Prototipo Software Para Visualizar la Cobertura de Radio Digital En Colombia

Giovanni Ernesto Ibañez Acevedo

Asesor

Mónica Andrea Rico Martínez

Universidad Nacional Abierta y a Distancia – UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería ECBTI

Ingeniería Electrónica

2025

Agradecimientos

Agradezco a la profesora. Mónica Andrea Rico Martínez quien, con su compromiso, entrega y profesionalismo, me ha hecho el respectivo acompañamiento en este proyecto de grado, permitiéndome ser apoyo para aprendizaje y desenvolvimiento de todas las competencias que aquí son demostradas.

Dedicatoria

A mis, amigos y demás personas que han creído en mi crecimiento personal y profesional y me han apoyado de una u otra forma para lograr mis objetivos académicos.

A mi señora madre por su especial empeño en apoyarme moral y económicamente para esforzarme día a día en mi proceso de aprendizaje cuyo progreso lleno de mucho esfuerzo, entusiasmo y sacrificio me ha traído hasta esta situación donde se me da la oportunidad de demostrar muchas de las aptitudes actuales.

A la comunidad universitaria de las UNAD por su dedicación y contribución directa o indirecta para que este proceso educativo se haya dado de manera amena y eficaz.

Resumen

La Comisión de Regulación de Comunicaciones (CRC) ha evidenciado la necesidad de estandarizar y ampliar el uso de la radio digital en Colombia. Por ello, ha iniciado estudios preliminares enmarcados en un proyecto mayor, donde la UNAD implementa el estudio de diferentes lugares de recolección de datos en determinadas coordenadas geográficas claves del país por medio de simulaciones.

Con el tiempo, la base de datos de coordenadas en los cuales se ha simulado la cobertura, actualmente estos datos de propiedad UNAD se alojan en un sitio web de xirio, de forma poco estructurada y son exportables en formato KMZ. Este proyecto recolecta y muestra estos datos de acuerdo con la coordenada correspondiente, en el mapa colombiano mediante un nuevo sitio web. Al hacer click en un punto de interés, se deben mostrar detalladamente información detallada mediante un dashboard de datos que muestra un análisis puntual y global de la información recolectada y calculada.

Este aplicativo pretende presentar datos grafica e interactivamente con el fin de mejorar su accesibilidad por parte de los usuarios finales y testear su viabilidad y aplicación.

Este documento expone el diseño e implementación de un dashboard en Power BI, desarrollado a partir de datos no estructurados provenientes de Xirio. Posteriormente, se presenta el diseño, implementación y pruebas de un prototipo de aplicativo web capaz de gestionar, guardar y exponer al público dashboards como el anteriormente mencionado. Se comprueba la viabilidad de estas implementaciones.

Palabras clave: Dashboard, KMZ, DAB, Servidor Web, API

Abstract

The Communications Regulation Commission (CRC) has identified the need to standardize and expand the use of digital radio in Colombia. As a result, it has launched preliminary studies as part of a larger project in which the UNAD conducts research in various key geographic locations throughout the country using simulations to collect data.

Over time, a database of coordinates, where coverage simulations have been performed, has been developed. These data, currently owned by UNAD, are hosted on a Xirio website in an unstructured format and are exportable in KMZ format. This project aims to gather and display these data based on their corresponding coordinates on a map of Colombia through a newly developed website. By clicking on a point of interest, a data dashboard is displayed, offering both detailed and global analyses of the collected and calculated information.

This application aims to present data graphically and interactively in order to improve its accessibility for end users and test its feasibility and application.

This document outlines the design and implementation of a Power BI dashboard developed from unstructured Xirio data. It then presents the design, development, and testing of a prototype web application capable of managing, storing, and publicly displaying dashboards like the one previously mentioned. These implementations have been proven to be feasible.

Keywords: Dashboard, KMZ, DAB, Servidor, Web, API

Tabla de Contenido

Introducción 14

Generalidades 16

- Problema de Investigación 16
- Objetivos de la Investigación 18
- Objetivo General 18
- Objetivos específicos 18
- Justificación 19
- Marco Referencial y Conceptual 21
- Software Para Visualizar Datos 21
- Dashboard 23
- Ciclo de Vida del Software 24
- Tecnologías Utilizadas Para la Creación de Dashboards 25
- Conceptos Principales de Radio Digital 26
- Estándares de radio digital 28
- Estado del Arte 32
- Metodología 35
- Planificar 35
- Hacer 35
- Verificar 35
- Actuar 36

Requerimientos de Software 37

- Análisis del Dominio 38
- Modelo de Análisis 39
- Propósito del Prototipo de Software 40
- Alcance del producto / Software 40

Diseño del Dashboard y Aplicativo Web 42

- Análisis y Visualización de Datos 42

Recopilación de Datos 42
Limpieza y Transformación 42
Exploración, transformación y modelado de datos 44
Visualización de datos 45
Entorno Operativo del Aplicativo web 49
Requerimientos Funcionales 50
Flujo de Datos Principal 57
Funcionalidad de actualización 58
Clases y Características de Usuarios 59
Caso de uso de Usuario Administrador 59
Caso de uso de Usuario Visitante 66
Desglose de Casos de Uso 68
Diseño UI 74

Implementación del Prototipo de Dashboard y Aplicativo Web 76

Conceptualización 76
Arquitectura Cliente-Servidor 76
Arquitectura Frontend con Vue 3, Composition API y TypeScript 77
Arquitectura de Laravel 11 para API REST con Protección JWT 78
Uso de MySQL y Apache para la Administración con Laravel 11 83
Implementación del dashboard 83
Implementación de Fronted 85
Configuración Inicial 85
Disposición General 87
Contenedor Fronted 89
Servicios 95
Implementación de Backend 99
Configuración Inicial 100
Implementación de Migraciones 104
Implementación de Modelos 106
Implementación de EndPoints 109

Implementación de Controladores 111
Implementación de JWT 119
Prueba de Acceso al Sitio 128
Prueba de Registro del Primer Administrador 130
Prueba de Inicio de Sesión 133
Prueba de Otorgamiento de Autorizaciones 135
Pruebas de Cierre de Sesión 138
Pruebas de Registro y Cambio de Datos 140
Pruebas Gestión de Dashboards 146
Conclusiones 152
Referencias Bibliográficas 154

Lista de Tablas

Tabla 1 *Iniciar Sesión, caso de Uso* 68

Tabla 2 *Registro de Administrador, Caso de Uso* 69

Tabla 3 *Otorgar Autorización, Caso de Uso* 70

Tabla 4 *Cerrar Sesión, Caso de Uso* 70

Tabla 5 *Regresar, Caso de uso* 71

Tabla 6 *Cambiar Contraseña, Caso de Uso* 71

Tabla 7 *Actualizar Datos de Administrador, Caso e Uso* 72

Tabla 8 *Crear Dashboard, Caso de Uso* 73

Tabla 9 *Métodos de Peticiones HTTP* 80

Tabla 10 *Códigos de Respuesta HTTP* 82

Tabla 11 *Controladores-Descripción* 113

Tabla 12 *Relación Entre Métodos Controladores, Modelos y Endpoints* 114

Tabla 13 *Datos del Primer Administrador* 131

Tabla 14 *Datos de Autorizaciones* 135

Tabla 15 *Datos registro de Admin1 Y Admin2* 140

Tabla 16 *Datos de Cuadro de Dashboards* 147

Lista de Figuras

Figura 1 *Modelo de Requerimientos*.....38

Figura 2 *Entradas y Salidas Para el Análisis del Dominio*.....38

Figura 3 *Elementos del Modelo de Análisis*.....39

Figura 4 *Muestra Original de la Agrupación de Columnas de la Plantilla*.....43

Figura 5 *Muestra del Resultado Después de la Transformación de Formato*.....43

Figura 6 *Diseño previo del Dashboard*.....49

Figura 7 *Diagrama de Flujo de Datos Principal*.....58

Figura 8 *Diagrama de Flujo de Caso de Uso de Administrador*.....61

Figura 9 *Entidad 'Map' de la Base de Datos*.....62

Figura 10 *Icono Menú Hamburguesa*.....63

Figura 11 *Entidad 'Admin' de la Base de Datos*.....65

Figura 12 *Entidad 'Authorization' de la Base de Datos*.....66

Figura 13 *Diagrama de Flujo de Caso de Uso de Visitante*.....67

Figura 14 *Cuadro de Información de Dashboard*.....67

Figura 15 *Captura del Dashboard Generado*.....85

Figura 16 *Estructura Fronted en Vue 3*.....88

Figura 17 *Distribución de Componentes*.....91

Figura 18 *Árbol de Directorios de Componentes*.....93

Figura 19 *Relación entre Componentes View Y UI de Vue*.....94

Figura 20 *Relación de Bloques Funcionales del Fronted*.....95

Figura 21 *Diagrama General de Backend*.....104

Figura 22 *Relación de Controladores con Modelos y Endpoints*..... 112

Figura 23 *Cifrado Hash de Contraseñas*..... 118

Figura 24 *Implementación de JWT en laravel 11* 121

Figura 25 *Activación de Apache y MySQL*..... 128

Figura 26 *Ejecución de Migraciones*..... 129

Figura 27 *Acceso al Sitio*..... 130

Figura 28 *Acceso al Sitio, Respuesta del API*..... 130

Figura 29 *Formulario de Registro De Admin0*..... 131

Figura 30 *Popup Alerta de Creación del Registro*..... 132

Figura 31 *Evidencia Conexiones al Endpoint de Registro* 132

Figura 32 *Consulta SQL de Admin0*..... 133

Figura 33 *Diligenciamiento de Formulario de Inicio de Sesión*..... 133

Figura 34 *Conexiones al Endpoint de Login*..... 134

Figura 35 *Panel de Administrador de Admin0* 134

Figura 36 *Diligenciamiento de Autorización de Admin1* 136

Figura 37 *Diligenciamiento de Autorización de Admin2* 136

Figura 38 *Peticiones a endpoint de Autorizaciones* 137

Figura 39 *Resultado de Consulta de Tabla 'Authorization'*..... 137

Figura 40 *Vista de Autorizaciones Creadas* 138

Figura 41 *Vista de Alerta de Sesión Cerrada* 139

Figura 42 *Evidencia de Conexión Al Endpoint De Logout*..... 139

Figura 43 *Evidencia de Conexión al Endpoint de Logout*..... 140

Figura 44 *Prueba de Registro de Admin1* 141

Figura 45 *Prueba de Registro de Admin2* 141

Figura 46 *Evidencia de Cambio de Datos De Admin2* 142

Figura 47 *Cambio de Contraseña* 143

Figura 48 *Diligenciamiento de Formulario de Eliminar Cuenta* 144

Figura 49 *Peticiones a endpoints Durante la Prueba de Gestión de Dashboards* 148

Figura 50 *Vista de Formulario de Carga de Dashboards* 149

Figura 51 *Formulario de Edición de Dashboards* 149

Figura 52 *Vista Preliminar del Dashboard* 150

Figura 53 *Estado del Panel Público* 150

Figura 54 *Estado Final de la Tabla 'Map'* 151

Lista de Apéndices

Apéndice A *Código de Limpieza y Transformación de Datos en Excel Mediante VB*..... 157

Apéndice B *Resumen de Consultas DAX* 161

Apéndice C *Documentación Sobre el Modelo de Consultas DAX*..... 166

Apéndice D *Códigos MainV.vue y AdminV.vue*..... 171

Introducción

La era digital actual se distingue por una transformación constante en el ámbito de la informática y en las formas de comunicación. Uno de los sectores en los que se han logrado avances significativos es la radiodifusión, donde la implementación de señales digitales ha permitido mejoras sustanciales en términos de calidad y eficiencia.

Un factor crítico para el éxito de esta transición es el monitoreo y la visualización de la potencia de señal, ya que resulta esencial garantizar una cobertura adecuada y una calidad óptima en la transmisión. El análisis e interpretación de los datos de potencia de señal proporciona información relevante sobre la eficiencia del sistema, la detección de posibles problemas en la transmisión y la toma de decisiones orientadas a mejorar la calidad de la señal.

Con el propósito de respaldar el estudio sobre la implementación de la radiodifusión digital en Colombia, se plantea el desarrollo de un aplicativo destinado a la visualización de datos de potencia de señal. Dicha herramienta permitirá a los profesionales del sector de las telecomunicaciones y a los responsables de la implementación de la radiodifusión digital en el país analizar y evaluar en tiempo real la calidad de la señal, identificar posibles fallas o interferencias y adoptar medidas correctivas de manera eficiente.

En el marco del proceso de implementación de la radiodifusión digital en Colombia, se propone el desarrollo de un aplicativo web para la visualización y el análisis de datos de potencia de señal. Esta herramienta proporcionará a los profesionales del sector telecomunicaciones la capacidad de evaluar la calidad de la señal en tiempo real, detectar posibles fallas o interferencias y aplicar medidas correctivas de forma eficiente.

Este prototipo de software adopta un modelo cliente-servidor, en el cual se utilizará Vue 3 para el desarrollo del frontend y Laravel para la construcción del backend, empleando MySQL

como gestor de base de datos. Se detalla de manera estructurada cada una de las tecnologías empleadas y su aplicación práctica, abordando los fundamentos teóricos, los requisitos de software, las etapas de análisis de datos previos mediante Excel y Power BI para la generación del dashboard, así como la arquitectura del sistema.

Se espera que este aplicativo contribuya de manera significativa al proceso de digitalización de la radiodifusión en Colombia, proporcionando una herramienta eficiente y moderna que facilite el análisis técnico y la toma de decisiones estratégicas, promoviendo con ello el avance tecnológico y socioeconómico del país.

Este trabajo escrito documenta en el primer capítulo, las generalidades, contextualización necesaria sobre la realización de este trabajo como lo es el problema, los objetivos, justificación, marco y metodología. El segundo capítulo describe los requerimientos de software, el tercer capítulo el diseño, el cuarto la implementación detallada del prototipo del producto y en el cuarto capitulo se presentan sus respectivas pruebas de validación de software.

Generalidades

En el presente capitulo se presenta un contexto de la propuesta del proyecto, referente al planteamiento del problema, justificación, objetivos, marco teórico y conceptual, estado del arte y enfoque metodológico.

Problema de Investigación

En la actualidad, la tecnología de radiodifusión analógica sigue en auge y se está llevando a cabo un proceso global de migración tecnológica al cual nuestro país pretende sumarse debido a las muchas ventajas y oportunidades que esta tecnología ofrece.

La radiodifusión analógica se divide, según el tipo de modulación, en AM (amplitud modulada) y FM (frecuencia modulada). En AM, se transmite la información mediante cambios de amplitud en una señal de alta frecuencia denominada portadora, mientras que la señal moduladora, que determina los cambios en la portadora, tiene una frecuencia muy inferior. Por otro lado, en FM, la información se refleja en los cambios de frecuencia de la señal portadora, los cuales son determinados por la señal moduladora.

Debido a la gran diferencia entre los órdenes de frecuencia de las señales moduladoras y portadoras, y a su naturaleza analógica, la oportunidad de utilizar el ancho de banda se ve comprometida en muchos casos. Las señales analógicas, una vez procesadas, se interpretan directamente como sonidos, lo que evidencia su vulnerabilidad ante ruidos. (Olmo & Nave, 1999).

En contraste, la radio digital utiliza valores fijos y discretos de frecuencias que se traducen en valores numéricos determinados y, al ser procesados, se convierten en información. La recuperación de la información depende del tamaño de los mensajes recibidos, pero debido a

que los valores fijos tienen pocos valores posibles, los errores y ruidos pueden descartarse fácilmente. (Fox & Waisbord, 2021)

Según Ministerio de Tecnologías de la Información y las Comunicaciones y la Agencia Nacional del Espectro (2012), actualmente, la Agencia Nacional del Espectro (ANE) es la entidad encargada en Colombia no solo de brindar soporte técnico, sino también de la gestión de políticas, control y vigilancia de su uso. El espectro radioeléctrico se considera un bien nacional y, por lo tanto, su uso debe ser protegido y regulado por el Estado para garantizar cobertura y derecho de uso equitativo tanto para el sector público como para el sector privado. Jurídicamente, el Ministerio de Tecnologías de la Información y las Comunicaciones (MinTic) cumple funciones de administrador estatal con el fin de ejercer gestión y control, garantizando igualdad de oportunidades para su acceso y uso, evitando así prácticas monopólicas en el uso de este bien.

Dada la tendencia actual de transición hacia la tecnología de la radio digital, el sector de las TIC (conformado por la ANE, ANTV, MinTic, SIC) evidencia la necesidad de implementar un aplicativo que brinde información sobre la cobertura de la tecnología de radiodifusión digital en un contexto geográfico, para facilitar la toma de decisiones en búsqueda de soluciones que se acoplen a las necesidades y cumplan con las normativas y políticas vigentes.

En la actualidad, existen herramientas de software como Xirio que realizan cálculos simulados de cobertura radioeléctrica. Sin embargo, el aplicativo a desarrollar pretende complementar estas funcionalidades, facilitando la exposición de información de cálculos y toma de decisiones.

Existe un aplicativo web que brinda este servicio para dicha tecnología. El visor de espectro es implementado por la ANE (Agencia Nacional del Espectro del gobierno colombiano)

y consta de mapas interactivos donde se presenta el globo terrestre, aunque los filtros de búsqueda se limitan a Colombia. En él se plasman puntos de información relacionados con la planeación, atribución, vigilancia y control del espectro radioeléctrico en Colombia, incluyendo datos técnicos como el ancho de banda, la cobertura, la ganancia y la potencia del transmisor, entre otros.

Con base en esto, se plantea la realización de un aplicativo web tipo dashboard similar al anteriormente descrito. En este aplicativo, se presentan barras informativas y objetos embebidos provenientes de la herramienta de análisis y visualización de datos Power BI. De esta manera, se proporcionan métricas sobre los parámetros necesarios para la implementación de estaciones de radio digital. Para realizar lo anterior, se plantea entonces la siguiente pregunta de investigación: ¿Cómo se diseña y desarrolla un prototipo web que permita la visualización de las métricas de cobertura del servicio de la radiodifusión digital, para la toma de decisiones?

Objetivos de la Investigación

Objetivo General

Desarrollar un prototipo de software que permita la visualización de las métricas de cobertura del servicio de radiodifusión digital a través de un aplicativo web.

Objetivos Específicos

Realizar el levantamiento de los requerimientos del prototipo software, teniendo en cuenta las necesidades del país y los estudios realizados a la fecha sobre radio digital.

Diseñar y realizar el aplicativo web para visualizar las métricas de cobertura de radiodifusión digital mediante un dashboard con apoyo en el lenguaje de programación Java Script.

Realizar la integración de los elementos del prototipo, base de datos, aplicativo y *dashboard* mediante el lenguaje de programación seleccionado.

Validar el prototipo software, mediante pruebas de funcionalidad.

Documentar el prototipo mediante la realización del documento final.

Justificación

El espectro radioeléctrico es un elemento esencial para el desarrollo de los países, ya que su aprovechamiento conlleva beneficios en diversas áreas y está vinculado con un sinnúmero de actividades dentro de la sociedad. No solo se considera un recurso clave para el acceso y uso de los servicios públicos de telecomunicaciones, sino que también tiene aplicaciones en ámbitos privados, comerciales, de consumo, defensa y seguridad nacional, educación, ciencia, investigación, salud, seguridad pública, agricultura y el desarrollo del trabajo en distintas áreas del conocimiento y la industria humana. Debido a su impacto transversal en múltiples sectores económicos y sociales, su administración eficiente resulta fundamental para la modernización tecnológica, la libre competencia y el bienestar general.

Según el Ministerio de Tecnologías de la Información y las Comunicaciones y la Agencia Nacional del Espectro, el continuo avance tecnológico en el campo de las telecomunicaciones juega un papel clave en la economía de un país, facilitando la generación y explotación de recursos, la creación de empleo y el crecimiento económico. Además, la difusión de información a través de estas tecnologías contribuye directamente al bienestar de la población. En las últimas décadas, la demanda de estos servicios ha aumentado exponencialmente, así como la diversidad de aplicaciones que pueden aprovechar el espectro radioeléctrico. Su uso adecuado impulsa el desarrollo nacional y permite la inclusión de poblaciones vulnerables en zonas de difícil acceso, promoviendo una sociedad más informada, eficiente y modernizada tecnológicamente.

En este contexto, se hace necesario fortalecer la gestión del espectro radioeléctrico y facilitar su migración tecnológica, especialmente en el marco de los planes para la implementación de la radiodifusión digital. La transición a esta tecnología implica diversos retos de carácter económico, social, jurídico y técnico que requieren un análisis profundo y herramientas adecuadas para su estudio. Por ello, el presente proyecto propone la implementación de una plataforma tipo Dashboard con fines informativos, enfocada en facilitar la correlación y análisis detallado de los datos asociados a la disponibilidad del espectro radioeléctrico y su geolocalización.

El aplicativo suministra información estructurada para facilitar su análisis y toma de decisiones. Este Dashboard fue diseñado para permitir a los usuarios visualizar patrones relevantes en los datos, lo que facilitará la optimización de la red y la implementación de estrategias para la prestación del servicio de radiodifusión digital. Además, abordará aspectos técnicos fundamentales, como la identificación de rutas de mantenimiento más eficientes, la optimización en la localización de transmisores, la evaluación de la viabilidad económica de implementación de la tecnología en determinadas zonas y la comparación entre distintos estándares tecnológicos disponibles.

A diferencia de sistemas convencionales de monitoreo, este aplicativo ofrece herramientas interactivas de análisis de datos en tiempo real, lo que permitirá realizar ajustes estratégicos de manera ágil y fundamentada. Con ello, se busca mejorar la capacidad de respuesta ante desafíos técnicos y administrativos en la gestión del espectro radioeléctrico.

Asimismo, la plataforma será escalable y adaptable a nuevas necesidades, lo que garantizará su utilidad a lo largo del tiempo.

El prototipo de aplicativo informativo es limitado a las zonas cuyas simulaciones hayan sido realizadas y almacenadas en el sitio web de referencia (Xirio), garantizando así el acceso a datos confiables y actualizados. Al ser una herramienta de acceso público, cualquier persona interesada podrá consultarla, independientemente de su relación con los proveedores del servicio, promoviendo así el conocimiento y uso del espectro radioeléctrico de manera más eficiente. Adicionalmente, se priorizó la facilidad de actualización del sitio, ya que la base de datos crecerá de manera constante con nuevos puntos de interés.

Finalmente, este proyecto cobra especial importancia, ya que permite analizar las características técnicas de las simulaciones necesarias para evaluar y adoptar un estándar de radiodifusión digital en el país. Además, al contar con un sitio de dominio propio, garantiza mayor autonomía en la administración del contenido y una actualización ágil y eficiente de la plataforma, facilitando la consolidación de información relevante para el estudio y la optimización del uso del espectro radioeléctrico.

Marco Referencial y Conceptual

En esta sección fueron identificadas y revisadas las principales teorías, modelos y enfoques que se han desarrollado en relación con el tema investigado para proporcionar un enfoque teórico y conceptual. Además, se incluyen los hallazgos y conclusiones más relevantes de investigaciones previas, así como las áreas de conocimiento que están articuladas con el proyecto.

Software Para Visualizar Datos

Según Olmeda-Gómez (2014), se define la visualización de la información diciendo que es ”el uso de representaciones visuales interactivas de datos abstractos, no físicos, para ampliar el conocimiento”, esto incluye varios procesos mediante los cuales se extrae información y se

procesa con el fin de obtener subconjuntos de ellos para ser mostrados en espacios visuales-espaciales que sean interactivos con el usuario de la visualización, facilitando su comprensión, reconocimiento de patrones y formas de ordenamiento mediante patrones visuales.

Cada proyecto de visualización difiere, según:

- Si los datos están estructurados o no;
- Las técnicas empleadas para la limpieza, selección y almacenamiento de los datos.
- Los algoritmos empleados para simplificar las vistas.
- La definición de los parámetros del escenario donde se presentan.
- El propósito de uso.

La importancia de transformar datos en bruto en representaciones visuales recae en la cercanía y naturalidad de las abstracciones para el usuario, en el hecho de que facilita su análisis, estructura, funcionalidad, relaciones.

Los datos geoespaciales difieren de otros, ya que describen objetos o fenómenos con una localización específica en el mundo real. Estos mapas pueden subdividirse y superponer puntos, manchas, líneas de diferentes colores para referir lugares de extracción de datos. Estas figuras pueden tener distintas formas de acuerdo con las distintas propiedades de los datos. A su vez una visualización suele incluir representaciones interactivas, dónde se pueden incluir en un dashboard diagramas de barras, diagramas circulares y estructuras visuales de ingreso de datos para cambiar mediante interactividad, la especificidad del mapa según su interés sesgado por los datos que cada vez son mostrados.

Según Comunicación Numérica (2021), a la hora de diseñar un dashboard hay que definir la naturaleza de la abstracción de los datos que son finalmente mostrados, teniendo en cuenta cuales se tratan de métricas (datos que muestran variables en bruto, tipo numéricas que

representan cantidades físicas o entradas directas), indicadores simples o datos concluyentes tipo kpi (coeficiente de eficacia con que logra un determinado objetivo), tomando importancia entonces el análisis de la fuente, objetivos o limites críticos de los datos, calidad de las métricas, frecuencia.

A la hora de formular kpi toma importancia su clasificación por tipos como lo son numéricos, de progreso y de cambio. Hay que tener en cuenta las limitaciones de un dashboard como el no saturar al usuario con mucha información y extenderse fuera del pantallazo inmediatamente visible, yendo a lo puntal y prevaleciendo el uso del color, imágenes y figuras geométricas. Si se requiere mostrar muchos datos tipo tablas dinámicas es mejor considerar el uso de un Reporte (Rodríguez & Hernández, 2018).

Dashboard

La definición formal de dashboard presenta vaguedades, pues muchos productos pueden así denominarse. Sin embargo, algunas de las características que ponen de acuerdo con diferentes autores son: Tablero de control que presenta inclusión de mecanismos de visualización grafica como semáforos, indicadores, medidores similares a niveles y velocímetros, entre otros, de tal forma que comprenden una metáfora útil para la inteligencia empresarial. Se intenta proporcionar una visión general de algo que está sucediendo en el negocio en el momento.

Un dashboard, aunque incluye texto y gráficos, debe centrarse en gráficos, mostrar una pantalla única que presente información puntual para lograr objetivos específicos, para esto se deben definir cuidadosamente indicadores claves de rendimiento (KPI).

Los medios más habituales para la presentación de un dashboard comprenden el uso de un navegador web, lo que facilita la actualización de datos en tiempo real, aspecto útil en la mayoría de los casos.

Los dashboards pretenden monitorear información de manera rápida y sencilla, pueden ser usados para diferentes temas de forma concisa, esto se debe a que no es posible obtener muchos detalles de un primer vistazo. Esto puede implicar cambiar a otra pantalla o utilizar métodos de navegación. Un panel de control cumple su función principal al indicar con un solo vistazo que se deben tomar acciones. Además, es especialmente útil si proporciona acceso directo a la información adicional necesaria para tomar decisiones (Few Stephen, 2006).

Ciclo de Vida del Software

Existen muchos modelos y metodologías SDLC, pero consisten en distinguir una serie de fases o pasos, eso garantiza un mejor ordenamiento, interoperabilidad, escalabilidad y eficiencia del sistema, un flujo más definido de la información y por lo tanto se permite tener mejor seguridad de esta para ser trasmitida, procesada y almacenada. También permite la mitigación de diferentes riesgos durante el proceso del desarrollo y dada la obtención del producto final. A continuación, se describe el denominado ciclo de vida clásico o modelo de la cascada.

Fase de Comunicación. Se determina y documenta la necesidad que conlleva al propósito del desarrollo. Se definen los llamados requerimientos del prototipo de software.

Fase de Planeación. Comprende la estimación de recursos necesarios, objetivos, impacto, riesgos, estándares y metodologías a seguir para llevar a cabo el seguimiento general. Se realiza un cronograma de actividades a seguir.

Fase de Modelado. Fueron considerados aspectos tanto visuales como de ordenamiento lógico, analizando lo más apropiado según el flujo de la información y el rendimiento. Aquí se comprenden el diseño de la interfaz y los diferentes lenguajes de modelado UML, esto especialmente útil en proyectos grandes y cuando se dispone de un variado equipo de trabajo.

Fase de Construcción. Comprende la escritura de código como tal y las pruebas que se van realizando, así como la integración de los diferentes módulos lógicos y preparación final.

Fase de Despliegue. Se realiza del "deploy", subir a un servidor accesible para los usuarios finales en el caso del software web o la entrega en el caso de otros tipos de software. Se define la asistencia técnica que se dará y se aplican planes de retroalimentación como corrección de bugs o actualizaciones por otros motivos como el agregar nuevas funcionalidades.

Esta fase, al menos por ahora no es implementada para el presente producto al tratarse de un prototipo.

Tecnologías Utilizadas Para la Creación de Dashboards

Dada la popularidad de los dashboard web, muchas tecnologías utilizadas para su elaboración tienen directa relación con el desarrollo web, como lo son:

- HTML/CSS: HTML (HyperText Markup Language) se utiliza para definir la estructura y el contenido del dashboard, mientras que CSS (Cascading Style Sheets) se utiliza para aplicar estilos y dar formato al diseño visual.
- JavaScript: Es un lenguaje de programación ampliamente utilizado para agregar interactividad y funcionalidad dinámica a los dashboards. Frameworks como React, Angular y Vue.js son populares en el desarrollo de dashboards web.
- Otros lenguajes y tecnologías utilizadas tienen que ver directamente con el tema de la analítica de datos, como lo son:
- Python: Es un lenguaje de programación versátil que se utiliza en el desarrollo de dashboards, especialmente en aplicaciones de análisis de datos y visualización. Las bibliotecas populares como Flask, Django y Dash permiten crear dashboards interactivos y personalizados.

- R: Es un lenguaje de programación estadística ampliamente utilizado en el análisis de datos y la visualización. Los paquetes como Shiny y R Markdown permiten crear dashboards interactivos y reportes dinámicos.
- SQL: Es un lenguaje utilizado para interactuar con bases de datos y extraer datos para su visualización en dashboards. Se utiliza en combinación con otros lenguajes de programación para acceder y procesar los datos.
- Tableau: Es una herramienta popular utilizada para crear dashboards interactivos y visualizaciones de datos sin necesidad de programación. Permite conectarse a diversas fuentes de datos y ofrece una amplia gama de funcionalidades de visualización.
- Power BI: Es una plataforma de análisis de datos y creación de informes desarrollada por Microsoft. Permite a los usuarios conectarse a diversas fuentes de datos, limpiar y transformar los datos, y crear visualizaciones interactivas y paneles de control personalizados. Power Bi ofrece varias ventajas y utilidades: fácil conexión a múltiples fuentes de datos, interactividad y asistencia para la creación de objetos de visualización, dinamismo y actualización en tiempo real. Una ventaja clave de Power Bi es que se puede compartir fácilmente y tiene una funcionalidad que permite exportar el dashboard creado mediante una etiqueta iframe insertable en el maquetado html de un sitio web (Dan Clark. 2020).

Conceptos Principales de Radio Digital

La radio digital es un tipo de radiodifusión (DAB) de audio, caracterizada por la emisión de señal digital, lo cual dista mucho de la tradicional radio analógica. Dada su naturaleza digital, su codificación se caracteriza porque cada unidad de información se representa mediante valores discretos, en lugar de valores dentro de cierto rango. Una comparación de como una secuencia de valores digitales se reconstruye como audio es el funcionamiento de los discos compactos, que

tienen un principio funcional similar, pero esta vez en lugar de la reflexión o ausencia de esta al aplicar luz dada la presencia o no presencia de material reflectante, se establece una frecuencia única, generalmente muy alta, siendo lo estrictamente necesario para su funcionamiento. Sin embargo, para su estabilidad y respaldo funcional en el mundo real, DAB emplea la tecnología de modulación OFDM (Orthogonal Frecuency Division Multiplex) que permite establecer redes de frecuencia única SFN, permitiendo mantener el respaldo en otra frecuencia, cuando una determinada frecuencia se vea atenuada por el medio.

La radio digital terrestre es una tendencia tecnológica ligada a la globalización que tiene muchas ventajas con respecto a la tradicional radio basada en la transmisión de ondas análogas denominadas portadoras. Aunque el enfoque inicial latinoamericano es seguir el enfoque norteamericano para la transición comercial hacia esta tecnología (como propiedad privada y financiamiento publicitario), la realidad de la mezquindad de las políticas poco democráticas que ha querido mantener pocas regulaciones y alto control sobre la radio, no lo ha permitido (Federal Communications Commission. 2017).

En Colombia, la radiodifusión ha venido desarrollándose en respuesta al desarrollo internacional, un ejemplo de esto es la adaptación de la audiencia a la nueva forma de consumo de contenido digital. Dicho esto, se había venido adaptando un modelo hibrido, donde el estado controla canales de distribución, en ocasiones el contenido de la programación, la libre competencia publicitaria e impulsa el extendimiento de la cobertura. El planteamiento actual del modelo colombiano de radiodifusión trata de ser inclusivo, a pesar de la pujanza económica de la industria y para ello la descentralización de su control y producción donde los canales regionales fomenten sus propias industrias, audiencias y culturas (Aparicio, Calero, 2021).

Entre las ventajas tecnológicas de la radio digital sobre la radio tradicional se encuentran el mayor aprovechamiento de las bandas de frecuencia ya que múltiples señales digitales pueden ser enviadas en una misma frecuencia, hay mayor robustez e inmunidad al ruido proveniente del ambiente y otras interferencias. Otros problemas como la pérdida de calidad cuando se escucha radio en un vehículo en movimiento también son solucionados. Todas estas características hacen de la radio digital muy atrayendo y percibirse como el salto tecnológico obvio a proseguir. Por lo tanto, la llegada de SDARS (servicio de audio digital terrestre) se presenta en los EE.UU como competidora con proyección de reemplazar el tradicional servicio de radio (D. H. Layer, 2001).

Varlamov, Varlamov y Dolgopyatova (2019) señalaron que la eficiencia de una red de radiodifusión se relaciona con los costos de implementación y operación referidos a la cantidad de oyentes y área de servicio. Para estaciones de radio comerciales, se requiere gran densidad de oyentes, por lo tanto, se centra en zonas muy pobladas como zonas urbanas dónde se utilizan bandas de muy alta frecuencia (VHF). Para obtener rentabilidad es necesario tener bajo consumo energético, esto implica baja densidad de transmisores y utilización de bandas de baja frecuencia. Para saber que tan factible sea esto es necesario considerar, aparte de la densidad poblacional, las condiciones ambientales locales. Para ambientes poco poblados, es mejor considerar uso de baja frecuencia y topologías de redes asíncronas. En la latitud correspondiente a la región ártica, es recomendable la utilización de radiodifusión digital en la gama LF del estándar DRM con sistema estándar de antenas.

Estándares de Radio Digital

A la hora de implementar radio digital es muy necesario tener en cuenta criterios para elegir adecuadamente el estándar de distribución, pues a diferencia de la radio análoga, es frecuentemente utilizada con un estándar común, mientras que los estándares de radio digital son

apuestas comerciales nacionales. Los estándares tecnológicos de radio digital terrestre se clasifican en cuatro grandes grupos: el DAB europeo y sus derivaciones y mejoras; el IBOC estadounidense; el ISDB japonés y el mundial DRM.

El Eureka 147. El DAB europeo de aplicación mundial. Nació de la mano de la Unión Europea en 1986 como proyecto (de hecho, 147 es el número del experimento dentro del programa Eureka) de la mano de dos grupos de investigación. Se trata de un estándar aceptado e implantado en diversos grados en Europa, Canadá, Brunei, India, Israel, Hong Kong, China, Malasia (en su planificación del espectro se ha incluido el DAB y el DRM), Nueva Zelanda, Singapur, Sudáfrica, Taiwán, Turquía y Australia, el DAB europeo estaba coordinado por la UER (Unión Europea de Radiodifusión) y formado por radiodifusores públicos y los consorcios más destacados de la electrónica europea con la finalidad de desarrollar un estándar digital que solventara los principales problemas de difusión así como que permitiera ubicar y conducir con fuerza al medio radio en su entrada hacia la nueva era digital.

