2021-04

SISTEMA DE GESTION DE ARTICULOS, MURO DE DESAFIOS Y GESTION DE USUARIOS PARA RED SOCIAL DEL MINISTERIO DE SALUD

AGUILERA BADILLA, OSCAR

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

"SISTEMA DE GESTIÓN DE ARTÍCULOS, MURO DE DESAFÍOS Y GESTIÓN DE USUARIOS PARA RED SOCIAL DEL MINISTERIO DE SALUD"

ÓSCAR AGUILERA BADILLA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

PROFESORA GUÍA: MARÍA JOSÉ ESCOBAR

PROFESOR CORREFERENTE: AGUSTÍN GONZÁLEZ

Abril, 2021

Resumen

El ministerio de salud, con el ministro Emilio Santelices Cuevas anuncia el año 2018 comenzar una modernización de los procesos de atención y una inclusión de la ciudadanía en la creación de proyectos e ideas que aporten a mejorar el sistema de salud chileno. En este contexto, se crea el sitio web Hospital Digital el cual contiene, entre otros servicios, un espacio de interacción social entre profesionales de la salud y las personas. Esta plataforma web es bautizada como Salud Mejor y está orientado a entregar información específica sobre las condiciones de salud más comunes y una construcción de desafíos a través de un muro de publicaciones y comentarios. Este módulo interactiva es construido tomando como ejemplo las redes sociales más utilizadas: Facebook, Linkedin y Twitter.

Este documento muestra el desarrollo de la red social Salud Mejor, el cual contiene un módulo para gestionar la creación de artículos sobre salud pública, un muro de publicaciones y comentarios para la interacción entre profesionales de la salud y ciudadanos, un sistema de clasificación de publicaciones mediantes botón "me gustaz un gestor de organizaciones y usuarios.

En la realización de este proyecto se utiliza el framework de PHP Laravel para programar el servidor, GraphQL como lenguaje de consultas y las librerías de JavaScript ReactJS para la interfaz gráfica del cliente junto con HTML y con el lenguaje de diseño de hojas de estilo CSS. Todo esto está montado en una aplicación web en la nube de Azure capaz de soportar un gran flujo de consultas.

Para que el ministerio de salud aprueba este proyecto es necesario tener un diseño robusto en la base de datos, ordenando las relaciones de manera que la reacción del servidor sea rápida y se debe cumplir la condición de que el sitio sea responsivo, es decir, que pueda ser visualizado sin ningún problema desde cualquier dispositivo móvil o computador.

Abstract

The Ministry of Health, together with Minister Emilio Santelices Cuevas, announces in 2018 the beginning of a modernization of the care processes and the inclusion of citizens in the creation of projects and ideas that contribute to improving the Chilean health system. In this context, the Hospital Digital website is created which contains, among other services, a space for social interaction between health professionals and people. This web platform is baptized as Better Health and is oriented to deliver specific information on the most common health conditions and a construction of challenges through a wall of publications and comments. This interactive module is built taking as an example the most used social networks: Facebook, Linkedin and Twitter.

This document shows the development of the Salud Mejor social network, which contains a module to manage the creation of articles on public health, a wall of publications and comments for the interaction between health professionals and citizens, a classification system of publications by means of a "like"button and a manager of organizations and users.

In carrying out this project, the PHP Laravel framework is used to program the server, GraphQL as the query language and the JavaScript ReactJS libraries for the client's graphical interface together with HTML and the CSS style sheet design language. All of this is assembled in an Azure cloud web application capable of supporting a large flow of queries.

For the Ministry of Health to approve this project, it is necessary to have a robust design in the database, ordering the relationships so that the reaction of the server is fast and the condition that the site is responsive must be met, that is, that can be viewed without any problem from any mobile device or computer.

Glosario

- Backend: El código de programa que se ejecuta desde el lado del servidor, en PHP en este caso.
- Frontend: El código que se ejecuta en el navegador que actúa como cliente. Es con lo que el usuario final se encuentra y usa HTML, CSS y ReactJS.
- Consulta: Método para acceder a una base de datos, la cual puede crear, modificar, borrar y obtener datos. Éstas se realizan con un lenguaje de manipulación de datos, por ejemplo SQL.
- Lógica de negocio: Parte de un sistema que codifica su relación con el exterior, determinando la manera en que la información puede ser creada, almacenada y cambiada.
- Persistencia: Acción de preservar la información de un objeto de forma permanente y de su recuperación para ser leído.
- Clave foránea: Se refiere a una columna de las filas de una tabla de base de datos que relaciona una fila con otra fila de otra tabla con una única clave primaria.
- Clave primaria: Es una columna en una tabla de base de datos cuyo valor identifica de manera única y exclusiva a una fila de la tabla. Generalmente se usa el identificador *id* para esto.
- REPL: Del inglés **Read-Eval-Prin-Loop**, bucle de lectura, evaluación, impresión. Es una consola de lenguage de alto nivel interactivo, como por ejemplo la que entrega Python o Tinker en el caso de Laravel.
- ORM: del inglés **Object Relational mapping**, es una técnica de programación para convertir el sistema de datos utilizado en la programación orientada a objetos al sistema utilizado en las bases de datos relacionales. Creando una base de datos

orientada a objetos virtual entregando las posibilidades de uso de esta orientación.

- API REST: Interfaz de programación de aplicaciones que utiliza la arquitectura de Transferencia de estado representacional (Representational State Transfer).

- HTTPS: Protocolo seguro de transferencia de hipertexto. Seguridad de la capa de transporte del sistema OSI que tiene como objetivo encriptar la información entre el cliente y el servidor.

- DOM: Del inglés (Documento Object Model), es la interfaz de programación para los documentos HTML que define de qué maneras los programas pueden acceder para modificar su estructura, estilo y contenido.

- Responsivo: Característica del front-end que garantiza que la visualización de la interfaz de usuario va a responder a los distintos tamaños de las pantallas de los dispositivos usados en la visualización. Acomodando los elementos para que sea cómo leer y entender lo mostrado.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El año 1997 nace SixDegrees.com[1] [2] la web que es considerada la primera red social del mundo y que lleva su nombre por estar basada en la teoría de los seis grados se separación del psicólogo Stanley Milgram. Desde aquí en adelante la red informática mundial (WWW)[3] crece rápidamente y cada vez hay más personas interconectadas por un computador, un teléfono inteligente u otros dispositivos electrónicos. El compartir información se hace cada vez más normal y con las redes sociales como Facebook, Linkedin, MySpace, YouTube, etc. se consolidan las distintas maneras de hacerlo.

Estas redes sociales están enfocadas en tener contacto con otras personas de manera entretenida y dinámica, permitiendo a sus usuarios manejar una cierta cantidad de componentes para compartir sus fotos, vídeos, comentarios, mensajes, etc. Sin embargo, la necesidad de conectar y generar redes sociales comienza poco a poco a llenar otros espacios, por ejemplo, el del mundo laboral, empresarial, relaciones amorosas, oferta de artistas, etc.

Esta forma de comunicación comienza inevitablemente a ser necesaria y buscada en universos más formales no cercanos a los ejemplos antes mencionados. Los servicios de salud, las instituciones gubernamentales, educacionales, funcionamiento interno de las empresas, etc. comienzan a buscar la manera de generar redes a través de los nuevos conceptos y paradigmas que van apareciendo.

Por otra parte, el MINSAL, con el ministro Emilio Santelices, generan el año 2018

[1] https://en.wikipedia.org/wiki/SixDegrees.com
[2] https://culturacientifica.com/2020/12/01/historia-de-la-primera-red-social/
[3] https://es.wikipedia.org/wiki/World_Wide_Web

un plan estratégico en el cual se define la necesidad de conectar más rápidamente a los ciudadanos con las instituciones de salud pública. En palabras del ministro: *"El proyecto Hospital Digital permitirá generar una cartera de servicios hacia y desde hospitales y también atención primaria, integrando los distintos niveles de complejidad de tal forma que un paciente pueda acceder a un diagnóstico desde su centro de salud familiar y ser informado en el Hospital Digital. El otro gran eje, es que toda la información desde los Centros de Salud convergerán en el Hospital Digital y desde ahí serán depositados en la plataforma de datos del Departamento de Inteligencia Sanitaria del MINSAL, para ser explotados mediante metodologías de Machine Learning (Inteligencia Artificial)(07-09-2018)"*[4][5][6]

Es entonces que en el contexto de este proyecto nace uno de sus servicios: **Salud Mejor**. El que pretende utilizar el paradigma de las redes sociales para conectar a los ciudadanos con profesionales de la salud y generar un repositorio de información y discusión amparada por el MINSAL.

Por todo lo anterior es que se desarrolla en esta memoria de título la creación, modelado e implementación de esta red social que permita al MINSAL de Chile entregar un nexo entre los servicios de salud pública y las necesidades de la gente, a través de la presentación artículos y un muro desafíos que motiven la conversación abierta entre profesionales de la salud y los ciudadanos.

1.1. Objetivos

A continuación se describen los objetivos del proyecto, los antecedentes que contextualizan la labor, el enfoque empleado para realizarla y las herramientas utilizadas.

1. Crear una plataforma web para la exposición de artículos sobre condiciones de salud con información oficial del MINSAL.

2. Crear un muro de publicaciones para que lo usuarios de la plataforma puedan compartir sus opiniones y desafíos respecto a los artículos publicados por el MINSAL de Chile.

3. Crear un sistema de publicaciones, comentarios y "me gusta"que permita a profe-

[4]https://ehealthreporter.com/web/es/noticia/como-sera-el-hospital-digital/
[5]https://www.hospitaldigital.gob.cl/hospital-digital/que-es-hospital-digital
[6]https://www.t13.cl/noticia/nacional/hospital-digital-alcanzara-19-millones-atenciones-anuales-asi-funcionara

sionales de la salud, funcionarios de la red de salud pública y ciudadanos compartir conversaciones, ideas, proyectos, necesidades e inquietudes.

4. Crear un sistema jerárquico que permita tener usuarios con distinto nivel de permisos. Entregando herramientas administrativas para grupos de usuarios, grupos de comentarios, grupos de publicaciones y artículos oficiales.

5. Crear una sistema de registro de instituciones de la salud que permita asociar a usuarios de la aplicación con estas y que entregue información importante sobre la organización y sus funcionarios.

6. Crear un sistema de inicio de sesión con redes sociales Facebook y Google. De manera que se pueda monitorear el uso de la plataforma y entrega datos al MINSAL.

1.2. Antecedentes del trabajo de título

El comienzo de este trabajo parte de la necesidad del MINSAL de crear una red social en la que exista interacción entre pacientes y profesionales donde se discutan desafíos respecto a algunas temáticas específicas. De las reuniones realizadas entre los grupos de innovación y diseño del MINSAL y los técnicos de ingeniería y diseño de la empresa Kauel se genera una maqueta de las interacciones esperadas y la interfaz gráfica del sitio dando inicio a la construcción de la plataforma según los requerimientos del MINSAL.

1.3. Enfoque empleado

Para el enfoque de desarrollo se trabaja en una arquitectura cliente - servidor como es lo habitual en el desarrollo web. Ejecutando por separado lo que se refiere a la base de datos necesaria para sustentar las consultas al sitio y la interfaz gráfica con la lógica servida en los navegadores de los usuarios.

Las directrices del diseño son la optimización de los recursos por parte del cliente y la disminución del tiempo de consulta por parte del servidor. El sitio debe poder ser ejecutado en un computador de escritorio, portátil o cualquier dispositivo móvil y tener una simple relación entre las tablas de la base de datos. Todo esto requiere de una programación modular, orientada a objetos, con mapeo objeto-relacional (ORM)

y responsiva.

1.4. Herramientas utilizadas

Para realizar la plataforma se utiliza el entorno de trabajo *Laravel* de código abierto para el desarrollo de aplicaciones y servicios web con PHP. Basado en el patrón de arquitectura de software MVC *Modelo - Vista - Controlador*, con Composer como su gestor de paquetes y Eloquent como el ORM base. Para la gestión de las consultas se utiliza GraphQL. Un lenguaje de consultas que usa un sistema de tipos, campos y argumentos que reemplaza a la filosofía de API REST, desarrollado por Facebook.

Por la parte del cliente, se utiliza el lenguaje de programación orientado a objetos JavaScript con una librería creada por Facebook llamada ReactJS, la cual está basada en componentes donde, cada elemento en la interfaz gráfica puede ser estructurado como si fuera una etiqueta HTML con propiedades y estados a través del uso de clases.

1.5. Alcances

Los alcances de este trabajo de título comprenden:

- Facilitar la entrega de información oficial del MINSAL a pacientes sobre las condiciones más comunes e importantes a nivel nacional.
- Entregar un sistema de creación de artículos de salud en una plataforma web.
- Facilitar el intercambio de comentarios entre profesionales de la salud y pacientes a nivel nacional.
- Entregar un sistema de publicaciones tipo muro con comentarios y "likes".

Capítulo 2

Planteamiento del diseño

A continuación se explican los detalles de los requerimientos realizados por el MINSAL para la plataforma,

2.1. Problema a solucionar

El MINSAL de Chile plantea la necesidad de crear una plataforma web donde puedan desarrollar temas de salud presentando artículos específicos sobre condiciones y enfermedades. Junto a esto se requiere un muro de publicaciones para que profesionales de la salud y personas puedan interactuar al respecto de estos artículos, planteando desafíos, respondiendo comentarios y dando "likes".

Para esto se requiere el desarrollo de un sistema que contenga un formulario para la creación de artículos, un sistema de gestión de usuarios para entregar permisos y editar información, un sitio donde se listen los artículos y puedan ser mostrado y, finalmente un sitio dinámico con un muro de publicaciones para el desarrollo de las discusiones.

Otro de los requerimientos importantes es que las personas puedan acceder con sus redes sociales de Facebook para poder realizar sus publicaciones en el muro de desafíos.

2.2. Estado del arte

En lo que respecta a redes sociales destinadas a la interacción entre personas con condiciones médicas, que deseen mejorar el sistema de salud a través de la interacción con personal profesional, no existen nada parecido. La única red social encontrada que está orientada a la salud es Health Unlocked [1], la cual está enfocada en la creación de comunidades que ayuden a pacientes a entender su enfermedad a través de la experiencia de otros. En la figura 2.1 podemos observar el muro principal de los servicios de esta red social, en donde se observan publicaciones de personas que quieren compartir su condición, ser leídos y obtener compañía a través de las respuestas. Este muro es muy similar al funcionamiento que se pide, sin embargo, al momento de entregar la solicitud, las personas del MINSAL nombraban como ejemplo el diseño que tiene la red social laboral Linkedin como referencia.

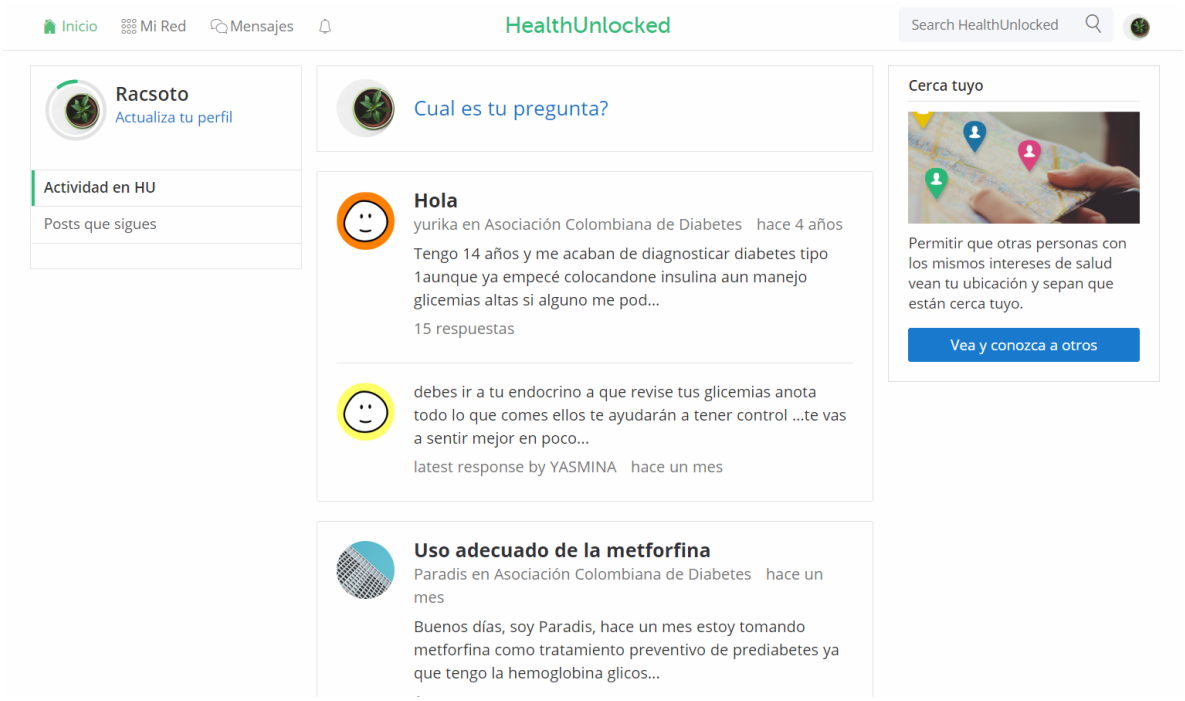

Figura 2.1: Muro principal de plataforma Health Unlocked

2.3. Arquitectura de software

Para la realización de este proyecto se utiliza la arquitectura de software cliente-servidor en la cual las tareas se reparten entre los proveedores de recursos y servicios (servidor), y los demandantes (clientes). Un cliente, a través de una conexión a internet realiza una petición HTTPS al servidor que responde con los contenidos y recursos necesarios. En la imagen 2.2 se grafica este modelo.

