UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Aplicación web para la gestión de los comerciales en una empresa

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Catalán Calabuig, Carles

Tutor/a: Esparza Peidro, Javier

CURSO ACADÉMICO: 2022/2023

Resumen y palabras clave

La aplicación web VentaStatistics que ha sido desarrollada con el stack MERN (MongoDB, Express, React, Node.js) permite a los administradores establecer potenciales puntos de venta para comerciales y realizar un seguimiento de las ventas que realizan.

El perfil de tipo administrador podrá acceder a funciones administrativas. También podrá gestionar los potenciales puntos de venta, agregar información relevante y asignar comerciales a estos puntos. Los comerciales pueden registrar sus ventas y hacer un seguimiento mediante gráficos y estadísticas.

Los administradores tienen acceso a un panel de administración donde pueden ver un resumen de las ventas de cada comercial. Además, la aplicación incluye un calendario para que los comerciales administren sus hitos y los administradores puedan establecer hitos con fechas y descripciones específicas.

Palabras clave

Gestión comercial

Programación

MERN Stack

Arquitectura REST

Resum i paraules clau

La aplicació web VentaStatistics, que ha sigut desenvolupada amb el stack MERN (MongoDB, Express, React, Node.js), permet als administradors establir potencials punts de venda per a comercials i realitzar un seguiment de les vendes que realitzen.

El perfil de tipus administrador podrà accedir a funcions administratives. També podrà gestionar els potencials punts de venda, afegir informació rellevant i assignar comercials a aquests punts. Els comercials poden registrar les seues vendes i fer un seguiment mitjançant gràfics i estadístiques.

Els administradors tenen accés a un panell d'administració on poden veure un resum de les vendes de cada comercial. A més a més, l'aplicació inclou un calendari perquè els comercials administren els seus assoliments i els administradors puguen establir assoliments amb dates i descripcions específiques.

Paraules clau

Gestió comercial

Programació

MERN Stack

Arquitectura REST

Summary and keywords

The web application VentaStatistics, developed with the MERN stack (MongoDB, Express, React, Node.js), allows administrators to establish potential sales points for salespeople and track their sales.

The administrator profile can access administrative functions. They can also manage potential sales points, add relevant information, and assign salespeople to these points. Salespeople can register their sales and track them using charts and statistics.

Administrators have access to an administration panel where they can view a summary of each salesperson's sales. Additionally, the application includes a calendar for salespeople to manage their milestones, and administrators can set milestones with specific dates and descriptions.

Keywords

Sales Management

Programming

MERN Stack

REST architecture

Índice

Índice de ilustraciones

1 – Introducción

En esta sección, se proporcionará una introducción a la motivación detrás de este trabajo de fin de grado. Además, se realizará un breve estudio del mercado de productos similares al que se busca implementar, explicando las razones que me impulsaron a desarrollar esta aplicación, centrándome principalmente en su utilidad para las empresas.

1.1 – Motivación

Gestionar la información de ventas es de vital importancia en el entorno laboral actual. En un mundo en constante cambio, el control de la facturación se ha convertido en uno de los aspectos más cruciales para el éxito empresarial. Una aplicación de seguimiento de ventas y gestión de potenciales puntos de venta ofrece una solución eficiente al centralizar y automatizar esta información, lo que a su vez ahorra tiempo y recursos en la gestión.

Nuestro software proporcionaría numerosas ventajas para las empresas. En primer lugar, al proporcionar estadísticas y gráficos detallados, facilitaría la toma de decisiones informadas y estratégicas. La identificación de áreas de venta y el establecimiento de estrategias efectivas se vuelven más precisos y eficientes.

Además, la aplicación permitiría un seguimiento exhaustivo de las ventas anuales, lo que brinda una supervisión constante y la posibilidad de tomar medidas correctivas de manera oportuna. Asimismo, ayudaría a los comerciales a mantenerse organizados y enfocados en sus objetivos a través del calendario de hitos, lo que aumentaría su productividad y logro de metas.

La comunicación y colaboración entre administradores y comerciales se verían mejoradas gracias a la coordinación y sincronización que brinda el software. Esto facilitaría la transmisión de información relevante, la asignación de tareas y la optimización del trabajo en equipo.

En resumen, nuestro software de seguimiento de ventas y gestión de puntos de venta ofrece una solución integral para optimizar la gestión de ventas en las empresas. Proporciona eficiencia, mejora la toma de decisiones estratégicas, garantiza un seguimiento detallado de las ventas, fomenta la organización y planificación, y promueve una comunicación efectiva y colaboración entre todo el equipo. Es una valiosa herramienta para aumentar la productividad y maximizar los resultados en el ámbito de las ventas.

1.2 – Estudio de mercado

Efectuaremos un análisis de las siguientes aplicaciones:

- **SalesLoft**: Solución integral de gestión de ventas que brinda a los equipos comerciales las herramientas necesarias para asignar puntos de venta, realizar seguimiento de ventas y alcanzar hitos en un calendario, todo ello con el objetivo de aumentar la productividad y maximizar los resultados en el proceso de ventas.

- **Outreach**: Ofrece análisis y métricas en tiempo real que ayudan a los equipos a evaluar el rendimiento, identificar áreas de mejora y tomar decisiones informadas. La plataforma se integra con otras herramientas y sistemas CRM populares

- **VanillaSoft**: Proporciona una solución integral que combina herramientas de automatización, gestión de leads y comunicación con los clientes.

- **Nimble**: Es una herramienta de gestión de relaciones con clientes (CRM) que permite asignar puntos de venta a comerciales, dar seguimiento a las ventas y establecer hitos en un calendario. Uno de sus puntos fuertes es la integración con las redes sociales.

- **Base CRM**: Ofrece funcionalidades de automatización de ventas, como la gestión de tareas, la programación de correos electrónicos y la generación de informes.

La intención de la aplicación es mantener una metodología de integración continua y así eventualmente integrar todas las funcionalidades en una misma plataforma, ya que los usuarios prefieren la centralización de las funcionalidades en un mismo sitio.

1.3 – Objetivos del proyecto

A continuación son presentados los objetivos principales del trabajo, que es el desarrollo de una aplicación web con el propósito de alcanzar una serie de objetivos empresariales y funcionales. Estos objetivos abarcan tanto las ventajas esperadas por los usuarios de la aplicación como las capacidades y funcionalidades que el software proporcionará. Es esencial comprender esta distinción para comprender el enfoque y los beneficios del proyecto en su conjunto.

En cuanto a los objetivos empresariales, se refieren a las ventajas y mejoras que los usuarios esperan obtener mediante el uso de la aplicación desarrollada. Estas ventajas pueden incluir, por ejemplo, la optimización de la gestión de ventas, el aumento

de la eficiencia en los procesos comerciales, la toma de decisiones informadas y estratégicas, y la maximización de los resultados en el ámbito de las ventas. Estos objetivos están estrechamente vinculados a los beneficios que las empresas desean lograr al implementar esta solución tecnológica.

Por otro lado, los objetivos funcionales se refieren a las capacidades y funcionalidades específicas que el software proporcionará a los usuarios. Estas capacidades pueden incluir el seguimiento detallado de las ventas, la gestión de potenciales puntos de venta, la generación de estadísticas y gráficos, la organización a través de un calendario de hitos, la facilitación de la comunicación y colaboración entre los diferentes roles de administradores y comerciales, entre otras. Estas capacidades son fundamentales para cumplir con los objetivos empresariales y brindar una solución integral y eficiente.

1.3.1 – Objetivos empresariales

- **Automatización y eficiencia**: El objetivo es optimizar los procesos de gestión de ventas y seguimiento de comerciales. La aplicación busca automatizar tareas manuales, reducir errores y ahorrar tiempo en actividades administrativas, lo que se traduce en una mayor eficiencia operativa.

- **Mejora en la toma de decisiones**: La aplicación proporciona información y estadísticas sobre las ventas realizadas por los comerciales. El objetivo es permitir a los administradores tomar decisiones más informadas y estratégicas, identificar oportunidades de mejora y asignar recursos de manera efectiva.

- **Seguimiento y control**: La aplicación permite a los administradores realizar un seguimiento de las ventas realizadas por los comerciales por años. El objetivo es tener una visión clara del rendimiento de cada comercial, identificar desviaciones y tomar medidas correctivas de manera oportuna.

- **Organización y planificación**: La inclusión de un calendario con hitos permite establecer objetivos y metas para los comerciales. El objetivo es mejorar la organización y la planificación de actividades, así como proporcionar una guía clara para los comerciales en su trabajo diario.

- **Comunicación y colaboración**: La aplicación facilita la comunicación entre administradores y comerciales, permitiendo el intercambio de información relevante y la colaboración en tiempo real. El objetivo es fomentar la colaboración, el trabajo en equipo y la alineación de objetivos.

1.3.1 – Objetivos funcionales

- **Establecer potenciales puntos de venta**: El objetivo es permitir al administrador crear y gestionar los potenciales puntos de venta donde los comerciales realizarán sus actividades de ventas. Esto implica la capacidad de agregar nueva información sobre los puntos de venta, como ubicación, descripción y detalles relevantes.