El DAB fue especialmente concebido para la recepción exterior y en movilidad ya desde los inicios del proceso de estandarización que ofrece calidad similar a la del CD (Sieber & Weck, 2004, p. 8), que representa un uso eficaz del espectro radioeléctrico, que puede utilizarse en todos los canales de transmisión (terrestre, cable y satélite) y que no necesita resintonización. El desarrollo actual del DAB, pasada más o menos una década de sus primeros pasos, es desigual y va desde su cierre en Suecia y Finlandia hasta el que se considera el país europeo de mayor desarrollo digital, Gran Bretaña, pasando por una situación de stand-by en España o el también éxito de Singapur, con un 99% de cobertura.

El DAB+, por su parte, es una mejora del DAB hecha pública a finales de 2006 y basada en los códecs de audio cuya oficialidad como estándar aprobó el ETSI en febrero de 2007. Algunas de las excelencias del DAB+ son, entre otras:

- Pueden radiarse más emisoras en un múltiplex.
- Los consumidores pueden elegir más emisoras.
- Un uso más eficiente del espectro radioeléctrico.
- Costes de transmisión más bajos.

Una lista muy parecida a la que salió en su momento para comparar el DAB con la FM.

El DMB (Digital Multimedia Broadcasting) es un estándar multimedia basado en el DAB, una innovación arquitectural, desarrollado en Corea del Sur, para distribuir radio y televisión a terminales móviles y portátiles. DMB, en referencia al audio, proporciona buena calidad con una "*bit rate"* (ratio de bits, proporción) menor, lo cual significa que caben más emisoras en el mismo ancho de banda.

Se dice que el hecho de que el DMB se base en el clásico paradigma de radiodifusión, dando libre acceso a todos sus contenidos y servicios, financiándose con publicidad, resulta una característica positiva desde el punto de vista social pero una debilidad desde el estrictamente económico, dado que puede dificultar la implantación de modelos de negocio de mayor eficiencia. Dentro de la guerra de estándares y discusiones sobre sus bondades, se establece una comparativa entre el DAB+ y el DMB al que califica de "no recomendable" para servicios de radio y añade que no soporta algunas de las características que se espera que tengan en un futuro inmediato los servicios de radio.

El IBOC es el estándar adoptado oficialmente por los Estados Unidos en octubre de 2002, cuando fue aprobado por la FCC (Federal Communications Commission) a pesar de que Eureka

147 fuera claramente superior. Se busco la favorabilidad de preservación de las estructuras económicas existentes a pesar del progreso.

El IBOC se concibió por encima de todo como una solución barata, ya que, contrariamente al DAB, que requiere que se habiliten nuevas bandas de frecuencias, trabaja sobre el rango de frecuencias existentes que están siendo utilizadas, y proteccionista, pues persigue principalmente que la estructura radiofónica norteamericana se mantenga tal y como está.

El DRM (Digital Radio Mondiale) se formó como consorcio en 1998 de la mano de radiodifusores y fabricantes de todo el mundo para desarrollar un sistema que digitalizara las bandas del espectro situadas por debajo de los 30MHz. aprovechando, en la línea del IBOC, bandas de frecuencia que ya están siendo utilizadas (ondas largas, medias y cortas); además posibilita el simulcasting (analógico y digital).

ISDB. Un mismo estándar para radio y televisión. La radio digital empezó en Japón en 2003 con el estándar ISDB de la mano del ISDB-T (Integrated Services Digital Broadcasting - Terrestrial). El estándar japonés para la digitalización tanto de la radio como de la televisión, aunque las pruebas se remontan a los '80 y el estándar técnico, a finales de los '90. Tras pugnar con el estadounidense ATSC y el europeo DVB-T, Brasil adoptó, Brasil adoptó en 2006 una optimización del ISDB, el SBTVD (Sistema Brasileiro de Televisao Digital) que trabaja con el sistema de compresión de vídeo MPEG-4 (el japonés trabaja con MPEG-2) y codifica el audio diferente. En radio, sin embargo, optó por el IBOC. En base, el ISDB pretende ofrecer un sistema universal para todo tipo de radiodifusión digital terrestre.

Un receptor portátil ISDB-TSB (Sound Broadcasting) puede recibir TV estándar y parcialmente transmisiones en alta definición, así como proporcionar sonido de calidad CD y

datos. Según Nakahara (2004), el sistema proveería a los radiodifusores en AM de servicios de audio de gran calidad; a los de FM, de servicios multimedia y a la audiencia, de recepción estable incluso en condiciones de movilidad, así como audio de calidad e información útil (Bonet, 2011).

Estado del Arte

Consiste en una revisión exhaustiva y actualizada de los avances, investigaciones y conocimientos más recientes en un campo específico de estudio. Su objetivo es identificar y analizar las investigaciones más relevantes y actuales en relación con el tema de investigación. En esta sección se busca recopilar y sintetizar los estudios más recientes publicados en revistas científicas, conferencias, tesis doctorales y otros medios académicos confiables.

Desarrollo de un Dashboard para la Toma de Decisiones Estratégicas, en Torno a los Negocios de las Facilidades de Casualty. Este trabajo fue realizado por académicos de la universidad Javeriana de Colombia, enfocándose en una investigación acerca de los negocios de las facilidades, en el área de Casualty para la empresa Múnich RE. La cual es una reaseguradora para conocidas aseguradoras de varios países. Algunos de sus clientes son Sura, seguros Bolivar, Axa Colpatria, Hidrohituango, entre otros. Se detectaron debilidades a la hora de disponer adecuadamente la información para ser utilizada de forma efectiva, dado que es recaudada y organizada de forma ineficiente y sesgada en una sola herramienta de uso común, además de no haber indicadores adecuados. Esto se refleja en pérdida de clientes potenciales, análisis erróneo de resultados y confusión a la hora de abordar las etas y estrategias a seguir.

Para abordar este problema planteado se argumenta la necesidad de un Dashboard. Pero antes, para proceder a un diagnóstico completo de las necesidades de Casuality, proceden a realizar una matriz DOFA y a establecer mejores métricas, indicadores y kpi, teniendo en cuenta

que se trate de información claramente ordinal y cuantitativa para que se adapta adecuadamente a los disantos elementos visuales del Dashboad, el cual se plantea para que ayude a la toma de decisiones y establecimiento de estrategias, en torno a estos negocios. Este primer prototipo se realiza en el programa Excel (Parra & Hernández, 2022).

Implementación de un Dashboard para el Seguimiento de un Portafolio de Proyectos. Este trabajo surge de la necesidad de implementar un sistema que permita ver la gestión y avance del portafolio de proyectos de una organización teniendo indicadores que relacionen el tiempo para lograr la toma de decisiones oportunas y responsables. Para esto se implementa un *dashboard* construido con la herramienta de inteligencia de negocios QlikView y lenguaje de tratamiento de datos R, los índices a tratar serán de desempeño de cronograma y desempeño de costos, avances, issues(eventualidades) y riesgos (Arias, 2015).

Aplicación web Creada con Shiny para el Análisis de la Variabilidad de las Materias Primas en la Fabricación de Piensos. La implementación de este dashboard surge de la necesidad de automatizar el proceso de análisis de datos de los ingredientes principales del pienso que constituye la nutrición determinante en la producción animal, específicamente ganado porcino. Esto permitirá evaluar la administración de nutrientes y aminoácidos mejorando la calidad de estos piensos a la hora de ser fabricados.

Para la realización de esta herramienta y análisis de datos se utiliza la herramienta de software R, especialmente su paquete libre *Shiny* el cual permite la creación del dashboard web con lenguaje R sin la necesidad de utilizar conocimientos de códigos HTML, CSS o JavaScript.

Luego de la implementación y uso del aplicativo se concluye que se ha logrado la consecución de objetivos como lo son automatizar el proceso de obtención de dietas adecuadas, centralizar ficheros de datos, evaluar la composición temporal del producto en cuanto a

componentes clave como la proteína y lisina, todo esto sin conocimientos técnicos por parte de un operario. Algunas limitaciones se hacen evidentes en cuanto a que parámetros se dejan fijos en la aplicación como constantes cuando en realidad podría haber la necesidad de cambiarlos (Casio, 2017).

IEC Collaboration Platform. Este documento presenta una guía de usuario sobre el funcionamiento de un dashboard implementado el cual, para su acceso, primeramente, presenta un recuadro de login. Enseguida se presentan unos cuadros que presentan los accesos a las principales opciones de la aplicación como lo son “Bussines group”, “my workspaces”, “my favourites”, “my settings”, “my documents”, ”my discussions”, “my task”. Enseguida de ingresar a una determinada opción se presentan slidebars que clasifican las opciones en estructuras tipo árbol y se clasifica la información en diferentes pestañas (IEC 2019).

Definición e Implementación de Dashboard para el Seguimiento y Control de Indicadores en Oficinas de Nueva EPS. Este proyecto plasma la implementación de un dashboard para dar seguimiento y control de indicadores gestión de las oficinas de atención al afiliado (O.A.A.) en nueva EPS para apoyar la toma de decisiones y enfocar estrategas para mejorar la atención a los afiliados. para este proyecto fue importante seleccionar los kpi adecuados para medir los rendimientos a distintos niveles de la compañía. una herramienta de control cuyo desempeño será medido es la recién implementada “di giturno 5 de control”, otra herramienta a la que se da seguimiento es un modelo de gestión recientemente implementado. las etapas de estas métricas tomadas se dan a distintos niveles jerárquicos con los cuales los procesos, tomas de decisiones y tiempos se vean beneficiados (Parra, Vélez, Vásquez, & Fandiño, 2018).

Metodología

En el actual proyecto se sigue la metodología PHVA explicada por Moyano Hernández y Villamil Sandoval (2021). A continuación, se detallan cada una de las etapas de la metodología.

Planificar

Durante la planeación se definen los objetivos y el alcance de estos, así como el cronograma, recursos, herramientas y procedimientos a seguir.

Se realiza el levantamiento de los requerimientos del prototipo software, teniendo en cuenta las necesidades del país y los estudios realizados hasta la fecha sobre radio digital. Se realizan bosquejos básicos de funcionamiento y apariencia visual, y de acuerdo con esto se define una arquitectura de software adecuada con las herramientas que permiten su desarrollo. También se define el origen de los datos y la forma en la que estos son consumidos.

Hacer

Durante la implementación se realizó una comunicación entre cliente y servidor web desde donde son consumidos los datos y se realizó la respectiva configuración. Se realizó un maquetado de la parte visual del dashboard para finalmente dotarlo del dinamismo correspondiente y el consumo de los datos para una correcta interacción con el usuario final. Por último, se unen los diferentes módulos funcionales de software para que opere el sitio web.

Verificar

Durante la etapa de verificación se comprobó que objetivos originalmente planteados hayan sido alcanzados y ante cualquier falla, se procedió a la corrección respectiva hasta alcanzar la satisfacción. Para la detección de fallas fue realizada una etapa de pruebas donde se verificaron cada uno de los casos de uso resultantes de forma manual (pruebas *end to end*).

Por último, se redactó este documento final donde se contemplan aspectos importantes abarcados en esta propuesta (como marco teórico, estado del arte, entre otros). A modo de informe se consigna el progreso del proyecto y se anexa toda la documentación técnica requerida, así como las conclusiones y posibilidades de mejoras en cuanto a escalabilidad y funcionalidad.

Actuar

Durante la etapa denominada "actuar", se realiza el despliegue final en el servicio que la universidad disponga para ello. Se recibe retroalimentación, en caso de ser necesaria, y se realiza un análisis sobre su aporte a la investigación del proyecto global de radiodifusión digital, así como una exposición del trabajo realizado.

No aplica la realización de esta etapa en el presente trabajo escrito al tratarse de un prototipo de producto.

Requerimientos de Software

Según Roger S. Pressman. (2010), los requerimientos de software consisten en una serie de objetivos los cuales deben obedecer a distintas consideraciones lógicas esenciales y físicas (de implementación). El problema esencial debe ser enfrentado eficazmente mediante la división del problema principal en problemas más específicos para llevar a cabo la representación de un diseño del software a elaborar, esto se denomina modelo de análisis. Para llevar a cabo este objetivo se usan graficas siempre que sea posible. Análisis es lo que ocurre cuando se obtienen los requerimientos.

Durante el modelado de requerimientos la atención debe centrarse más en el *qué* que en el *cómo* centrándose en un enfoque iterativo para ir llenando posibles vacíos ya que es posible que el cliente no este seguro con precisión de varios aspectos del sistema. La tarea del analista debe ser modelar en base a lo conocido y acercarse a un modelo base.

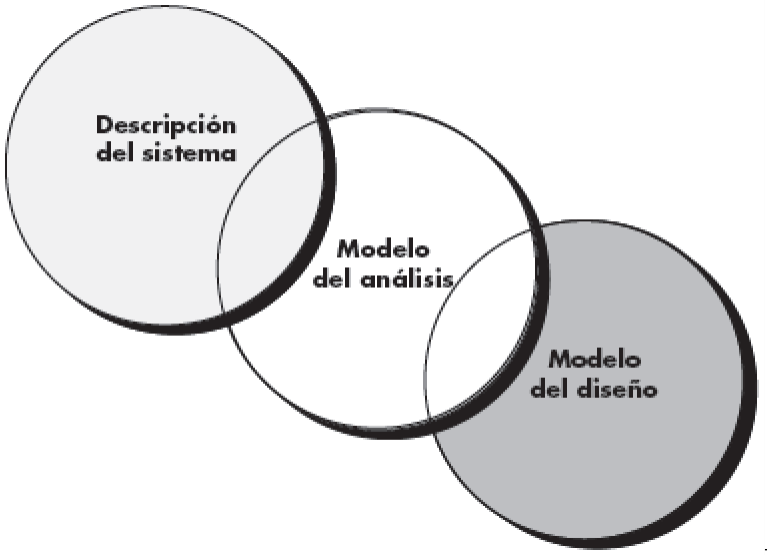
Para acercarse a los objetivos del modelo requerimientos se deben responder tres preguntas principales:

- Describir qué quiere el cliente.
- Establecer una base para la creación de un diseño de software.
- Definir un conjunto de requerimientos que puedan validarse una vez construido el software.

El modelo de análisis es un puente entre la descripción del nivel del sistema centrado en lo general o las funcionalidades del software. En la figura se describe el modelo de requerimientos.

Figura 1

Modelo de Requerimientos



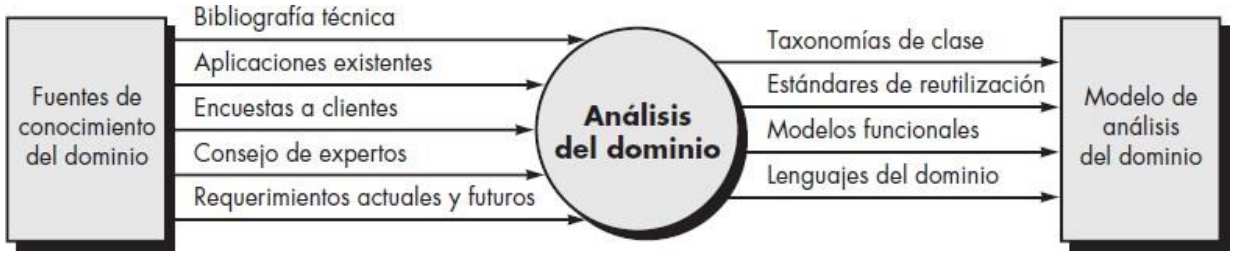
Nota Roger S. Pressman. (2010). (p.128)

Análisis del Dominio

En esta sección se describen los proyectos con necesidades y soluciones similares para reutilizar patrones e interfaces. En la figura 2 se puede observar la forma cómo se realiza el análisis de dominio, con sus respectivas entradas y salidas.

Figura 2

Figura 2 Entradas y Salidas Para el Análisis del Dominio



Nota. Roger S. Pressman. (2010). (p.129)

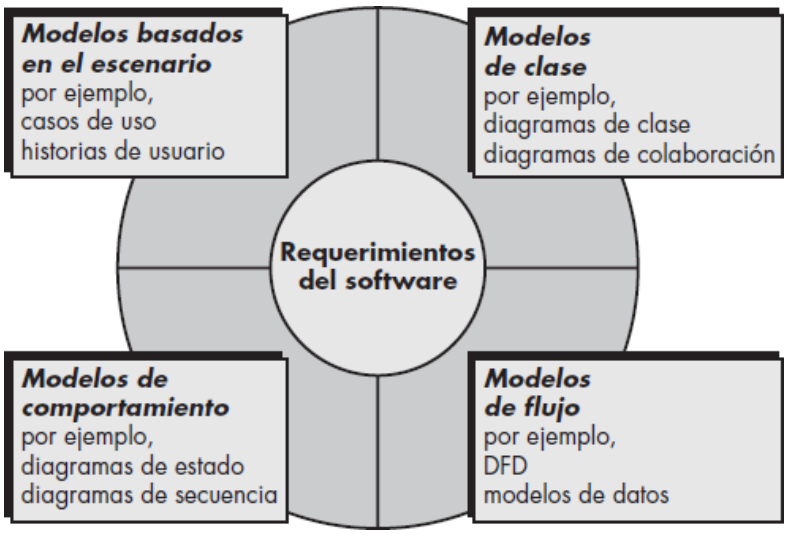
En este caso la fuente de conocimiento del dominio es la bibliografía técnica, las aplicaciones mencionadas en el apartado de Estado de Arte de este documento en las cuales se basa la propuesta de maquetación del Dashboard, el proyecto de investigación de la convocatoria 908 de Minciencias. Así mismo, las salidas son el lenguaje del dominio y el modelo funcional del desarrollo software.

Modelo de Análisis

Se establece los diferentes tipos de modelos para la documentación de los requerimientos software. En este caso se usa el modelo de análisis basado en el escenario, debido a que principalmente se tendrá en cuenta dos casos de uso para el sitio web: Visitante y Administrador, también se tendrá en cuenta el modelo de flujo para optimizar la trayectoria de los datos. Esta decisión se basa en lo explicado en el cuadrante superior izquierdo de la figura 3 que se muestra a continuación.

Figura 3

Elementos del Modelo de Análisis



Nota. Roger S. Pressman. (2010). (p.131)

En este caso se hará énfasis en el modelo de comportamiento próximamente (capitulo 3.6) describiendo el flujo de datos principal, el rol básico de visitante y el rol de administrador mediante diagramas de flujo en bloques.

Propósito del Prototipo de Software

El propósito del prototipo software de este proyecto es el desarrollo de una Aplicación web que exponga los datos en forma de dashboard.

La entrada de información consiste en archivos en formato KMZ cuyo propósito es la exposición de datos de frecuencia de señal vinculados a diferentes posiciones geográficas.

Estos datos son generados por las simulaciones realizadas por personal de la UNAD para apoyar el proyecto de investigación titulado “Recomendaciones regulatorias para un esquema de transmisión digital de radiodifusión sonora de acuerdo con las necesidades y recursos físicos de la radio en Colombia”, donde se relacionan inicialmente 4 ciudades del país.

Cada cierto tiempo son suministrados archivos en formato .kmz cuyo propósito es la exposición de datos de frecuencia de señal vinculados a diferentes posiciones geográficas.

Estos datos son importados a Microsoft Excel y procesados con el fin de realizar el respectivo dashboard que incluye un mapa, esto mediante su importación a power Bi. Posteriormente es generado un tag el cual se inserta en DOM el sitio web.

El sitio web tiene la posibilidad de registrar usuarios que contaran con la capacidad de insertar desde un pequeño formulario, datos nuevos para nuevos dashboard en caso de ir agregando más archivos en un futuro.

Alcance del producto / Software

- Propósito u objetivo general.

Exposición y estructuración de datos de simulaciones de radio frecuencia en posiciones geográficas mediante Dashboard embebido en un sitio web.

Almacenamiento y exposición de dashbaords de diferentes localizaciones en un solo sitio w
eb.

- Beneficios que brinda al área de negocio y organización.

La estructuración de datos generada mediante este Dashboard online permitirá un rápido y fácil acceso a la información. Visualización ordenada que permitirá identificar patrones y realizar análisis que permitan sacar conclusiones relacionadas con los resultados obtenidos que para la investigación. La obtención de estos datos será facilitada en un solo sitio centralizado.

- Objetivos y metas.

La razón de ser de un dashboard consiste básicamente en ser una herramienta que permita un fácil entendimiento mediante la interpretación visual de los datos mediante diagramas fácilmente interpretables. Mediante la implementación de este dashboard en concreto en un sitio web, se ayuda a la investigación principal al disponer datos que ayuden en la toma de decisiones en el marco de los estudios que se llevan a cabo para la implementación de la radio digital en Colombia.

Esta investigación a su vez tiene propósitos sociales en cuanto al potencial económico que se desata al avanzar en el proceso de implementación de la tecnología de la radio digital, trayendo progreso mediante empleo, diversificación y ampliación de la comunicación, entre otros.

Diseño del Dashboard y Aplicativo Web

Este apartado describe lo que se tiene en cuenta para el diseño del software y corresponde a la etapa de planificación necesaria para contextualizar los datos, metodologías, tecnologías y otros conceptos clave necesarios como guía y fuente anticipados para la implementación.

Análisis y Visualización de Datos

Según Maté, Pérez y Calvo (2019), el análisis de datos sigue una serie de etapas que permiten transformar información bruta en conocimiento útil para la toma de decisiones según la naturaleza de los datos y tu contexto de uso. Durante ese proyecto se ha llevado a cabo algunas de las etapas del análisis de datos allí descritas como lo son: Definición de problema: Identificar los objetivos del análisis de datos y preguntas clave. Esto se encuentra sustentado según la justificación y marco teórico del presente trabajo de grado.

Recopilación de Datos

De esto se encargó la universidad mediante la implementación de las simulaciones con el software *xirio*. Dichos datos fueron suministrados en un documento de Excel titulado *estadísticas plantilla Medellin.xls.*

Limpieza y Transformación

Sabiendo de antemano que el posterior análisis será llevado a cabo en Power Bi, cuya importación de datos se facilita cuando se presenta una tabla estrictamente definida (columnas y registros) y cuyos datos de posición geográfica tienen un formato diferente.

Transformación de Formato. Se refiere a la transformación de la forma en que los datos que se presentan en el archivo *plantilla Medellin.xls* en cuento a disposición y semántica propia de los datos.

Primeramente, se lleva el estado mostrado en la Figura 4 hacia el resultado mostrado en la Figura 5. Esto se logra mediante los siguiente puntos clave:

- Tras observarse la disposición en la fila 1 de celdas combinadas que desempeñan como título conceptual de cada columna de potencia que hay debajo. Esta información útil se concatena al título propiamente dicho de cada potencia.
- Eliminación de las celdas combinadas que desempeñan como título descriptivo.
- Eliminación de columnas repetidas: Se verifica que la información geográfica es repetida para cada señal (potencia o ruido) y por lo tanto solo se conservan bajo A y B, el resto de las columnas Latitud y Longitud son eliminadas.
- Cambio de los nombres de las columnas que quedan para describir los datos según las referencias de las celdas de título eliminadas.

Figura 4

Muestra Original de la Agrupación de Columnas de la Plantilla

	A	B	C
1	Cob Mult. IBOC AM Hibrido		
2	Latitud	Longitud	Potencia

Nota. Archivo *estadísticas plantilla Medellin.xls*

Figura 5

Figura 5 Muestra del Resultado Después de la Transformación de Formato

	A	B	C
1	Latitud	Longitud	Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)

Nota. Archivo *estadísticas plantilla Medellin.xls*

En seguida, de gran importancia es la transformación de los datos de posición geográfica suministrados en formato Grados, Minutos y Segundos (DMS) a formato en columnas separadas

longitud y latitud estándar WGS84, sistema de coordenadas GPS que utiliza grados decimales y conversión de oeste y sur a negativos. La conversón obedece a la siguiente fórmula.

$$DD = \text{Grados} + \text{Minutos}/60 + \text{Segundos}/3600$$

Otra limpieza y transformación de datos necesaria fue el cambio de la palabra "Infinity" presentada en algunos valores de SNR Fm debido a que esto representa un error en el posterior análisis de datos al requerirse solo entradas numéricas. A conveniencia se cambian estas cadenas de texto por el valor 10000000, un numero significativamente mayor al resto de datos de su conjunto qué, para efectos prácticos al ser una cantidad enorme, puede ser interpretado de la forma deseada.

Para automatizar este proceso de transformación descrito, teniendo en cuenta que se proporcionaran este tipo de plantillas con datos para futuros datos de otros acotes geográficos, se implementó un código Visual Basic. Este código y su respectiva documentación mediante comentarios se muestran en el Anexo A.

A continuación, detallo la utilización de dicho código en el archivo:

- Desplegar panel de VB mediante el atajo alt+11, importar archivo: Modulo2.bs.
- Escoger modulo1, macro1, correr la macro mediante F5.
- Guardar cambios y cerrar.

Exploración, Transformación y ModeladodDe Datos

Para esto se realizan algunos cálculos estadísticos y valiéndose de técnicas de segmentación entre otros, preparar visualizaciones. Esto se realiza en Power Bi y en resumidas cuentas representa el proceso de diseño del dashboard, unidad central de este proyecto. Primero se accede a los datos mediante los siguientes pasos: o Informe en blanco, importar datos de Excel, Adjuntar archivo, seleccionar vista preliminar, cargar. El modelado de datos se lleva a

cabo mediante las diferente medidas y columnas calculadas del lenguaje DAX, preparadas para ser dispuestas en los diferentes elementos visuales (ver Anexo B).

Visualización de datos

A continuación, se listan y describen los elementos visuales y funcionales incluidos durante la construcción del Dashboard en Power Bi.

Mapa. Este elemento visual se sitúa en la parte derecha del dashboard, se configura añadiendo las columnas Latitud y Longitud en los respectivos campos junto, junto con una columna calculada llamada *actual_*, esto permite dar dinamismo al funcionamiento del mapa pues *actual_* es básicamente una columna cuyos valores cambian según la medida escogida en el segmentador.

Segmentador de Opciones. Este elemento es configurado como oculto en este dashboard, originalmente presenta la funcionalidad de mostrar una lista de columnas las cuales el usuario debería seleccionar directamente para ser analizadas, sin embargo, en este dashboard se le da gobierno mediante un navegador de marcadores que posteriormente es descrito. Para que esta funcionalidad original no interfiera ni haya confusiones, es que se ha configurado como oculto. Este segmentador agrupa diferentes medidas utilizadas en el dashboard y que se ajustan a los filtros de las diferentes visualizaciones para cambiar según la información relacionada según la selección. La medida mencionada anteriormente, *actual_*, encabeza esta lista de medidas, luego están: *población_*: cumple la función de conectarse con un segmentador de opciones que luego de terminar su configuración se deja oculto, *desviación_e_actual_*, *max_acutal_*, *min_actual*, *prom_actual*, *varianza_actual_* las cuales respectivamente calculan desviación típica, valor máximo, y varianza, siempre comparando el valor presentado en actual_ con su respectiva columna y, finalmente promedio de la respectiva columna de datos: *prom_actual_*.

Este elemento también cuenta con un segmentador de campos y es básicamente una tabla en la que se relacionan los nombres de los campos entre los cuales '*switchea'* el análisis, sus referencias a la tabla y el orden (números del 0 al 5).

Tarjeta. Esta se ubica en la parte superior izquierda y muestra de manera interactiva medidas significativas sobre la medida de la tabla original seleccionada por el segmentador. Tales medidas son: valor máximo, valor mínimo, promedio, varianza y desviación típica, además se presenta una medida llamada Valor seleccionado, este por defecto permanece indefinido, pero toma valor al hacer click sobre alguna de las burbujas del mapa. Seleccionar un valor del mapa permite también la interactividad en posteriores visualizadores seleccionados.

Gráficos de Columnas Agrupadas. Se utilizan dos de estos elementos para hacer comparaciones separadas de Promedios de potencias y Promedios de SNR (relación señal ruido) ya que es lo adecuado dada su diferente dimensionalidad. La interactividad de estos gráficos responde al bocadillo que el usuario seleccione en el segmentador. Además del promedio, las columnas de estos gráficos presentan opcionalmente un estado en el que cada columna presenta además del valor de promedio mencionado, un valor resaltado, correspondiente al valor especifico correspondiente al registro de la última coordenada seleccionada en el mapa. A continuación, se describen estos gráficos:

- Promedio SNR [dB]: Se ubica a la izquierda y en su orden presenta los valores de relación señal ruido así: Columna color cian, Am Hibrido, Columna azul, FM Hibrido, Columna naranja, DAB.
- Promedio Potencia [dBu]: Se ubica a la derecha y en su orden presenta los valores de potencia de las señales en el mismo orden que el grafico anterior.

Histograma. En este contexto, el histograma es un gráfico de barras que representa la distribución de una variable numérica. Cada barra corresponde a un intervalo de valores (grupos) y su altura indica la frecuencia con la que los datos caen dentro de ese rango. En otras palabras, muestra la densidad de datos en diferentes segmentos de la variable analizada, permitiendo visualizar patrones de distribución y tendencias.

Si bien es lógico que, entre más datos, más grupos, el número de columnas (grupos) no fue escogido con este escaso criterio. El número de grupos escogido se basó en la afirmación citada a continuación: No existen reglas fijas para determinar el número idóneo de intervalos, hasta el punto de que a veces se hacen varias pruebas hasta conseguir resaltar las características del fenómeno. Cuando no existen otras indicaciones, un valor comúnmente aceptado es un numero próximo a raíz cuadrada de N, siendo N el número total de observaciones (Hernández Martín, 2012, p. 24). En este caso, el número de observaciones (registros) es de 100, por lo tanto, se han configurado 10 columnas (grupos) para cada histograma de cada una de las señales de la tabla original.

En el dashboard existen 6 de estos histogramas, pero de cara al usuario se percibe uno a la vez, dando la percepción de ser uno solo que cambia los datos que analiza dinámicamente según el navegador de marcadores que escoge la señal cuyos datos serán presentados en el informe. Además, si se selecciona un lugar específico del mapa, en el histograma también se muestra mediante una barra resaltada de un color cian más intenso al cian de las barras que fue configurado por defecto.

Navegador de Marcadores. Permite la interacción con el usuario para que mediante el uso de actual_ cuyos valores cambian según el bocadillo seleccionado. Se ubica en la parte superior del mapa es la medida presentada según el tamaño de burbuja en el mapa. Estas 6

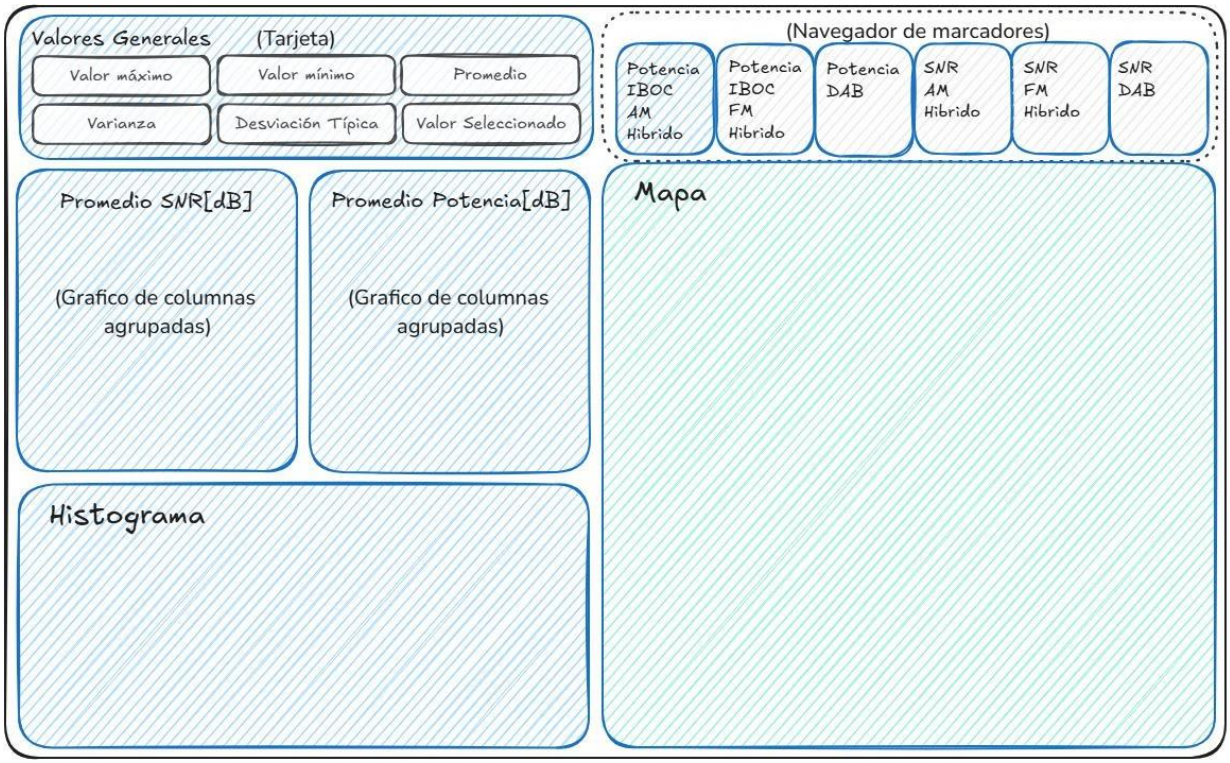
medidas y se corresponden consecutivamente con las señales de la tabla original, Potencia IBOC am Hibrido, Potencia IBOC FM Hibrido, Potencia DAB, SNR AM Hibrido, SNR FM Hibrido, SNR DAB.

Este elemento habilita la opción de configurar una lista de botones de selección que permite filtrar visualizaciones entre “activas” a “inactivas” y elegir entre diferentes escenarios de análisis. En este caso se configuran 6 capas, las cuales permitieron visualizar un histograma por cada capa, el cual presenta un análisis de frecuencia para cada una de las 6 señales de la tabla original, debido a que no fue posible vincular directamente el segmentador de opciones con el histograma para cambiar dinámicamente sus valores analizados. En lugar de esto, se superpusieron los 6 histogramas (uno para cada señal) y en cada capa de visualizaciones se cambió cual de estos estaría visible respectivamente.

A continuación, se muestra una imagen del diseño del Dashboard (ver Figura 6).

Figura 6

Diseño Previo del Dashboard



Nota. Bosquejo de mi autoría en *excalidraw.com*

Entorno Operativo del Aplicativo web

El aplicativo web necesita conectividad constante para su correcto funcionamiento. Se desarrolla con lenguajes de programación HTML, CSS, Javascript para visualización de la interfaz de usuario (*Fronted*) que se carga el visitar el sitio web con la url del sitio (la cual en modo de desarrollo es *http://localhost*) y cuyos cambios se renderizan dinámicamente en una única página web (SPA). La tecnología utilizada para el desarrollo de dicho *Fronted* es el *Framework* Vue.js en su versión 3.5.10.

Para controlar los permisos necesarios para que el administrador se le permita acceder al formulario cargar un nuevo dashboard, y guardar persistentemente los existentes, se utiliza la

base de datos MySQL. Para generar los endpoint necesarios para conectar el *fronted* con dicha base de datos y toda la lógica adicional necesaria para mantener la integridad y validez de los datos, administración de la sesión, etc se utiliza un API REST (Backend) realizado con el *Framework* Laravel 11 el cual es utiliza el lenguaje de programación php para estos propósitos.

Requerimientos Funcionales

Los requerimientos funcionales describen las actividades que el sistema debe realizar bajo ciertas condiciones. A continuación, se presentan los requerimientos funcionales organizados por funcionalidad del sistema.

Visualización del Panel Público. El sitio web muestra por defecto, apenas se accede, un panel público donde se visualizarán los cuadros de información de los dashboards creados que estén habilitados como públicos. Al inicio, el panel público estará vacío puesto que al estrenar el sitio la base de datos estará vacía.

Prioridad: Alta

Acciones iniciadoras y comportamiento esperado:

Al acceder al dominio principal, el usuario visualiza el panel público.

Si existen dashboards públicos, estos se mostrarán en la interfaz.

En la esquina superior derecha se encuentra un menú hamburguesa con opciones:

Iniciar sesión

Crear registro de administrador

Cerrar sesión

Información

Requerimientos Funcionales:

REQ-1: El sistema debe presentar en el panel público únicamente los cuadros de dashboards marcados como públicos.