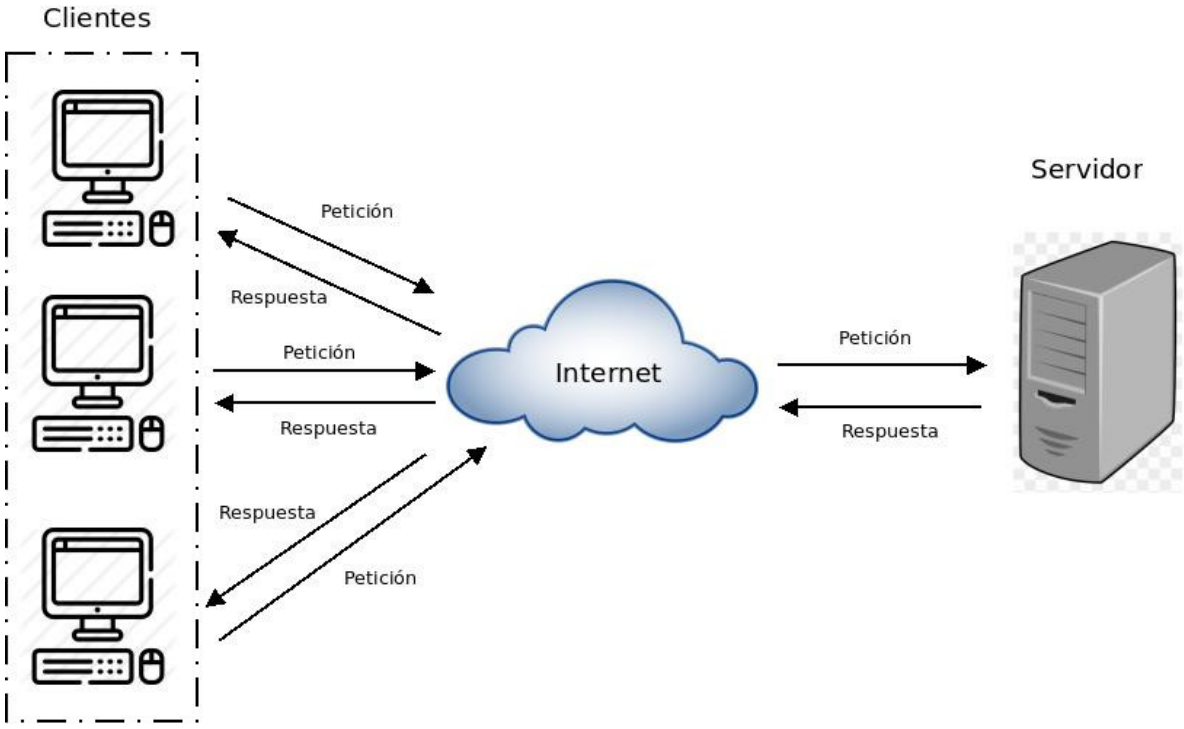

Figura 2.2: Diagrama de arquitectura cliente - servidor

El servidor es quien maneja el flujo de información accediendo a una base de datos, filtrando las consultas a través de *middlwares* y respondiendo según las peticiones del cliente. A todo este concepto le llamaremos **Back-End**. Por otra parte, el cliente podría ser un navegador web en un computador de escritorio, un teléfono inteligente o un software de escritorio, el cual muestra al usuario una interfaz gráfica amigable e intuitiva. A este concepto le llamaremos **Front-End**.

2.4. Estructura del servidor Back-End

El servidor es quien recibe peticiones de un usuario a través de un cliente, las cuales son respondidas en la manera que se cumplan las condiciones se seguridad y estén dentro del conjunto de servicios entregados. El encargado de discriminar si esta petición será procesada o no es llamado *middlware*. En general, es un programa que se encarga de las tareas de gestión de datos, servicios de aplicaciones, mensajería, autenticación y gestión de API. Una vez que se pasa por este filtro, el servidor ejecuta la acción necesaria para generar una respuesta. La imagen 2.3 muestra la lógica del servidor a implementar, el cual tiene que actuar según si el usuario está o no con una sesión de autenticación entregada por facebook para así acceder a ciertas partes de la base de datos, de la API o a los archivos almacenados.

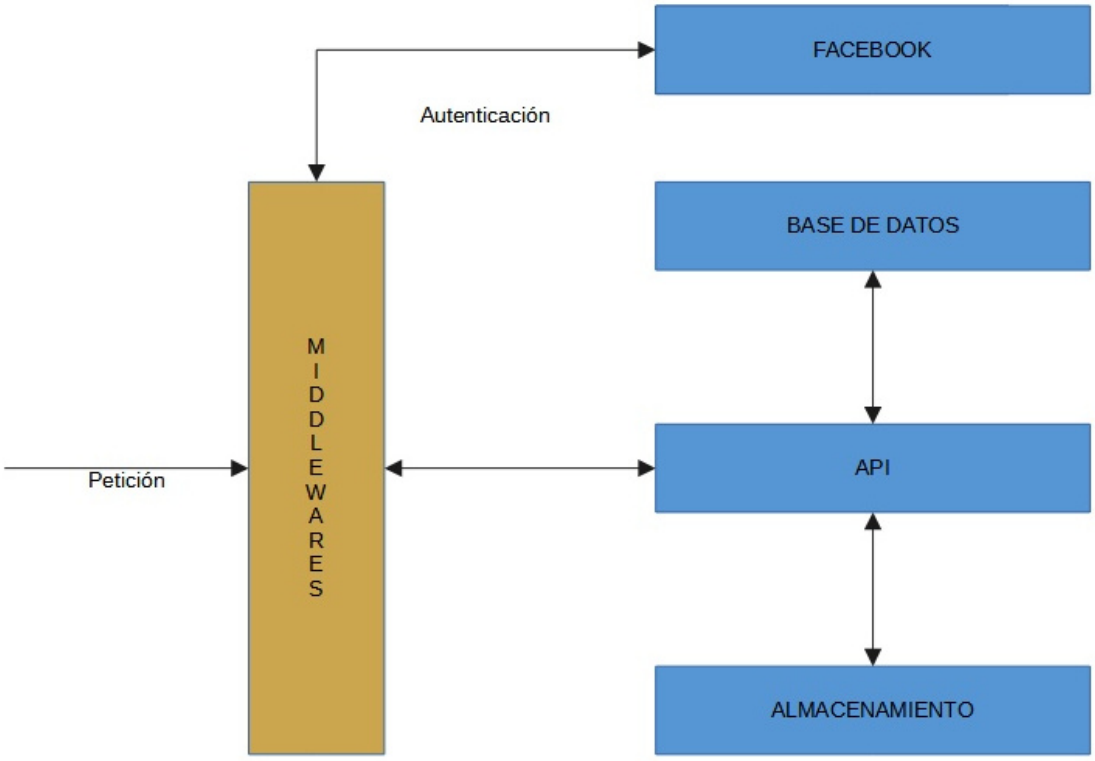

Figura 2.3: Modelo interno del servidor

2.4.1. Sistema de autenticación

El proyecto tiene como requerimiento el uso de las credenciales de Facebook y de Google para poder utilizar ciertos componentes de la aplicación web. Para esto se utiliza *OAuth 2.0* [1], un estándar (Framework) de autorización abierto para APIs, nos permite utilizar la autenticación de terceros para acceder a un cliente en específico. En el proceso encontramos diferentes roles que participan como muestra la figura 2.4:

- Dueño del recurso: Corresponde al usuario que da autorización a un aplicación de terceros, en este caso Facebook o Google, para acceder a su cuenta y realizar algunas acciones en su nombre.

- Cliente: Es la aplicación a la que se quiere tener acceso con las credenciales de terceros.

- Servidor de autorización: Responsable de gestionar las peticiones de autorización, verificando la identidad y emitiendo una serie de tokens de acceso a la aplicación cliente.

- Servidor de recursos protegidos: Sería la API de nuestra aplicación, la cual aloja los recursos a los que el usuario quiere acceder con sus credenciales de terceros.

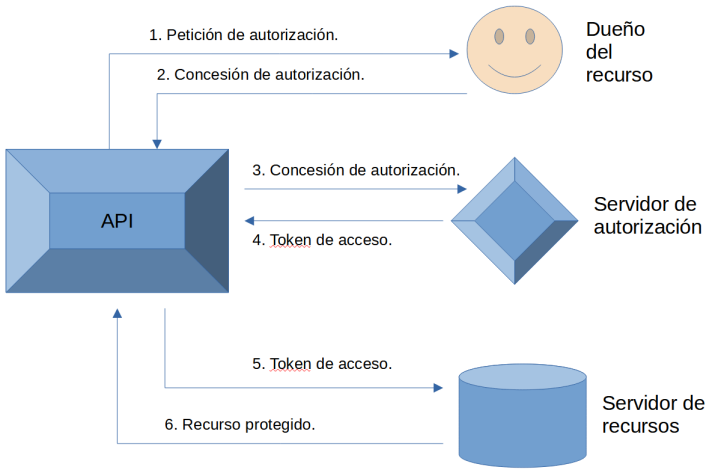

Figura 2.4: Modelo de autenticación con OAuth2.0

[1] https://oauth.net/2/

2.4.2. Diseño de la base de datos *Back - End*

Para el funcionamiento de la aplicación web, deben existir tablas de base de datos que manejen el flujo de información de acuerdo a los requerimientos, las cuales se describen a continuación.

Se crea la **tabla de usuarios** donde se guardará información referente a la sesión, nombre, permisos, foto de perfil, correo electrónico, organización a la que pertenece y cargo.

Para la **gestión de los artículos** a publicar, se crea la tabla *articles*, con el campo de la información del usuario que la crea, la categoría a la que pertenece, sub categoría, título, contenido, imagen asociada y estadísticas. Las categorías y sub categorías son elementos que un usuario con permisos especiales puede crear, leer, editar y eliminar. Por lo que se crean en una tabla separada llamadas *categories* y *subcategories* respectivamente. Cada artículo debe contener datos estadísticos que complementen la información, por ejemplo, la cantidad de centros médicos en los cuales atender la enfermedad de la que habla el artículo. Para listar estos datos, se crea la tabla *article_figures* donde se guardará una cifra, un contexto explicativo y un icono. Esto se grafica en la base conceptual de presentación de artículos vista en la imagen 2.10.

Para el **muro de desafíos** se crea la tabla *challenges* donde estos se registren, guardando el usuario que lo crea, el último editor del mismo, el artículo asociado, el contenido y el enlace de algún documento que se pueda adjuntar. Estos desafíos abren una discusión pública **que puede ser comentada** por cualquiera de los usuarios registrados en la aplicación. Para manejar esta información se crea la tabla *comments* encargada de guardar el comentario realizado, el usuario que comenta, el último editor del mismo, el desafío asociado, su contenido y alguna URL opcional que tiene relación con un documento que el usuario puede cargar al sistema. Estos documentos son cargados al sistema de almacenamiento de la aplicación dejando su meta-información en una tabla en la base de datos llamada *docs*, la cual guarda el usuario, el último editor, el nombre del archivo y su URL.

Cuando un usuario crea un desafío, este puede ser respaldado por cualquier otro usuario utilizando el **sistema de *likes***, por lo tanto se crea la tabla *likes* que guarda la información del usuario que lo realiza, el desafío marcado con el like y el id del elemento mismo.

Finalmente, se crea una **tabla para las organizaciones** que se asocien al portal, registrando el usuario que las crea, el nombre, la descripción, su misión y visión, su logotipo y dirección de enlace a su sitio web.

Todo este esquema de diseño, se puede observar con detalle en la figura 2.5. Donde la relación más importante es la del usuario con cada una de las cosas que éste puede crear y publicar. Cabe destacar que se crea un sistema de permisos según el tipo de usuario descrito. Los niveles son: "super_admin", .ªdmin", "minsal", "general".

Si bien en la imagen 2.5 aparecen las tablas *publications* y *projects*, son elementos que están fuera del alcance de esta memoria, pero que en resumen pretenden satisfacer la necesidad de tener publicaciones realizadas por un profesional de la salud y la presentación de proyectos en desarrollo de las organizaciones de salud asociadas a la aplicación.

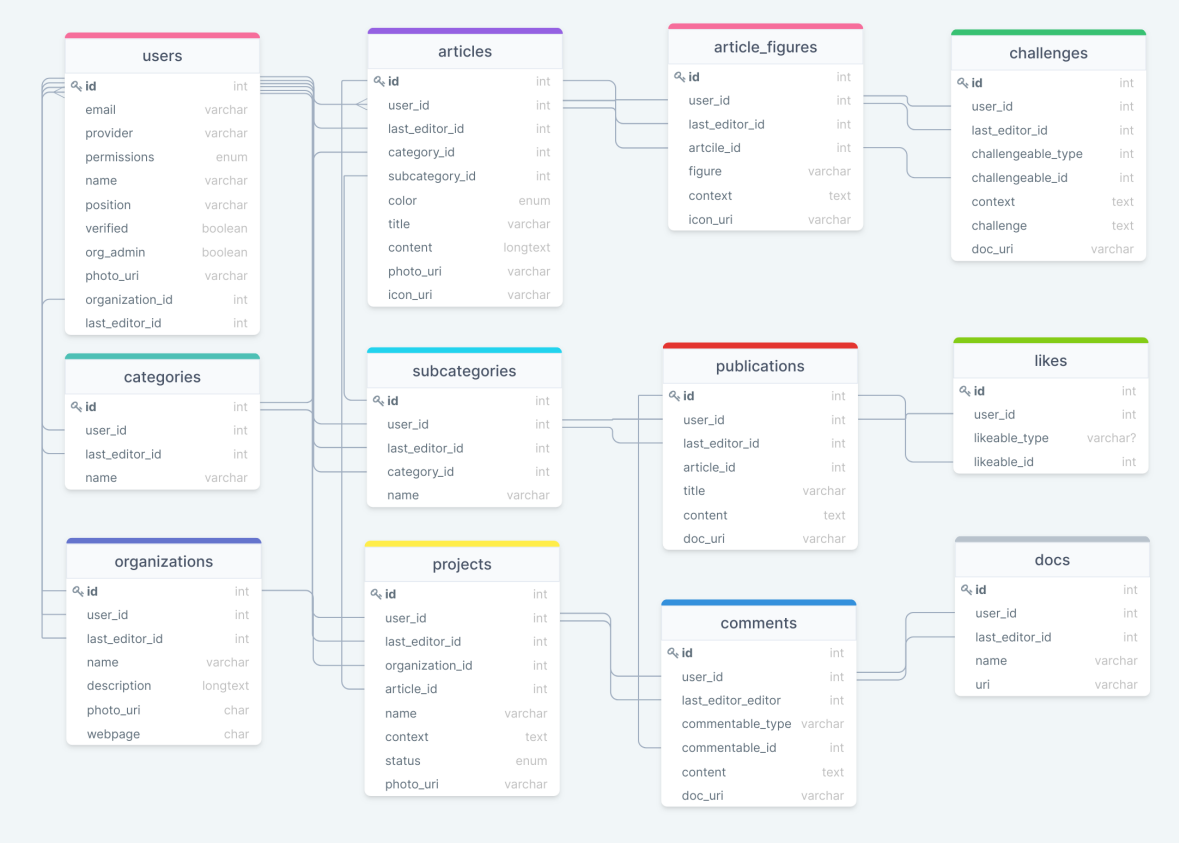

Figura 2.5: Diagrama del diseño de base de datos

2.5. Diseño de maqueta para la interfaz de usuario *Front - End*

En diciembre del 2018, el MINSAL junto con su comité editorial definen a grandes rasgos qué es *Salud Mejor* y como se enmarca en el contexto de *Hospital Digital*. En esta definición se señala que *"es un espacio para proponer y discutir desafíos que generen impacto, abierto a la ciudadanía; que contenga un repositorio para una base de conocimientos e información sobre los proyectos implementados en salud."*[2]. Además se deciden los lineamientos generales sobre el diseño de la plataforma, entregándose el logotipo oficial como muestra la figura 2.6.

Figura 2.6: Logotipo oficial del sitio Salud Mejor

A su vez, se determina la paleta de colores que el sitio debe utilizar para su diseño. Estos se pueden observar en la figura 2.7.

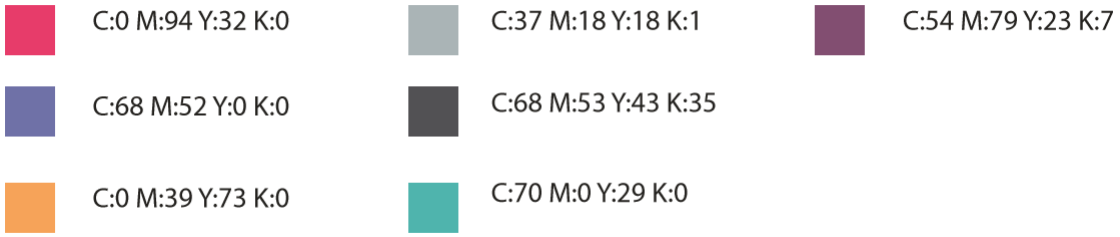

Figura 2.7: Paleta de colores utilizada en el sitio

[2]Comité editorial SM v1.0 06.12.18 - MINSAL

2.5.1. Estructura de la página principal

El concepto estructural de la página principal contiene:

Barra de navegación superior	
Parte izquierda	**Parte derecha**
Logo del sitio	Administrador de artículos
Muro de desafíos	Administrador de proyectos
Publicación de conocimientos	Administrador de usuario
Muro de proyectos	Cerrar sesión
Panel deslizante de imágenes informativas	
Álbum de acceso a artículos publicados	
Accesos directos con explicación de secciones del sitio	
Vitrina de organizaciones participantes	
Pie de página	

Tabla 2.1: Estructura de la página principal

Según la propuesta emanada desde el equipo de diseño del MINSAL podemos observar algunas de las partes que conforman la página principal. En la figura 2.8 se observa la barra de navegación superior y el control deslizante. En la figura 2.9 se observa la propuesta para el menú de botones de selección de artículos junto con un par de mensajes que entregan contexto al usuario.

