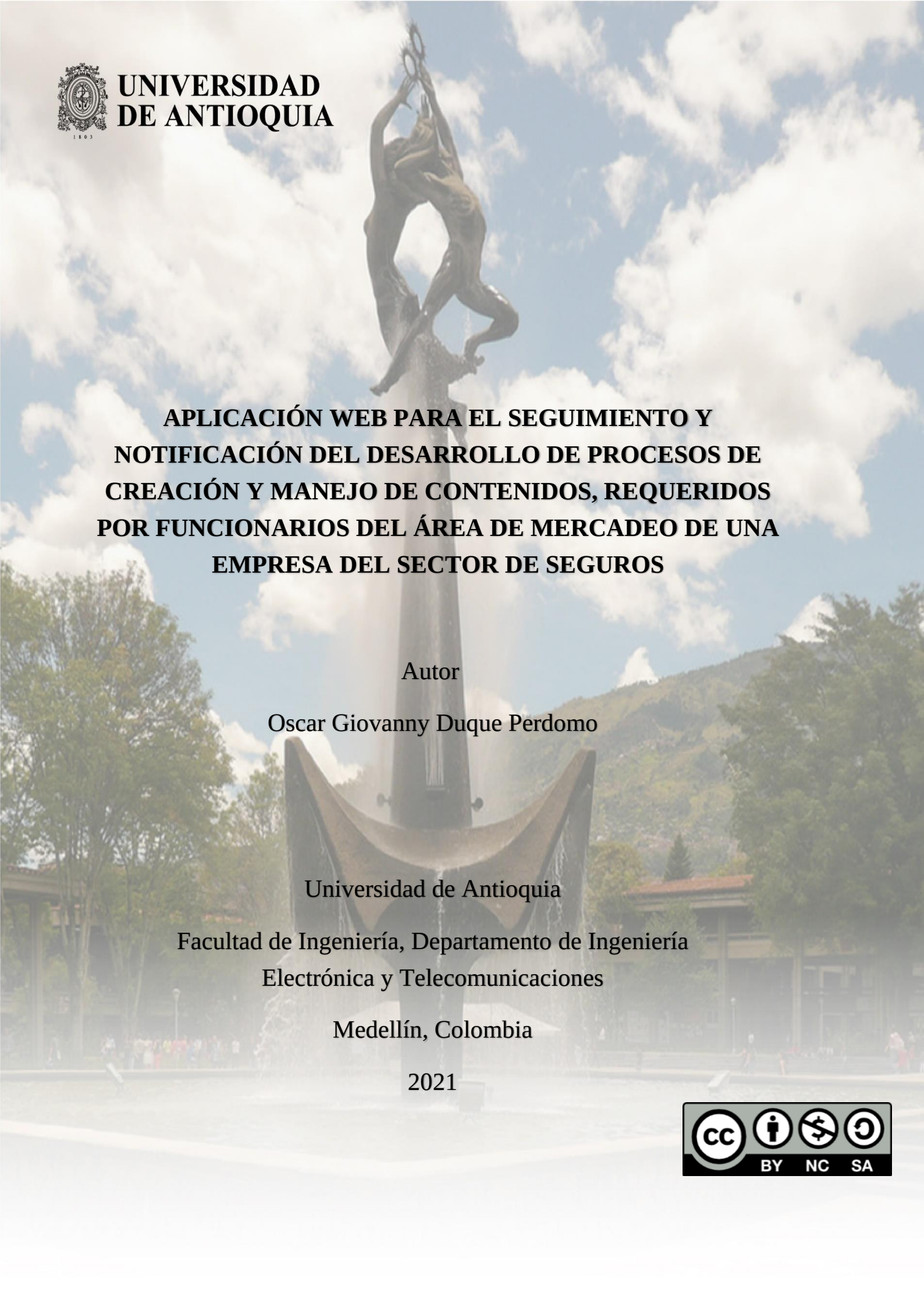

UNIVERSIDAD DE ANTIOQUIA

APLICACIÓN WEB PARA EL SEGUIMIENTO Y NOTIFICACIÓN DEL DESARROLLO DE PROCESOS DE CREACIÓN Y MANEJO DE CONTENIDOS, REQUERIDOS POR FUNCIONARIOS DEL ÁREA DE MERCADEO DE UNA EMPRESA DEL SECTOR DE SEGUROS

Autor

Oscar Giovanny Duque Perdomo

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería Electrónica y Telecomunicaciones

Medellín, Colombia

2021

Aplicación web para el seguimiento y notificación del desarrollo de procesos de creación y manejo de contenidos, requeridos por funcionarios del área de Mercadeo de la Empresa del sector de seguros

Oscar Giovanny Duque Perdomo

Informe de práctica presentado como requisito para optar al título de:

Ingeniero Electrónico

Asesores:

Gustavo Adolfo Patiño Álvarez
Profesor Universidad de Antioquia
Asesor interno

Skaidre Trakimas Gutiérrez
Líder de Equipo
Asesor externo

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería Electrónica y Telecomunicaciones
Medellín, Colombia
2021

Tabla de contenido

1. Resumen

Con el pasar de los años y la constante evolución de la tecnología, toda empresa grande, mediana o pequeña dedicada a la reparación, mantenimiento, producción o desarrollo, necesita prestar un buen servicio y esto se puede lograr en parte con una comunicación rápida y transparente con el cliente, aumentando así el prestigio y popularidad en el mercado. Cuando se presta un servicio es fundamental tener un canal de comunicaciones con el cliente para informar cuáles son las bases para generar una cuenta de cobro e informar el progreso o complicaciones del proceso. Puede suceder que una empresa que tiene a su cargo muchos procesos pierda comunicación con estos, sean o no prioritarios. Cuando esto sucede se incumple no solo con los procesos en cuestión sino con los demás procesos que deben ser detenidos para cumplir con lo pactado o explicar el motivo del retraso.

Teniendo en cuenta lo expuesto anteriormente, en la empresa Pragma S.A, específicamente en su equipo de desarrolladores dedicados a atender las solicitudes de su cliente, funcionarios del área de mercadeo de la Empresa del sector de seguros, se presentó la necesidad de optimizar la forma en que se realizan las peticiones que tiene dicho cliente, a fin de que se lograra tener de forma clara, precisa y confiable, la información del estado en el cual se encontraba su petición o solicitud realizada. Por tal motivo, en el desarrollo de esta práctica académica en la empresa Pragma S.A, se realizó una propuesta para dar una posible solución tecnológica a estas necesidades de comunicación, que presentaba la Empresa del sector de seguros respecto sus solicitudes. En este documento, se pueden encontrar los resultados obtenidos durante el desarrollo del proyecto, los cuales parten de la investigación realizada de todos los recursos necesarios para la implementación de este. El resultado final que se obtuvo fue una aplicación web, funcional, intuitiva y visualmente agradable para los usuarios que hacen uso de ella, la cual cumple con los objetivos propuestos en el proyecto. En esta aplicación se muestra una interfaz inicial donde se muestran dos formularios (usados para la recolección de datos), uno de inicio de sesión y otro de registro si es un usuario nuevo. Dependiendo del rol de la persona que va ingresar, se muestra en pantalla una interfaz personalizada. Si el usuario que ingresa es de Pragma S.A, se muestra una interfaz en específico donde se gestionan las respectivas solicitudes del Cliente (Empresa del sector de seguros) de una manera determinada, explicada posteriormente con más detalle. Después de que estas solicitudes se gestionan, se le suministra al Cliente un código compuesto de letras (en este proyecto se le asignó el nombre de "*TICKET*") asociado a su solicitud, para que este al iniciar sesión pueda consultar en qué estado está su petición.

2. Introducción

Debido a la enorme evolución que ha tenido la tecnología, todo empezó con tareas y actividades que se realizaban de forma manual dando como resultado procesos muy poco eficientes. A pesar de esto, con el transcurrir del tiempo, estas acciones se fueron optimizando al grado que, en la actualidad, la gran mayoría de las actividades se realizan de manera digital, como por ejemplo, compras de productos tales como alimentos, ropa, electrodomésticos, pago de facturas como los servicios públicos, internet, telefonía, transacciones bancarias, entre otros. Una empresa de tipo tecnológica o que ya esté adaptada al mundo digital, tiene a su cargo muchos procesos, por lo cual podría presentar diferentes problemas, uno muy significativo sería que se pierda comunicación con el cliente y sus procesos. Con este tipo de problemas han surgido diferentes tipos de soluciones como por ejemplo las que están dirigidas al seguimiento de

satélites, torneos deportivos, carreras, seguimiento de paquetes, etc. Se puede encontrar aplicaciones como: Gaia Mission la cual permite conocer el progreso de Gaia, un satélite de la Agencia Espacial Europea (ESA) que recogerá datos de mil millones de estrellas para construir un mapa en 3D de nuestra galaxia [1], también los servicios de entrega como Servientrega, Tcc, Fedex, donde se puede encontrar el recorrido de un paquete enviado. En muchos campos se pueden encontrar una gran cantidad de aplicaciones que cumplen el mismo objetivo: orientar e informar en tiempo real los progresos de cada actividad. En campos específicos se pueden encontrar este tipo de aplicaciones, pero solo dan un informe parcial y no son accesibles para todo el público.

Por lo tanto, el objetivo del presente proyecto de Semestre de Industria era crear una aplicación web que presentara una interfaz amigable e intuitiva para el cliente con el propósito de facilitar este proceso y ayudar con el flujo de trabajo acumulado de peticiones, de modo que se tenga un orden y nivel de prioridad a cada proceso en específico. Además de lo anterior, se requería que la aplicación web que se desarrolló, notificara al cliente en qué etapa iba su petición, validando constantemente con este la información que faltara. Tomando como inicio las necesidades que fueron expuestas anteriormente, se procedió con un estudio e investigación detallada de las características que poseían las tecnologías de programación necesarias para la realización de la aplicación, con el fin de analizar el mejor rendimiento respecto al objetivo final de esta. Adicionalmente se hizo un estudio de las reglas de negocio que consistían en analizar las etapas por las que pasaban las solicitudes que realiza el cliente, para tener más clara la estructura del flujo de los procesos, desde el momento de su petición hasta su entrega final. El alcance que se logró tener fue la aplicación funcionando en su manera más práctica y cumpliendo principalmente el objetivo principal del proyecto que consistía en que el cliente pudiera estar informado de los estados en los que se encontraba sus solicitudes. Las principales limitaciones que se pudieron observar durante el desarrollo de la aplicación, fue el tiempo de desarrollo que llevó estudiar las herramientas que se iban a utilizar para la implementación de esta. Adicionalmente los cambios que se hicieron en el uso de estas herramientas para el desarrollo final de la aplicación y del poco tiempo que se tuvo para implementar el proyecto en sí mismo.

La metodología que se usó se dividió de la siguiente manera:

1. Se hizo una planificación de todo lo que conllevaba realizar en el proyecto.
2. Se hizo un amplio estudio y revisión bibliográfica, con la finalidad de obtener información de las herramientas a utilizar y que mejor se adaptaban al proyecto, por cuestiones de facilidad y tiempo limitado.
3. Se hizo el desarrollo del proyecto para cumplir su objetivo final.
4. Se hizo entrega del proyecto funcionando.

Este estudio que se realizó para el desarrollo de la aplicación web tiene mucho significado en el campo de la tecnología, ya que se puede poner en práctica todos los conocimientos adquiridos para realizar cualquier tipo de aplicaciones web, lo cual puede contribuir a crear sistemas que generen solución de problemas relacionados con la comunicación, actualmente basado en el uso de la Internet. También se pueden llevar a cabo procesos que se realizan de forma manual o que no están optimizados a un sistema tecnológico para dar un enfoque más moderno.

3. Objetivos

3.1. Objetivo general

Implementar una aplicación web mediante diferentes frameworks basados en Java, que permita visualizar en tiempo real el estado detallado del desarrollo de procesos de creación y manejo de contenidos, requeridos por funcionarios del área de mercadeo de la Empresa del sector de seguros.

3.2. Objetivos específicos

- Evaluar las tareas requeridas en procesos de creación y manejo de contenidos, a fin diseñar los componentes de software necesarios en una aplicación web que permita la visualización del desarrollo de dichos procesos.

- Estructurar y gestionar los diferentes niveles de tiempo de respuesta y priorización para cada proceso de creación de contenidos requerido por la Empresa del sector de seguros. Dicha gestión se podrá realizar a partir de los criterios de prioridad definidos por la empresa.

- Implementar el aplicativo web basado en frameworks como React y Spring Boot que permita y notifique el seguimiento en tiempo real del estado de desarrollo de cada proceso.

- Crear una interfaz amigable e intuitiva que actúe como un canal de comunicación directo entre cliente-administrador, a fin de mejorar el servicio prestado actualmente.

- Evaluar el rendimiento final de la aplicación en un entorno funcional y operativo, de acuerdo con parámetros de desempeño establecidos por la empresa Pragma S.A. y la Empresa del sector de seguros.

4. Marco Teórico

Para el correcto entendimiento de este documento, es necesario tener claro los conceptos teóricos que soportaron el desarrollo del proyecto. De esta forma, en esta sección se podrá encontrar información y contexto sobre la empresa donde se desarrolló la práctica académica, los lenguajes de programación utilizados, además de algunos términos claves del desarrollo de software.

4.1. Contexto general del proyecto

4.1.1. Pragma S.A

Es una empresa de servicios y tecnologías de la información, donde se desarrollan soluciones tecnológicas para resolver los retos y las necesidades del mundo digital. Así impactan en la vida

de las personas y mejoran el potencial de sus clientes [2]. Se especializan en la transformación digital, integración y automatización de procesos, activación de marcas, portales, internet de las cosas, entre otros. La empresa se encuentra dividida en 6 Vicepresidencias que son: Mercados Ágiles, Negocios Ágiles, Equipos de Alto Desempeño, Administrativa y Financiera, Nuevos Negocios y de Talento. Es entonces en la Vicepresidencia de Negocios Ágiles, donde se encuentra el equipo que trabaja para la Empresa del sector de seguros, específicamente para el área de mercadeo, en el cual se realizó el proyecto de la aplicación web. Donde el problema que se quiso resolver con el presente proyecto, era la falta de información respecto al estado en el que se encontraban los procesos de creación y manejo de contenidos, que solicitaban los funcionarios de la Empresa del sector de seguros. Debido a esto nació la idea de realizar un proyecto de una aplicación web que llevara a cabo una solución a dicho problema, donde los funcionarios pudieran estar informados del estado en el que se encontraban los procesos que solicitaban.

4.1.2. La Empresa del sector de seguros

Es una empresa que opera principalmente en el sector Seguros. Conecta con sus contactos clave, proyectos, accionistas, noticias relacionadas y más. Esta empresa cuenta con operaciones en Colombia. Algunos temas relacionados a sus desarrollos son: Pensiones y Fondos de Pensiones. Es la segunda mayor administradora de fondos de pensiones y cesantías del país con cerca de 1,6 millones de afiliados. La empresa administra tres fondos para seguro de cesantía, pensiones voluntarias y pensiones obligatorias. Además es una sociedad anónima, sociedad de Servicios Financieros y Administradora de fondos de Pensiones y Cesantías. Su propósito es *"acompañar a sus clientes para que logren sus sueños"* [3].

4.1.3. Detalles concretos

Para dar un contexto más detallado de lo explicado anteriormente, se define como "clientes", a los funcionarios o empleados del área comercial de mercadeo de la Empresa del sector de seguros, que son los encargados de realizar las solicitudes de los procesos a la empresa Pragma S.A. Estos procesos son creación y manejo de contenidos que pasan por varias etapas (tareas) y validaciones antes de su entrega final. Para el desarrollo de esto, se requería una interfaz amigable, intuitiva y fácil de usar, tanto para el cliente como para el administrador, que se compone de cada una de las personas encargadas del proceso asignado.

4.1.4. Creación y manejo de Contenidos

La creación de contenidos es un término que en marketing digital significa producir textos, artículos, imágenes, videos, audios, entre otros, que ofrecen información o entretenimiento, y cumplen objetivos particulares de atracción de tráfico web y clientes potenciales [4].
Con esta definición establecida, se puede dar un enfoque más claro de los procesos que solicitaban los funcionarios del área de Mercadeo de la Empresa del sector de seguros, donde la creación y manejo de estos contenidos está orientado a diseñar o montar archivos PDF, artículos, imágenes, informes, cambiar estilos o funcionalidades de una página, entre otros, dependiendo de lo que solicite el cliente.

4.1.5. Descripción de Solución Propuesta

Ya que el esquema propuesto inicialmente para la solución del problema estuvo en constante evolución, construcción y adaptación, según las necesidades que se lograron evidenciar en el desarrollo del proyecto y teniendo en cuenta el problema presentado en las secciones anteriores, se muestra en la figura 1, un diagrama de bloques donde se puede observar la solución que fue implementada.

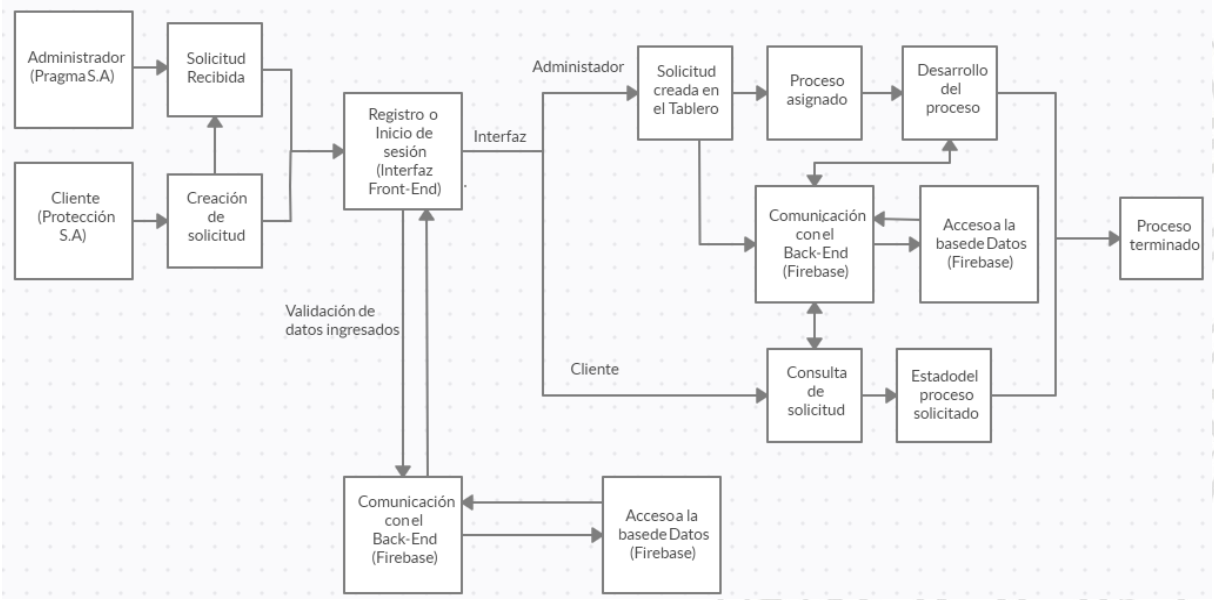

Figura 1. Sistema de solución final implementado.

En la primera etapa de bloques de la Figura 1, se puede observar que se tienen dos roles, uno para el cliente (La Empresa del sector de seguros) y otro para el Administrador (Trabajadores de Pragma), donde cada uno tiene permisos diferentes de acceso. Si es el cliente, según la lógica de negocio, lo primero que sucede es que este crea la solicitud del proceso previamente, la cual envía por correo electrónico a los Administradores. Ambos, tanto el cliente como el Administrador deben pasar por el proceso de iniciar sesión o registrarse, donde se muestra tanto la funcionalidad como la interfaz desarrollada con ReactJS en la parte *front-end*. Al intentar iniciar sesión o registrarse, los datos ingresados pasan por el *back-end*, planteado inicialmente para ser desarrollado en Spring-Boot, que es el vínculo de comunicación entre el *front-end* y la base de datos para su respectiva validación. Debido al estudio que se realizó de este y de la base de datos a emplear, se realizó un cambio en la plataforma que se usó, en este caso la elección fue Firebase (en la sección de Anexos se dará una explicación más detallada de por qué fue mejor cambiar de *framework* para el desarrollo del proyecto) tanto para el *back-end* como para la base de datos. Para validar los datos al registrarse o iniciar sesión, se muestra un mensaje de alerta si el correo ingresado a la hora del registro no es del dominio de Pragma o La Empresa del sector de seguros, entonces si una persona intenta registrarse o iniciar sesión y no es de ninguna de las dos entidades, no lo podrá hacer.

Luego de registrarse o iniciar sesión, ya sea cliente o Administrador, la interfaz que se despliega en pantalla, también desarrollada con ReactJS, es personalizada, dependiendo del rol que se tenga.Si es el Administrador, este recibe la solicitud del cliente vía correo electrónico y al iniciar sesión se despliega un tablero (componente visual que contiene entradas de texto, cajas

contenedores, botones, entre otros, se puede visualizar mejor en la sección de los Resultados), en el cual se procede a crear la solicitud que hizo el cliente. Esta solicitud se crea en una tarjeta (componente visual de caja contenedora) ingresando en "Añadir una Tarea" en el tablero, donde en esta tarjeta se especifica el nivel de prioridad de la solicitud y el nombre de esta. Cuando se genera esa tarjeta, automáticamente se genera un TICKET (definido en la sección 1 del Resumen), el cual se entrega al cliente para que pueda consultar el estado en el que se encuentra su solicitud. Esta tarjeta se puede entender mejor con la figura 2 que se muestra a continuación:

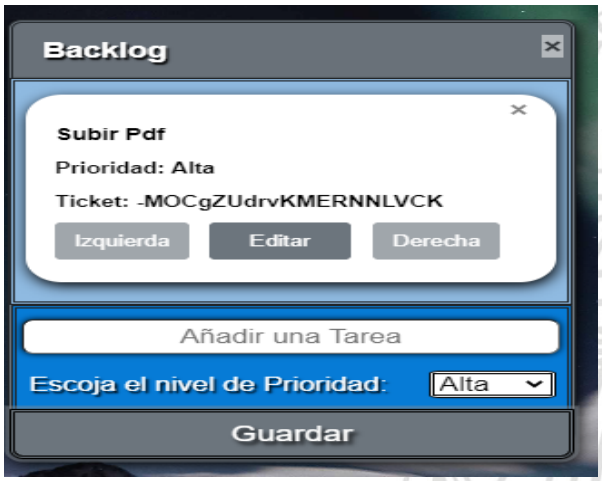

Figura 2. Ejemplo de tarjeta de una solicitud.