REQ-2: El ‘menú hamburguesa’ debe desplegarse correctamente al hacer clic sobre él.

REQ-3: Si no hay dashboards públicos, el panel debe mostrarse vacío sin errores.

REQ-4: La opción Información debe estar siempre disponible para todos los usuarios.

REQ-5: No será posible cerrar sesión si no hay sesión iniciada.

Gestión de Roles de Usuario. Descripción: El sistema maneja dos roles principales: Administrador y Visitante.

Prioridad: Alta.

Acciones iniciadoras y comportamiento esperado:

Los visitantes podrán acceder al panel público, pero no podrán utilizar a otras funcionalidades del sistema (iniciar sesión, registrar, cerrar sesión), por ello se espera el registro del primer administrador tan pronto sea desplegado en sistema.

Un usuario administrador podrá realizar gestiones dentro del sistema una vez haya iniciado sesión.

Requerimientos Funcionales:

REQ-6: Un usuario visitante no debe tener acceso a opciones administrativas.

REQ-7: Un usuario visitante solo podrá acceder a la opción "Información" dentro del menú.

REQ-8: Un usuario administrador podrá acceder al panel de administración después de iniciar sesión correctamente.

Registro de Administrador

Descripción:

El sistema debe permitir registrar un administrador cuando la base de datos esté vacía. Este administrador será Administrador Principal por defecto.

Prioridad: Alta.

Acciones iniciadoras y comportamiento esperado:

Si no hay administradores registrados, el usuario podrá crear el primer administrador.

Este administrador tendrá privilegios de Administrador Principal.

Una vez registrado, se habilitarán las opciones de inicio de sesión.

Requerimientos Funcionales:

REQ-8: Si la base de datos está vacía, debe permitir el registro del primer administrador.

REQ-9: El primer administrador registrado debe ser Administrador Principal automáticamente.

REQ-10: A excepción del primer administrador registrado, no se debe permitir registrar otro sin tener sus permisos correspondientes.

Inicio de Sesión de Administrador. Descripción: Un administrador debe poder iniciar sesión para acceder al Panel de Administración.

Prioridad: Alta.

Acciones iniciadoras y comportamiento esperado:

- Si no hay administradores registrados, no se podrá iniciar sesión y se mostrará un mensaje de error.
- Si las credenciales ingresadas son incorrectas, el sistema debe mostrar un mensaje de error.
- Si las credenciales son correctas, el usuario accederá al Panel de Administración.

- El usuario administrador tiene la opción de navegar hacia el panel público libremente y hace uso de la función iniciar sesión, pero esta se encuentra activa, se dirigirá al panel de administración

Requerimientos Funcionales:

REQ-11: Si sistema debe validar si hay sesión activa antes de iniciar sesión, el tal caso le dirigirá la directamente al panel de administración.

REQ-12: El sistema debe validar la correspondencia de las credenciales antes de permitir el inicio de sesión.

REQ-13: Si el usuario ingresa credenciales incorrectas, el sistema debe mostrar un mensaje de error.

REQ-14: Si el usuario ingresa credenciales correctas, se debe redirigir al Panel de Administración.

Gestión de Dashboards. Descripción: Un administrador podrá registrar y gestionar dashboards, incluyendo su información adicional.

Prioridad: Media.

Acciones iniciadoras y comportamiento esperado:

- Un administrador podrá crear dashboards y asignarles información.
- Un administrador podrá modificar los cuadros de información de dashboards existentes.
- Un administrador podrá eliminar dashboards existentes.
- Un administrador podrá definir si los cuadros de información de dashboards serán públicos o privados.

Requerimientos Funcionales:

REQ-15: El sistema debe permitir registrar un nuevo cuadro de información de dashboard con su información.

REQ-16: Un administrador podrá modificar los cuadros de información de dashboards.

REQ-17: Un administrador podrá eliminar cuadros de información de dashboards existentes.

REQ-18: Se debe poder marcar cuadros de información de dashboards como públicos o privados.

Gestión de Administrador Principal. Descripción: Un Administrador Principal podrá crear nuevos administradores y gestionar sus permisos.

Prioridad: Alta.

Acciones iniciadoras y comportamiento esperado:

- Un Administrador Principal podrá autorizar el registro de nuevos administradores mediante número de documento y correo electrónico.
- El Administrador Principal podrá definir si el nuevo usuario será Administrador Principal o no.
- Un Administrador Principal podrá ver y eliminar autorizaciones creadas.

Requerimientos Funcionales:

REQ-18: El sistema debe permitir a un Administrador Principal registrar nuevos administradores.

REQ-19: El Administrador Principal podrá definir si el nuevo usuario tendrá privilegios de Administrador Principal o no.

REQ-20: Un Administrador Principal podrá ver y eliminar autorizaciones 9.7.

Cierre de Sesión. Descripción:

El sistema debe permitir que un administrador cierre sesión en cualquier momento de la sesión. La sesión, por defecto, dura una hora.

Prioridad: Media.

Acciones iniciadoras y comportamiento esperado:

- Al hacer clic en la opción Cerrar sesión, el usuario será redirigido al panel público.
- La sesión del usuario debe cerrarse correctamente y no debe permitir acceso a funciones administrativas hasta que vuelva a iniciar sesión.

Requerimientos Funcionales:

REQ-21: El sistema debe permitir a un administrador cerrar sesión en cualquier momento siempre que la sesión se encuentre activa.

REQ-22: Después de cerrar sesión, el usuario debe ser redirigido al panel público.

REQ-23: Un usuario sin sesión activa no podrá acceder al panel de administración.

REQ-24: Después de una hora de sesión activa, esta debe cerrarse automáticamente.

Gestión de Cuenta de Administrador. Descripción: Un administrador podrá editar sus propios datos y eliminar su cuenta si lo desea.

Prioridad: Alta.

Acciones iniciadoras y comportamiento esperado:

- Un administrador podrá modificar su información personal, como nombre, correo electrónico, documento y nombre de usuario.
- Un administrador podrá cambiar su contraseña.

- Un administrador podrá eliminar su cuenta, pero si es el único Administrador principal registrado, se le impedirá eliminar su cuenta hasta que otro administrador asuma ese rol.
- Si solo queda un usuario administrador, este asumirá como administrador principal.
- Requerimientos Funcionales:

REQ-25: Un administrador podrá acceder a las opciones del panel de administración haciendo click en el menú hamburguesa.

REQ-26: Un administrador podrá acceder a su perfil y modificar sus datos personales.

REQ-27: El sistema debe validar que el correo electrónico modificado no esté registrado previamente.

REQ-28: Un administrador podrá cambiar su contraseña mediante un formulario de actualización.

REQ-29: Un administrador podrá eliminar su cuenta desde la opción correspondiente en el panel de administración, debe introducir su contraseña.

REQ-30: Si un administrador es el único Administrador Principal, el sistema no le permitirá eliminar su cuenta.

REQ-31: Al eliminar una cuenta, el sistema debe verificar si queda solo una cuenta restante, esta obtendrá privilegios de administrador principal si no los tiene.

REQ-32: Una vez eliminada la cuenta, la sesión debe cerrarse automáticamente y el usuario debe ser redirigido al panel público.

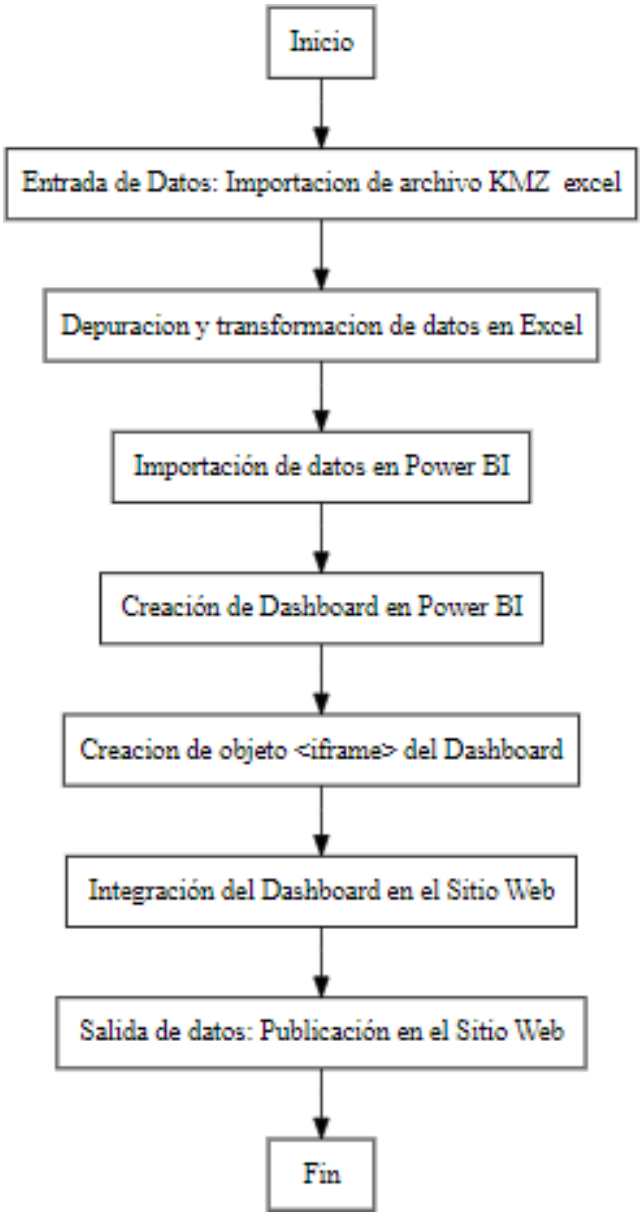
REQ-33: Una vez cambiada la contraseña de administrador, la sesión debe cerrarse automáticamente y el usuario debe ser redirigido al panel público.

Flujo de Datos Principal

Describiendo el flujo principal de los datos que van a ser expuestos y que son el objetivo de la investigación, se presenta una secuencia de pasos que comienzan cuando el usuario encargado de la administración del sitio, llamado Administrador, recibe el archivo KMZ, lo importa a Excel, realiza el respectivo análisis, depuración y transformación de datos y los lleva a Power Bi para generar el dashboar y enseguida genera el objeto html '*iframe'* para ser insertado en el sitio web que se encargara de exponer el dashboard de manera pública. Véase la Figura 7.

Figura 7

Diagrama de Flujo de Datos Principal



Nota. Autoría propia

Funcionalidad de actualización

Se hace evidente cuando se observa que no se trata de un sitio estático cuyo aspecto e información no cambia desde su implementación y despliegue.

En cambio, es necesario que cada que se reciban datos de nuevas pruebas de simulaciones de señal de radio digital ligada a información geográfica, se genere un nuevo dashboard. Es necesario que un analista se encargue de la importación de los archivos KMZ, la analítica de los datos, la generación del dashboard y la generación del objeto html necesario para su publicación en el sitio web, como lo muestra el anterior diagrama de flujo. Por lo tanto, desde este punto de vista, el sitio tiene sentido de existencia cuando se le está alimentando con urls de dashboards.

Clases y Características de Usuarios

Describen los elementos y características de los roles de los usuarios que existen en la aplicación de software.

Caso de uso de Usuario Administrador

Se hace necesaria la implementación de este usuario para lograr optimizar la labor de carga de información cada vez que se necesite disponer la información de dashboards nuevos.

Si esto no se realizara de esta manera, sería necesario editar todas las veces el código fuente del sitio web y volver a hacer despliegue de la aplicación.

Sin embargo, esto hace necesario el uso de un sistema backend que comprende el consumo y escritura de una base de datos. Este se encarga de guardar las credenciales del usuario administrador para gestionar su caso de uso incluido el cómo guardar la información de los diferentes dashboards generados.

Para tales efectos se utiliza el guardado de datos en la base de datos relacional MySQL, sin embargo, no se utilizó estructura relacional como tal (uso de llaves foráneas, consultas estructuradas, normalización) pues no fue necesario dada la simplicidad del caso.

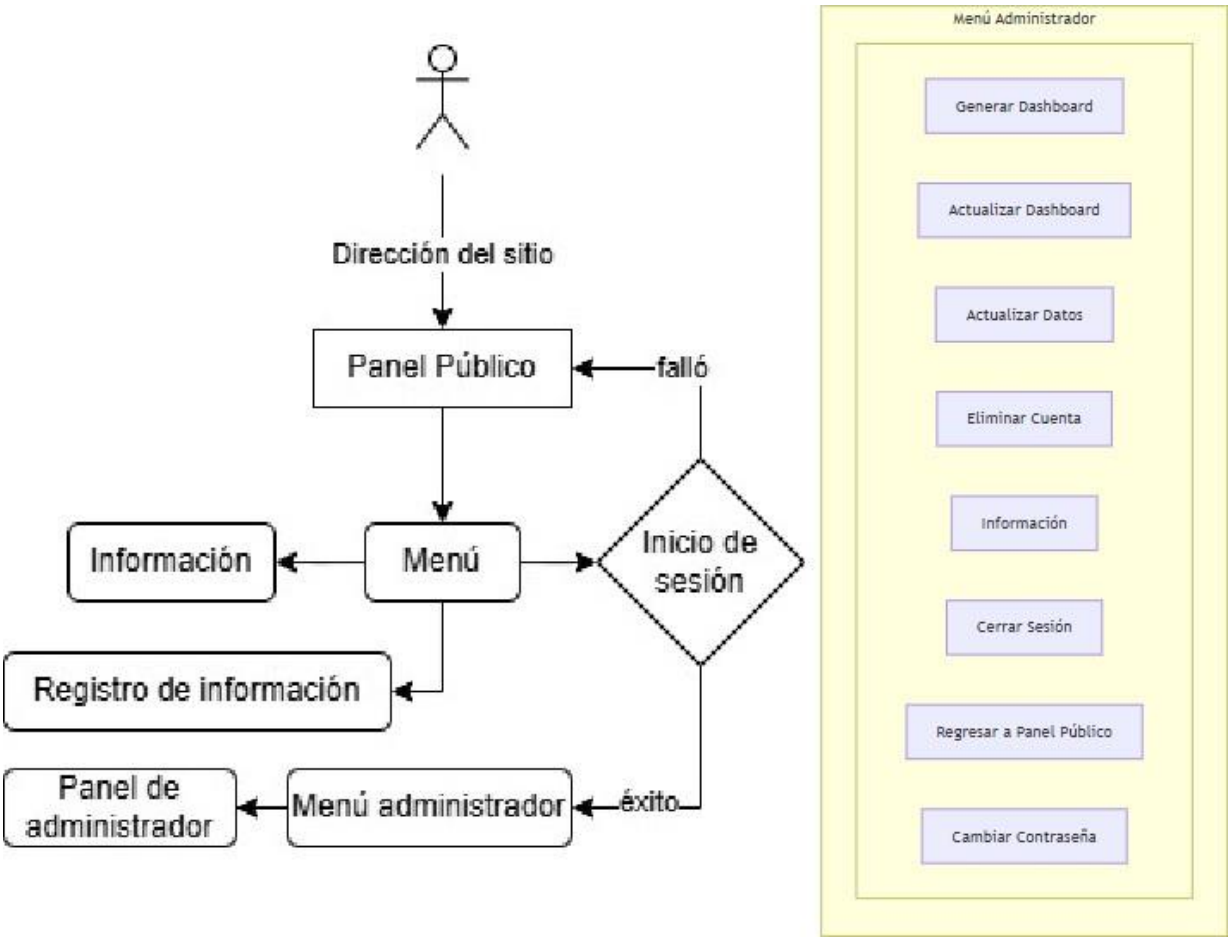
La recopilación de información se lleva a cabo en el *fronted* por medio de formularios, los cuales se relacionan con algunos de los bloques mencionados en la figura 8, estos datos son formateados a formato *json*, un conocido formato de intercambio de información el cuales recibido por los endpoint backend y transformado mediante métodos de clase llamados controladores en consultas y escrituras SQL. Para ello, laravel utiliza Eloquent, un ORM que permite relacionar entidades relacionales con instancias de clases de POO y mediante sencillos métodos de objetos realizar las respectivas labores de lectura, escritura, eliminación y actualización de registros en las respectivas entidades de la base de datos.

A continuación, se describen las diferentes etapas de Figura 8 y se describirán las clases y atributos que son utilizados.

Panel Público. Su principal función es mostrar cuadros con dashboards e información adicional. Cada unidad de estas estructuras de datos se guarda en una entidad(tabla) llamada *map* de la base de datos, para mostrarla se abordará directamente la consulta sql *'DESCRIBE map'* en *'php MyAdmin'* la cual arroja la respectiva una descripción. Los resultados de esta instrucción se muestran en el ejemplo de la Figura 9.

Figura 8

Diagrama De Flujo de Caso de Uso de Administrador



Nota. Autoría propia

Figura 9

Entidad 'Map' de la Base de Datos

```
DESCRIBE map;
```

[Editar en línea] [Editar] [Crear código PHP]

Opciones extra

Field	Type	Null	Key	Default	Extra
id	bigint(20) unsigned	NO	PRI	*NULL*	auto_increment
post	tinyint(1)	NO		*NULL*	
title	varchar(255)	NO		*NULL*	
description	varchar(255)	NO		*NULL*	
author	varchar(255)	NO		*NULL*	
urlDashboard	varchar(255)	NO		*NULL*	
place	varchar(255)	NO		*NULL*	
created_at	timestamp	YES		*NULL*	
updated_at	timestamp	YES		*NULL*	

Nota. Archivo de mi autoría base de datos *proy*, tabla *map*

A continuación, se describen los elementos de la entidad *map*:

- *Id:* clave autoincremental utilizada por el sistema para garantizar una clave única que identifique cada registro, es un número único que se asigna al momento de crear un registro.
- *post:* este campo es utilizado en el sistema para determinar si este *map* será visible en el panel público o no. Por lo tanto, en todo el sistema se trata como una variable tipo booleana. MySQL no admite datos booleanos y por lo tanto esto es transformado por laravel, para ser guardado y leído en la base de datos así: *true: 01, false: 0.*
- *description:* Descripción sobre dicho dashboard que puede incluir toda la información adicional que se considere pertinente, se admiten hasta 2000 caracteres.
- *title:* Titulo asociado al dashboard, es validado por el fronted con la restricción de admitir letras y espacios, su longitud es de 5 a 30 caracteres.

- *author*. En este campo se guarda el nombre del administrador que creo el registro en esta tabla de la base de datos. En este caso su nombre no se obtiene de formulario, sino que es obtenido de la cookie de la sesión actual teniendo en cuenta que la acción de creación de un registro map solo es posible con la sesión iniciada.
- *urlDashboard*. Este campo corresponde con la URL mencionada en la página 42 que el administrador debe llenar en este campo a la hora de cargar un map y es de gran importancia porque constituye el dashboard propiamente dicho.
- *place*: Este campo pretende registrar el nombre de la ciudad o zona correspondiente a la región geográfica donde se levantaron los datos del actual dashboard.
- *created_at:* este campo es creado automáticamente por laravel y es útil para registrar la fecha en la que el registro es creado.
- *updated_at*: laravel guarda en este campo la fecha de la última vez que se realiza una operación de escritura sobre el actual registro.

Menú. Se trata de un menú que originalmente se presenta con un icono menú hamburguesa flotante (Figura 10) el cual responde al click desplegando opciones dependiendo el lugar de la aplicación web en la cual está siendo desplegado, aclarando que, desde el punto de vista de la codificación, es el mismo componente cuando se despliega el menú administrador.

Figura 10

Icono Menú Hamburguesa

Nota. Icono de Google fonts

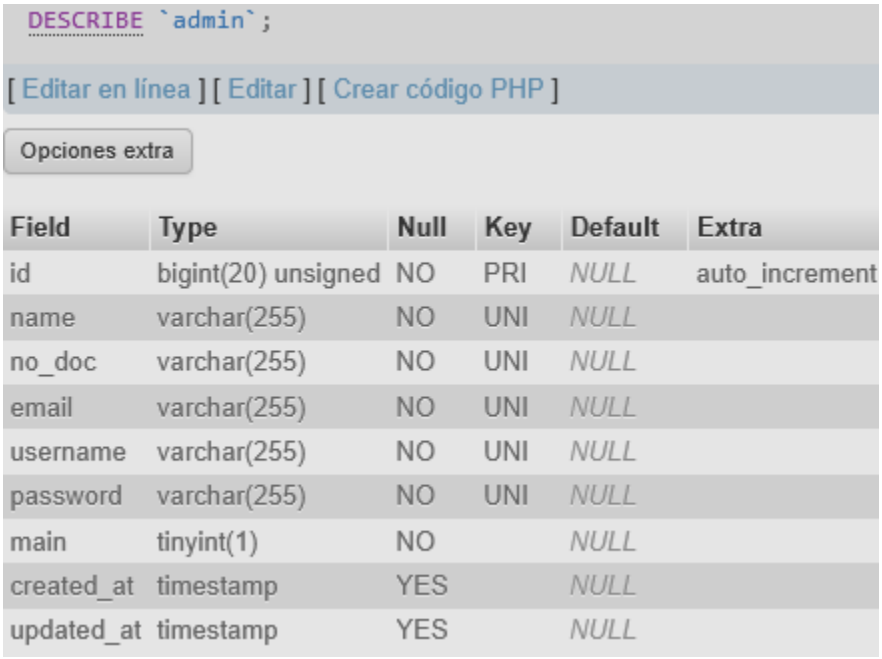
Las funciones desplegadas son Ingresar, Registrarse, Cerrar Sesión, Información. Al volver a hacer click en el menú hamburguesa, estas funciones se repliegan.

Iniciar Sesión. Al hacer click en iniciar Sesión, el sistema verifica si hay sesión activa, en tal caso se routea hacia el panel de administración, si no es así, se despliega en formato modal un formulario que pide las credenciales necesarias: *username* y *contraseña* para ser comprobada su existencia en la tabla *admin*.

Registro. Esta opción despliega en una modal el formulario necesario para registrar los datos de administrador en una base de datos. Los datos aquí pedidos son Nombre, No. documento, E-mail, Usuario, Contraseña. Estos datos son guardados en una entidad (tabla) llamada *admin* de la base de datos (Figura 11). Allí en la tabla *admin* se presenta adicionalmente un campo id numérico autoincremental como el anteriormente descrito, los campos *updated_at*, *created_at* que cumplen la misma función como los anteriormente descritos en la tabla *map*. Adicionalmente se hace presente el campo *main*, este se define de igual forma al campo post de la tabla *map*, pero en este caso su función es describir si el administrador registrado es principal (*main=true*) o no (*main=false*). Por defecto en primer administrador creado será principal.

Figura 11

Entidad 'Admin' de la Base de Datos



```
DESCRIBE `admin`;
```

[Editar en línea] [Editar] [Crear código PHP]

Opciones extra

Field	Type	Null	Key	Default	Extra
id	bigint(20) unsigned	NO	PRI	*NULL*	auto_increment
name	varchar(255)	NO	UNI	*NULL*	
no_doc	varchar(255)	NO	UNI	*NULL*	
email	varchar(255)	NO	UNI	*NULL*	
username	varchar(255)	NO	UNI	*NULL*	
password	varchar(255)	NO	UNI	*NULL*	
main	tinyint(1)	NO		*NULL*	
created_at	timestamp	YES		*NULL*	
updated_at	timestamp	YES		*NULL*	

Nota. Archivo de mi autoría base de datos *proy*, tabla *admin*

Menú Administrador. Una vez iniciada sesión, el sitio renderiza el subdominio */AdminV* el cual representa el panel de administrado, en la parte superior derecha se encuentra un icono menú hamburguesa como el de la figura 10. Al hacer click sobre él, se despliegan las siguientes opciones: Regresar a inicio, Cambiar contraseña, Actualizar administrador, Eliminar cuenta, Cerrar sesión. En el caso de un administrador principal, también se presenta la opción Autorizaciones.

A continuación, describiré los datos pedidos por los formularios en los modales que se despliegan en este menú:

- Cambiar contraseña: contraseña actual, contraseña nueva y confirmar *contraseña nueva.*
- Actualizar administrador: Nombre, N. documento, Email, Usuario.

- Eliminar cuenta: Contraseña, Confirmar contraseña.
- Autorizaciones (Administrador principal): *no*_doc, *email*, *main.*

Para el caso de las autorizaciones se hace uso de otra entidad en la base de datos como se describe en la figura 12, los campos id que funge como llave primaria autoincremental, *created_at* y *updated_at* son creados automáticamente por el backend.

Figura 12

Entidad 'Authorization' de la Base De Datos

```
DESCRIBE `authorization`;
```

[Editar en línea] [Editar] [Crear código PHP]

Opciones extra

Field	Type	Null	Key	Default	Extra
id	bigint(20) unsigned	NO	PRI	*NULL*	auto_increment
no_doc	varchar(255)	NO	UNI	*NULL*	
email	varchar(255)	NO	UNI	*NULL*	
main	tinyint(1)	NO		*NULL*	
created_at	timestamp	YES		*NULL*	
updated_at	timestamp	YES		*NULL*	

Nota. Archivo de mi autoría, base de datos *proy*, tabla *authorization*

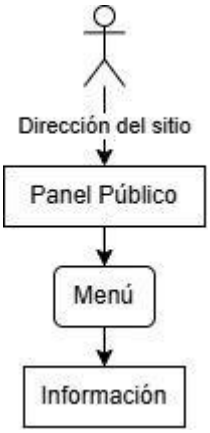
Caso de uso de Usuario Visitante

El caso de uso más común, por defecto es el del visitante que ingresa al sitio web (Figura 13) y es para quien esta desarrollado el aplicativo. Por lo tanto, se hace una breve investigación de ux-ui (experiencia de usuario e interfaz de usuario) para que su navegación en el sitio sea atractiva y la obtención de la información buscada ser optima. De acuerdo con ello y para que sepa el contexto en el que se encuentra se incluirá una pestaña donde se explica el propósito del aplicativo, logos de las entidades relacionadas, una barra de navegación fluida para acceder a los

principales dashboard disponibles y una paleta de colores sobria, pero de buen contraste para que no canse la visión.

Figura 13

Diagrama de Flujo de Caso de Uso de Visitante



Nota. Archivo de mi autoría

Los cuadros de dashboard con información adicional cargan en cascada en el centro de la página, su visualización está desactivada por defecto, esto aumenta la velocidad de carga del sitio y permite que solo el visitante pueda enfocarse en la información necesaria como se muestra en la imagen de referencia de la Figura 14.

Figura 14

Cuadro de Información de Dashboard

Nota Archivo de mi autoría

Para activar o desactivar la visualización del dashboard simplemente se hace click en el icono en forma de ojo de la esquina izquierda.

Desglose de Casos de Uso

Aquí se describen mediante un formato de tablas conciso las principales caractieristicas de los casos de uso.

El primero de los casos de uso es el de inicio de sesión, vital para obtener los permisos de administrador que rigen las funcionalidades principales del aplicativo web (ver Tabla 1).

Tabla 1

Iniciar Sesión, Caso de Uso

ID	1
Caso de uso	Inicio de sesión
Actores	Administrador
Propósito	Adquirir permisos
Descripción	1 Acceder al menú, iniciar sesión 2 Diligenciar y enviar el formulario 3 Se verifican las credenciales en la base de datos, tabla *admin* 4 Renderizado de las diferentes opciones del panel y menú de administración 5 Si es el administrador principal, podrá ver adicionalmente la opción del caso de uso 3
Precondiciones	Estar registrado en el sistema, estar la sesión cerrada
Postcondiciones	Es creada una sesión con los datos del administrador
Extensiones síncronas	Desde el backend se envía un *jwt* y es guardado dentro de una cookie del navegador por el sistema fronted, contiene la información de administrador*: main, name, no_doc, email, username*

Nota. Describe el caso de uso de inicio de sesión. Si se accede a esta opción con la sesión abierta, el sitio *routeará* al panel de administrador. Autor

Para que el inicio de sesión tenga sentido, es necesario que las credenciales ingresadas coincidan con las guardadas, para esto es necesario que dicho administrador este previamente registrado. El caso de uso de registro de administrador se describe en la Tabla 2.

Tabla 2

Registro de Administrador, Caso de Uso

ID	2
Caso de uso	Registro de administrador
Actores	Usuario
Propósito	Adquirir permisos y cargar datos
Descripción	1 Acceder a la opción de registro de administrador el menú 2 Diligenciar y enviar el formulario
Precondiciones	Estar vacía la tabla *admin* de la base de datos o tener autorización por email y *no_doc* registrados en tabla *authorization*
Postcondiciones	La información es cargada en la base de datos, tabla *admin*
Extensiones síncronas	

Nota. Describe el registro de administrador, los datos de un nuevo administrador son guardados con persistencia y sus credenciales de acceso son creadas. Autor.

Para evitar que cualquier visitante se registre como administrador -especialmente sin verificación de correo- se restringe el proceso de registro. Solo será posible si los datos ingresados coinciden con una tabla de autorizaciones previamente definida. Esta tabla se alimenta desde el caso de uso "otorgar autorización" (ver Tabla 3). Por ello, se recomienda que el primer administrador se registre en un entorno controlado.

Tabla 3

Otorgar Autorización, Caso de Uso

ID	3
Caso de uso	Otorgar autorización
Actores	Administrador Principal
Propósito	Autorizar
Descripción	1 Acceder a la opción menú, autorizaciones 2 Diligenciar y enviar el formulario
Precondiciones	Sesión iniciada, panel de administrador
Postcondiciones	Se cargan los datos *email* y *no_doc* en la base de datos en tabla *authorization*, se actualiza la lista de autorizaciones visibles en la segunda pestaña de la opción
Extensiones síncronas	

Nota. Describe el registro de administrador, los datos de un nuevo administrador son guardados con persistencia y sus credenciales de acceso son creadas. Autor

Cerrar sesión es necesario para la seguridad de los usuarios y protección de la información y opciones privadas del panel de administración cuando en un determinado caso este deja de ser utilizado después de alguna rutina de trabajo (ver Tabla 4).

Tabla 4

Cerrar sesión, Caso de Uso

ID	4
Caso de uso	Cerrar sesión
Actores	Administrador
Propósito	Remover permisos
Descripción	1 Se accede a la opción dentro del menú 2 El sitio se dirige al panel público, con todas las opciones correspondientes
Precondiciones	Tener una sesión de administrador iniciada
Postcondiciones	Se cierra la sesión y se borra el token correspondiente
Extensiones síncronas	Se borra la cookie del token guardada localmente

Nota. Describe el cierre de sesión, proceso con el cual se regresa al panel público y se revocan todos los permisos obtenidos por el inicio de sesión. Autor

Para revocar los permisos de administración y si aplica, regresar al panel público, se necesita la funcionalidad ‘regresar’ (ver Tabla 5).

Tabla 5

Regresar, caso de uso

ID	5
Caso de uso	Regresar
Actores	Administrador
Propósito	Ir a pantalla de inicio
Descripción	1 Hacer click en la opción regresar dentro del menú 2 El sitio se dirige al panel público, con todas las opciones correspondientes
Precondiciones	Tener una sesión de administrador iniciada
Postcondiciones	
Extensiones síncronas	

Nota. Describe la funcionalidad de regresar para acceder al panel público sin necesidad de perter los permisos de administrador. Autor

Cambiar la contraseña es una esencial funcionalidad de ciberseguridad. Ver Tabla 6.

Tabla 6

Cambiar Contraseña, Caso de Uso

ID	6
Caso de uso	Cambiar contraseña
Actores	Administrador
Propósito	Cambio de contraseña
Descripción	1 Hacer click en la opción dentro del menú 2 Se pedirá la contraseña antigua y contraseña nueva con confirmación 3 Enviar el formulario
Precondiciones	Tener una sesión de administrador iniciada
Postcondiciones	Realizar actualización de *password* en la tabla *admin* para el administrador actual
Extensiones síncronas	

Nota. Describe cómo trabaja la funcionalidad de cambio de contraseña. Autor

Actualizar datos para un administrador es necesario para mantener actualizada su información personal y reforzar su seguridad, este caso de uso se describe en la Tabla 7.

Tabla 7

Actualizar Datos de Administrador, Caso de Uso

ID	7
Caso de uso	Actualizar datos de administrador
Actores	Administrador
Propósito	Cambiar datos
Descripción	1 Hacer click en la opción dentro del menú 2 Diligenciar el formulario con los campos que se deseen actualizar 3 Enviar el formulario
Precondiciones	Tener una sesión de administrador iniciada, se pedirá contraseña
Postcondiciones	Realizar el correspondiente *update* en la base de datos
Extensiones síncronas	Se actualizan en local los datos visibles del administrador

Nota. Describe la funcionalidad de actualización de datos de administrador. Autor La

funcionalidad más representativa para los objetivos propuestos se describe a continuación en el caso de uso expuesto en la Tabla 8. Permite adjuntar la URL de un dashboard generado con el servicio en línea de publicación de Power Bi, agregar datos adicionales y subir estos datos para su persistencia en la base de datos.

Tabla 8

Crear Dashboard, Caso de Uso

ID	8
Caso de uso	Crear Dashboard
Actores	Administrador
Propósito	Cargar los datos de un dashboard
Descripción	1 Diligenciar los correspondientes datos en el panel de administrador 2 Enviar el formulario 3 Funciones opcionales de vista previa, eliminar y actualizar dashboard
Precondiciones	No repetirse datos existentes en la base de datos
Postcondiciones	Cargar o actualizar los datos en la base de datos
Extensiones síncronas	Se actualizan en local los dashboards visibles

Nota. Describe la funcionalidad de crear un dashboard o cuadro de información adicional a la url de dashboard, este conjunto de datos es persistido en la base de datos.. Autor

A su vez la finalidad de este procedimiento se complementa con la finalidad de exponer estos datos, hecho que se evidencia en el caso de uso de explorar dashboards (ver Tabla 9).

Tabla 9

Explorar Dashboards, Caso de Uso

ID	9
Caso de uso	Explorar dashboards
Actores	Visitante, Administrador
Propósito	Visitar el panel público del sitio
Descripción	1 Cargar la url del sitio web 2 Explorar los cuadros de información y dashboards según necesidad 3 Si le es necesario, abrir modal de información del menú
Precondiciones	
Postcondiciones	
Extensiones síncronas	

Nota. Describe la funcionalidad de exponer dashboard en el panel público para dar sentido a la aplicación web como software de visualización de datos. Autor

Diseño UI

Se remite al diseño de interfaces visuales atractivas, intuitivas y funcionales que faciliten la interacción del usuario con el contenido. Detalles como transiciones cortas y fluidas mejoran la experiencia al hacer que la navegación se sienta natural y dinámica, evitando pausas innecesarias en el flujo de la actividad.

La correcta elección de contrastes garantiza una mejor legibilidad y accesibilidad, permitiendo que los elementos clave destaquen sin abrumar al usuario. Además, tonos sobrios aportan elegancia y modernidad y facilitan tener una estética limpia y profesional que ayude a centrar la atención y la claridad visual en el contenido de la interfaz.

Para la implementación de este sitio web se tuvo en cuenta una paleta de colores sobrios centrada en colores como tonos de azul, blanco y negro, así como tonos de naranja y amarillo en cifras a destacar del dashboard. Esta paleta de colores tiene em propósito de no abrumar o cansar al usuario mientras se mantiene el buen contraste para la fácil lectura de los dashboards y navegación.

El minimalismo en el diseño de interfaces web se ha convertido en una tendencia dominante debido a su capacidad para mejorar la experiencia del usuario a través de la simplicidad y la claridad visual. Al contrario que en los diseños anteriores, que estaban saturados de elementos gráficos, colores alegres y animaciones innecesarias, el minimalismo apuesta por el espacio en blanco, las fuentes limpias y una paleta de colores reducida.

Esto no solo facilita la navegación y la comprensión de un vistazo, sino que también mejora la velocidad de carga de la web y la accesibilidad desde diferentes dispositivos. Además, en un panorama digital donde la saturación de información es un recurrente, los diseños

minimalistas permiten que los usuarios puedan centrarse en el contenido que necesitan sin distracciones.

Su auge también tiene que ver con las tendencias en branding y marketing, ya que las marcas están interesadas en trasladar su sofisticación, modernidad y eficiencia a través de interfaces intuitivas y bonitas de ver.