- **Asignación de comerciales a puntos de venta**: El objetivo es permitir al administrador asignar a los comerciales a los puntos de venta correspondientes. Esto implica la capacidad de asociar a cada comercial con uno o varios puntos de venta, lo que define su área de responsabilidad y trabajo.

- **Seguimiento de ventas**: El objetivo es proporcionar un seguimiento anual de las ventas realizadas por los comerciales en cada punto de venta. Esto implica registrar y almacenar información sobre las ventas, como productos vendidos, cantidad y fecha, para poder analizar y evaluar el desempeño de cada comercial y punto de venta.

- **Calendario de hitos**: El objetivo es permitir a los comerciales establecer hitos en un calendario para marcar tareas, metas o eventos importantes relacionados con su trabajo. Esto implica la capacidad de programar y visualizar los hitos en un calendario, lo que ayuda a los comerciales a mantenerse organizados y cumplir con sus objetivos.

- **Análisis y generación de informes**: El objetivo es proporcionar herramientas de análisis que permitan al administrador evaluar el desempeño de los comerciales y los puntos de venta. Esto implica la capacidad de generar gráficos que muestren estadísticas de ventas, tendencias y resultados clave, lo que ayuda en la toma de decisiones y la identificación de áreas de mejora.

1.4 - Estructura de la memoria

La memoria constará de los siguientes apartados:

- **Apartado 2**. Análisis y definición del problema.

  Listado de los requisitos funcionales y características de la aplicación en detalle.

  Diagramas de casos de uso UML.

- **Apartado 3**. Diseño de la aplicación y solución de la problemática que nos plantea.

  Arquitectura de la aplicación.

  Descripción a alto nivel la arquitectura de la aplicación, incluyendo las capas principales (GUI, lógica de negocio, capa de datos) y cómo se conectan entre sí.

  Tecnologías utilizadas, explicación de las tecnologías utilizadas.

  Descripción de las capas y componentes de la aplicación a alto nivel.

- **Apartado 4**. Demostración de la aplicación.

  Pequeña demostración de la aplicación con capturas de pantalla y elementos visuales para mostrar las funcionalidades implementadas y recorrer la aplicación paso a paso.

- **Apartado 5**. Conclusiones y trabajos futuros:

  Conclusiones obtenidas a partir del proyecto, impresiones generales y logros alcanzados.

  Trabajos futuros y mejoras que se podrían implementar para complementar o ampliar la aplicación en el futuro.

- **Apartado 6**. Bibliografía:

  Bibliografía utilizada durante el proyecto.

- **Apartado 7**. Anexos:

  Documentación adicional de la preparación del entorno.

2 – Análisis

Imaginemos una pequeña empresa PYME que se dedica a la venta de productos. La empresa cuenta con un equipo de comerciales encargados de visitar diferentes puntos de venta y realizar ventas directas a los clientes.

Sin embargo, la empresa se enfrenta a desafíos para gestionar de manera efectiva sus puntos de venta y realizar un seguimiento adecuado de las ventas realizadas por sus comerciales. Esto puede deberse a la falta de un sistema centralizado para administrar la información relacionada con los puntos de venta, así como a la dificultad de realizar un seguimiento preciso de las ventas realizadas por cada comercial.

Aquí es donde nuestro software de seguimiento de ventas y gestión de puntos de venta entra en juego. Nuestra aplicación proporcionará a la empresa una solución eficiente para controlar y gestionar sus puntos de venta. Mediante la implementación de un sistema intuitivo y fácil de usar, los administradores podrán establecer y mantener actualizada la información relevante de cada punto de venta, como ubicación, descripción y detalles adicionales

Además, con la funcionalidad de seguimiento de ventas, los comerciales podrán registrar sus transacciones de manera sencilla y precisa. Esto permitirá un seguimiento detallado de las ventas realizadas, generando informes y estadísticas que brindarán una visión clara del rendimiento individual y colectivo de los comerciales.

El calendario de hitos es otra característica esencial de nuestra aplicación. Con esta herramienta, los comerciales podrán establecer metas y tareas importantes en sus actividades diarias. Esto promoverá una gestión más efectiva del tiempo y los recursos, mejorando así el desempeño y los resultados.

Esta sería la definición de la primera iteración del proyecto, ya que más adelante sería optimo la integración con plataformas de gestión de recursos empresariales (ERP) u otras plataformas de misma naturaleza.

2.1 – Diagrama de casos de uso (UML)

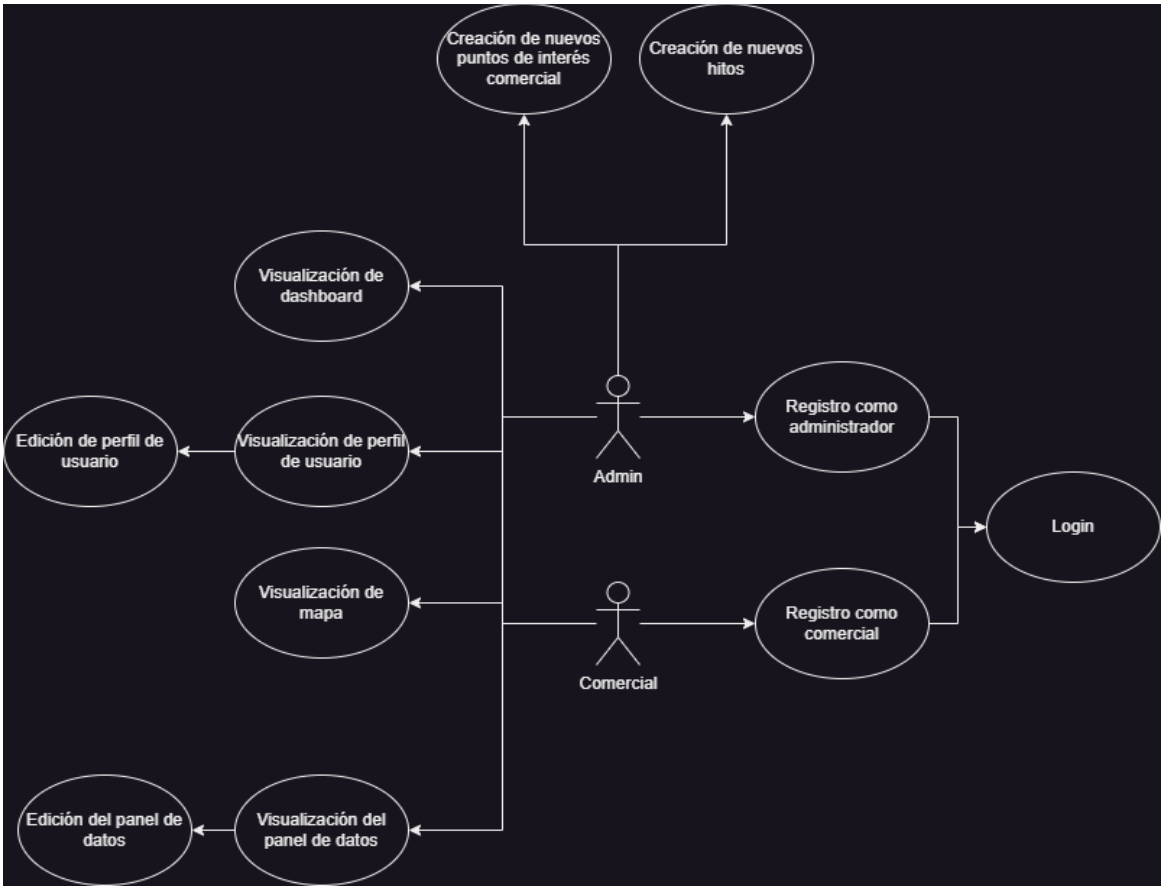


Ilustración 1: Diagrama de casos de uso UML

2.2 – Gestión de usuarios

El registro de sesión es un proceso fundamental que permite a los usuarios acceder a la aplicación y utilizar las funcionalidades correspondientes según su rol, ya sea como administrador o usuario comercial. A continuación, se detalla cómo funciona este registro de sesión:

2.2.1 - Gestión como usuario:

Los comerciales pueden registrarse como usuarios proporcionando la información necesaria, como nombre, correo electrónico y contraseña.

Se verifica la validez de la información proporcionada y se crea una cuenta de usuario en la base de datos de la aplicación.

Una vez registrado, el usuario comercial puede acceder a la aplicación utilizando su correo electrónico y contraseña. Su identificador único será en el correo electrónico.

2.2.2 - Gestión como administrador:

El administrador, que posee una clave única, puede acceder a una función especial de registro.

Se verifica la clave única del administrador y se le permite acceder a la aplicación.

El administrador puede utilizar su propia cuenta con credenciales especiales para acceder a la vista de administrador.

2.2.3 - Acceso a funcionalidades:

Después de iniciar sesión, la aplicación verifica el rol del usuario (administrador o comercial) y le muestra las funcionalidades correspondientes según su perfil.

Los administradores tienen acceso a todas las funcionalidades y características de gestión, como establecer potenciales puntos de venta así como asignar metas comerciales o hitos.

Los usuarios comerciales tienen acceso a las funcionalidades relacionadas con su trabajo diario, como registrar ventas, ver su calendario de hitos o ver su progreso en las ventas anuales.