Los estados son creados por el Administrador y se representan en el tablero como "listas" que se muestran en forma de columna, las cuales son el contenedor de las tarjetas (solicitudes), empezando por la primera lista que se llama "Backlog". El Backlog o pila de producto en un proyecto consiste en una lista con todos los requerimientos iniciales del producto que se va a desarrollar, en este caso son las solicitudes del cliente. El Administrador también es el encargado de crear los demás estados (listas) para el seguimiento de la solicitud ya que, dependiendo del proceso escogido para realizar, pasa a una etapa de desarrollo que se subdivide en diferentes tareas. Toda la información tanto de las listas como de las tarjetas que se añaden para las solicitudes se guardan a través del *back-end* ofrecido por Firebase y se almacena en una de las bases de datos que ofrece este. A medida que van cambiando las solicitudes de estado, estas notifican en tiempo real al cliente, ya que esto funciona en tiempo real gracias a una de las bases de datos que contiene Firebase llamada "RealTime DataBase (explicado en la sección 4.8.1)", al igual que por la comunicación con el *back-end* que ofrece este mismo al realizar consultas de la información que se requiere. Una vez se realice todo el desarrollo de la solicitud pedida por el cliente, esta pasa a un estado final llamado "DONE" donde se indica que el proceso se terminó. Ahora si el cliente inicia sesión, ve una interfaz inicial donde se le pide que ingrese el Ticket asociado a su petición y que el Administrador le debió suministrar para su consulta. Si intenta ingresar un Ticket que no exista o sea errado, se muestra un mensaje de alerta con error para que ingrese de nuevo, en caso de que el Ticket sea correcto, se despliega un tablero con las columnas o listas de los estados más representativos, que se definieron para una mayor facilidad de visualización para el cliente. Además, se puede visualizar la tarjeta con el nombre de la solicitud en su respectivo estado actual, donde todas estas consultas que se hacen para saber dónde se encuentra la solicitud, se hacen a través de Firebase. Para tener mayor claridad, en las secciones posteriores se pueden ver imágenes que facilitan la comprensión de

lo explicado anteriormente. Tanto las herramientas, los términos *front-end* y *back-end* como los lenguajes serán explicados en las siguientes secciones.

4.2. HTML, CSS y JavaScript

A continuación se presenta una definición formal de los lenguajes utilizados para la realización del proyecto y sus características más relevantes para tener un contexto más claro y su relación con este. HTML (Lenguaje de marcado de hipertexto) es el componente básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, se utilizan otras tecnologías generalmente para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript) que serán definidos más adelante. Un elemento HTML se distingue de otro texto en un documento mediante "etiquetas", que consisten en el nombre del elemento rodeadas por "<" y ">". El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas. Es decir, se puede escribir en mayúsculas, minúsculas o una mezcla. Por ejemplo, la etiqueta <title> se puede escribir como <Title>, <TITLE> o de cualquier otra forma.
"Hipertexto" se refiere a enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre sitios web. Los enlaces son un aspecto fundamental de la Web. HTML utiliza "marcado" para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador web. El marcado HTML incluye "elementos" especiales como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img> y muchos otros [5]. Es necesario enfatizar que HTML no es un lenguaje de programación, sino que es un lenguaje de etiquetas que sirve para indicar cómo va ordenado el contenido de una aplicación, ya sea web o móvil.

Se menciona el lenguaje CSS (Cascading Style Sheets - CSS), debido a que este surgió ante la necesidad de facilitar el desarrollo basado en HTML, estableciendo una fuerte relación entre ellos. El lenguaje de hojas de estilo en cascada (CSS por sus siglas en inglés), se usa para estilizar los elementos escritos en un lenguaje marcado como HTML, de esta forma, CSS separa el contenido de la representación visual del sitio. La relación entre HTML y CSS es muy fuerte. Dado que HTML es un lenguaje de marcado (es decir, constituye la base de un sitio) y CSS enfatiza el estilo (toda la parte estética de un sitio web), van de la mano [6].
Por último, se tiene el lenguaje de programación JavaScript, siendo el más importante y el más potente en la web. Hoy en día, JavaScript ha crecido de manera acelerada y es el lenguaje de programación que se utiliza en casi todos los sitios web del mundo, su poder está disponible principalmente en el *front-end*, agregando mayor interactividad a la web [7]. Con este lenguaje es posible usar bibliotecas o también llamados Frameworks como ReactJS, ayudando a crear una mejor experiencia de usuario (para un mayor contexto, estos términos como "*front-end*", "Framework" y "ReactJS" serán definidos con mayor detalle en las secciones posteriores). JavaScript es un lenguaje de programación, o de secuencias de comandos, que permite implementar funciones complejas en páginas web. Cada vez que una página web hace algo más que mostrar información estática para que los usuarios la vean, muestra oportunas actualizaciones de contenido, mapas interactivos, animación de gráficos 2D/3D, desplazamiento de máquinas reproductoras de vídeo, etc., se puede dar una gran certeza que probablemente JavaScript está involucrado [8]. Es la tercera capa del pastel de las tecnologías

web estándar como se ve en la Figura 3, dos de las cuales (HTML y CSS) se han cubierto con detalle anteriormente.

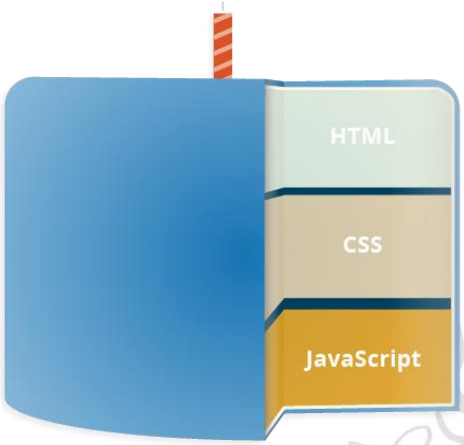

Figura 3. Ejemplo capas de la programación web.

Se puede ver de la Figura 3 que en la capa superior se encuentra HTML que muestra el contenido de una página web, luego sigue la capa de CSS que muestra los estilos visuales de esta y por último la capa de JavaScript que se encarga de la funcionalidad de la página. Esta explicación que se acaba de dar, es fundamental para la realización de la aplicación web del proyecto, debido a que se hace uso de estas 3 capas de la programación web para parte de su implementación. Ahora se debe tener en cuenta que además de esas 3 capas, la aplicación cuenta con tres partes fundamentales que se dividen en el *front-end*, el *back-end* y la Base de Datos. El *front-end* hace referencia a la parte que interactúa con los usuarios, es conocida como el lado del cliente, básicamente es todo lo que se ve en la pantalla cuando se accede a un sitio web o a una aplicación. Quienes desarrollan el *front-end* son responsables del código orientado a la inmersión del sujeto en el sitio. Para lograrlo, los expertos deben conocer los tres elementos esenciales definidos con anterioridad como son HTML, CSS y JavaScript.
El *back-end* se refiere al interior de las aplicaciones que residen en el servidor, esta parte consiste en un servidor, una aplicación y una base de datos, la cual se encarga de manejar toda la información almacenada por los usuarios [9]. La base datos permite almacenar gran número de información de una forma organizada para su futura consulta, realización de búsquedas, nuevo ingreso de datos, etc. Se puede realizar de una forma rápida y simple desde un ordenador [10]. Estas 3 partes fundamentales se profundizarán en la próxima sección.

4.3. Frameworks

Definiendo el término principal “Framework”, mencionado en repetidas ocasiones en este proyecto y utilizado constantemente en el desarrollo de software, se puede definir de manera simple como un esquema (un patrón o esqueleto) para el desarrollo y/o implementación de una determinada aplicación. También se usa para escribir código o desarrollar una aplicación de manera más fácil, garantizando mayor productividad en comparación con los métodos más convencionales usados, de modo tal que lleva a la minimización de costos mediante la reducción de horas de trabajo dedicadas al desarrollo de software [11]. Existen muchas ventajas en su uso, tales como que este no necesita plantearse una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que “rellenar”. Es más fácil encontrar herramientas (utilidades, bibliotecas) adaptadas al framework concreto para facilitar el

desarrollo. Hay otras razones más para utilizar un framework a la hora de programar, tales como [12]:

1. Evitar escribir código repetitivo: La mayoría de los proyectos tienen partes comunes necesarias para su funcionamiento, tales como, acceso a base de datos, validación de formularios o seguridad. Un framework evita tener que programar estas partes, de modo que resulta más fácil centrarse en programar la aplicación.

2. Utilizar buenas prácticas: Los frameworks están basados en patrones de desarrollo, normalmente MVC (Modelo-Vista-Controlador) que ayudan a separar los datos y la lógica de negocio de la interfaz con el usuario. Gracias a ello, se pudo aplicar en el proyecto donde se pudo obtener todo de una manera más ordenada, separando código de estilos visuales, contenido y funcionalidad.

4.4. Desarrollo *front-end, back-end* y Base de Datos

4.4.1. *Front-end*

Explicando de forma práctica el término "*front-end*", es la parte del desarrollo web que se dedica al diseño de un sitio web, desde la estructura del sitio hasta los estilos como colores, fondos, tamaños, hasta llegar a las animaciones y efectos. El *front-end* es básicamente la parte del diseño web, pero esto no significa que no toque código. Tanto el *front-end* como el *back-end* están en contacto con código todo el tiempo; dentro del área de *front-end* se trabaja con lenguajes mayormente del lado del cliente, como HTML y CSS para darle estructura y estilo al sitio; y Javascript para complementar los anteriores, y darles dinamismo a los sitios web [13]. Es bueno dejar claro y recordar nuevamente en este punto: HTML y CSS no son lenguajes de programación, son lenguajes de etiquetas y estilo. Solo Javascript es un lenguaje de programación, y este último es en donde se requiere realmente una lógica de programación. *Front-end* es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios. Un programador *front-end* debe saber de códigos HTML, CSS y JavaScript para poder usar algunos frameworks o bibliotecas que expanden sus capacidades para crear cualquier tipo de interfaces de usuarios. React, Bootstrap [14], son algunos de ellos. Donde React es el que más se estará definiendo a profundidad ya que es el principal en este proyecto.

4.4.2. *Back-end*

Después de hablar sobre *front-end* y el trabajo que desempeña, es hora de hablar del "*back-end*", que es el lado opuesto. El *back-end* es la parte del desarrollo que se dedica a la lógica de un sitio web, encargado de que todo funcione como debería. El *back-end* es la parte que de alguna manera no es visible para el usuario ya que no se trata de diseño, o elementos gráficos, se trata de programar las funciones que tendrá un sitio [15]. *Back-end* es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El *back-end* también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas. Algunos de los lenguajes de programación de *back-end* son Python, PHP, Ruby, C# y Java, y así como en *front-end*, cada uno de los anteriores tiene diferentes frameworks que

permiten trabajar mejor según el proyecto que se está desarrollando [16]. El *back-end* va desde la programación de las funciones del sitio hasta bases de datos, e incluso más. El *back-end* trabaja todo el tiempo con lenguajes de programación, lenguajes que requieren de una lógica ya que esta área es también la encargada de optimizar recursos, de la seguridad de un sitio y demás. Cosas que el usuario no ve de primeras pero que existe código detrás que está haciendo su trabajo [17].

4.4.3. Base de datos

Como se explicó brevemente al final de la sección "4.2. HTML, CSS y JavaScript", una base de datos permite almacenar gran número de información de una forma organizada para su futura consulta, realización de búsquedas, nuevo ingreso de datos, etc. Se puede realizar de una forma rápida y simple desde un ordenador. Cada base de datos se compone de una o más tablas que guardan un conjunto de datos. Éstas se dividen en columnas y filas:

- **Columnas:** Guardan una parte de la información sobre cada elemento que se quiere guardar en la tabla.

- **Filas:** Cada una conforma un registro.

Los sistemas de gestión de base de datos (SGBD, o *Database Management System*) [18] son un tipo de software muy específico, que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Está compuesto por un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de consulta. En la **Figura 4** se observan las principales características de una base de datos.

Figura 4. Características principales en una base de datos.

4.5. Frameworks de desarrollo

En la actualidad existen muchos tipos de frameworks que permiten crear aplicaciones web de alto desempeño, utilizando tecnologías web tales como HTML, CSS, JavaScript. Este es el caso de ReactJS [19], el cual se centra en la experiencia del desarrollo frontend, ya que es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. ReactJS es mantenido por Facebook y la comunidad de software libre, y se integra muy bien con plataformas en la nube como Firebase [20], para el desarrollo back-end y servicio de bases de datos en tiempo real, que sirve como intermediario entre el *front-end* y el motor de bases de datos. Uno de los objetivos principales al usar este framework es proporcionar una gama de características no funcionales que son comunes a grandes clases de proyectos (como servidores integrados, seguridad, métricas, comprobaciones de estado y configuración externa). Estas definiciones se darán a mayor profundidad en las siguientes secciones.

4.6. ReactJS

React o ReactJS es una biblioteca Javascript focalizada en el desarrollo front-end de interfaces de usuario, desarrollada inicialmente por Facebook. Es un software libre donde su creación se realizó en base a unas necesidades concretas, derivadas del desarrollo de la web de la popular red social. Además de facilitar el desarrollo ágil de componentes de interfaces de usuario, el requisito principal con el que nació React era ofrecer un elevado rendimiento, mayor que otras alternativas existentes en el mercado. Para ello, alrededor de React existe un completo ecosistema de módulos, herramientas y componentes capaces de ayudar al desarrollador a cubrir objetivos avanzados con relativamente poco esfuerzo. Por tanto, React representa una base sólida sobre la cual se puede construir casi cualquier cosa con Javascript. Además, facilita mucho el desarrollo, ya que ofrece muchas cosas ya listas, en las que no se necesita invertir tiempo de trabajo. Sirve para desarrollar aplicaciones web de una manera más ordenada y con menos código que si se usa Javascript puro o bibliotecas como jQuery centradas en la manipulación del DOM (abreviatura de Document Object Model[1]). Permite que las vistas se asocien con los datos, de modo que si cambian los datos, también cambian las vistas. Con respecto a bibliotecas sencillas como jQuery, React aporta una serie de posibilidades muy importantes. Al tener las vistas asociadas a los datos, no se necesita escribir código para manipular la página cuando los datos cambian. Esta parte en bibliotecas sencillas es muy laboriosa de conseguir y es algo que React hace automáticamente.
También, en comparación con jQuery, permite una arquitectura de desarrollo más avanzada, con diversos beneficios como la encapsulación del código en componentes, que ofrecen una serie de ventajas más importantes que los *plugin* (aplicación o programa informático que se relaciona con otra para agregarle una función nueva y generalmente muy específica), como la posibilidad de que esos componentes interaccionen entre sí, algo que sería muy difícil de conseguir con *plugins*. ReactJS solapa por completo las funcionalidades de jQuery, por lo que

1 DOM (Document Object Model) es la estructura de objetos que genera el navegador cuando se carga un documento y se puede alterar mediante Javascript para cambiar dinámicamente los contenidos y aspecto de la página [22]

resulta una evolución natural para todos los sitios que usan esa biblioteca. Podrían convivir pero no es algo que realmente sea necesario y recargaría un poco la página, por lo que tampoco sería muy recomendable [21].

Por otro lado, en comparación con frameworks como Angular [23], React se queda a mitad de camino, pues no incluye todo lo que suele ofrecer un framework completo. Pero a partir de todo el ecosistema de React se llega más o menos a las mismas funcionalidades, así que se puede considerar como una buena alternativa. No se puede decir de una manera objetiva si es ReactJS es mejor o peor que otras alternativas, porque eso ya entra más en el terreno de la opinión. La diferencia es que React le pone más inteligencia a la necesidad de actualizar una vista cuando es necesario y lo consigue mediante el "DOM Virtual" o "Virtual DOM". A lo largo del estudio de React se hablará varias veces sobre el concepto de "Virtual DOM", que es una de las principales características de React. De momento, en líneas generales se puede decir que el virtual DOM es una representación del DOM pero en memoria, que usa React para aumentar sensiblemente el rendimiento de los componentes y aplicaciones *front-end*. El Virtual DOM se basa en una idea bastante sencilla e ingeniosa. Básicamente hace que, cuando se actualiza una vista, React se encargue de actualizar el DOM Virtual, que es mucho más rápido que actualizar el DOM del navegador (DOM real). Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera. Es algo muy potente, pero que se hace de manera transparente para el desarrollador, que no necesita intervenir en nada para alcanzar ese mayor rendimiento de la aplicación [21]. Como se muestra en la **Figura 1** de la Descripción de la Solución propuesta en la sección **4.1.5**, en este proyecto, ReactJS se utilizó para crear las interfaces visuales con las que interacciona el usuario al usar la aplicación web. Cuya implementación será explicada en la sección 5.2.1 de la Metodología. De igual modo, en este proyecto se utilizó el concepto de DOM virtual para dar contexto de cómo funciona en segundo plano la visualización de elementos cada vez que se carga una vista en específico de la aplicación web, esto es algo propio de ReactJS que se ejecuta en su funcionamiento interno y que no tenía que ser implementado en el proyecto.

4.6.1. Composición de componentes

Se definen los ”*componentes*” en ReactJS, como las funciones de JavaScript. Aceptan entradas arbitrarias (llamadas propiedades o “props”) y devuelven a React elementos que describen lo que debe aparecer en la pantalla [24]. Así como en programación funcional se pasan funciones como parámetros para resolver problemas más complejos, creando lo que se conoce como composición funcional, en ReactJS se puede aplicar este mismo patrón mediante la composición de componentes. Las aplicaciones se realizan con la composición de varios componentes. Estos componentes encapsulan un comportamiento, una vista y un estado. Pueden ser muy complejos, pero es algo de lo que no hay necesidad de preocuparse cuando se está desarrollando la aplicación, porque el comportamiento queda dentro del componente y no hay que complicarse por esto una vez se ha realizado.
En resumen, al desarrollar se crearán componentes para resolver pequeños problemas, que por ser pequeños son más fáciles de resolver y son más fáciles de visualizar y comprender. Luego, unos componentes se apoyarán en otros para resolver problemas mayores y al final la aplicación será un conjunto de componentes que trabajan entre sí [25]. Este modelo de trabajo tiene varias ventajas, como la facilidad de mantenimiento, depuración, escalabilidad, etc. Y fue tenido en

cuenta en este proyecto para la creación de componentes o elementos que eran reutilizables en diferentes visualizaciones de la aplicación.

4.6.2. Desarrollo Declarativo vs Imperativo

En la experiencia de desarrollo con bibliotecas más sencillas como jQuery, o el propio "Vanilla Javascript (Javascript puro)" se realiza un estilo de programación imperativo. En ese estilo se realizan scripts que paso por paso tienen que informar sobre qué acciones o cambios en el DOM se deben realizar. Hay que ser muy concisos en esas acciones, especificando con detalle cada uno de los cambios que se quieren realizar. La forma imperativa de declarar obliga a escribir mucho código, porque cada pequeño cambio se debe definir en un script y cuando el cambio puede ser provocado desde muchos sitios, cuando se agregan eventos, el código comienza a ser poco mantenible. Sin embargo, el estilo de ReactJS es más declarativo, en el que se cuenta con un estado de la aplicación y sus componentes reaccionan ante el cambio de ese estado. Los componentes tienen una funcionalidad dada y cuando cambia una de sus propiedades ellos producen un cambio. En el código de una aplicación, se tendrá ese componente, y en él se declarará de dónde vienen los datos que él necesita para realizar su comportamiento. Se podrá usar tantas veces como se quiera declarando que se quiere usar y declarando también los datos que él necesita para funcionar [25]. En el caso del proyecto descrito en este documento, el estilo declarativo de React fue el implementado para la realización de este, ya que se usaron componentes funcionales donde por ejemplo se solicitaban datos, que al ingresarlos y enviarlos producían un cambio de estado. Este cambio de estado se veía reflejado en un cambio visual para el usuario, ya sea en la creación de elementos nuevos o cambio de vista de la página.

4.6.3. Performance gracias al DOM Virtual

El desempeño de React es muy alto, gracias a su funcionamiento. Se refiere al desempeño a la hora del renderizado de la aplicación. El renderizado, al hablar de páginas o aplicaciones web, se refiere al proceso de mostrar en el navegador la página correspondiente. En un sitio web, el servidor tiene simplemente archivos HTML, y se los manda al cliente cuando los pide. El navegador se encarga de analizar el HTML y mostrar el resultado en pantalla. Esto que hace el navegador es el renderizado. Esto se consigue por medio del DOM Virtual. No es que React no opere con el DOM real del navegador, pero sus operaciones las realiza antes sobre el DOM Virtual, que es mucho más rápido. El DOM Virtual está cargado en memoria y gracias a la herramienta que diferencia entre él y el real, el DOM del navegador se actualiza. El resultado es que estas operaciones permiten actualizaciones de hasta 60 frames por segundo, lo que producen aplicaciones muy fluidas, con movimientos suavizados [25].

4.6.4. Elementos JSX

ReactJS no retorna HTML. El código embebido dentro de Javascript, parece HTML pero realmente es JSX [25]. Son como funciones Javascript, pero expresadas mediante una sintaxis propia de React llamada JSX. Lo que produce son elementos en memoria y no elementos del DOM tradicional, con lo cual las funciones no ocupan tiempo en producir pesados objetos del navegador, sino simplemente elementos de un DOM virtual. Todo esto es mucho más rápido.
La primera parte de una etiqueta JSX determina el tipo del elemento React. Los tipos en mayúsculas indican que la etiqueta JSX se refiere a un componente React. Estas etiquetas se

compilan en una referencia directa a la variable nombrada, por lo que si se usa la expresión JSX <Foo />, Foo debe estar dentro del alcance [25]. JSX es la sintaxis usada en todo el proyecto ya que es propia React al no retornar HTML, como se puede observar en la **sección 5.4** de la Metodología.

4.6.5. Componentes con y sin estado

React permite crear componentes de diversas maneras, pero hay una diferencia entre componentes con y sin estado. Los componentes llamados *stateless* son los componentes que no tienen estado, digamos que no guardan datos en su memoria. Eso no quiere decir que no puedan recibir valores de propiedades(*props*), pero esas propiedades siempre las llevarán a las vistas sin producir un estado dentro del componente. Estos componentes sin estado se pueden escribir con una sencilla función que retorna el JSX que el componente debe representar en la página. Este tipo de componentes se definen como funciones en vanilla js [25] (JavaScript puro) y no tienen ni trabajan con estado. Los únicos datos con los que trabajan este tipo de componentes son con las *props* (propiedades) recibidas, además no permite trabajar con sobrescribir los métodos de su ciclo de vida. Las ventajas de este tipo de componentes es que son sencillos de escribir, fácilmente testeables y mejoran el rendimiento [26]. Este tipo de componentes son los más simples, pues sólo se utilizan para representar la información que les llega como parámetros. En algunas ocasiones, estos componentes pueden transformar la información con el único objetivo de mostrarla en un formato más amigable, sin embargo, estos componentes no consumen datos de fuentes externas ni modifican la información que se les envía. Las propiedades o simplemente *props*, son parámetros que se le envían a un componente durante su creación. Los *props* pueden ser datos para mostrar o información para inicializar el estado. Como regla general, las *props* son inmutables, lo que quieres decir, que son de solo lectura. Para obtener las propiedades de un componente es necesario obtenerlas mediante el prefijo *this.props* , seguido del nombre de la propiedad [27].

Los componentes llamados *statefull* son un poco más complejos, porque son capaces de guardar un estado. Su principal diferencia es que se escriben en el código de una manera más compleja, generalmente por medio de una clase ES6 [25] (el más reciente estándar del popular lenguaje de programación Javascript conocido como ECMAScript 2015), en la que podemos tener atributos y métodos para realizar todo tipo de operaciones. Los componentes *statefull*, con estado, necesitan tener un método *render()* que se encarga de devolver el JSX que usa para representarlo en la página. Los *statefull* se usan para resolver problemas mayores, mientras que los *stateless* se usan más para interfaces de usuario. En la figura 5 se observa el método *render()* de uno de los componentes *statefull* desarrollados en el presente proyecto:

```
render() {
  return (
    <div>
      {this._getTemplate()}
    </div>
  );
}
```

Figura 5. Método render().