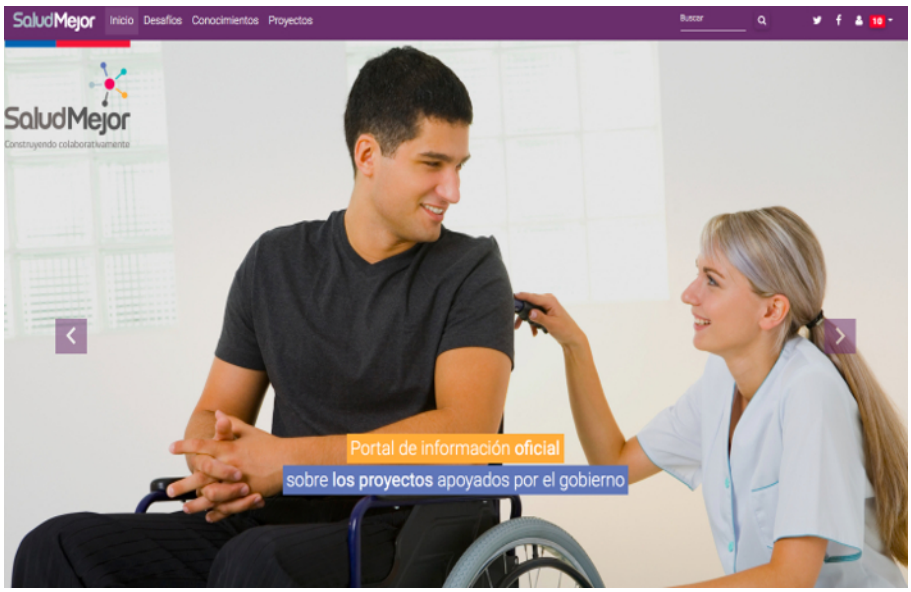

Figura 2.8: Propuesta barra de navegación superior y control deslizante

Figura 2.9: Propuesta panel de selección de artículos y presentación

El menú de botones de selección de artículos debe adecuar el espacio automáticamente dependiendo de la cantidad de artículos creados.

El resto de las secciones se dejan a criterio de diseño posterior. Sin embargo, con estas maquetas ya se tiene una idea de la forma en que el contenido debe ordenarse y diseñarse.

2.5.2. Estructura de visualización de artículos

Para la visualización de artículos se establecen las siguientes secciones:

- Imagen de encabezado que tenga relación con el artículo, que ayude a insertarse con el contenido.
- Logotipo de presentación con el nombre del artículo para contextualizar el contenido.
- Texto del artículo.
- Cifras significativas en relación al artículo que dimensionen estadísticas importantes en relación a la enfermedad o condición.

En la figura 2.10 podemos observar la base conceptual del diseño que el MINSAL desea inicialmente para la sección de artículos.

Figura 2.10: Base conceptual del diseño de la presentación de artículos

2.5.3. Estructura del muro de desafíos

Para aclarar la estructura de esta sección se utilizan ejemplos de otras redes sociales. Se pide básicamente una dinámica como la de Linkedin [3]. Con un muro de publicaciones y un botón para publicar al inicio, sin embargo, el diseño elimina el formato de tres columnas a solo dos. Es decir, el muro a la izquierda y a la derecha filtros de búsqueda y otros botones de interés que a esa fecha aún no están bien definidos.

En la figura 2.11 podemos observar la estructura conceptual que se rescata de las conversaciones con el MINSAL. La parte izquierda está bien definida, pero la derecha se deja abierta a cambios en esta instancia. En el transcurso del proyecto se agrega la idea de filtrar desafíos por temática, exigiendo una relación entre ellos; también se habla de mostrar una lista de los proyectos más recientes y de los desafíos más comentados o con más "likes".

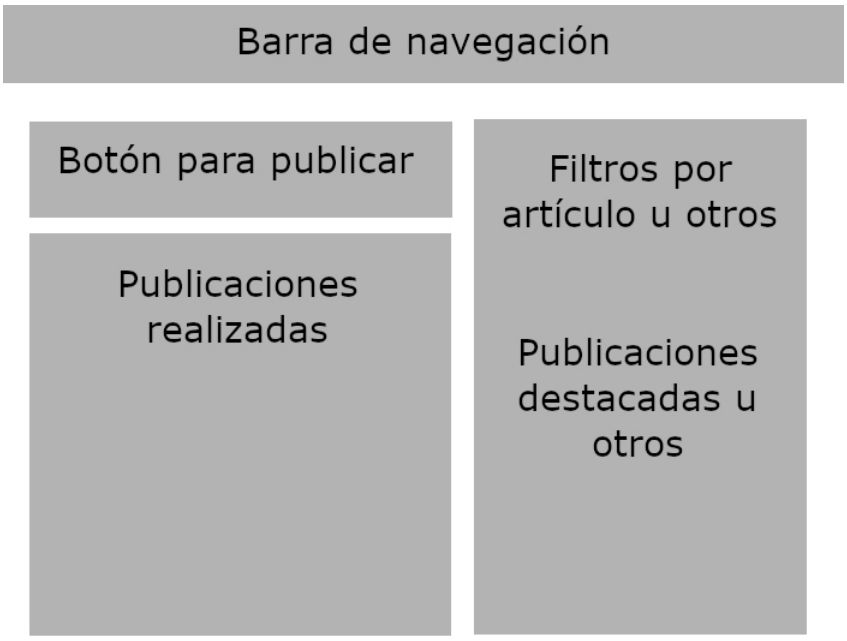

Figura 2.11: Maqueta conceptual del muro de desafíos

Con lo anterior, se procede entonces al diseño de la base de datos que está detrás del funcionamiento de la plataforma y de la construcción del sitio en sí mismo utilizando las herramientas mencionadas en la sub-sección 1.4.

[3] www.linkedin.com

Capítulo 3

Implementación del Back-End

Para la implementación de la base de datos, se utiliza el marco de desarrollo para PHP llamado *Laravel* [2]. Su instalación queda explicada en el apéndice **??**.

Laravel utiliza el patrón de arquitectura MVC (Modelo - Vista - Controlador) [3] para gestionar el proceso entre el servidor y el cliente:

- **Modelo**: Representa lo conocido por la base de datos y el servidor de backend, puede ser un objeto o un conjunto de objetos. Éstos contienen los campos o datos que maneja el sistema, su lógica de negocio y el mecanismo de persistencia. Son, habitualmente, las clases en PHP que están asociadas con tada tabla de la base de datos.

- **Vista**: Información que se envía al cliente, la interfaz de usuario y los mecanismos de interacción dentro de esa interfaz.

- **Controlador**: Intermediario entre el modelo y la vista. Gestiona los flujos de información y transformaciones de los datos.

Además de esto, Laravel ofrece un sistema de clases PHP para facilitar la creación del esquema de una base de datos. Permite definir las tablas y sus columnas fácilmente a través de migraciones compatibles con MySQL 5.6+, PostgresSQL 9.4+,SQLite 3.8+ y SQLServer2017+. La configuración de la base de datos se localiza en */config/database.php* donde se especifican los tipos de conexiones y sus credenciales.

3.1. Estructura de archivos de un proyecto en Laravel

En la tabla 3.1, se describe cada uno de los directorios que contiene un proyecto en Laravel y en la tabla 3.2 los directorios de la carpeta **app** donde se trabajan la mayoría de los archivos.

Directorio	**Descripción**
app	Contiene el código base de toda la aplicación, principalmente contiene los modelos de Eloquent que serán útiles al programar la API.
bootstrap	Contiene los códigos de programas que permiten el inicio rápido de la aplicación y el caché
config	Contiene los archivos de configuración del proyecto
database	Contiene los archivos de la base de datos: Migraciones y semilleros
public	Directorio donde inicia la aplicación desde un cliente web y contiene los scripts en JavaScript minificados, CSS, imágenes y otros archivos públicos
resources	Contiene el código en JavaScript que usa en el frontend con librerías como React.js, AngularJs o VueJs.
routes	Contiene los archivos que definen todas las rutas existentes, tanto las que se acceden desde el cliente web y tienen asociada una vista, como las que acceden a la API.
storage	Contiene los archivos de sesión, caché, planillas compiladas y otros.
test	Contiene todos los casos de testeo
vendor	Contiene todas las dependencias PHP que se instalan a través de Composer.

Tabla 3.1: Descripción del directorio raíz de un proyecto Laravel

Directorio	**Descripción**
Console	Contiene los comandos de la interfaz de línea de comandos artisan
Exceptions	Contiene todos los archivos que se hacen cargo de las excepciones.
Http	Contiene los filtros, las consultas y los controladores.
Providers	Contiene los archivos de los proveedores de servicios.

Tabla 3.2: Descripción del directorio app de un proyecto Laravel

3.2. Creación de migraciones

Una migración se refiere al código de programación con el cual se establecerán las características específicas de una tabla de la base de datos. En esta se definen la cantidad de campos (columnas) que llevará cada fila, el tipo de dato que este campo debe contener y las posibles relaciones que este campo tiene con otros campos de otras tablas de la base de datos.

En Laravel, se deben crear en el directorio **/database/migrations** cada una de las clases que se usarán en la base de datos como representación de las tablas de datos. Para esto, Laravel utiliza un kit de herramientas llamado *Illuminate* [1] [2], este contiene las clases en PHP con las que se gestionan las bases de datos. Específicamente las clases pertenecientes al espacio de nombres *Schema* y *Blueprint* que entregan métodos usuales que ayudan a la gestión de la base de datos. Laravel también entrega una interfaz de línea de comandos llamado *artisan* para facilitar el uso de las clases del kit de herramientas. Para saber los comandos posibles se debe escribir en un terminal de comandos, en la ruta del proyecto:

```
1 php artisan list
```

Finalmente, Lavarel incluye un REPL llamado *Tinker* con el cual se pueden ejecutar comandos que ayuden a probar si nuestra estructura de base de datos está bien definida. Para ejecutar esta interfaz de debe escribir el comando:

```
1 php artisan tinker
```

A continuación, en el script 3.1, se describen las migraciones relacionadas con la tabla de artículos y sus relaciones. La cual se crea con el comando:

```
1 php artisan make:migration CreateArticlesTable
```

[1] https://github.com/illuminate/database
[2] https://laravel.com/docs/5.0/schema

```php
<?php
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateArticlesTable extends Migration{

    public function up(){
        Schema::create('articles', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')->references('id')->on('users');
            $table->integer('last_editor_id')->unsigned()->nullable();
            $table->foreign('last_editor_id')->references('id')->on('users');
            $table->integer('category_id')->unsigned();
            $table->foreign('category_id')->references('id')->on('categories');
            $table->integer('subcategory_id')->unsigned();
            $table->foreign('subcategory_id')->references('id')->on('subcategories');
            $table->enum('color',['bg00','bg01','bg02','bg03','bg04','bg05','bg06','bg07','bg08','bg09'])->default('bg00');
            $table->string('title');
            $table->text('content');
            $table->string('photo_uri')->nullable();
            $table->string('icon_uri')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
    }
    public function down(){
        Schema::dropIfExists('articles');
    }
}
```

Script 3.1: Creación de la migración **articles** para la base de datos

Por definición e instrucción de los creadores de *Laravel* se deben crear las migraciones con el prefijo **Create** y el sufijo **Table**, de este modo se creará automáticamente la tabla llamada *articles* en el sistema de gestión de base de datos compatible cuando se ejecute la migración.

Esta tabla define, en primer lugar, el campo *id* que servirá de identificador único (clave primaria) para cada artículo creado. El segundo campo, llamado *user_id*, se refiere al identificador único del usuario que ha creado el artículo, por lo que este campo se relaciona con una clave foránea con la tabla *users*. El siguiente campo es también una clave foránea para la tabla *users*, pero tiene relación con el último usuario que editó el contenido del artículo. El cuarto campo es una clave foránea que relaciona el artículo creado con la categoría a la que éste pertenece. Así mismo, el quinto campo lo hace con la sub-categoría a la que el artículo pertenece. El siguiente campo es usado para darle un identificador de color al artículo, seleccionando uno de los strings de la lista. Se definen también los campos *title* como un string que contiene el título del artículo, *content* como el contenido de texto del artículo, *photo_uri* como la dirección en la cual encontrar una imágen asociada al artículo, *icon_uri* como una dirección para un ícono asociado, los *timestamps* crean los campos *created_at* y *edited_at* que guardan las fechas de creación y última edición, respectivamente; y por último se establece la regla *softDelete* para que, al eliminar el artículo, sea solo de manera simbólica y no se vaya de la base de datos.

De la misma manera se definen el resto de las migraciones, relacionando cada campo según la figura 2.5.

3.3. Creación de modelos

Laravel cuenta con un ORM llamado *Eloquent* que permite que cada tabla de la base de datos tenga su respectivo *modelo*, este modelo es una clase (dentro de la estructura de programación orientada a objetos) con la cual se pueden realizar una serie de métodos que permiten hacer consultas para crear, leer, actualizar o eliminar información en la base de datos, establecer la ejecución de las relaciones u otras funciones necesarias relacionadas con el modelo, por ejemplo, podríamos definir un método que entregue el total de usuarios en la base de datos como un único número entero.

En particular, se muestra en el scripit 3.2 la creación del modelo para los artículos, el cual relaciona una clase en PHP con la estructura de la migración en una table de la base de datos. Para su creación se debe considerar la convención de usar **snake case** en la definición, de esta manera el modelo **Article** supone la preexistencia de la tabla **articles**. De la misma manera, Eloquent supone que cada tabla tiene su clave primaria llamada **id**, en caso contrario debe ser definida *$primaryKey* como propiedad protegida. Las demás convenciones para Eloquent quedan definidas en la documentación oficial de Laravel[3].

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6  use Illuminate\Database\Eloquent\SoftDeletes;
7  use \Askedio\SoftCascade\Traits\SoftCascadeTrait;
8
9  class Article extends Model{
10     use SoftCascadeTrait, SoftDeletes;
11
12     protected $fillable = [
13         'user_id',
14         'last_editor_id',
15         'category_id',
16         'subcategory_id',
17         'title',
18         'content',
```

[3]https://laravel.com/docs/5.8/eloquent#eloquent-model-conventions

```
19          'color',
20          'photo_uri',
21          'icon_uri',
22      ];
23      protected $appends = [
24          'figures_count',
25          'publications_count',
26          'challenges_count',
27          'projects_count',
28          'followers_count'
29      ];
30      protected $dates = ['deleted_at'];
31      protected $softCascade = [
32          'figures',
33          'publications',
34          'challenges'
35      ];
36
37      public function category(){
38          return $this->belongsTo('App\Category');
39      }
40
41      public function subcategory(){
42          return $this->belongsTo('App\Subcategory');
43      }
44
45      public function user(){
46          return $this->belongsTo('App\User');
47      }
48
49      public function last_editor(){
50          return $this->belongsTo('App\User');
51      }
52
53      public function figures(){
54          return $this->hasMany('App\Article_Figure');
55      }
56
57      public function getFiguresCountAttribute(){
58          return $this->figures()->count();
59      }
60
61      public function publications(){
```

```
62          return $this->hasMany('App\Publication');
63      }
64
65      public function getPublicationsCountAttribute(){
66          return $this->publications()->count();
67      }
68
69      public function challenges(){
70          return $this->morphMany('App\Challenge', '
                challengeable');
71      }
72
73      public function getChallengesCountAttribute(){
74          return $this->challenges()->count();
75      }
76  }
```

Script 3.2: Creación del modelo **article** de Eloquent

Se define en primer lugar, que el modelo tiene activado *softDeteles*, con esto se añade un campo *deleted_at* a la tabla para determinar la fecha de borrado simbólico, de esta manera la eliminación del registro es virtual pero físicamente seguirá en la base de datos. Luego se especifican los campos que podrán ser asignados masivamente con la variable *$fillable*. Por otra parte se determinan los campos apéndice que se podrán obtener por medio de los métodos públicos listados. Por último, se definen los modelos que se relacionan con el softDelete. Para esto, se utiliza una librería de terceros llamada *Askedio softDalete*[4].

Las relaciones entre modelos se deben definir con métodos públicos de cada clase. Existen distintos tipos de relación, por ejemplo, uno es a uno como muestra el método *public function user()* donde se define que cada artículo pertenece a un usuario creador. El resto de los tipos de relaciones son del tipo uno es a muchos, muchos es a muchos y muchos es a uno las cuales pueden ser vistas en detalle en la documentación oficial de Laravel en la sección *Eloquent - Relationships* [5]. Luego de las relaciones se definen métodos para obtener los campos apéndices de la clase, por ejemplo, *getFiguresCountAttribute* la cual utiliza la función de PHP *count()* para retornar el total de figures (línea 57).

[4]https://github.com/Askedio/laravel-soft-cascade
[5]https://laravel.com/docs/5.8/eloquent-relationships

3.4. GraphQL

GraphQL [6] es un lenguaje de consultas y manipulación de datos para APIs el cual, en palabras simples, gestiona todo tipo de obtención y modificación de datos por medio de solo unas pocas rutas, las cuales por lo general tienen relación con los permisos de acceso. Esta tecnología reemplaza en cierta medida el uso de API REST ya que no es necesario tener una cantidad de rutas de consultas HTTP por cada una de las funciones que queremos que el servidor realice (POST ó GET) para interactuar con la base de datos o entregar algún tipo de resultado. Al contrario, GraphQL gestiona todo en una sola ruta usando las consultas con la información exacta y necesaria con una nomenclatura fácil de entender, muy similar al formato JSON [7].

Para levantar el servidor de GraphQL en Laravel se utilizará una librería de terceros llamada *rebing/graphql-laravel*[8]. La cual se instala con el siguiente comando:

```
1 composer require rebing/graphql-laravel
```

Una vez instalada esta librería, se trabaja en base a cuatro elementos importantes: el ***esquema***, los ***tipos***, las ***queries*** y las ***mutaciones***.

3.4.1. Esquema de GraphQL

El esquema define las consultas que se podrán realizar con el servidor de graphql, separa tanto queries y mutaciones según el tipo de usuario que está realizando la consulta, asociando a cada nombre de consulta la clase PHP correspondiente. También se especifican los *tipos* existentes y la clase asociada a ellos.

En este caso existirán dos tipos de usuario, el invitado (por defecto) y el usuario autenticado con el servidor. Esta configuración debe especificarse en el archivo */config/graphql.php*, como se muestra en el ejemplo del script 3.3. El modelo completo lo podemos observar en el gráfico de la imagen 3.1.

[6] https://graphql.org/learn/
[7] https://www.json.org/json-es.html
[8] https://github.com/rebing/graphql-laravel