2.2.4 - Opciones de cierre de sesión:

Se debe incluir un botón o enlace para cerrar sesión de manera segura en la aplicación. Esto permite al usuario/administrador salir de la aplicación cuando ya no la necesite, protegiendo la privacidad y la seguridad de su cuenta.

2.2.5 - Seguridad y autenticación:

El registro de sesión debe implementar medidas de seguridad para proteger la información confidencial y garantizar que solo los usuarios autorizados puedan acceder a

la aplicación.

Esto implica el uso de técnicas de autenticación segura, como el almacenamiento seguro de contraseñas y la transmisión de datos cifrados, en nuestro caso, haremos uso del algoritmo de encriptación HS256.

2.3 – Gestión de perfil

La pantalla de perfil de usuario/administrador proporciona a los usuarios una visión general de su información personal y les permite realizar acciones relacionadas con su cuenta y configuración.

2.3.1 - Información personal

Se muestra la información personal del usuario, como nombre, la empresa para la que trabaja, correo electrónico además de permitirnos cambiar la contraseña. Esto ayuda a identificar rápidamente la cuenta a la que se ha iniciado sesión.

2.3.2 - Foto de perfil

Se muestra una foto de perfil del usuario por defecto, esta puede ser actualizada por el usuario. Esto ayuda a personalizar la cuenta y facilita el reconocimiento visual del usuario.

2.3.3 - Datos de contacto

Se muestra la información de contacto del usuario, dirección, ciudad, país y código postal. Esto proporciona una forma de contacto adicional y facilita la comunicación con el usuario si es necesario.

2.3.4 - Sobre mi

Permite mostrar una breve descripción del usuario, permitiendo una mayor personalización y sentido de humanización. Es importante proporcionar mecanismos de personalización para un mayor sentido de vinculación corporativa.

2.3.5 - Configuración de cuenta

Se permite al usuario realizar cambios en su configuración de cuenta, como cambiar la contraseña y actualizar la información de contacto. Esto brinda al usuario el control sobre su cuenta y personalización de la aplicación según sus necesidades.

2.3.6 - Acciones de gestión de perfil

Para los usuarios comerciales y administradores será posible cambiar toda la información del perfil, exceptuando su estado de usuario comercial así como la empresa para la que trabajan.

2.4 – Gestión de estadísticas

La pantalla de dashboard es una vista centralizada que muestra un resumen visual de los datos relevantes y estadísticas clave. Incluye las siguientes gráficas y datos:

2.4.1 - Gráfica de ventas por mes

Esta gráfica muestra la cantidad de venta establecidos en cada mes. Es representada mediante una línea, donde el eje X representa los meses y el eje Y representa el número de ventas.

2.4.2 - Piechart de objetivos totales

Este gráfico circular muestra la distribución porcentual de los objetivos totales establecidos. Cada porción del pastel representa un objetivo; objetivo de ventas y ventas realizadas y su contribución relativa al objetivo total en formato porcentual.

2.4.3 - Gráfico de columnas de comparativas de ventas

Este gráfico muestra una comparativa entre las ventas realizadas y los objetivos establecidos para cada mes. Cada columna representa el desempeño de ventas en relación con los objetivos, permitiendo visualizar rápidamente si se han superado o no.

2.4.4 - Datos clave

Son mostrados datos clave relacionados con el rendimiento de ventas, como el número total de ventas, las ganancias obtenidas, la cantidad de clientes captados y la cantidad de productos vendidos. Estos datos proporcionan una instantánea rápida del rendimiento general.

2.4.5 - Listado de hitos

Se muestra un listado de los hitos establecidos, indicando cuáles de ellos han sido realizados y cuáles están pendientes. Cada hito esta marcado con una indicación visual (una marca de verificación) para identificar aquellos que han sido completados.

En resumen, la pantalla de dashboard presenta una serie de gráficas que muestran información clave relacionada con las ventas, objetivos de ventas, comparativas de ventas, así como datos cuantitativos relevantes. También se incluye un listado de hitos para mantener un seguimiento de las metas establecidas y su progreso.

2.5 – Gestión de objetivos potenciales

La pantalla del mapa de objetivos potenciales tiene como objetivo mostrar visualmente en un mapa las zonas de venta potenciales para un comercial seleccionado, así como proporcionar una lista con descripciones de cada uno de los marcadores.

A continuación, se explican los diferentes sectores de la pantalla:

2.5.1 - Selección de comercial

En la parte superior de la pantalla, se encuentra un área interactiva que permite al usuario seleccionar uno de los comerciales mediante un listado desplegable o un campo de búsqueda con filtro. Al hacer clic en el botón 'Seleccionar comercial', se abre un modal que muestra la lista de comerciales disponibles para seleccionar.

Una vez seleccionado el comercial, la pantalla se actualiza automáticamente para mostrar los potenciales puntos de venta de este comercial.

2.5.2 - Mapa de zonas de venta

En el centro de la pantalla se muestra un mapa interactivo. En este mapa, se colocan marcadores o alfileres que representan las potenciales zonas de venta relacionadas con el comercial seleccionado.

Al hacer 'click' en alguno de los marcadores será mostrada una breve descripción de la zona.

2.5.3 - Lista de marcadores con descripciones

A la izquierda del mapa, se muestra una lista vertical con los marcadores de las potenciales zonas de venta. Cada marcador en la lista puede tener una descripción breve y relevante para la zona de venta correspondiente.

Al hacer clic en un marcador de la lista, da una visión más cercana de la zona a la que apunta.

2.5.4 - Interactividad y navegación

El usuario puede interactuar con el mapa, como hacer zoom, arrastrar y explorar diferentes áreas de interés.

2.5.5 - Inserción de nuevos marcadores

Los usuarios administradores pueden actualizar e insertar nuevos marcadores en el mapa. Haciendo 'click' en este será mostrado un pequeño modal donde podremos introducir una breve descripción de la zona de interés. La pantalla será recargada de forma automática.

En resumen, la pantalla del mapa de objetivos potenciales muestra un mapa interactivo con marcadores que representan las potenciales zonas de venta para un comercial seleccionado. Junto al mapa, se proporciona una lista con descripciones de cada marker para una referencia rápida. Esta pantalla permite al usuario visualizar de manera clara y práctica las zonas de venta y obtener información detallada sobre cada una de ellas.

2.6 - Gestión de hitos

La pantalla del calendario de hitos ofrece una interfaz completa y funcional para gestionar los hitos de los comerciales. A continuación, se describe cada elemento y funcionalidad presente en esta pantalla:

2.6.1 – Selección de comercial

Primeramente, en la parte superior de la pantalla, se encuentra un botón con el texto "Selecciona un comercial". Al hacer clic en este botón, se abre un modal que muestra una lista desplegable ademas de una barra de búsqueda para seleccionar el comercial deseado. Esto permite al administrador filtrar y ver los hitos específicos de un comercial en particular.

2.6.2 – Calendario

Una vez seleccionado el comercial, se muestra el calendario en la pantalla principal. El calendario puede presentarse en diferentes vistas, como la vista de agenda, la vista semanal o la vista mensual. Esto brinda al usuario flexibilidad para visualizar los hitos en la forma que mejor se adapte a sus necesidades.

2.6.4 - Modal de información del hito

Al hacer clic en un hito en el calendario, se muestra un modal con información detallada sobre ese hito en particular. El modal puede mostrar el estado del hito (completado/no completado), descripción y fecha límite. Esta información permite al usuario comprender mejor el hito y tomar las acciones necesarias.

2.6.5 - Modal para generar un nuevo hito

Si el usuario hace clic en un día del calendario en el que no hay hitos programados, se abre un modal que permite al usuario crear un nuevo hito. El modal puede incluir campos para ingresar la descripción del hito. Esto facilita la creación rápida y precisa de nuevos hitos.

2.6.6 - Navegación por el calendario

El usuario puede navegar por el calendario hacia adelante o hacia atrás para ver los hitos en diferentes meses o semanas.

2.6.7 - Tabla con datos de rendimiento de los comerciales

En la parte inferior de la pantalla, se muestra una tabla con los datos de rendimiento de los comerciales. Estos datos incluyen la cantidad de productos vendidos,

clientes captados, número de ventas, ganancias de ventas y el objetivo total para cada comercial. Esto proporciona una visión general del rendimiento de los comerciales en un formato tabular y permite realizar comparaciones y análisis.

2.7 – Gestión de datos comerciales

La pantalla de "Panel de datos" ofrece una visualización completa de los datos relevantes, permitiendo a los administradores ver y establecer objetivos de ventas mensuales para los comerciales. A continuación, se describe en detalle cada elemento y funcionalidad presente en esta pantalla:

2.7.1 - Paginador por años

En la parte superior de la pantalla, se encuentra un paginador que permite al usuario navegar entre los años para ver los datos correspondientes a cada año. Esto proporciona una forma conveniente de acceder a datos históricos y realizar comparaciones entre diferentes períodos.

2.7.2 - Selección de comercial

Junto al paginador, hay un botón con el texto "Selecciona un comercial". Al hacer clic en este botón, se abre un modal que muestra una lista desplegable para seleccionar el comercial deseado. Esto permite al usuario filtrar y ver los datos específicos de un comercial en particular.