Donde el método *render()* devuelve un componente visual dado por la función "*_getTemplate()*". Esta función contiene una porción de código donde según su lógica hay un cambio de estado dependiendo de las acciones que tenga el usuario.
Los componentes con estado se distinguen de los anteriores, debido a que estos tienen un estado asociado al componente, el cual se manipula a medida que el usuario interactúa con la aplicación. Este tipo de componentes en ocasiones consumen servicios externos para recuperar o modificar la información. Un ejemplo típico de componentes con estados, son los *formularios*, pues es necesario ligar cada campo a una propiedad del estado, el cual, al modificar los campos afecta directamente al estado [27]. Los formularios en el contexto del proyecto es un conjunto de botones, entradas de texto, casillas de verificación, etc, que permiten introducir datos y enviarlos al servidor web para su procesamiento. En la figura 6 se observa un ejemplo de *formulario* desarrollado en el presente proyecto, cuya implementación se explica en la sección 5.2.1 de la Metodología.

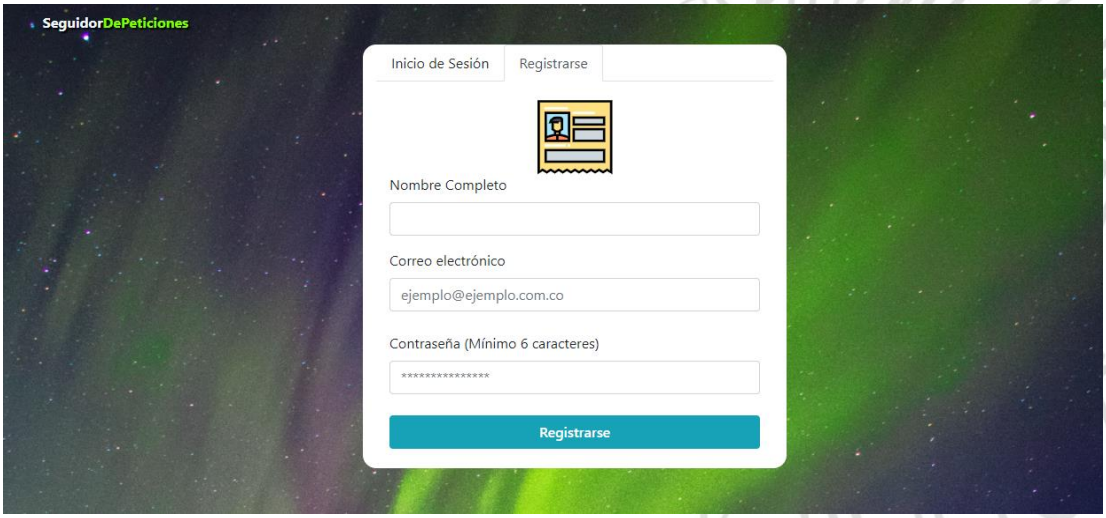

Figura 6. Formulario de Registro.

4.6.6. Ciclo de vida de los componentes

React implementa un ciclo de vida para los componentes. Son métodos que se ejecutan cuando pasan cosas comunes con el componente. A partir de la versión 16.3 de React se agregaron nuevos métodos del ciclo de vida de un componente para mejorar el rendimiento, buenas prácticas y así obtener una mejor calidad de los componentes creados [28]. Principalmente este cambio es debido a componentes con funcionalidad asíncrona, esto es muy importante porque normalmente el mundo real es asíncrono y los componentes que se crean son utilizados en el mundo real por personas. Los siguientes métodos son los recomendados a utilizar en un componente y fueron los más usados en el proyecto:

- constructor()
- render()
- componentDidMount()

constructor()

El constructor es un método de la mayoría de los lenguajes de programación orientada a objetos, y se utiliza para crear la instancia de una clase. En React, el constructor se usa para crear la instancia del componente. Cabe mencionar que después de la ejecución de este método, el componente aún no se muestra en el navegador, este proceso de mostrar, es decir, insertarlo en el DOM, se le llama Montar o Mount en inglés. Como buena práctica de programación es importante ejecutar *super()* dentro de un constructor para que realice cualquier llamada a los padres constructores [28]. En el caso de React se debe llamar con las *props* recibidas en el constructor, o sea, *super(props).* Esto permite poder acceder a las *props* a través de *this.props* dentro del constructor. En la figura 6 se indica un ejemplo del uso de estas instrucciones. El estado de un componente en el constructor se define así:

```
constructor (props) {
    super(props);
    this.state = {
        propiedad: 'Algún valor'
    }
}
```

Figura 7. Definición del constructor en React.

Si no se define ningún estado en el constructor, entonces no se necesita.
El constructor se usa normalmente para las siguientes dos cosas:

- Definir el estado local con un componente a través de *this.state.*
- Para enlazar el objeto *this*(la instancia del componente) a los métodos que son utilizados en el método *render().* Estos métodos son usados como manejadores de eventos.

Para enlazar la referencia de la instancia de un componente a los métodos que son utilizados en el método render() y que normalmente son los manejadores de eventos, en la figura 8 se observa un ejemplo de estos modos de usar el constructor en React [28]:

```
class Padre extends React.Component {
    constructor (props) {
        super(props);
        this.state = { src: '' }
        this.cambiarAnimal = this.cambiarAnimal.bind(this);
    }

    cambiarAnimal () {
        this.setState({
            src: 'Algúna url que apunte a una imagen de un animal'
        });
    }

    render() {
        return (
          <div>
            <Animal src={this.state.src}/>
            <button onClick={this.cambiarAnimal}>Cambiar animal</button>
          </div>
        );
    }
}
```

Figura 8. Ejemplo del constructor en React.

Se analiza que el método *this.cambiarAnimal* podrá acceder a la instancia del componente a través de *this* y así utilizar *this.setState()* para cambiar el estado.

render()

Este método es obligatorio en cualquier componente, pues como su nombre lo dice, se utiliza para obtener los elementos finales a visualizar o pintar en el navegador. Debe ser una función pura, es decir, no debe modificar las *props*, no debe modificar el *state* ni realizar operaciones del DOM. El método Render permite ayudar a React a visualizar y renderizar lo que se ha escrito. Todos los componentes tienen el método render, que retorna la interfaz (HTML).
En el método render solo se puede ver la interfaz en pantalla hasta que se empiece a retornar(return) [28]. En el ejemplo de la Figura 9, se puede observar que el método *render()* retorna en la interfaz una imagen creada con la etiqueta <img>.

```
...
render() {
    return (
        <img className="cat__img"
          src={this.state.src}
          />
     );
}
...
```

Figura 9. Ejemplo del método render().

componentDidMount()

Este método se ejecuta cuando el componente está listo en el DOM, eso quiere decir que siguiendo el razonamiento explicado en el método render(), se ejecuta después de que React inserte o modifique el DOM con *ReactDOM.render(),* por eso es útil tanto para realizar operaciones con el DOM, como agregar eventos y/o modificar elementos internos. Dentro de este método es seguro cambiar el *state* o estado, pero si se ejecuta *this.setState()* provocará que el componente se vuelva a renderizar [28]. La documentación oficial de React [29] advierte tener cuidado con esto, pues puede causar problemas de rendimiento por renderizar el componente varias veces; sin embargo, es necesario para los casos de tomar medidas y posiciones de algunos elementos antes de renderizar, por ejemplo, el tamaño y posición de modales, etc. Para ver el uso de este método se explica el siguiente ejemplo de la figura 10:

Figura 10. Ejemplo del método componentDidMount().

Si se revisa la consola del navegador, se verá que render() se ejecuta dos veces, también si se da click en el botón Cambiar animal, se nota que de nuevo render() se ejecuta dos veces. ¿Por qué sucede esto?, sucede porque dentro del método *componentDidMount()* se agrega un escuchador de eventos para la carga de la imagen del animal, al ejecutarse *this.onImgLoad()* dentro se ejecuta *this.setState()* y esta función provoca que el componente se vuelva a renderizar para mostrar las medidas exactas de la imagen cuando se termina de cargar.
En la sección 5.4 de la Metodología se describe el modo y uso de este método dentro de la aplicación desarrollada en este proyecto.

4.6.7. Chrome DevTools

Es un conjunto de herramientas de creación web y depuración integrado en Google Chrome . El uso de DevTools permite depurar el sitio y crear un perfil de él. Cuenta con la capacidad de depurar código, incluso corriendo la aplicación directamente desde el celular o usando el *localhost*, con una excelente calidad de respuesta. Además, DevTools deja iterar la distribución y el diseño del sitio mediante la libre manipulación del DOM y CSS, también contiene un panel llamado *Console* para registrar información de diagnóstico durante el desarrollo, o se puede usar para interactuar con el código JavaScript de la página. Adicionalmente, cuenta con otros paneles que permiten ubicar puntos de interrupción en el código y así depurarlo; o para obtener información sobre recursos solicitados y descargados. También para explorar los eventos y así

mejorar el rendimiento de ejecución de la página [30]. En el proyecto fue utilizado principalmente para ayudar a corregir posibles errores que salían en la consola del navegador como por ejemplo el que se ve en la figura 11 y a modificar el estilo visual de los componentes de la aplicación como se ve en la figura 12:

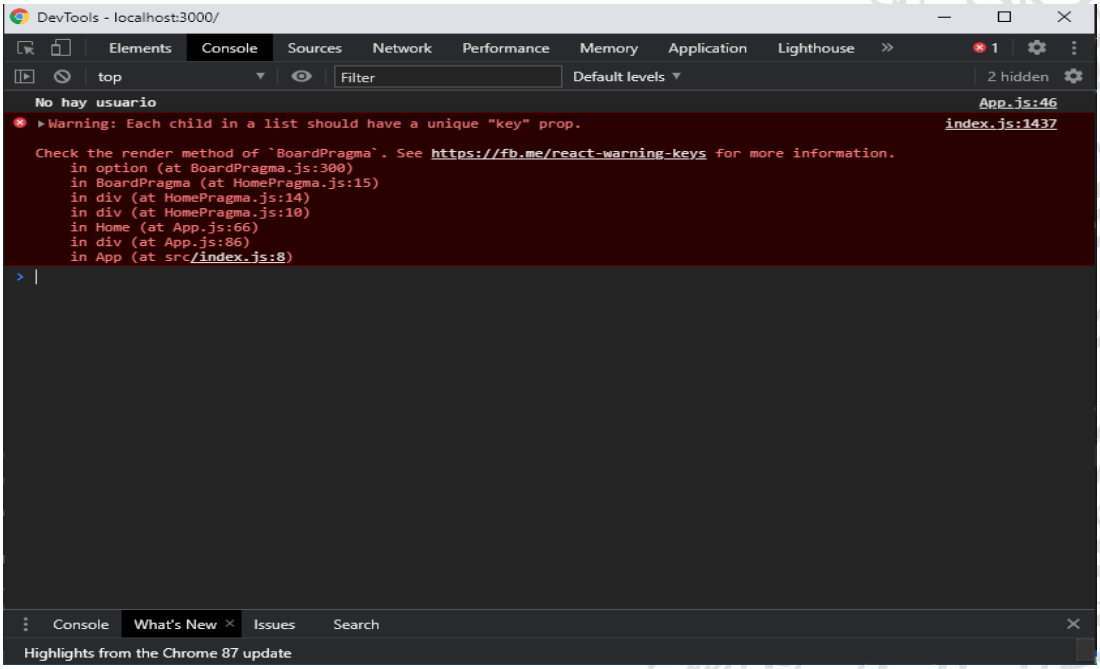

Figura 11. Error en Consola de Google Chrome.

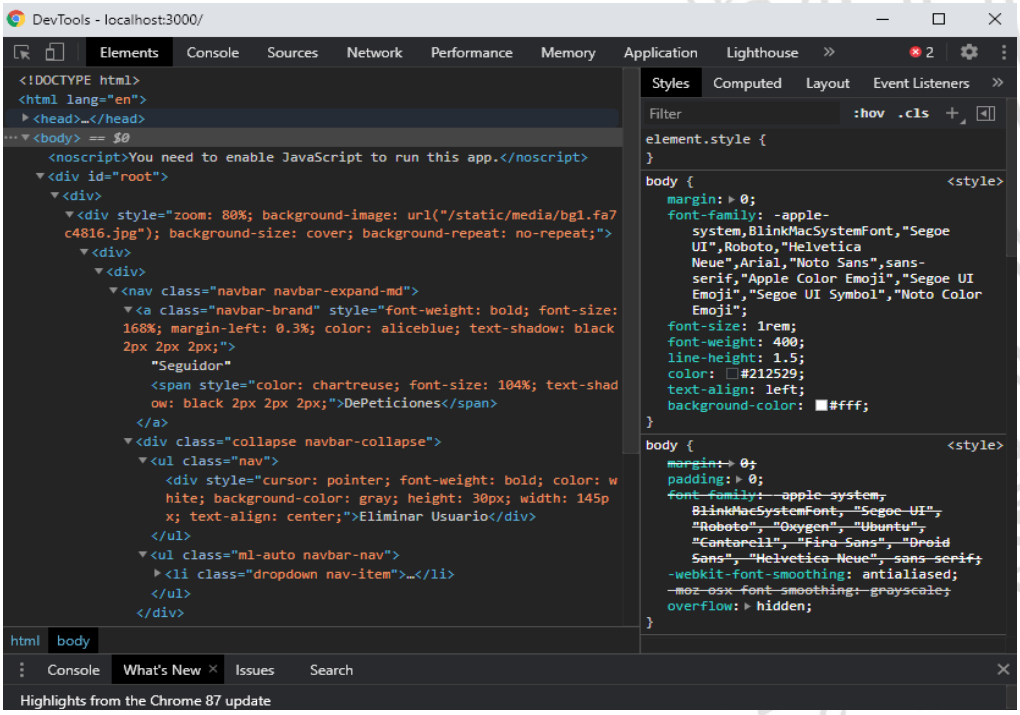

Figura 12. Ejemplo modificación de estilos.

Como se puede observar, en la parte lateral derecha se encuentra una pestaña llamada “Styles”, donde se encuentran todas las propiedades de los estilos CSS que se pueden modificar, dependiendo del componente en pantalla que se escoja para modificar su estilo.

4.6.8. ReactJS aplicado al proyecto

Con ReactJS se crearon las interfaces de usuario que fueron:

- El formulario de registro e inicio de sesión.
- La barra superior de cierre de sesión.
- El tablero donde el cliente de La Empresa del sector de seguros puede visualizar el estado de sus solicitudes mediante un TICKET que se le entrega para realizar su consulta.
- El tablero de Pragma donde se asignan las tareas requeridas por el cliente y estas generan un TICKET el cual es entregado a este para su posterior consulta.

Con ReactJS se desarrolló toda la parte Front-End de la aplicación con todos los estilos visuales CSS y la parte funcional.

4.7. Spring Boot y AWS

Después de analizar cuál era la mejor opción que se iba a utilizar para el *back-end* (Spring Boot) y la base de datos (AWS), además teniendo en cuenta el tiempo, nivel de dificultad del servicio y muchos otros aspectos analizados con mayor detalle en la sección de Anexos, se decidió usar Firebase, pero esto no significa que no se haya dedicado un tiempo de investigación y aprendizaje a estos. Estos conceptos y principales características se explicarán a continuación.

4.7.1. Spring Boot

Spring Boot es una de las tecnologías dentro del mundo de Spring de las que más se está hablando últimamente. ¿Qué es y cómo funciona Spring Boot?. Para entender el concepto primero se debe reflexionar sobre cómo se construyen aplicaciones con Spring *Framework*. Este es un *framework* para el desarrollo *back-end*, que facilita la creación de aplicaciones de todo tipo en Java, entre otros [31]. SpringFramework es una herramienta que nace con la intención de simplificar y facilitar la construcción de aplicaciones JEE(apps java), etc [32]. Según el sitio oficial, se puede definir Spring Boot de la siguiente manera: *“Es una solución para crear aplicaciones basadas en Spring de una manera rápida, autónoma y con características deseables para producción”*. Al momento, resulta de mucho interés contar con herramientas que apoyen al desarrollo rápido de aplicaciones. Y esta es la motivación principal para presentar esta primera entrada sobre el tema [33].
Spring Boot es un *framework* desarrollado, publicado y mantenido por la organización Pivotal, buscando solucionar varios problemas de su predecesor Spring Core *Framework*. Como la palabra lo indica, “Boot”, en este contexto, es ARRANCAR/INICIAR- Spring, originalmente Spring Core *Framework* tenía un proceso muy tedioso y difícil de depurar para su configuración antes de poder iniciar una aplicación, incluso un “Hola Mundo”. Actualmente Spring Boot hace

toda la configuración inicial para que solamente se tenga que presionar Run o Play (dependiendo del IDE) para tener una aplicación Spring en ejecución. Una de las grandes ventajas de Spring Boot aparte de su "mágica" configuración es que las aplicaciones son consideradas "STAND ALONE" o "SER ÚNICO" en su traducción, ya que el *framework* tiene un servidor Embebido [34].

Entre las principales características de Spring Boot se encuentran [33]:

- Configuración sugerida para iniciar rápidamente con un proyecto (Starters).
- Configura automáticamente Spring, cuando sea posible.
- Características listas para producción: métricas, seguridad, verificación del estatus, externalización de configuración, etc.
- No genera código y no requiere configuración XML.
- Se puede integrar los proyectos de React y Spring-Boot en el editor de texto Visual Studio Code [35].
- Proporciona modularidad.
- Escalabilidad de la aplicación sin necesidad de modificar las clases.
- Evita la dependencia entre clases.

En la figura 3 se observa el esquema para el desarrollo de una aplicación con Spring-Boot.

Figura 13. Desarrollo de una aplicación.

4.7.2. AWS

AWS (Amazon Web Services), la "informática en la nube" consiste en la entrega bajo demanda de recursos de TI a través de Internet. Antes de que existiera la informática en la nube, se tenía que crear centros de datos basados en suposiciones de cuáles serían los picos máximos. Si el diseño no satisfacía los picos máximos reales, los clientes sufrían las consecuencias. Si se planificaba en exceso y se superaba la máxima necesidad, se acababa pagando por recursos que en realidad no se necesitaban. Gracias a la informática en la nube, en lugar de tener que diseñar y crear propios centros de datos, se accede a un centro de datos y a todos sus recursos a través de Internet, lo que permite escalar en forma ascendente o descendente en función de las necesidades reales, sin tener que planificar con la peor situación posible en mente. Con AWS, se puede acceder a servidores, bases de datos, almacenamiento y componentes de aplicaciones de nivel superior en cuestión de segundos. Se puede considerar los recursos como temporales y disponibles y, de este modo, se libera de la falta de flexibilidad y las limitaciones de una infraestructura de Tl fija y finita. Al aprovechar la potencia de la nube de AWS, el enfoque de

administración de cambios, pruebas, fiabilidad y planificación de la capacidad es más ágil y eficiente. Con el uso de la nube de AWS, se pueden reducir riesgos, escalar de forma automática la capacidad de cómputo para satisfacer las necesidades, garantizar una cobertura de confianza incluso ante un desastre natural y proteger los datos. La informática en la nube puede ayudar a reducir los riesgos gracias a su agilidad, que le permite aprender y adaptarse a los cambios con rapidez.

Esta agilidad reduce el costo de los cambios. La nube permite todos esos enfoques de administración de riesgos. Gracias a la informática en la nube, las compañías pueden responder de manera rápida y elástica a las condiciones cambiantes del mercado. Esto facilita la escalabilidad, la agilidad y la innovación. Además, con servicios pre desarrollados que se pueden montar con rapidez como componentes básicos, es posible automatizar la entrega de software y crear medidas de seguridad y conformidad. En la informática en la nube, el término escalabilidad hace referencia a la capacidad de cambiar el tamaño de los recursos según sea necesario. Con el uso de AWS, los clientes pueden aumentar, reducir y adaptar su consumo de servicios para satisfacer requisitos estacionales, lanzar nuevos servicios o productos, o simplemente adaptarse a nuevas orientaciones estratégicas. AWS cuenta con una cantidad de servicios y de características incluidas en ellos que supera la de cualquier otro proveedor de la nube, ofreciendo desde tecnologías de infraestructura como cómputo, almacenamiento y bases de datos hasta tecnologías emergentes como aprendizaje automático e inteligencia artificial y análisis e internet de las cosas. Esto hace que llevar las aplicaciones existentes a la nube sea más rápido, fácil y rentable y permite crear casi cualquier cosa que se pueda imaginar. AWS también tiene la funcionalidad más completa dentro de esos servicios [36]. Por ejemplo, AWS ofrece la más amplia variedad de bases de datos que están diseñadas especialmente para diferentes tipos de aplicaciones, de modo que se puede elegir la herramienta adecuada para el trabajo a fin de obtener el mejor costo y rendimiento [37].

Hay 3 Maneras de usar AWS [36]:

1. Consola de Administración de AWS: Interfaz gráfica fácil de usar compatible con la mayoría de Amazon Web Services.

2. Interfaz de línea de comandos CLI: Acceso a los servicios mediante un comando discreto.

3. Kit de desarrollo de software SDK: Incorporación de la conectividad y funcionalidad de la gama más amplia de servicios en la nube de AWS a su código.

En las Figuras 14 y 15 se muestra el resumen de cómo usar AWS.

Figura 14. Formas de usar AWS.

Figura 15. Conclusiones de usar AWS.

Después de este punto es necesario aclarar que luego del estudio de Spring-boot y AWS, por cuestiones de tiempo limitado y complejidad que implicaba el uso de estos para el desarrollo del proyecto, se procedió a migrar a la plataforma Firebase. Esta será explicada con más detalle en la próxima **sección 4.8** y ofrece una mayor ventaja a la hora de tener los dos servicios de *back-end* y base de datos en una sola plataforma. Estas ventajas de Firebase respecto a Spring-boot y AWS será explicado más al detalle en la sección de los Anexos. Debido al uso de Firebase, este también requirió de un tiempo de estudio e investigación para aplicarlo al desarrollo del proyecto.

4.8. Firebase como *back-end* y Base de Datos

Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Está disponible para distintas plataformas (iOS, Android y web), con lo que es más rápido trabajar en el desarrollo. Es un servicio capaz de proveer un *back-end* en la nube con una fuente de datos en tiempo real y bibliotecas para acceder a ella desde aplicaciones web, iOS y Android.

Firebase es un claro ejemplo de las posibilidades de desarrollo en la nube. A partir de un servicio web ofrece la posibilidad de programar aplicaciones con datos que se sincronizan en tiempo real a través de múltiples dispositivos, evitando muchas de las tareas engorrosas en las que se tendría que dedicar tiempo al programar. Es básicamente una base de datos remota, alojada en la nube y capaz de ser accedida desde navegadores y *apps* para dispositivos, que tiene como principal característica el responder en tiempo real a los cambios realizados en los datos. En la práctica, mediante Firebase se puede escribir datos en una base de datos, y que estos datos se comuniquen a todos los clientes conectados a esa misma fuente de datos. Su función esencial es hacer más sencilla la creación de tanto aplicaciones webs como móviles y su desarrollo, procurando que el trabajo sea más rápido, pero sin renunciar a la calidad requerida. Sus herramientas son variadas y de fácil uso, considerando que su agrupación simplifica las tareas de gestión a una misma plataforma. Es especialmente interesante para que los desarrolladores no necesiten dedicarle tanto tiempo al backend, tanto en cuestiones de desarrollo como de mantenimiento. Firebase dispone de diferentes funcionalidades, que se pueden dividir básicamente en 3 grupos: Desarrollo (Develop), Crecimiento (Grow) y Monetización (Earn), a los que hay que sumar la Analítica (Analytics). Para el proyecto solo se hizo énfasis en la parte del Desarrollo (Sección 5.3 de la Metodología). Desarrollo o *Develop* en Firebase, como su nombre indica, incluye los servicios necesarios para el desarrollo de un proyecto de aplicación móvil o web. Estos contribuyen a que el proceso sea más rápido, puesto que se dejan determinadas actividades a mano de Firebase, mientras que otras permiten optimizar diversos aspectos para conseguir la calidad deseada. Para la base de datos usada en el proyecto que proporcionaba Firebase, se escogió Realtime database [38]. Para entender mejor cuales son los servicios asociados al Desarrollo en Firebase de una aplicación como la de este proyecto en específico se tiene la figura 16.