```php
<?php
return [
    'prefix' => 'graphql',
    'controllers' => \Rebing\GraphQL\GraphQLController::class
        . '@query',
    'default_schema' => 'default',
    'schemas' => [
        'default' => [
            'query' => [
                'users' => App\GraphQL\Query\UserQuery::class,
                'articles' => App\GraphQL\Query\ArticleQuery::
                    class,
            ],
            'mutation' => [],
            'middleware' => [],
            'method' => ['post'],
        ],
        'authenticated' => [
            'query' => [
                'users' => App\GraphQL\Query\UserQuery::class,
                'organizations' => App\GraphQL\Query\
                    OrganizationQuery::class,
            ],
            'mutation' => [
                'newArticle' => App\GraphQL\Mutation\
                    NewArticleMutation::class,
            ],
            'middleware' => ['auth:api'],
            'method' => ['post'],
        ],
    ],
    'types' => [
        'user' => App\GraphQL\Type\UserType::class,
        'organization' => App\GraphQL\Type\OrganizationType::
            class,
    ],
];
```

Script 3.3: Ejemplo de configuración de esquema graphql

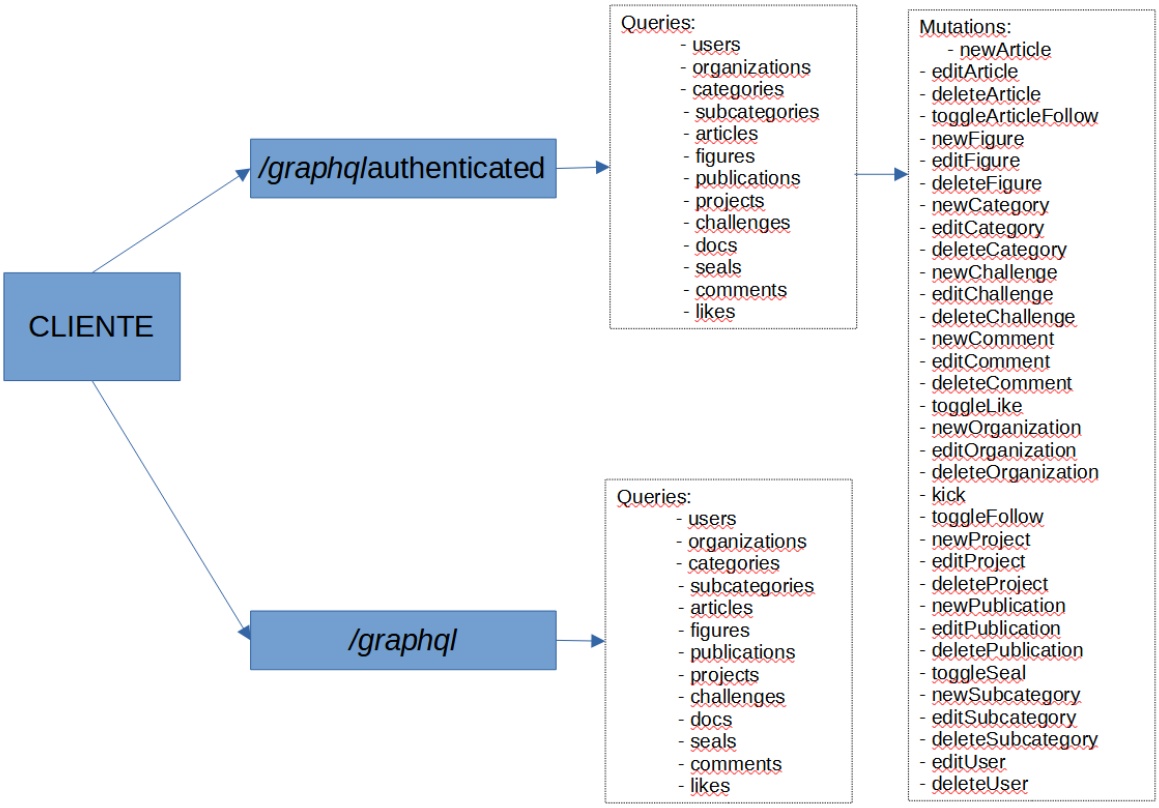

Figura 3.1: Modelo de esquema graphql para quieries y mutaciones

3.4.2. Tipos de GraphQL

Los *tipos* en graphql se definen como un objeto que puede ser buscado en el servidor y que contiene los campos de datos que se pueden pedir, siendo estos el elemento básico de un *esquema*. En el caso de PHP se definen los campos según la regla establecida por la librería *Rebing/graphql-laravel*. A continuación podemos observar un ejemplo de como se define el tipo *ArticleType* de GraphQL con algunos de sus campos en el script 3.4. Este archivo se encuentra en la nueva ruta */app/GrahQL/Type/ArticleType.php* creada una vez que se instala el paquete de *Rebing/graphql-laravel*.

```
1
2 namespace App\GraphQL\Type;
3
4 use Rebing\GraphQL\Support\Type as GraphQLType;
5 use GraphQL\Type\Definition\Type;
6 use App\Article;
```

```
7  use GraphQL;
8
9  class ArticleType extends GraphQLType
10 {
11     protected $attributes = [
12         'name' => 'ArticleType',
13         'description' => 'Art culos',
14         'model' => Article::class,
15     ];
16     public function fields() {
17         return [
18             'id' => [
19                 'type' => Type::int(),
20                 'description' => 'ID del art culo',
21             ],
22             'user' => [
23                 'type' => GraphQL::type('user'),
24                 'description' => 'Usuario creador',
25             ],
26             'last_editor' => [
27                 'type' => GraphQL::type('user'),
28                 'description' => 'Ultimo usuario editor',
29             ],
30             'category' => [
31                 'type' => GraphQL::type('category'),
32                 'description' => 'Categor a',
33             ],
34             'subcategory' => [
35                 'type' => GraphQL::type('subcategory'),
36                 'description' => 'Subcategor a',
37             ],
38             'title' => [
39                 'type' => Type::string(),
40                 'description' => 'T tulo del art culo',
41             ],
42             'content' => [
43                 'type' => Type::string(),
44                 'description' => 'Contenido del art culo',
45             ],
46             'figures' => [
47                 'type' => Type::listOf(GraphQL::type('
                       article_figure')),
48                 'description' => 'Cifras del articulo',
```

```
49                ],
50                'figures_count' => [
51                    'type' => Type::int(),
52                ],
53            ];
54    }
55 }
```

Script 3.4: Creación del tipo **article** de GraphQL

En primer lugar se instancian las clases necesarias para definir un *tipo* de GraphQL. Luego se crea la clase *ArticleType* que extiende de la clase *GraphQLType* todos sus métodos y atributos. Dentro de esta definición se asignan los atributos protegidos locales *name*, *description* y *model*, donde el tercero tiene relación con el modelo de *Eloquent* definido previamente. A continuación se define la función pública *fields()* que retorna todos los campos que será posible pedir al servidor. Éstos tienen relación con los campos de la base de datos de la tabla *articles*, las relaciones predefinidas en las migraciones (sección 3.2) y los métodos públicos definidos en el modelo *Articles*(sección 3.3).

De la misma manera que en el script 3.4, se definen los siguientes tipos de GraphQL:

ArticleFigureType.php	Cifras de los artículos que contienen una descripción y una cifra importante.
CategoryType.php	Categorías para clasificar los artículos
SubCategoryType.php	Sub categorías para clasificar los artículos.
ChallengeType.php	Desafíos que se harán por los usuarios respecto a la temática tratada en los artículos.
CommentType.php	Comentarios realizados en un desafío.
DocType.php	Documento asociado a alguna publicación, ya sea un desafío o un comentario.
LikeType.php	Reacción de los usuarios cuando les gusta un comentario.
OrganizationType.php	Organizaciones que están registradas.
UserType.php	Usuarios registrados.

Tabla 3.3: Lista de tipos de GraphQL desarrollados

3.4.3. Consultas en GraphQL

Las consultas en GraphQL se hacen a través de las dos rutas creadas en el archivo de configuración al definir el esquema: */graphql* para las consultas sin autenticar y

graphql/authenticated para las consultas de usuarios autenticados con el servidor a través de sus cuentas de Facebook o Google como muestra la imagen 3.1.

Para definir una consulta se utiliza la librería *rebing/graphql-laravel*, que nos entrega comandos automáticos para ser usados con artisan. Para crear una nueva query llamada *ArticleQuery* debemos escribir el siguiente comando[9]:

```
1 php artisan make:graphql:query ArticleQuery
```

Con esto, se crea el archivo */app/GraphQL/Query/ArticleQuery.php* el cual se muestra en el siguiente script 3.5.

```
1  <?php
2
3  namespace App\GraphQL\Query;
4
5  use GraphQL\Type\Definition\Type;
6  use GraphQL\Type\Definition\ResolveInfo;
7  use Rebing\GraphQL\Support\SelectFields;
8  use Rebing\GraphQL\Support\Query;
9  use Rebing\GraphQL\Support\Facades\GraphQL;
10 use App\Article;
11 use Carbon\Carbon;
12
13 class ArticleQuery extends Query
14 {
15     protected $attributes = [
16         'name' => 'ArticleQuery',
17         'description' => 'Retorna informacion de articulos'
18     ];
19
20     public function type()
21     {
22         return GraphQL::paginate('article');
23     }
24
```

[9]https://github.com/rebing/graphql-laravelcreating-a-query

```
25      public function args()
26      {
27          return [
28              'limit' => [
29                  'type' => Type::int(),
30                  'description' => 'Limite de articulos por
                        pagina',
31              ],
32              'page' => [
33                  'type' => Type::int(),
34                  'description' => 'Numero de pagina que se
                        revisara',
35              ],
36              'sortByAsc' => [
37                  'type' => Type::string(),
38                  'description' => 'Columna por la que se
                        ordenaran los datos, en orden ascendente',
39              ],
40              'sortByDesc' => [
41                  'type' => Type::string(),
42                  'description' => 'Columna por la que se
                        ordenaran los datos, en orden descendente',
43              ],
44              'month' => [
45                  'type' => Type::int(),
46                  'description' => 'Mes por el que se filtrara',
47              ],
48              'week' => [
49                  'type' => Type::int(),
50                  'description' => 'Semana por la que se
                        filtrara',
51              ],
52              'id' => [
53                  'type' => Type::int(),
54                  'description' => 'ID del articulo',
55              ],
56              'user_id' => [
57                  'type' => Type::int(),
58                  'description' => 'Usuario creador',
59              ],
60              'last_editor_id' => [
61                  'type' => Type::int(),
62                  'description' => 'ID del ultimo usuario editor
```

```
                ',
63              ],
64              'category_id' => [
65                  'type' => Type::int(),
66                  'description' => 'ID de la categoria',
67              ],
68              'subcategory_id' => [
69                  'type' => Type::int(),
70                  'description' => 'ID de la  subcategoria',
71              ],
72              'title' => [
73                  'type' => Type::string(),
74                  'description' => 'Titulo del articulo',
75              ],
76              'content' => [
77                  'type' => Type::string(),
78                  'description' => 'Contenido del articulo',
79              ],
80          ];
81      }
82
83      public function resolve($root, $args, SelectFields $fields
            , ResolveInfo $info)
84      {
85          $where = function ($query) use ($args) {
86              if (isset($args['id']))
87                  $query->where('id', $args['id']);
88
89              if (isset($args['user_id']))
90                  $query->where('user_id', $args['user_id']);
91
92              if (isset($args['last_editor_id']))
93                  $query->where('last_editor_id', $args['
                        last_editor_id']);
94
95              if (isset($args['category_id']))
96                  $query->where('category_id', $args['
                        category_id']);
97
98              if (isset($args['subcategory_id']))
99                  $query->where('subcategory_id', $args['
                        subcategory_id']);
100
```

```
101             if (isset($args['title']))
102                 $query->where('title', 'LIKE', '%' . $args['
                        title'] . '%');
103
104             if (isset($args['content']))
105                 $query->where('content', 'LIKE', '%' . $args['
                        content'] . '%');
106
107             if (isset($args['month']))
108                 $query->whereMonth('updated_at', '=', $args['
                        month']);
109
110             if (isset($args['week'])) {
111                 $start = Carbon::now()->setISODate(date("Y"),
                        $args['week']);
112                 $end = Carbon::now()->setISODate(date("Y"),
                        $args['week'] + 1);
113                 $query->whereBetween('updated_at', [$start,
                        $end]);
114             }
115         };
116         $temp = Article::with(array_keys($fields->getRelations
                ()))->where($where)->get();
117         if (isset($args['sortByAsc']) && !isset($args['
                sortByDesc']))
118             $temp = $temp->sortBy($args['sortByAsc']);
119
120         if (!isset($args['sortByAsc']) && isset($args['
                sortByDesc']))
121             $temp = $temp->sortByDesc($args['sortByDesc']);
122
123         if (isset($args['limit']) && isset($args['page']))
124             $temp = $temp->paginate($args['limit'], $args['
                    page']);
125         else
126             $temp = $temp->paginate();
127         return $temp;
128     }
129 }
```

Script 3.5: Creación de la query para el tipo **ArticleType** de GraphQL

Toda query de GraphQL se compone de los mismos elementos:

Definición de espacio de nombres	App\GraphQL\Query
Librerías necesarias	Rebing, modelo asociado a la query, GraphQL Type
Definición de la clase	ModelNameQuery extends Query
Atributos protegidos	nombre y descripción
Declaración de la función type()	Donde se usa el tipo de GraphQL definido en el esquema.
Declaración de la función args()	Define los argumentos que se pueden usar como filtros al momento de realizar la consulta, especificando el tipo de dato y una descripción del mismo.
Declaración de la función \resolve()	Resuelve la query según los argumentos de la consulta y devolver el resultado al cliente.

Tabla 3.4: Lista de consultas de GraphQL desarrolladas

En particular en la definición de la query **ArticleQuery** del script 3.5 se definen elementos que tiene relación con la paginación de la información, de modo que se puedan utilizar los datos ordenadamente en la interfaz gráfica de la aplicación web o en el cliente que las necesite. Entregando la opción de ordenar de manera ascendente o descendente respecto a alguno de cualquiera de sus campos y de mostrar la información definiendo el límite de elementos a mostrar en la primera página de la paginación.

3.5. Sistema de acceso e inicio de sesión

Para gestionar el sistema de acceso a usuarios se utilizan dos librerías oficiales de Laravel, **Passport** [10] y **Socialite**[11]. El primero se encarga de gestionar el token de acceso entre el cliente y nuestra API y el segundo se encarga de pedir el token de acceso a la API de terceros. Para entender mejor el flujo de la autenticación del sistema, se gráfica la figura 3.2.

1. El cliente a través de la ruta */auth/facenook* le pide a socialite que se comunique con facebook para que este ejecute la petición de autorización de la cuenta asociada.

[10]https://laravel.com/docs/5.8/passport
[11]https://laravel.com/docs/5.8/socialite

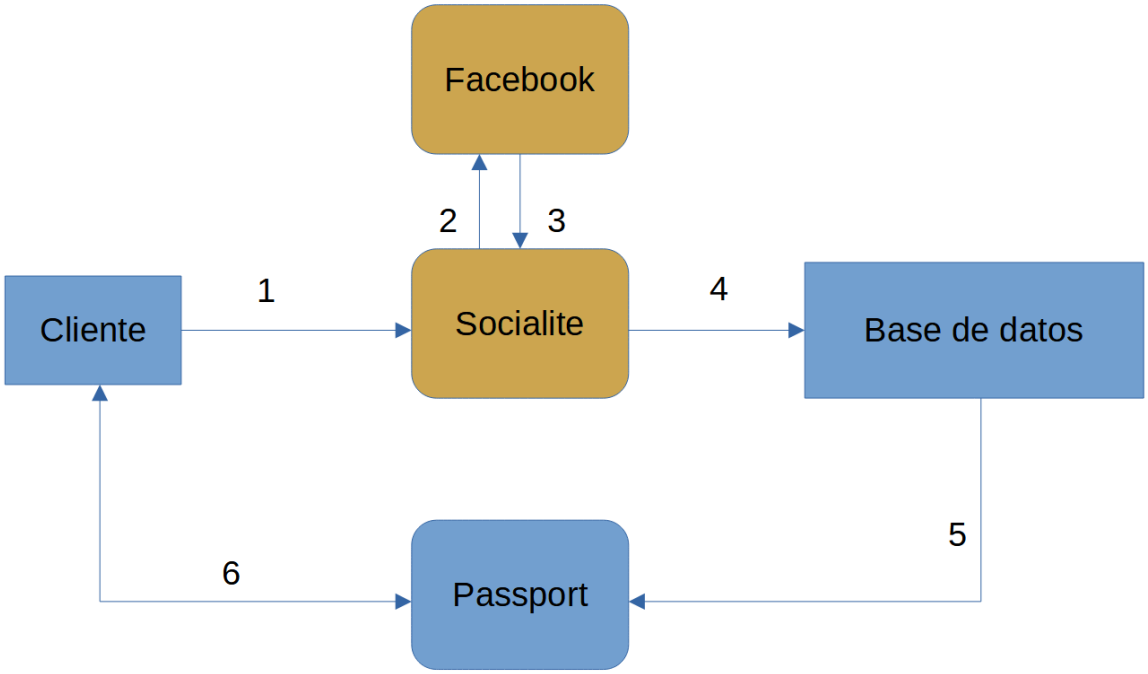

Figura 3.2: Modelo del flujo de la autenticación

2. Socialite le pide a facebook la información del usuario y abre la petición de autorización.

3. Una vez que se autoriza a facebook a entregar la información, este la envía de vuelta a socialite quién desglosa los datos del usuario (Nombre, email y foto de perfil).

4. Luego se verifica que este usuario esté en la base de datos, si no está lo registra en la misma.

5. Una vez que el usuario está en la base de datos, Passport genera un token de acceso con el cual el cliente puede realizar sus consultas.

6. El cliente recibe el token, el cual debe ser enviado en los headers de la consulta HTTP cada vez que esta se realice.

3.6. Rutas

A continuación se listan las rutas generadas para poder comunicarse con el servidor y ejecutar los servicios antes mencionados.

Método	**Ruta**	**Middleware**
GET	auth/{provider}	web,guest
GET	auth/{provider}/callback	web,guest
POST	api/docs/create	api,auth:api
POST	graphql	web,guest
POST	graphql/{authenticated}	api,auth:api

Tabla 3.5: Rutas creadas para la comunicación con el servidor

Se muestra en la tabla 3.5 las rutas de comunicación para la autenticación, la ruta para subir archivos y las rutas de GraphQL. Cabe señalar aquí la importanciá de GraphQL, ya que si no se usara existirían muchas más rutas, una por cada GET y POST que se necesita para cada modelo de Eloquent. En cambio, con GraphQL se realiza todo esto solo con dos rutas.

3.7. Alimentación de la base de datos

Para poder realizar pruebas y verificar el funcionamiento de cada una de las rutas de GraphQL, se alimenta la base de datos con fábrica de datos y sembradores. Para esto se utiliza una librería de PHP llamada *Faker* [12] con la que se pueden construir datos falsos para cada uno de los tipos de campos en la base de datos. En el script 3.7 se detalla un ejemplo de una fábrica de datos para el modelo de artículos. Para crear la fábrica y el semillero se utilizan los siguientes comandos:

```
1 php artisan make:factory ArticleFactory
```

```
1 php artisan make:seeder ArticleSeeder
```

[12] https://github.com/fzaninotto/Faker

```php
<?php

use Faker\Generator as Faker;
$factory->define(App\Article::class, function (Faker $faker) {
    $usr_id = App\User::all()->random()->id;
    $cat_id = App\Category::all()->random()->id;
    $subcat_id = App\Subcategory::all()->random()->id;
    $colors = ['bg00', 'bg01', 'bg02', 'bg03'];
    return [
        'title'          => $faker->sentence(1),
        'content'        => $faker->text(5000),
        'user_id'        => $usr_id,
        'category_id'    => $cat_id,
        'subcategory_id' => $subcat_id,
        'color'      => $faker->randomElement($colors),
        'photo_uri' => 'http://via.placeholder.com/1920x600',
        'icon_uri'  => 'http://via.placeholder.com/80x80',
    ];
});
```

Script 3.6: Creación de una fábrica para los artículos

```php
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class ArticlesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     * @return void
     */
    public function run()
    {
        factory(App\Article::class, 4)->create();
    }
}
```

Script 3.7: Creación de un semillero para los artículos

Para la creación de la fábrica de los artículos es necesario tener datos de usuarios, categorías y sub categorías previamente, ya que estos se relacionan con claves foráneas. Una vez que están todos los semilleros creados, se ejecutan con el comando:

```
1 php artisan db:seed
```

3.8. Pruebas de las rutas

Para realizar pruebas de las rutas, se levanta un servidor local utilizando la herramienta de windows AMPPS, la cual instala el servidor web Apache, la base de datos MySql, PHP, Perl y Python. Además de esto, se instala un software para poder realizar consultas, en este caso utilizamos Insomnia REST [13].

En la imagen 3.3 se observa la respuesta del servidor al pedir la lista de artículos de la base de datos ordenados por el título de forma descendente.

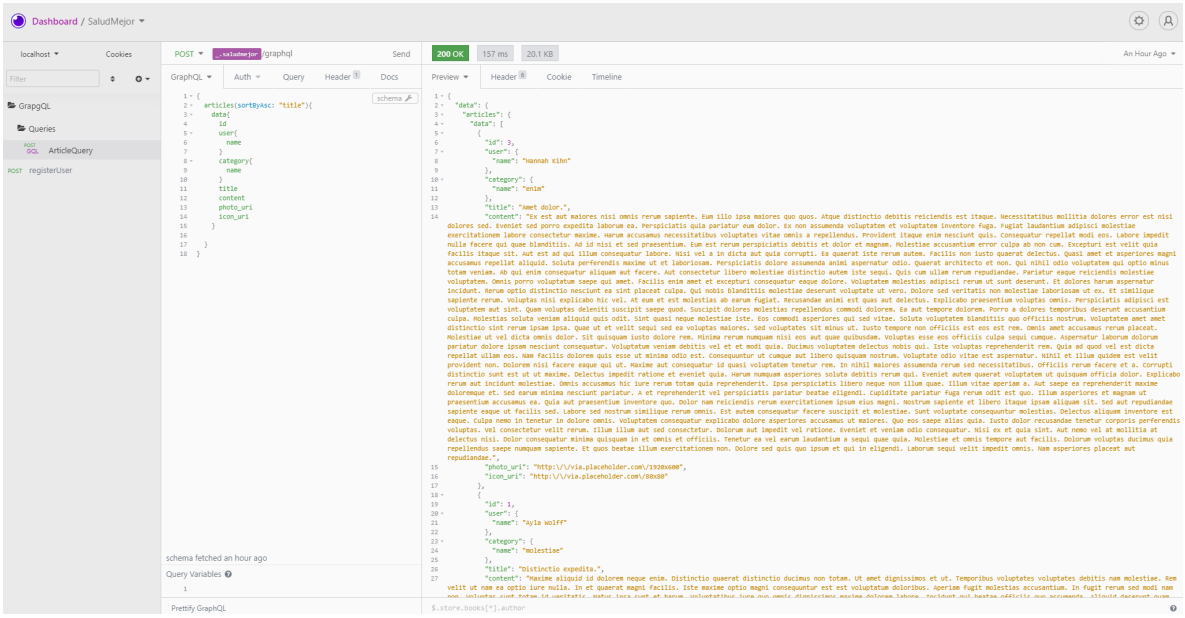

Figura 3.3: Resultado de prueba a consulta GraphQL a los artículos

[13]https://www.insomnia.rest/

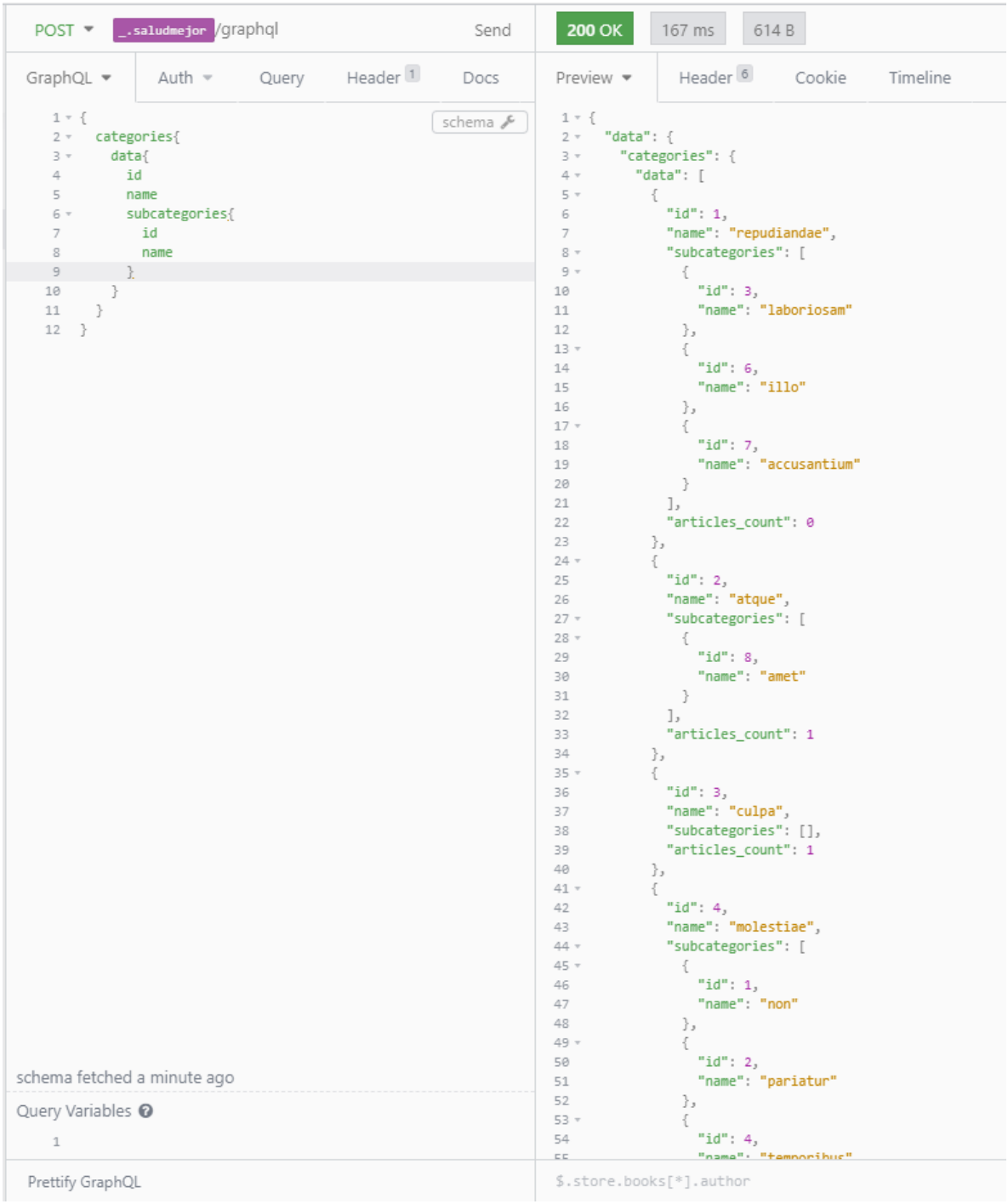

Figura 3.4: Respuesta del servidor a consulta GraphQLQ a las categorías

En la figura 3.4 podemos observar la respuesta del servidor a la petición de las categorías. Al mismo tiempo pedimos la información de las sub categorías asociadas a cada categoría. Esta es otra característica importante de GraphQL, ya que al definir

las relaciones en los tipos, se puede recibir la información en cascada de cada una de estas relaciones.

Capítulo 4

Implementación de la interfaz de usuario

Para establecer la estructura de los componentes que tendrá la interfaz gráfica, se define en primer lugar, el mapa del sitio.

4.1. Mapa del sitio

El diseño estructural del sitio web separa cada vista según muestra la figura 4.1.

4.2. Blade e integración de ReactJS

Para el desarrollo de la interfaz de usuario del sitio web se utiliza la librería de JavaScript RecatJs [1]. Laravel reconoce el uso de esta librería automáticamente cuando ejecutamos el comando:

```
1 php artisan preset react
```

[1] https://es.reactjs.org/docs/hello-world.html

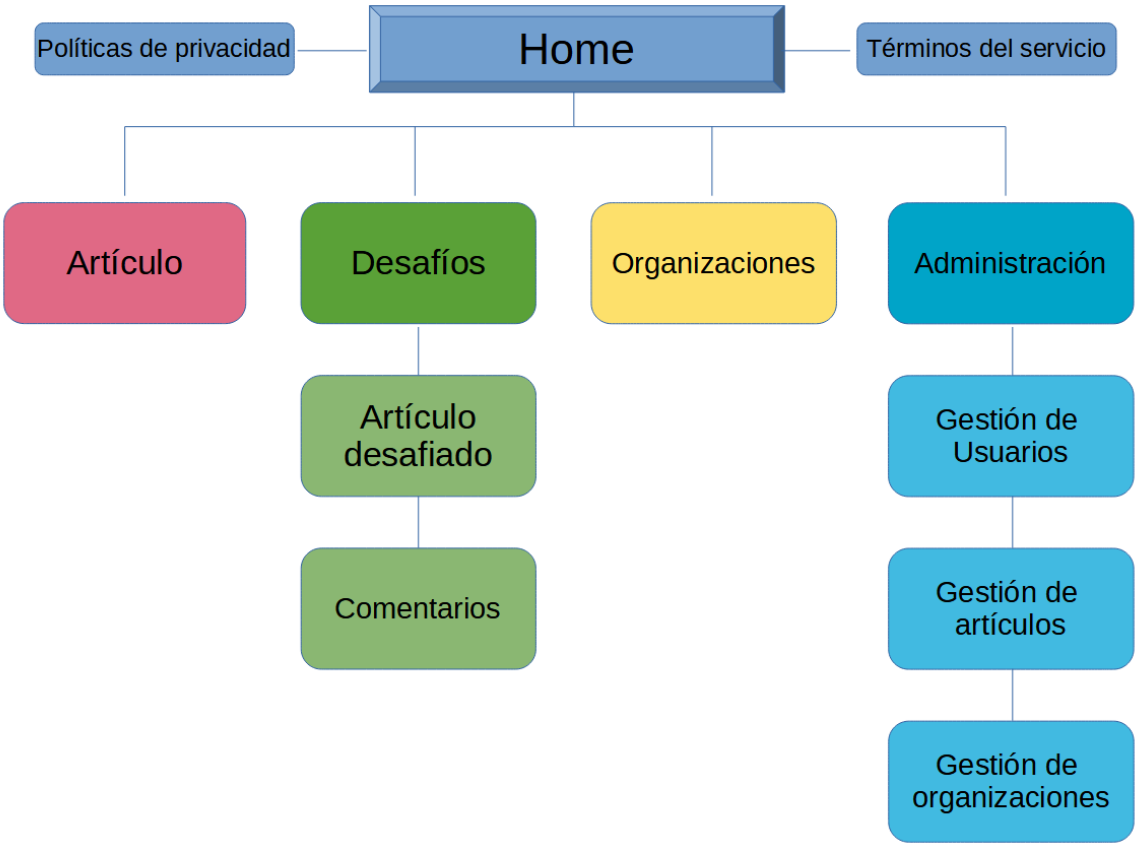

Figura 4.1: Mapa del sitio Salud Mejor

Si bien Laravel presenta la opción de utilizar plantillas para cada vista a través de Blade [2], se concentra la programación de la interfaz utilizando la arquitectura de ReactJS y posicionando el esqueleto principal con Blade.

Con Blade se definen, en un archivo PHP padre */resources/views/app.blade.php*, las etiquetas principales **html**, **head** y **body**. En el head de este archivo se importa el archivo JavaScript que contendrá todo el código generado por ReactJS. Dentro de <body> se construye una etiqueta padre con el atributo ïd=content"para contener todo lo demás. Finalmente se crean cuatro etiquetas div con distintos identificadores, uno para la barra de navegación de invitado, otro para la barra de navegación de usuario logeado, otra para el contenido completo y la última para el pie de página. Un esquema de todo lo anterior se muestra en la figura 4.2 y el script 4.1 muestra la estructura principal y única en los views de blade.

[2]https://Laravel.com/docs/5.8/blade

Figura 4.2: Flujo de información entre el cliente y las respuestas del servidor para los archivos del front-end

```
<body>
    <div id="app">
        @guest
            <div id="navBar"></div>
        @endguest
        @auth
            <div id="navBarAuth"></div>
        @endauth
        <div id="content"></div>
        <div id="footer"></div>
    </div>
</body>
```

Script 4.1: Estructura del HTML padre de Blade

Para que ReactJS pueda renderizar su código se utiliza el DOM [3] [4] que reconoce cada identificador de las etiquetas y ejecuta el código donde corresponda y la extensión del lenguaje JavaScript llamada JSX[5] la cual produce elementos de ReactJs que pueden ser llamados por el DOM[6].

[3]https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction
[4]https://dom.spec.whatwg.org/
[5]https://es.reactjs.org/docs/introducing-jsx.html
[6]https://es.reactjs.org/docs/rendering-elements.html

Por ejemplo, para renderizar el contenido de la aplicación, se reconoce la etiqueta con identificador *'content'* como muestra la línea 49 del script 4.2. De la misma manera, se renderizan las clases de ReactJS correspondientes a las barras de navegación y el pie de página.