2.7.3 - Establecimiento de objetivos de ventas mensuales

Después de seleccionar un comercial, se muestran diferentes campos de entrada donde los administradores pueden establecer los objetivos de ventas mensuales para ese comercial en particular. Estos campos permiten ingresar los valores deseados para cada mes y se guardan para su posterior seguimiento y comparación.

2.7.4 - Establecimiento de otros objetivos

Además será posible ingresar valores para el número de ventas, clientes captados y cantidad de productos vendidos por parte de los usuarios comerciales.

3 – Diseño

En esta sección, se presentará el diseño y arquitectura de la aplicación, destacando las diferentes capas y cómo se conectan entre sí para brindar una solución integral.

3.1 – Arquitectura de la aplicación

En nuestro proyecto, hemos utilizado el patrón de diseño Modelo-Vista-Controlador (MVC) aplicado a una arquitectura de 3 capas para desarrollar nuestra aplicación.

3.1.1 – Patrón de diseño MVC

El patrón MVC es ampliamente reconocido y utilizado en el desarrollo de software debido a sus beneficios en términos de estructura, organización y mantenibilidad del código.

El patrón MVC se compone de tres componentes principales:

1. **Modelo (Model)**: El modelo representa los datos y la lógica subyacente de nuestra aplicación. Aquí es donde se manejan los datos, se realizan cálculos y se implementa la lógica de negocio. En nuestra aplicación, el modelo se encarga de interactuar con la base de datos no relacional y proporcionar los datos necesarios para la vista y el controlador.

2. **Vista (View)**: La vista es la capa de presentación de nuestra aplicación. Es responsable de mostrar la información al usuario y proporcionar una interfaz con la que puedan interactuar. En nuestro caso, la vista mostrará los datos del modelo de una manera comprensible y proporcionará opciones para que el usuario interactúe con la aplicación.

3. **Controlador (Controller)**: El controlador actúa como intermediario entre el modelo y la vista. Recibe las acciones y las entradas del usuario desde la vista y las maneja apropiadamente. También se encarga de actualizar el modelo en función de las acciones del usuario y de actualizar la vista en consecuencia. El controlador asegura que la comunicación entre el modelo y la vista sea fluida y separa la lógica de negocio del código de presentación.

Hemos utilizado el patrón MVC en nuestra aplicación por varias razones:

1. **Separación de responsabilidades**: El patrón MVC nos permite separar claramente las responsabilidades entre el modelo, la vista y el controlador. Esto facilita el desarrollo y la comprensión del código, ya que cada componente tiene un propósito definido y no se mezclan las lógicas de presentación y de negocio.

2. **Modularidad y reutilización de código**: Al dividir la aplicación en componentes distintos, el código se vuelve más modular y reutilizable. Podemos hacer cambios en el modelo sin afectar la vista o el controlador, lo que facilita la escalabilidad y el mantenimiento a largo plazo.

3. **Soporte de múltiples interfaces de usuario**: El patrón MVC nos permite desarrollar interfaces de usuario diferentes utilizando el mismo modelo y controlador. Esto es especialmente útil cuando tenemos que desarrollar pantallas para diferentes tipos de usuario, como en nuestro caso para administrador y usuario comercial.

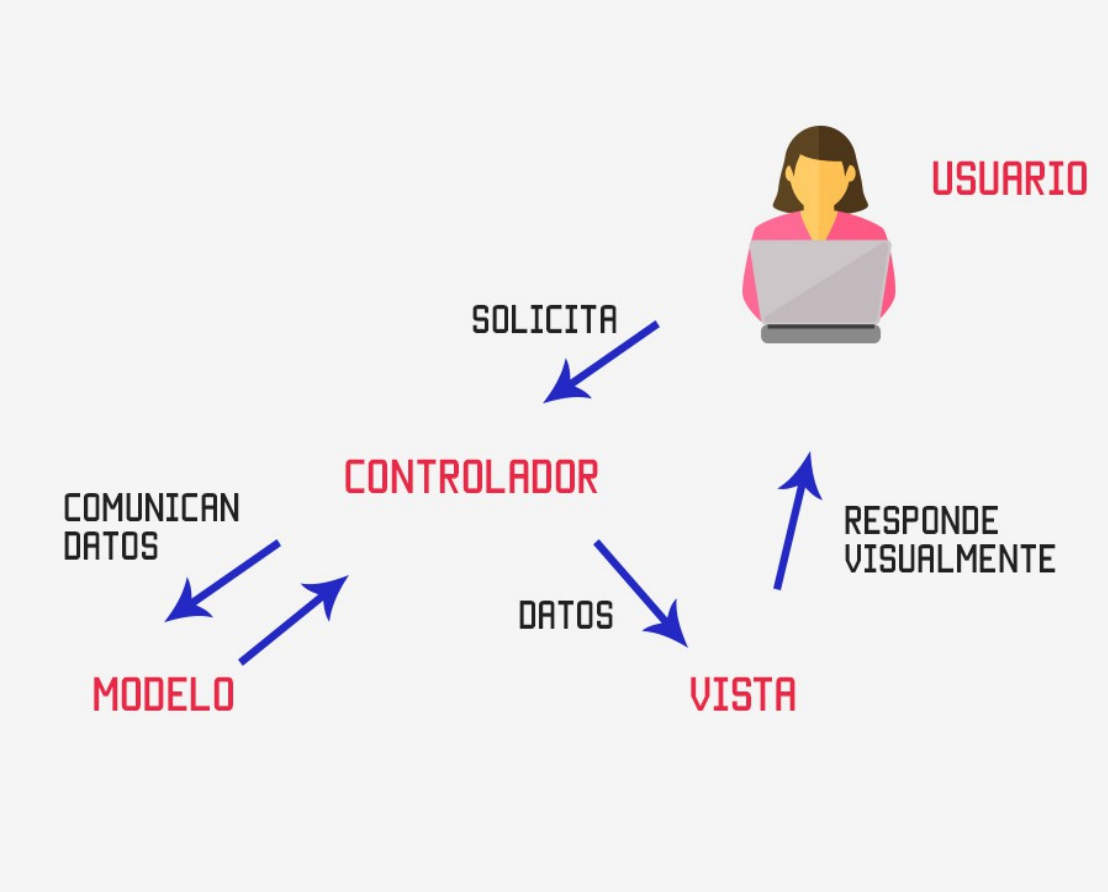


Ilustración 2: MVC

3.1.2 - Arquitectura de 3 capas

Respecto a la arquitectura, se basa en el estilo de arquitectura de 3 capas.

Capa 1 (GUI): Esta capa es la que interactúa directamente con los usuarios. Su función principal es presentar la información y permitir la interacción con el sistema. Aquí se encuentran los elementos visuales, la lógica de presentación y la gestión de la interfaz de usuario.

Capa 2 (Servicio REST): En esta capa, se encuentra la lógica y la funcionalidad central de la aplicación. Se encarga de procesar y gestionar los datos, realizar operaciones, aplicar reglas de negocio y garantizar que la aplicación funcione correctamente.

Capa 3 (Base de datos MongoDB): Esta capa se encarga de interactuar con la base de datos u otros medios de almacenamiento de datos. Es responsable de recuperar, almacenar y actualizar la información necesaria para que la lógica de negocio funcione adecuadamente.

El objetivo principal de este enfoque es separar las preocupaciones y mantener un alto nivel de modularidad, lo que facilita el mantenimiento, la escalabilidad y el desarrollo paralelo de diferentes partes de la aplicación.

Algunos conceptos importantes aplicados a las capas son los siguientes:

Arquitectura REST (Capa 2): REST es un estilo arquitectónico que se basa en el protocolo HTTP y utiliza sus verbos (GET, POST, PUT, DELETE, etc.) para realizar operaciones sobre los recursos. En este enfoque, los recursos son identificados por URLs y se manipulan a través de las operaciones HTTP adecuadas. Permite una comunicación cliente-servidor sin estado, lo que significa que cada solicitud del cliente contiene toda la información necesaria para comprenderla.