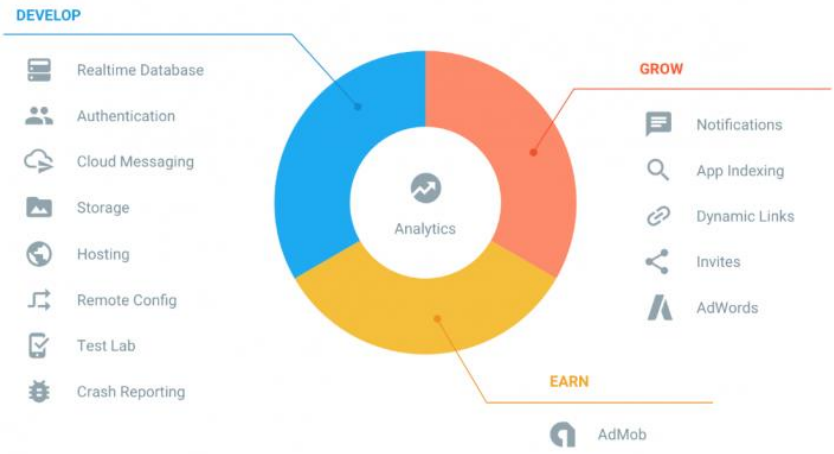

Figura 16. Servicios de Firebase para Develop.

4.8.1. Realtime database

Una de las herramientas más destacadas y esenciales de Firebase son las bases de datos en tiempo real. Estas se alojan en la nube, son NoSQL (Los datos almacenados no requieren estructuras fijas como tablas) y almacenan los datos como JSON (es un formato ligero de intercambio de datos). Permiten alojar y disponer de los datos e información de la aplicación en tiempo real, manteniéndolos actualizados, aunque el usuario no realice ninguna acción [38]. Firebase envía automáticamente eventos a las aplicaciones cuando los datos cambian, almacenando los datos nuevos en el disco. Aunque no hubiera conexión por parte de un usuario, sus datos estarían disponibles para el resto y los cambios realizados se sincronizarían una vez restablecida la conexión. El uso que se le dió a esta base de datos para el proyecto fue específicamente para que el cliente de La Empresa del sector de seguros viera el cambio de estado de sus solicitudes en tiempo real a medida que se almacenan datos para su visualización.

4.8.2. Autenticación de usuarios

La identificación de los usuarios de una app es necesaria en la mayoría de los casos si estos quieren acceder a todas sus características. En el caso de la aplicación desarrollada en el presente proyecto, la identificación se requiere para el registro o inicio de sesión de los usuarios que llenen los formularios tal como se muestra en la figura 5. Para ello, Firebase ofrece un sistema de autenticación que permite tanto el registro propiamente dicho (mediante email y contraseña), como el acceso utilizando perfiles de otras plataformas externas (por ejemplo, de Facebook, Google o Twitter), una alternativa muy cómoda para usuarios reacios a completar el proceso, lo cual se observa en la figura 17.

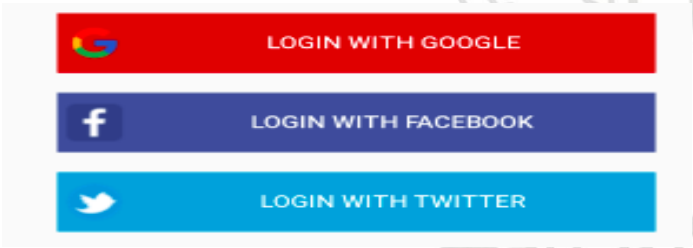

Figura 17. Alternativas de Inicio de Sesión.

Así, este tipo de tareas se ven simplificadas, considerando también que desde aquí se gestionan los accesos y se consigue una mayor seguridad y la Empresa del sector de seguros de los datos. Se debe mencionar que Firebase puede guardar en la nube los datos de inicio de sesión con total seguridad, evitando que una persona tenga que identificarse cada vez que abra la aplicación [38].

4.8.3. Almacenamiento en la nube

Firebase cuenta con un sistema de almacenamiento, donde los desarrolladores pueden guardar los ficheros de sus aplicaciones (y vinculándolos con referencias a un árbol de ficheros para mejorar el rendimiento de la app) y sincronizarlos. Al igual que la mayoría de herramientas de Firebase, es personalizable mediante determinadas reglas [38]. Este almacenamiento es de gran ayuda para tratar archivos de los usuarios (por ejemplo, fotografías que hayan subido), que se

pueden servir de forma más rápida y fácil. También hace la descarga de referencias a ficheros más segura.

4.8.4. Hosting

Firebase también ofrece un servidor para alojar las apps de manera rápida y sencilla, esto es, un hosting estático y seguro. Proporciona certificados de seguridad SSL y HTTP2 de forma automática y gratuita para cada dominio, reafirmando la seguridad en la navegación. Funciona situándolas en el CDN (es básicamente un conjunto de servidores ubicados en diferentes puntos de una red que contienen copias locales de ciertos contenidos como vídeos, imágenes, música, documentos, webs, etc.) de Firebase, una red que recibe los archivos subidos y permite entregar el contenido. Y dado este servicio ofrecido por Firebase se hará el despliegue de la aplicación web. En la figura 18 se observa un claro ejemplo de este servicio, donde se muestra el dominio donde se desplegó la aplicación final.

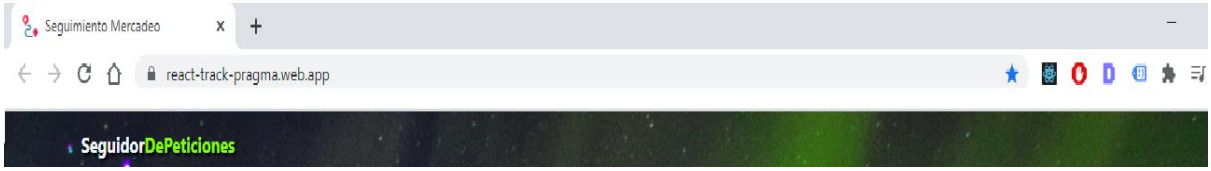

Figura 18. Alternativas de Inicio de Sesión.

4.8.5. Firebase aplicado al proyecto

Con Firebase se hizo uso de su servicio de almacenamiento de datos en tiempo real para almacenar la información de los usuarios al registrarse y de las tareas y listas creadas desde el tablero de Pragma.
Ahora respecto a Spring-Boot que es la parte Back-end, es completamente independiente de Firebase. Firebase es básicamente un servicio proporcionado por Google para que el desarrollador no necesite realizar una codificación del lado del servidor. SpringBoot es un marco que se usa generalmente para el desarrollo de la aplicación del lado del servidor. Al usar firebase, debe pagar a Google por el almacenamiento de datos y el uso de API. Si se compara el uso de la aplicación Firebase y Spring-Boot en términos de costo de la misma cantidad de almacenamiento de datos, la cantidad que debe pagar por firebase es mucho más. Aunque esta cantidad está justificada, ya que no necesita escribir ni una sola línea de código para el extremo del servidor y todas las conexiones del servidor, firebase mantiene el almacenamiento de datos y la seguridad. Aunque si se está comenzando, se recomienda que se aprenda y use Spring-Boot en lugar de Firebase. Pero si la aplicación exige una base de datos en tiempo real, es mejor optar por Firebase para encargarse de esa función. En el caso del proyecto se eligió Firebase porque ofrece un dominio gratuito para desplegar la aplicación y alojar la información del proyecto sin ningún problema, dejando claro que por ser versión gratuita tiene algunas limitaciones, que la versión paga no tiene, pero en este caso no fue necesario ya que la aplicación no es tan robusta y compleja para hacer uso de esta.

4.9. Integración de ReactJs con Firebase

Lo que se logra en el proyecto con las definiciones ya abordadas anteriormente fue hacer la integración de ambos, siendo ReactJS el pilar fundamental de la aplicación web, donde se invoca desde el código a las funciones que trae Firebase para el envío y solicitud de información.

5. Metodología

A continuación, se describen las etapas y actividades que se realizaron para el desarrollo del proyecto en el transcurso de la práctica académica.

5.1 Etapa 1: Evaluar las tareas requeridas para la creación de contenidos y acercamiento a los *frameworks* de desarrollo *front-end* y *back-end*

Esta primera etapa fue de las más complejas del proyecto porque consistió en el estudio y acercamiento con herramientas tecnológicas, de las cuales no se tenía conocimiento alguno, además de revisar y adentrarse en la lógica del negocio del cliente. Primero, se hizo una presentación formal del proyecto, con qué tecnologías se planteaba trabajar inicialmente y un primer esquema de la estructura que iba a tener el proyecto en general. Adicionalmente, se recibieron explicaciones para entender la lógica del negocio con la que se trabajaba en el equipo. Dicho esquema mencionado anteriormente se encuentra en el siguiente diagrama de bloques de la Figura 19:

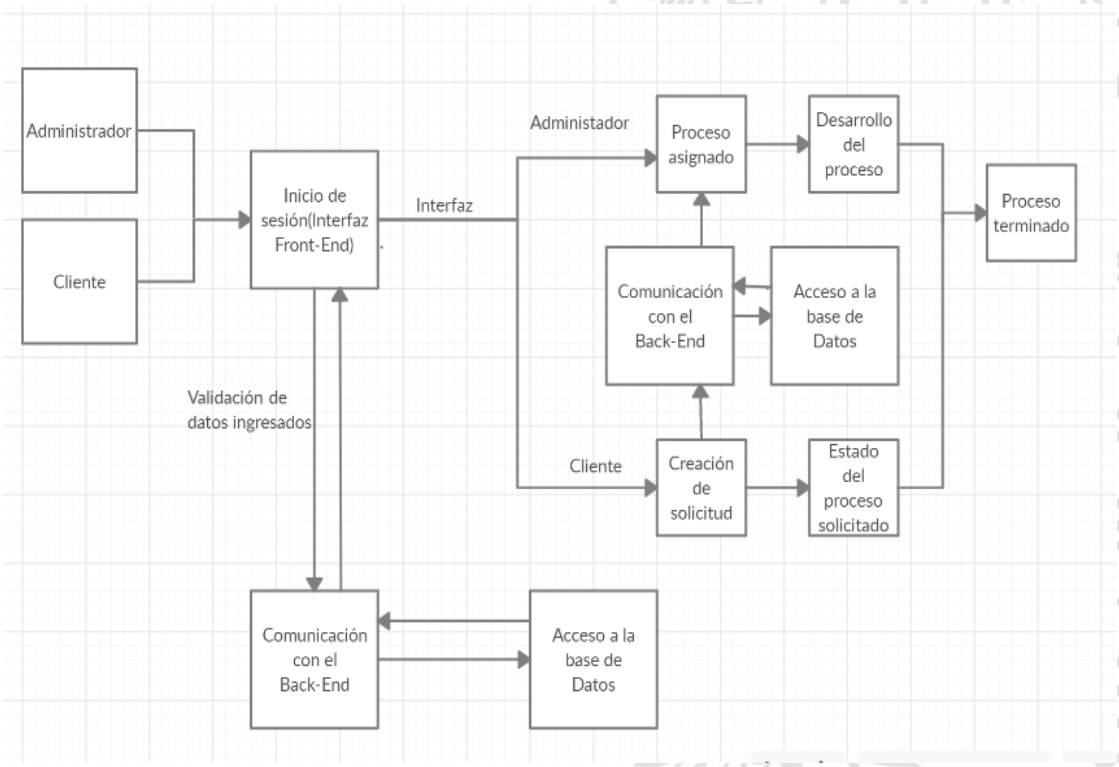

Figura 19. Sistema propuesto inicialmente.

Este esquema estuvo en constante evolución, construcción y adaptación, según las necesidades que se lograron evidenciar en el desarrollo del proyecto cambiando así el sistema propuesto

inicialmente y siendo así implementado el de la figura 1 del Marco Teórico. Posteriormente fue necesario realizar una capacitación por medio de la documentación, cursos y vídeos tutoriales referentes a ReactJS y Spring-Boot para así tener una mejor claridad en los conceptos y principales características de estos. Las principales características y conceptos referentes a ReactJS y Spring Boot utilizados en el proyecto se explicaron en **la sección 4.6 y 4.7** del Marco Teórico. A pesar de no utilizar Spring Boot no quiere decir que no se le dedicó un tiempo de estudio, debido a esto se anexa la información recopilada de este tras su estudio e investigación.

5.1.1. Ejemplos prácticos

Luego de obtener los conocimientos básicos, se procedió a realizar ejemplos para afianzar los temas aprendidos durante el tiempo de entrenamiento; además, los ejemplos fueron basados en la estructura que iba tener el proyecto, es decir, contenía aspectos como elaboración de componentes, creación de formularios, envío y consultas de información, funcionalidad, entre otros, y todos ellos existentes en la realización del proyecto. A continuación, se puede observar las figuras con algunos de los ejemplos que se hicieron para el aprendizaje de ReactJS:

Figura 20. Ejemplo creación de componentes.

En esta figura se puede ver un ejemplo de creación de componentes en React, donde la imagen y cada línea de texto que se observa como “Pruebahola msn” es un componente diferente, el cual puede ser reutilizado y simplemente se le cambia el texto.

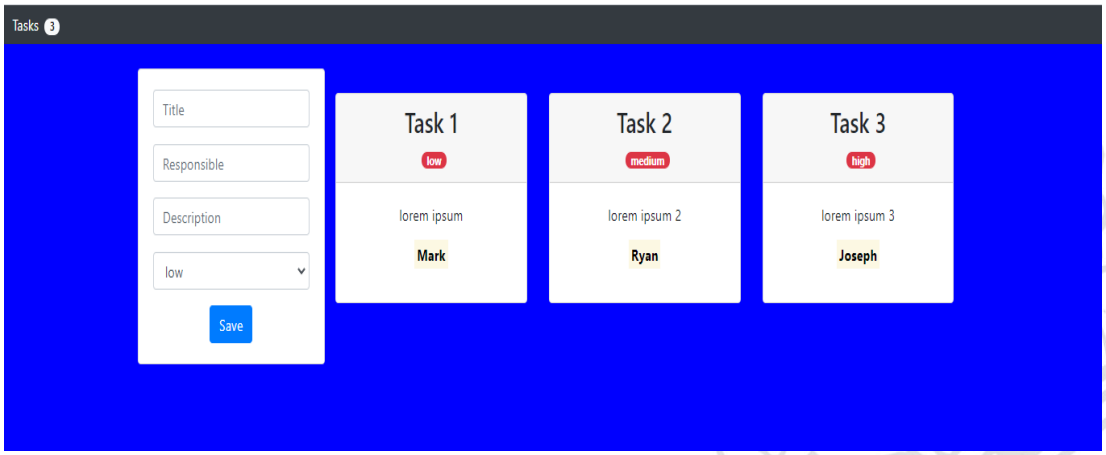

Figura 21. Ejemplo creación de formulario y Tareas.

En esta figura se muestra un formulario con 4 campos diferentes para completar la información y un botón para guardar esta. Al guardar estos datos se genera una card o tarjeta como por ejemplo la de "Task 1" que se compone de una serie de textos. Cada vez que se guarda información nueva en el formulario, se crea una tarjeta nueva con esos datos. Todos estos componentes visuales, tanto el formulario como las tarjetas fueron implementados como componentes de React y con Boostrap para la parte visual. Esto fue parte fundamental para el desarrollo de la aplicación del proyecto.

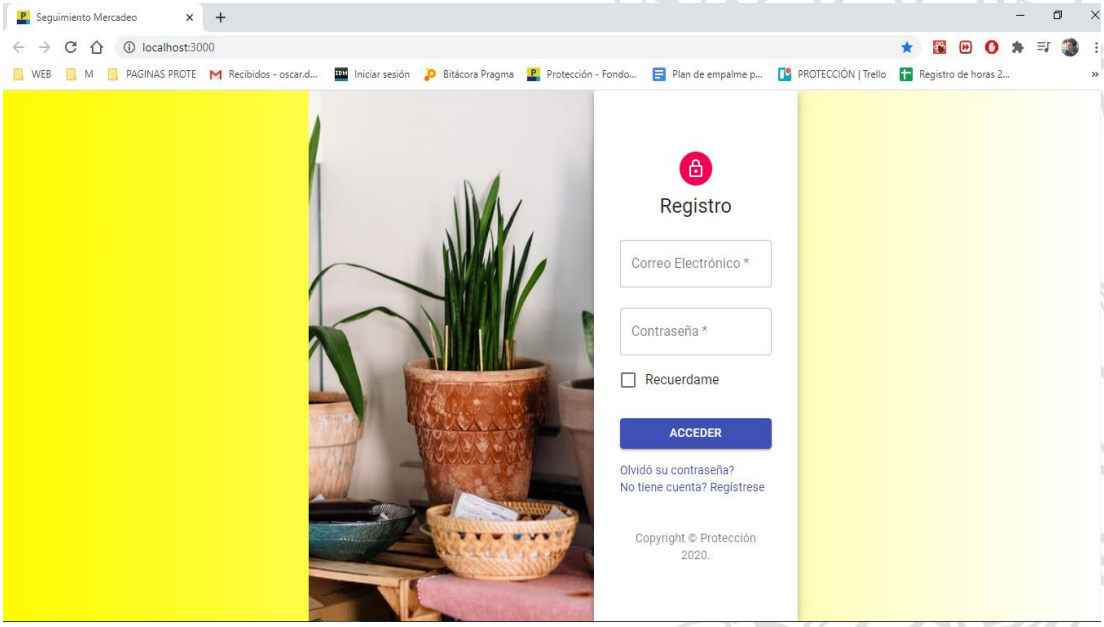

Figura 22. Ejemplo creación de formulario Inicio de Sesión.

En esta figura se ve un formulario con las mismas características que el de la figura 21, como los campos que se deben llenar, un botón para el acceso y entre otros, pero con mejor diseño, estilos visuales y más orientado a lo que se iba a realizar en el proyecto como se verá en la sección de Resultados.

5.1.2 Reestructuración de la propuesta inicial de solución

Debido a que el sistema de la figura 19 estuvo en constante construcción y por cambio de Spring-boot a Firebase, el esquema que se usó finalmente para la implementación fue el de la figura 1 de la **sección 4.1.5** del Marco Teórico.

Las herramientas usadas en el desarrollo de los bloques de la figura 1 fueron ReactJS para la parte *front-end* y Firebase para la parte *back-end* y base de datos. Los lenguajes utilizados fueron: el lenguaje de etiquetas HTML, el lenguaje de diseño gráfico para estilos visuales CSS y Javascript para las funcionalidades de la aplicación.

5.2 Etapa 2: Acercamiento a AWS y definición de criterios para los diferentes niveles de prioridad de los procesos solicitados.

En la segunda etapa, al igual que el caso de Spring-boot, a pesar de no utilizar AWS como sistema de almacenamiento de base de datos en la nube, se realizó una capacitación por medio de la documentación, cursos y vídeos tutoriales, para así tener una mejor claridad en los conceptos y principales características de este, que se explicó en la sección **4.7.2** del Marco Teórico. En esta etapa también se definieron los diferentes niveles de prioridad con las personas encargadas de crear las solicitudes que se reciben del cliente. Estas prioridades fueron definidas de la siguiente manera:

- **Alta:** La solicitud es de carácter urgente y se debe parar cualquier otra solicitud que esté en curso para darle prioridad a esta. Al ser de carácter prioritario, esta solicitud se debe realizar lo más pronto posible y el mismo día que la hayan asignado.

- **Media:** La solicitud puede ser realizada durante el transcurso del mismo día que se asignó.

- **Baja:** Por lo general este tipo de prioridad es referente a una solicitud que fue asignada con días de anterioridad y se da una fecha en específico para realizarse, donde se tiene uno o más días hábiles para llevarla a cabo.

5.2.1. Desarrollo *front-end* de interfaz de registro e inicio de sesión

Ahora se procede a explicar cómo fue el desarrollo *front-end* de la interfaz de inicio de sesión de la aplicación web. Lo primero que se hizo fue trabajar en Visual Studio Code (VSCode) que es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto [35].

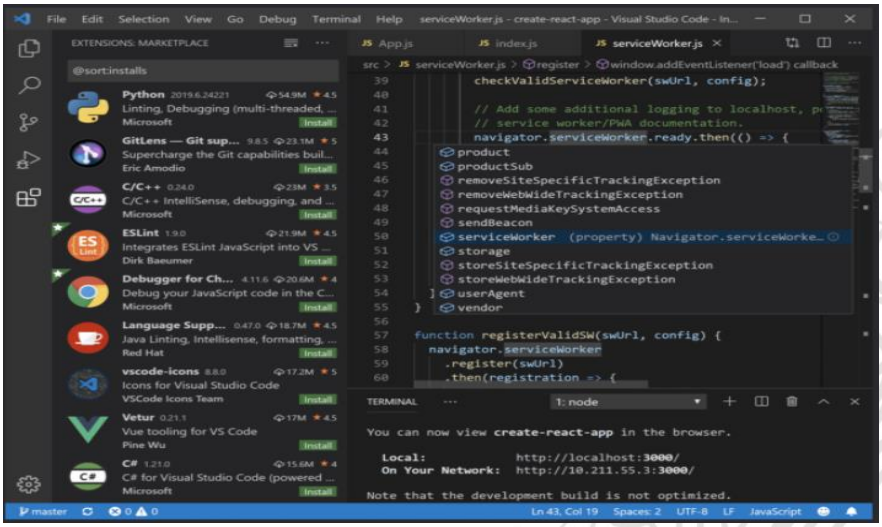

Figura 23. VSCode

En la figura 23 se puede observar la interfaz de desarrollo donde se hace la programación de código. Ya estando en este editor, se creó el proyecto de ReactJS con el siguiente comando:

"npx create-react-app proyecto_grado"

npx viene de *npm*, el cual se puede usar a través de **Nodej.js** que es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono. Para poder usar este, hubo que instalarlo. Ahora bien, en Node.JS el código se encuentra estructurado por módulos, que a medida que se va trabajando se necesita agregar más de estos según sea el caso y es en este punto donde justamente entra **NPM (Node Package Manager)** [39]. NPM es un gestor de paquetes desarrollado en su totalidad bajo el lenguaje JavaScript, a través del cual se puede obtener cualquier biblioteca con tan solo una sencilla línea de código, lo cual permite agregar dependencias de forma simple, distribuir paquetes y administrar eficazmente tanto los módulos como el proyecto a desarrollar en general. Para poder utilizar *npm* había que instalarlo desde la consola del editor de texto VSCode, como se ve a continuación:

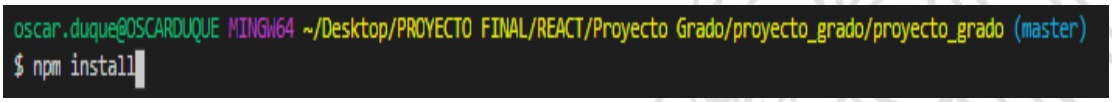

Figura 24. Instalación de npm.

