsividad**: Bootstrap ofrece un sistema de cuadrícula sensible que facilita el diseño de sitios web que se adaptan de manera automática y fluida a diferentes tamaños de pantalla. Esto garantiza la mejor experiencia de usuario en computadoras de escritorio, tabletas y dispositivos móviles.
- **Diseño consistente y profesional**: El framework proporciona una amplia gama de componentes preestilizados, como botones, formularios, barras de navegación y carruseles. Estos componentes tienen una apariencia profesional y coherente, lo que permite crear interfaces visuales atractivas y consistentes en todo el sitio web.
- **Facilidad de uso**: es bastante sencillo de usar. La documentación detallada, los ejemplos de código y las plantillas listas para usar facilitan el proceso de desarrollo.
- **Compatibilidad cruzada**: Bootstrap se ha probado en múltiples navegadores y versiones, lo que garantiza la compatibilidad cruzada. Esto ahorra tiempo y esfuerzo en el desarrollo al evitar la necesidad de realizar ajustes específicos para cada navegador.
- **Personalización y flexibilidad**: Bootstrap permite personalizar y adaptar el aspecto visual según las necesidades del proyecto a pesar de ofrecer un conjunto de estilos predefinidos. Tiene una variedad de variables y mezclas que facilitan la personalización de estilos y la creación de diseños exclusivos.

En resumen, Bootstrap ofrece un conjunto de herramientas y componentes que facilitan el desarrollo web y aseguran que el diseño sea responsivo, consistente y atractivo en varios navegadores y dispositivos. Su flexibilidad y facilidad de uso lo convierten en una opción popular para la creación de sitios web modernos y adaptables.

Hemos utilizado de las capacidades de Bootstrap para ahorrarnos código CSS, por ejemplo, en lo que se refiere a la responsividad (gracias a las clases que Bootstrap provee para ello, como col-*, col-sm-*, col-md-*, col-lg-*, y col-xl-*), para lograr esa consistencia que se persigue en todo el proyecto.

También hemos hecho uso de los componentes de Bootstrap, pero, los hemos personalizado en estilos, dentro del fichero custom.css.

En dicha hoja de estilos, tenemos también definidos los colores que se definen en nuestra guía de estilos, y muchas clases y media queries que utilizamos para poder personalizar a nuestro gusto cada elemento que se visualiza en la web, como, por ejemplo, para las animaciones, entre otras.

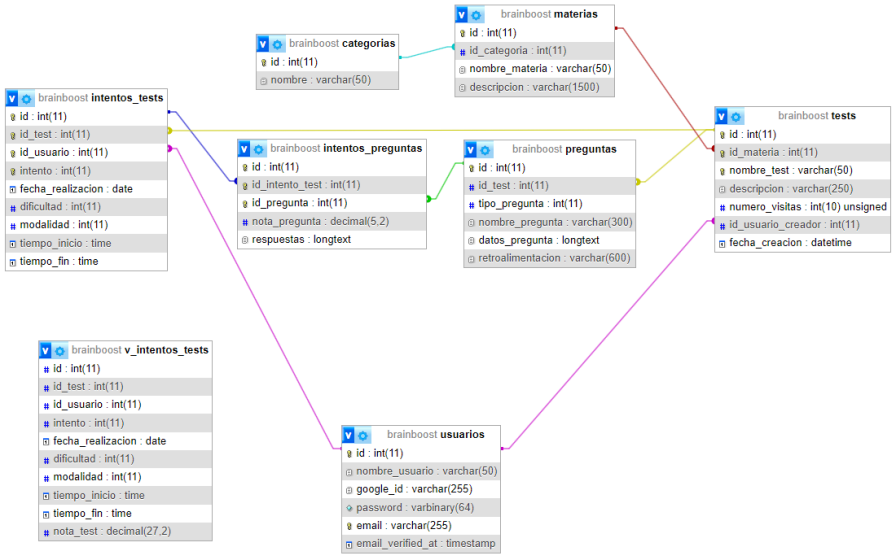
3.- Arquitectura del proyecto.

3.1.- Backend.

Para el backend se desarrollaron modelos para hacer el uso de las tablas de la base de datos, los controladores que dan funcionalidad a dichos modelos y APIs que permiten enviar y recibir la información al/desde el frontend, además de completar todo aquello que se no se hace en las rutas y controladores.

La realizacion de backend se dividio en proporcionar vistas con funcionalidad realizada en parte por backend ya que era la forma mas optima o en proporcionar informacion al front para visualizacion, manejo y guardado de datos.

3.1.1.- Modelos.



- tests: La tabla de los test almacena la información general sobre los test.
- preguntas: las preguntas se almacenan individualmente en la tabla, llevan sus claves para poder relacionarlas con el test al que están asociadas.
- intentos_test: en esta tabla se almacena el intento realizado por el usuario de realizar el test. Aquí se guarda la información general del test realizado.
- intentos_preguntas: las preguntas contestadas durante la realización de test se almacenan individualmente en esta tabla, están relacionadas con las preguntas generales tanto con el intento del test realizado.
- usuarios: aquí se guarda la información sobre el usuario, la que nos permite identificarlo inequívocamente y saber en qué estado se encuentra, registrado, verificado o si se ha autentificado usando Google.
- categorias: información sobre las categorías de materias.
- materias: tienen relación e información que permite saber la categoría de la materia y a que materia pertenece cada uno de los test, además de información sobre la propia materia.
- V_intentos_test: es la vista que agrupa la informacion de varios

3.1.2.- Vistas.

Blade juega un papel muy importante a la hora de generar las vistas.