Para aplicar el minimalismo se tuvo en cuenta concretamente, se tuvo en cuenta no dejar de forma permanente letreros que indiquen lo que deben hacer los botones del menú de navegación y se dejaron las opciones ocultas por defecto, es por ello que el menú a pesar de tener bastantes opciones, solo se presenta con un pequeño icono flotante (el menú hamburguesa) y las opciones que implican formularios o despliegue de información adicional no implican el llamado a un nuevo subdominio, simplemente se despliega un modal (pop-up).

Implementación del Prototipo de Dashboard y Aplicativo Web

En esta sección se describe detalladamente de modo similar a un informe el cómo se implementó este prototipo de software en cada una de sus partes y tiene que ver con la palabra ‘hacer’ del ciclo de mejora continúa planteado inicialmente en la metodología PHVA.

Conceptualización

En esta sección se describen las arquitecturas y tecnologías utilizadas en la implementación del aplicativo web.

Arquitectura Cliente-Servidor

La arquitectura cliente-servidor se basa en la interacción entre dos actores principales: el cliente y el servidor. En este modelo, el cliente realiza solicitudes de recursos o servicios, mientras que el servidor se encarga de procesarlas y devolver las respuestas correspondientes.

Este enfoque facilita la separación de responsabilidades, permitiendo que el cliente maneje la presentación y la experiencia del usuario, mientras que el servidor se especializa en la lógica de negocio y la gestión de datos.

Gracias a esta división, ambos componentes pueden escalar de manera independiente, lo que mejora la capacidad para gestionar un mayor número de usuarios y operaciones de forma eficiente. Concretamente en cliente se denomina fronted y el servidor, backend.

El frontend y el backend, en este caso se comunican a través de solicitudes HTTP (URL Requests), intercambiando datos en formato JSON (JavaScript Object Notation). Este mecanismo permite que el frontend solicite información o envíe datos al backend, que luego responde con la información requerida.

Arquitectura Frontend con Vue 3, Composition API y TypeScript

Vue 3 es un framework de JavaScript que, mediante su enfoque basado en componentes, simplifica la creación de interfaces de usuario dinámicas y modulares. En el código fronted de vue 3 de esta implementación, se usa Composition API, una forma moderna de definir y ordenar el código de los componentes en vue 3. La incorporación de la Composition API aporta una mayor flexibilidad y mejor organización del código, ya que permite agrupar y reutilizar la lógica de los componentes de manera estructurada.

Con Composition API se pueden definir funciones reactivas y gestionar propiedades y métodos que pueden compartirse entre distintos componentes, sin depender del modelo tradicional basado en "opciones" de versiones anteriores. Esto mejora la claridad del código y facilita su mantenimiento, ya que toda la funcionalidad relacionada permanece agrupada dentro de un único bloque lógico.

Además, vue 3 presenta arquitectura MVC (modelo vista controlador), en esta implementación, el modelo se ve representado por los archivos de la carpeta *services* que definen la interacción con la base de datos y las interfaces que, a modo de contrato, definen los tipos y campos que deben ser utilizados en las diversas implementaciones de datos. La vista se ve representada en por los archivos *UI components* y su logia que se encarga de definir la vista a renderizar en cada momento en el navegador. El controlador se ve definido por las acciones del *<script setup>* de los componentes y el router que permiten gestionar la línea de ejecución de los diferentes casos de uso.

Uso de TypeScript en Vue 3. TypeScript es un lenguaje de programación desarrollado por Microsoft que extiende JavaScript mediante la inclusión de tipado estático y otras características avanzadas. A diferencia de JavaScript, donde las variables pueden contener

cualquier tipo de dato sin restricciones, TypeScript permite definir el tipo de datos esperado, lo que reduce errores y mejora la legibilidad del código, permitiendo así detectar errores durante la etapa de edición de código para reducir problemas en etapas posteriores donde son más difíciles de detectar y corregir.

En el contexto de Vue 3, TypeScript se usa para:

- Mejorar la seguridad del código: Detecta errores de tipo antes de la ejecución.
- Facilitar la escalabilidad: Permite desarrollar aplicaciones grandes y complejas con menos riesgo de errores inesperados.
- Favorecer el autocompletado y documentación: Los editores de código como VS Code ofrecen mejores sugerencias y detección de errores gracias al tipado de TypeScript.
- Optimizar el trabajo en equipo: Un código tipado es más fácil de entender y mantener, especialmente en proyectos colaborativos.

Arquitectura de Laravel 11 para API REST con Protección JWT

Laravel 11 se destaca como un framework de PHP diseñado para el desarrollo de aplicaciones web y servicios backend robustos. En el ámbito de las API REST, Laravel proporciona un conjunto de herramientas que simplifican la definición de rutas, la creación de controladores y la gestión de bases de datos. Su enfoque basado en la convención sobre configuración agiliza el desarrollo y reduce la necesidad de configuraciones complejas.

Laravel (incluida su versión 11) se basa fundamentalmente en la arquitectura MVC (Modelo-Vista-Controlador), que estructura la aplicación en tres partes principales:

- **Modelo (Model)**
  - Se encarga de la interacción con la base de datos y la definición de la lógica de negocio.

- En Laravel, los modelos utilizan Eloquent, su ORM integrado, para mapear las tablas de la base de datos a clases en PHP.

- **Vista (View)**

- Contiene la capa de presentación (en aplicaciones web tradicionales) o la salida (en el caso de una API REST).

- En esta API REST, la "Vista" no suele ser una plantilla HTML, sino las respuestas en formato JSON.

- **Controlador (Controller)**

- Centraliza la lógica que vincula modelos y vistas.

- Define los métodos que responden a las peticiones (requests) y devuelven las respuestas adecuadas (responses).

- En el caso de una API REST en Laravel, los controladores se ubican usualmente en la carpeta app/Http/Controllers y gestionan las rutas definidas en routes/api.php.

En cuanto a seguridad y autenticación, la implementación de JSON Web Tokens (JWT) es clave para proteger los recursos y verificar la identidad de los usuarios. El funcionamiento de JWT se basa en tokens digitales firmados, los cuales son enviados en cada solicitud del cliente al servidor. Si el token es válido, se garantiza que el usuario autenticado tiene los permisos adecuados y que las operaciones solo están disponibles para quienes cuenten con las credenciales correctas. Esto fortalece la seguridad en la comunicación y evita accesos no autorizados a información sensible.

Peticiones HTTP. HTTP (HyperText Transfer Protocol) es el protocolo principal que usan los navegadores y servidores web para intercambiar información. Cuando un cliente (por ejemplo, un navegador) solicita un recurso (como una página web o un archivo) al servidor,

envía una petición HTTP compuesta de distintos elementos (método HTTP, URL, cabeceras, cuerpo, etc.). El servidor procesa esa petición y devuelve una respuesta, que contiene un código de estado y, opcionalmente, datos en su cuerpo (por ejemplo, HTML o JSON, XML). En este caso se usan únicamente respuestas JSON que son desestructuradas en el fronted.

En la siguiente tabla se explican los métodos estandarizados de solicitudes HTTP utilizados en esta implementación.

Los métodos HTTP son esenciales en una API porque definen el tipo de acción que se desea realizar sobre un recurso. La Tabla 9 relaciona los métodos las comunes y que son implementados en el actual proyecto.

Tabla 9

Métodos de peticiones HTTP

Método	Propósito	Ejemplo de uso
GET	Solicitar un recurso del servidor.	Obtener datos de una API (por ejemplo, GET /users para obtener la lista de usuarios).
POST	Enviar datos al servidor para crear un nuevo recurso.	Crear un nuevo registro (por ejemplo, POST /users con los datos de un nuevo usuario en el cuerpo).
PATCH	Actualizar parcialmente un recurso existente en el servidor.	Modificar sólo un campo o parte de un recurso (por ejemplo, PATCH /users/1 para actualizar el correo).
DELETE	Eliminar un recurso existente del servidor.	Borrar un registro (por ejemplo, DELETE /users/1 para eliminar al usuario con ID=1).

Nota. Describe métodos HTTP vitales en la consolidación de los endpoint constituyentes del API y su correcto uso. Autor.

Durante la codificación de los métodos con los cuales se estructuran las respuestas que serán enviadas se incluyen códigos de respuesta estandarizados los cuales añaden información adicional y concreta sobre lo que ha ocurrido durante la petición http que dio lugar a dicha respuesta. Estos códigos permiten establecer un lenguaje común entre el cliente (por ejemplo, un navegador o una aplicación) y el servidor.

- Códigos 2xx (Éxito): Informan que el servidor ha procesado correctamente la petición.
- Códigos 3xx (Redirección): Indican que la petición requiere algún paso adicional para completarse, por ejemplo, seguir otra URL.
- Códigos 4xx (Error del cliente): Señalan que la petición tiene algún fallo (mal formada, no autorizada, etc.).
- Códigos 5xx (Error del servidor): Indican que el servidor no pudo procesar correctamente la petición por un problema interno.

Los códigos de respuesta HTTP informan si una operación fue exitosa (como el 200 OK), si hubo errores del cliente (como el 400 Bad Request) o del servidor (como el 500 Internal Server Error), lo que facilita el diagnóstico, la depuración y una gestión eficiente de errores en las aplicaciones que consumen la API.

La siguiente tabla (Tabla 10) hace un resumen explicativo de algunos de estos códigos.

Tabla 10

Códigos de Respuesta HTTP

Código	Mensaje	Descripción
200	OK	La petición se procesó con éxito y el servidor devuelve el recurso o la información solicitada.
201	Created	Se ha creado un nuevo recurso en el servidor, normalmente como resultado de una petición POST exitosa.
204	No Content	La petición se ha realizado con éxito, pero no hay contenido en la respuesta. Suele usarse tras una operación de borrado o actualización exitosa.
400	Bad Request	La petición está mal formada o faltan parámetros obligatorios.
401	Unauthorized	El cliente no está autenticado. Es decir, necesita credenciales válidas (como un token o inicio de sesión).
403	Forbidden	El servidor entiende la petición, pero el cliente no tiene permisos para acceder al recurso.
404	Not Found	El recurso solicitado no existe en el servidor.
405	Method Not Allowed	El método HTTP usado (GET, POST, etc.) no está permitido en ese recurso.
500	Internal Server Error	Error genérico cuando el servidor encuentra una situación que no sabe manejar.
502	Bad Gateway	El servidor actuó como puerta de enlace o proxy y recibió una respuesta inválida desde otro servidor.
503	Service Unavailable	El servidor no está disponible temporalmente (por mantenimiento o sobrecarga).

Nota. Esta tabla muestra los códigos de respuesta HTTP más utilizados, la columna mensaje muestra lo que significan y se incluye la columna descripción explicativamente. Autor

Uso de MySQL y Apache para la Administración con Laravel 11

MySQL es un sistema de gestión de bases de datos relacional que organiza la información en tablas y permite realizar consultas de manera eficiente. Se utiliza ampliamente en aplicaciones web por su fiabilidad y velocidad.

Apache, por su parte, es un servidor web que se encarga de recibir las peticiones de los usuarios y enviar las respuestas correspondientes a través del protocolo HTTP. Para que phpMyAdmin, que es una herramienta escrita en PHP, pueda funcionar, necesita ejecutarse en un entorno web. Además, apache interpreta los archivos PHP y sirve la interfaz de phpMyAdmin al navegador, permitiendo la administración de MySQL de forma sencilla y visual. Mediante PHP por ejemplo es que se pudieron obtener las imágenes descriptivas de las tablas de la base de datos: Figura 9, Figura 11, Figura 12.

Laravel 11 requiere un servidor web que procese los archivos PHP. Al usar Apache, Laravel puede funcionar correctamente para gestionar tanto las rutas como las vistas y la lógica de negocio, incluyendo la conexión y la administración de datos.

Implementación del Dashboard

Es esta etapa se hizo uso del software Power Bi para, posterior al modelado de datos en lenguaje DAX descrito en el Anexo B y dando continuidad a lo descrito en el subtítulo ‘visualización de datos’ del capítulo anterior, disponer los elementos visuales dispuestos según la disposición descrita en la Figura 6. En el anexo C este modelo y su conceptualización se documenta con más detalle.

Una vez dispuesto, coloreado, dimensionado y configurado este dashboard, es publicado en el servicio en línea oficial de Power Bi donde se obtiene un tag *iframe* de html que permite que sea insertado fácilmente en un sitio web.

Este tag es:

```
<iframe     title=""
src="https://app.powerbi.com/view?r=eyJrIjoiMDA3YjZlZmYtMmZhNC00Y2NmLTg
1ZjktMjAwM2Y4MjNlYzZjIiwidCI6ImZjMDA1NDdhLTI0YmItNGU0Zi05ZDYxLTczZmNhNWViOWRm
MyIsImMiOjR9"
                frameborder="0"
                allowfullscreen="true">
</iframe>
```

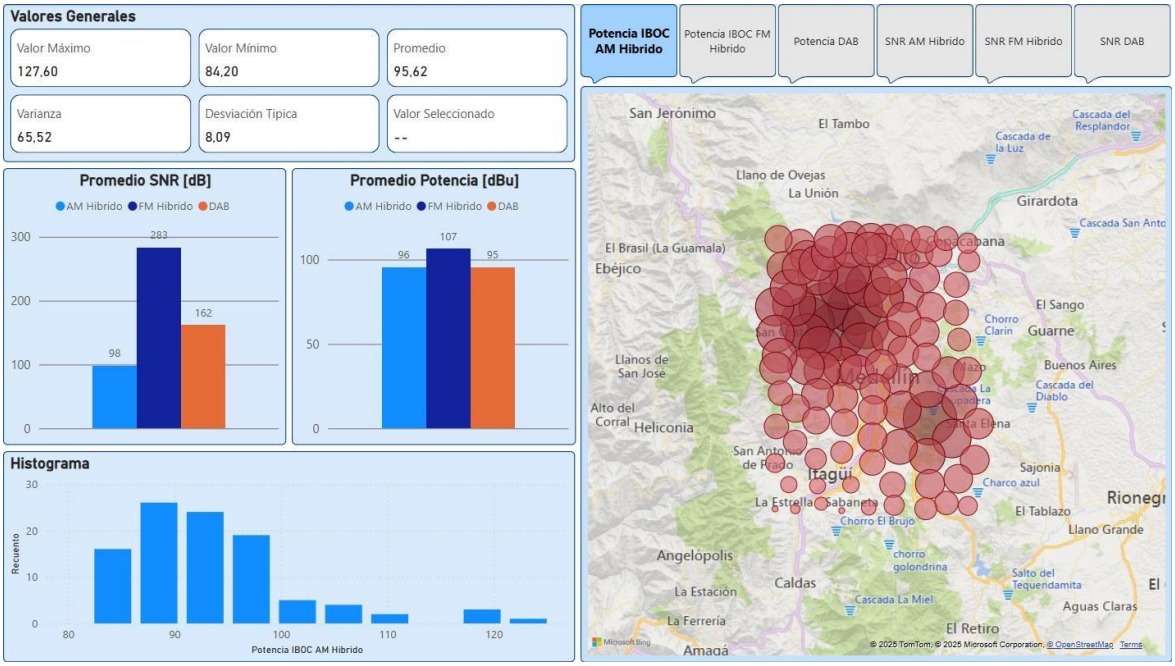
Aquí se tomará la URL generada:

https://app.powerbi.com/view?r=eyJrIjoiMDA3YjZlZmYtMmZhNC00Y2NmLTg1ZjktMjAwM2Y4MjNlYzZjIiwidCI6ImZjMDA1NDdhLTI0YmItNGU0Zi05ZDYxLTczZmNhNWViOWRmMyIsImMiOjR9 para ser guardada en la base de datos ya que el tag iframe se dispondrá de otra manera para ser manipulado en el framework de desarrollo fronted.

Una vista previa de este dashboard generado se muestra a continuación, ver Figura 15.

Figura 15

Captura del Dashboard Generado



Nota. Archivo de mi autoría dashboard.pbix

Implementación de Fronted

Para consultar el código completo de esta implementación puede visitarse el repositorio remoto de github gieribac de autoría propia:

https://github.com/gieribac/pg_dashboard_fronted_vue

También es posible consultar directamente los componentes vista principales AdminV y MainV en el anexo D.

Configuración Inicial

La instalación se hizo teniendo en cuenta el entorno de sistema operativo Windows 10.

- Instalación de node.js: Se visita el sitio web de node.js y se instala la última versión recomendada (20.18.0) en el sitio web https://nodejs.org/.

- Creación del proyecto de vue 3 desde npm (Node Packege Manager) con Vite y Typescript:

```
npm create vite@latest dashboards_dr --template vue-ts
```

En las opciones es necesario escoger el uso de ruter y que no se utilizara preprocesador de estilos.

- Instalación de dependencias de vite y vue:

```
npm install
```

- Instalación de dependencias necesarias para el manejo de la cookie y decodificación de jwt:

  - `npm install @vuelidate/core @vuelidate/validators js-cookie jwt-decode vue@^3.5.10 vue-router@^4.5.0`

  - `npm install --save-dev @types/js-cookie @vitejs/plugin-vue typescript@^5.5.3 vite@^5.4.8 vue-tsc@^2.1.6`

- Ejecución en modo de desarrollo:

```
npm run dev
```

Esto ejecuta el servidor de desarrollo en http://localhost:5173/, si se requiere parar este servidor de desarrollo es necesario presionar *ctrl+c* en la consola que fue ejecutado

- Ejecución de compilación: Esto se realiza una vez se ha testeado y comprobado el desarrollo para ponerlo en producción. Se generan archivos de javascript minificados y optimizados, los cuales se ubican en una carpeta llamada *dist/:*

```
npm run build
```

Para previsualizar la versión compilada de la aplicación se utiliza:

```
npm run preview
```

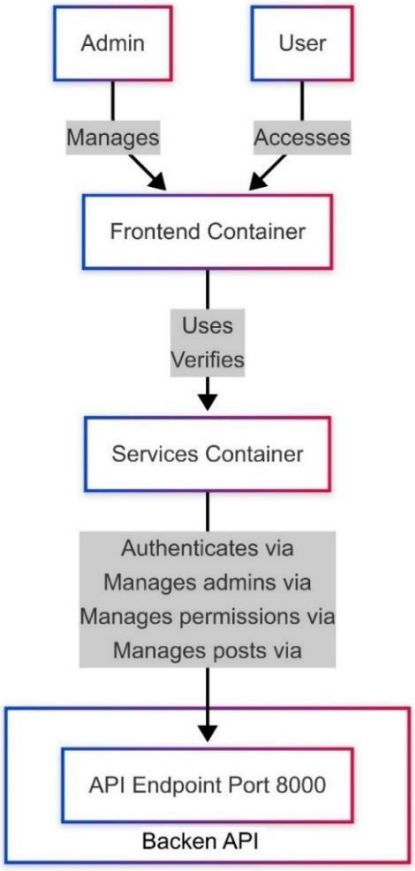
Disposición General

La implementación lógica se basó en los dos perfiles principales: administrador y visitante. Un archivo importante de la aplicación es *router/index.ts,* un archivo de typescript el cual se encarga de especificar lo que se va a renderizar según el subdominio. En la aplicación hay dos subdominios: *MainV* y *AdminV.* Si un usuario llegar a digitar algo diferente como subdominio del sitio, *router/index.ts* está configurado para redirigir el sitio hacia el subdominio *MainV.*

La navegación hacia AdminV está condicionada a la validez del token guardado en la cookie del navegador (existe y aún no expira), de lo contrario el sitio se redirige hacia MainV. Dentro de *router/index.ts* se relaciona el renderizado de los dos componentes de vistas principales: *Admiv.vue* si el subdiminio *es AdminV y MainV.vue* si el subdominio es *MainV* los cuales son llamados dentro del Fronted Container (*App.vue*) por el tag <*router-view*/>. Ver Figura 16.

Figura 16

Estructura Fronted en Vue 3



Nota. Autoría propia

La implementación actual se denomina aplicativo web porque, en contraste con un sitio estático, se actualiza a medida que consume datos desde la base de datos. Por esta razón se han estructurado archivos con clases, las cuales se encargan de realizar el respectivo *fetching (*obtención de datos mediante el método *fetch* de javascript, de fuentes externas*)* apuntando a los *endpoint* del API backend. A esta etapa se le denomina *service conainer* (contenedor de servicios) porque los métodos de clase utilizados cuando son instanciadas dentro de los componentes para consumir o cargar datos se denominan servicios.

Contenedor Fronted

Como toda página web estándar, cuenta con un archivo principal de HTML llamado index.html, ubicado en la carpeta raíz del proyecto. En su etiqueta script, este archivo enlaza al código src/main.ts, el cual a su vez llama a App.vue, que actúa como el componente raíz o contenedor principal de los componentes de Vue.

Dentro de App.vue, se renderizan los componentes principales conocidos como View Components, llamados así porque son gestionados por el router. Los demás componentes conforman distintas partes de la interfaz, como el cuadro de dashboard, el formulario de digitalización del dashboard, el menú, los modales(popups) de opciones y los formularios de diversos casos de uso. Estos se invocan dentro de cada vista según corresponda y se denominan UI Components, ya que forman parte concreta de la interfaz visible.

El componente Dashboard Main es un caso particular dentro de los View Components, ya que, aunque no está directamente relacionado con el router, puede contener elementos de UI, como cuadros de información del dashboard que se mostrarán a los usuarios si existen datos en la base de datos.

Dentro del contenedor frontend, también existen los "core components" (o componentes núcleo), que son módulos o piezas de código esenciales para el funcionamiento global del proyecto. Estos no forman parte directa de la lógica de las vistas ni del diseño de la interfaz, pero proporcionan servicios clave. Entre ellos se encuentran los archivos router/index.ts, main.ts, authHelpers.ts y App.vue, los cuales desempeñan funciones fundamentales dentro de la aplicación:

- Router (Vue Router): Gestiona la navegación dentro de la aplicación de Vue, asignando rutas (URLs) a componentes o vistas. Permite crear páginas o secciones sin necesidad de recargar

toda la ventana del navegador, facilitando así la arquitectura de una Single-Page Application (SPA). El código fuente que implementa esta funcionalidad se encuentra en el archivo correspondiente, cuyo contenido puede consultarse en el sistema de respaldo y control de versiones proporcionado por el repositorio de *GitHub: router/index.tsh*.

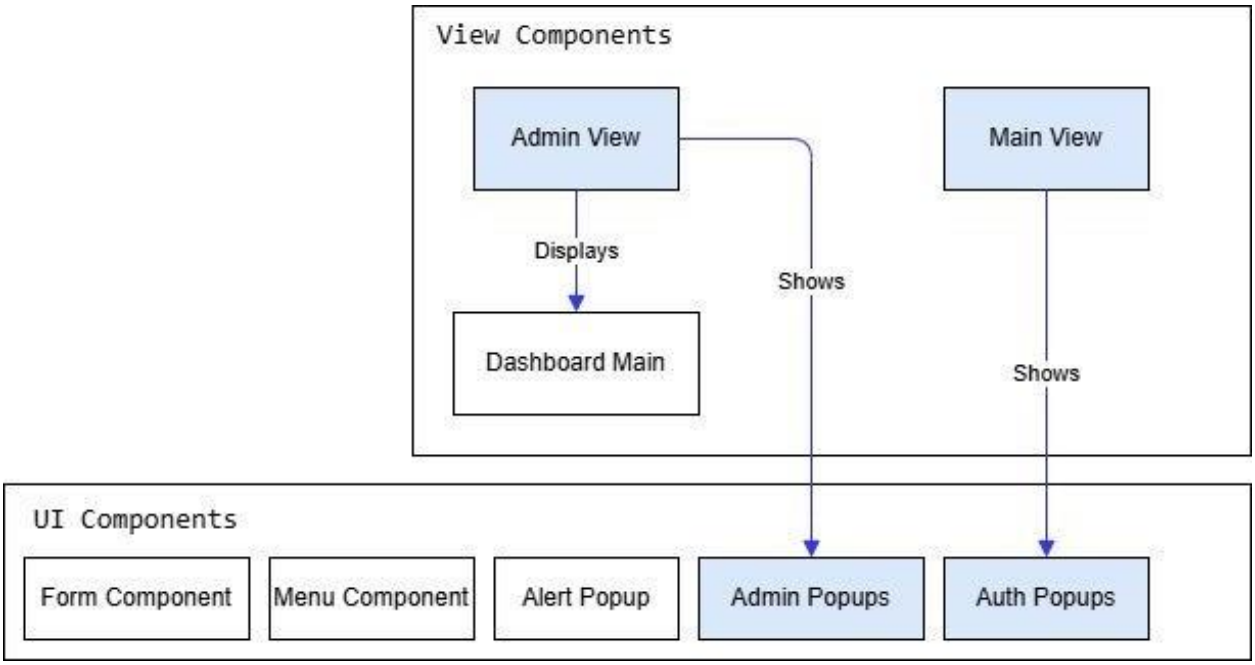
- Auth Manager (Gestor de Autenticación): Se encarga de validar y almacenar el estado de autenticación del usuario (JWT, tokens, cookies, etc.). Generalmente, proporciona métodos como login, logout, checkAuth o refreshToken, además de permitir que otros componentes o servicios verifiquen si el usuario está autenticado y qué permisos tiene.

Estos componentes núcleo son los encargados de gestionar la lógica con la que los UI Components y View Components se renderizan dentro de la aplicación.

En la Figura 17, se presenta la arquitectura funcional de esta implementación de componentes.

Figura 17

Distribución de Componentes



Nota. Autoría propia

Hay un archivo que presta funcionalidades necesarias para las funciones que requieren que el usuario este autenticado como lo son conseguir la información del token, este es *authHelpers.ts*.

A continuación, se muestran algunos servicios que hacen uso de algunas de estas funcionalidades:

```
//AdminService.ts
//Método para obtener todos los registros de administrador
    async fetchAll(): Promise<void> {
        try {
            if (!isTokenValid()) {
               this.routerToMain(); // Redirige al usuario a la página
               de inicio
                throw new Error('Token vencido');
            }
            const token: string = returnToken();
            const response: Response = await fetch(url,{
```

```
                headers: {
                    'Authorization': `Bearer ${token}`
                },}
            );
            if (response.status === 401) {
                  this.routerToMain(); // Redirige si el token es
                  inválido desde el servidor
                  throw new Error('No autorizado. Token inválido o
                  vencido');
            }
            const json: AdminRegData[] = await response.json();
            this.admins.value = json;
        } catch (e) {
            console.error('Error fetching admins: ', e);
        }
    }


// AuthorizedService.ts
// Método para registrar authorization
    async regAuthorization(data: MAData): Promise<boolean> {
        try {
            if (!isTokenValid()) {
                  this.routerToMain(); // Redirige al usuario a la
                  página de inicio
                throw new Error('Token vencido');
            }
            const token: string = returnToken();
            const response = await fetch(url, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify(data),
            });
            if (response.status === 401) {
                  this.routerToMain(); // Redirige si el token es
                  inválido desde el servidor
                throw new Error('No autorizado. Token inválido o
                vencido');
            }
            if (!response.ok) {
                const errorData = await response.json();
                  throw new Error(errorData.message || 'Error en el
                  registro.');
            }
            const responseJson = await response.json();
```

```
            this.authoriationsData.value.push(responseJson.data);
            return true;
        } catch (err) {
            console.log(err);
            return false;        }     };
```

Los archivos completos correspondientes pueden consultarse en *services/AuthService.ts* y *services/AuthorizedService.ts*.

La clasificación de archivos en carpetas fue hecha clasificándolos según su funcionalidad (*screaming architecture*). A continuación, se presenta el respectivo árbol de directorios mostrando la disposición de los componentes (el resto de los archivos de las carpetas *src* y *components* se ha omitido de la representación). Ver figura 18

Figura 18

Árbol de Directorios de Componentes

```
src
├── components/
│   ├── adminPop_up/
│   │   ├── ChangePass.vue
│   │   ├── DestroyUser.vue
│   │   ├── ManageAuthorizations.vue
│   │   ├── PatchUpdate.vue
│   │   └── PrevDashboard.vue
│   ├── mainPop_up/
│   │   ├── LoginPopup.vue
│   │   └── RegPopup.vue
│   ├── AlertPopup.vue
│   ├── DashboardMain.vue
│   ├── FormD.vue
│   └── MenuComponent.vue
│
├── views/
│   ├── AdminV.vue
│   └── MainV.vue
├── App.vue
```

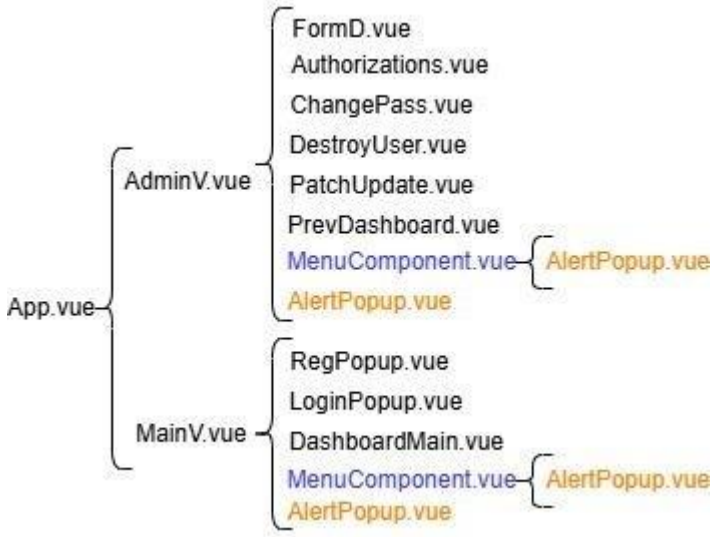
Nota. Autoría propia

Los formularios que permiten la interacción con el usuario permiten el intercambio de datos tales como información y eventos, se renderizan dentro de los *UI components* (carpeta *components/)*, esta información viaja verticalmente por medio de *props* (información viaja de forma descendente de un componente padre al hijo) y *emmits* (un evento originado en el componente hijo efectúa la ejecución de una función en el padre). Esto, hasta los elementos padre principales que conectan los servicios (carpeta *views/*).

La conexión entre componentes padres e hijos se evidencia por los *imports* en la parte superior del tag script de cada componente padre. Sin embargo, la funcionalidad de un mismo componente hijo varía según el componente padre desde el que se está ejecutando y los datos que le sean enviados (ver Figura 19).

Figura 19

Relación entre Componentes View y UI de Vue



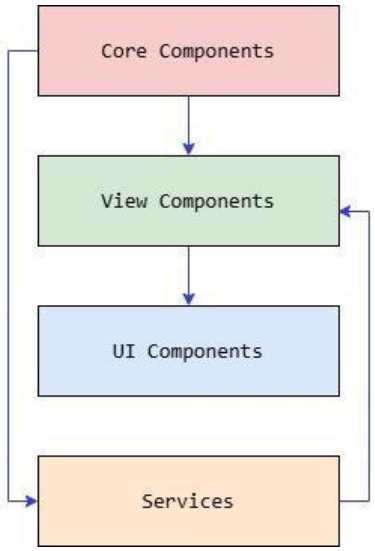
Nota. Autoría propia.

En la Figura 20 puede verse cómo interactúan las 4 clasificaciones funcionales de componentes donde Core Components provee datos y funcionalidad a *services* como lo son

gestión del token traído del backend por *services* como lo son su guardado en la cookie local y su invocación para ser enviado y extraído de información cuando sea necesario. También provee datos a componentes View y al router para validar lógica de permisos.

Figura 20

Relación de Bloques Funcionales del Fronted



Nota. Autoría propia

A su vez los servicios trabajan a solicitud de los View components para interactuar con la base de datos y gestionan el renderizado de los UI Componentes.

Servicios

Las funcionalidades principales que permiten dotar al sitio web del dinamismo necesario para ser definido como aplicación web son implementadas en los archivos dispuestos en la carpeta *services/* y el propósito funcional de los más importantes se describe a continuación:

- *AdminService.ts:* Define y suministra la clase *AdminService*, la cual provee métodos que permiten conectarse a la base de datos a la *url* base *'http://127.0.0.1:8000/api/admin/'* para la mayoría de sus metodos es necesario hacer uso de recursos provistos por *authHelpers.ts.* Cabe

destacar que en ocasiones el token jwt es necesario ser enviado al backend para los privilegios del uso del endpoint, para esto se envía el token en la cabecera de la solicitud así: `headers: { 'Authorization': `Bearer ${token}`…`

Algunos de estos métodos son:

- Obtener todos los registros de administradores al *endpoint.* Mediante *async fetchAll()* con el método GET el cual cuando no se menciona, está incluido por defecto: *await fetch(url,{…*
- Registrar un nuevo administrador en la base de datos mediante la url base mencionada y el método `HTTP POST: await fetch(url, { method: 'POST',…`
- Actualizar los datos del administrador actual en la base de datos (*async updateAdmin*), para ello es necesario hacer uso del método HTTP PATCH y enviar el id del administrador en la url: *`await fetch(`${url}${id}`, {method: 'PATCH',…`* El id es necesario obtenerlo del token que se ha guardado localmente cuando se ha iniciado la sesión, también es necesario tener en cuenta que estos datos que han sido actualizados deben actualizarse localmente para que se renderice información actualizada, en este caso esto se realiza en el archivo *AdminV,vue* cuando hace uso de este método.

- *AuthService.ts:* Provee y suministra la clase *AuthService* la cual suministra métodos relacionados con la autenticación, siendo la *url* base llamada *urlAdminAuth*: ‘*http://127.0.0.1:8000/api/adminlog/':*
  - Login (Autenticarse como administrador para adquirir privilegios): Mediante *async login* se hace *fetch* con el método POST: *await fetch(`${urlAdminAuth}login`, { method: 'POST',…* este método utiliza *authHelpers* para hacer uso de una funcionalidad que le permita guardar el token jwt enviado desde el servidor en la cookie del navegador.

- Cerrar sesión (*async logout*): Cómo es necesario estar autenticado y borrar el token, también aquí es necesario utilizar los métodos necesarios desde *authHelpers* (*getTokenFromCookie* y *clearTokenFromCookie* respectivamente). Para ello se utiliza *fetch* con el método HPPT *POST y el* parámetro *logout*:

```
await fetch( `${urlAdminAuth}logout`, {method: 'POST',…
```

- Cambiar contraseña: Para esto es necesario comprobar el token (si no es válido, se redirige el sitio al panel publico), se obtiene la id del actual administrador del token de sesión guardado y se utiliza el método *fetch* con la *url* mencionada, la *id* y el método HTTP *PATCH* así:

```
await fetch(`${urlAdminAuth}updatepassword`, {method: 'PATCH',…
```

- Eliminar el administrador actual: Es necesario para esto, comprobar la validez del token (*isTokenValid*) y decodificarlo para obtener el id actual (*getDecodedToken*) gracias a los metodos provistos por authHelpers.ts

```
await fetch(`${urlAdminAuth}${id}`,{method: 'DELETE'…
```

- *AuthorizedService.ts: Este archivo se basa en la conección a la url 'http://127.0.0.1:8000/api/authorized/'* con el fin de crear las gestiones necesarias para la gestión de autorizaciones, lo que permite obtener todas las autorizaciones existentes, registrar nueva, eliminar una autorización. Estos métodos hacen uso de la función *isTokenValid* de *authHelpers.ts*. Otros métodos presentes no hacen uso de esta función, pero son necesarios para la gestión local como lo son el uso del router para cambiar el direccionamiento y el *getter getAuthoriationsData* para el consumo de los datos guardados en la clase.