```
1  export default class App extends Component {
2      constructor(props) {
3          super(props)
4          this.state = {}
5      }
6      render() {
7          let id = document.getElementById('app').dataset.user
8          let chosenClient
9          if (!id) {
10             id = 0
11             chosenClient = client
12         }
13         else {
14             id = parseInt(id)
15             chosenClient = mutationClient
16         }
17         return (
18         <ApolloProvider client = {chosenClient}>
19             <Query query = {GET_USER} variables = {{id: id}}>
20                 {({loading, error, data: {users}}) => {
21                 if (loading) return <Loading/>
22                 if (error) return <div>Error</div>
23                 let user = users.data[0]
24                 let orgs = "-"
25                 if (typeof user != 'undefined') {
26                     localStorage.setItem("uID", user.id)
27                     if (user.organization != null)
                           localStorage.setItem("oID", user.
                           organization.id)
28                     localStorage.setItem("uTP", user.type)
29                     localStorage.setItem("org", user.org_admin
                           )
30                     localStorage.setItem("vrf", user.verified)
31                     if (user != []) user.
                           organizationsfollowing.map(o => { orgs
                          += o.id+"-" })}
```

```
32                  return(
33                      <Router>
34                      <React.Fragment>
35                          <Route exact path="/" component = {
                                Home} user = {user}/>
36                          <Route path="/home/" component = {Home
                                }/>
37                          <Route path="/explorar/:id" component
                                = {Explore} />
38                          <Route path="/desafios" component ={
                                ChallengesPage}/>
39                          <Route path="/desafios-tematica/:id"
                                component ={ChallengeArticle}/>
40                          <Route path="/perfil" component = {
                                Profile} />
41                          <Route path="/gestionar-usuarios"
                                component={UsersList}/>
42                          <Route path="/gestionar-tematicas"
                                component={ArticleCreate}/>
43                          <Route path="/lista-organizaciones"
                                component={OrganizationsPage}/>
44                          <Route path="/organizaciones/:id"
                                component={OrganizationProfile}/>
45                      </React.Fragment>
46                      </Router>
47                      )}}
48                  </Query>
49              </ApolloProvider>
50          );
51      }
52  }
53
54  if (document.getElementById('content')) ReactDOM.render(<App
        />, document.getElementById('content'))
```

Script 4.2: Clase que contiene toda la aplicación de ReactJS

Toda clase de ReactJs se compone de los mismos tres elementos básicos:

- Constructor: Función pública de la clase correspondiente que inicializa en estado en el que se encuentra el componente.
- **render()**: función que se ejecuta al montar el componente en el DOM, contiene las lógicas previas para gestionar la información dinámica a mostrar.
- **return()**: elemento de la función render() que retorna la información a renderizar. En esta pueden existir elementos de ReactJS heredados de otras clases que respondan con otras funciones render() hijas.

Además de los elementos antes mencionados, también se pueden declarar funciones públicas o privadas dentro de la clase del componente asociado.

Otras librerías importantes en el desarrollo del front-end corresponden a *react-router-dom* [7] y @apollo/client [8]. La primera permite asociar las clases (componentes) de React a una ruta en específico, con esto, podemos renderizar partes del código dependiendo de la ruta en la que estemos. La segunda es la encargada de comunicarse con el servidor de GraphQL, construir las consultas y gestionar las respuestas.

En el script 4.2, podemos observar los 3 elementos básicos de cualquier clase en ReactJS. En primero lugar, el constructor del estado inicial no declara ninguna variable. En la función **render()** la primera operación realizada intenta obtener el id del usuario autenticado guardada por Blade en la etiqueta div padre. Si no existe, se asigna el cliente de invitado de Apollo, si existe, se asigna el cliente de Apollo para las consultas de usuario autenticado.

Luego en la función **return()**, se inicializa el servicio de ApolloClient y se realiza la primera consulta para obtener la información del usuario para ser guardada en el localStorage [9] del navegador web. Finalmente, se inicializa el componente de ReactRouter para establecer qué componentes de React son renderizados según las rutas correspondientes. Un esquema de esta parte puede ser visto en la figura 4.1.

[7] https://reactrouter.com/
[8] https://www.apollographql.com/docs/react/v2
[9] https://developer.mozilla.org/es/docs/Web/API/Window/localStorage

4.3. Componentes de ReactJS

Cada componente de ReactJS es un código de JavaScript que define una función o una clase. Lo importante es que conceptualmente aceptan entradas arbitrarias (llamadas "props") y devuelven a React elementos que describen lo que debe aparecer en la pantalla [10].

Para ayudar al diseño de la interfaz gráfica, se utilizan componentes de React de terceros desde la librearía de MDBReact[11]. La cual integra toda la gama de elementos que entrega el framework para diseño web bootstrap[12].

4.3.1. Barra de navegación

La barra de navegación tiene los enlaces para las secciones principales, entre ellas, la sección de desafíos, conocimientos y proyectos. Además, contiene el botón para ingresar a la plataforma a través de facebook o google. Sin embargo, cuando el usuario está dentro de la plataforma con una cuenta y tiene permisos de administrador, puede gestionar los artículos, las organizaciones, los proyectos y los usuarios. La figura 4.3 muestra la barra de navegación para invitados y la figura 4.4 muestra todas las opciones que tiene un administrador.

Figura 4.3: Barra de navegación para usuarios invitados

Figura 4.4: Barra de navegación para usuarios con cuenta súper administrador

Si el usuario ingresa a la plataforma con alguna autenticación pero no tiene permisos elevados, solo podrá acceder a la sección de edición de su propia información de usuario. Si tiene el acceso MINSAL podrá publicar artículos. Sí el usuario es administrador de una organización, podrá publicar proyectos, editar la información de su organización y

[10] https://es.reactjs.org/docs/components-and-props.html
[11] https://mdbootstrap.com/docs/react/
[12] https://getbootstrap.com/docs/5.0/getting-started/introduction/

editar a los usuarios que son parte de la misma. La creación de organizaciones y gestión de usuario solo la pueden realizar los usuarios que tiene permisos de administrador.

4.3.2. Home

Como se planificó con el MINSAL en la sección 2.5, la primera visualización del sitio contiene:

1. **Panel de imágenes deslizantes**:

   Para la creación de este panel se utiliza el componente de ReactJs llamada *Carousel* entregado por la librería mdbreact[13], como muestra la figura 4.5.

Figura 4.5: Resultado de la renderización del Carousel de MDBReact

   La configuración del componente contiene las siguientes propiedades:

   - activeItem = 1: Señala el índice de la imagen a mostrar por defecto.
   - length=3: Determina que la cantidad de imágenes a mostrar serán 3.
   - showControls=false: No muestra los controles deslizantes laterales.
   - showIndicators=true: Sí muestra los indicadores inferiores.
   - interval=2000: El intervalo de cada imagen es de 2 segundos.

2. **Panel de artículos**:

   Este panel tiene la característica de que cuando se publique un nuevo artículo, el menú se ajusta automáticamente para mantener de manera proporcional la distri-

[13]https://mdbootstrap.com/docs/react/advanced/carousel/#docsTabsAPI

bución de los botones. Para esto se crean dos clases importantes: ArticlesAlbum, que ordena cada botón dinámicamente y ArticlesButton que muestra el ícono, el nombre del artículo y gestiona la redirección al contenido del mismo. Desde Home se realiza una query con Apollo para obtener los artículos existentes en la base de datos, estos se pasan como propiedad a la primera clase ArticleAlbum. Se realiza un mapeo de cada uno de los botones dentro de un componente de MDBReact llamado *MDBBtnGroup* [14] que define un espacio flex (FlexBox)[15] con la clase CSS *flex-wrap*[16] activada. Esto permite que los elementos hijos se agrupen automáticamente en una o varias filas sin sobre pasar el ancho total determinado del espacio. También se define la clase CSS *justify-content: center* para que los botones estén centrados. El resultado de esto lo podemos observar en la figura 4.6.

Figura 4.6: Caja Flex con botones de artículos

También puede observarse en la figura 4.6 una presentación sobre qué es salud mejor. Esto es un simple texto contenido en el componente de MDBReact llamado *MDBContainer*[17], el cual está diseñado para reaccionar responsivamente dependiendo del tamaño de la pantalla y ajustar el texto automáticamente usando las *Media Queries* de CSS [18]. Cada uno de estos botones lleva a la vista del artículo correspondiente en la ruta */explorar/id* con el id del mismo, redirección que es gestionada con un link por el componente ArticleButton.

3. **Accesos directos explicativos**:

En la figura 4.7 podemos observar una sección explicativa de las diferentes cosas que el usuario puede realizar en el sitio. Para la construcción de esta parte se

[14]https://mdbootstrap.com/docs/react/components/button-group/
[15]https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout
[16]https://developer.mozilla.org/es/docs/Web/CSS/flex-wrap
[17]https://mdbootstrap.com/docs/react/layout/overview/
[18]https://developer.mozilla.org/es/docs/Web/CSS/Media_Queries/Using_media_queries

utilizan los componentes de MDBReact *MDBRow* y *MDBCol*, los cuales utilizan FlexBox para ordenar en filas y columnas flexibles (responsivas).

Figura 4.7: Explicación de cada una de las partes medulares del sitio

Cabe señalar que el alcance de esta memoria trabaja hasta la sección de planteamiento de desafíos. La sección para compartir conocimientos y revisar los proyectos de las organizaciones se trabaja posteriormente a la realización de este documento.

4. **Presentación de organizaciones**:

La plataforma debe incluir la participación de organizaciones relacionadas al área de la salud con el objetivo de validar la información contenida, mostrar sus proyectos y participar activamente en las respuestas de los desafíos que los usuarios comunes presenten.

En la figura 4.8 se muestra el resultado de la programación de esta sección. Para su realización se utiliza la misma técnica que en la presentación de los botones de artículos, creando una clase padre llamada *OrganizationsAlbum* y una hija

OrganizationButton. La primera crea el ambiente FlexBox para contener a los hijos una vez que se obtiene la información desde la base de datos con Apollo. Esta consulta se realiza al mismo tiempo que la encargada de pedir la información de los artículos.

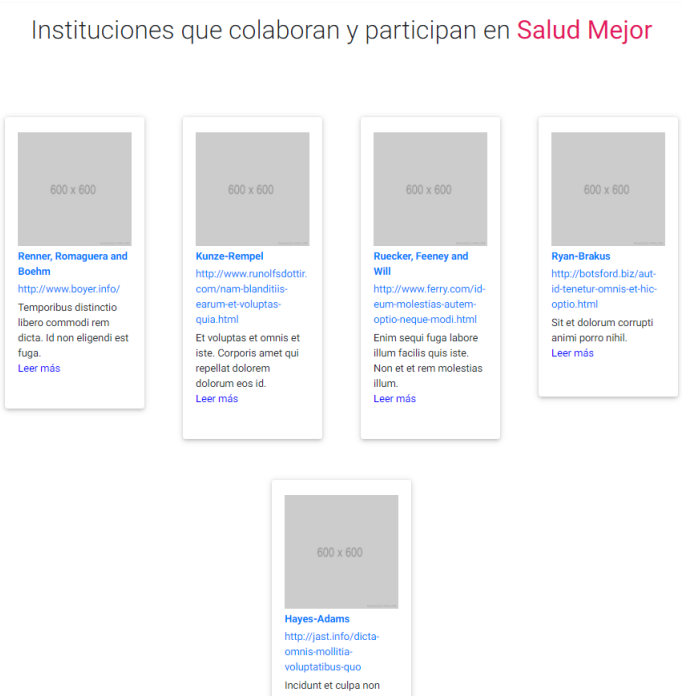

Figura 4.8: Panel de presentación de organizaciones involucradas con la plataforma

Respecto a la distribución de archivos, la figura 4.9 muestra la dependencia de las clases de ReactJS creadas.

4.3.3. Muro de desafíos

A continuación se detalla cada uno de los elementos que conforman el muro de desafíos:

1. **Desafíos**: La sección de desafíos tiene como objetivo mostrar una lista de las publicaciones realizadas por usuarios registrados referentes a temas relacionados con los artículos, ordenados desde el más reciente al más antiguo. Cada publicación puede contener la siguiente información:

   - Foto, nombre y cargo del autor de la publicación.

   - Fecha de la publicación.

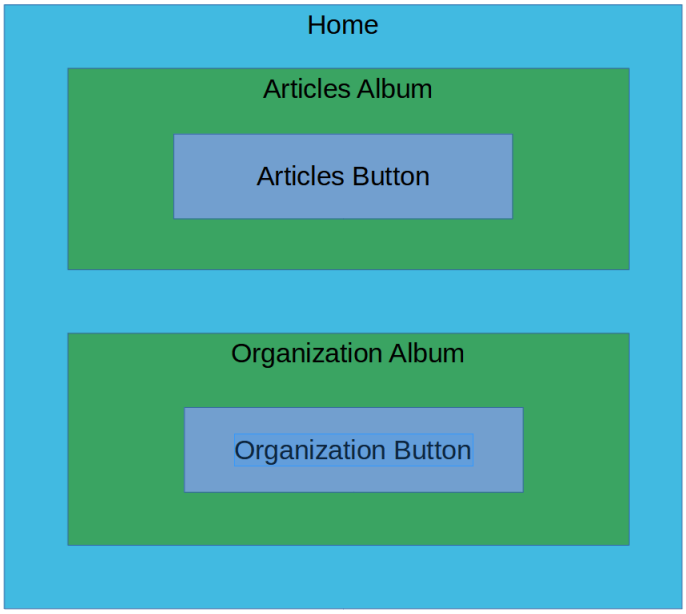

Figura 4.9: Distribución de archivos de las clases de ReactJS creadas

- Texto que responde a la pregunta ¿Cuál es el desafío?
- Texto que responde a la pregunta ¿Por qué planteas este desafío?
- Archivo asociado a la publicación.
- Cantidad de "me gusta"que los usuarios le han dado a la publicación como método de valorizarla.
- Lista de comentarios que se han realizado a la publicación.
- Formulario web para realizar un comentario.

El componente mostrado en la figura 4.10 es llamado **ChallengeButton** y todas las publicaciones de desafíos llaman a este componente y son renderizados en un contenedor padre llamado *ChallengesAlbum* encargado de delimitar la zona de cada desafío y mantener el orden responsivo de los mismos.

2. **Botón para publicar:**

   El componente que permite publicar un desafío aparece solo si el usuario está autenticado en la plataforma, de lo contrario, solo se renderiza directamente el álbum de desafíos. La figura 4.11 muestra el diseño de este botón, el cual tiene siempre el mismo ancho que columna que contiene los desafíos.

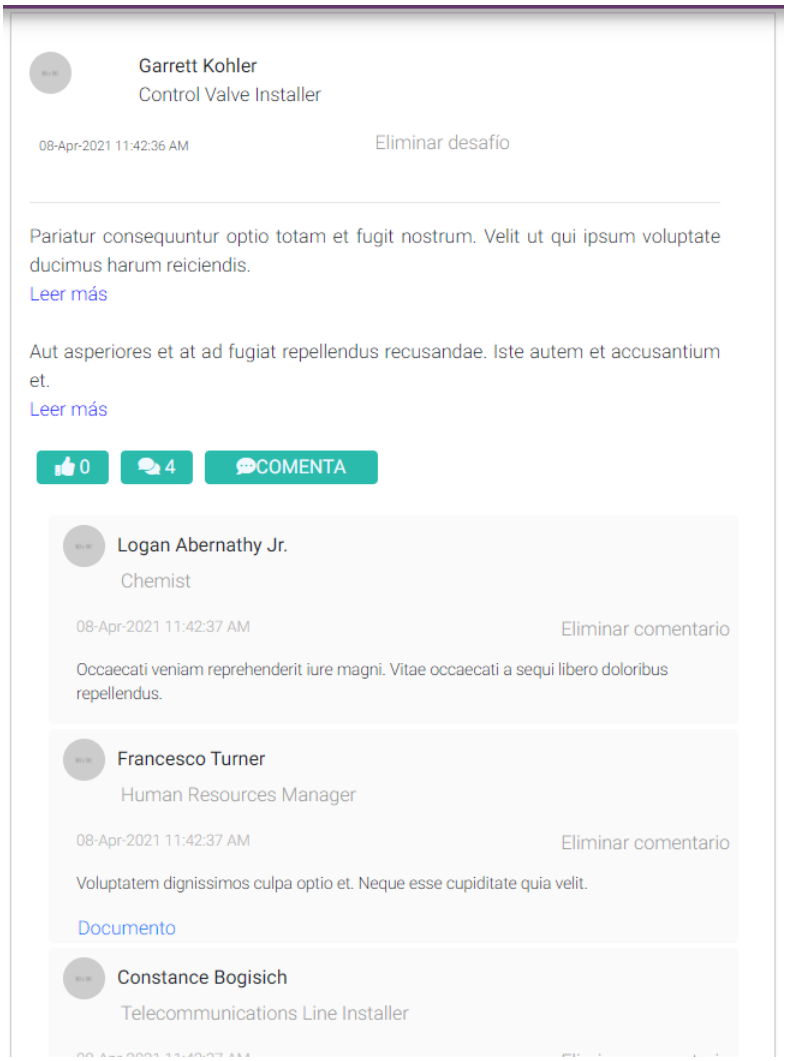

Figura 4.10: Ejemplo de estructura de un desafío

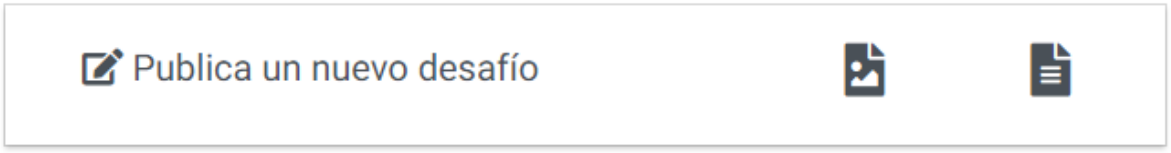

Figura 4.11: Botón para publicar un desafío

Una vez que este botón es presionado es lanzado un modal con el formulario para completar un nueva publicación de desafío. Un modal es un cuadro que aparece sobre la página, bloqueando todas las funciones para concentrar el foco en una acción particular, dejando un fondo traslúcido detrás como una cortina a toda la ventana del navegador. La figura 4.12 muestra el formulario para poder publicar un nuevo desafío y el efecto cortina que contextualiza y focaliza la atención en

esta ventana emergente.

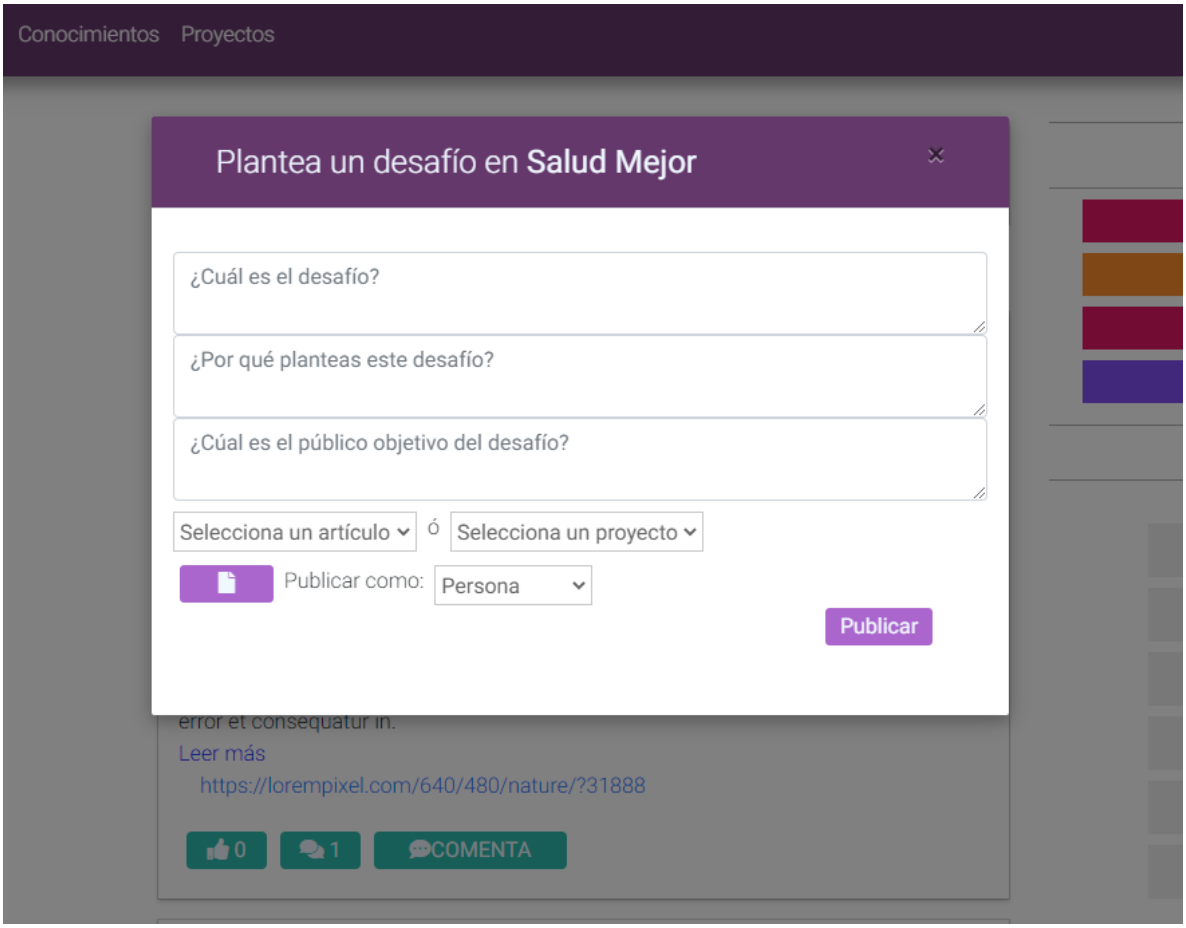

Figura 4.12: Ventana modal con formulario para publicar un desafío

3. **Panel lateral derecho:**

   Acompañando al álbum de desafíos, al costado derecho se posiciona un panel con dos elementos: Accesos directos a los desafíos en contexto de un artículo o proyecto específico y una lista con los desafíos más comentados. La figura 4.13 muestra la lista de artículos resaltando el que está con el cursor del ratón encima, manteniendo la estructura de colores que se muestra en la vitrina de artículos del Home.

   Por otra parte, se muestra la lista de proyectos, todos del mismo color para darle un toque más formal y contextual distinto.

   Para mostrar los desafíos más comentados, se utilizan las mismas clases que para mostrar los desafíos, sin embargo, la consulta que se realiza con Apollo dentro de

Figura 4.13: Panel de accesos directos para desafíos asociados a artículos o proyectos

la clase ChallengeAlbum cambia sus parámetros para que se cumpla la condición. La figura 4.14 muestra el resultado de esta sección.

Cabe señalar que los desafíos más comentados no se encierran en recuadros de modo que se le quite un poco el foco de atención respecto a la lista de desafíos recientes.

Para entender mejor la distribución de las clases se muestra en la figura 4.15 la jerarquía de los archivos involucrados.