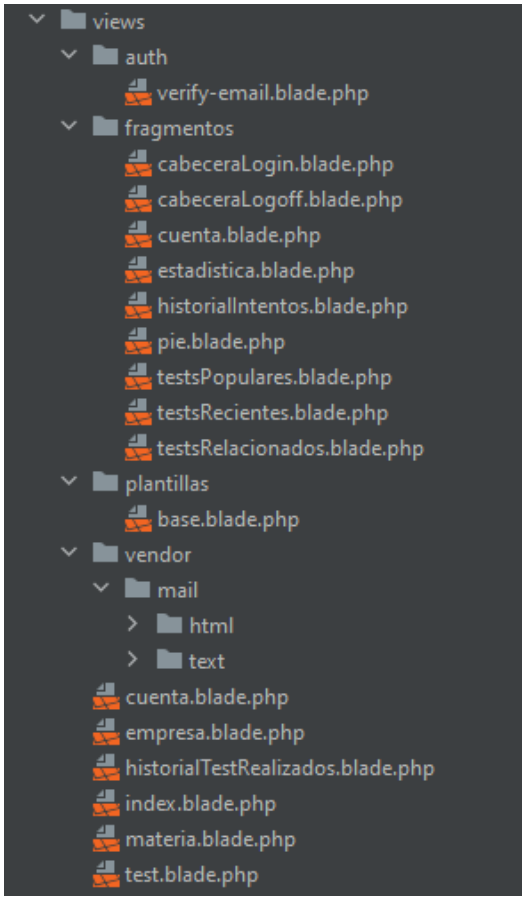
Hemos elaborado una plantilla base, sobre la que se basan el resto de vistas. Se llama base.blade.php.

Además, hemos creado diferentes fragmentos que importamos según nuestras necesidades.

- Los fragmentos de cabeceraLogin.blade y cabeceraLogoff.blade se usan para tener dos cabeceras diferentes, una para cuando el usuario no está logeado, y la otra para cuando si lo está.
- El fragmento pie de página se usa en todas las vistas así que se decidió ponerlo como fragmento para poder editar fácilmente esa parte.
- Los fragmentos de cuenta y estadística se usan para mostrar información del perfil de usuario, en concreto para mostrar las estadísticas del usuario y la posibilidad de cambiar contraseña y ver el nombre del perfil de usuario.
- Los tres fragmentos testPopulares, testRecientes, testRelacionados se refieren a los test realizados por el usuario o que se sugiere hacer al usuario. Se muestran en la pantalla principal una vez logueado el usuario.
- HistorialIntentos nos ofrece la posibilidad de tener la lista completa de los test realizados.

Respecto a las vistas, tenemos:

- Auth: en esta vista tenemos la vista relacionada con la confirmación de la cuenta de correo.
- Vendor: aquí están agrupadas todas las vistas y textos relacionados con el envió de email, como pueden ser plantillas, etc.
- Cuenta: la vista de la cuenta agrupa en si dos fragmentos, el de cuenta y el de estadistica agrupandolos en el mismo bloque.
- Empresa: proporciona la información sobre nosotros.
- HistorialTestRealizados: muestra el historial de test realizados haciendo uso del fragmento diseñado para ese fin.
- Index: nos muestra la agrupación de testPopulares, testRecientes, testRelacionados mostrando los test relacionados, realizados o sugeridos para el usuario.
- Materia: muestra la información sobre cada materia y proporciona una lista con los test asociados a dicha materia.
- Test: es la vista que permite mostrar las preguntas de los test, ya sea en el test que se está haciendo o para mostrar test realizados.



3.1.3.- Controladores.

En la aplicación BrainBoost se desarrollaron los siguientes controladores de API:

- `IntentosPreguntaController`: Gestiona las preguntas realizadas en los test, permitiendo almacenar, obtener información relacionada con estas preguntas.
- `MateriasController`: Gestiona las materias, proporcionando métodos para obtener una lista de todas las materias de una categoria.
- `PreguntasController`: Gestiona las preguntas, ofreciendo métodos para obtener una lista de todas las preguntas, mostrar las preguntas de un test específico.
- `TestController`: Gestiona los tests, brindando métodos para obtener una lista de todos los tests, mostrar un test específico.
- Estos controladores proporcionan una interfaz de programación de aplicaciones (API) eficiente y segura para administrar las preguntas de intento, las materias, las preguntas y los tests en la aplicación BrainBoost.

En la aplicación BrainBoost se desarrollaron varios controladores. En resumen son los siguientes:

- `MateriaController`:
  - `index()`: Muestra la página de una materia específica y los tests asociados a ella.
- `PreguntaController`:
  - `showFirst()`: Muestra la primera pregunta en formato JSON.

    - `showAll()`: Muestra todas las preguntas en una colección.
- `TestController`:
    - `getPopularTests($limit)`: Obtiene los tests más populares en forma de array.
    - `incrementarVisitas($id)`: Incrementa el número de visitas de un test específico.
    - `showTest(Request $request)`: Muestra un test específico en una vista o devuelve un error 404.
    - `testSugeridos($numeroTests)`: Obtiene tests sugeridos en forma de array.
- `UsuariosController`:
    - `googleAuthCallback()`: Maneja la autenticación con Google y la creación o actualización de usuarios.
    - `logintoapp(Request $request)`: Maneja el inicio de sesión en la aplicación.
    - `salir()`: Maneja el cierre de sesión en la aplicación.
    - `registrar(Request $request)`: Maneja el registro de un nuevo usuario en la aplicación.
    - `cambiarpassword(Request $request)`: Maneja el cambio de contraseña del usuario autenticado.
- `VIntentosTestController`:
    - `historialTestRealizados()`: Obtiene el historial de tests realizados por un usuario en forma de vista.
    - `ultimosTestRealizados()`: Obtiene los últimos tests realizados por un usuario en forma de array.
    - `getCuentaView()`: Obtiene la vista de la cuenta del usuario con información estadística sobre sus tests.