- *PostService.ts.* Este archivo se encarga de gestionar los servicios relacionados con la conexión con la base de datos de los dashboard y su información complementaria mediante la implementación de la clase llamada *PostService*. Los métodos *presentes fetchAll, createPost,*

deletePost, updatePost son análogos a los metodos creados para la gestion de registros de administrador del archivo *AdminService.ts* sin embargo en este caso la url base es *'http://127.0.0.1:8000/api/maps/'*. Todos estos, a excepción de *fetchAll*, requieren verificación del token pues su uso de privado del administrador. También es destacable mencionar que adicionalmente se implementa un método llamado *fetchAllPublics* el cual incluye un filtro como parámetro cuya ocupación es pedir los cuadros de dashboard que se encuentren guardados como públicos en la base de datos, pues estos serán renderizados en el panel público. Este método al tener relación directa con el panel público, tampoco requiere comprobación del token: await fetch(`${url}?filter=public`)…

- *authHelpers.ts.* Este archivo provee funciones de vital importancia y estrecha relación con los *core components* pues permiten gestionar el token jwt suministrado por el backend. Este archivo hace uso de las librerías *'js-cookie'* y *'jwt-decode'* que permiten la manipulación de la cookie del navegador y del jwt (json web token) respectivamente. Las funciones aquí definidas son consumidas por archivos de la carpeta *services* y otros *core components* para proveer datos protegidos, autenticación, gestión de rutas, entre otras. Estas funciones son:
    - *returnToken:* Retorna el token tras extraerlo desde la cookie y comprobar que tiene tipo de dato *'string'*.
    - *setTokenInCookie:* Recibe como parámetro el token suministrado por el backend y lo guarda en la cookie.
    - *isAuthenticated.* Comprueba si el token guardado localmente en la cookie existe y no está vacío.

- getDecodedToken. Comprueba el token y lo retorna decodificado si es válido. Esto para extraer información guarda del administrador actual. Los atributos que son guardados en el token son (extraído del archivo de tipado interfaces/DecodedToken.ts):

sub: string; // ID del administrador

main: boolean; // Indica si el administrador es principal

name: string; // nombre

no_doc: string; //número de documento

email:string; //dirección de correo electrónico

username: string; //nombre de usuario

- getTokenFromCookie*:* Retorna el token desde la cookie del navegador.
- clearTokenFromCookie*:* Borra la cookie que guarda el token jwt.
- isTokenValid*:* Comprueba la validez del token, esto incluye que no haya expirado comprobando el tiempo actual y el tiempo de expiración configurado (una hora).

Implementación de Backend

En esta implementación de utiliza Laravel 11 para la creación de API REST con salidas formato JSON en los endpoint (puntos de conexión de URL que permiten el acceso a recursos mediante protocolo http para conexión cliente servidor) de manera clara y eficiente.

Laravel 11 utiliza Artisan, su interfaz de línea de comandos, que permite generar controladores, modelos y migraciones con comandos simples, reduciendo el tiempo de desarrollo. También proporciona Eloquent ORM, un sistema de mapeo objeto-relacional que simplifica las consultas a la base de datos. Esto básicamente se trata de modelar una tabla MySQL existente con una determinada instancia de clase que tiene los mismos atributos y utilizar métodos propios de Eloquent para realizar consultas y operaciones que de otra forma

habría que hacer directamente con lenguaje SQL. Además, Laravel incorpora mecanismos avanzados de middleware para gestionar autenticación, autorización y validación de solicitudes, garantizando seguridad y eficiencia en las API.

En términos de seguridad, manejo de sesión y gestión de acceso a los endpoint e información adicional, se utiliza jwt mediante una librería externa ya que Laravel 11 no cuenta de forma nativa Laravel no cuenta con esta tecnología.

El repositorio de código completo de mi autoría de este backend puede consultarse en *https://github.com/gieribac/pg_dashboard_backend*.

Configuración Inicial

- Requisitos y Configuración de Laravel 11 con XAMPP

Laravel 11 requiere los siguientes componentes para su correcto funcionamiento:

- Servidor web: Se necesita un servidor de alojamiento de aplicaciones, en este caso Apache.
- Base de datos: Laravel requiere una base de datos para almacenar información de forma permanente, en este caso MySQL.
- PHP: Laravel está escrito en PHP, por lo que es fundamental contar con un intérprete de PHP instalado en el sistema.

Dado que XAMPP incluye Apache, MySQL y PHP en su instalación, además de phpMyAdmin (una herramienta gráfica para la gestión de bases de datos desde el navegador), es una opción práctica para configurar el entorno de desarrollo.

- Instalación y Configuración de XAMPP
- Descargar e instalar XAMPP desde Apache Friends, asegurándose de seleccionar los complementos necesarios.

- Configurar PHP: Abrir la configuración de Apache y localizar el archivo php.ini, buscar zip (Ctrl + B, escribir "zip"), descomentar la línea eliminando el ‘;’ al inicio para habilitar la extensión ZIP.
- Iniciar los servicios desde el panel de control de XAMPP: Activar Apache y MySQL.
- Acceder a phpMyAdmin desde el botón Admin en el panel de XAMPP para gestionar la base de datos y comprobar que está en línea.

- Instalación de Composer

Otra herramienta esencial es Composer, un administrador de paquetes para PHP que permite instalar Laravel y sus dependencias.

- Descargar Composer desde getcomposer.org.
- Instalarlo siguiendo las instrucciones de la página oficial.

- Instalación de Laravel 11

Laravel también requiere Node.js, pero si ya se ha instalado previamente para el desarrollo del frontend con Vue 3, no es necesario volver a instalarlo.

Para instalar Laravel 11, se recomienda utilizar PowerShell y ejecutar el siguiente comando:

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

A continuación, se debe ubicar dentro de la carpeta de instalación de XAMPP:

- Acceder a *xampp/htdocs/*.
- Crear una nueva carpeta (por ejemplo, 1proyectos_laravel), que servirá como directorio de trabajo para los proyectos Laravel.
- Dentro de esta carpeta, ejecutar el siguiente comando para crear un nuevo proyecto:

```
laravel new proy
```

- Acceso y Ejecución de Laravel

Laravel es un framework diseñado originalmente para el desarrollo Full Stack, permitiendo la creación tanto del backend como del frontend. El punto de acceso del proyecto es el archivo *public/index.php.*

Para acceder desde el navegador, se puede utilizar:

http://localhost/1proyectos_laravel/proy/public

Donde:

- 1proyectos_laravel es la carpeta contenedora del proyecto.
- proy es el nombre del proyecto creado.
- *localhost* (http://localhost) generalmente corresponde al puerto 127.0.0.1.

También es posible iniciar el servidor de desarrollo de Laravel con el siguiente comando:

```
php artisan serve
```

Este comando ejecutará un servidor de desarrollo temporal. Para detenerlo, basta con presionar *Ctrl + C* en la terminal donde se ejecutó.

Es importante recordar que, antes de trabajar en Laravel, los servicios de MySQL y Apache deben estar activos en XAMPP.

- Instalación de API: Esto prepara el entorno para la creación de API REST, para esto crea el archivo sto creara el archivo *api.php* dentro de la carpeta *routes/* y preparara configuraciones y la preparación en *boostrap/app.php.* Además, establece una ruta raíz de acceso para todos los endpoints que serán creados, esta es *localhost:8080/api/:*

```
php artisan install:api
```

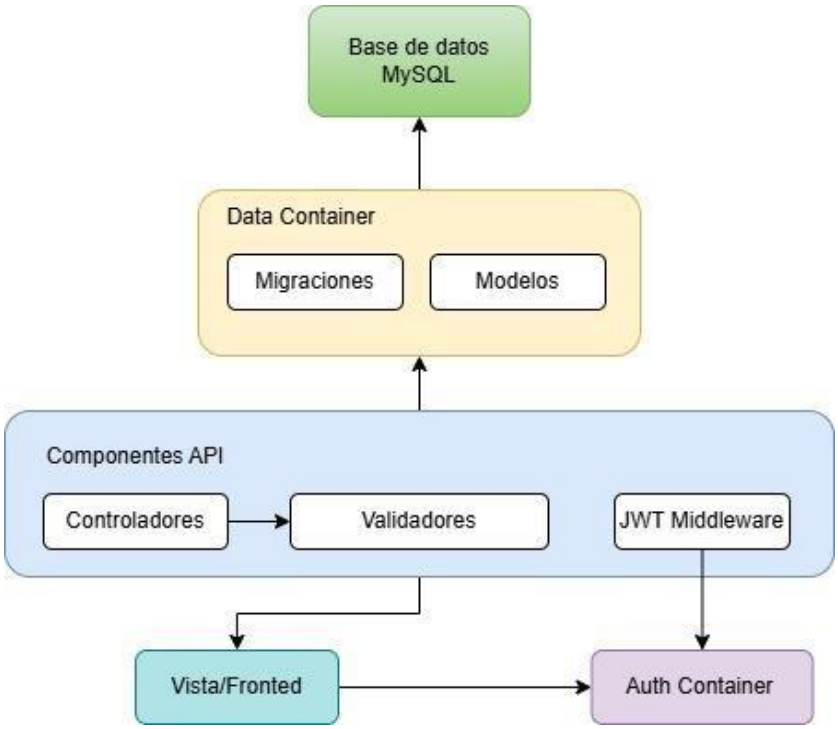
Una vez hecho esto, se configura el método *withRouting* dentro del archivo *bootstrap/app.php:*

```
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        //web: __DIR__.'/../routes/web.php',
        api: _DIR_.'/../routes/api.php',
```

Es importante aclarar que estructura del backend, siguiendo el flujo de datos implementado dada la arquitectura de laravel 11 corresponde con el diagrama presentado en la Figura 21. Estos bloques funcionales que se presentan. son explicados en los siguientes subtítulos.

Figura 21

Diagrama General de Backend



Nota. Autoría propia

Implementación de Migraciones

Las migraciones se consideran parte del modelo (según la arquitectura MVC) y cumplen con la función de definir las tablas de la base de datos, incluyendo los tipos de datos, sus propiedades y el modelo relacional si aplica. Para ello se debe crear un archivo de migración con el nombre deseado de la tabla. Esto se lleva a cabo mediante una función de PHP que incluye algunas palabras clave de Illuminate.

Para crear esta migración es necesario utilizar el comando:

```
php artisan make:migration create_maps_table
```

donde *maps* es el nombre que se desea darle a la tabla en la base de datos, esto crea un

archivo en la carpeta *database/migrations/*, dicho archivo debe ser editado para personalizar la migración. Opcionalmente, se puede agregar *--create=maps* al final del comando. Esto genera automáticamente la estructura de la tabla con el método *Schema::créate*.

El código a continuación corresponde a la migración con la que se crea la tabla 'map' cuya tabla una vez generada con esta migración puede verse en la Figura 9.

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('map', function (Blueprint $table) {
            $table->id();
            $table->boolean('post');
            $table->string('title');
            $table->string('description');
            $table->string('author');
            $table->string('urlDashboard');
            $table->string('place');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('map'); }};
```

Una vez editadas las migraciones, se debe correr (editar la base de datos), para ello se usa el comando `php artisan migrate`.

Y si hay que corregir algo, se debería revertir la ultima migración mediante el comando `php artisan migrate:rollback`.

Otros comandos útiles en este contexto pueden ser:

- Revertir y volver a ejecutar las migraciones en un solo paso:

```
php artisan migrate:refresh
```

- Revertir un numero especifico de migraciones, por ejemplo 2:

```
php artisan migrate:rollback --step=2
```

- Listar todas las migraciones aplicadas y su estado:

```
php artisan migrate:status
```

En esta carpeta (database/migrations/) se crean las migraciones para las tablas del proyecto: *map, admin, authorization.*

Implementación de Modelos

En laravel, los modelos son una parte fundamental del patrón MVC, representan las talas de las bases de datos, lo que permite interactuar con ellas mediante Eloquent ORM. Los modelos deben corresponder en estructura con las tablas creadas mediante las migraciones, por consiguiente, los modelos que son creados son *Map, Admin, Authorization* (por conención se usa mayúscula inicial). Para crear estos modelos se usan los comandos `php artisan make:model Map, php artisan make:model Admin, php artisan make:model Authorization.`

Esto crea los correspondientes archivos en la carpeta *app/Models/*. Luego de crear los archivos modelo, deben ser editados los campos para describir la tabla de MySQL, a continuación, el caso *Map.*

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Casts\Attribute;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Map extends Model
{
    use HasFactory;
    protected $table = 'map';//nombre del modelo
    protected $fillable = [ //campos alterables
```

```
        'post',
        'title',
        'description',
        'author',
        'urlDashboard',
        'place'
    ];
    protected function post(): Attribute
    {
        return Attribute::make(
            get: fn ($value) => (bool) $value,
        );
    }
}
```

Una característica importante de la base de datos MySQL es que no guardan tipo de datos booleano. Cuando laravel intenta guardar un dato booleano este se guarda de la siguiente forma: 1 para *true* y 0 para *falso*. Por esta razón se ha incluido la función *post* la cual se encarga de crear una operación que antecede a cualquier consulta en el modelo y que consiste en convertir a booleano de nuevo los números guardados. Esta implementación de denomina accesor para el atributo post. Funciona así:

- *Attribute::make()* se usa para definir un Accessor y, opcionalmente, un Mutator (para modificar antes de guardar). En este caso, solo define un Accessor.
- *get:* define la transformación cuando se obtiene el valor del atributo.
- Usa una función flecha (*fn*) para convertir $value en booleano (true o false).
- *(bool) $value* convierte cualquier valor en true o false:
    - *0, null, '', false → false*
    - *1, 42, "hello", true → true*

A continuación, se edita el modelo *Admin*:

```
<?php
namespace App\Models;
```

```
use Illuminate\Foundation\Auth\User as Authenticatable;
use Tymon\JWTAuth\Contracts\JWTSubject;
class Admin extends Authenticatable implements JWTSubject
{
    protected $table = 'admin';//nombre del modelo
    protected $fillable = [ //campos editables
        'name',
        'no_doc',
        'email',
        'username',
        'password',
        'main'
    ];
     protected $casts = [
        'main' => 'boolean',
        'password' => 'hashed',
    ];


    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
}
```

En este modelo se incluyen otras configuraciones adicionales:

- *protected $hidden = ['password', 'remember_token',];* Sirve para evitar exponer los datos *'password', 'remember_token'* por seguridad.

- *protected $casts = ['main' => 'boolean', 'password' => 'hashed'];* Mediante la propiedad protegida $cast de laravel, *‘main’* será siempre convertido a tipo de dato booleano y password será siempre encriptado mediante método hash si no lo está.

Implementación de EndPoints

La implementación de los endpoint se lleva a cabo definiendo rutas de acceso de la carpeta routes/. De forma predeterminada laravel dispone de los archivos *console.php* y *web.php; console.php* sirve para definir comandos de consola personalizados, *web.php* sierve como punto de acceso para solicitudes web tradicionales de la interfaz de usuario creada con laravel (teniendo en cuenta su enfoque como framework full stack). Ambos archivos no son utlizadfos en este proyecto. El eltrada a esta API REST depende del archivo api.php, allí se definen, para comenzar las rutas de acceso a *maps*.

- Archivo api.php

```
//api.php
require __DIR___. '/admin.php';
require __DIR__. '/admin-auth.php';
require __DIR__. '/authorized.php';
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\MapController;
Route::prefix('maps')->group(function () {
    Route::get('/',[MapController::class,'index']);
    Route::get('/{id}',[MapController::class,'show']);
    Route::post('/',[MapController::class,'store']);
    Route::put('/{id}',[MapController::class,'update']);
    Route::patch('/{id}',[MapController::class,'updatePartial']);
    Route::delete('/{id}',[MapController::class,'destroy']);
});
```

Algo para destacar en esta implementación es que:

- *Se* utiliza require '*__DIR__*'. para referencias los archivos que también hacen parte de esta API de lo contrario no funcionarán.

- Hay que importar (*require*) y referenciar el nombre de la clase controladora utilizada para procesar los datos asociados a cada endpoint.

- *Importar route*, librería para crear las rutas.

- Se nombra cada método de la clase que se encargara de gestionar los datos de cada endpoint.
- Se *define* el método HTTP de API REST con el cual se estandariza el comportamiento de cada endpoint, como se muestra en la Tabla 9.

Posteriormente, se definen las rutas asociadas a las tablas restantes en archivos separados así: *'authorization' (authorization.php), 'admin' (admin.php)y 'admin-auth'(admin-auth.php).* Así como en el fronted, se ha han repartido archivos, ambos con acceso a la tabla *'admin'* con el fin de ordenar mejor las responsabilidades y dejar aquellas ligadas a la autenticación en *admin-auth.php.*

- Archivo *authorized.php:*

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AuthorizationController;
Route::prefix('authorized')->group(function () {
    Route::get('/', [AuthorizationController::class, 'index']);
//Consulta general
    Route::post('/', [AuthorizationController::class, 'create']); //
Crear un registro
    Route::delete('/{id}', [AuthorizationController::class, 'delete']);
// Eliminar un registro
```

- Archivo *admin.php:*

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AdminController;
Route::prefix('admin')->group(function () {
    Route::get('/', [AdminController::class, 'index']);
    Route::post('/', [AdminController::class, 'store']); //crear
registro
    Route::get('/{id}', [AdminController::class, 'show']);
    Route::patch('/{id}', [AdminController::class, 'updatePartial']);
});
```

- Archivo *admin-auth:*

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AdminAuthController;

Route::prefix('adminlog')->group(function () {
    Route::post('/login', [AdminAuthController::class, 'login']);
    Route::post('/logout', [AdminAuthController::class, 'logout']);
    Route::get('/me', [AdminAuthController::class, 'me']);
    Route::patch('/updatepassword', [AdminAuthController::class,
'updatePassword']);
    Route::delete('/{id}', [AdminAuthController::class, 'destroy']);
});
```

Implementación de Controladores

Los archivos de los controladores se han dispuesto en relación directa uno a uno con los archivos de *routes* y modelo, a excepción del modelo *Admin* que naturalmente, hace uso de dos controladores. Cada uno de los archivos controladores codifican una clase heredada de Controller (una clase interna de Laravel). Esta clase de denomina controlador. Esto proporciona funcionalidades como *Middleware* (filtro de lógica personalizada) que se pueden utilizar en el constructor de la clase, métodos de manejo de autorizaciones, trabajos en cola, validaciones, manejo de *Requests*, entre otros.

Para crear un controlador, por ejemplo, el archivo *mapController* se aconseja dedicar una carpeta llamada *'Api'*, para ello *artisan* provee el siguiente comando:

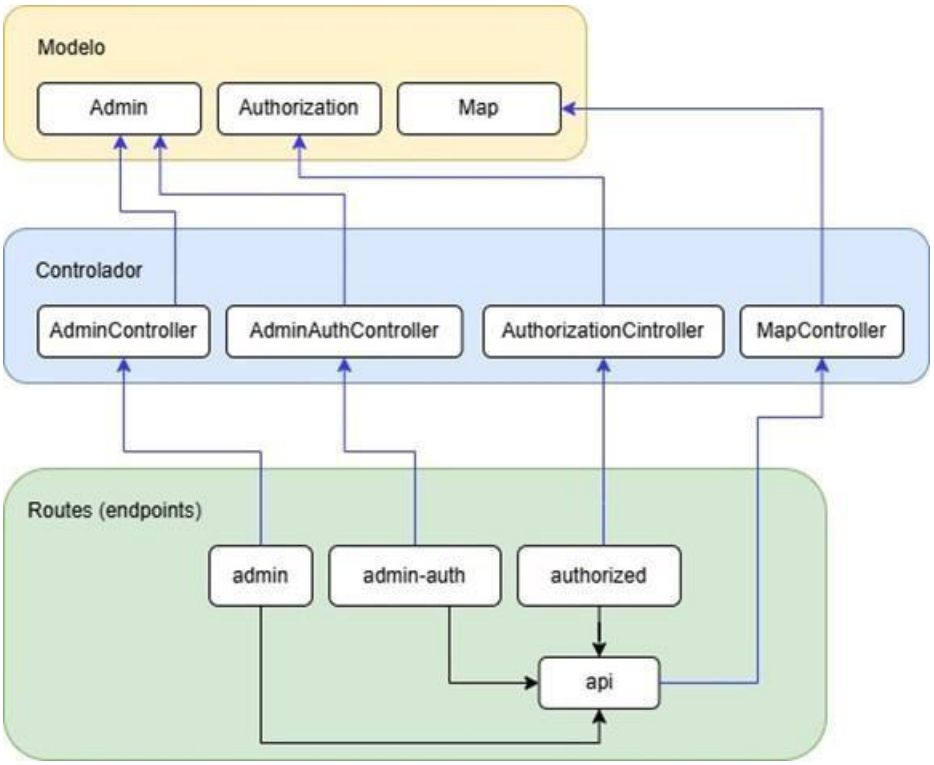
```
php artisan make:controller Api/mapController
```

La ejecución de este comando crea la carpeta *app/Http/Controllers/Api* y el archivo *mapController.php.*

La disposición de los controladores y su relación con otros modulos puede verse en la Figura 22.

Figura 22

Relación de Controladores con Modelos y Endpoints



Nota. Autoría propia

El funcionamiento de un controlador consiste en definir los métodos de clase consumidos en los endpoint. En general estos métodos reciben los datos que vienen de sus respectivos endpoints y proveen la lógica necesaria para consultar la base de datos, por ello utilizan un '*require'* que apunta a los modelos necesarios mediante Eloquent ORM, pasando por una validación y lógica de datos según la lógica de negocio y formatean la respuesta que, enviada de nuevo al cliente, lo cual incluye cuerpo de la respuesta en JSON y código de respuesta como se explica en la Tabla 10.

En la Tabla 11 y la Tabla 12 se recopila la información de los controladores, sus métodos de clase y su relación con los modelos y endpoints.

Tabla 11

Controladores-Descripción

Clase	Método	Descripción
MapController	index	Consultar todos los registros de la tabla 'map'
MapController	show	Consultar un registro por 'id' de 'map'
MapController	store	Crear un registro en 'map'
MapController	update	Sobreescribir un registro buscado por 'id', en 'map',
MapController	updatePartial	Actualizar parcialmente un registro ‘map’ buscado por 'id'
MapController	destroy	Buscar por 'id' un registro en 'map' y eliminarlo
MapController	getMapsByRequest	Método estático de clase
AdminController	index	Consultar los administradores
AdminController	show	Consultar un administrador por 'id'
AdminController	store	Registrar administrador, requiere consultar tabla 'authorization'
AdminController	updatePartial	Actualizar los datos de un administrador por 'id'
AdminController	update	Método en disponibilidad
AdminAuthController	login	Método para iniciar sesión
AdminAuthController	logout	Método para cerrar sesión
AdminAuthController	me	Verificar sesión en el servidor
AdminAuthController	updatePassword	Actualizar contraseña de administrador por 'id'
AdminAuthController	destroy	Eliminar administrador por 'id'
AuthorizationController	index	Consultar todas las autorizaciones
AuthorizationController	create	Crear una autorización
AuthorizationController	delete	Eliminar una autorización por 'id'

Not. Esta tabla muestra los controladores creados, los métodos que utilizan y la descripción de cada uno de ellos. Autor

Tabla 12

Relación Entre Métodos Controladores, Modelos y Endpoints

Clase	Método	Modelo	Endpoint
MapController	index	Map	get: /api/maps/
MapController	show	Map	get: /api/maps/{id}
MapController	store	Map	post: /api/maps/
MapController	update	Map	put: /api/maps/{id}
MapController	updatePartial	Map	patch: /api/maps/{id}
MapController	destroy	Map	delete: /api/maps/{id}
MapController	getMapsByRequest	Map	
AdminController	index	Admin	get: /api/admin/
AdminController	show	Admin	get: /api/admin/{id}
AdminController	store	Admin-Authorization	post: /api/admin/
AdminController	updatePartial	Admin	patch: /api/admin/{id}
AdminController	update	Admin	
AdminAuthController	login	Admin	post: /api/adminlog/login
AdminAuthController	logout	Admin	post : /api/adminlog/logout
AdminAuthController	me	Admin	get: /api/adminlog/me
AdminAuthController	updatePassword	Admin	patch: /api/adminlog/{id}
AdminAuthController	destroy	Admin	delete: /api/adminlog/{id}
AuthorizationController	index	Authorization	get: /api/authorized/
AuthorizationController	create	Authorization	post: /api/authorized/
AuthorizationController	delete	Authorization	delete: /api/authorized/{id}

Nota. Esta tabla muestra los controladores creados, los métodos que utilizan y la descripción de cada uno de ellos, completando la información de la Tabla 11. Autor

A continuación, se muestra la codificación que resultó menos trivial dada la lógica de negocio involucrada: El método controlador *store* de la clase *AdminController*:

```
public function store(Request $request)
    {
        try {
            // Validar los datos proporcionados (sin incluir 'main')
            $validator = Validator::make($request->all(), [
                'name' => 'required',
                'no_doc' => 'required|unique:admin,no_doc',
                'email' => 'required|email|unique:admin,email',
                'username' => 'required|unique:admin,username',
```

```
                'password' => 'required|string|unique:admin,password',
            ]);

            if ($validator->fails()) {
                return response()->json([
                    'message' => 'Error en la validación de los
datos',
                    'errors' => $validator->errors(),
                    'status' => 400
                ], 400);
            }

            // Verificar si no hay registros en la tabla 'admin'
            if (!Admin::exists()) {
// Si no hay administradores, el primero será 'main' = true
                $admin = Admin::create([
                    'name' => $request->name,
                    'no_doc' => $request->no_doc,
                    'email' => $request->email,
                    'username' => $request->username,
                    'password' => Hash::make($request->password),
                    'main' => true
                ]);

                return response()->json([
                    'admin' => $admin,
                    'status' => 201
                ], 201);
            }

// Obtener el valor de 'main' desde la tabla 'authorization'
$authorization = Authorization::where('no_doc', $request->no_doc)
                ->where('email', $request->email)
                ->first();

            if (!$authorization) {
                return response()->json([
'message' => 'El documento o email no están autorizados.',
                    'status' => 403
                ], 403);
            }

            // obtener main de authorization
            $main = $authorization->main ?? false;
```

```
            // Crear el nuevo registro admin
            $admin = Admin::create([
                'name' => $request->name,
                'no_doc' => $request->no_doc,
                'email' => $request->email,
                'username' => $request->username,
                'password' => Hash::make($request->password),
                'main' => $main
            ]);

            return response()->json([
                'admin' => $admin,
                'status' => 201
            ], 201);

        } catch (\Exception $e) {
            return response()->json([
                'message' => 'Error al procesar la solicitud.',
                'error' => $e->getMessage(),
                'status' => 500
            ], 500);
        }
    }
```

Este método sirve al endpoint *post: /api/admin/* para registrar un administrador. La lógica de negocio que explica su implementación es la siguiente:

- Su función principal es registrar un nuevo administrador.
- Los campos recibidos desde el fronted son *'name', 'no_doc', 'email', 'username', 'password'* y deben cumplir las respectivas validaciones.
- Se condiciona que si no hay administradores registrados (la tabla *'Admin'* está vacía) entonces se añadirá el campo 'main' con valor true.
- Si hay administradores registrados, entonces el valor 'main' a guardar, debe ser el que corresponde con *'no_doc'* y *'email'* en la tabla *'Authorization'* según los proporcionados en la

solicitud (*$request*). Esto mantiene la consistencia puesto que *'no_doc'* y *'email'* son valores únicos en la tabla *'Authorization'*.

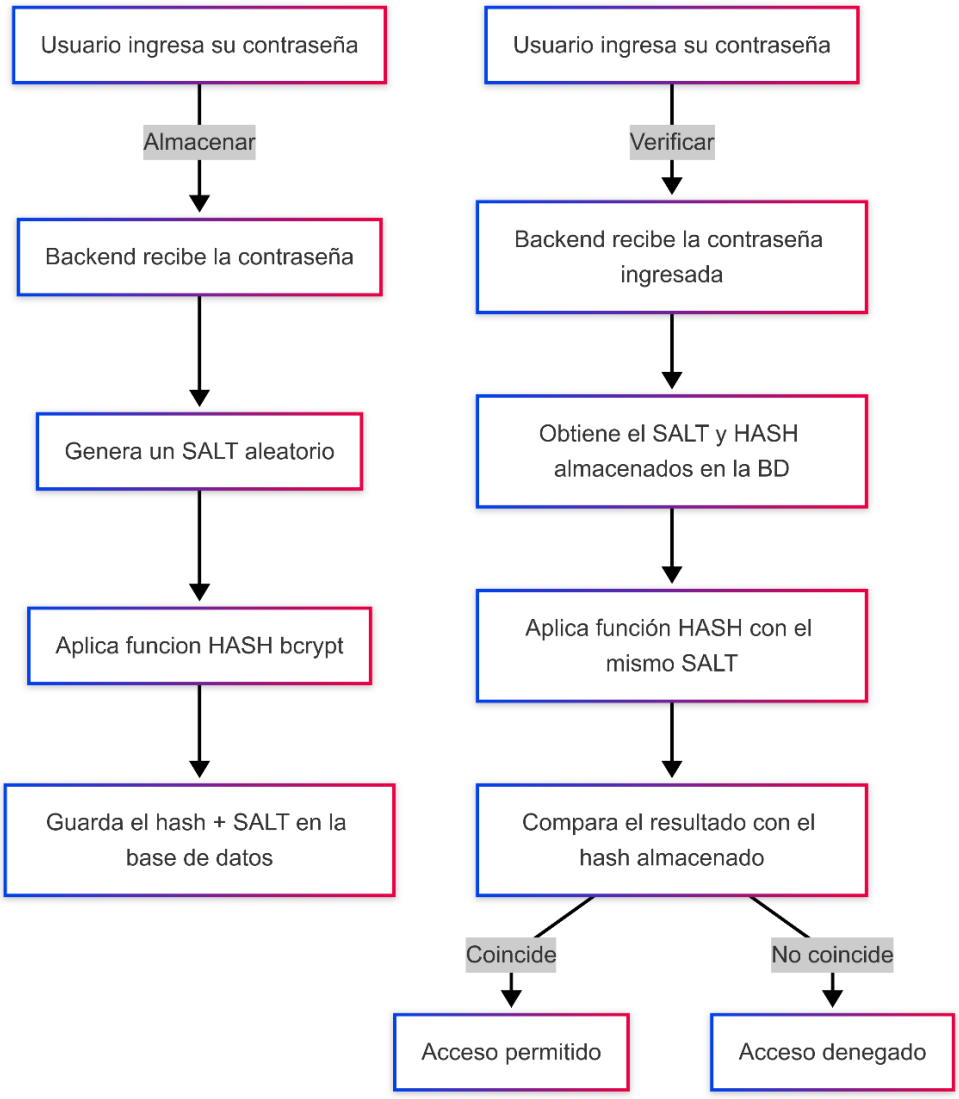
- Dada la condición anterior, este método hace consumo de la instancia de clase Authorization.

Un tema destacable en la implementación de los controladores 'AdminAuthController' y 'AdminController' es el uso del cifrado Hash a la hora de guardar la contraseña en la base de datos pues por seguridad esta debe ser cifrada y a la hora de comprobar una contraseña enviada desde el fronted con una contraseña cifrada y guardada lo cual es requerido en el cambio de contraseña o actualización de datos. Para esto es necesario el uso de lirberia hash la cual se llama en los respectivos archivos mediante *'use Illuminate\Support\Facades\Hash'*. La Figura 23 muestra este proceso en dos partes:

- Almacenar la contraseña: Se usa un algoritmo HASH con un SALT aleatorio para mejorar la seguridad antes de guardarla en la base de datos.
- Verificar la contraseña: Se extrae el SALT y el HASH almacenados, se aplica la misma función de hash a la contraseña ingresada y se compara el resultado con el hash guardado.

Figura 23

Cifrado Hash de Contraseñas



Nota. Autoría propia

El algoritmo *Bcrypt* es un método de *hashing* diseñado específicamente para almacenar contraseñas de manera segura. Se basa en el cifrado *Blowfish* y tiene características que lo hacen resistente a ataques de fuerza bruta.

Una de sus principales fortalezas es el uso de Salt, que es un valor aleatorio agregado a la contraseña antes de aplicar el *hashing*. Esto evita que dos contraseñas idénticas generen el mismo hash, lo que protege contra ataques con tablas de búsqueda precomputadas (*rainbow* tables).

Para consultar el código completo de estos controladores puede examinarse el repositorio remoto de mi autoría en *app/Http/Controllers/Api*.

Implementación de JWT

Funcionamiento de JWT. JWT estandariza la autenticación y el intercambio seguro de información entre un cliente (frontend) y un servidor (backend) sin necesidad de mantener sesiones en la base de datos. Básicamente un token JWT es un objeto que contiene secuencias de datos encriptados.

JWT en la autenticación funciona de la siguiente forma:

- Inicio de sesión: Administrador envía sus credenciales desde el fronted.
- Generación del token: El backend verifica las credenciales y genera un JWT firmado con una clave secreta.
- Entrega del token: El JWT se envía al frontend, que lo almacena en una cookie de forma segura.
- Acceso a recursos protegidos: En cada solicitud protegida, el frontend envía el token en los encabezados (Authorization: Bearer TOKEN).
- Verificación del token: El backend decodifica el JWT usando la clave secreta (JWT_SECRET) y verifica su validez. Esta clave secreta suele establecerse dentro del archivo *.env* (variables de entorno) del backend.

- Acceso concedido o denegado: Si el token es válido, el backend responde con los datos protegidos; si no, devuelve un error 401 (No autorizado).

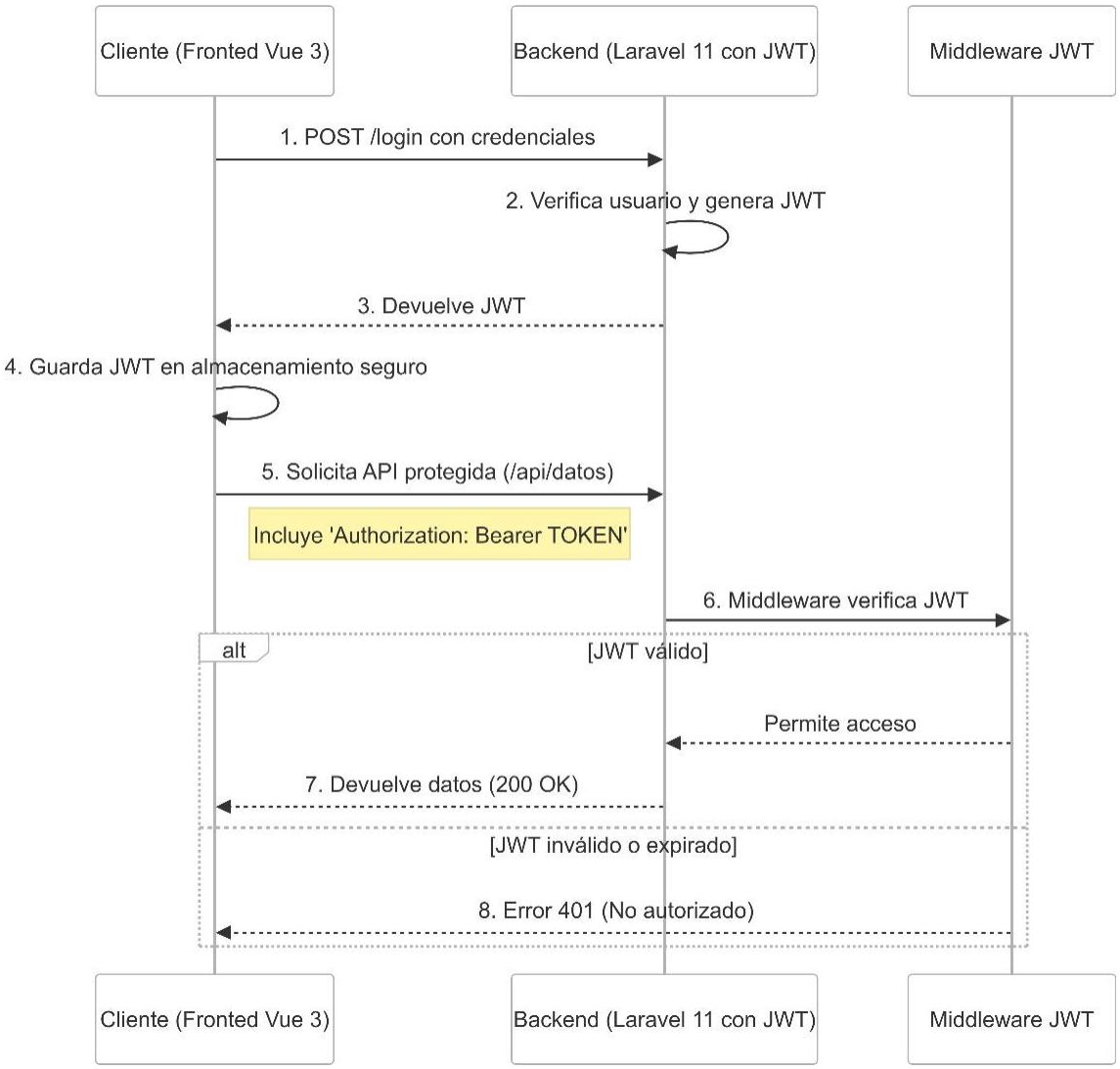
Algunas de las ventajas de utilizar JWT son:

- *No necesita sesiones en la base de datos.*
- *Seguridad basada en firma y expiración del token.*
- *Rápido* y ligero, ideal para API REST.

En la Figura 24 puede verificarse el ciclo de vida del token en laravel 11 y muestra como es generado, verificado y consumido el token, además los filtros Middleware JWT verifican los tokens enviados por el fronted hacia las rutas protegidas para dar o no acceso a los recursos.

Figura 24

Implementación de JWT en Laravel 11



Nota. Archivo de mi autoría

Instalación y Configuración Inicial. Más allá de la codificación, otras configuraciones son necesarias durante la creación del API para preparar su funcionamiento. Se describen los siguientes pasos:

- Establecer la configuración de conexión a la base de datos.

En el archivo archivo *.env* se digitan las configuraciones siguientes:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=proy
DB_USERNAME=root
DB_PASSWORD=
```

- Habilitación de la API y la excepción de autenticación de actualización

De forma predeterminada, la ruta API de laravel 11 no está habilitada en laravel 11. Se debe habilitarla:

```
php artisan install:api
```

Como esta ha sido ejecutada para la elaboración de API REST, solo resta actualizar a excepción de autenticación de nuestro middleware de API para que no redirija al inicio de sesión, sino que arroje una excepción, esto se realiza en el archivo ***bootstrap/app.php***:

```
<?php
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;
use Illuminate\Auth\AuthenticationException;
use Illuminate\Http\Request;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        //
    })
    ->withExceptions(function (Exceptions $exceptions) {
$exceptions->render(function (AuthenticationException $e, Request $request) {
            if ($request->is('api/*')) {
                return response()->json([
                    'message' => $e->getMessage(),
                ], 401);
            }
```

```
        });
    })->create();
```

- Instalación y configuración del paquete de autenticación JWT.