Luego de instalar NPM, Node.js y crear el proyecto se procedió a crear la interfaz visual tanto del Registro como del Inicio de Sesión de los usuarios. Esta implementación se pudo llevar a cabo con la ayuda una biblioteca que contiene ReactJS llamada ***"reactstrap".*** Esta biblioteca contiene componentes de React Bootstrap 4 [40], que favorecen la composición y el control. Es un conjunto de herramientas de código abierto para desarrollar con HTML, CSS y JavaScript. Prototipa rápidamente las ideas, crea una aplicación completa, con amplios componentes prediseñados y potentes complementos basados en jQuery. Incluye plantillas de diseño basadas en HTML y CSS para tipografía, formularios, botones, tablas, navegación, modales, carruseles de imágenes y muchos otros, así como complementos de JavaScript opcionales. Con esta ayuda

se facilita más el diseño de componentes visuales. Para poder usar esta biblioteca es necesario ejecutar el comando "***npm* install --save reactstrap react react-dom"** la cual instala las dependencias necesarias para utilizarse adecuadamente. Como React funciona con una lógica de creación de componentes reutilizables, se creó cada formulario como un componente individual, tanto el formulario de Registro como el de Inicio de Sesión. El formulario de registro consta de 3 campos para que la persona que se vaya a registrar llene estos que son: Nombre Completo, Correo Electrónico y Contraseña. Estos campos se colocan con el fin de obtener la información de cada usuario que se registre y validarlo con la base de datos. Además, consta de una imagen y del botón de "Registrarse" el cual manda esa información a través de Firebase para su verificación. El formulario de inicio de sesión es muy similar al de registro ya que consta de 2 campos para que la persona que se vaya a loguear llene estos que son: Correo Electrónico y Contraseña. Estos campos se colocan con el fin de obtener la información de cada usuario que quiera iniciar sesión y validarlo con la base de datos. Además consta de una imagen, del botón de "Acceder", el cual manda esa información a través de Firebase para su verificación y el botón de "Olvidó su contraseña?" la cual sirve para restablecer la contraseña por si el usuario la olvidó.

Luego ambos formularios fueron insertados en un componente principal para mayor facilidad en su uso. La funcionalidad del cambio entre formularios se debe a una lógica de funcionamiento *"Toogle"* o alternar, con la ayuda de componentes visuales llamados "Nav" que son barras de navegación que proporciona "reactstrap" y aplicando la lógica de funcionalidad con JavaScript. Es necesario aclarar que todo el proyecto de la aplicación web fue hecho basado en una **single-page application (SPA)**, o aplicación de página única, es una aplicación web (o sitio web) que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como si fuera una aplicación de escritorio. En un SPA todos los códigos de HTML, JavaScript, y CSS se cargan una sola vez o los recursos necesarios se cargan dinámicamente cuando lo requiera la página, normalmente como respuesta a las acciones del usuario. La página no tiene que cargarse de nuevo en ningún punto del proceso y tampoco es necesario transferir a otra página. La interacción con las aplicaciones de página única, pueden involucrar comunicaciones dinámicas con el servidor web que está detrás [41]. Los formularios se implementaron con la ayuda de "reactstrap" ya que ofrece elementos visuales que ya están hechos y simplemente se accede a ellos importando los que se necesiten en la cabecera del código como se ve a continuación:

import { Form, FormGroup, Label, Input, Button } from "reactstrap";

Con este código se está teniendo acceso a los elementos ya prediseñados de un formulario, etiquetas, entradas de textos y botones, que son de lo que se componen los formularios de la aplicación. Por ejemplo, para mostrar el diseño de un formulario prediseñado se define el siguiente código:

```
<FormGroup>
  <Label>Correo electrónico</Label>
  <Input
    value={this.state.email}
    onChange={this._handleChange}
```

```
            type="email"
            name="email"
            placeholder="ejemplo@ejemplo.com.co"
          />
        </FormGroup>
```

Esta porción de código corresponde al campo donde se pide ingresar el correo electrónico. Donde la etiqueta <FormGroup> agrupa los elementos del formulario, <Input> es la entrada de texto que se solicita ingresar y sus atributos, value para almacenar el correo que se ingrese en el campo, onChange es una función que detecta el tecleo cuando se ingresa el correo y lo almacena, type para indicar que el tipo del campo es email, name para darle el nombre al campo y placeholder para colocar un texto por defecto que aparezca en el campo del texto a ingresar. Para las validaciones del formulario de inicio de sesión se definió el siguiente código:

```
if ( v_email !== '' && this.state.password !== ''){
  if (element_pragma.join('') === pragma_email || element_prote.join('') === prote_email) {
    firebase.auth().signInWithEmailAndPassword(this.state.email, this.state.password)
    .catch(error => {
      alert("El usuario no se encuentra registrado.");
    });
  }else{
    alert("El usuario no se encuentra registrado.");
  }
 }else{
  alert("Campo(s) vacío(s), intente nuevamente");
 }
```

En esta porción de código se valida mediante un condicional *if* , si los campos que ingresa el usuario no están vacíos, si lo están muestra una ventana de alerta con el mensaje: "Campo(s) vacío(s), intente nuevamente" y si los campos no están vacíos pero no se encuentran en la base de datos entonces muestra una ventana de alerta con el mensaje: "El usuario no se encuentra registrado.". Ahora si el ingreso de la información es correcto entonces se valida con otro *if*, si el correo ingresado es del dominio de “@pragma.com.co” o el de la Empresa del sector de seguros, esto con el fin de saber si el que inicia sesión es un Administrador de Pragma o un Cliente de La Empresa del sector de seguros.

5.3 Etapa 3: Estudio e implementación del desarrollo *back-end* para el aplicativo web del seguimiento de procesos.

En esta etapa se hizo una investigación de cómo implementar la parte *back-end* y de almacenamiento de la información enviada a través del formulario. Debido al estudio previo que se hizo tanto de Spring-Boot como de AWS, se decidió cambiar a Firebase, el cual también se le dedicó un tiempo de estudio, ya que este es especialmente interesante para que los desarrolladores no necesiten dedicarle tanto tiempo al *back-end*, tanto en cuestiones de desarrollo como de mantenimiento, procurando que el trabajo sea más rápido, pero sin renunciar a la calidad requerida. Firebase ofrece un servicio capaz de proveer de un *Backend* en la nube con una fuente de datos en tiempo real y bibliotecas para acceder a ella desde aplicaciones web, iOS y Android. Es un claro ejemplo de las posibilidades de desarrollo en la nube. A partir de un servicio web ofrecen la posibilidad de programar aplicaciones con datos que se sincronizan en tiempo real a través de múltiples dispositivos, evitando muchas de las tareas engorrosas en las que se tendría que dedicar tiempo al programar. Para entender mejor, es básicamente una base de datos remota, alojada en la nube y capaz de ser accedida desde navegadores y apps para dispositivos, que tiene como principal característica responder en tiempo real a los cambios realizados en los datos. En la práctica, mediante Firebase se puede escribir datos en una base de datos y que estos datos se comuniquen a todos los clientes conectados a esa misma fuente de datos [42]. Dicho esto, lo primero que se hizo fue hacer el registro online en la página oficial de Firebase para acceder a sus beneficios, luego se procedió a crear el proyecto como se ve a continuación:

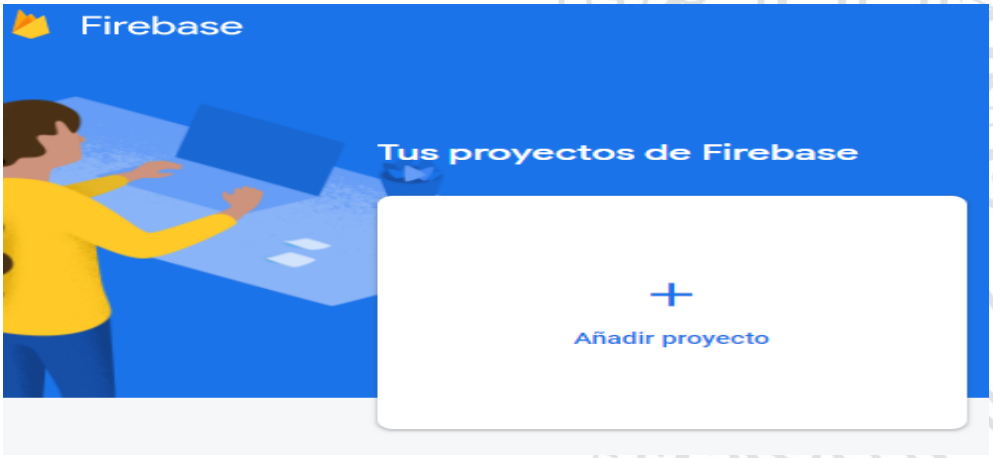

Figura 25. Creación inicial del proyecto.

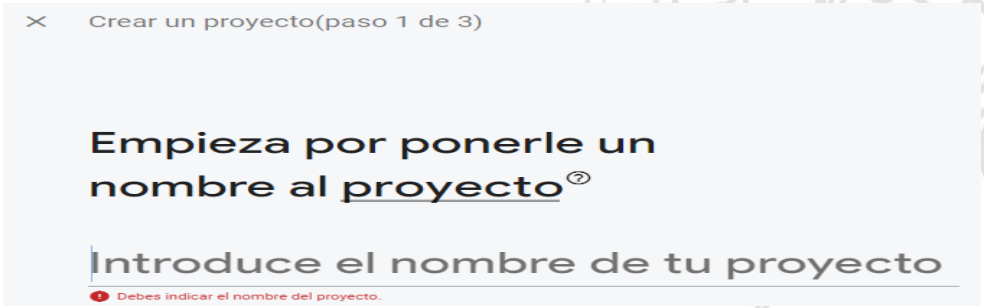

Figura 26. Paso 1, nombre del proyecto.

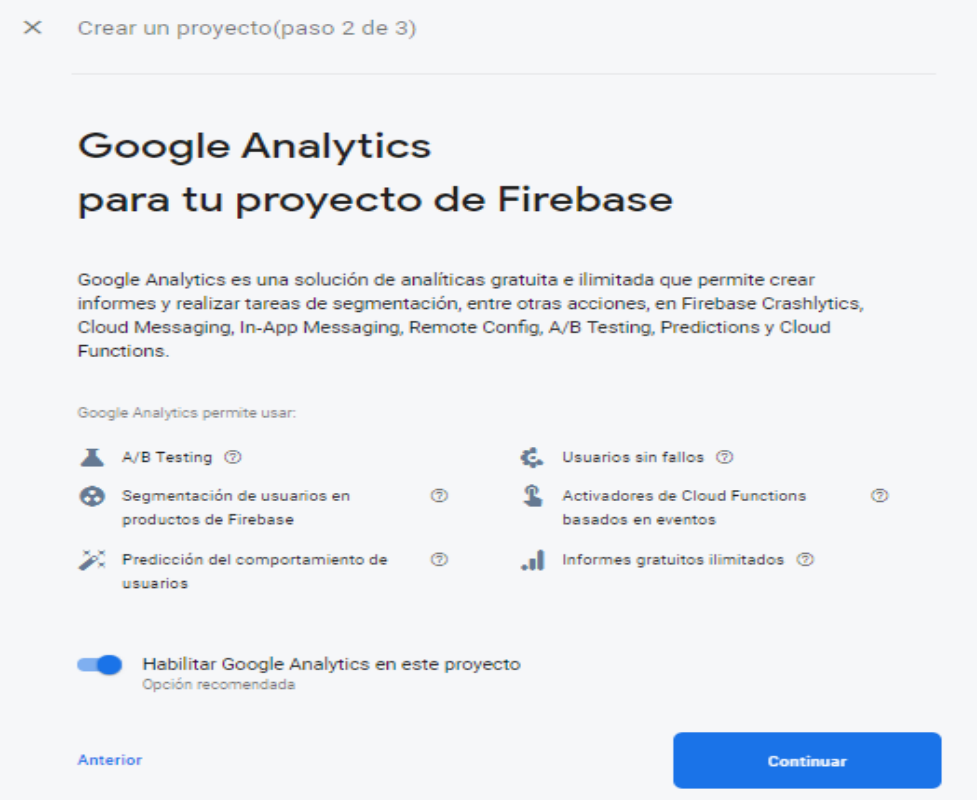

Figura 27. Paso 2, incluir servicio de Google Analytics.

En la figura 27 se observa que se encuentra la opción de habilitar o no Google Analytics. Si se habilita continúa un paso más, pero en este caso no se hizo entonces ya se procede a la creación del proyecto.

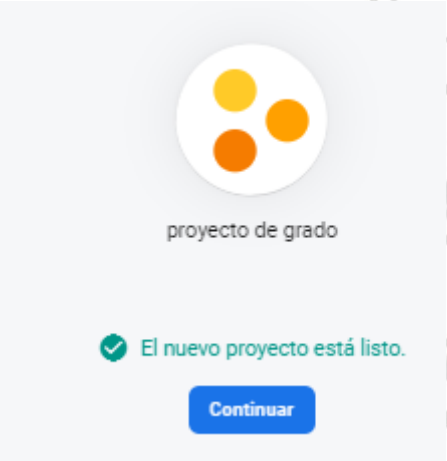

Figura 28. Proyecto listo.

Después de crear el proyecto, la página se redirecciona a la página principal de este, donde sale un panel en la parte lateral izquierda, en este aparecen todos los servicios que ofrece Firebase para el desarrollo de las aplicaciones que se puede observar en la imagen a continuación:

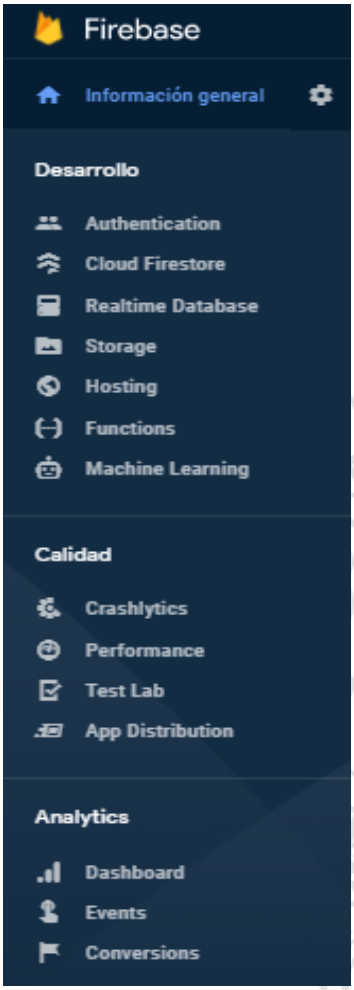

Figura 29. Panel de Servicios.

En este panel se encuentran todos los servicios que ofrece Firebase para el desarrollo de aplicaciones. En el caso del presente proyecto, de las opciones observadas en el Panel de Servicio mostrado en la figura 29, solo se hizo énfasis en las opciones de "**Authentication**", "**Realtime *Database***" y "**Hosting**", debido a que según el estudio previo que se hizo de Firebase y sus servicios, estos eran los que más se adecuaban a las necesidades del proyecto. Para este primer desarrollo que era el envío de información de autenticación para el Inicio de Sesión y el Registro, se usó el servicio de *Authentication*. Pero antes hay que explicar que para poder configurar Firebase en el Proyecto de React, hacer la debida integración entre ambos y poder hacer uso de sus servicios, se debía agregar una porción de código que entrega Firebase e implementarlo en React. Esto se hace estando en el proyecto de React, donde se creó un archivo **indexf.js** para guardar la configuración de Firebase y poder hacer la correcta conexión con este. El archivo debe ser similar a este código ejemplo, pero en el caso del proyecto fue reemplazado con los datos asociados a este:

```
var firebaseConfig = {
  apiKey: "AIzaSyDOCAbC123dEf456GhI789jKl01-MnO",
  authDomain: "myapp-project-123.firebaseapp.com",
  databaseURL: "https://myapp-project-123.firebaseio.com",
```

```
  projectId: "myapp-project-123",
  messagingSenderId: "65211879809",
  appId: "1:65211879909:web:3ae38ef1cdcb2e01fe5f0c",
  measurementId: "G-8GSGZQ44ST"
};
```

De esta configuración los campos más relevantes que se pueden explicar son authDomain: que es el nombre del dominio que tendrá la página de la aplicación que se cree en Firebase, databaseURL: que es el dominio donde está alojada la base de datos de la app y projectId: el identificador que se le asigna al proyecto. El resto de campos son valores aasignados por Firebase. Luego se configuró en la consola de VsCode el proyecto de Firebase ejecutando el comando de "firebase init" el cual vincula el directorio local de la app con Firebase, genera un archivo firebase.json (un archivo obligatorio para Firebase Hosting) y solicita que se especifique un directorio raíz público que contenga tus archivos estáticos públicos (HTML, CSS, JS, etcétera). En la configuración predeterminada, Firebase busca un directorio que se llame "public" como se ve en la figura 30:

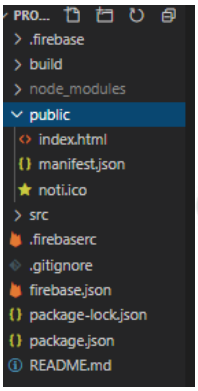

Figura 30. Archivos del proyecto en React.

Si se deseaba, se podía especificar un nombre de directorio público. Para ello, tendría que modificar directamente el archivo firebase.json. Después de realizar toda la configuración e integración con Firebase, se hizo uso de la documentación para usar sus funciones y poder vincular con la base de datos como se ve en la Figura 31.

```
firebase
.auth()
.createUserWithEmailAndPassword(this.state.email, this.state.password)
.catch(error => {
  this.setState({
    message: error.message
  });
});
```

Figura 31. Creación de usuario en Firebase.

Lo que hace .createUserWithEmailAndPassword es crear un formulario que permite a los usuarios nuevos registrarse en la app mediante su dirección de correo electrónico y una contraseña. Cuando un usuario completa el formulario, este válida la dirección de correo electrónico y la contraseña que proporcionó el usuario para después pasarlos al método

createUserWithEmailAndPassword. Esto se hizo para el formulario de Registro. Para el formulario de Inicio de Sesión se utilizó otra función de Firebase como se ve en la figura 32.

```
firebase
.auth()
.signInWithEmailAndPassword(this.state.email, this.state.password)
.catch(error => {
  console.log(error);
});
```

Figura 32. Función para validar el Inicio de Sesión.

Esta función crea un formulario que permite a los usuarios existentes acceder con su dirección de correo electrónico y una contraseña. Cuando un usuario complete el formulario, llama al método signInWithEmailAndPassword. La validación de ambos formularios se hizo con lógica condicional, en la que se validaba el Registro o Inicio de Sesión mediante el dominio de los correos electrónicos, donde si la persona que intenta ingresar no es de Pragma S.A o La Empresa del sector de seguros, no lo deja ingresar y le muestra un mensaje en pantalla, el cual en código JavaScript se llama una función "alert()" que es una ventana emergente como se observa en la figura 33.

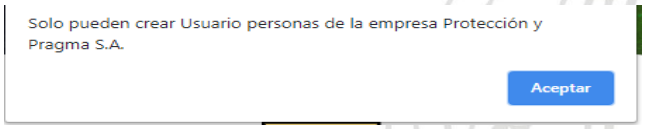

Figura 33. Alerta de Registro fallido.

Además, para la opción *"si olvidó la contraseña"*, se usó una función que ofrece Firebase, que envía un correo electrónico de restablecimiento de contraseña. Para enviar un correo electrónico de restablecimiento de contraseña a un usuario, se puede usar el método sendPasswordResetEmail como se ve en la figura 34.

```
var auth = firebase.auth();
var emailAddress = this.state.email;
auth.sendPasswordResetEmail(emailAddress).then(function() {
  alert("Se envió un correo a su cuenta para restablecer contraseña");
}).catch(function(error) {
  alert("Escriba un correo para restablecer contraseña");
});
```

Figura 34. Restablecer contraseña.

En la Figura 35 se observa un ejemplo de cómo se ve desde la consola de Firebase a 2 usuarios Registrados usando el formulario de registro de la app como el de la figura 6 del Marco Teórico:

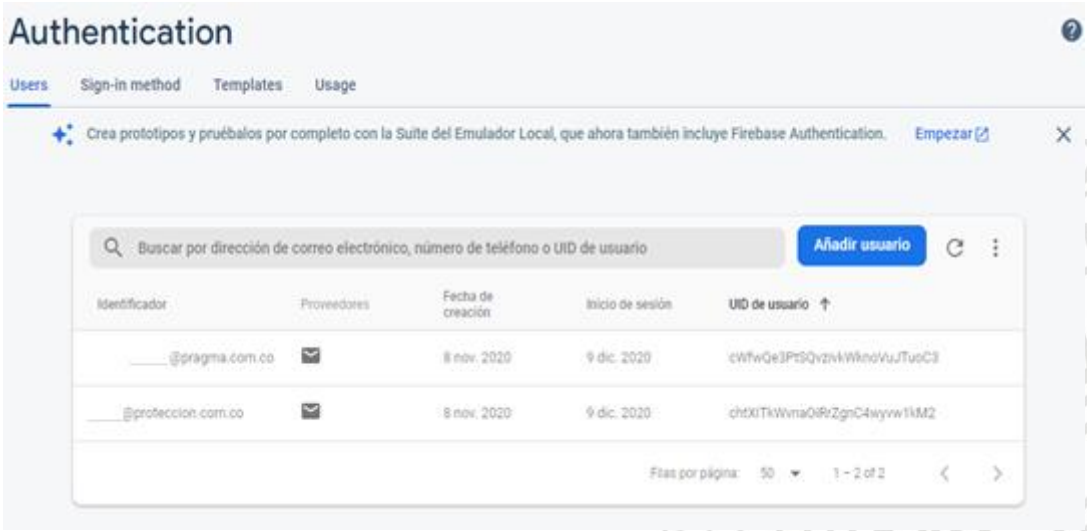

Figura 35. Usuarios en consola de Firebase.

Y por último se observa que en el componente "componentDidMount" se coloca la funcionalidad de que busque en la base de datos la información si hay usuario o no. Si no hay usuario registrado en la base de datos, este saca un mensaje en la consola, mostrando que los usuarios en la base de datos son nulos, pero si ya hay usuarios registrados se crea una lista de usuarios donde se le asigna a cada uno una clave(key) y el nombre de usuario (UserName) para poder acceder a esta información posteriormente.

```
componentDidMount() {
  const myList_users = firebase.database().ref("users/");
  myList_users.on("value", snapshot => {
    const myList_usersFromDatabase = snapshot.val();
    if (myList_usersFromDatabase === null) {
      console.log("Users at our firebase is null");
    } else {
      const list_users = Object.keys(snapshot.val()).map(key => {
        return {
          key: key,
          UserName: myList_usersFromDatabase[key].UserName
        };
      });
      this.setState({
        dataUsers: list_users
      });
    }
  });
}
```

Figura 36. Búsqueda y almacenamiento en la Base de Datos.