3.1.4.- Rutas.

Las rutas de la API en resumido son las siguientes:

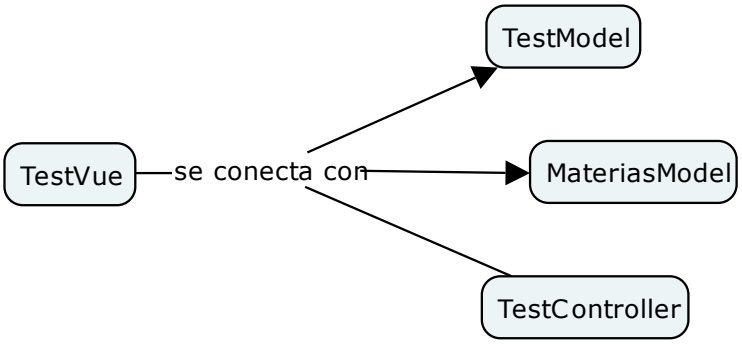
- ` **/materias** `: Devuelve la materia solicitada.
- ` **/pregunta** `: Devuelve las preguntas solicitadas.
- ` **/test** `: Devuelve el test solicitado.
- ` **/intentos-preguntas/insert** `: Guarda la información sobre el test y las preguntas realizadas por el usuario.

Las rutas web en resumido son las siguientes:

- **`/materias`:** Devuelve todas las materias.
- **`/pregunta`:** Devuelve las preguntas.
- **`/test`**: Devuelve los tests.
- **`/intentos-preguntas/insert`:** Guarda la información sobre el test y las preguntas realizadas por el usuario.
- **`/`:** Ruta principal para la página de inicio.
- **`/empresa`:** Ruta para la página de información de la empresa.

- **`/google-auth/redirect`:** Ruta para redireccionar a la autenticación de Google.
- **`/google-auth/callback`:** Ruta de devolución de llamada para la autenticación de Google.
- **`/login`:** Ruta para la página de inicio de sesión.
- **`/logintoapp`**: Ruta para el inicio de sesión en la aplicación.
- **`/registro`:** Ruta para la página de registro.
- **`/registrar`**: Ruta para registrar un nuevo usuario.
- **`/salir`**: Ruta para cerrar sesión.
- **`/email/verify`:** Ruta para la página de verificación de correo electrónico.
- **`/email/verify/{id}/{hash}`:** Ruta para verificar el correo electrónico del usuario.
- **`/email/verification-notification`:** Ruta para enviar la notificación de verificación de correo electrónico.
- **`/cuenta`:** Ruta para la página de gestión de la cuenta de usuario.
- **`/cambiarpassword`:** Ruta para cambiar la contraseña de usuario.
- **`/testhistorial`:** Ruta para ver el historial de los tests realizados.
- **`/materia/{nombreMateria}`:** Ruta para acceder a la página de una materia específica.
- **`/test/{idTest}`:** Ruta para acceder a la página de un test específico.
- **`/tests/{id}/incrementarVisitas`:** Ruta para incrementar el contador de visitas de un test.
- **`/intentos_pregunta`**: Ruta para guardar información sobre los intentos de test.
- **`/preguntas_realizadas`:** Ruta para mostrar las preguntas realizadas en un intento de test.

3.2.- Frontend.



Hemos programado unos módulos que se comunican entre sí.

- **TestModel** (en concreto, la clase Test) es la base de todos los datos que se van a almacenar en memoria durante la ejecución de nuestro frontend.
- **MateriasModel**: es donde se almacenan las materias que hay.
- **TestController** es quien recupera esos datos de la BB.DD (haciendo uso de las APIs programadas en backend) y los mete dentro del TestModel y MateriasModel.

Hemos diseñado una estructura de Frontend donde Vue se conecta con nuestros propios "módulos" programados, de forma que Vue se acopla al proyecto, dando solución exclusivamente la creación de la interfaz de usuario interactiva, dejando para TestModel y TestController las tareas de almacenar y descargar los datos que se necesitan mostrar mediante Vue.

De esta forma, hemos conseguido una de las metas que nos propusimos como grupo: la modularidad. Podríamos cambiar el framework, y seguir usando los mismos módulos. Es decir, son totalmente independiente del framework que utilicemos.

3.2.1.- TestModel.js.

Se trata de un fichero JavaScript que posee toda la lógica de validación de los datos que entran a los objetos de la clase Test, y posee los métodos getters y setters para dicha clase. Posee unos enumerados:

- **Para los tipos de pregunta**: (respecto a esto, ver el apartado final de conclusión)

```
export const TipoPregunta = {
    NONE: 0, // No se ha definido el tipo de pregunta
    MULTIPLE_RESPONSE: 1, // Múltiples respuestas, única opción correcta
    MULTIPLE_RESPONSE_MULTIPLE_CHOICE: 2, // Múltiples respuestas, de múltiple elección (varias respuestas correctas)
    UNIQUE_RESPONSE: 3, // Escribir la respuesta en una caja de texto
    FILL_IN_GAPS: 4, // Rellenar huecos (hay que rellenarlos todos)
    FILL_GAPS_GIVEN_ONE: 5, // Rellenar huecos dado uno (Ejemplo: verbos irregulares)
    FILL_TABLE: 6 // Rellenar tabla
}
```