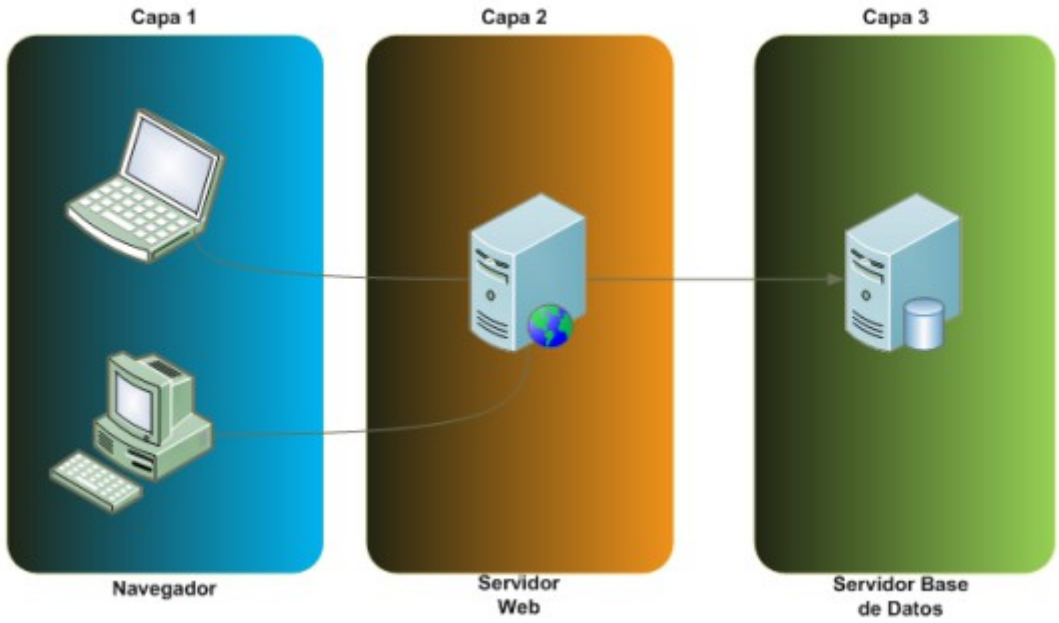
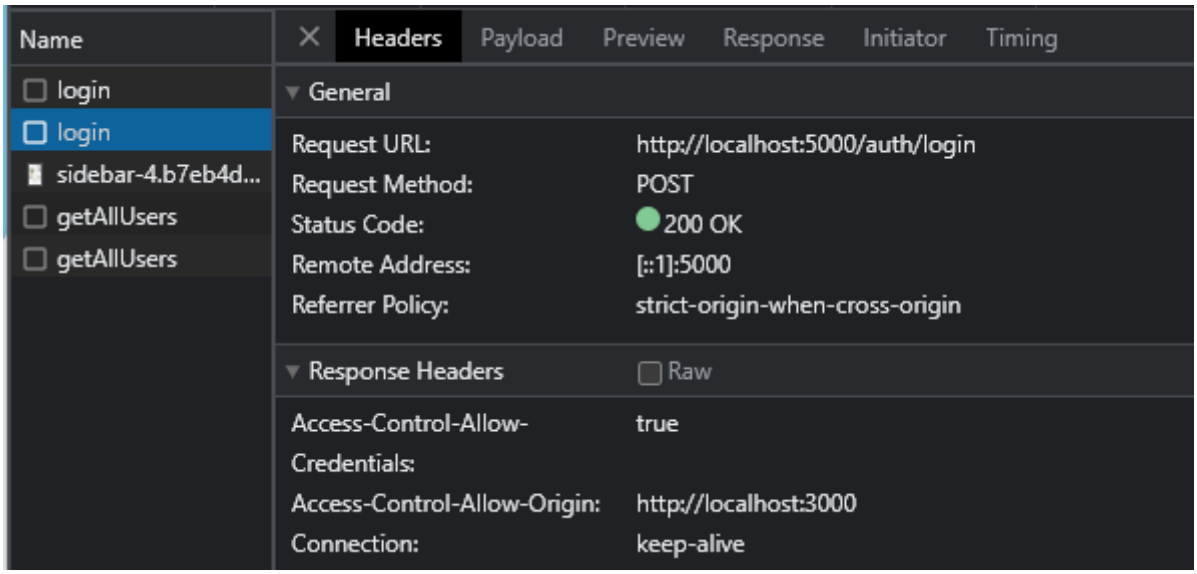


Ilustración 3: REST

Client-side rendering con React (Capa 1): En esta arquitectura, utilizamos React, un framework de JavaScript para construir interfaces de usuario interactivas. React permite el rendering del lado del cliente, lo que significa que la mayor parte del procesamiento y generación de la interfaz de usuario se realiza en el navegador del cliente en lugar de en el servidor. Esto mejora la experiencia del usuario al reducir la latencia y permitir una interfaz de usuario más fluida y receptiva. React utiliza un enfoque basado en componentes para construir la interfaz de usuario, lo que facilita la reutilización y el mantenimiento del código.

Comunicación cliente-servidor vía AJAX (Capa 2): En nuestra arquitectura, cuando se realizan solicitudes AJAX desde el cliente al servidor, los datos se envían y reciben en formato JSON. Esto implica que el cliente empaqueta la información en objetos JSON y la envía al servidor a través de una solicitud AJAX, y el servidor procesa la solicitud y devuelve la respuesta en formato JSON



MongoDB como base de datos (Capa 3): Para almacenar los datos de nuestra aplicación, hemos utilizado MongoDB, una base de datos NoSQL orientada a documentos. MongoDB almacena los datos en formato BSON (Binary JSON) y permite una gran flexibilidad en la estructura de los documentos almacenados. No requiere un esquema fijo y admite consultas rápidas y escalables. MongoDB se integra bien con aplicaciones modernas y es adecuado para escenarios donde se necesitan cambios frecuentes en la estructura de los datos.

3.2 – Tecnologías utilizadas

A continuación describiremos con mas detalle las tecnologías que hemos utilizado en el desarrollo del proyecto:

3.2.1 – Tecnologías en Capa 1: GUI

React: Es un popular framework de JavaScript utilizado para construir interfaces de usuario interactivas y eficientes.

Las principales características de React son las siguientes:

- Componentes: React se basa en un enfoque basado en componentes, lo que significa que la interfaz de usuario se divide en componentes reutilizables. Los componentes pueden ser pequeñas piezas de la interfaz, como botones o formularios, o componentes más grandes que encapsulan funcionalidades más complejas. Los componentes de React se componen entre sí para construir la interfaz de usuario completa.

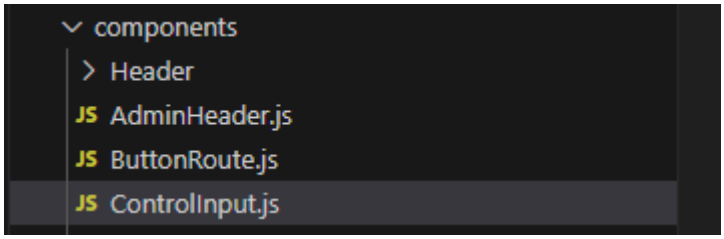


Ilustración 4: Componentes

Estos son algunos de los componentes usados, como hemos comentado, pueden ser componentes más grandes como el header o pequeños componentes altamente reutilizados a lo largo de la aplicación como un control de entrada de texto.

- **Virtual DOM**: React utiliza un Virtual DOM (Documento de Objeto Modelo Virtual) para mejorar el rendimiento de las actualizaciones de la interfaz de usuario. El Virtual DOM es una representación en memoria de la estructura de la interfaz de usuario. Cuando se realizan cambios en los datos que afectan a la interfaz de usuario, React compara el Virtual DOM anterior con el nuevo y aplica solo los cambios necesarios para actualizar la interfaz de usuario en el navegador. Esto reduce la carga en el navegador y mejora el rendimiento de la aplicación.