Se ejecutan los siguientes comandos para extraer la versión más reciente y publicar la configuración:

- composer `require php-open-source-saver/jwt-auth`
- `php artisan vendor:publish --provider="PHPOpenSourceSaver\JWTAuth\Providers\LaravelServiceProvider"`

Genere una clave secreta. Esto agregará valores de configuración JWT en el archivo *.env*:

`php artisan jwt:secret`

Actualización de la configuración de protección de autenticación en el archivo *config/auth.php*

```
'defaults' => [
        'guard' => 'api',
        'passwords' => 'users',
    ],
 'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],

        'api' => [
            'driver' => 'jwt',
            'provider' => 'users',
        ],
    ],
```

- Configuración de tiempo de una hora de duración de validez del token (duración de la sesión)

En el archivo *config/jwt.php* verificar que *'ttl' => env('JWT_TTL',60).*

Usos de JWT. Este apartado explica la implementación de los tres usos principales que se le dan en este proyecto de backend al JWT:

- Actualización del modelo *'Admin'* para envío de información al Fronted dentro del token.

Implementación del contrato *Tymon\JWTAuth\Contracts\JWTSubject* en el modelo e implementación de los métodos *getJWTIdentifier* y *getJWTCustomClaims* en *app/Models/Admin.php*

```
public function getJWTIdentifier()
    {
        return $this->getKey();
    }

    /**
     * Devolver un array con cualquier información personalizada que quieras
agregar al token.
     */
    public function getJWTCustomClaims()
    {
        return [
            'main'=>$this->main,
            'name'=>$this->name,
            'no_doc'=>$this->no_doc,
            'email'=>$this->email,
            'username'=>$this->username
        ];
    }
```

Este par de métodos se han añadido al final de la clase *Admin* en el archivo *Admin.php* con el fin darle uso del JWT

La función *getJWTIdentifier* es requerida por la librería JWTAuth para devolver el identificador único del usuario que se usa en los tokens JWT.

```
public function getJWTIdentifier()
{
    return $this->getKey();
}
```

getKey() devuelve la clave primaria (id) del usuario. Cuando un usuario inicia sesión, el token incluirá este identificador. Por ejemplo, si admin_id = 1, el token contendrá un json así:

```
{
    "sub": 1
}
```

getJWTCustomClaims se encarga de hacer que información adicional de este modelo se incluya en el token, por lo tanto cuando el token sea obtenido en el Fronted tendrá acceso a la información *'main', 'name', 'no_doc', 'email', 'username'* del respectivo administrador.

- Validaciones de información en el backend – actualización del modelo controlador '*AdminAuthController*'.

Este modelo hace varios usos de JWT para:

- Generar un token para el administrador que inicia sesión, función *'login'*.
- Invalidar el token una vez que se ha cerrado sesión, función *'logout'*.
- Validar el token enviado desde el Fronted, función 'me'.
- Validar autenticación para permitir el cambio de contraseña, función '*updatePassword*'.
- Verificar la correspondencia de la referencia de usuario enviado por la id con el token de la sesión actual para permitir el borrado de registro de administrador, función *'destroy'*.

- Protección de rutas.

Algunas rutas deben ser protegidas con un middleware que exige que el token sea enviado desde el fronted como primer requisito antes de hacer uso del controlador. Este es *'middleware('jwt.auth')'* y se implementa en los endpoints definidos en la carpeta routes/ como este código no es extenso se colocará explícitamente a continuación.

En el archivo *api.php* que contiene los endpoints relacionados con consultas a *'map'* como en el archivo *admin.php y admin-auth.php* que contienen los endpoints relacionados con consultas a *'admin'* se ha incluido el middleware solo en las rutas que son siempre privadas (se consumen en panel de administrador del Fronted).

```
<?php
//api.php
require __DIR__. '/admin.php';
require __DIR__. '/admin-auth.php';
require __DIR__. '/authorized.php';
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\MapController;

Route::prefix('maps')->group(function () {
    Route::get('/',[MapController::class,'index']);
    Route::get('/{id}',[MapController::class,'show'])-
>middleware('jwt.auth');
    Route::post('/',[MapController::class,'store'])->middleware('jwt.auth');
    Route::put('/{id}',[MapController::class,'update'])-
>middleware('jwt.auth');
    Route::patch('/{id}',[MapController::class,'updatePartial'])-
>middleware('jwt.auth');
    Route::delete('/{id}',[MapController::class,'destroy'])-
>middleware('jwt.auth');
});


<?php
//admin.php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AdminController;

Route::prefix('admin')->group(function () {
    Route::get('/', [AdminController::class, 'index'])-
>middleware('jwt.auth');
    Route::post('/', [AdminController::class, 'store']);
    Route::get('/{id}', [AdminController::class, 'show'])-
>middleware('jwt.auth');
    Route::patch('/{id}', [AdminController::class, 'updatePartial'])-
>middleware('jwt.auth');
});


<?php
//admin-auth.php
```

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AdminAuthController;

Route::prefix('adminlog')->group(function () {
    Route::post('/login', [AdminAuthController::class, 'login']);
    Route::post('/logout', [AdminAuthController::class, 'logout'])-
>middleware('jwt.auth');
    Route::get('/me', [AdminAuthController::class, 'me'])-
>middleware('jwt.auth');
Route::patch('/{id}', [AdminAuthController::class, 'updatePassword'])-
>middleware('jwt.auth');
    Route::delete('/{id}', [AdminAuthController::class, 'destroy'])-
>middleware('jwt.auth');
});
```

Cuando todo el grupo de endpoints requiere el middleware se puede resumir su implementación definiéndola para todo el grupo como en el caso de *'authorized.php'* que provee consultas para la tabla *'authorization'*:

```
<?php
//authorized.php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AuthorizationController;