- **Para los tipos de modalidad**:

```
export const TipoModalidad = {
    // ESTUDIAR: 1,
    PRACTICAR: 2,
    DESAFÍO: 3,
    REVISAR: 4
}
```

- **Para los tipos de dificultad**:

```
export const TipoDificultad = {
    FÁCIL: 1,
    DIFÍCIL: 2
}
```

- **Para los tipos de errores** (por ejemplo, al hacer un setter, se usan los métodos de validación implementados, y si estos no cumplen la validación, entonces se lanza una excepción con el tipo de error):

```
export const ErroresTest = [
    "__ERR_TEST_OBJECT_INVALID",
    "__ERR_TEST_ID_INVALID",
    "__ERR_ATTEMPT_TEST_ID_INVALID",
    "__ERR_QUESTION_ID_INVALID",
    "__ERR_QUESTION_INVALID",
    "__ERR_RESPONSE_INVALID",
    "__ERR_RESPONSES_INVALID", // Para cuando se pasa un array entero de respuestas del usuario
    "__ERR_TEST_NAME_INVALID",
    "__ERR_TEST_DESCRIPTION_INVALID",
    "__ERR_MARK_INVALID", // Nota de test inválida
    "__ERR_COMPLETION_DATE_TEST_INVALID", // Fecha de realización inválida
    "__ERR_END_TIME_TEST_INVALID",  // Tiempo de fin de test inválido
    "__ERR_TEST_INFO_FETCH", // Error al recuperar la información del test del servidor
    "__ERR_QUESTIONS_FETCH", // Error al recuperar las preguntas del test del servidor
    "__ERR_ATTEMPT_FETCH", // Error al recuperar la información del intento del test del servidor
    "__ERR_MODALITY",
    "__ERR_DIFFICULTY"
]
```

```
export const CodigosErrorTest = (() => {
    let codigos = Array();
    let cod = -1;

    for (let i = 0; i < ErroresTest.length; i++) {
        codigos.push(cod);
        /*-- Incrementa el código de error (en valor negativo) --*/
        cod--;
    }
    return codigos;
})();
```

```
export const MensajesErrorTest = (() => {
    let mensajes = {};
    /*-- Añade mensajes a la lista de mensajes --*/
    mensajes[ErroresTest[0]] = {
        errorName: ErroresTest[0],
        errorCode: CodigosErrorTest[0],
        message: "Se esperaba un objeto de tipo Test."
    }
    mensajes[ErroresTest[1]] = {
        errorName: ErroresTest[1],
        errorCode: CodigosErrorTest[1],
        message: "Se esperaba un id numérico mayor de 0 como id de test."
    }
    mensajes[ErroresTest[2]] = {
        errorName: ErroresTest[2],
        errorCode: CodigosErrorTest[2],
        message: "No has especificado un id de intento de test válido."
    }
    mensajes[ErroresTest[3]] = {
        errorName: ErroresTest[3],
        errorCode: CodigosErrorTest[3],
        message: "Se ha introducido un id de pregunta inválido."
    }
    mensajes[ErroresTest[4]] = {
        errorName: ErroresTest[4],
        errorCode: CodigosErrorTest[4],
        message: "Se esperaba un objeto JSON como pregunta, o bien tiene una estructura incorrecta."
    },
```

En esta última captura de pantalla, es donde se definen los mensajes para cada error. Para no extendernos mucho en la redacción de esta memoria, sugiero que sea consultado el código fuente del proyecto (fichero TestModel.js).

3.2.2.- TestController.js.

Proporciona una clase que almacena un objeto de tipo Test (del fichero TestModel.js). Sobre él, realiza las operaciones de descarga de información a través de las APIs del backend, haciendo llamadas fetch. Las operaciones de descarga son las siguientes:

- **downloadInfoAboutTestByIdTest**: descarga información sobre el test en general.
- **downloadQuestionsByIdTest**: descarga las preguntas del test.
- **sendInfoIntentoTestUsuario**: es el método encargado de enviar al servidor la información de un intento de test (es decir, las respuestas).
- **downloadInfoIntentoUsuario**: es el método encargado de recibir la información de un intento de test (es decir, descarga las respuestas de un intento de un testrealizado por un usuario).
- **aumentarVisitas**: es el método que se encarga de llamar a la API para que aumente en una visita un determinado test. Con el fin de que, en la página de inicio, aparezcan los tests más populares.

3.2.3.- MateriasModel.js.

Igual que para el caso del TestModel.js, hemos definido una serie de errores, para temas de validación de datos.

```
const ErroresMateria = [
    "__ERR_SUBJECT_ID_INVALID",
    "__ERR_SUBJECT_NAME_INVALID" // Nombre materia inválido
]

const CodigosErrorMateria = (() => {
    let codigos = Array();
    let cod = -1;

    for (let i = 0; i < ErroresMateria.length; i++) {
        codigos.push(cod);
        /*-- Incrementa el código de error (en valor negativo) --*/
        cod--;
    }
    return codigos;
})();
const CodigosErrorInversa = inversaArray(CodigosErrorMateria);
```

```
const MensajesErrorMateria = (() => {
    let mensajes = {};
    /*-- Añade mensajes a la lista de mensajes --*/
    mensajes[ErroresMateria[0]] = {
        errorName: ErroresMateria[0],
        errorCode: CodigosErrorMateria[0],
        message: `Id de materia inválido. Valores válidos: {0,${ErroresMateria.length - 1}}"`
    }
    mensajes[ErroresMateria[1]] = {
        errorName: ErroresMateria[1],
        errorCode: CodigosErrorMateria[1],
        message: "El nombre de materia especificado no existe."
    }
    //

    return mensajes;
})();
```

Con la función iniMaterias(), se descargan los datos de las materias desde la API del servidor.