En este punto es donde entraba el servicio de Firebase de "**Realtime *Database***" ya que ahí es donde se recibe y almacena la información en una colección o rama de datos como se ve a continuación:

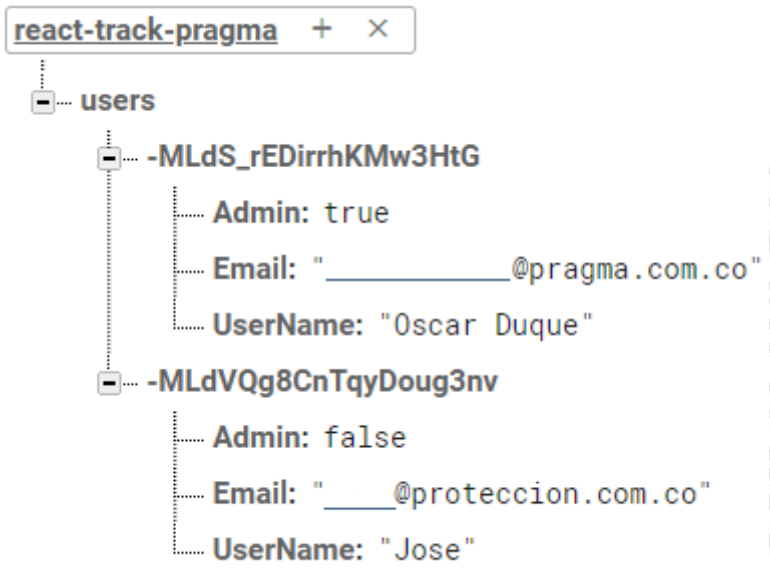

Figura 37. Colección de Usuarios.

En la figura 37 se puede observar que cada usuario creado con su respectiva Key y propiedades, como el nombre de usuario, el correo y una propiedad llamada "**Admin**". Esta propiedad o atributo es de valor booleano ya que, si es un usuario ingresado con dominio de Pragma, el valor de Admin es igual a "*True*" y si es de La Empresa del sector de seguros es igual a "*False*", separando así los permisos y visualización de ambos usuarios. Esta es una base de datos alojada en la nube con formato JSON, el cual es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, se considera un formato independiente del lenguaje [43]. Los datos se sincronizan a tiempo real con la aplicación. La comunicación es bidireccional, es decir que cuando se modifica datos en la base de datos, estos cambios se van reflejados directamente en la web y viceversa [44].

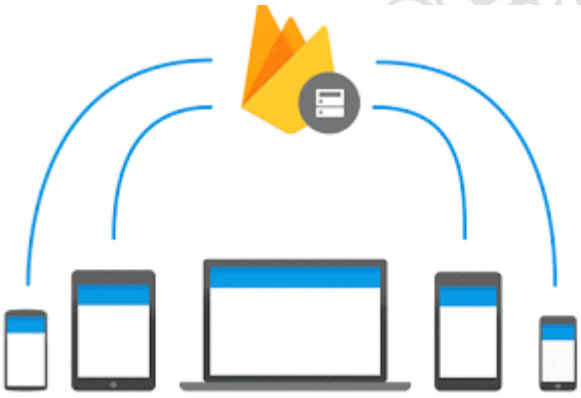

Figura 38. Sincronización en todos los dispositivos en tiempo real.

Uno de los requerimientos para la la Empresa del sector de seguros de acceso a los datos en Firebase son las reglas del *Database*, donde empezando es mejor desactivar la autenticación, para poder trabajar con la base de datos sin necesidad de obligar a loguearse para trabajar con ella. Estas reglas de seguridad de Firebase Realtime Database determinan quién tiene acceso de

lectura y escritura a la base de datos, cómo se estructuran los datos y qué índices existen. Estas reglas se alojan en los servidores de Firebase y se aplican automáticamente en todo momento. Cada solicitud de lectura y escritura sólo se completará si lo permiten las reglas que se definan. Según la configuración predeterminada, estas no permiten que nadie acceda a la base de datos, a fin de protegerla de cualquier abuso hasta que se tenga tiempo para personalizar las reglas o configurar la autenticación [45]. Luego, con el transcurso del proyecto y siendo el Inicio de Sesión indispensable en los formularios del proyecto (ver sección 5.2.1) se tenía que cambiar las reglas de seguridad dentro de la "sección de reglas" del *database*. Firebase como servicio *back-end* debe implementar reglas de seguridad para poder definir qué se puede hacer con los datos almacenados en la base de datos, quién puede leer o escribir, entre otras cosas. Obviamente es algo necesario, porque si se administran usuarios alojados en la base de datos lo más seguro es que estos escriban en algún momento datos privados, que no se quiera que otros usuarios puedan leer. Si se tienen roles en la aplicación y por ejemplo solo los administradores pueden crear determinados tipos de ítem, se tendrá que proteger la aplicación para que no todos los usuarios puedan escribir en esos ítems. En general, Firebase ofrece un sistema sencillo para escribir las reglas de seguridad de una aplicación, por medio de un archivo de texto que tiene notación JSON. Las reglas de seguridad de Firebase son bastante fáciles de escribir, sin embargo, el lenguaje a veces es un poco abstracto. Además, dado que se deben escribir reglas que den cabida a todas las posibles situaciones, la dificultad puede ser elevada para dar con la fórmula correcta, sobre todo si no se está acostumbrado a trabajar con este sistema. Las reglas iniciales con las que se empieza a crear el proyecto por defecto y que Firebase no recomiendo son las siguientes:

```
{
  "rules": {
      ".read": true,
      ".write": true
  }
}
```

Figura 39. Reglas iniciales en Firebase.

Como se puede ver en la figura 39, se utiliza una notación JSON para definir las reglas. En el caso mostrado en dicha figura, se están otorgando permisos de lectura y escritura para cualquier entidad que esté en la base de datos Firebase. La regla para el permiso de lectura comienza con ".read", mientras que la regla para el permiso de escritura se escribe en el ítem ".write". El valor true indica un permiso bajo cualquier situación, lógicamente esto está bien para que la aplicación permita empezar a trabajar, pero no será lo adecuado cuando esta aplicación ya esté en un dominio de internet y sea de acceso público ya que no será muy seguro que todos los usuarios tengan los mismos permisos de acceso a los datos. Debido a esto se hizo un cambio en las reglas de la manera como se muestra en la Figura 40.

Figura 40. Reglas para el proyecto en Firebase.

Lo mostrado en la Figura 40 quiere decir que, cuando el usuario autenticado sea distinto de *null* (se tiene cualquier usuario correctamente logueado), se otorgará permiso de lectura y de escritura según el usuario que se loguee.

5.4 Etapa 4: Desarrollo de la interfaz principal de la aplicación y la comunicación con la base de datos mediante el *back-end*.

En esta etapa se realizaron las interfaces principales de la aplicación, que consta de dos tableros personalizados, uno para los usuarios de Pragma con permisos de Administrador y otro para los usuarios de La Empresa del sector de seguros. Además, se implementó con las funcionalidades de Firebase la comunicación con la base de datos para el alojamiento de todos los datos ingresados por el Administrador y el acceso a la información por parte del cliente. El primer tablero que se desarrolló fue el de los usuarios de Pragma, el cual consta de un componente principal que contiene, primero un componente de Navegación, el cual tiene el nombre de "Seguidor de Peticiones" en el costado izquierdo, el cual fue creado haciendo uso de la sintaxis propia de React, JSX explicada en la sección 4.6.4 del Marco Teórico, como se observa en la figura:

```
render() {
  return (
    <div>
      <Navbar expand="md">
        <NavbarBrand >Seguidor<span >DePeticiones</span></NavbarBrand>
```

Figura 41. Reglas para el proyecto en Firebase.

Una opción de "eliminar el usuario" y una opción desplegable al costado derecho, el cual indica al usuario si desea salir, lo cual constituye la opción de "cerrar la sesión". Es necesario dejar en claro que todos los componentes también fueron diseñados en su parte visual con la ayuda de "reactsrap" explicado anteriormente.

Figura 42. Tablero para Administradores de Pragma.

Para eliminar el usuario se usó la función de Firebase mostrada en la la Figura 43.

```
var user = firebase.auth().currentUser;

user.delete().then(function() {
  alert("Usuario eliminado.");
}).catch(function(error) {
  alert("No se pudo eliminar usuario");
});

};
```

Figura 43. Función para eliminar usuarios.

En esta figura se puede observar que en la variable “user” se almacena el usuario que está autenticado, para luego utilizar esa variable y llamar a la función “delete()” donde se ve que si se elimina el usuario, muestra una ventana emergente con un mensaje de “Usuario eliminado” y si hay algún error, muestra un mensaje con “No se pudo eliminar usuario”. Para cerrar la sesión de un usuario se usa otra función que es:

```
firebase.auth().signOut();
```

Las funcionalidades de eliminar usuario y cerrar sesión aplica para ambos tableros, tanto el de usuarios de la empresa Pragma, como el de usuarios de la Empresa del sector de seguros. El segundo componente, es el del tablero donde se agregan las listas o estados por donde van a pasar las peticiones requeridas por el cliente. Luego de crear una lista aparece la opción de crear las tareas requeridas, donde aparecen las opciones de colocar el nombre de esta y su prioridad, que puede ser Alta, Media o Baja, estas ya fueron explicadas anteriormente (ver sección 5.2). Al crear las listas y tareas, todas estas son almacenadas en la base de datos de Firebase como se observa en la figura 44.

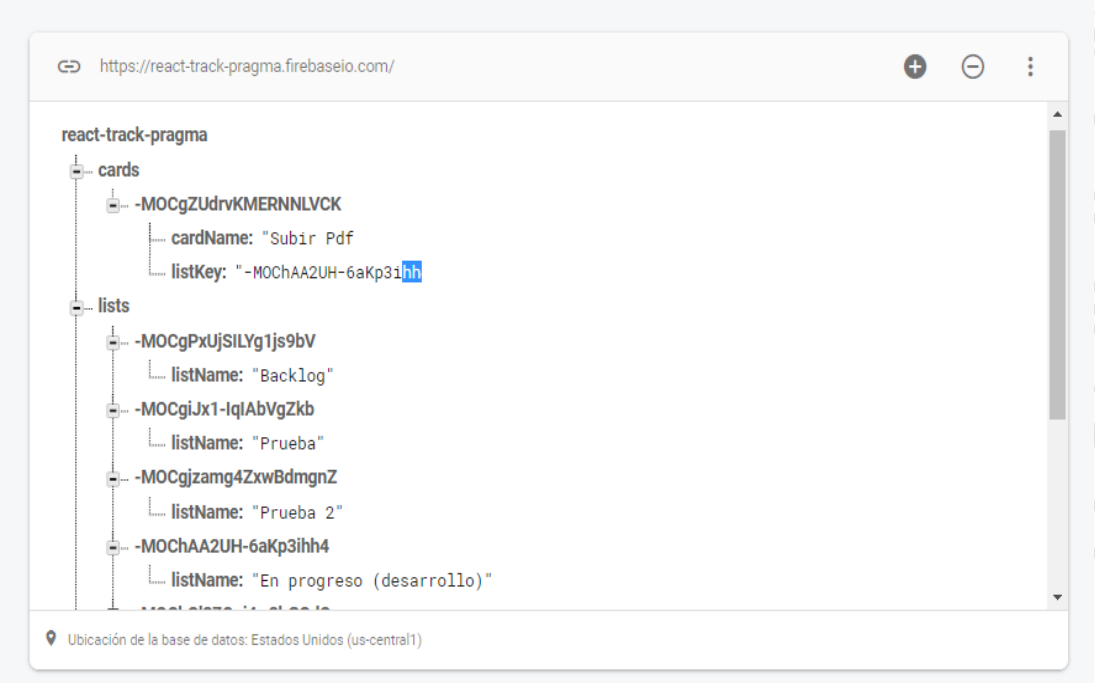

Figura 44. Función para eliminar usuarios.

En esta figura se visualiza la base de datos en forma de árbol, donde se almacenan los datos de las listas y tarjetas que se crean en la aplicación. En la figura 45 se presenta la lógica utilizada para almacenar esa información en la base de datos.

```
if (this.state.listName === "") {
  alert("La lista no puede estar vacía");
} else {
  const newListKey = firebase
    .database()
    .ref("lists/")
    .push().key;

  firebase
    .database()
    .ref("lists/")
    .update({
      [newListKey]: {
        listName: this.state.listName
      }
    });
  this.setState({
    listName: "" //es para que el campo donde uno digita se borré después de crear la lista
  });
}
```

Figura 45. Función para guardar las listas.

En dicha figura se puede observar que mediante un condicional *if*, si el nombre de la lista que se vaya a almacenar está vacío, sale un mensaje de alerta, y si no, se crea una variable que usa una función de Firebase para crear una colección de datos con el nombre de *"lists"*, donde se van a almacenar todos los datos de las listas creadas, como se muestra en la figura 44. En esa figura se puede observar que se almacena el nombre de la lista y una clave o *key* de referencia de cada lista almacenada. Esta misma lógica aplica para guardar las tarjetas de las tareas que se crean, tal como se ve en la Figura 46.

```
if (this.state.cardName === "") {
  alert("La tarjeta no puede estar vacía");
} else if(this.state.value !== ""){
  const newCardKey = firebase
    .database()
    .ref("cards/")
    .push().key;

  firebase
    .database()
    .ref("cards/")
    .update({
      [newCardKey]: {
        listKey: key.key,
        cardName: title.title
      }
    });

  this.setState({
    cardName: ""
  });
}else{
  alert("Escoja un nivel de Prioridad");
}
};
```

Figura 46. Función para guardar las tarjetas de las tareas.

Se logra observar que aplica la misma lógica anterior que mediante un condicional *if*, si el nombre de la tarjeta que se vaya a almacenar está vacío, sale un mensaje de alerta, y si no, se crea una variable que usa una función de Firebase para crear una colección de datos con el nombre de *"cards"*, donde se van a almacenar todos los datos de las tarjetas creadas, a diferencia de la lógica anterior, esta cuenta con dos excepciones. La primera excepción es que, al almacenar la información, se guarda la *key*, el nombre de la tarjeta y la *key* de la lista en la que se encuentra la tarjeta. Esto para dar una relación de dónde se encuentra la tarjeta a medida que cambia de estado o de lista. Y la otra excepción es que dentro de la condición "else" se encuentra la prioridad, si no se escoge alguna al momento de crear la tarjeta de la solicitud, sale una alerta de que se escoja una prioridad. Estas tarjetas también cuentan con un *TICKET* que está asociado con la *key* de la tarjeta. Este *TICKET* es el que se le debe entregar al cliente para que pueda realizar la consulta de su petición. Además, las tarjetas cuentan con la opción de 3 botones los cuales sirven para desplazarse entre listas y editar el texto del nombre que tenga la tarjeta. Y para eliminar una lista de la base de datos, se hizo con esta función:

```
firebase
  .database()
  .ref(`lists/${key}`)
  .remove();
```

Figura 47. Función para eliminar listas.

Y para eliminar una tarjeta, se realiza con la misma función, el único cambio es cambiar *"lists"* por *"cards*". A medida que la tarjeta se va moviendo de lista en la parte del Administrador, al mismo tiempo, en tiempo real se está modificando lo que ve el cliente en su tablero, esto se debe al servicio que presta Firebase. Se puede ver mejor de manera gráfica como en la figura 48.

Figura 48. Paso de solicitud en tiempo real.

Al momento que la tarjeta se mueve también manda una notificación al cliente que se muestra en el navegador, si el usuario da el permiso para que se muestre. Esto se hizo por si el usuario no está en la página se le pueda notificar si su petición se movió de estado. Esta notificación se hizo con esta función que se ve a continuación:

```
Notification.requestPermission()
  .then(() => new Notification(
    "Su solicitud está en: "+ list.listName,
    { "icon": "/noti.ico"
    }
  ));
```

Figura 49. Función para mandar notificaciones.

Este servicio de notificación ya lo ofrece React y solo es necesario llamar el método de "Notification" donde se logra observar que accede a pedir permiso al usuario y cuando éste acepte, le muestre la notificación con un icono agregado como se ve en la figura 50.

Figura 50. Interfaz con aviso de notificación.

De lo que se observa de la figura 50, se tienen 5 listas que se despliegan cuando el cliente ingresa el TICKET de su solicitud, estas listas son "Backlog, Asignada, En progreso (desarrollo), Validación con cliente y DONE". Estas listas fueron definidas al entender el flujo de trabajo por el que pasaban las solicitudes desde que el cliente de La Empresa del sector de seguros las realiza, hasta que el Administrador de Pragma las recibe y luego procede a verificar que se ejecute para avisar al Cliente cuando ya culmine. Ahora hablando del tablero que ve el cliente, este fue desarrollado de forma más simple e intuitiva para el usuario, donde al igual que lo explicado anteriormente en el tablero del Administrador, consta de dos componentes, el primero es el de Navegación que se explicó anteriormente (sección 5.4) y el segundo el que contiene la opción de consulta de la solicitud. La consulta es posible ya que se hace con el TICKET que es entregado por los Administradores de Pragma. El cliente procede a ingresar el TICKET en el campo de ingreso de texto del tablero personalizado y hace la consulta, si el campo donde se ingresa está vacío o el TICKET se ingresó mal, se muestra un mensaje de alerta para que lo intente nuevamente. Cuando el cliente logra consultar su TICKET, el campo de consulta se bloquea hasta que se quiera hacer una nueva consulta. Luego de consultar su solicitud, se despliega una tabla donde se puede observar sólo los estados principales por donde pasa su solicitud y en qué estado se encuentra actualmente. Para la consulta de dónde se encuentra la tarjeta se realiza de igual forma que en la del tablero de Pragma y es haciendo uso de las funciones de Firebase y el acceso a la base de datos. Cuando se quiere hacer otra consulta,

la tabla que se muestra con el proceso desaparece para realizar una nueva consulta como se ve en la figura 51.

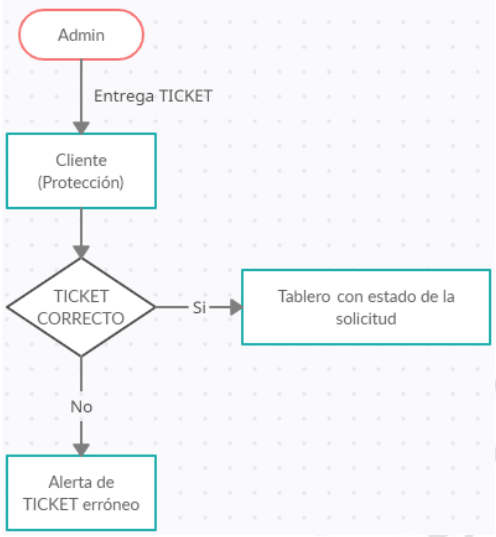

Figura 51. Diagrama de flujo.

Debido a que Firebase provee el servicio *back-end*, entonces en este caso no había que preocuparse con su comunicación con la base de datos sino simplemente hacer uso de las funciones de este para acceder a los datos almacenados en la base de datos.

5.5 Etapa 5: Evaluación final del rendimiento de la aplicación y realización del informe final.

La fase final del proyecto consistió en realizar todas las posibles evaluaciones del rendimiento que tuvo el proyecto, para luego plasmarlas en este documento en la sección de resultados y análisis. Los resultados más relevantes de esta etapa se pueden observar en la sección de resultados y análisis, con sus respectivas imágenes.

6. Resultados y análisis

6.1. Resultados

Debido a que se obtuvieron múltiples resultados, estos se van a presentar en subsecciones.

6.1.1. Formularios

6.1.1.1. Registro

Se puede evidenciar el resultado final que se obtuvo al implementar las interfaces y funcionalidad a través de ReactJS. En la creación de los formularios, el primero que se realizó fue el de Registrarse, donde el resultado puede observarse a continuación:

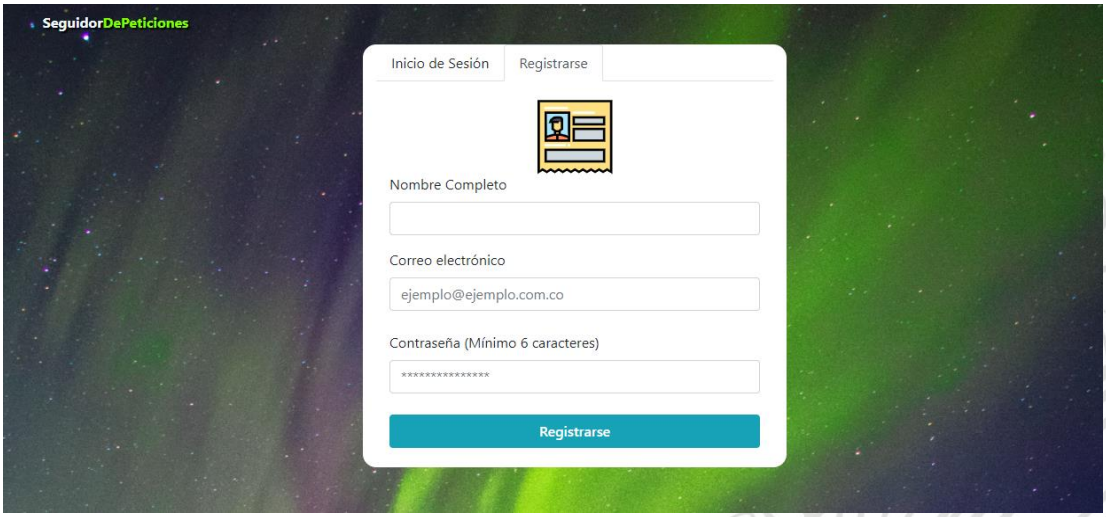

Figura 52. Formulario de Registro.

En este formulario se hace el Registro de los usuarios nuevos donde se piden 3 datos de ingreso: Nombre Completo, Correo electrónico y Contraseña. Se hicieron una serie de validaciones para el formulario en general, los campos del correo electrónico y la contraseña. Para el formulario en general se hace una validación, que al dar click en el botón de Registrarse y estar todos los campos o algún campo en específico vacío, entonces muestra el mensaje que se ve a continuación:

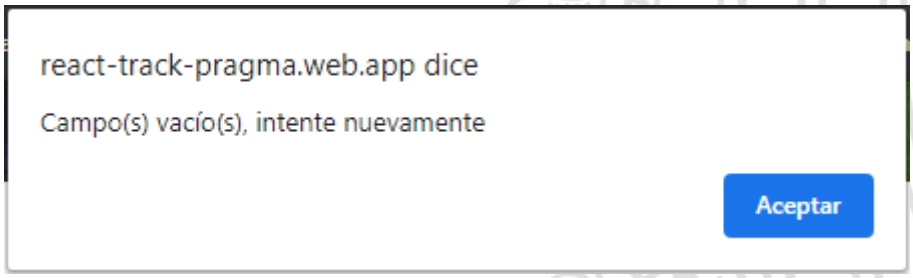

Figura 53. Mensaje de alerta, campos vacíos.

Como se ve en la figura, el mensaje que se muestra en esta y en todos los posteriores que dice "react-track-pragma.web.app" es el nombre del dominio de la página, que se le dió al momento de crearla en el servicio de Hosting que ofrece Firebase. Donde la intención del nombre es, react porque en esa tecnología fue hecho, track de seguimiento en inglés, pragma la empresa donde se hizo el desarrollo y .web.app agregado por Firebase.Para la validación del correo electrónico, se muestran dos alertas, la primera si el correo que se ingresa para registrarse no es del dominio de Pragma S.A o de la Empresa del sector de seguros, entonces se muestra el siguiente mensaje:

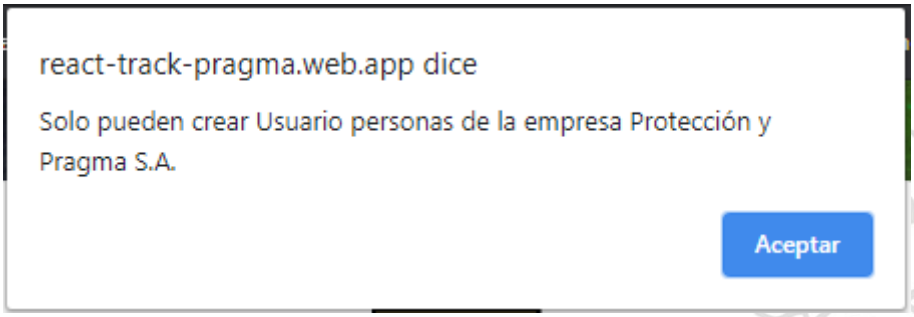

Figura 54. Mensaje de alerta, correo sin autorización.