Route::prefix('authorized')->group(function () {
    Route::get('/', [AuthorizationController::class, 'index']); //Consulta
general
    Route::post('/', [AuthorizationController::class, 'create']); // Crear un
registro
    Route::delete('/{id}', [AuthorizationController::class, 'delete']); //
Eliminar un registro
})->middleware('jwt.auth');
```

Pruebas de Validación de Software

Se realizan pruebas funcionales siguiendo los casos de uso mencionados en las tablas 1 a la 8 y dejando evidencias del comportamiento del sistema. Estas pruebas son realizadas en entorno de desarrollo para comprobar todos los requisitos funcionales.

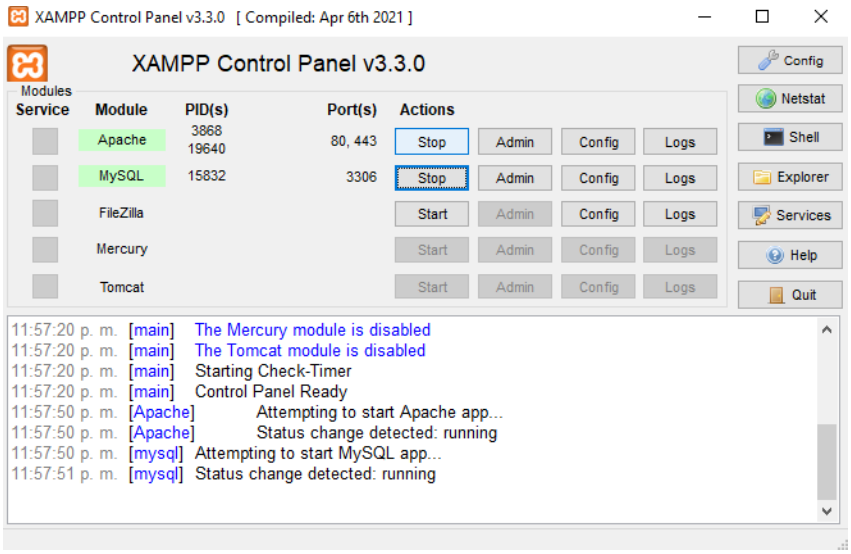
Prueba de Acceso al Sitio

Para probar el aplicativo, se procede a arrancar los servidores de desarrollo correspondientes a Fronted, Backend, Apache y MySql.

Primero se procede a activar Apahe y MySQL en XAMP. Es necesario que este sea el primer paso pues de esto depende el funcionamiento del Backend, el cual a su vez es necesario arrancar primero para que el fronted tenga datos con los cuales funcionar (ver Figura 25).

Figura 25

Activación de Apache y MySQL



Nota. Autoría propia.

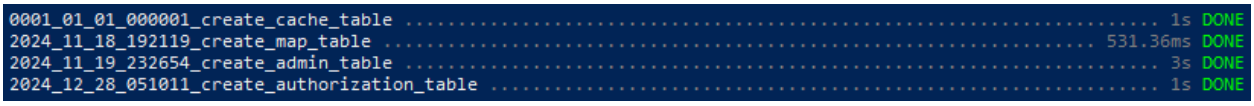
El entorno de desarrollo es visual studio code, para arrancar el servidor de backed se ingresa en la terminal al correspondiente directorio de la carpeta de desarrollo: *'\proy\'* y se

ejecuta el comando ‘*php artisan serve’,* esto responde con el mensaje *‘Server running on [http://127.0.0.1:8000]’*. Para asegurar que la base de datos esté vacía, en este mismo directorio se ejecuta el comando *‘php artisan migrate:fresh’*, esto elimina todas las tablas y vuelve a ejecutar las migraciones. La respuesta a la ejecución de este comando finaliza con la confirmación de creación de las tablas. Se incluye la tabla llamada “cache”, la cual se incluye en la instalación de la librería externa *tymon-jwt*. Esta tabla contiene los campos *key*, *value* y *expiration.* La configuración de esta librería dictamina que en esta tabla se guarde una lista negra de los tokens que han sido revocados, por seguridad y por lo tanto es actualizada cada que se cierra sesión.

Puede verse la ejecución de migraciones (lo que crea las tablas en la base de datos) en la Figura 26.

Figura 26

Ejecución de Migraciones



Nota. Autoría propia.

A continuación, en otra ventana de VS Code, se ubica el directorio correspondiente al proyecto ‘\dashboards_dr\’ en la terminal para ejecutar Fronted. Se escribe el comando ‘*npm run dev’,* como resultado esto arroja una *url* que debe ser direccionada en un navegador web que se esté usando para el desarrollo, en este caso *‘mozilla firefox*’: ‘*Local: http://localhost:5173/’*.

Este sitio carga vacío, puede verse que redirecciona al subdominio ‘MainV’, lo cual es el comportamiento esperado según la configuración dada el en archivo ‘/*router/index.ts’.* Además, excepto por el icono de menú hamburguesa, carga vació, lo que es correcto al no haber datos

para exponer en la base de datos. Esto puede comprobarse también porque la terminal donde se ejecuta el servidor del backend muestra que se ha ejecutado la consulta de cuadros ‘map’ al API. Se aprecian dos peticiones cuando solo se ha hecho una consulta, esto es porque la primera de ellas se denomina presolicitud la cual verifica que la conexión es válida. Ver Figura 27.

Figura 27

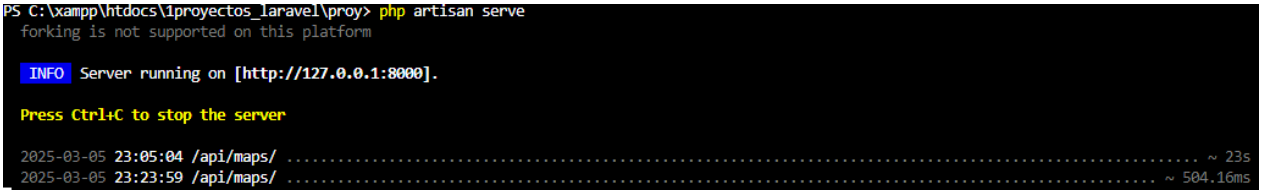
Acceso al Sitio

Nota. Autoría propia

En la Figura 28 se muestra la comunicación con el servidor durante el acceso al sitio para consumir el endpoint que permite acceder a los dashboards públicos.

Figura 28

Acceso al Sitio, Respuesta del API



Nota. Autoría propia.

Prueba de Registro del Primer Administrador

Según el flujo de datos, lo primero que es necesario es registrar el primer administrador (denominado *admin0*), el cual por defecto adquiere privilegios de administrador principal y con cuyos datos se realizan las pruebas de inicio de sesión, gestión de autorizaciones para registro de otros administradores, cerrado de sesión.

Para esto se utilizan los siguientes datos de prueba en el respectivo formulario de registro de administrador, ver Tabla 13:

Tabla 13

Datos del Primer Administrador

Nombres de campo	Dato
Nombre	Alejandra Rocio
No. documento	1021428759
Email	admin0@hotmail.com
Usuario	admin0
Contraseña	1021428759

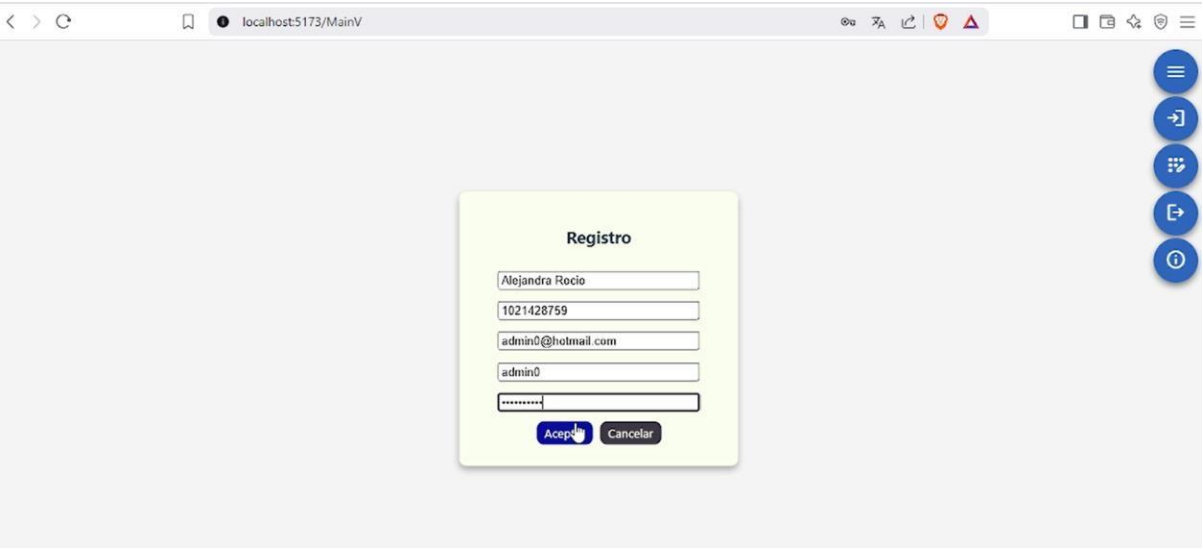
Nota. Esta tabla muestra los primeros datos de prueba utilizados con el rol de administrador.

Autor

En la evidencia a continuación puede verse el diligenciamiento del formulario de registro con estos datos(*admin0*), Ver Figura 29.

Figura 29

Formulario de Registro de Admin0

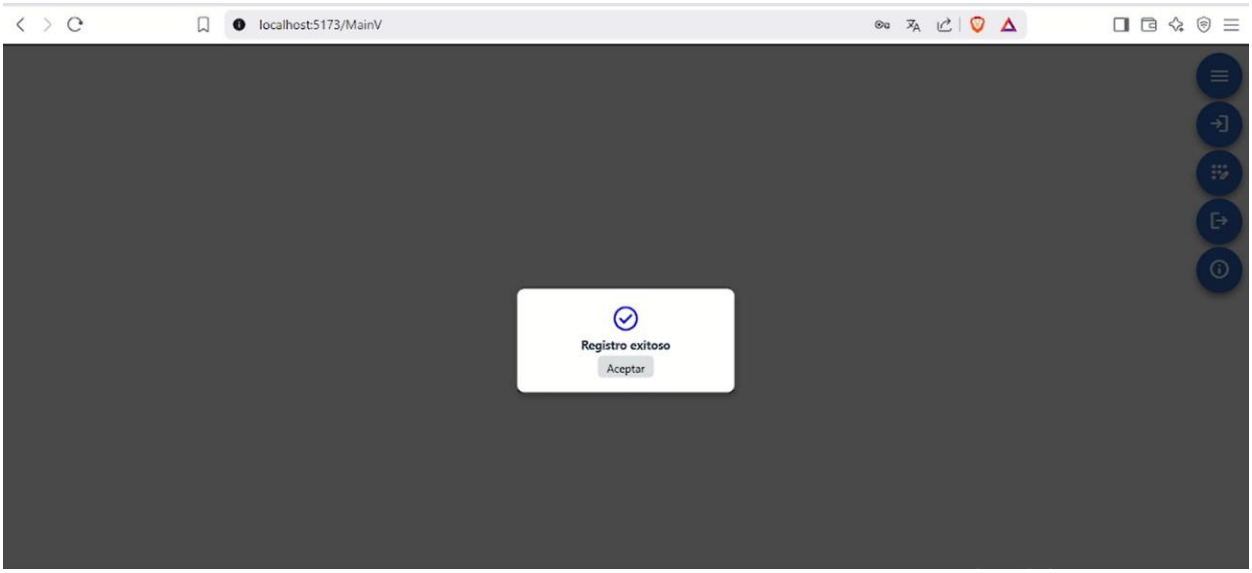


Nota. Autoría propia

El sitio anuncia mediante un *popup* de alerta, la creación exitosa del registro. Esto se evidencia en la Figura 30.

Figura 30

Popup Alerta de Creación del Registro

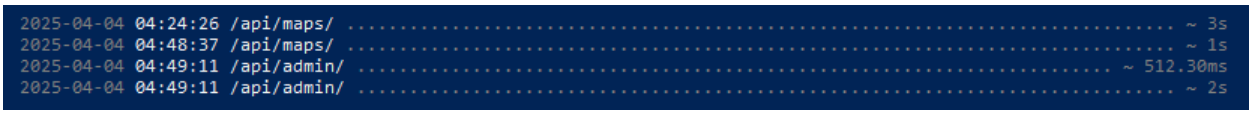


Nota. Autoría propia

Cuando se presiona aceptar (figura 29), se ejecuta asincrónicamente la conexión con el API REST para enviar los datos al respectivo *endpont* como se evidencia en la Figura 31.

Figura 31

Evidencia Conexiones al Endpoint de Registro



Nota. Autoría propia

Finalmente, se evidencia que como resultado se guarda en la base de datos diligenciados, nótese la inclusión de un índice numérico autoincremental que comienza en 1, la contraseña cifrada y los datos adicionales dele modelo gestionados por el backend que guardan el momento

de creación y actualización del registro, para la obtención de esta evidencia se ejecuta la consulta SQL: *SELECT* * *FROM `admin`*. Ver Figura 32.

Figura 32

Consulta SQL de Admin0

id	name	no_doc	email	username	password	main	created_at	updated_at
1	Alejandra Rocio	1021428759	admin0@hotmail.com	admin0	$2y$12$DXt9EgFDt.D44lGGfV.ek.dvfkpcR.aselb0hgZAen5...	1	2025-04-04 09:49:13	2025-04-04 09:49:13

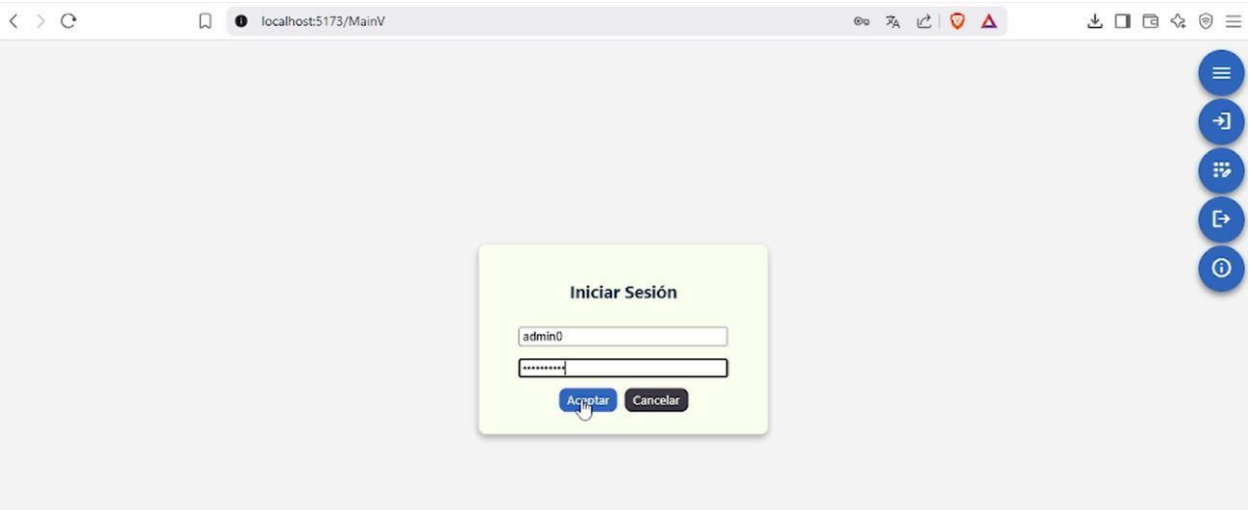
Nota. Autoría propia

Prueba de Inicio de Sesión

Esta prueba es realizada con los anteriores datos (usuario: admin0 y contraseña: 1021428759). Adicionalmente se comprueba que este administrador cuenta con todos los privilegios (puede gestionar autorizaciones). Para la evidencia del diligenciamiento del formulario de inicio de sesión, ver Figura 33.

Figura 33

Diligenciamiento de Formulario de Inicio de Sesión



Nota. Autoría propia

Al enviar este formulario, se ejecutan las respectivas conexiones a endpoint, esto se evidencia en la Figura 34.

Figura 34

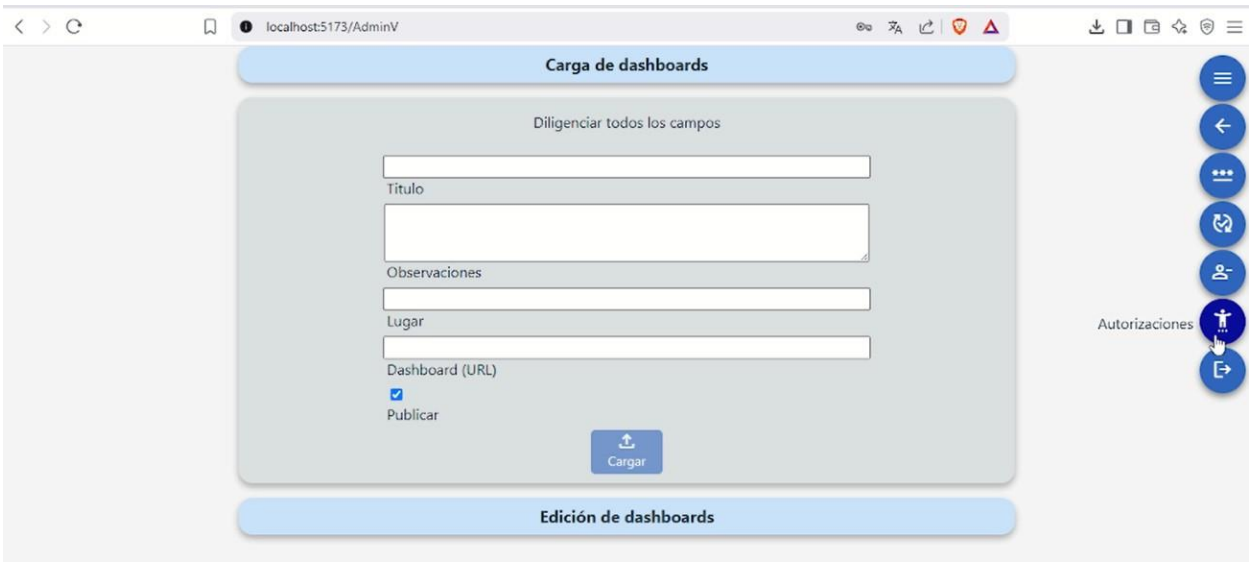
Conexiones al Endpoint de Login

Nota. Autoría propia

Como resultado del *login* exitoso, la aplicación ingresa al panel de administrador (subdominio *AdminV*). Al navegar por el menú, se evidencia también la presencia de las opciones, incluido Autorizaciones, lo que constata los privilegios de *admin0.* La evidencia de esto se muestra en la Figura 35.

Figura 35

Panel de Administrador de Admin0



Nota. Autoría propia

Prueba de Otorgamiento de Autorizaciones

En esta prueba admin0 ingresa a la opción de autorizaciones y diligencia autorizaciones para el registro de dos nuevos administradores. Los datos que se diligencian se muestran en la Tabla 14.

Tabla 14

Datos de Autorizaciones

Nombres de campo	Dato
No. documento	1088822
Email	admin1@gmail.com
Admin Principal	true
No. documento	10237455
Email	Admin2@gmail.com
Admin Principal	false

Nota. Esta tabla muestra los datos registrados como primera autorización. Autor

Se evidencia el diligenciamiento de los datos de autorización de *admin1* que cuenta con privilegios de administrador principal en la Figura 36.

Figura 36

Diligenciamiento de autorización de admin1

Nota. Autoría propia

Se evidencia el diligenciamiento de los datos de autorización de *admin2*, que no cuenta con privilegios de administrador principal en la Figura 37.

Figura 37

Diligenciamiento de Autorización de Admin2

Nota. Autoría propia

Se evidencian las peticiones a los respectivos *endpoint* cada que se envían las autorizaciones. Nótese también una petición al *endpoint 'maps'* pues se sincronizan los mapas que deberían mostrarse en el fondo del panel de administrador y que de momento no existen. Ver Figura 38.

Figura 38

Peticiones a Endpoint de Autorizaciones

Nota. Autoría propia

Cuando estas peticiones son procesadas por el *backend* se guardan los datos en la mase de datos *'proy'*, esto se comprueba mediante la consulta SQL: *SELECT * FROM `authorization`*. Véase la evidencia en la Figura 39.

Figura 39

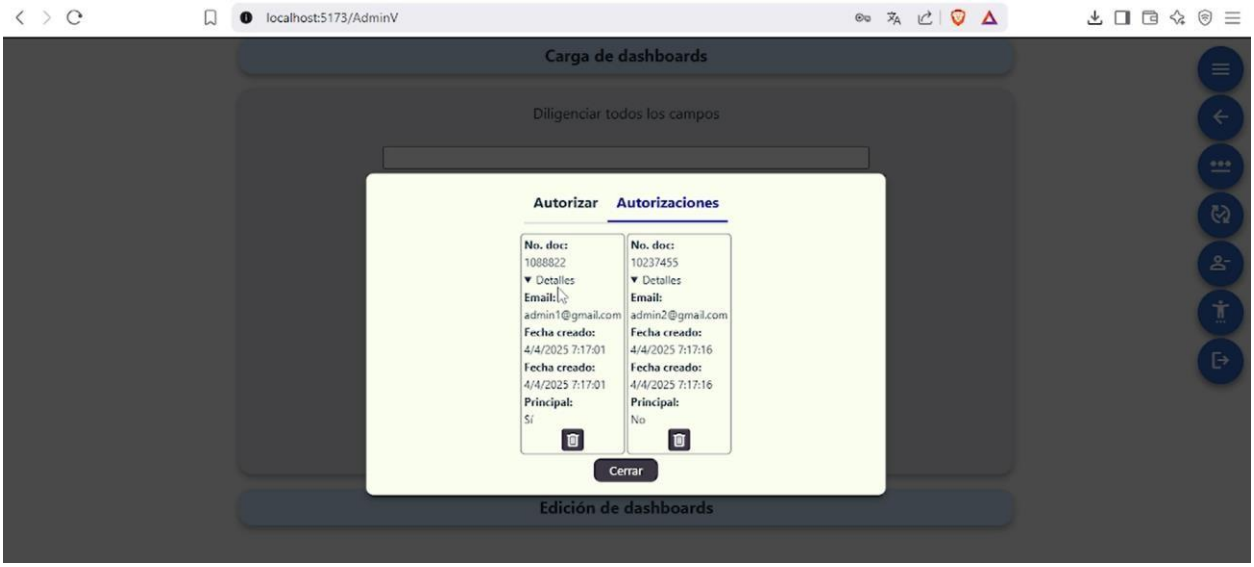
Resultado de Consulta de Tabla 'Authorization'

id	no_doc	email	main	created_at	updated_at
1	1088822	admin1@gmail.com	1	2025-04-04 12:17:01	2025-04-04 12:17:01
2	10237455	admin2@gmail.com	0	2025-04-04 12:17:16	2025-04-04 12:17:16

Nota. Autoría propia

Las dos autorizaciones creadas pueden ser consultadas y, dado el caso eliminadas, en la pestaña autorizaciones, como se evidencia en la Figura 40.

Figura 40 *Vista de Autorizaciones Creadas*



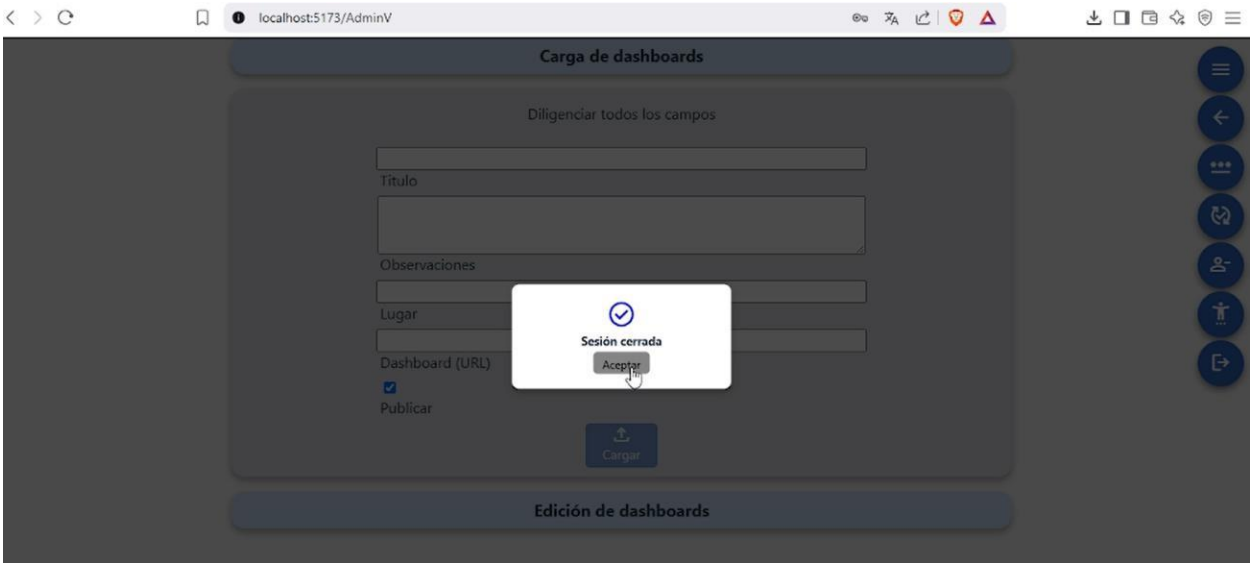
Nota. Autoría propia

Pruebas de Cierre de Sesión

Cuando se accede al menú y cierra la sesión, el sistema elimina el *JWT* y mediante la popup de alerta, da el resultado exitoso. Además, registra el token revocado y el registro temporal en la tabla ‘cache’ de la base de datos. Véase la vista de este resultado en la Figura 41.

Figura 41

Vista de Alerta de Sesión Cerrada



Nota. Autoría propia

Luego de mostrar esta alerta, la aplicación procede a redirigir el sitio al panel público (subdominio *MainV*). Como parte del proceso de cerrado de sesión, se evidencia en la Figura 42 la respectiva conexión al *endpoint*.

Figura 42

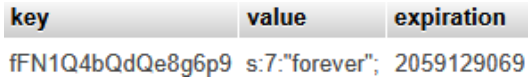
Evidencia de Conexión al Endpoint de Logout

Nota. Autoría propia

Se evidencia mediante la consulta SQL: *SELECT * FROM `cache`*, el registro del token revocado en la Figura 43.

Figura 43

Evidencia de conexión al endpoint de logout



key	value	expiration
fFN1Q4bQdQe8g6p9	s:7:"forever";	2059129069

Nota. Autoría propia

Pruebas de Registro y Cambio de Datos

Durante esta prueba se crean los registros para los *admin1* y *admin2* basándose en las autorizaciones creadas cuyos datos se encuentran en la Tabla 14 se accede al panel de administración con cada uno de ellos para comprobar sus privilegios. Posteriormente, con la sesión de *admin2*, se realizan cambios en algunos sus datos de registro y se cambia la contraseña, lo que cierra la sesión. Se comprueba el inicio de sesión con la contraseña nueva. Por último se procede a borrar la cuenta (lo que cierra la sesión). Los datos utilizados para las pruebas de registro en los respectivos formularios se evidencian en la Tabla 15.

Tabla 15

Datos Registro de Admin1 y Admin2

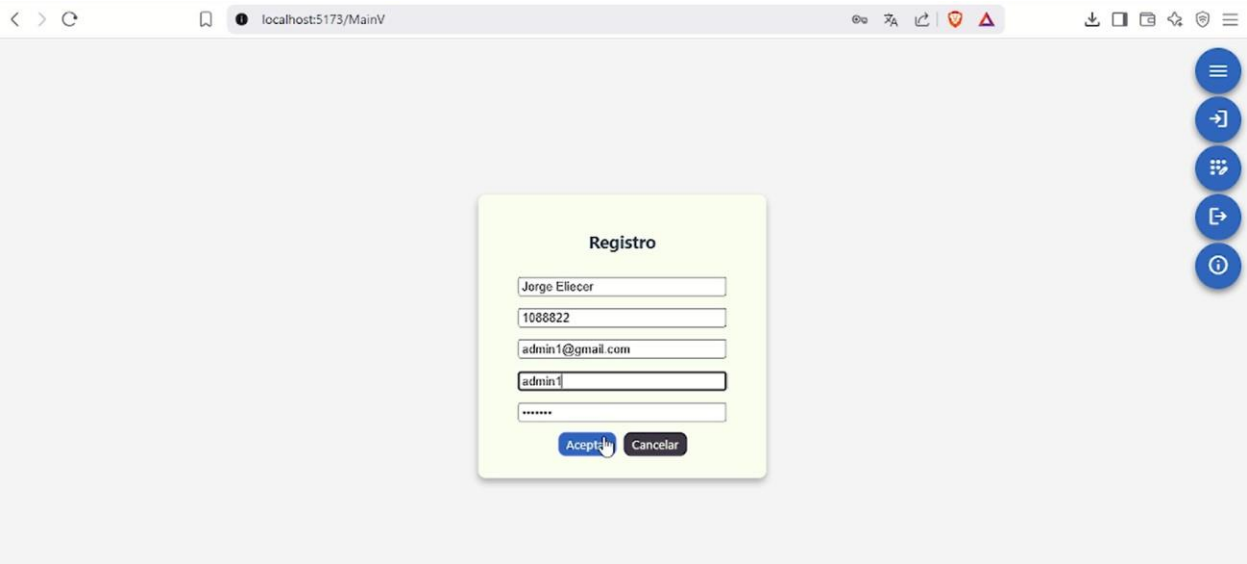
Nombres de campo	Dato
Nombre	Jorge Eliecer
No. documento	1088822
Email	admin1@gmail.com
usuario	admin1
contraseña	1088822
Nombre	Ana Maria
No. documento	10237455
Email	admin2@gmail.com
usuario	admin2
contraseña	10237455

Nota. Esta tabla muestra los datos de prueba creados durante el registro de los siguientes 2 administradores de ensayo. Autor

Se evidencia el llenado del formulario con los datos de admin1 en la Figura 44.

Figura 44

Prueba de Registro de Admin1

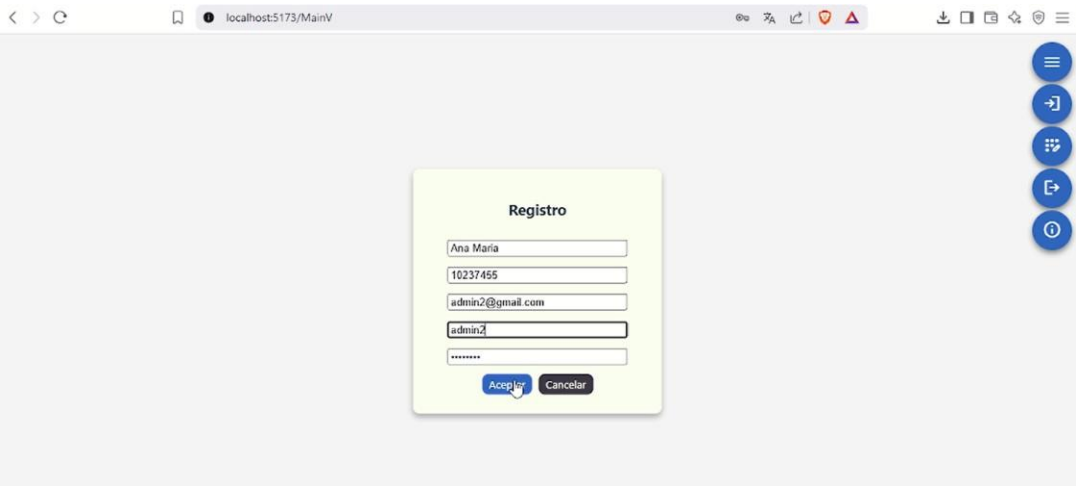


Nota. Autoría propia

Se evidencia el diligenciamiento del formulario con los datos de admin2 (Ver Figura 45).

Figura 45

Prueba De Registro de Admin2

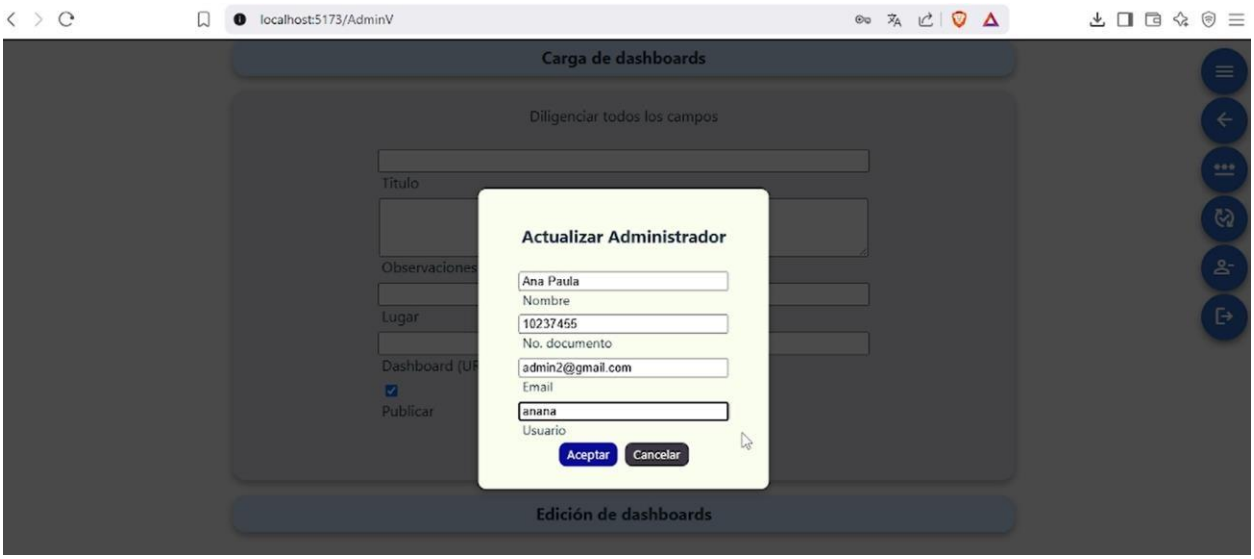


Nota. Autoría propia

Para la prueba de actualización de datos de administrador, es preciso iniciar sesión con el administrador cuyos datos serán cambiados, en este caso admin2. En este caso su nombre es cambiado a ‘Ana Paula’ y su usuario a ‘anana’. Se evidencia el diligenciamiento del respectivo formulario (accediendo en el panel de administrador), cambiando los datos de admin2 en la Figura 46.

Figura 46

Evidencia de Cambio de Datos de Admin2



Nota. Autoría propia

De forma similar, se accede al formulario de cambio de contraseña, en el cual exige la contraseña antigua y dos veces la contraseña nueva: ‘qwerty’. Esta prueba es evidenciada en la Figura 47.

Figura 47

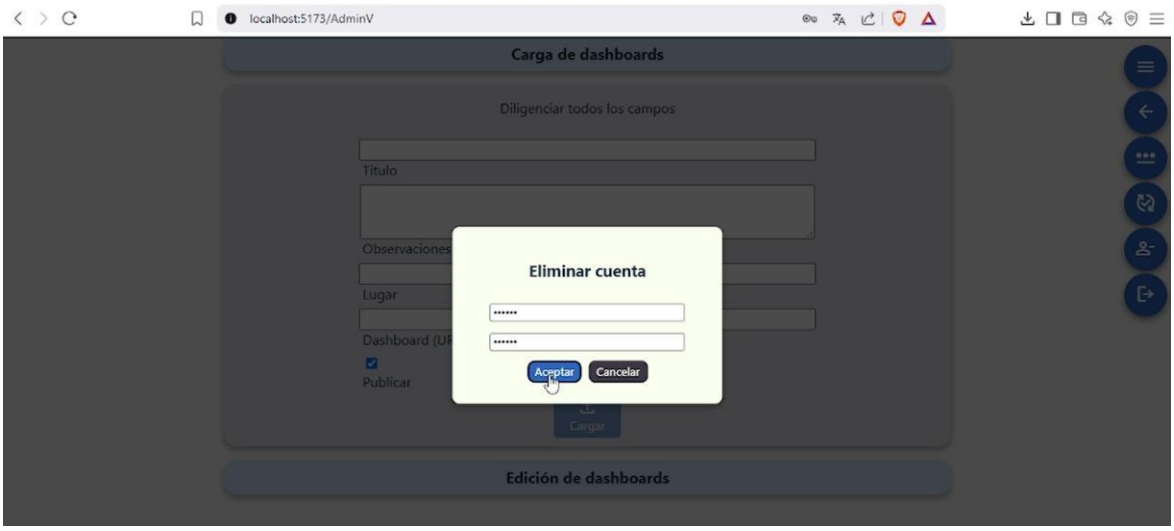
Cambio de Contraseña

Nota. Autoría propia

En seguida, se procede a realizar la prueba de eliminar cuenta. Para esto se accede a la correspondiente opción en el menú y se debe confirmar diligenciando dos veces la contraseña. El diligenciamiento de este formulario se evidencia en la Figura 48.

Figura 48

Diligenciamiento de Formulario de Eliminar Cuenta

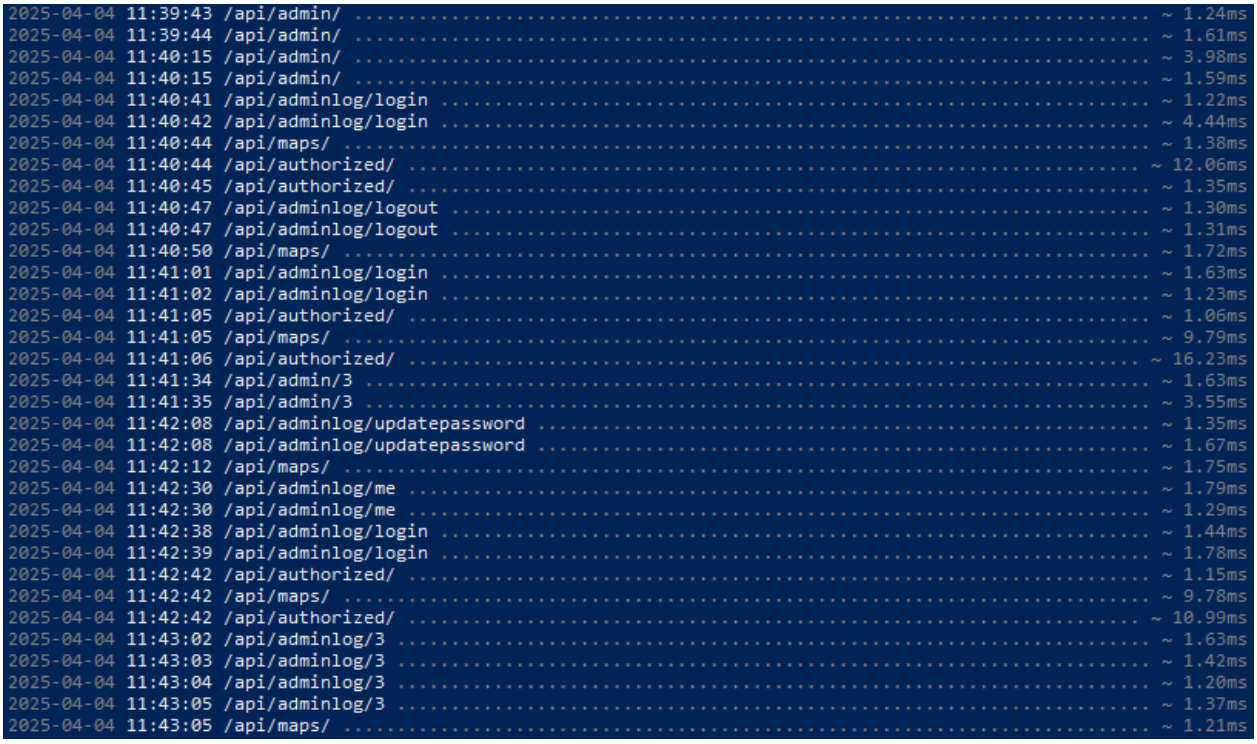


Nota. Autoría propia

Durante las pruebas de registro, y cambio de datos (actualización, cambio de contraseña y eliminación de cuenta) se realizan diversas peticiones a diferentes endpoinds como se evidencia en la Figura 49.

Figura 49

Peticiones a API Durante Pruebas de Registro y Cambio de Datos



Nota. Autoría propia

La imagen a continuación muestra el resultado de la consulta SQL: *SELECT * FROM `admin`* y muestra el estado final de la tabla ‘admin’ demostrando que se ha eliminado el administrador (Ver Figura 50).

Figura 50

Consulta Final SQL de Tabla Admin

id	name	no_doc	email	username	password	main	created_at	updated_at
1	Alejandra Rocio	1021428759	admin0@hotmail.com	admin0	$2y$12$klAbedeYsTjHKbQJRGE3Ku5JMXSN4arruvezTfOAan3...	1	2025-04-04 16:36:47	2025-04-04 16:36:47
2	Jorge Eliecer	1088822	admin1@gmail.com	admin1	$2y$12$.dST0JRYP59PIVnrpspdRO/NqpL.kRR7cln0WdBra7a...	1	2025-04-04 16:39:45	2025-04-04 16:39:45

Nota. Autoría propia

Pruebas Gestión de Dashboards

Se inicia sesión con *admin1* y se crea un cuadro de información de dashboard con la url de dashboard anteriormente generada:

https://app.powerbi.com/view?r=eyJrIjoiMDA3YjZlZmYtMmZhNC00Y2NmLTg1ZjktMjAwM2Y4

MjNlYzZjIiwidCI6ImZjMDA1NDdhLTI0YmItNGU0Zi05ZDYxLTczZmNhNWViOWRmMyIsImMi

OjR9

Luego se procede a crear otros dos dashboards con información ficticia donde uno se configura público y otro no, se comprueba su posteo en el panel público y luego se eliminan los dejando solo el primer cuadro de información. La información de los tres dashboards de prueba diligenciados durante esta prueba se evidencia en la Tabla 16.

Tabla 16

Datos de Cuadro de Dashboards

Nombres de campos	Datos
Cuadro de datos de dashboard1	
Titulo	Plantilla Medellín
Observacione	De datos de prueba suministrados
Lugar	Medellín
Dashboard (URL)	https://app.powerbi.com/view?r=eyJrIjoiMDA3YjZlZmYtMmZhNC00Y2NmLTg1ZjktMjAwM2Y4MjNlYzZjIiwidCI6ImZjMDA1NDdhLTI0YmItNGU0Zi05ZDYxLTczZmNhNWViOWRmMyIsImMiOjR9
Cuadro de datos de dashboard2	
Titulo	Springield Ciudad
Observacione	La vida es aquello que te sucede mientras estás ocupado haciendo otros
Lugar	Springield
Dashboard (URL)	Lorem ipsum dolor sit amet consectetur adipisicing elit. Consectetur quod suscipit accusantium, autem magnam ea
Cuadro de datos de dashboard3	
Titulo	Praderas
Observacione	Datos ocultos de la existencia o inexistentes
Lugar	Manizales
Dashboard (URL)	officia commodi libero totam ut, veritatis culpa tempora quo dolor? Inventore, libero necessitatibus. Pariatur, sequi.

Nota. Esta tabla contiene la información de los primeros 3 cuadros de dashboard creados con datos de prueba. Autor

Durante esta prueba se vuelve a iniciar sesión con '*admin1*' para poder acceder al panel de administrador y se hacen pruebas de gestión de cuadros de información de dashboards, comenzando por el diligenciamiento de la información de dashboard1 (ver Tabla 16) en el formulario situado debajo del título 'Carga de dashboards' del panel de administrador. Luego de

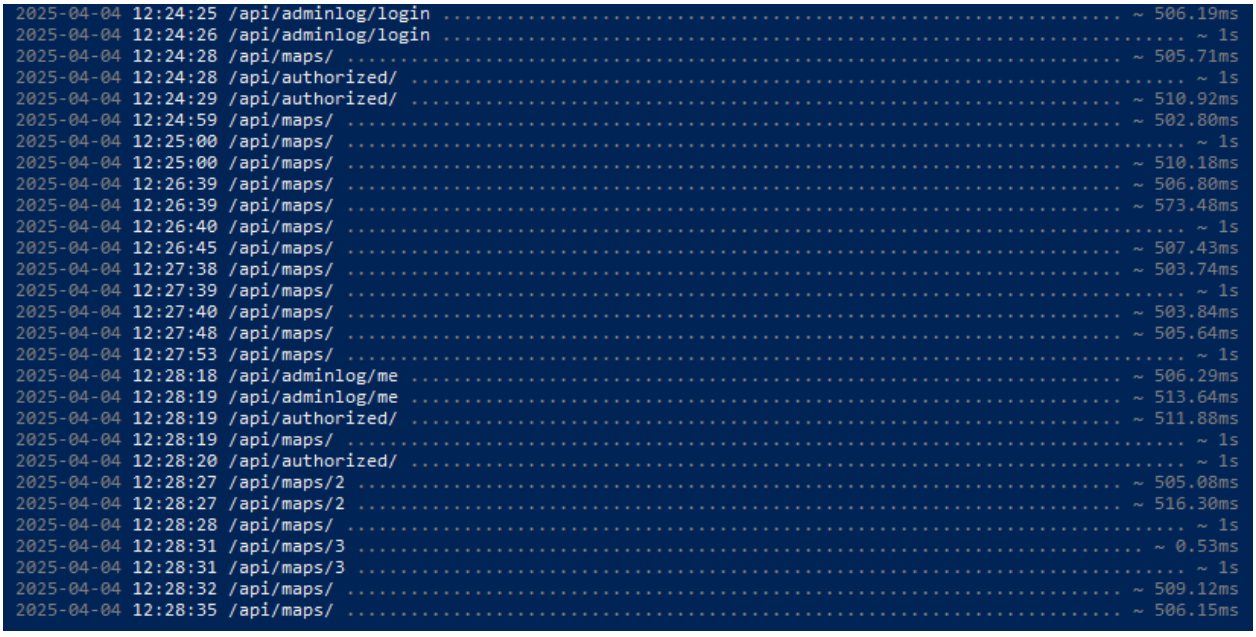
presionar ‘cargar’ este aparece debajo del título ‘edición de dashboards’ donde se prueba la función de previsualización. Donde se muestra la información desplegada tal y como será vista cuando sea dispuesta en el panel público. Se incluyen algunas pruebas de interactividad del dashboard propiamente dicho.

En seguida se procede con el diligenciamiento de los otros dos cuadros de información y se comprueban las funcionalidades de previsualización, actualización, eliminación. Finalmente, se comprueba que solo dashboards habilitados con check público son visibles en el panel público.

Durante estos procedimientos se hacen diversas peticiones al backend, especialmente a los endpoint ‘/api/maps/’ como se evidencia en la Figura 51.

Figura 51

Peticiones a Endpoints Durante la Prueba de Gestión de Dashboards

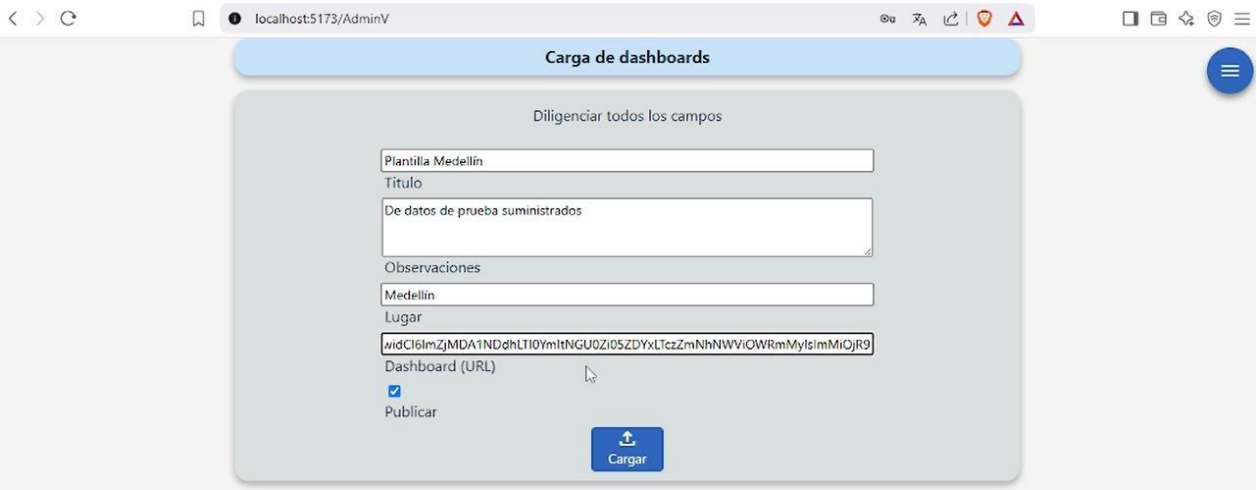


Nota. Autoría propia

Se evidencia la vista con el formulario de carga de dashboards en la Figura 52.

Figura 52

Vista de formulario de carga de dashboards

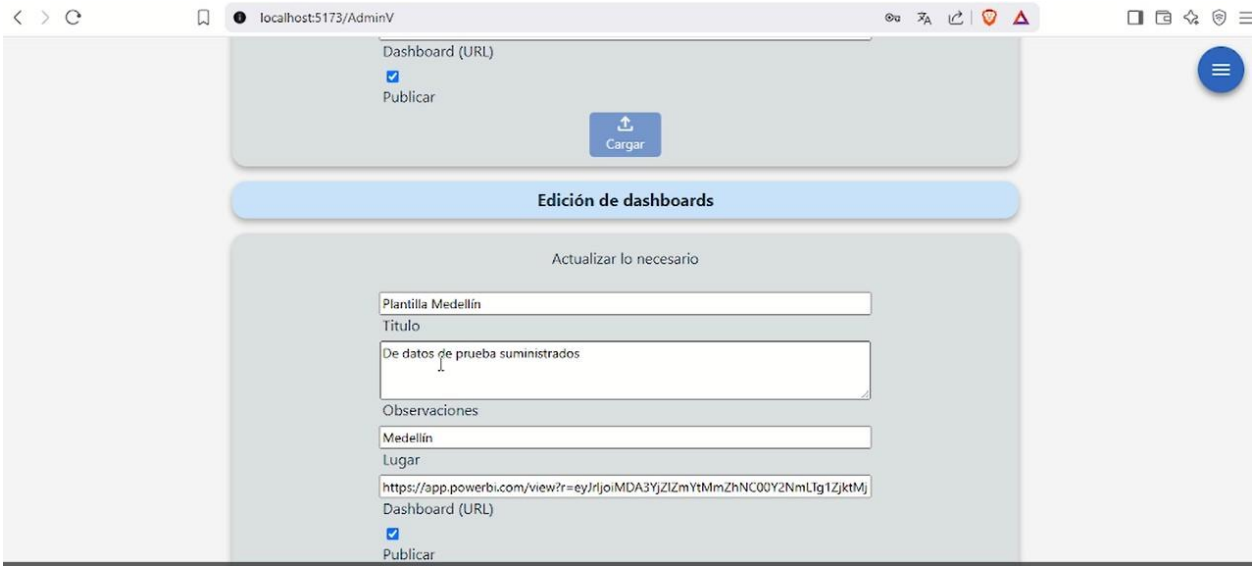


Nota. Autoría propia

Se evidencia el formulario de edición de dashboards en la Figura 53.

Figura 53

Formulario de Edición de Dashboards

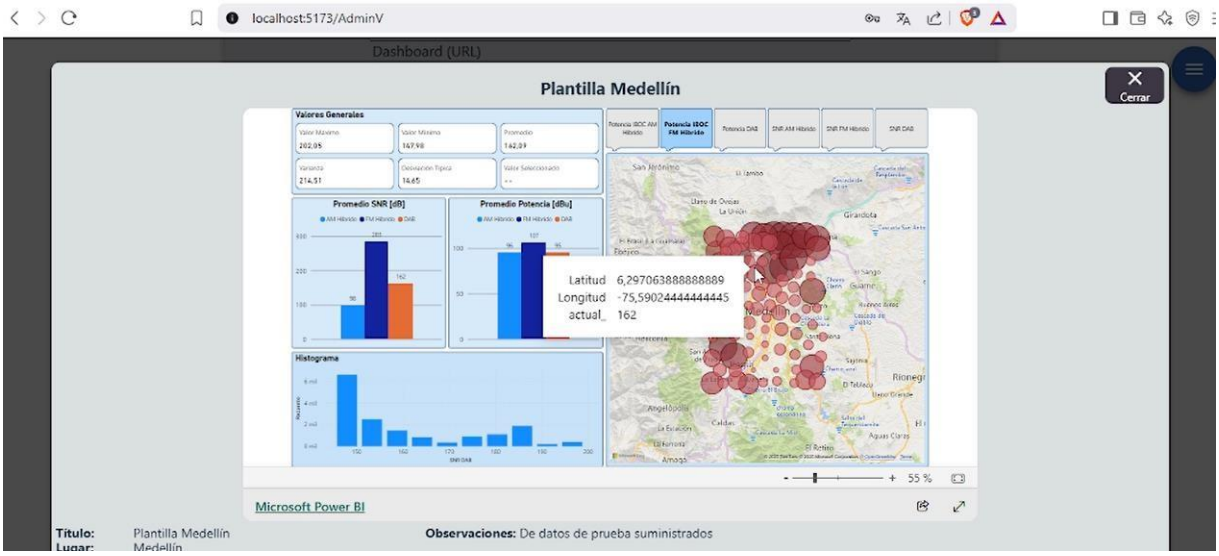


Nota. Autoría propia

Se evidencia la vista preliminar del dashboard1 en la Figura 54.

Figura 54

Vista Preliminar del Dashboard

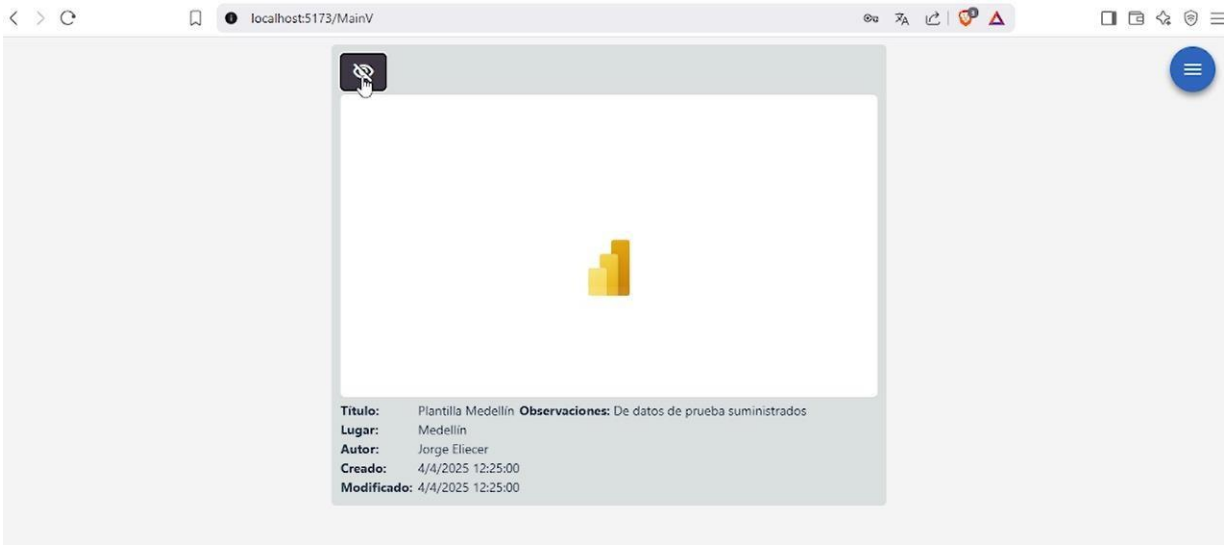


Nota. Autoría propia

Se evidencia el estado final del panel público con el dashboard1 en la Figura 55.

Figura 55

Estado del panel público



Nota. Autoría propia

Finalmente, se evidencia el estado final de la tabla ‘map’ luego de las pruebas realizadas. Para ello se ejecuta la consulta SQL: *SELECT * FROM `map`*. El resultado de esta consulta se evidencia en la figura 56.

Figura 56

Estado Final de la Tabla 'Map'

id	post	title	description	author	urlDashboard	place	created_at	updated_at
1	1	Plantilla Medellín	De datos de prueba suministrados	Jorge Eliecer	https://app.powerbi.com/view?r=eyJrIjoiMDA3YjZlZmY...	Medellín	2025-04-04 17:25:00	2025-04-04 17:25:00

Nota. Autoría propia

Conclusiones

El desarrollo del prototipo de software propuesto permitió lograr el desarrollo de un prototipo de software de aplicativo web que facilita la visualización de las métricas de cobertura del servicio de radiodifusión digital. Este aplicativo brindó una solución acorde a las necesidades del país permitiendo el análisis en tiempo real de los datos relacionados con la calidad y alcance de la señal digital.

En primer lugar, se logró realizar el levantamiento de requerimientos del prototipo, teniendo en cuenta las necesidades del contexto social colombiano y los estudios existentes sobre la transición a la radiodifusión digital.

Se logró también el establecimiento de una secuencia semi automatizada de pasos de análisis de datos para la creación de un dashboard de Power Bi partiendo de datos numéricos y geográficos ordenados en una hoja de cálculo de Excel inicial.

Posteriormente, se llevó a cabo el diseño y desarrollo del aplicativo web, integrando tecnologías modernas como el lenguaje de programación JavaScript, PHP y herramientas para la construcción de dashboards interactivos. Esta fase culminó en un sistema funcional que posibilitó el cómodo acceso de un cliente a la información de los dashboard mediante un navegador web.

Asimismo, se integraron de manera efectiva los distintos elementos del prototipo, incluyendo la base de datos, el aplicativo web y el dashboard, garantizando su correcto funcionamiento como un sistema unificado. Las pruebas de validación realizadas confirmaron la estabilidad, utilidad y precisión del software desarrollado.

Es destacable la capacidad de este aplicativo para gestionar el alojamiento de múltiples dashboards según necesidades futuras, permitiendo la administración por parte de distintos

administradores encargados, sin requerir conocimientos técnicos avanzados. Esta funcionalidad se basa en el uso de formularios web sencillos, lo que facilita su adopción y manejo.

Durante el desarrollo de este prototipo de software, se logró justificar la relevancia de las principales etapas de un proyecto de desarrollo, así como la importancia de comprender las metodologías y arquitecturas utilizadas. Se evidenció la necesidad de un estudio riguroso de los requerimientos del sistema, del análisis preliminar de datos y de una estricta etapa de validación del software para poder garantizar la calidad del producto. Asimismo, se reconoció la importancia de ofrecer interfaces intuitivas y visualmente agradables para el usuario final, que faciliten la interpretación de la información presentada.

Referencias Bibliográficas

Aparicio, F. C. (2021). CHAPTER SIX. The Colombian Media: Modes and Perspective in Television. En E. Fox & S. Waisbord (Eds.), Latin Politics, Global Media (pp. 89–106). University of Texas Press. https://doi.org/10.7560/725362-007

Arias García, C. A. (2015). Implementación de un dashboard para el seguimiento de un portafolio de proyectos [Tesis de grado, Universidad Pontificia Bolivariana]. Repositorio Institucional UPB. https://repository.upb.edu.co/bitstream/handle/20.500.11912/2995/Informe%20Final%20Andres%20Arias.pdf?sequence=1&isAllowed=y

Bonet, M. (2011). La radio digital, estándares tecnológicos y plataformas de distribución. Portal de la Comunicación InCom-UAB. https://incom.uab.cat/portalcom/wp-content/uploads/2020/01/29_esp.pdf

Casino Durán, J. (2017). Aplicación web creada con Shiny para el análisis de la variabilidad de las materias primas en la fabricación de piensos [Trabajo de fin de grado, Universidad Autónoma de Barcelona]. Repositorio Digital UAB. https://ddd.uab.cat/pub/tfg/2017/189581/TFG_joaquincasinoduran.pdf

Clark, D. (2020). Beginning Power BI: A Practical Guide to Self-Service Data Analytics with Excel and Power BI. Apress. https://data.vk.edu.ee/PowerBI/Opikud/Beginning%20Microsoft%20Power%20BI,%203rd%20E dition.pdf

Comunicación Numérica. (2021, 31 de marzo). Crea un dashboard de KPI's [Archivo de vídeo]. YouTube. https://www.youtube.com/watch?v=IdJ52yEUDIo

Federal Communications Commission. (2017). La radio digital. https://www.fcc.gov/sites/default/files/la_radio_digital.pdf

Few, S. (2006). Information Dashboard Design: The Effective Visual Communication of Data. O'Reilly. http://public.magendanz.com/Temp/Information%20Dashboard%20Design.pdf

Fox, E., & Waisbord, S. (2021). Introduction. En E. Fox & S. Waisbord (Eds.), Latin Politics, Global Media (pp. ix–xxiv). University of Texas Press. https://doi.org/10.7560/725362-001

Hernández Martín, Z. (2012). Métodos de análisis de datos [Apuntes]. Universidad de La Rioja, Servicio de Publicaciones.

International Electrotechnical Commission. (2019). IEC Collaboration Platform User Guide. https://www.iec.ch/members_experts/tools/pdf/CollaborationGuide.pdf

Layer, D. H. (2001). Digital radio takes to the road. IEEE Spectrum, 38(7), 40–46. https://doi.org/10.1109/6.931882

Maté, L. S., Pérez Trenard, D. O., & Calvo González, M. (2019). Introducción a la limpieza y análisis de los datos. Universitat Oberta de Catalunya.

Ministerio de Tecnologías de la Información y las Comunicaciones & Agencia Nacional del Espectro. (2012). Fundamentos de gestión nacional del espectro radioeléctrico (Primera ed.). https://www.ane.gov.co/Documentos%20compartidos/ArchivosDescargables/Planeacion/poli- lineamientos-manuales/Manuales/ManualGestionEspectro/Titulo_I.pdf

Moyano Hernández, F. A., & Villamil Sandoval, D. C. (2021). Análisis del ciclo PHVA en la gestión de proyectos: Una revisión documental. Revista Politécnica, 17(34), 55–69. https://doi.org/10.33571/rpolitec.v17n34a4

Olmo, M., & Nave, R. (1999). Señales de radiodifusión. HyperPhysics. Recuperado el 19 de enero de 2023 de http://hyperphysics.phy-astr.gsu.edu/hbasees/Audio/bcast.html#c3

Olmeda-Gómez, C. (2014). Visualización de información. El Profesional de la Información, 23(3), 213–220. https://doi.org/10.3145/epi.2014.may.01

Parra, D., Vélez, J., Vásquez, E., & Fandiño, A. (2018). Definición e implementación de dashboard para el seguimiento y control de indicadores en Nueva EPS [Trabajo de especialización, Politécnico Grancolombiano]. Repositorio Institucional Poligran. https://acortar.link/8dLBuX

Parra Rodríguez, L. N., & Hernández Martínez, A. G. (2022). Desarrollo de un dashboard para la toma de decisiones estratégicas en torno a los negocios de las facilidades de casualty [Trabajo de grado, Universidad de los Andes]. Repositorio Uniandes. https://acortar.link/0Z21C5

Pressman, R. S. (2010). Ingeniería de software: Un enfoque práctico (7ª ed.). McGraw-Hill.

Sieber, A., & Weck, C. (2004). Digital terrestrial television: Trends and developments. EBU Technical Review, (299), 1–12. https://tech.ebu.ch/docs/techreview/trev_299-weck.pdf

Varlamov, O., Varlamov, V., & Dolgopyatova, A. (2019). Digital radio broadcasting network in the Arctic region. En 2019 24th Conference of Open Innovations Association (FRUCT) (pp. 457–462). IEEE. https://doi.org/10.23919/FRUCT.2019.8711933

Apéndices

Apéndice A

Código de Limpieza y Transformación de Datos en Excel Mediante VB

Este anexo muestra el código de Visual Basic implementado para ser llamado en forma de macro en la Plantilla de datos originales suministrados para prepararlos para su posterior uso en PowerBi.

```
Sub Macro1()
    ' ------------------------------------------------------------------------------
    ' ETAPA 1: TRANSFORMACIÓN DE FORMATO
    ' ------------------------------------------------------------------------------
    ' Se observó que en la fila 1 hay celdas combinadas que funcionan como títulos de cada columna de potencia.
    ' Estos títulos se concatenan con el nombre de la columna respectiva para preservar la información relevante.