3.2.4.- TestVue.js.

Aquí es donde se crea la App con Vue. He aquí los dos fragmentos del código más importantes.

```
import { createApp } from 'vue'
import TestVue from './views/main.js.vue'
import * as TestModel from "./TestModel.js";
import {TestController} from "./TestController.js";

document.addEventListener('DOMContentLoaded', () => {

    const app = createApp({
        data() {
            return {
                tiposPregunta: TestModel.TipoPregunta,
                testObj: new TestModel.Test(),
                testCtrl: null,
                url: new URL(document.location.href)
            }
        },
```

En esta captura de pantalla es donde se definen los objetos de tipo Test, y el objeto testCtrl se declara a null, porque es creado después desde el evento created() del ciclo de vida de Vue, pasándole por argumento al constructor del TestController el objeto de tipo Test.

```
app.component("testvue", TestVue)
app.mount("#appVue");
```

En esta captura de pantalla, se añade el componente TestVue (correspondiente al main.js.vue)

3.2.5.- test.blade.php*.

Aunque este fichero es del backend, aquí es donde se inyecta el código de Vue:

```
<div id="bloque-preguntas" class="row seccion-mb sombra_borde redondeado s-materia">
    <div class="col-12 seccion-mb">
        <testvue :testObj="testObj" :testCtrl="testCtrl"></testvue>
    </div>
</div>
```

en concreto, la etiqueta <testvue> importa el componente que se especificó en el app.component del apartado anterior. A continuación, en el siguiente apartado, se explica este componente (main.js.vue).

3.2.6.- main.js.vue

Aquí es donde están definidas todas las vistas de la interfaz de usuario de los tests:

```
<template>
    <eligemodalidad :testobj="testobj" :testctrl="testctrl"
        v-if="testobj.getModalidad() == null">
    </eligemodalidad>
    <!-- <preview :testobj="testobj" :testctrl="testctrl"
        v-if="testobj.getModalidad() == TestModel.TipoModalidad.ESTUDIAR">
    </preview> -->
    <eligedificultad :testobj="testobj" :testctrl="testctrl"
        v-if="testobj.getModalidad() != null &&
            //   testobj.getModalidad() != TestModel.TipoModalidad.ESTUDIAR &&
            testobj.getDificultad() == null">
    </eligedificultad>
    <dotest :testobj="testobj" :testctrl="testctrl"
        v-if="testobj.getModalidad() != null && testobj.getDificultad() != null">
    </dotest>
</template>
```

Cuando el usuario elija una modalidad, automáticamente, <eligemodalidad> desaparece, y aparece <eligedificultad>. Y cuando ambas están elegidas, desaparece <eligedificultad> y aparece <dotest>, que es la vista que muestra las preguntas para poder contestar a ellas.

Cada una tiene su propia vista, la cual puede ser consultada en el código fuente.

3.2.7.- Resto de vistas.

- **eligeModalidad.js.vue**: es la que permite al usuario elegir la modalidad (práctica o desafío).
- **eligeDificultad.js.vue**: es la que permite al usuario elegir la dificultad (fácil o difícil).
- **dotest.js.vue**: es la vista que muestra las diferentes preguntas.
- **tipo1.js.vue**: es la vista que permite mostrar una pregunta de tipo 1.
- **alert.js.vue**: componente que permite mostrar carteles de Bootstrap, con estilos del custom.css.
- **modal.js.vue**: componente que permite mostrar ventanas modales de Bootstrap. Al final no se ha hecho uso de ella.

3.2.8.- login.js

Es un mini script que se encarga de mostrar y ocultar la ventana modal de inicio de sesión/registro cuando no estamos logeados y hacemos clic en los botones de “Iniciar sesión” o “Registro”.

3.2.9.- scrollBtn.js.

Es un miniscript que se encarga de hacer funcionar el botón de subir hacia arriba cuando estamos abajo en la página. Nos referimos a este botón:

3.2.10.- api_rest.js.

Es una librería que hemos implementado para poder comunicarnos con las APIs que hemos desarrollado. El método más usado es el obtenerJSON. Dicho método, dada una URL, un método, unas cabeceras HTTP (headers) y unos datos (que se incorporan al body de la petición), el método hace todo el trabajo por nosotros para comunicarnos con el servidor.

3.2.11.- json.js.

Es una pequeña librería pensada para utilidades relacionadas con los JSON. Para el caso de uso en nuestro proyecto, solo hemos necesitado implementar un método: objToJSON, que permite convertir un objeto de cualquier clase en JSON.

3.2.12.- globals.js.

Este fichero es muy importante de cara al despliegue de la aplicación. Contiene las rutas de la API, y las rutas del servidor donde se quiera alojar.

```
/*==================================
        ENUMERADOS
  ==================================*/
/**
 * Enumerado de modos disponibles de aplicación
 */
const ModeAppEnum = {
    PRODUCTION: 1,
    LOCALDEBUG: 2
}

const ModeAppDirecciones = {
    // 1: "https://www.clinicadentalsanandres.com/BrainBoostLaravel/brainboost/public", // PRODUCTION
    1: "https://brainboost.es", // PRODUCTION
    2: "http://brainboost.com" // LOCALDEBUG Santi
    // 2: "http://127.0.0.1:8000" // LOCALDEBUG Eugenio
    // 2: "" // LOCALDEBUG Juan Carlos
}

/*==================================
          CONSTANTES
  ==================================*/
const modeApp = ModeAppEnum.PRODUCTION; // Flag a cambiar
```

Recomendamos cambiar la constante modeApp a PRODUCTION, y en la constante ModeAppDirecciones, en el 2, es necesario poner el servidor donde se aloje nuestro proyecto. Esto es fundamental para que funcionen bien las llamadas a las rutas de las APIs.