La segunda validación, muestra un mensaje de alerta cuando el usuario (con el dominio de correo electrónico permitido) que intenta registrarse, ya lo había hecho con la misma cuenta de correo electrónico y muestra el siguiente mensaje:

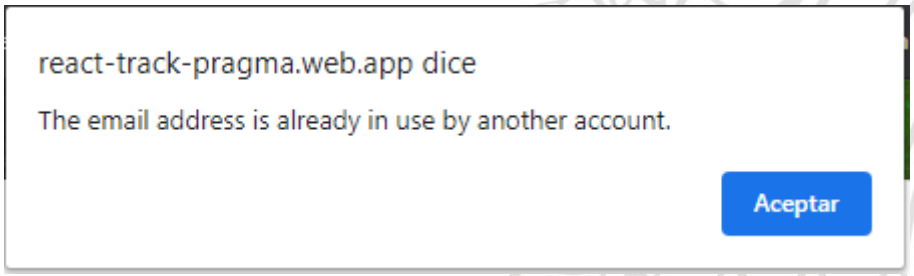

Figura 55. Mensaje de alerta, correo ya registrado.

Este mensaje lo muestra en inglés por defecto ya que es generado automáticamente por los servicios prestados por Firebase, no siendo posible modificar el idioma porque no brinda la opción de español. Por último, se hace una validación en el correo electrónico donde la contraseña debe ser al menos de 6 caracteres, sino se muestra el siguiente mensaje de alerta:

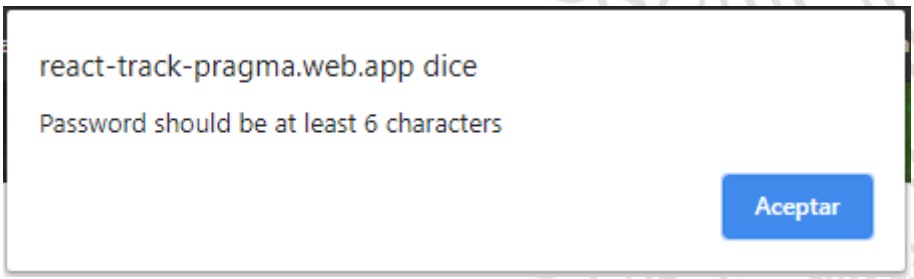

Figura 56. Mensaje de alerta, contraseña muy corta.

6.1.1.2. Inicio de Sesión

Ahora para el formulario de Inicio de Sesión que se ve a continuación, al igual que en el de Registro, también se hicieron validaciones:

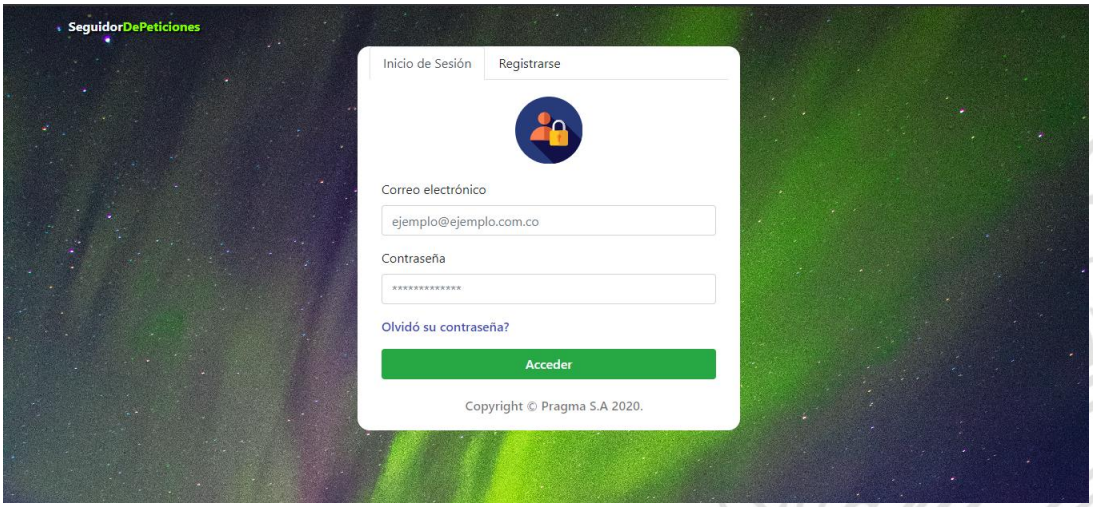

Figura 57. Formulario de Inicio de Sesión.

En este formulario se realiza el Inicio de Sesión de los usuarios donde se piden 2 datos de ingreso: Correo electrónico y Contraseña. Se hicieron una serie de validaciones para el formulario en general, los campos del correo electrónico y la contraseña. La primera validación se hizo de forma idéntica a la ***Figura 53. Mensaje de alerta, campos vacíos,*** donde si se dejan todos los campos vacíos o alguno en específico, entonces sale un mensaje de alerta indicando esto. La segunda validación es respecto a los usuarios que no se encuentran registrados en la base de datos entonces se muestra un mensaje de alerta como se observa a continuación:

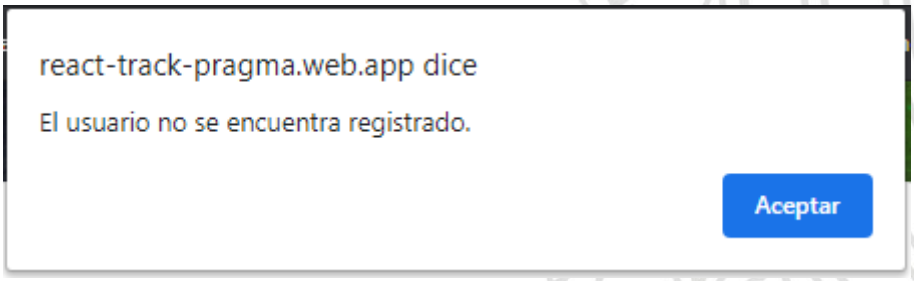

Figura 58. Mensaje de alerta, fallo de Inicio de Sesión.

Del resultado final de la creación de formularios, se obtuvo también la funcionalidad de cambiar entre estos, seleccionando la pestaña al que se quiere acceder.

6.1.2. Tablero Personalizado Administrador (Pragma S.A)

Después de tener un Registro o Inicio de sesión exitoso, se muestra en pantalla un tablero personalizado. En este caso si es un Administrador de la empresa Pragma, se ve la interfaz de la Figura 40.

Figura 59. Tablero para Administradores de Pragma.

En la figura 59 se puede observar una tarjeta con un campo que dice “Añadir una lista”, donde se crean las listas o estados (ver sección 5.4 de la Metodología), que conforman el paso de las peticiones que se piden. Se hizo una validación en la que, si se le da clic en “Guardar” y el campo donde se añade el nombre de la lista está vacío, se muestra la siguiente imagen:

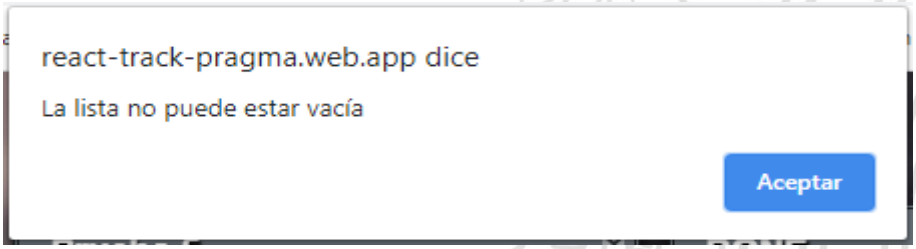

Figura 60. Alerta de lista vacía.

Se tienen que crear 5 listas en específico por defecto, las cuales se muestran al cliente de La Empresa del sector de seguros. Estas listas que se deben crear son “Backlog”, “En progreso (desarrollo)”, “Validación con cliente” y “DONE”. Cabe mencionar que es posible crear listas con diferentes nombres que no serán vistos por el cliente y serán de uso interno para los Administradores. Se le da un nombre a la lista que se quiere crear, por ejemplo “Backlog” y se da clic en el botón Guardar, donde finalmente se muestra la lista creada como se ve a continuación:

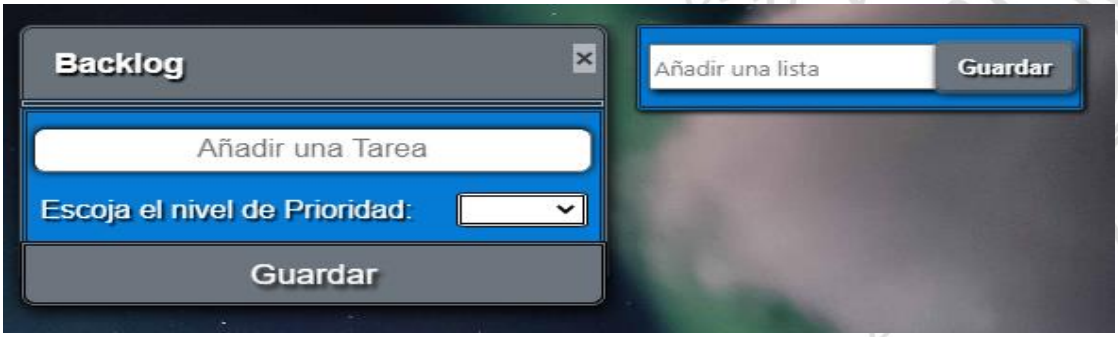

Figura 61. Lista creada.

Como se muestra en la figura 61, al crear la lista se genera una tarjeta interna donde hay dos campos: el primero es un campo donde se ingresa el nombre de la petición que solicita el Cliente y el segundo el nivel de Prioridad de este, el cual puede ser Baja, Media o Alta. En la figura 62 se muestra lo dicho anteriormente.

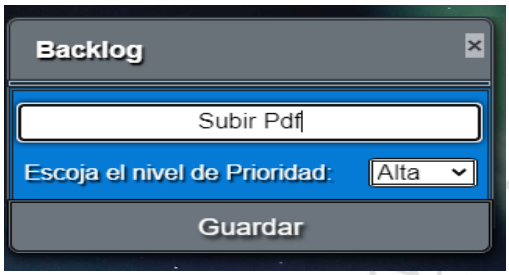

Figura 62. Creación de Requerimiento.

En esta tarjeta de la figura 62 se realiza igualmente las respectivas validaciones, la primera es respecto al nombre de la tarea o petición, el cual no puede estar vacío; esto muestra un mensaje de alerta como se ve a continuación:

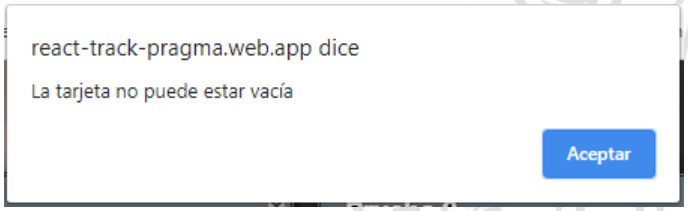

Figura 63. Alerta de tarjeta vacía.

y la segunda validación es respecto a la prioridad, ya que se debe escoger alguna sino se muestra el siguiente mensaje de alerta:

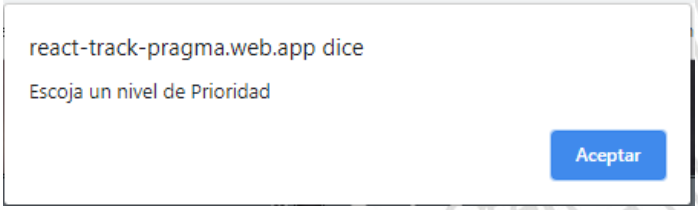

Figura 64. Alerta de que no se escogió Prioridad.

Al ingresar los datos solicitados se da clic en guardar para generar la tarjeta con los datos como se muestra en la figura 65.

Figura 65. Tarjeta de la petición.

La tarjeta se crea con los campos ingresados como el nombre, la prioridad y un parámetro adicional que es generado y asociado con la clave o key que se almacena en Firebase y es el Ticket que se le debe entregar al cliente para consultar su petición. Adicional a esto se tienen 3 botones, los cuales dos de ellos son para desplazarse entre listas. El primer botón "Izquierda" se bloquea en la primera lista ya que no se tiene más listas a la izquierda de esta y se habilita al pasar de lista. El segundo botón "Derecha" se bloquea si no existen listas a la derecha de la actual. Por último, el tercer botón "Editar" el cual muestra una ventana modal, esta ventana consiste en un cuadro que aparece sobre la página, bloqueando todas las funciones para concentrar el foco en una acción particular, donde se le pide al usuario que realice una acción específica. En este caso las acciones son dos botones y un campo para editar el texto, el primer botón es "Editar", que sirve para cambiar el nombre de la petición solicitada o "Cancelar" para no modificar el nombre que en este caso, como se ve en la figura 46 está en el campo para editar el nombre que es "Subir Pdf". En la siguiente imagen se puede apreciar mejor lo explicado:

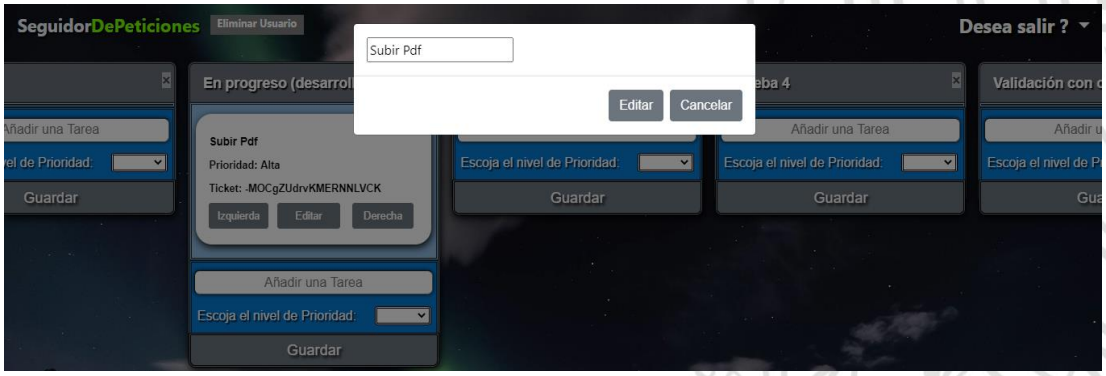

Figura 66. Ventana modal "Editar".

Ahora en la siguiente imagen se puede observar una vista más amplia de una tarjeta con una petición ya creada en la lista llamada "Backlog". Además, se pueden apreciar otras listas de prueba creadas entre el "Backlog" y "En progreso (desarrollo)" que son las listas principales:

Figura 67. Vista general.

Se puede observar en la ***figura 68 y 69*** el desplazamiento de la petición entre listas (cómo se almacenan esas listas se puede ver en la sección 5.4, figura 45 de la Metodología) con ayuda de los botones de "Derecha" e "Izquierda":

Figura 68. Desplazamiento entre listas.

Figura 69. Desplazamiento entre listas.

Hay que recordar que a medida que la petición se va desplazando entre listas, el Cliente de la Empresa del sector de seguros estará viendo en tiempo real en su tablero personalizado ese cambio de estado o de lista de sus solicitudes. Ya que tanto el tablero de Pragma como el de La Empresa del sector de seguros están estrechamente relacionados, por lo que cada vez que se mueva una petición o solicitud en el tablero de Pragma, se verá reflejado en tiempo real en el tablero de La Empresa del sector de seguros. Ahora pasando a otro componente visual que se

tiene en la parte superior de la interfaz, se puede observar que al dar clic en “Desea salir?” se despliega la opción de “Cerrar Sesión” la cual redirige a la pantalla inicial de los formularios. Esta opción se puede ver a continuación:

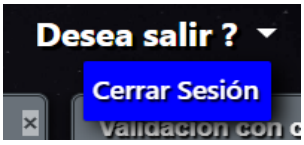

Figura 70. Opción Cerrar Sesión.

Y por último se tiene la opción de “Eliminar Usuario”, al dar clic en esta, se elimina el usuario de la base de datos y ya no se podrá ingresar nuevamente. Esta opción se ve en la parte superior de la interfaz, se puede observar en la siguiente imagen:

Figura 71. Opción Eliminar Usuario.

Estas opciones de la parte superior de la interfaz, tanto “Eliminar Usuario” como “Desea salir?” se ven igual en el tablero personalizado del Cliente que se explicará a continuación, por lo cual no es necesario explicarlo nuevamente. Se puede ver más claramente en la sección 5.4 de la Metodología.

6.1.3. Tablero Personalizado Cliente (Empresa del sector de seguros)

Al igual que el tablero anterior, después de tener un Registro o Inicio de sesión exitoso, se muestra en pantalla un tablero personalizado (Ver más sección 5.4 de la Metodología.). En este caso si es un Cliente de La Empresa del sector de seguros se ve la interfaz de la siguiente imagen:

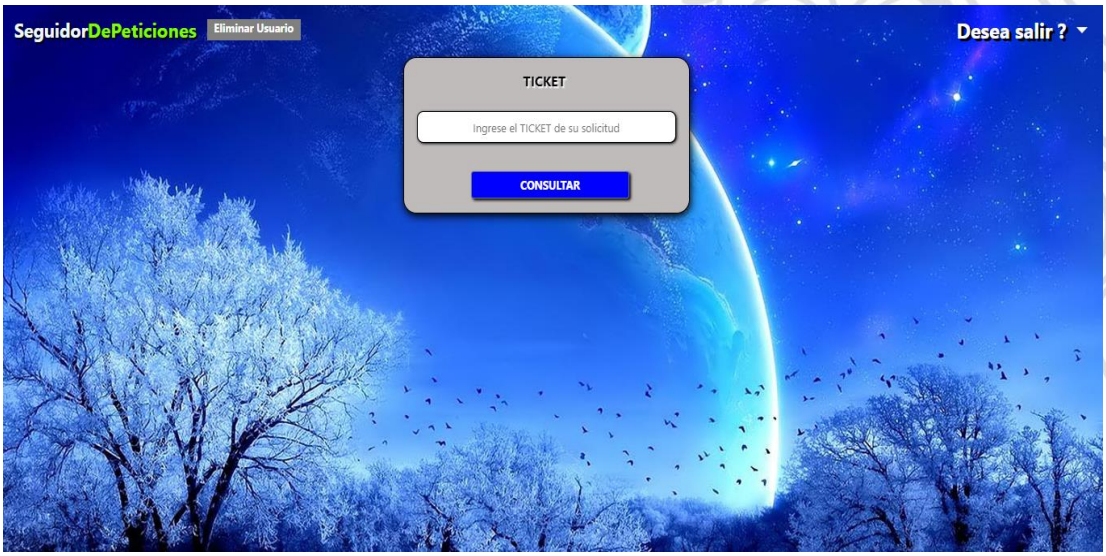

Figura 72. Tablero para Clientes de La Empresa del sector de seguros.

En esta interfaz se tiene la tarjeta contenedora, donde el cliente debe ingresar en el campo "Ingrese el TICKET de su solicitud", el TICKET que el Administrador de la empresa Pragma le suministra para poder realizar su consulta. En este campo se hicieron dos validaciones que fueron: La primera, que si no se ingresa el TICKET en el campo de texto y se da clic en el botón "CONSULTAR", entonces sale un mensaje de alerta como se ve a continuación:

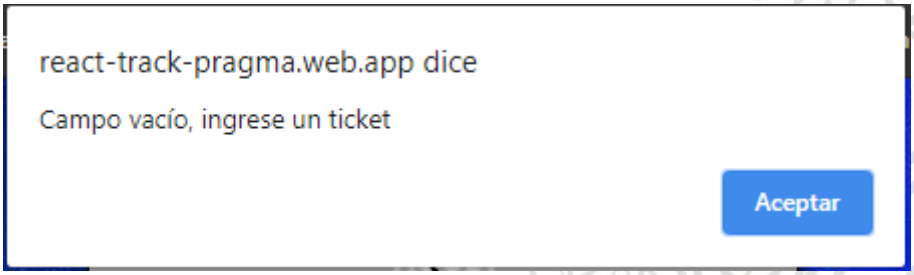

Figura 73. Alerta de campo del TICKET vacío.

Y la segunda validación, que si se ingresa un TICKET que no existe porque aún no haya sido creado o se haya ingresado mal, entonces sale el siguiente mensaje de alerta:

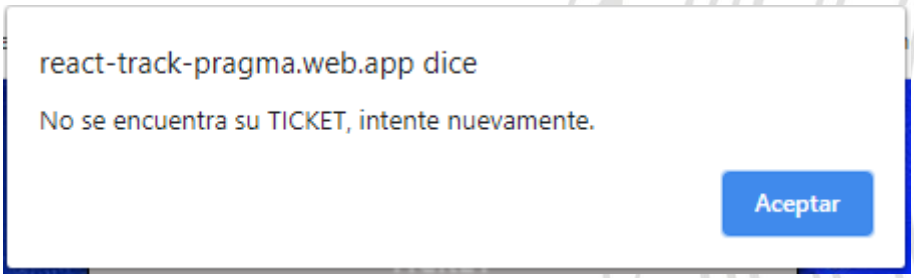

Figura 74. Alerta de TICKET mal ingresado.

Luego de esto, si el TICKET se ingresa correctamente como por ejemplo en la imagen a continuación, que es uno válido, entonces se despliega un tablero en la parte inferior a esta tarjeta (Ver más sección 5.4 de la Metodología.):

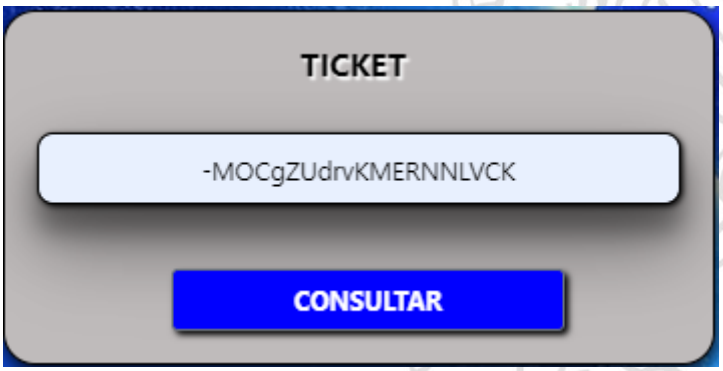

Figura 75. Ingreso de TICKET válido.

Para poder continuar, cuando se muestra el despliegue de la tabla que contiene las listas o estados de la petición del Cliente, se muestra una ventana emergente del navegador para

autorizar si el usuario quiere o no, que la aplicación le muestre notificaciones a través del navegador, como se ve en la siguiente imagen:

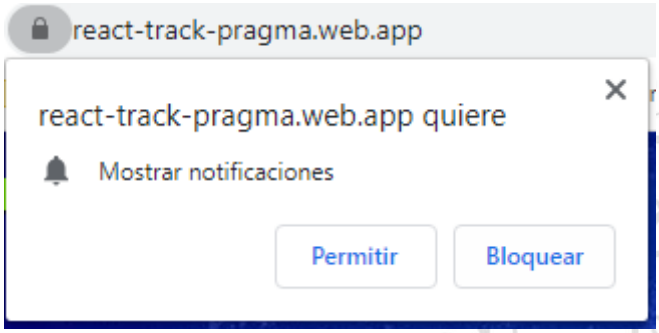

Figura 76. Ventana emergente para permitir notificaciones.