En esta oportunidad podemos observar una de las características más importantes de ReactJS, que permite reutilizar el código ya que maneja los componentes como si fueran etiquetas HTML nuevas, las cuales pueden ser dinámicas y cambiar según las propiedades ingresadas. Por ejemplo, la clase *ChallengeAlbum* se utiliza por primera vez activando la propiedad $rendermode = 1$ en cambio en la segunda oportunidad este valor cambia a $rendermode = 2$. Al llegar al último elemento de este árbol, la clase *ChallengeButton* esta propiedad se hereda y permite elegir un retorno de la función render distinto dependiendo del valor de *rendermode*.

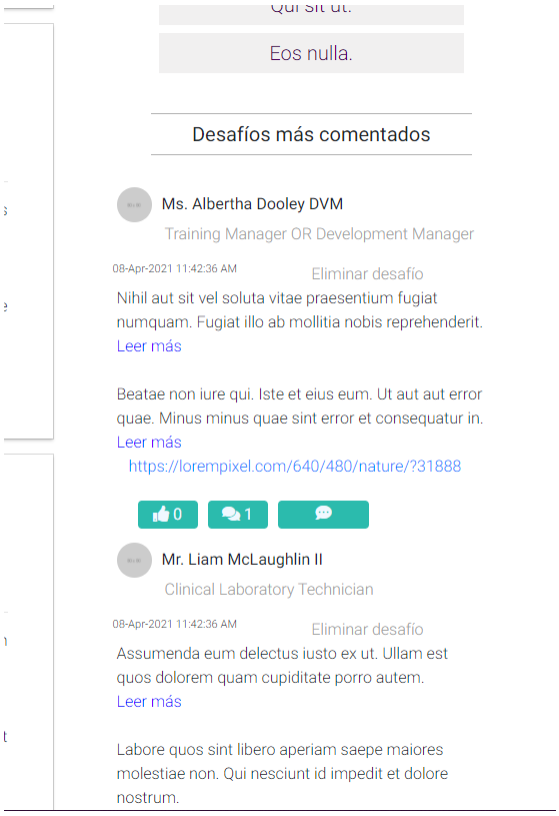

Figura 4.14: Panel de desafíos más comentados

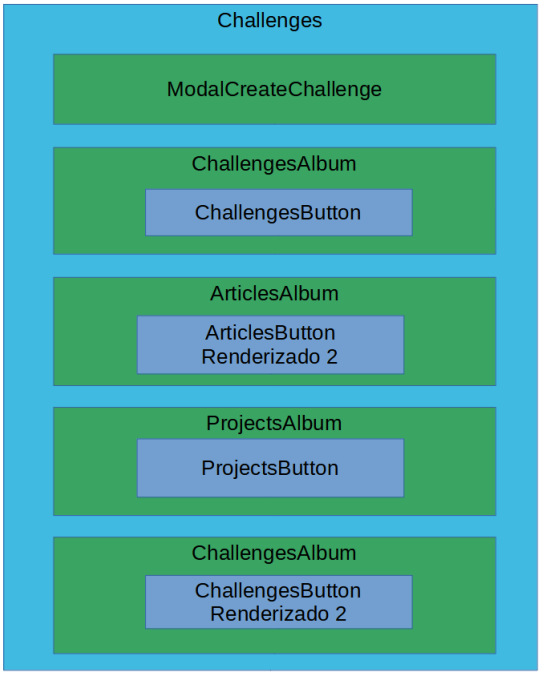

Figura 4.15: Gráfico de representación de archivos con las clases para muro de desafíos

4.4. Gestor de artículos

Esta sección del sitio está pensada para que los usuarios con permisos de administrador o generador de contenido puedan crear, editar y eliminar artículos. Su composición consta de varios elementos importantes:

4.4.1. Visor de artículos actuales

El primer elemento de este gestor corresponde a una lista de los artículos creados, donde se muestra el nombre de cada uno y dos botones asociados a la edición y eliminación del mismo. La figura 4.16 muestra el resultado de esta parte.

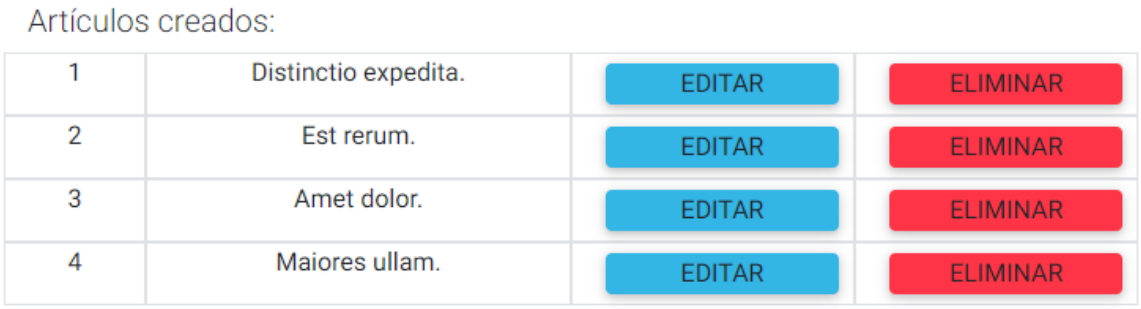

Figura 4.16: Panel de artículos creados para ser editados o eliminados

Al hacer clic en el botón EDITAR se completan los recuadros del formulario con la información contenida del artículo seleccionado. De esta manera, se puede cambiar esta información.

4.4.2. Formulario de creación

Para poder crear o editar un artículo, es necesario tener un conjunto de campos de formulario acorde a la definición del modelo de base de datos del mismo. El primer elemento corresponde al título, el cuál no puede ser mayor a 40 caracteres. Le siguen dos botones para subir archivos, uno para la imagen superior de presentación y otro para el icono que caracterizará al artículo en la vitrina principal.

Luego esta la sección para determinar a qué categoría y sub categoría pertenece. Pudiendo, además, crear una nueva categoría o sub categoría en caso de necesitarlo. Cabe señalar que las sub categorías no tienen dependencia directa de las categorías,

por lo que se pueden crear indistintamente cada una. Luego de esto, está el selector de color del artículo, con el cual se pintará el fondo de los botones de acceso al mismo..

El contenido del artículo es escrito dentro de un componente de ReactJS de terceros llamado *React Rich Text Editor*[19]. El cual entrega herramientas de edición típicas, tales como texto en negrita, cursiva e itálica, viñetas, creación de enlaces externos y tamaños de letra. Todo esto se muestra en la figura 4.17

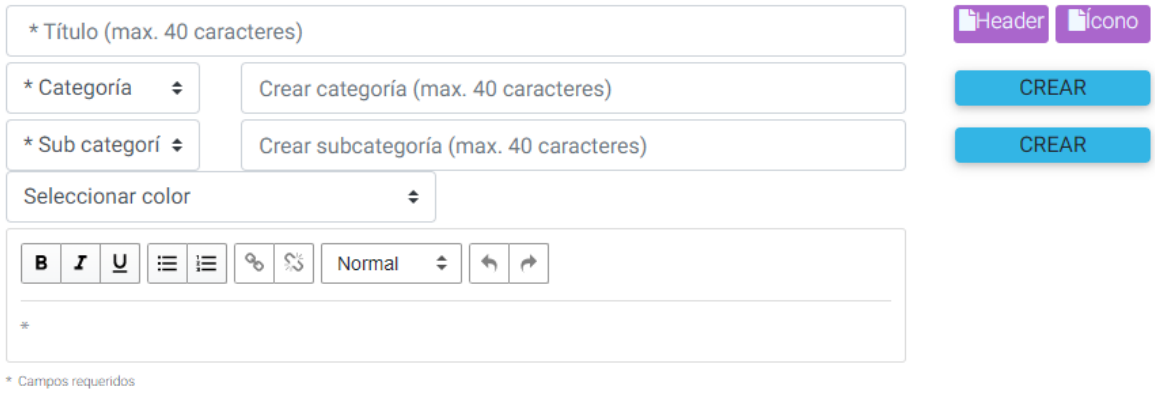

Figura 4.17: Formulario para la creación o edición de la información de un artículo

4.4.3. Creación de cifras

Finalmente, se agrega al gestor de artículos el sistema para poder asociar cifras, de modo que se entregue información cuantitativa respecto variables importantes a considerar en el tema tratado. Cada cifra tiene un título y un contenido el cual puede ser llenado con un formulario como muestra la figura 4.18.

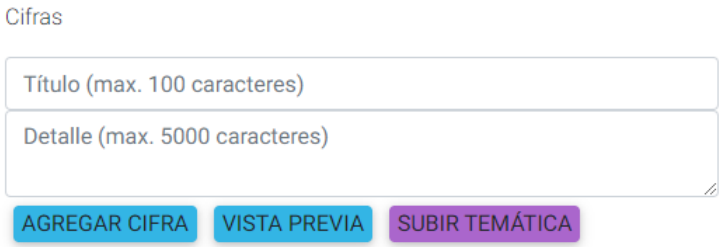

Figura 4.18: Panel de creación de cifras

[19]https://github.com/sstur/react-rte

El sistema con todos los elementos en su conjunto, puede visualizarse en la figura 4.19. Como prueba se rellenan algunos de los campos y se crean dos cifras de prueba para mostrar cómo se visualizan. También se muestran en la figura 4.20 los colores disponibles para elegir.

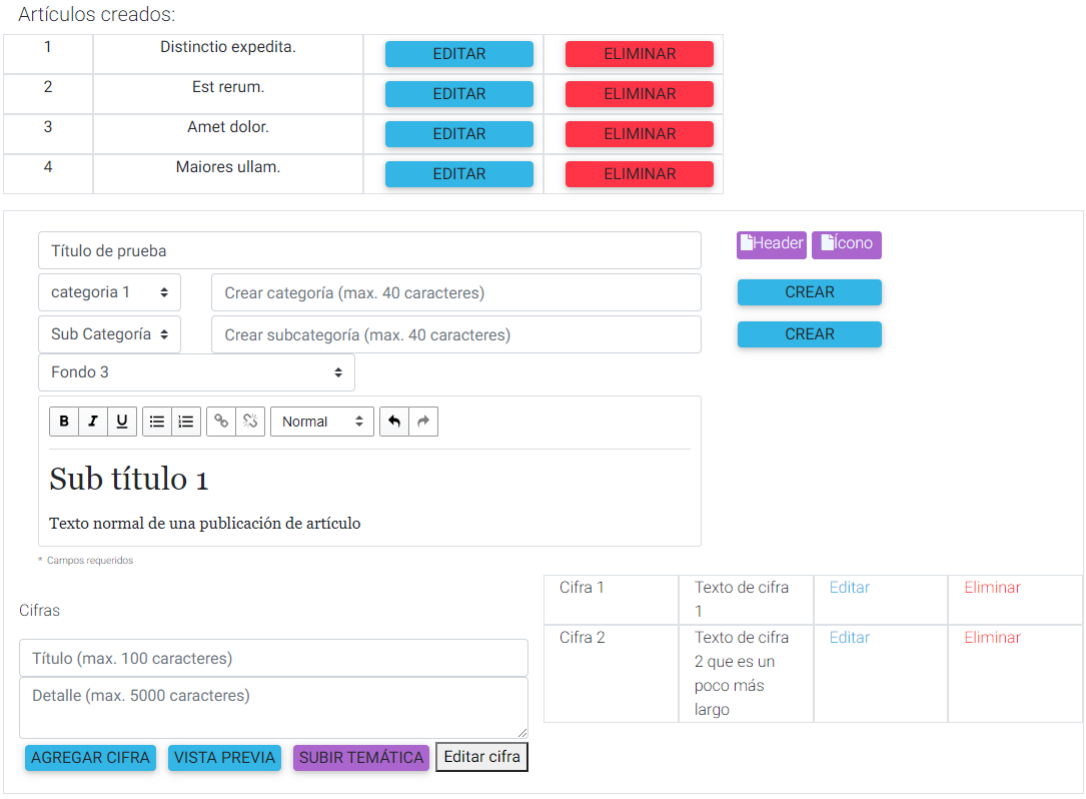

Figura 4.19: Gestor de artículos

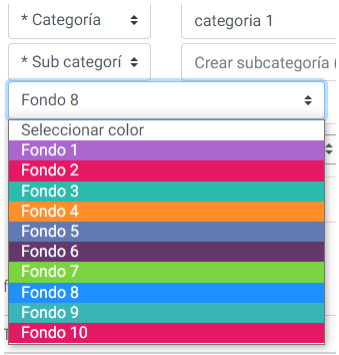

Figura 4.20: Paleta de colores para artículos

4.5. Gestor de usuarios

En panel de gestión de usuarios permite solo editar la información y eliminarlos del sistema ya que cada uno de estos entra con credenciales de otras redes sociales previamente validadas. Se genera el archivo *UsersList.jsx* que mostrará la lista, en este se realiza la consulta de todos los usuarios de la plataforma y se muestra cada uno utilizando un componente hijo llamado *DataTablePage* encargado de renderizar una tabla utilizando el componente de MDBReact *MDBDataTable*[20]. En esta tabla se muestra: El id del usuario, foto de perfil, nombre, ocupación, institución, su estado de verificación, su rango de permisos y un botón para abrir el modal de edición. La figura 4.21 muestra el resultado.

El id del usuario se genera automáticamente una vez que se realiza el inicio de sesión, la foto de perfil es extraída de este inicio de sesión con la red social de preferencia, la ocupación e institución son campos que tienen información solo si se edita al usuario correspondiente y se le asigna una de las organizaciones asociadas a la plataforma.

Figura 4.21: Lista de usuarios

Para la edición de un usuario se debe hacer clic en alguno de los botones editar. Una vez hecho esto, se abre un modal con un formulario de edición como muestra la figura 4.22.

[20]https://mdbootstrap.com/docs/react/tables/datatables/#docsTabsAPI

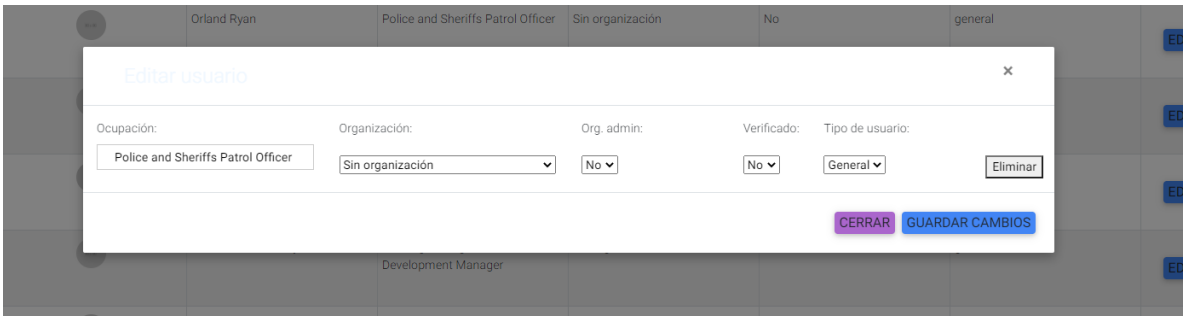

Figura 4.22: Formulario de edición de usuario

Capítulo 5

Integración del sistema y puesta en marcha

Para poder implementar el proyecto en un ambiente de producción final, es necesario realizar una serie de pasos importantes que aseguren el buen funcionamiento de la aplicación considerando los requisitos que la nube de Microsoft Azure toma en cuenta.

5.1. Repositorio Git

El primer paso para la implementación es tener el código fuente en alguna herramienta de control de versiones. En este caso se utiliza GitHub[1].

Esta herramienta permite desarrollar el código en conjunto con más personas e ir llevando un registro histórico de los cambios realizados, pudiendo tener distintas ramas de desarrollo paralelas. En el caso de este proyecto se manejan dos ramas; una llamada *master* y otra llamada *dev*. La primera tiene el código revisado, por lo tanto el registro de esta rama tiene que ser con cada cambio que haya pasado ciertos parámetros de evaluación. La segunda tiene el código en desarrollo y sus versiones no necesariamente tienen código terminado o funcional, es en esta rama donde se realizan los primeros cambios en el código y se hacen las primeras pruebas. Sin embargo, por temas de derechos de autor, el código fuente de este proyecto no puede ser publicado, ya que le pertenece al MINSAL.

[1]https://github.com/

5.2. Nube de Microsoft Azure

Azure[2] proporciona una vasta cantidad de servicios en la nube para desarrolladores. Para la implementación de este proyecto se utilizan específicamente tres recursos: Un servidor de base de datos SQL Server, un servidor web IIS de escalado rápido llamado Web App Service y una cuenta de almacenamiento para datos no estructurados que permita guardar archivos binarios grandes *BlobStorage*.

El primer paso para comenzar a preparar el entorno de ejecución productiva consiste en crear un grupo de recursos que contenga todo lo necesario. En la figura 5.1 podemos observar el formulario para la creación del mismo.

Figura 5.1: Formulario para la creación de un grupo de recursos

5.2.1. Configuración SQL Server

Para poder tener una base de datos en SQL Server se necesitan dos elementos: Un servidor SQL y una base de datos asociada a este servidor. Si queremos crear estos servicios se debe elegir en la lista de Azure la opción base de datos SQL, la cual nos llevará al panel de configuración inicial como muestra la figura 5.2.

[2]https://azure.microsoft.com/es-es/

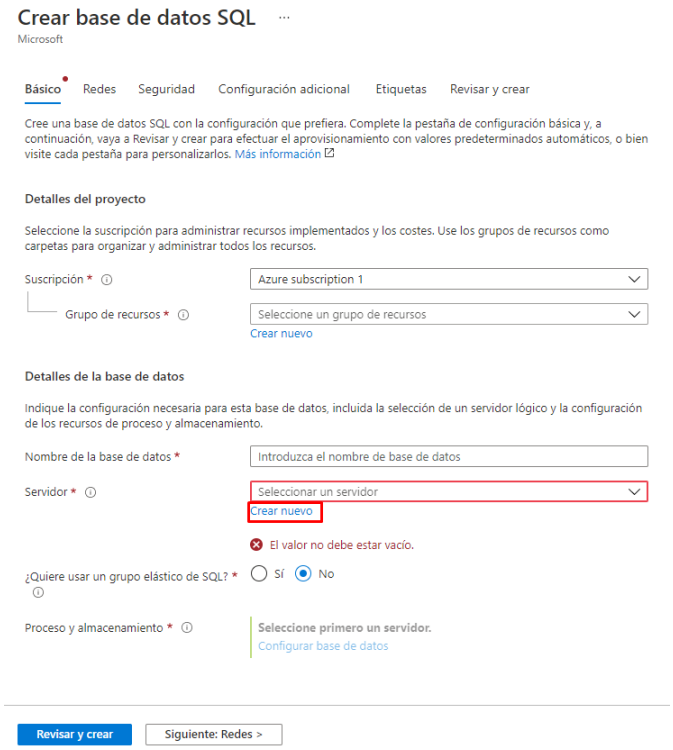

Figura 5.2: Formulario de azure para la creación de una base de datos SQL

Esta base de datos tiene que estar necesariamente asociada a un servidor SQL, el cual debe ser creado con el link encerrado en el recuadro rojo de la figura 5.2. Al hacer clic en *crear nuevo* se abre la ventana mostrada en la figura 5.3.

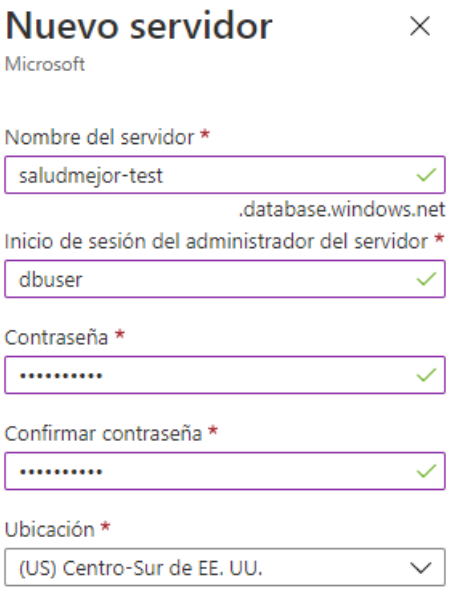

Figura 5.3: Formulario para crear un nuevo SQL Server

Un SQL Server entrega una ruta de destino para poder acceder, por lo tanto se deben tener credenciales de autenticación: nombre de usuario administrador y contraseña. Finalmente, se debe elegir la zona en la que se levantará el servidor. Esta decisión se debe tomar teniendo en cuenta en qué lugar del mundo se encontrará el cliente que accederá a nuestra base de datos.

Luego de configurar los elementos principales, las siguientes opciones tienen relación con las redes que podrán acceder a este recurso, la intercalación de los caracteres a usar según el idioma y etiquetas de uso interno para identificar al recurso.

5.2.2. Cuenta de almacenamiento

Una cuenta de Azure Storage contiene todos los objetos de datos de Azure Storage: blobs, archivos, colas, tablas y discos. La cuenta de almacenamiento proporciona un espacio de nombres único para los datos de Azure Storage que es accesible desde cualquier lugar del mundo a través de HTTP o HTTPS. Los datos de la cuenta de Azure Storage son duraderos y altamente disponibles, seguros y escalables a gran escala [3].

[3]https://docs.microsoft.com/es-es/azure/storage/common/storage-account-overview

Para comenzar, se elige el tipo de cuenta de uso general V2 ya que es la de menos costo y es compatible con blobs (Binary Large Object), lo que satisface las necesidades para nuestra aplicación. La figura 5.4 muestra el formulario para crear una nueva cuenta de almacenamiento.

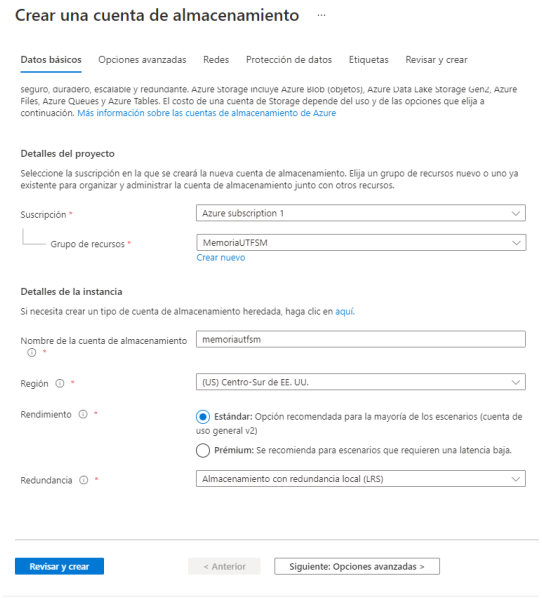

Figura 5.4: Formulario para creación de una cuenta almacenamiento

La cuenta de almacenamiento de BlobStorage entrega la oportunidad de acceder a la información por medio de REST y también a través de uan URL por cada archivo guardado, lo que la hace perfecta para una aplicación web.

5.2.3. Servidor web App Service

El servidor web se levanta en un servicio de Azura llamado App Service, por lo que para crearla basta con insertar un nuevo recurso al grupo del proyecto. Este recurso puede levantar un servidor en varios tipos de lenguajes de programación con librerías para servidores backend, PHP, NodeJS, Python, Ruby, ASP .Net y Java. En particular se elige PHP ya que el proyecto está hecho en Laravel. La figura 5.5 muestra la lista a completar para crear este recurso.

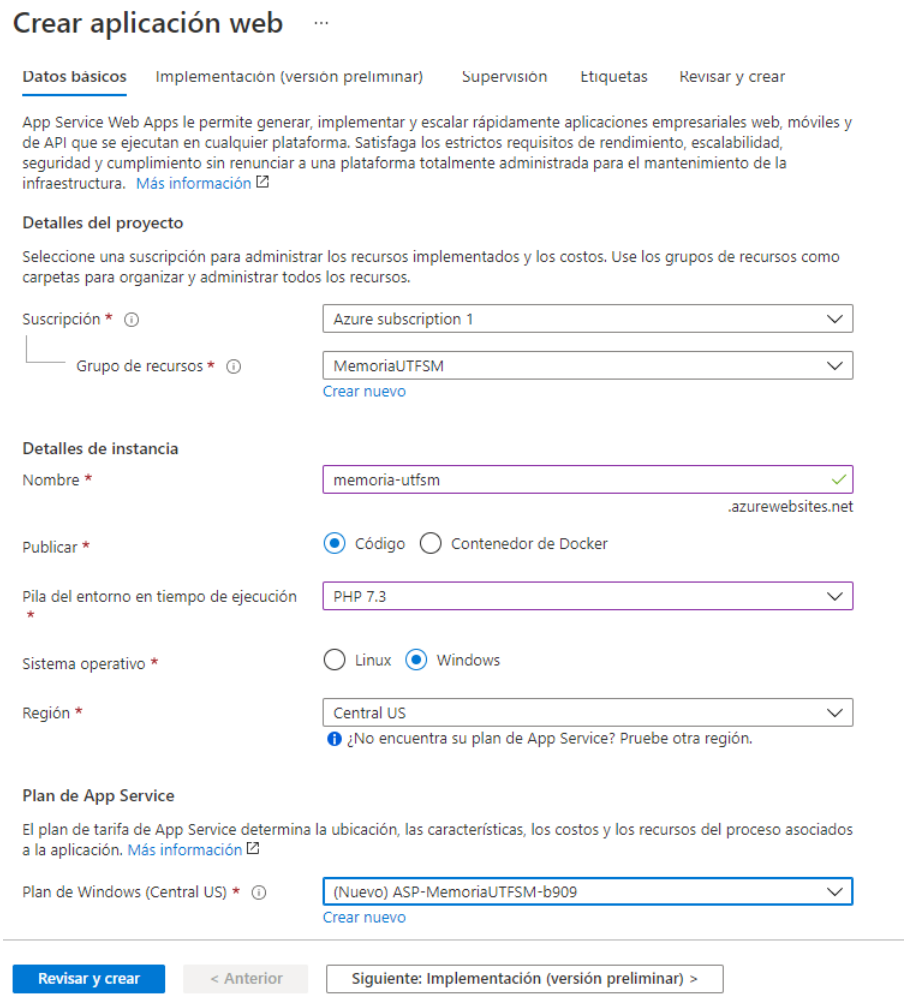

Figura 5.5: Formulario para crear una Web App

Una vez instalado este recurso, debemos configurarlo. Se debe agregar la extensión de composer desde el menú para poder instalar los paquetes y librerías PHP de Laravel, en segundo lugar se prepara el centro de implementación para reconocer el repositorio desde Git. Una vez listo esto, se debe configurar la aplicación con las variables de entorno importantes tales como la llave principal que encripta la información del servidor, las llaves de comunicación para el servicio de almacenamiento y las credenciales para la base de datos. Con esto, se debe acceder a la línea de comandos que Azure Web App entrega y correr los comandos para realizar las migraciones y comenzar a utilizar la aplicación.

Capítulo 6

Conclusión

Para crear el proyecto Salud Mejor se pasó por un proceso de diseño, desarrollo, testeo e implementación y cada uno de esos procesos tiene algo que decir respecto a las conclusiones.

6.1. Proceso de diseño

Este proceso consistió en una cantidad no menor de reuniones con el cliente en las cuales se conversaba de manera abstracta sobre lo que se quería realizar y sobre como se iba desarrollando cada hito de la planificación. Algo importante a notar de estas reuniones es que este tipo de proyectos puede escalar y seguir escalando casi infinitamente, por lo tanto, es importante acotarlo y llegar a un acuerdo con el cliente sobre los alcances reales del mismo.

Las reuniones tienen que estar enfocadas en un tema definido, por ejemplo, tratar exclusivamente la visualización del muro de desafíos o la forma en que se presentarán los artículos y que acciones puede realizar el usuario autenticado con estos. Afortunadamente, se contaba con un equipo multidisciplinario de periodistas, diseñadores, ingenieros, profesionales de la salud y políticos, lo que provocó una sinergia positiva a la hora de llegar a acuerdos.

6.2. Proceso de desarrollo

Las características del proyecto, necesitaban de una plataforma de programación que permitiera escalar fácilmente el proyecto, con un sistema de creación de bases de datos versátil y con herramientas que entreguen una fácil construcción de la capa de negocio. Es por estas razones que se elige Laravel, ya que entrega las herramientas que cumplen todo lo antes mencionados y porque el diseño de la base de datos requiere de un sistema relacional que conecte los diferentes grupos de información.

La programación del BackEnd y del FrontEnd están directamente relacionadas y tienen que conversar mutuamente ya que a veces puede pasar que una función en particular puede ser resuelta tanto por uno como el otro. Es en ese momento donde se debe decidir si se la hará la carga de la ejecución de esa función al servidor o al cliente, considerando que los usuarios pueden tener computadores o teléfonos inteligentes de gama baja con poco poder de procesamiento. En general, las funciones que dinamizan la interfaz gráfica y que requieren de una rápida reacción para hacer sentir al usuario que las cosas funcionan bien son cargadas en el código JavaScript que se envía al cliente. Por otra parte, las peticiones de información y la entrega de resultados que necesiten saber datos son cargadas al servidor.

6.3. Ambiente de desarrollo y producción

Un ambiente de desarrollo debe ser creado con un servidor local que permita emular el comportamiento que tendría el servicio una vez que esté publicado en el ambiente de producción. Sin embargo, hay que tener en cuenta que la latencia de un servidor local es mucho menor a la de uno en producción, por lo tanto es importante programar teniendo en cuenta este factor, gestionando las posibles demoras de tiempo con componentes visuales que mantengan al usuario informado y satisfecho.

Para el ambiente de producción se utiliza Azure, pero existen otras plataformas, tales como Google Cloud, Amazon Web Services, Heroku, Digital Ocean, entre otras. La decisión de usar la nube de Microsoft reside solo en el hecho de que la empresa a cargo del proyecto tiene una suscripción a esta plataforma. El servicio no está directamente en Chile, ya que el servidor más cercano está en el sur de Brasil. Sin embargo, el uso de la fibra óptica internacional permite tener buena latencia entre la comunicación con el cliente y el servidor.

Bibliografía

[1] Servicio de red social para la salud. [Online]. Available: https://healthunlocked.com/

[2] Php framework for web artisans. [Online]. Available: www.lavarel.com

[3] Mvc xerox parc 1978-79. [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

Apéndices

Apéndice A

Laravel: Instalación y configuración

A.1. Instalación

Para poder instalar laravel se requiere tener previamente en el sistema operativo lo siguiente:

- PHP - 7.1.3 o superior.
- Composer [1]

. Composer es un gestor de paquetes de PHP con el cual se pueden instalar muchas librerías de este lenguaje. Para instalar laravel se debe ejecutar el comando:

```
1    composer global require laravel/installer
```

Una vez instalado, se puede crear un nuevo proyecto de laravel llamado "saludmejorütilizando el comando:

```
1    laravel new saludmejor
```

Se incluyen entonces todas las librerías de PHP necesarias para el funcionamiento del framework.

[1] www.getcomposer.org

A.2. Configuración

La configuración se realiza en un archivo de entorno llamado *.env*. En este se agregan valores macro que serán leídos por el archivo *config/app.php y otros más*.

Script A.1: Archivo de entorno de Laravel

```
1  APP_NAME = Laravel
2  APP_ENV = local
3  APP_KEY = base64:SD8sRAFFhycn7Y1eSSpZTHI1bbu0/0
       RXhHw1K0DVIjo=
4  APP_DEBUG = true
5  APP_URL = http://localhost
6
7
8  DB_CONNECTION = mysql
9  DB_HOST = 127.0.0.1
10 DB_PORT = 3306
11 DB_DATABASE = saludmejor
12 DB_USERNAME = root
13 DB_PASSWORD = password
14
15 FACEBOOK_ID =
16 FACEBOOK_SECRET =
17 FACEBOOK_REDIRECT_URL = ${APP_URL}/auth/facebook/
       callback
18
19 GOOGLE_ID =
20 GOOGLE_SECRET =
21 GOOGLE_REDIRECT_URL = ${APP_URL}/auth/google/callback
```

En este archivo se configura el nombre de la aplicación, el estado del entorno (desarrollo o producción), la llave con la cual el servicio interno encriptará las consultas y por último las credenciales de la base de datos a utilizar. Para el entorno de desarrollo se utiliza un servidor local con MariaDB y phpMyAdmin de XAMPP. Para el entorno de producción se utiliza SQLServer como servicio de Azure Web Application.

Finalmente, se configuran las credenciales de las aplicaciones Facebook y Google para que *Passport* realice las conexiones de usuario.

Otro archivo importante dentro de la configuración es *composer.json*, en éste se quedas definidos todos los paquetes PHP que se están utilizando.

Script A.2: Archivo de paquetes PHP instalados

```
{
"name": "laravel/laravel",
"type": "project",
"description": "The Laravel Framework.",
"keywords": [
    "framework",
    "laravel"
],
"license": "MIT",
"require": {
    "php": "^7.1.3",
    "askedio/laravel-soft-cascade": "^5.7",
    "fideloper/proxy": "^4.0",
    "laracasts/utilities": "^3.0",
    "laravel/framework": "5.8.*",
    "laravel/passport": "^7.3",
    "laravel/socialite": "^4.0",
    "laravel/tinker": "^1.0",
    "matthewbdaly/laravel-azure-storage": "^1.2",
    "predis/predis": "^1.1",
    "rebing/graphql-laravel": "^1.20"
},
"require-dev": {
    "beyondcode/laravel-dump-server": "^1.0",
    "filp/whoops": "^2.0",
    "fzaninotto/faker": "^1.4",
    "mockery/mockery": "^1.0",
    "nunomaduro/collision": "^2.0",
    "phpunit/phpunit": "^7.0"
},
"config": {
    "optimize-autoloader": true,
    "preferred-install": "dist",
    "sort-packages": true
},
}
```