Además, posee un método calcularRuta, que genera rutas relativas en función de la ruta pasada por parámetro (que será la ruta actual en la que se encuentre el usuario navegando en este momento), para poder calcular las rutas relativas hacia las APIs.

3.2.13.- ¿Dónde se importan todos estos scripts?

Estos scripts se importan dentro de la plantilla base de Blade (fichero base.blade.php). A continuación, se muestran fragmentos de este fichero:

```
<!-- JavaScript: Cabecera login -->
<script src="{!! asset('js/login.js') !!}"></script>

<!-- JavaScript: páginas en general -->
<script src="{!! asset('js/main.js') !!}"></script>

<!-- Boton Scroll up -->
<script src="{!! asset('js/scrollBtn.js') !!}"></script>
```

```
<!-- JavaScript: lógica test -->
@isset($enableScriptTest)
    @if ($enableScriptTest == true)
        <script src="{!! asset('js/utilidades.js') !!}"></script>
        <script src="{!! asset('js/globals.js') !!}"></script>
        <script src="{!! asset('js/JSON/api_rest.js') !!}"></script>
        <script src="{!! asset('js/JSON/json.js') !!}"></script>

        <script src="{!! asset('js/MateriaModel.js') !!}"></script>
        {{--
        <script src="{!! asset('js/TestModel.js') !!}"></script>
        <script src="{!! asset('js/TestController.js') !!}"></script> --}}
        {{-- <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script> <!-- Importa la librería de Vue.js --> --}}
        {{-- <script src="{!! asset('js/TestVue.js') !!}"></script> --}}
        @vite('public/js/TestVue.js')
    @endif
@else
    {{-- @vite() --}}
@endisset
```

Como se puede apreciar en esta última captura (las importaciones de scripts que hay comentadas), ya no es necesario importar desde aquí TestModel, TestController ni la librería de vue, ni TestVue, porque ahora Vue está instalado como un paquete npm, y se han hecho las importaciones y exportaciones de JavaScript correspondientes.

Por otra parte, TestVue.js se importa mediante la librería Vite, hace de intermediario entre Laravel y Vue.

4.- Librerías.

4.1.- Composer.

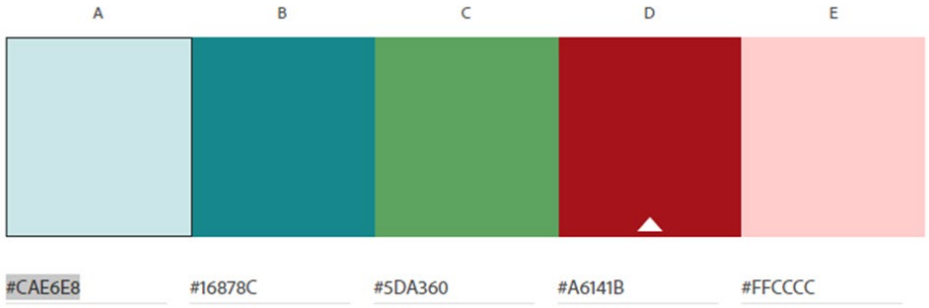
Las librerias que se han usado para el correcto funcionamiento de la aplicación son las sigueintes:

1. guzzlehttp/guzzle: Una biblioteca para hacer solicitudes HTTP de manera sencilla en PHP.
2. laravel/framework: El framework Laravel, utilizado para desarrollar aplicaciones web robustas y escalables.
3. laravel/sanctum: Proporciona un sistema simple y liviano para autenticación de API basado en tokens.
4. laravel/socialite: Permite la autenticación de usuarios a través de proveedores de servicios de terceros como Google, Facebook, etc.
5. laravel/tinker: Proporciona una consola interactiva para interactuar con la aplicación Laravel.
6. laravel/ui: Ofrece características de interfaz de usuario prediseñadas, como autenticación y scaffolding, para aplicaciones Laravel.
7. barryvdh/laravel-ide-helper: Ayuda a mejorar el soporte de IDE al generar archivos de ayuda para Laravel.
8. fakerphp/faker: Una biblioteca para generar datos de prueba falsos, como nombres, direcciones, etc.
9. laravel/pint: Proporciona una API fluida para formatear y manipular valores numéricos.
10. laravel/sail: Ofrece un entorno de desarrollo local ligero basado en contenedores Docker para aplicaciones Laravel.
11. mockery/mockery: Un framework de simulación de objetos para pruebas unitarias en PHP.
12. nunomaduro/collision: Proporciona una interfaz de línea de comandos mejorada con información detallada sobre excepciones.
13. phpunit/phpunit: Un marco de pruebas unitarias para PHP.
14. spatie/laravel-ignition: Proporciona una herramienta de depuración y seguimiento de errores para aplicaciones Laravel.

4.2.- Librerías para Vue (Vite).

5.- Guía de estilos.

Para el aspecto visual de la aplicación hemos elegido unos colores apagados para que sean más acogedores a la vista y no carguen tanto los ojos cansados de los estudiantes.



De aquí usamos todos.