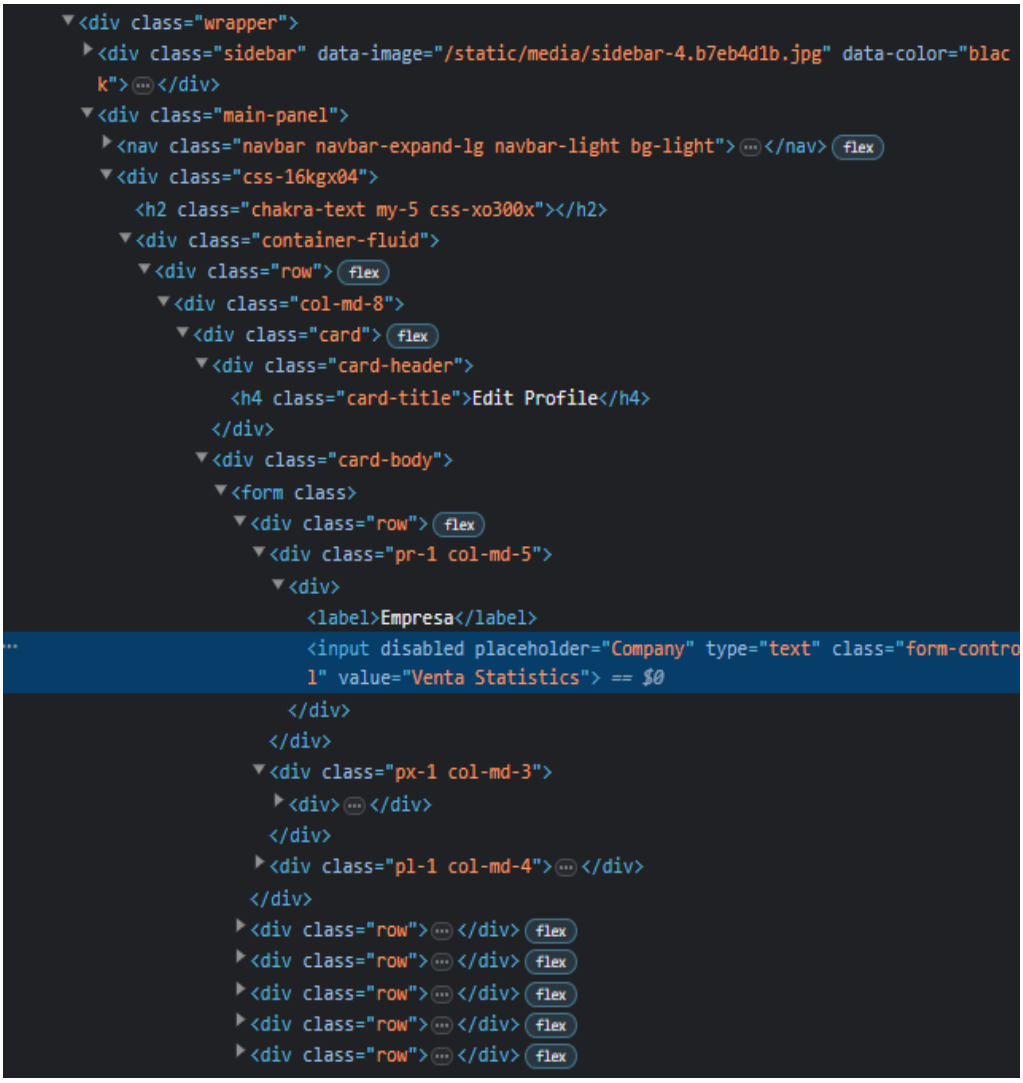


Ilustración 5: Representación visual del DOM Tree

- **JSX**: React utiliza JSX, una extensión de sintaxis que permite escribir código HTML-like dentro de JavaScript. JSX combina HTML y JavaScript en un solo archivo, lo que facilita la construcción de componentes y el renderizado de elementos en la interfaz de usuario. JSX se compila a JavaScript puro antes de ser interpretado por el navegador.

```
return (
  <AuthLayout
    footer={
      <Stack spacing={6}>
        <Button
          isLoading={loading}
          onClick={handleSubmit(onSubmit)}
          colorScheme={"blue"}
          variant={"solid"}
        >
          Sign in
        </Button>
        <Flex justifyContent="space-between" style={{marginLeft: "-20px"}}>
          <ButtonRoute route={PAGE_KEYS.ForgotPassword}>
            <Button
              colorScheme="blue"
              variant="outline"
              size="sm" // Establece el tamaño del botón a "sm" (pequeño)
            >
              ¿Olvidaste tu contraseña?
            </Button>
          </ButtonRoute>
          <ButtonRoute route={PAGE_KEYS.RegisterPage}>
            <Button colorScheme="blue" variant="outline" size="sm">
              Registrarse!
            </Button>
          </ButtonRoute>
```

Ilustración 6: Ejemplo de JSX

- **Unidireccionalidad de datos**: React sigue el flujo de datos unidireccional, lo que significa que los datos fluyen desde el componente padre hacia los componentes hijos. Esto facilita el seguimiento de los cambios de datos y mantiene una estructura clara y predecible en la aplicación.

- **Reutilización de componentes**: La naturaleza modular de React permite una fácil reutilización de componentes. Los componentes de React pueden ser utilizados en diferentes partes de la aplicación, lo que ahorra tiempo de desarrollo y promueve la consistencia en la interfaz de usuario.

```
return (
  <Box>
    <ControlInput
      name={"name"}
      control={control}
      rules={{
        required: "Escribe el nombre!",
        minLength: {
          value: 4,
          message: "El nombre debe tener al menos 4 caracteres!",
        },
        maxLength: {
          value: 50,
          message: "El nombre debe tener un máximo de 50 caracteres!",
        },
      }}
      errorMessage={errors?.name?.message}
    />
```

Ilustración 7: Utilización del componente Control Input

- **Comunidad y ecosistema**: React cuenta con una comunidad activa y un amplio ecosistema de bibliotecas y herramientas complementarias. Esto incluye bibliotecas populares como React Router para el enrutamiento, Redux para la gestión del estado y Axios para realizar solicitudes HTTP, entre muchas otras.

```
<Nav>
    <li>
        <div className="nav-link" onClick={() => history.push(PAGE_KEYS.HomePage)}>
        <i className="nc-icon nc-chart-pie-35" />
        <p>Dashboard</p>
        </div>
    </li>
</Nav>
<Nav>
    <li>
        <div className="nav-link" onClick={() => history.push(PAGE_KEYS.ProfilePage)}>
        <i className="nc-icon nc-circle-09" />
        <p>Perfil de usuario</p>
        </div>
    </li>
</Nav>

<Nav>
    <li>
        <div className="nav-link" onClick={() => history.push(PAGE_KEYS.ChatPage)}>
        <i className="nc-icon nc-pin-3" />
        <p>Mapa</p>
        </div>
    </li>
</Nav>
```

Ilustración 8: Uso de react-router en la variable 'history'

CSS (Cascading Style Sheets): Es un lenguaje de hojas de estilo utilizado para definir el aspecto y la presentación de un documento HTML. CSS se utiliza para describir cómo se deben mostrar los elementos HTML en una página web, controlando aspectos como colores, fuentes, márgenes, alineación, diseños y efectos visuales.

Las características principales de CSS son las siguientes:

- **Selectores**: Los selectores de CSS se utilizan para identificar y seleccionar los elementos HTML a los que se les aplicarán los estilos. Los selectores pueden basarse en el nombre de la etiqueta, el atributo class o id, la relación con otros elementos y otros criterios.

```css
.all-icons .font-icon-detail {
  text-align: center;
  padding: 45px 0px 30px;
  border: 1px solid #e5e5e5;
  border-radius: 6px;
  margin: 15px 0;
}
.all-icons .font-icon-detail input {
  margin: 25px auto 0;
  width: 100%;
  text-align: center;
  display: block;
  color: #aaa;
  font-size: 13px;
}

#map {
  position: relative;
  width: 100%;
  height: 100%;
}

.map-container {
  width: 100%;
  height: 100vh;
  max-height: 100vh;
}
```

Ilustración 9: CSS del container mapa

- **Propiedades y valores**: CSS utiliza una amplia gama de propiedades y valores para definir el aspecto de los elementos seleccionados. Por ejemplo, las propiedades pueden controlar el tamaño de fuente, el color del texto, los márgenes, los bordes, las sombras, los fondos y muchas otras características visuales.

- **Cascada y especificidad**: La "Cascada" en CSS se refiere al proceso mediante el cual se resuelven los conflictos entre estilos aplicados a un elemento. CSS sigue reglas de especificidad para determinar qué estilo tiene prioridad en caso de conflictos. Los estilos pueden ser heredados de elementos padres y se pueden anular o modificar utilizando reglas más específicas.

- **Clases y ID**: Las clases y los ID son atributos que se pueden agregar a los elementos HTML para aplicar estilos específicos. Las clases permiten aplicar estilos a varios elementos que comparten la misma clase, mientras que los ID identifican de manera única un elemento para aplicar un estilo específico.

Bootstrap: Bootstrap es un framework de diseño front-end.

Bootstrap nos ha proporcionado un enfoque rápido para el desarrollo de las interfaces gracias al aprovechamiento de las clases y componentes predefinidos, así pues ha sido posible desarrollar una interfaz coherente y comprensiva de forma rápida y homogénea.

Algunas de las características de bootstrap son las siguientes:

- **Grid system**: Bootstrap proporciona un sistema de cuadrícula flexible y receptivo que permite organizar y distribuir el contenido en diferentes tamaños de pantalla. El sistema de cuadrícula de Bootstrap utiliza clases CSS predefinidas para crear columnas y filas, lo que facilita la creación de diseños flexibles y adaptables a diferentes dispositivos.

- **Componentes predefinidos**: Bootstrap incluye una amplia gama de componentes y elementos de interfaz de usuario predefinidos, como botones, formularios, navegación, alertas, carruseles, tarjetas y muchos más. Estos componentes se implementan utilizando clases CSS y se pueden personalizar y combinar para adaptarse a los requisitos específicos del proyecto.

- **Tipografía y estilos prediseñados**: Bootstrap proporciona un conjunto de estilos y clases CSS predefinidas que permiten una tipografía y estilos consistentes en todo el sitio. Esto incluye opciones de estilos de texto, encabezados, listas, tablas y otras etiquetas HTML.

- **Responsividad**: Bootstrap está diseñado para ser completamente responsivo, lo que significa que los sitios web construidos con Bootstrap se adaptan automáticamente a diferentes tamaños de pantalla, como dispositivos móviles, tabletas y computadoras de escritorio. Esto se logra mediante el sistema de

cuadrícula y las clases CSS responsivas de Bootstrap.

3.2.2 – Tecnologías en Capa 2: Servicio REST

Node.js: Node.js es un entorno de tiempo de ejecución de JavaScript basado en el motor de JavaScript V8 de Chrome. A diferencia de JavaScript en el navegador, que se ejecuta en el cliente, Node.js permite ejecutar código JavaScript en el lado del servidor.

Las características principales de Node.js son las siguientes:

- **Entorno de tiempo de ejecución del lado del servidor**: Node.js permite ejecutar código JavaScript en el lado del servidor.

- **Basado en el motor V8 de Chrome**: Node.js utiliza el motor V8 de JavaScript desarrollado por Google para compilar y ejecutar el código JavaScript. El motor V8 es altamente eficiente y rápido, lo que permite que las aplicaciones desarrolladas en Node.js sean escalables y de alto rendimiento.