Después de permitir esto, se puede observar que en la parte inferior derecha de la interfaz se muestra la notificación que avisa dónde se encuentra la petición, esta notificación tiene el siguiente aspecto:

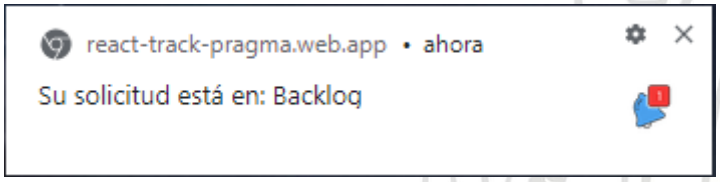

Figura 77. Notificación.

La interfaz completa con la notificación, la petición y su desplazamiento entre estados, se ve en las ***figuras 78, 79 y 80*** de la siguiente manera (Ver más sección 5.4 de la Metodología.):

Figura 78. Interfaz con aviso de notificación.

Figura 79. Interfaz con aviso de notificación.

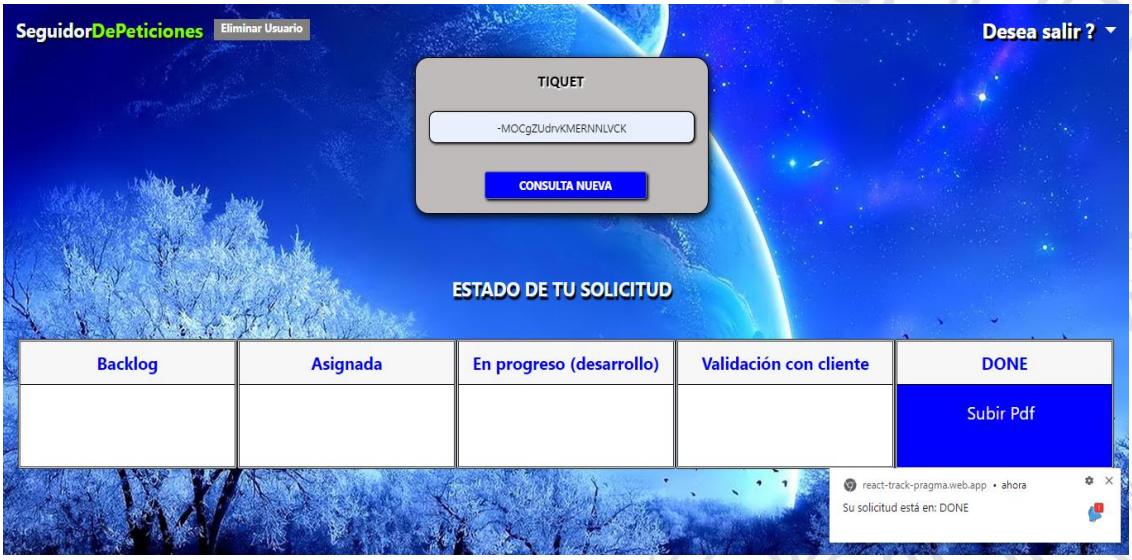

Figura 80. Interfaz con aviso de notificación.

Con esto se termina de mostrar todo lo correspondiente a las interfaces y funcionalidades de desarrollo que se implementaron en el proyecto (Ver más sección 5.2.1 y 5.4 de la Metodología.).

Finalmente, la siguiente imagen muestra la estructura de como se ve alojada la información de las listas y tarjetas en la base de datos de Firebase con sus correspondientes atributos:

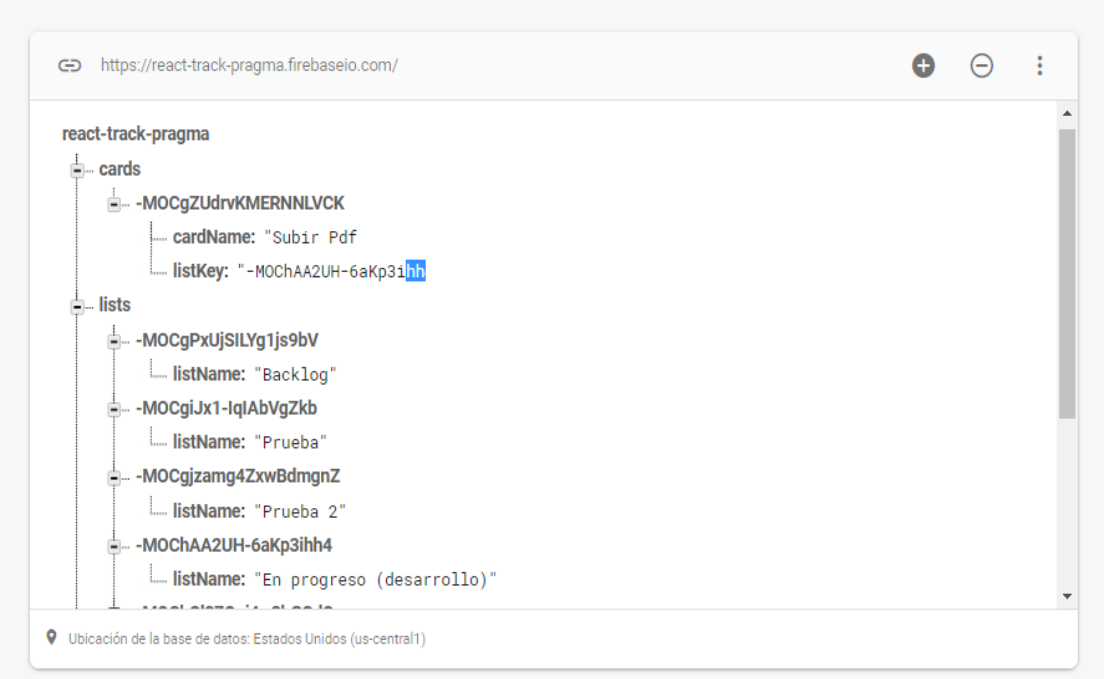

Figura 81. Información alojada en Firebase.

6.2. Análisis

A continuación, se presenta un análisis referente al rendimiento de la aplicación.

6.2.1. Rendimiento de la aplicación

Para esta evaluación del rendimiento hay que tener en cuenta que se hizo el despliegue de la aplicación en un dominio que proporciona Firebase de forma gratuita. Esto se hace con un servicio que ofrece Firebase que se había mencionado anteriormente en la sección 4.8.4 del Marco Teórico, Firebase Hosting, se asigna automáticamente un subdominio de Firebase. Este despliegue es posible, sacando el proyecto de React a producción, ejecutando en consola el comando "***npm* run build**". Esto genera unos archivos compilados los cuales usa Firebase para su despliegue en la red. Cuando se tienen los archivos, simplemente se ejecuta el comando "**firebase deploy**" y este despliega el proyecto en el dominio otorgado que en este caso fue: https://react-track-pragma.web.app/

Al evaluar el rendimiento de esta, se observó que la renderización de cada componente de la aplicación no era muy sutil para la visualización independiente de cada uno, ya que al acceder a secciones en específico de la aplicación, se lograba ver parte de la interfaz de otro componente, aunque es un cambio muy leve, alcanza a ser notorio a la vista del usuario. Esto puede ser solucionado revisando cómo se refrescan los componentes de la página y dar una prioridad en cómo se renderizan estos. Otro parámetro que se tuvo en cuenta para evaluar el rendimiento, fue con ayuda del **Chrome DevTools**. En la siguiente imagen se muestran 3 parámetros fundamentales a la hora de analizar el rendimiento, por ejemplo de la interfaz principal donde están los formularios:

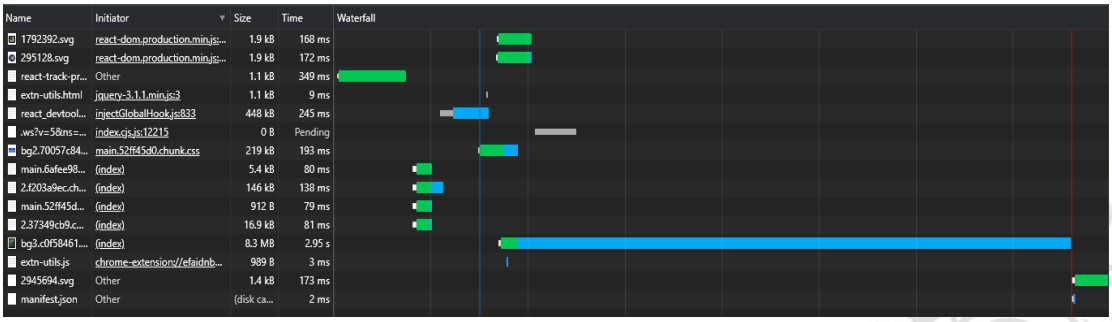

Figura 82. Gráfica de rendimiento 1.

La tabla devuelve demasiada información: Nombre del recurso, Estado del servidor, tipo de documento, etc. Pero todos estos se pudieron filtrar y dejarse los más relevantes, que se muestran en 3 columnas que son realmente útiles:

Size: Es el tamaño del recurso. En este caso cada recurso tiene sus particularidades: Por ejemplo, una imagen tiene un peso que se puede reducir a costa de perder calidad, y un CSS se puede reducir quitando comentarios o reduciendo líneas.

Time: El tiempo que tarda en cargarse no solo depende del tamaño, la ubicación y/o el rendimiento del servidor puede incidir en este factor.

Waterfall o cascada: Podemos ver, de forma gráfica, cuándo se cargan los elementos y cuánto tardan.

Se logra observar de la figura 82, lo que más demora en cargar y ralentiza la página es la imagen “bg3” del index, que es la imagen principal de la aplicación cuando se entra por primera vez, ya que se demora 2.95 segundos en cargar, esto se podría mejorar comprimiendo la imagen o cargando una de menor peso, ya que al comprimir la imagen se podría perder calidad en esta. También al observar el rendimiento de la página al iniciar sesión con usuario de Pragma, se logra ver la siguiente imagen:

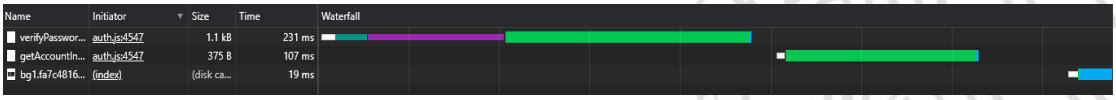

Figura 83. Gráfica de rendimiento 2.

Se logra analizar que lo que más tiempo demora en cargar la página es la verificación del usuario en la base de datos. Y por último se tomó el rendimiento la página al iniciar sesión con usuario de La Empresa del sector de seguros, donde se logra ver la siguiente imagen:

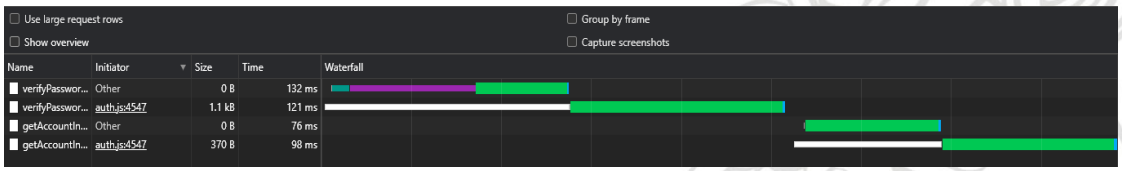

Figura 84. Gráfica de rendimiento 3.

Se logra analizar al igual que en la explicación anterior, que lo que más tiempo demora en cargar la página es la verificación del usuario en la base de datos.

7. Conclusiones

Se pudo evidenciar que, al evaluar las tareas requeridas para la realización del proyecto, se tuvo en cuenta muchos aspectos fundamentales como el tiempo que se necesitó para estructurar, definir y realizar el proyecto debido a la complejidad de los temas a tratar. Debido a esto, se tuvo que hacer un cambio en las herramientas que inicialmente se habían propuesto, ya que se necesitaba de más tiempo para adquirir los conocimientos necesarios.

Ya que se contaba con un desconocimiento inicial de cómo eran las reglas de negocio con las que se trabajan tanto en la empresa como en el equipo de desarrollo, esto dificulto aún más el proceso por lo que se tuvo que dedicar parte del tiempo a conocer el contexto y así poder idear un plan de desarrollo para abordar el proyecto. Conociendo esto ya se pudo definir, tanto las necesidades del cliente como los criterios de prioridad que se querían asignar a los procesos abordados.

También se reconoce que hay herramientas que facilitan mucho a los programadores a la hora de realizar sus desarrollos o aplicaciones, como en el caso del presente proyecto donde se encontraron herramientas muy útiles para el desarrollo como fueron Firebase, que prestaba sus servicios como back-end y base de datos, Bootstrap que ya contenía elementos prediseñados, Chrome DevTools que tiene una variedad de ayudas, como ver los errores en la consola del navegador o poder ajustar los estilos visuales de forma local para luego ser aplicada en el proyecto. Todas estas herramientas fueron de mucha utilidad por lo que ayudó a reducir el tiempo empleado en tareas que probablemente no tenían tanta importancia como otras pero que igualmente debían ser realizadas.

Se puede concluir también que a pesar de los inconvenientes que se pudieron haber presentado, se cumplió con el objetivo principal del proyecto más enfocado a la funcionalidad, pero sin descuidar la parte visual que era representada por una interfaz amigable para el usuario y fácil de manejar. Dado esto, se pudo establecer un canal de comunicación entre el cliente y el administrador, ya que ahora el cliente al realizar una solicitud podrá estar enterado de cuál es su situación o en qué estado se encuentra, esto se hizo con el fin de mejorar el servicio prestado actualmente.

Además, al evaluar el rendimiento final de la aplicación teniendo en cuenta el diseño y la funcionalidad, se debe dejar claro que, al ser una versión inicial, se pudo cumplir con su objetivo principal, pero dejando muchas posibilidades para mejorar. Se podrían valorar mejores formas de cargar imágenes, estructurar el código, mejorar el tiempo de respuesta de la página al acceder a diferentes elementos de esta, por ejemplo, al iniciar sesión o cuando se ingresa por primera vez a la aplicación.

8. Referencias Bibliográficas

1. Agencia Iberoamericana para la Difusión de la Ciencia y la Tecnología. (2020). La sonda espacial Gaia permitirá analizar estrellas en 3D. Disponible en: https://www.dicyt.com/noticias/la-sonda-espacial-gaia-permitira-analizar-estrellas-en-3d

2. PRAGMA.(s.f.). Acerca de. Linkedin. Disponible en: https://www.linkedin.com/company/somospragma/?originalSubdomain=co

3. Great Place To Work. (2019). Los mejores lugares para trabajar en Colombia. disponible en: https://www.greatplacetowork.com.co/es/listas/colombia2019/mas-de-500/

4. Workana. (s.f.). ¿Qué es la creación de contenidos?. Disponible en: https://www.workana.com/i/glosario/creacion-de-contenidos/

5. MDN contributors. (2020). HTML: Lenguaje de etiquetas de hipertexto. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

6. B,G. (2019). ¿ Qué es CSS?. Hostinger Tutoriales. Disponible en: https://www.hostinger.co/tutoriales/que-es-css/

7. Grados Caballero, J,G. (2020). ¿ Qué es Java Script?. Dev Code. tomado de: https://devcode.la/blog/que-es-javascript/

8. MDN contributors. (2020). ¿ Qué es Java Script?. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

9. Stefaniak, P. (2019). ¿Qué es Backend y Frontend?. Descubre comunicación. Disponible en: https://descubrecomunicacion.com/que-es-backend-y-frontend/

10. Netec. (s.f.). ¿Para qué sirve una base de datos?. Disponible en: https://www.netec.com/para-que-sirve-una-base-de-datos

11. NeoAttack. (s.f.). Framework. Disponible en: https://neoattack.com/neowiki/framework/

12. Orix. (2015). ¿Qué es un framework y para qué se utiliza?. Disponible en: https://www.orix.es/que-es-un-framework-y-para-que-se-utiliza

13. Luma Studio Digital. (s.f.). Desarrollo Frontend. Disponible en: https://www.lumastudio.com/frontend.html.

14. Guajardo,P. (2020). Bootstrap: guía para principiantes de qué es, por qué y cómo usarlo. rockcontent. Disponible en: https://rockcontent.com/es/blog/bootstrap/

15. Jeavor. (2017). Front-end y Back-end ¿Qué son?. desarrollandolo. Disponible en: https://www.desarrollandolo.com/front-end-y-back-end-que-son/detalle-blog

16. Chapaval, N. (2019). Qué es Frontend y Backend. platzi. Disponible en: https://platzi.com/blog/que-es-frontend-y-backend/

17. Falcon, C. (2014). Explicando que es Front-end y que es Back-End. Falcon Masters. Disponible en: http://www.falconmasters.com/web-design/que-es-front-end-y-que-es-back-end/#

18. Netec. (s.f.). ¿Para qué sirve una base de datos?. Disponible en: https://www.netec.com/para-que-sirve-una-base-de-datos

19. Wikipedia. (3 de enero de 2021). React. Disponible en: https://es.wikipedia.org/w/index.php?title=React&oldid=132125561

20. Wikipedia. (12 de noviembre de 2020). Firebase. Disponible en: https://es.wikipedia.org/w/index.php?title=Firebase&oldid=130874282

21. Álvarez, M. (2019). Qué es React. Por qué usar React. desarrollo web. Disponible en: https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html

22. Álvarez, M. (2008). Qué es el DOM. desarrollo web. Disponible en: https://desarrolloweb.com/articulos/que-es-el-dom.html

23. Quality Devs. (2019). ¿Qué es Angular y para qué sirve?. Disponible en: https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

24. React. (s.f). Componentes y propiedades. Disponible en: https://es.reactjs.org/docs/components-and-props.html

25. García, D. (2016). Características de React. desarrollo web. Disponible en: https://desarrolloweb.com/articulos/caracteristicas-react.html

26. Vega, A. (2017). Tipos de Componentes en ReactJS. Adrian Alonso dev. Disponible en: https://adrianalonsodev.medium.com/tipos-de-componentes-en-reactjs-f387a6f8e2b7

27. Blancarte, O. (2018). Componentes con estado y sin estado. oscar blancarte blog. Disponible en: https://www.oscarblancarteblog.com/2018/09/26/componentes-con-estado-y-sin-estado/

28. Cervantes, J. (2018). React: Métodos del ciclo de vida de un componente. pensemos web Disponible en: https://www.pensemosweb.com/react-metodos-ciclo-vida-componente/

29. React. (s.f). Empezando. Disponible en: https://es.reactjs.org/docs/getting-started.html

30. Chrome DevTools. (2020). Disponible en: https://developers.google.com/web/tools/chrome-devtools?hl=es

31. Álvarez, C. (2016). ¿Qué es Spring Boot?. arquitectura java. Disponible en: https://www.arquitecturajava.com/que-es-spring-boot/

32. Cuervas, J. (2019). ¿Qué es Spring Framework?. at sistemas. Disponible en: https://www.atsistemas.com/blog/qu-es-spring-framework-caractersticas-i

33. Aguilar, J. (2017). Desarrollo de aplicaciones Web con Spring Boot. software evolutivo. Disponible en: http://softwareevolutivo.com.ec/espanol-desarrollo-de-aplicaciones-web-con-spring-boot/

34. Ruiz, C. (2018). Spring Boot – ¿Qué es? y cómo se come?. cristian ruiz blog. Disponible en: http://cristianruizblog.com/spring-boot-que-es-y-como-se-come/

35. Wikipedia. (31 de Diciembre 2020). Visual Studio Code.. Disponible en: https://es.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=132071565

36. Amazon Web Services. (s.f.). AWS Cloud Practitioner Essentials (Second Edition) (Spanish): Introducción a la nube de AWS. Disponible en: https://www.aws.training/Details/eLearning?id=46156

37. Amazon Web Services. (s.f.). Informática en la nube con AWS. Disponible en: https://aws.amazon.com/es/what-is-aws/

38. López, S. (2020). Firebase: qué es, para qué sirve, funcionalidades y ventajas. digital55. Disponible en: https://www.digital55.com/desarrollo-tecnologia/que-es-firebase-funcionalidades-ventajas-conclusiones/

39. Muradas, Y. (2019). Qué es NPM y para qué sirve. openwebinars. Disponible en: https://openwebinars.net/blog/que-es-node-package-manager/

40. Best Free HTML/CSS Templates. (2019). Reactstrap: Componentes ReactJS Sin Estado Para Bootstrap 4. Disponible en: https://es.bestfreehtmlcsstemplates.com/plantillas/473/reactstrap-componentes-reactjs-sin-estado-para-bootstrap-4

41. Wikipedia. (29 de mayo de 2020). Single-page application. Disponible en: https://es.wikipedia.org/w/index.php?title=Single-page_application&oldid=126488860

42. Álvarez, M. (2016). Introducción a Firebase. desarrollo web. Disponible en: https://desarrolloweb.com/articulos/introduccion-firebase-backend-nube.html

43. Wikipedia. (18 de diciembre de 2020). JSON. Disponible en: https://es.wikipedia.org/w/index.php?title=JSON&oldid=131796966

44. Keren, G. (2018). Android FCM — Sharing is caring!. Disponible en: https://medium.com/@gadi.krn

45. Firebase. (2020). Explicación de las reglas de Firebase Realtime Database. Disponible en: https://firebase.google.com/docs/database/security?hl=es

46. Firebase. (2020). Precios. Disponible en: https://firebase.google.com/support/faq?hl=es-419

47. Amazon Web Services. (s.f.). Precios de AWS Amplify. Disponible en: https://aws.amazon.com/es/amplify/pricing/

9. Anexos

Firebase vs AWS y Spring-Boot

A continuación, se darán detalles de las características de cada servicio para al final hacer una comparación y análisis de por qué se escogió Firebase para el desarrollo final del proyecto.

Precios de Firebase [46]

Firebase ofrece dos planes para sus clientes. El Plan Spark es ideal para desarrollar aplicaciones grandes y pequeñas y proporciona un nivel gratuito. El plan Blaze funciona con un modelo de pago sobre la marcha y le cobra todos los límites excedidos en el nivel gratuito.

- Plan Spark, Nivel Gratuito, ideal para desarrollo y aplicaciones pequeñas.
- Plan Blaze, modelo de pago sobre la marcha e ideal para aplicaciones de producción grande.

¿Cuáles son los beneficios de Firebase?

- Las mejores capacidades de ejecución en tiempo real
- Consola fácil de usar y completamente integrada
- Se ejecuta en la nube de Google
- Escalabilidad

Precios de AWS Amplify [47]

El precio de Amplify es similar al de otros productos de AWS e incluye un nivel gratuito limitado y un modelo de pago sobre la marcha para uso adicional.

- Nivel gratuito: 5 GB de almacenamiento, 1000 compilaciones, caduca después de 12 meses
- Pago sobre la marcha: $0.01 por minuto de compilación, $0.023/GB/mes

¿Cuáles son los beneficios de AWS Amplify?

- Almacén de datos del dispositivo local
- Soporta bases de datos SQL y No-SQL
- Admite API GraphQL y REST
- Integración con AWS
- Escalabilidad

A pesar de los beneficios similares que ofrece AWS para el almacenamiento de datos, se escogió Firebase ya que venía incluido el uso del Back-end en sus servicios, permitiendo que el programador no deba preocuparse por la comunicación entre el Front-end y la base de datos, sino que el servicio viene integrado, con los dos en uno.