- El color A se usa para definir los bloques dentro de la web.
- El color B se usa para definir bloques dentro de otros bloques mas globales y resaltar algunos botones. Hay veces que se intercambia con el A, siendo B el color global y A el local.
- El color C se usa para resaltar información, por ejemplo, en el menú de navegación va cambiando según este el ratón encima. O algún punto de menú cambia a ese color cuando esta seleccionado con el ratón.
- Color D es color principal del Logo.
- Color E es color secundario del Logo.
- En el logo se usan también el color B y C para resaltar las dos palabras.

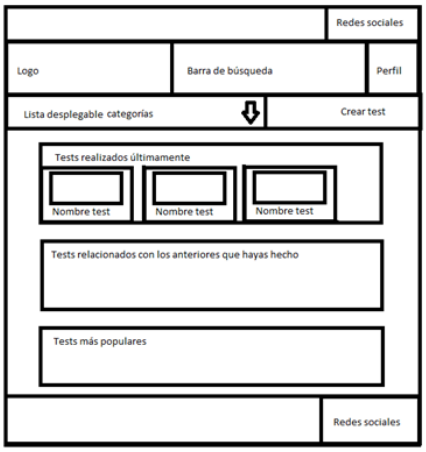
El logotipo del proyecto se compone de dos elementos, el icono en color rojo para destacar y de las palabras Brain Boost que vienen a continuación.

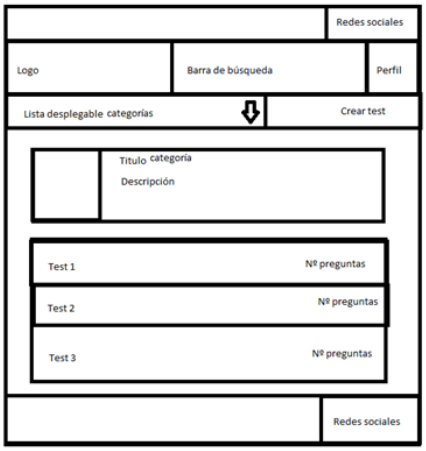
También hemos desarrollado otro icono que puede ser más adecuado para usarlo solo. O por ejemplo como icono de la aplicación móvil futura o icono de la página web.

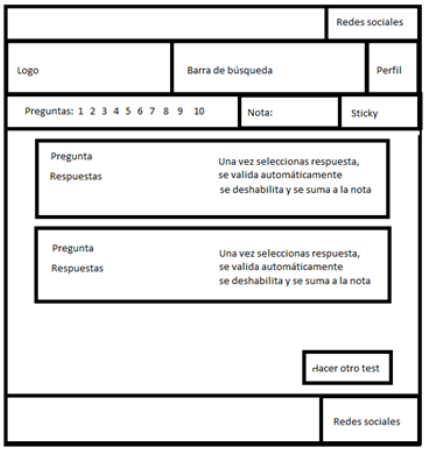
Se eligió el siguiente tipo de tipografía para los textos.

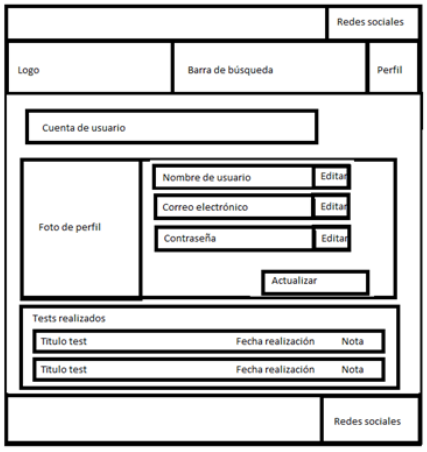
Tipo de texto	Tipo de fuente
Texto normal	Nunito
H2 y H3	Nunito color: #16878C
H1	Nunito color: #16878C

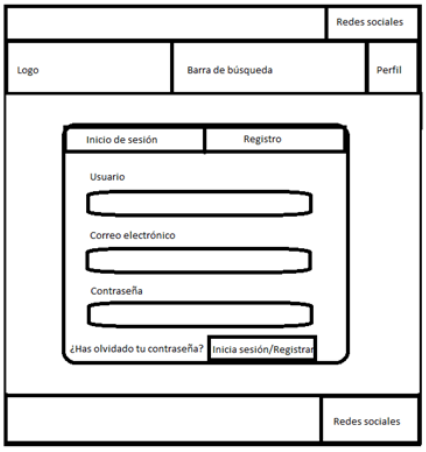
Para tener una idea inicial de la disposición del proyecto se realizó el siguiente wireframe.











6.- Dificultades encontradas.