Ilustración 10: Icono del Motor V8

- **Event-Driven y Non-Blocking I/O**: Node.js se basa en un modelo de E/S (entrada/salida) no bloqueante y basado en eventos. Esto significa que, en lugar de esperar a que una operación de E/S se complete antes de continuar, Node.js permite que el programa continúe ejecutando otras tareas mientras espera. Esto hace que Node.js sea muy eficiente en el manejo de solicitudes concurrentes y en la construcción de aplicaciones en tiempo real.

- **Node.js** es especialmente adecuado para aplicaciones de red, servicios API, aplicaciones en tiempo real y aplicaciones web escalables. Al aprovechar JavaScript tanto en el lado del cliente como en el lado del servidor, Node.js permite a los desarrolladores utilizar un lenguaje coherente en todos los aspectos de una aplicación web.

Ilustración 11: package.json de la parte servidora

- **NPM** (Node Package Manager): Node.js viene con NPM, que es el gestor de paquetes oficial de Node.js. NPM permite a los desarrolladores acceder a una amplia variedad de bibliotecas y módulos de terceros para facilitar el desarrollo de aplicaciones. Con NPM, los desarrolladores pueden instalar, administrar y actualizar fácilmente las dependencias de sus proyectos.

- **Amplia comunidad y ecosistema**: Node.js cuenta con una comunidad activa y un ecosistema próspero. Hay numerosas bibliotecas, marcos y herramientas disponibles que facilitan el desarrollo en Node.js. Algunos ejemplos populares son Express.js para la construcción de aplicaciones web, Socket.io para la comunicación en tiempo real y Mongoose para la interacción con bases de datos MongoDB.

Ilustración 12: Declaración de Express y Mongoose en AppServer

3.2.3 – Tecnologías en Capa 3: Base de datos MongoDB

MongoDb: Es un sistema de gestión de bases de datos NoSQL (Not Only SQL) de código abierto y orientado a documentos. A diferencia de las bases de datos relacionales tradicionales, MongoDB no utiliza tablas y filas, sino que almacena los datos en documentos flexibles en formato BSON (Binary JSON).

Ilustración 13: Icono de MongoDB

Las características principales de MongoDB son las siguientes:

- **Estructura de datos flexible**: MongoDB utiliza una estructura de documentos flexible que permite almacenar datos de forma dinámica y sin un esquema fijo. Cada documento puede contener diferentes campos con diferentes tipos de datos, lo que facilita la adaptación a cambios en la estructura de los datos.

- **Escalabilidad horizontal**: MongoDB es altamente escalable y permite el escalado horizontal al distribuir los datos en múltiples servidores o clústeres. Esto permite manejar grandes volúmenes de datos y soportar aplicaciones de alto rendimiento.

- **Consultas y operaciones avanzadas**: MongoDB proporciona un poderoso lenguaje de consultas que permite realizar operaciones avanzadas, como consultas complejas, agregaciones, indexación y búsqueda de texto completo. Esto facilita la manipulación y extracción de datos de manera eficiente.

- **Integración con lenguaje de programación**: MongoDB cuenta con controladores (drivers) disponibles para varios lenguajes de programación, lo que facilita la integración y la interacción con la base de datos en diferentes entornos de desarrollo.

- **Adopción de la comunidad y soporte empresarial**: MongoDB cuenta con una comunidad activa y una amplia base de usuarios, lo que ha impulsado su adopción y desarrollo continuo. Además, MongoDB ofrece opciones de soporte y servicios empresariales para empresas que requieren asistencia adicional.

3.3 – Capa 1: GUI

Primero, explicaremos la estructura de directorios que seguirá la capa cliente, seguidamente procederemos a explicar la capa cliente de nuestra aplicación web. La capa cliente es la parte de la aplicación que se ejecuta en el navegador web y es responsable de la interfaz de usuario, la interacción con el usuario y la presentación de datos. La capa cliente sigue el patrón MVC, explicado a continuación.

3.3.0 – Estructura de directorios

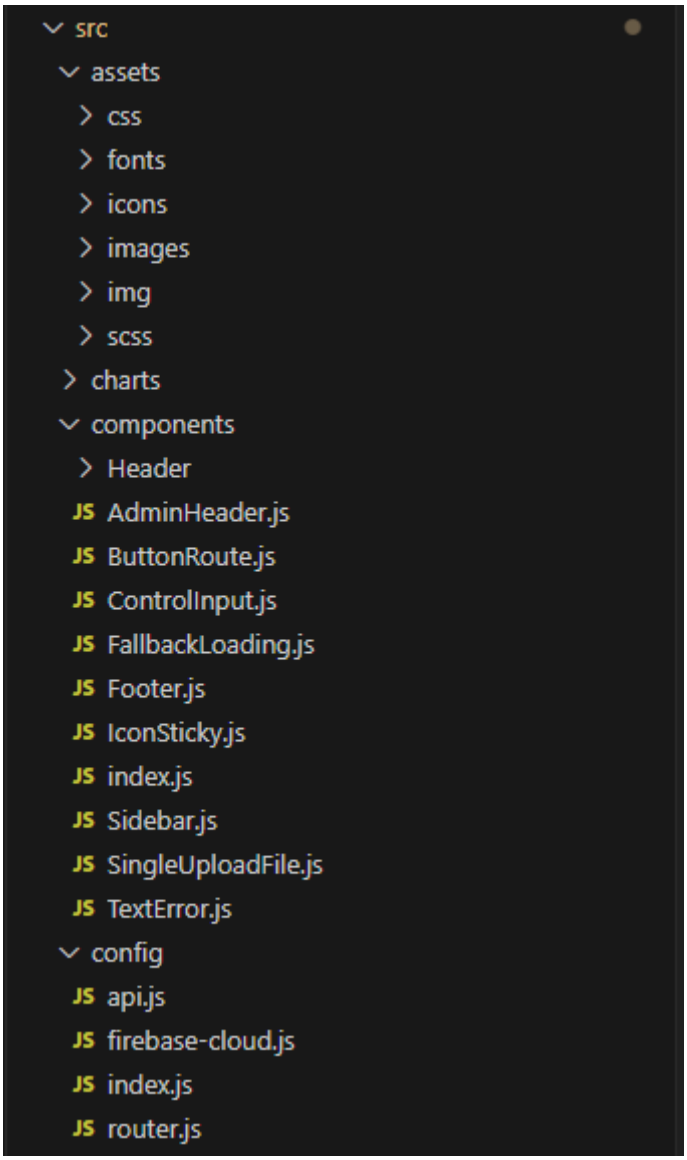


Ilustración 14: Estructura de directorios 1

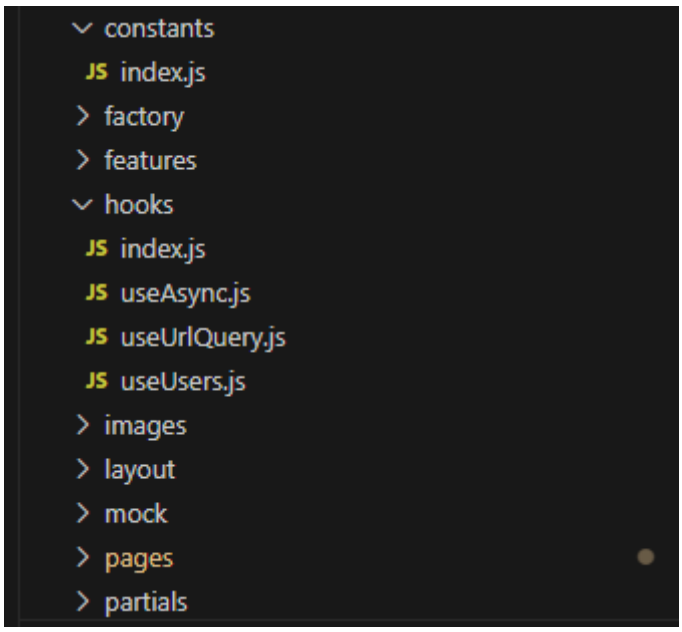


Ilustración 15: Estructura de directorios 2

Como directorios clave destacar:

- **components/**: Aquí es donde se ubican los componentes reutilizables de la aplicación, como encabezados, pies de página, barras laterales, etc.

- **pages/**: En este directorio, se ubican los componentes que representan páginas individuales de la aplicación. Cada página puede estar compuesta por uno o más componentes reutilizables.

- **App.js**: El componente principal de la aplicación React. Aquí se combinan los componentes y se define la estructura general de la aplicación.

- **index.js**: Este archivo es el punto de entrada de la aplicación React. Es el archivo que se ejecuta primero y renderiza el componente principal (App.js) en el elemento con el id "root" en el archivo public/index.html.