    Dim celdas As Variant
    celdas = Array("C2", "F2", "I2", "L2", "O2", "R2") ' Celdas donde se aplicará la transformación

    Dim k As Integer
    For k = LBound(celdas) To UBound(celdas)
        Dim celdaActual As Range
        Set celdaActual = Range(celdas(k))

        ' Concatenar con la celda de la fila superior (que contiene títulos)
        Dim celdaSuperior As Range
        Set celdaSuperior = celdaActual.Offset(-1, -2) ' Celda de arriba


        If Not IsEmpty(celdaSuperior) Then
            celdaActual.Value = celdaActual.Value & "_" & celdaSuperior.Value
        End If

        ' Obtener el valor de la celda inferior (potencia o ruido)
        Dim celdaInferior As Range
        Set celdaInferior = celdaActual.Offset(1, 0)

        If Not IsEmpty(celdaInferior) Then
```

```
                    Dim textoInferior As String
                    Dim textoLetras As String
                    textoInferior = celdaInferior.Value
                    textoLetras = ""

                    ' Filtrar solo caracteres tipo letra
                    Dim j As Integer
                    For j = 1 To Len(textoInferior)
                        Dim caracter As String
                        caracter = Mid(textoInferior, j, 1)
                        If caracter Like "[a-zA-Z]" Then
                            textoLetras = textoLetras & caracter
                        End If
                    Next j

                    ' Concatenar texto relevante
                    celdaActual.Value = celdaActual.Value & "_(" & textoLetras
& ")"
                End If

                ' Eliminación de puntos y reemplazo de espacios con guiones
bajos
                celdaActual.Value = Replace(celdaActual.Value, ".", "") '
Eliminar puntos
                celdaActual.Value = Replace(celdaActual.Value, " ", "_") '
Reemplazar espacios por "_"
            Next k

            ' ------------------------------------------------------------------------
            ' ETAPA 2: ELIMINACIÓN DE COLUMNAS REPETIDAS
            ' ------------------------------------------------------------------------
            ' Se observa que las columnas de latitud y longitud están
repetidas, solo se conservan las bajo "A" y "B".
            ' El resto se eliminan.

            Dim columnasAEliminar As Variant
            columnasAEliminar = Array("Q", "P", "N", "M", "K", "J", "H", "G",
"E", "D")

            For k = LBound(columnasAEliminar) To UBound(columnasAEliminar)
                Columns(columnasAEliminar(k)).Delete
            Next k

            ' Eliminar la fila 1 (celdas combinadas descriptivas)
            Rows(1).Delete

            ' ------------------------------------------------------------------------
            ' ETAPA 3: TRANSFORMACIÓN DE COORDENADAS DMS A WGS84
```

```
          ' ----------------------------------------------------------------------------
          ' Se transforman las coordenadas en formato Grados, Minutos y
Segundos (DMS)
          ' a Grados Decimales (DD) con la siguiente fórmula:
          ' DD = Grados + Minutos / 60 + Segundos / 3600
          ' Además, las coordenadas en hemisferio sur (S) y oeste (W) se
convierten a valores negativos.

          Range("A2:B2").Select
          Range(Selection, Selection.End(xlDown)).Select

          Dim cadena As String
          Dim grados As Double
          Dim minutos As Double
          Dim segundos As Double
          Dim direccion As String
          Dim posGrados As Integer
          Dim posMinutos As Integer
          Dim posSegundos As Integer

          ' Iterar sobre cada celda en el rango seleccionado
          Dim celda As Range
          For Each celda In Selection
              ' Obtener la cadena de texto con la coordenada
              cadena = celda.Value

              ' Encontrar posiciones de los delimitadores en la cadena
              posGrados = InStr(cadena, "º")
              posMinutos = InStr(cadena, "'")
              posSegundos = InStr(cadena, """")

              ' Extraer valores de grados, minutos y segundos
              grados = Val(Left(cadena, posGrados - 1))
              minutos = Val(Mid(cadena, posGrados + 1, posMinutos - posGrados
- 1))
              segundos = Val(Mid(cadena, posMinutos + 1, posSegundos -
posMinutos - 1))

              ' Extraer la dirección (N, S, E, W)
              direccion = Right(cadena, 1)

              ' Aplicar la conversión a grados decimales
              Dim resultado As Double
              resultado = grados + (minutos / 60) + (segundos / 3600)

              ' Si la dirección es "S" o "W", convertir a negativo
              If direccion = "S" Or direccion = "W" Then
                  resultado = resultado * -1
```

```
            End If

            ' Reemplazar el valor de la celda con la nueva coordenada en DD
            celda.Value = resultado
        Next celda

        ' ------------------------------------------------------------------------
        ' ETAPA 4: LIMPIEZA FINAL DE DATOS
        ' ------------------------------------------------------------------------
        ' Se eliminan caracteres adicionales en celdas seleccionadas y se
reemplazan valores erróneos.

        Range("C2").Select
        Range(Selection, Selection.End(xlToRight)).Select
        Range(Selection, Selection.End(xlDown)).Select

        Dim reemplazo As Double
        reemplazo = 10000000 ' Valor para reemplazo de errores

        For Each celda In Selection
            Dim textoActual As String
            textoActual = celda.Value

            ' Encontrar la posición del primer espacio y truncar el texto
            Dim posicionEspacio As Integer
            posicionEspacio = InStr(textoActual, " ")

            If posicionEspacio > 0 Then
                celda.Value = Left(textoActual, posicionEspacio - 1)
            End If

            ' Reemplazar valores "Infinity" por un número grande
            If LCase(Trim(celda.Value)) = "infinity" Then
                celda.Value = reemplazo
            End If
        Next celda
    End Sub
```

Apéndice B

Resumen de Consultas Dax

```
      DEFINE MEASURE 'Tabla1'[Promedio_med1] =
CALCULATE(AVERAGE('Tabla1'[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)]),
ALL('Tabla1'))
          MEASURE 'Tabla1'[Promedio_med2] =
CALCULATE(AVERAGE('Tabla1'[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)]),
ALL('Tabla1'))
          MEASURE 'Tabla1'[Promedio_med3] =
CALCULATE(AVERAGE('Tabla1'[Potencia_Cob_Multi_DAB__(dBu)]), ALL('Tabla1'))
          MEASURE 'Tabla1'[Promedio_med5] = CALCULATE(
          AVERAGE('Tabla1'[SNR_Interferencia_FM_Hibrido_(dB)]),
          'Tabla1'[SNR_Interferencia_FM_Hibrido_(dB)]<> 10000000)
          MEASURE 'Tabla1'[Medida1] =
SELECTEDVALUE('Tabla1'[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[Medida2] =
SELECTEDVALUE('Tabla1'[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[Medida3] =
SELECTEDVALUE('Tabla1'[Potencia_Cob_Multi_DAB_(dBu)])
          MEASURE 'Tabla1'[Medida4] =
SELECTEDVALUE('Tabla1'[SNR_Interferencia_AM_Hibrido_(dB)])
          MEASURE 'Tabla1'[Medida5] =
SELECTEDVALUE('Tabla1'[SNR_Interferencia_FM_Hibrido_(dB)])
          MEASURE 'Tabla1'[Medida6] =
SELECTEDVALUE('Tabla1'[SNR_Interferencia_DAB_(dB)])
          MEASURE 'Tabla1'[Max_Medida1] =
CALCULATE(MAX(Tabla1[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)]),ALL('Tabla1')
)
          MEASURE 'Tabla1'[Max_Medida2] =
CALCULATE(MAX(Tabla1[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Max_Medida3] =
CALCULATE(MAX(Tabla1[Potencia_Cob_Multi_DAB_(dBu)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Max_Medida4] =
CALCULATE(MAX(Tabla1[SNR_Interferencia_AM_Hibrido_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Max_Medida6] =
CALCULATE(MAX(Tabla1[SNR_Interferencia_DAB_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Max_Medida5] =
CALCULATE(MAX(Tabla1[SNR_Interferencia_FM_Hibrido_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Min_Medida1] =
CALCULATE(MIN(Tabla1[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)]),ALL('Tabla1')
)
          MEASURE 'Tabla1'[Min_Medida2] =
CALCULATE(MIN(Tabla1[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Min_Medida3] =
CALCULATE(MIN(Tabla1[Potencia_Cob_Multi_DAB_(dBu)]),ALL('Tabla1'))
```

```
          MEASURE 'Tabla1'[Min_Medida4] =
CALCULATE(MIN(Tabla1[SNR_Interferencia_AM_Hibrido_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Min_Medida5] =
CALCULATE(MIN(Tabla1[SNR_Interferencia_FM_Hibrido_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Min_Medida6] =
CALCULATE(MIN(Tabla1[SNR_Interferencia_DAB_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Prom_Medida1] =
CALCULATE(AVERAGE(Tabla1[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)]),ALL('Tabl
a1'))
          MEASURE 'Tabla1'[Prom_Medida2] =
CALCULATE(AVERAGE(Tabla1[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)]),ALL('Tabla
1'))
          MEASURE 'Tabla1'[Prom_Medida3] =
CALCULATE(AVERAGE(Tabla1[Potencia_Cob_Multi_DAB_(dBu)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Prom_Medida4] =
CALCULATE(AVERAGE(Tabla1[SNR_Interferencia_AM_Hibrido_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Prom_Medida6] =
CALCULATE(AVERAGE(Tabla1[SNR_Interferencia_DAB_(dB)]),ALL('Tabla1'))
          MEASURE 'Tabla1'[Prom_Medida5] =
CALCULATE(AVERAGE(AUX_SNR_FM[SNR_Interferencia_FM_Hibrido_(dB)]),ALL('AUX_SNR
_FM'))
          MEASURE 'Tabla1'[varianza1] =
VAR.P('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[varianza2] =
VAR.P('Tabla1'[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[varianza3] =
VAR.P('Tabla1'[Potencia_Cob_Multi_DAB_(dBu)])
          MEASURE 'Tabla1'[varianza4] =
 VAR.P('Tabla1'[SNR_Interferencia_AM_Hibrido_(dB)])
           MEASURE 'Tabla1'[varianza5] =
 VAR.P('Tabla1'[SNR_Interferencia_FM_Hibrido_(dB)])
           MEASURE 'Tabla1'[varianza6] =
VAR.P('Tabla1'[SNR_Interferencia_DAB_(dB)])
          MEASURE 'Tabla1'[desviacion_e1] =
STDEV.P('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[desviacion_e2] =
STDEV.P('Tabla1'[Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)])
          MEASURE 'Tabla1'[desviacion_e3] =
STDEV.P('Tabla1'[Potencia_Cob_Multi_DAB_(dBu)])
          MEASURE 'Tabla1'[desviacion_e4] =
 STDEV.P('Tabla1'[SNR_Interferencia_AM_Hibrido_(dB)])
         MEASURE 'Tabla1'[desviacion_e5] =
 STDEV.P('Tabla1'[SNR_Interferencia_FM_Hibrido_(dB)])
         MEASURE 'Tabla1'[desviacion_e6] =
STDEV.P('Tabla1'[SNR_Interferencia_DAB_(dB)])
          MEASURE 'segmentador'[actual_] = SWITCH (
          SELECTEDVALUE (  segmentador[segmentador Orden] ),
            0, [Medida1],
```

```
            1, [Medida2],
            2, [Medida3],
            3, [Medida4],
            4, [Medida5],
            5, [Medida6]
        )
            MEASURE 'segmentador'[max_actual_] = SWITCH (
            SELECTEDVALUE (  segmentador[segmentador Orden] ),
              0, [Max_Medida1],
            1, [Max_Medida2],
            2, [Max_Medida3],
            3, [Max_Medida4],
            4, [Max_Medida5],
            5, [Max_Medida6]
        )
            MEASURE 'segmentador'[min_actual_] = SWITCH (
            SELECTEDVALUE (  segmentador[segmentador Orden] ),
              0, [Min_Medida1],
            1, [Min_Medida2],
            2, [Min_Medida3],
            3, [Min_Medida4],
            4, [Min_Medida5],
            5, [Min_Medida6]
           )
            MEASURE 'segmentador'[prom_actual] = SWITCH (
            SELECTEDVALUE (  segmentador[segmentador Orden] ),
              0, [Prom_Medida1],
            1, [Prom_Medida2],
            2, [Prom_Medida3],
            3, [Prom_Medida4],
            4, [Prom_Medida5],
            5, [Prom_Medida6]
           )
            MEASURE 'segmentador'[poblacion] = SELECTEDVALUE
(segmentador[segmentador Campos])
            MEASURE 'segmentador'[varianza_actual_] = SWITCH (
            SELECTEDVALUE (  segmentador[segmentador Orden] ),
              0, [varianza1],
            1, [varianza2],
            2, [varianza3],
            3, [varianza4],
            4, [varianza5],
            5, [varianza6]
        )
            MEASURE 'segmentador'[desviacion_e_actual_] = SWITCH (
            SELECTEDVALUE (  segmentador[segmentador Orden] ),
              0, [desviacion_e1],
              1, [desviacion_e2],
```

```
      2, [desviacion_e3],
      3, [desviacion_e4],
      4, [desviacion_e5],
      5, [desviacion_e6]
)

EVALUATE
    SUMMARIZECOLUMNS(
        "Promedio_med1", 'Tabla1'[Promedio_med1],
        "Promedio_med2", 'Tabla1'[Promedio_med2],
        "Promedio_med3", 'Tabla1'[Promedio_med3],
        "Promedio_med5", 'Tabla1'[Promedio_med5],
        "Medida1", 'Tabla1'[Medida1],
        "Medida2", 'Tabla1'[Medida2],
        "Medida3", 'Tabla1'[Medida3],
        "Medida4", 'Tabla1'[Medida4],
        "Medida5", 'Tabla1'[Medida5],
        "Medida6", 'Tabla1'[Medida6],
        "Max_Medida1", 'Tabla1'[Max_Medida1],
        "Max_Medida2", 'Tabla1'[Max_Medida2],
        "Max_Medida3", 'Tabla1'[Max_Medida3],
        "Max_Medida4", 'Tabla1'[Max_Medida4],
        "Max_Medida6", 'Tabla1'[Max_Medida6],
        "Max_Medida5", 'Tabla1'[Max_Medida5],
        "Min_Medida1", 'Tabla1'[Min_Medida1],
        "Min_Medida2", 'Tabla1'[Min_Medida2],
        "Min_Medida3", 'Tabla1'[Min_Medida3],
        "Min_Medida4", 'Tabla1'[Min_Medida4],
        "Min_Medida5", 'Tabla1'[Min_Medida5],
        "Min_Medida6", 'Tabla1'[Min_Medida6],
        "Prom_Medida1", 'Tabla1'[Prom_Medida1],
        "Prom_Medida2", 'Tabla1'[Prom_Medida2],
        "Prom_Medida3", 'Tabla1'[Prom_Medida3],
        "Prom_Medida4", 'Tabla1'[Prom_Medida4],
        "Prom_Medida6", 'Tabla1'[Prom_Medida6],
        "Prom_Medida5", 'Tabla1'[Prom_Medida5],
        "varianza1", 'Tabla1'[varianza1],
        "varianza2", 'Tabla1'[varianza2],
        "varianza3", 'Tabla1'[varianza3],
        "varianza4", 'Tabla1'[varianza4],
        "varianza5", 'Tabla1'[varianza5],
        "varianza6", 'Tabla1'[varianza6],
        "desviacion_e1", 'Tabla1'[desviacion_e1],
        "desviacion_e2", 'Tabla1'[desviacion_e2],
        "desviacion_e3", 'Tabla1'[desviacion_e3],
        "desviacion_e4", 'Tabla1'[desviacion_e4],
        "desviacion_e5", 'Tabla1'[desviacion_e5],
        "desviacion_e6", 'Tabla1'[desviacion_e6],
```

```
    "actual_", 'segmentador'[actual_],
    "max_actual_", 'segmentador'[max_actual_],
    "min_actual_", 'segmentador'[min_actual_],
    "prom_actual", 'segmentador'[prom_actual],
    "poblacion", 'segmentador'[poblacion],
    "varianza_actual_", 'segmentador'[varianza_actual_],
    "desviacion_e_actual_", 'segmentador'[desviacion_e_actual_]
)
```

Apéndice C

Documentación Sobre el Modelo de Consultas DAX

Esta consulta DAX define varias medidas estadísticas para analizar la potencia y la interferencia de señales de radio en distintos formatos *(IBOC AM, IBOC FM, DAB).* Además, usa funciones *CALCULATE, AVERAGE, MAX, MIN, VAR.P, STDEV.P* y *SWITCH* para realizar cálculos sobre las columnas de la tabla 'Tabla1' y segmentarlas dinámicamente.

El objetivo de esta consulta es proporcionar resúmenes estadísticos y permitir la selección dinámica de métricas a través de segmentadores.

Las medidas se pueden categorizar en distintos grupos según su funcionalidad:

Medidas de Promedio

Estas medidas calculan el promedio de cada métrica sin considerar ningún filtro aplicado:

```
MEASURE 'Tabla1'[Promedio_med1] =
CALCULATE(AVERAGE('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)]),
ALL('Tabla1'))
```

- Calcula el promedio total de *'Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)',* ignorando filtros.
- Se usa *ALL('Tabla1')* para remover filtros y obtener la media general.

Otras medidas similares

- *Promedio_med2:* Calcula el promedio de `Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)'.`
- *Promedio_med3:* Calcula el promedio de *'Potencia_Cob_Multi_DAB__(dBu)'*.
- *Promedio_med5:* Calcula el promedio de *'SNR_Interferencia_FM_Hibrido_(dB)'*, pero excluye valores de 10000000.

Medidas de Selección de Valores

Estas medidas devuelven el valor seleccionado en un filtro (segmentador) para cada métrica.

```
DAX MEASURE 'Tabla1'[Medida1] =
SELECTEDVALUE('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)])
```

SELECTEDVALUE() devuelve el valor seleccionado en la tabla 'Tabla1'.

Si hay más de un valor seleccionado, la medida devuelve BLANK().

Otras medidas similares:

```
Medida2: 'Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)'
Medida3: 'Potencia_Cob_Multi_DAB__(dBu)'
Medida4 - Medida6: Aplicadas a las métricas 'SNR_Interferencia'.
```

Medidas de Máximos y Mínimos

Estas medidas calculan el valor máximo y mínimo de cada métrica en toda la tabla DAX

Documentación de Vista de Consultas DAX en Power bi

Esta consulta DAX define varias medidas estadísticas para analizar la potencia y la interferencia de señales de radio en distintos formatos *(IBOC AM, IBOC FM, DAB).* Además, usa funciones *CALCULATE, AVERAGE, MAX, MIN, VAR.P, STDEV.P* y *SWITCH* para realizar cálculos sobre las columnas de la tabla 'Tabla1' y segmentarlas dinámicamente.

El objetivo de esta consulta es proporcionar resúmenes estadísticos y permitir la selección dinámica de métricas a través de segmentadores.

Descripción de las Medidas DAX

Las medidas se pueden categorizar en distintos grupos según su funcionalidad:

Medidas de Promedio

Estas medidas calculan el promedio de cada métrica sin considerar ningún filtro aplicado:

```
MEASURE 'Tabla1'[Promedio_med1] =
CALCULATE(AVERAGE('Tabla1'[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)]),
ALL('Tabla1'))
```

Calcula el promedio total de *'Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)'*, ignorando filtros.

Se usa *ALL('Tabla1')* para remover filtros y obtener la media general.

Otras medidas similares

Promedio_med2: Calcula el promedio de *'Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)'*.

Promedio_med3: Calcula el promedio de *'Potencia_Cob_Multi_DAB__(dBu)'*.

Promedio_med5: Calcula el promedio de *'SNR_Interferencia_FM_Hibrido_(dB)'* pero excluye valores de 10000000.

Medidas de Selección de Valores

Estas medidas devuelven el valor seleccionado en un filtro (segmentador) para cada métrica.

```
MEASURE 'Tabla1'[Medida1] =
SELECTEDVALUE('Tabla1'[Potencia__Cob_Mult_IBOC_AM_Hibrido_(dBu)])
```

SELECTEDVALUE() devuelve el valor seleccionado en la tabla 'Tabla1'.

Si hay más de un valor seleccionado, la medida devuelve *BLANK()*.

Otras medidas similares

```
Medida2: 'Potencia_Cob_Mult_IBOC_FM_Hibrido_(dBu)'
Medida3: 'Potencia_Cob_Multi_DAB__(dBu)'
Medida4 - Medida6: Aplicadas a las métricas 'SNR_Interferencia'.
```

Medidas de Máximos y Mínimos

Estas medidas calculan el valor máximo y mínimo de cada métrica en toda la tabla:

```
MEASURE 'Tabla1'[Max_Medida1] =
CALCULATE(MAX('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)]),
ALL('Tabla1'))
```

MAX() devuelve el valor máximo de la columna.

ALL('Tabla1') se usa para ignorar cualquier filtro.

Otras medidas similares

Max_Medida2 - Max_Medida6: Calculan el máximo de las demás métricas.

Min_Medida1 - Min_Medida6: Aplican la función MIN() para obtener el valor mínimo.

Medidas de Varianza y Desviación Estándar

Se utilizan para calcular la variabilidad de los datos:

```
MEASURE 'Tabla1'[varianza1] =
VAR.P('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)])
MEASURE 'Tabla1'[desviacion_e1] =
STDEV.P('Tabla1'[Potencia_Cob_Mult_IBOC_AM_Hibrido_(dBu)])
```

VAR.P() calcula la varianza poblacional.

STDEV.P() calcula la desviación estándar poblacional.

Otras medidas similares

varianza2 - varianza6 y desviacion_e2 - desviacion_e6: Se aplican a otras métricas.

Resumen de EVALUATE

La consulta final usa *EVALUATE SUMMARIZECOLUMNS()* para devolver una tabla con todas las medidas:

```
EVALUATE
SUMMARIZECOLUMNS(
"Promedio_med1", 'Tabla1'[Promedio_med1],
"Promedio_med2", 'Tabla1'[Promedio_med2],
"Max_Medida1", 'Tabla1'[Max_Medida1],
"Min_Medida1", 'Tabla1'[Min_Medida1],
"varianza1", 'Tabla1'[varianza1],
```

```
"desviacion_e1", 'Tabla1'[desviacion_e1]
)
```

SUMMARIZECOLUMNS() crea una tabla con los valores de cada medida.

Se pueden agregar más columnas para calcular múltiples métricas en una sola consulta.

Conclusiones

- La consulta DAX calcula múltiples estadísticas (promedio, máximo, mínimo, varianza, desviación estándar) sobre distintas señales de potencia e interferencia.
- Las medidas con *SWITCH()* permiten que los usuarios seleccionen dinámicamente qué métrica visualizar en un segmentador.
- *EVALUATE SUMMARIZECOLUMNS()* estructura los datos en una tabla lista para análisis en Power BI.

Apéndice Dfuente

Códigos MainV.Vue y AdminV.Vue

Archivo *src/views/MainV.vue*

```
<script setup lang="ts">
  import { ref, Ref, onMounted } from 'vue';
  import MenuComponent from '../components/MenuComponent.vue';
  import RegPopup from '../components/mainPop_up/RegPopup.vue';
  import LoginPopup from '../components/mainPop_up/LoginPopup.vue';
  import PostService from '../services/PostService';
  import DashboardMain from '../components/DashboardMain.vue';
  import AlertPop_up from '../components/AlertPopup.vue';
  import AdminRegData from '../interfaces/AdminRegData';
  import TriggerAlertClass from '../class/TriggerAlertClass';
  import AdminLoginData from '../interfaces/AdminLoginData';
  import AuthService from '../services/AuthService';
  import { useRouter } from 'vue-router';
  import AdminService from '../services/AdminService';
  const router = useRouter();
  let lg_: AdminLoginData = {username:'', password: ''};
  const handleClose = () => {
    triggerAlert.set_showAlert(ref(false));
    if (triggerAlert.get_sm() === 'Sesión iniciada'){
      router.push({name:'AdminV'})
    }
  };

    // Función para montar la alerta
  const login= async (datalogin: AdminLoginData) => {
    try {
      const auth = new AuthService();
      const success = await auth.login(datalogin);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'Sesión iniciada');
        return
      } else {
        triggerAlert.set_sms(false,'Error en login');
      }
    } catch (e) {
        triggerAlert.set_sms(false,'Login fallido');
    }
  };

  const register = async (rg: AdminRegData) => {
    try {
```

```
      const adminService = new AdminService();
      const { registerUser, success } = await
adminService.useRegisterService();
      await registerUser(rg);
      triggerAlert.set_showAlert(ref(true));
      if (success.value) {
        triggerAlert.set_sms(true,'Registro exitoso');
      } else {
        triggerAlert.set_sms(false,'Registro fallido');
      }
    } catch (e) {
      console.log(e);
      triggerAlert.set_sms(false,'Registro falló');
    }
  };

  const service = new PostService();
  const posts = service.getPublicPosts();
  onMounted(async () => {
    await service.fetchAllPublics();
    console.log('posts',posts)
  })

  const showRegisterPopup: Ref<boolean> = ref(false);
  const showLoginPopup: Ref<boolean> = ref(false);
  const triggerAlert = new TriggerAlertClass;
  // Métodos para abrir y cerrar cada popup
  const registerPop_up = ():void => {
    showRegisterPopup.value = true;
    showLoginPopup.value = false;
  };

  const closeRegisterPop_up = ():void => {
    showRegisterPopup.value = false;
  };

  const loginPop_up = ():void => {
    console.log('loginp')
    showLoginPopup.value = true;
    showRegisterPopup.value = false;
  };

  const closeLoginPop_up = ():void => {
    showLoginPopup.value = false;
  };

  const closed = ():void => {
    showLoginPopup.value = false;
```

```
    showRegisterPopup.value = false;
  };

  const showInfo = ():void => {
    alert("mostranfo información")
  };
  console.log(posts)
</script>

<template>
  <MenuComponent @in="loginPop_up" @rg="registerPop_up" @cl="closed"
@info="showInfo" :datalogin="lg_"/>
  <RegPopup @pd="closeRegisterPop_up" @rgd="register" v-
if="showRegisterPopup" class="popup"/>
  <LoginPopup @pd="closeLoginPop_up" @lg="login" v-if="showLoginPopup"
class="popup"/>
  <DashboardMain
      v-for="(post, index) in posts"
      :key="index"
      :EXISTING_DASHBOARD="post"
    />
    <AlertPop_up
      v-if="triggerAlert.get_showAlert()"
      :sm="triggerAlert.get_sm()"
      :smstatus="triggerAlert.get_smstatus()"
      @close="handleClose"
    />

</template>
<style scoped>

  .popup {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: #fbfff1;
    padding: 20px;
    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.3);
    border-radius: 8px;
    z-index: 1000;
    width: 20%;
    min-width: 20%;
    display: flex;
    flex-direction: column;
    align-items: center;
  }
</style>
```

Archivo *src/views/AdminV.vue*

```
<script setup lang="ts">
  import { ref, Ref, onMounted} from 'vue';
  import { getDecodedToken } from '../services/authHelpers';
  import { EMPTY_ADMIN } from '../components/constantInfo/empty_admin';
  import router from '../router';
  import ChangePass from '../components/adminPop_up/ChangePass.vue';
  import DestroyUser from '../components/adminPop_up/DestroyUser.vue';
  import PatchUpdate from '../components/adminPop_up/PatchUpdate.vue';
  import FormD from '../components/FormD.vue';
  import MenuComponent from '../components/MenuComponent.vue';
  import ManageAuthorizations from
'../components/adminPop_up/Authorizations.vue';
  import MAData from '../interfaces/MAData';
  import AlertPop_up from '../components/AlertPopup.vue';
  import Dashboard_Data from '../interfaces/DashboardData';
  import  PostService  from  '../services/PostService';
  import DecodedToken from '../interfaces/DecodedToken';
  import AdminRegData from '../interfaces/AdminRegData';
  import AdminService from '../services/AdminService';
  import AuthService from '../services/AuthService';
  import TriggerAlertClass from '../class/TriggerAlertClass';
  import AuthorizedService from '../services/AuthorizedService';

     //service for dashboards in FormD
  const service = new PostService();
  const posts = service.getPosts();
  //service for authorizations
  const serviceMAData = new AuthorizedService;
  let authorizations = ref<MAData[]>([]);
  authorizations = serviceMAData.getAuthoriationsData();
  onMounted(async () => {
    await Promise.all([
      service.fetchAll(),
      serviceMAData.fetchAll()
    ]);
  })
  //service for sendDataAdmin
  const serviceAdmin = new AdminService;
  //service for sendDataChangePass
  const serviceAdminAuth = new AuthService;
  //consts
  const decodedToken: DecodedToken | null = getDecodedToken();
  const destroyUserPopup1: Ref<boolean> = ref(false);
  const patchUpdatePopup1: Ref<boolean> = ref(false);
```

```
  const passChangePopup1: Ref<boolean> = ref(false);
  const statusOA: Ref<boolean> = ref(false);
  let otorgarAutorizacion: boolean = false;
  let flag_updateDataView: Ref<boolean> = ref(false);
  let dataAdmin: AdminRegData = EMPTY_ADMIN;
  if (decodedToken !== null) {
    otorgarAutorizacion = decodedToken.main;
    const {name, no_doc, email, username} = decodedToken;
    dataAdmin = {name: name, no_doc: no_doc, email: email, username:
username};
  }

  const dForm: boolean = true;//true is Form1, false is form2
  const fRegresar = (): void => {
    router.push({ name: 'MainV' });
  }
  const fChange = (): void => {
    passChangePopup1.value = !passChangePopup1.value;
    destroyUserPopup1.value = false;
    patchUpdatePopup1.value = false;
    statusOA.value = false;
  }
  const fPatch = (): void => {
    patchUpdatePopup1.value = !patchUpdatePopup1.value;
    destroyUserPopup1.value = false;
    passChangePopup1.value = false;
    statusOA.value = false;
  }
  const fDestroy = (): void => {
    destroyUserPopup1.value = !destroyUserPopup1.value;
    patchUpdatePopup1.value = false;
    passChangePopup1.value = false;
    statusOA.value = false;
  }
  const fAutorizar = (): void => {
    statusOA.value = !statusOA.value;
    destroyUserPopup1.value = false;
    patchUpdatePopup1.value = false;
    passChangePopup1.value = false;
  }

  // instancia para montar la alerta
  const triggerAlert = new TriggerAlertClass;

  // Maneja el cierre de la alerta desde el hijo
  const handleClose = () : void => {
    const flag_to_main: boolean = ['clave actualizada', 'cuenta
eliminada'].includes(triggerAlert.get_sm());
```

```
    console.log(' handleClose ')
    if (flag_to_main) {
      fRegresar()
    }
    triggerAlert.set_showAlert(ref(false));
  };
  const updateDash = async (data: Dashboard_Data, id: string): Promise<void>
=> {
    try {
      const service = new PostService();
      const success = await service.updatePost(data, id);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'Datos actualizados');
        await updateDataView();
        return
      } else {
        triggerAlert.set_sms(false,'Actualizanción fallida');
      }
    } catch (e) {
      console.log('Error actualizando: ',e);
      triggerAlert.set_sms(false,'Error actualizando');
    }
  }
  const destroyDash = async (id: string): Promise<void> => {
    try {
      const success = await service.deletePost(id);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'Dashboard Eliminado');
        await updateDataView();
        return
      } else {
        triggerAlert.set_sms(false,'Fallido: eliminar');
      }
    } catch (e) {
      console.log('Error al eliminar: ',e);
      triggerAlert.set_sms(false,'Error al eliminar');
    }
  }
  const sendDash = async (data: Dashboard_Data): Promise<void> => {
    try {
      const success = await service.createPost(data);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'Dashboard Cargado');
        await updateDataView();
        return
```

```
      } else {
        triggerAlert.set_sms(false,'Fallido: Cargar Dashboard');
      }
    } catch (e) {
      triggerAlert.set_sms(false,'Error al Cargar');
      console.log('Error al Cargar Dashboard: ',e);
    }
  }
  const updateDataView = async () : Promise<void> => {
      flag_updateDataView.value = !flag_updateDataView.value;
      await service.fetchAll()
  }
  const sendDataUpdateAdmin = async (data:  Partial<AdminRegData>):
Promise<void> => {
    try {
      const success = await serviceAdmin.updateAdmin(data);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        Object.assign(dataAdmin,success.map);
        triggerAlert.set_sms(true,'Datos actualizados');
        return
      } else {
        triggerAlert.set_sms(false,'Fallido: actualizar datos');
      }
    } catch (e) {
      console.log('Error al Cargar datos de Admin: ',e);
      triggerAlert.set_sms(false,'Error al Cargar datos');
    }
  }
  const changeP = async (data: object): Promise<void> => {
    try {
      const success = await serviceAdminAuth.changePass(data);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'clave actualizada');
      } else {
        triggerAlert.set_sms(false,'error actualizando clave');
      }
    } catch (error) {
      triggerAlert.set_sms(false,'error: clave no actualizada');
    }
  }

  const delAccount = async (data: object): Promise<void> => {
    try {
      const success = await serviceAdminAuth.deleteAdmin(data);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
```

```
        triggerAlert.set_sms(true,'cuenta eliminada');
      } else {
        triggerAlert.set_sms(false,'error eliminando cuenta');
      }
    } catch (error) {
      triggerAlert.set_sms(false,'error: cuenta no eliminada');
    }
  }
  const sendAuthorization = async (data: MAData): Promise<void> => {
    try {
      const success = await serviceMAData.regAuthorization(data);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'autorización cargada');
      } else {
        triggerAlert.set_sms(false,'error cargando autorización');
      }
    } catch (error) {
      triggerAlert.set_sms(false,'error: autorización no cargada');
    }
  }
  const removeAuthorization = async (id: number): Promise<void> => {
    try {
      const success = await serviceMAData.deleteAuthorization(id);
      triggerAlert.set_showAlert(ref(true));
      if (success) {
        triggerAlert.set_sms(true,'autorización eliminada');
      } else {
        triggerAlert.set_sms(false,'error eliminando autorización');
      }
    } catch (error) {
      triggerAlert.set_sms(false,'error: autorización no eliminada');
    }
  }
</script>

<template>
  <div class="cont__main">
    <h2 >Carga de dashboards</h2>
    <FormD
      :flag="dForm"
      @sendDash="sendDash"
      :flagWatchData="ref(false)"
    />
    <h2>Edición de dashboards</h2>
    <FormD
      v-for="(post, index) in posts"
      :key="index"
```

```
      :flag="!dForm"
      :EXISTING_DASHBOARD="ref(post)"
      :flagWatchData="ref(flag_updateDataView)"
      @sendDash="updateDash"
      @deleteDash="destroyDash"
    />
    <ChangePass @flag="fChange"
      @datacp="changeP"
      v-if="passChangePopup1"
    />
    <PatchUpdate @flag="fPatch"
      v-if="patchUpdatePopup1"
      :EXISTING_ADMIN="ref(dataAdmin)"
      @sendData="sendDataUpdateAdmin"
    />
    <DestroyUser @flag="fDestroy" @datadestroy="delAccount" v-
if="destroyUserPopup1" />
    <ManageAuthorizations
      v-if="statusOA"
      :AUTHORIZATIONS="ref(authorizations)"
      @flag="fAutorizar"
      @dataMA="sendAuthorization"
      @delMA="removeAuthorization"
    />
    <MenuComponent :adminMain="otorgarAutorizacion" @eRegresar="fRegresar"
@eChange="fChange" @ePatch="fPatch" @eDestroy="fDestroy"
@eAutorizar="fAutorizar"/>
    <AlertPop_up
      v-if="triggerAlert.get_showAlert()"
      :sm="triggerAlert.get_sm()"
      :smstatus="triggerAlert.get_smstatus()"
      @close="handleClose"
    />
  </div>
</template>

<style scoped>
.cont__main {
  width: 100%;
  height: auto;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  justify-content: center;
}

.cont__main h2 {
```

```
  font-size: 18px;
  background-color: #c9e2f9;
  padding: .5rem 0;
  box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.3);
  border-radius: 15px;
  margin-top: 1em;
  width: 61.81vw;
  margin: auto;
}

.const__main {
  width: 100vw;
  margin: auto;
  box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.3);
}
@media (max-width: 768px) {
  .cont__main h2 {
    width: 85.5%;
  }
}
</style>
```