- **Al principio, dudábamos si utilizar frameworks, y estuvimos bastante tiempo así**. Hasta que no estuvimos seguros al 100%, habiéndonos documentado previamente, estuvimos desarrollando la BB.DD y la lógica del TestModel y TestController, ya que eso no requería de ningún framework.
    - Cuando decidimos utilizar los frameworks, todas las tareas que se hacían con PHP puro, se facilitaron enormemente gracias a esta decisión de utilizar Laravel.
- **Hemos tenido las típicas dificultades al realizar commits, push, pull y merge en el repositorio de GitHub**. Cuando trabajábamos varios en un mismo fichero, había que tener cuidado para no sobrescribir el trabajo del otro. Sin embargo, el trabajo en equipo nos ha permitido coordinarnos para evitar, en la medida de lo posible, que esto sucediera.
- **Respecto al sistema de autenticación (Login)**: La implementación de tablas no predeterminadas supuso la modificación de los métodos de registro y del login.
- **Respecto al sistema de autenticación vía Google Auth**: La dificultad alandida de este tipo de autentificación es la gestión de una API externa que hay que usar según sus reglas, como por ejemplo la restricción de uso desde dominios no autorizados, por eso dicha forma de autentificación no funcionara en nuevos entornos desplegados hasta que no se reconfigure por completo. Tanto la parte de Laravel, como la parte de la API en el control de Google.
- **La configuración del entorno del servidor**, haciendo funcionar todo en conjunto salvando los problemas que surgieron sobre todo en la compatibilidad de diferentes versiones de diferentes servicios como puede ser la versión de MariaDB con phpMyadmin o Php 8.2 con phpMyadmin.
- **La realización primero de la base de datos y después la adaptación de los modelos provoco que una parte del trabajo "automático" que facilita Laravel mediante migraciones se tuviera que hacer y realizar a mano, con su correspondiente penalización en tiempo**. Sin quitar que, al no tener dichas funcionalidades de Laravel usadas, se ha complicado en parte el uso de otras posibilidades como autentificación con Google, el registro de usuario, etc. Haciendo necesario realizar métodos más complejos.
- La realización del TestModel.js ha sido realmente desesperante, y quizás si desde el principio hubiésemos pensado hacer el frontend con Vue, se hubiese planteado de otra manera. Sin embargo, lo que parece una desventaja, puede no serlo, ya que ahora tenemos un TestModel que sirve para cualquier framework.
- **Respecto a Vue**: pese a su fácil curva de aprendizaje, nos hemos topado con que gran parte de la documentación que hay en Internet, es para versiones anteriores de Vue. Sin embargo, hemos podido hacerle frente utilizando la documentación oficial, la cual es muy buena, y apoyándonos en tutoriales más nuevos.
- **Sobre la creación de la Base de datos**: Hemos tenido que ir haciendo sucesivos cambios en la estructura inicial para ir incorporando información y funcionalidad que ha surgido sobre la marcha en el desarrollo del proyecto que, inicialmente, no se había tenido en cuenta o no era necesario según nuestro diseño inicial de la aplicación.

- **En el tema del diseño visual**: Decir que ha sido un gran problema ir acomodando cada uno de los bloques, aún haciendo uso de clases Bootstrap, para que todo encaje y no se pierda la estructura en cada una de las páginas. Más aún cuando se van incorporando bloques, secciones, botones y demás de forma dinámica en la plataforma.

7.- Conclusión del proyecto: nuestras sensaciones.

En conclusión, podemos decir que no ha sido un proyecto fácil de realizar. Nuestro trabajo en este proyecto ha sido una experiencia enriquecedora que nos ha permitido aplicar nuestros conocimientos en programación y desarrollo de aplicaciones web. Hemos superado muchos desafíos y obstáculos a lo largo del camino, pero gracias a nuestro trabajo en equipo y perseverancia, hemos logrado superarlos y alcanzar los hitos que nos propusimos en el anteproyecto.

El trabajo en equipo ha sido esencial para el éxito de este proyecto. Establecimos una serie de roles de forma general: Juan Carlos se encargó del diseño (con CSS) y maquetación (con Blade) de la mayoría de las vistas y pobló la base de datos; Eugenio se encargó de toda la parte de despliegue, y el backend (controladores, rutas, APIs, etc.); y Santiago se encargó de dar vida a la lógica de la realización de los tests con Vue, además de pequeños arreglos en JavaScript.

No podemos negar que ha habido dificultades en el camino. Nos enfrentamos a problemas técnicos, decisiones de diseño e imprevistos que requerían soluciones ingeniosas y colaboración. Sin embargo, cada reto superado ha sido una oportunidad de aprendizaje y crecimiento tanto individual como grupal. Hemos adquirido nuevos conocimientos técnicos, hemos mejorado nuestras habilidades de resolución de problemas y hemos aprendido cómo adaptarnos a las circunstancias cambiantes.

A lo largo de este proyecto, se ha hecho evidente el valor del trabajo en equipo. Hemos aprendido cómo trabajar juntos, saber escuchar a los demás y brindar apoyo mutuo pueden mejorar nuestras propias habilidades y producir resultados de mayor calidad. Además, hemos aprendido que la mejor manera de resolver los conflictos es de manera constructiva, llegando a un acuerdo y tomando decisiones consensuadas para avanzar de manera efectiva en el proyecto.

Como todo proyecto, se puede mejorar, y dejamos algunas de nuestras ideas expuestas en la conclusión para su implementación post-presentación del proyecto.

- En la BB.DD hemos metido algunas preguntas de otros tipos, que no solo son marcar una respuesta. Se puede implementar lógica en Vue para dichos tipos de preguntas, y hacer tests que no sean todo el rato 4 opciones de respuesta, una única respuesta correcta. (De ahí que en el TestModel hayamos dejado definidos dichos tipos de pregunta).
- Hacer que los usuarios puedan crear sus propios tests, y puedan compartirlos con amigos.
- Implementar un sistema para poder recuperar la contraseña de una cuenta, en caso de que el usuario la pierda (aunque es verdad que iniciando sesión con Google, se podría seguir entrando).

En resumen, hemos aprendido valiosas lecciones de este proyecto y hemos mejorado nuestras habilidades técnicas. Hemos demostrado que la cooperación, la perseverancia y la organización nos permiten alcanzar nuestros objetivos a pesar de los desafíos y obstáculos.

Estamos orgullosos del trabajo que hemos hecho y confiamos en que las habilidades y el conocimiento que hemos aprendido serán muy útiles en nuestro futuro profesional.