3.3.1 – Modelo

Cuando un usuario interactúa con la vista (por ejemplo, envía un formulario), el controlador captura esa interacción y utiliza la librería **Axios** para realizar una solicitud a la API REST. Esta solicitud puede ser una petición para obtener datos (por ejemplo, la llamada getAllUsers) o enviar datos al servidor (por ejemplo, la llamada updateUser para actualizar un usuario).

```
async updateUser(data) {
  try {
    return await this._axios.put(`${this._path}/update`, data);
  } catch (e) {
    return e;
  }
}

async getAllUsers() {
  try {
    return await this._axios.fetch(`${this._path}/getAllUsers`);
  } catch (e) {
    return e;
  }
}
```

Ilustración 16: Uso de Axios en el Modelo. Cliente

Axios se encarga de manejar la comunicación con la API REST de forma asíncrona, lo que significa que la interfaz de usuario no se bloqueará mientras espera la respuesta del servidor. Una vez que la respuesta se obtiene, Axios entrega los datos al controlador, y este a su vez actualiza la store de redux con los nuevos datos realizando las acciones necesarias.

```
export const UserReducer = (state = initReducer, { type, payload }) => {
  switch (type) {
    case genericType(SET_USER, ENUM_STATUS.PUSH_NORMAL):
    case genericType(UPDATE_USER, ENUM_STATUS.SUCCESS):
      return {
        ...state,
        loading: false,
        currentType: type,
        messageError: "",
        user: payload || state.user,
      };
    case genericType(UPDATE_USER, ENUM_STATUS.FETCHING):
      return {
        ...state,
        loading: true,
        currentType: type,
        messageError: "",
      };

    case genericType(UPDATE_USER, ENUM_STATUS.FAILURE):
      return {
        ...state,
        loading: false,
        currentType: type,
        messageError: payload,
      };

    default:
      return Object.assign({}, state);
  }
};
```

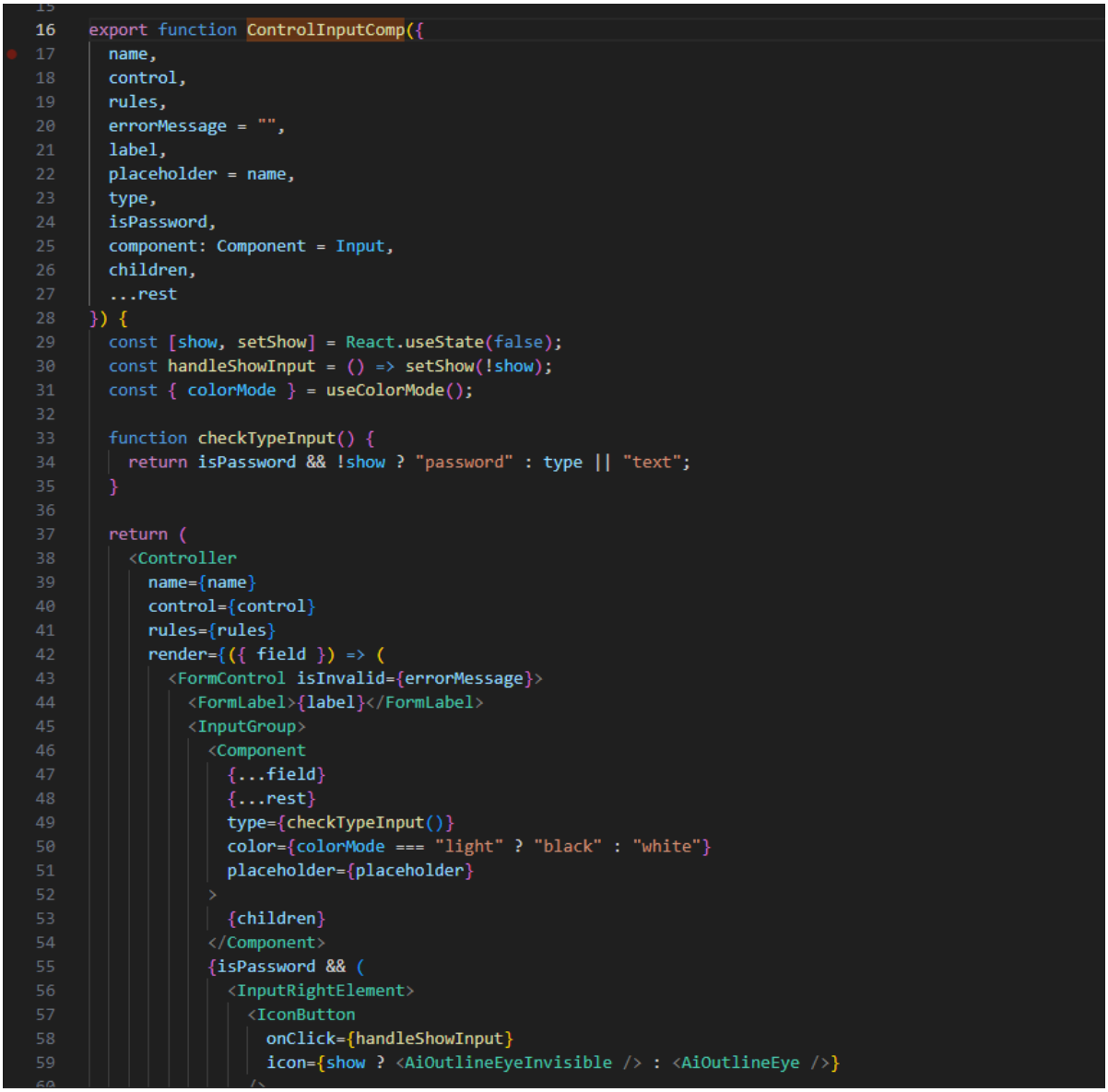
Ilustración 17: Reducer de Usuarios

Una vez que el modelo ha sido actualizado, el controlador actualiza la vista para reflejar los cambios, lo que puede implicar mostrar nuevos datos o actualizar la interfaz de usuario de acuerdo con la acción realizada.

3.3.2 – Vista

La interfaz de usuario de nuestra aplicación web se crea utilizando **JSX** (JavaScript XML). JSX es una extensión de JavaScript que permite escribir código HTML Enhanced similar en apariencia dentro de componentes de React. Las vistas son una parte fundamental de la arquitectura de React, ya que representan la forma en que se muestra la información y cómo los usuarios interactúan con la aplicación.

Un componente en React es una pieza de código independiente y reutilizable que encapsula la lógica y la presentación de una parte específica de la interfaz de usuario. Los componentes facilitan la organización del código y permiten una fácil reutilización en diferentes partes de la aplicación.



```
export function ControlInputComp({
  name,
  control,
  rules,
  errorMessage = "",
  label,
  placeholder = name,
  type,
  isPassword,
  component: Component = Input,
  children,
  ...rest
}) {
  const [show, setShow] = React.useState(false);
  const handleShowInput = () => setShow(!show);
  const { colorMode } = useColorMode();

  function checkTypeInput() {
    return isPassword && !show ? "password" : type || "text";
  }

  return (
    <Controller
      name={name}
      control={control}
      rules={rules}
      render={({ field }) => (
        <FormControl isInvalid={errorMessage}>
          <FormLabel>{label}</FormLabel>
          <InputGroup>
            <Component
              {...field}
              {...rest}
              type={checkTypeInput()}
              color={colorMode === "light" ? "black" : "white"}
              placeholder={placeholder}
            >
              {children}
            </Component>
            {isPassword && (
              <InputRightElement>
                <IconButton
                  onClick={handleShowInput}
                  icon={show ? <AiOutlineEyeInvisible /> : <AiOutlineEye />}
```

Ilustración 18: Componente ControlInputComp

El estado de los componentes en React se puede almacenar y gestionar a través de una store utilizando una biblioteca de gestión, en nuestro caso hemos hecho uso de Redux. Redux es una de las opciones más populares para manejar el estado global en aplicaciones de React, aunque existen otras alternativas como MobX o Context API.

La idea principal detrás de una store en Redux es mantener todo el estado de la aplicación en un solo objeto (la store) que se encuentra fuera de los componentes. Los componentes pueden acceder a este estado global cuando lo necesiten y actualizarlo mediante acciones.

Actualizar el estado global: Cuando el componente dispara una acción, Redux utiliza el reducer definido en el store para actualizar el estado global de acuerdo con la acción especificada. Después de cada actualización del estado, Redux notifica a todos los componentes conectados que el estado ha cambiado y que deben renderizarse nuevamente con los nuevos datos.

```
1   import { ENUM_STATUS, genericType, SET_USER, UPDATE_USER } from "../actions";
2
3   const initReducer = {
4     user: null,
5     loading: false,
6     currentType: "",
7     messageError: "",
8   };
9
10  export const UserReducer = (state = initReducer, { type, payload }) => {
11    switch (type) {
12      case genericType(SET_USER, ENUM_STATUS.PUSH_NORMAL):
13      case genericType(UPDATE_USER, ENUM_STATUS.SUCCESS):
14        return {
15          ...state,
16          loading: false,
17          currentType: type,
18          messageError: "",
19          user: payload || state.user,
20        };
21      case genericType(UPDATE_USER, ENUM_STATUS.FETCHING):
22        return {
23          ...state,
24          loading: true,
25          currentType: type,
26          messageError: "",
27        };
28
29      case genericType(UPDATE_USER, ENUM_STATUS.FAILURE):
30        return {
31          ...state,
32          loading: false,
33          currentType: type,
34          messageError: payload,
35        };
36
```

Ilustración 19: Reducer para Usuarios

3.3.3 – Controlador

El controlador es la parte encargada de recoger la información de los eventos ocurridos en los componentes y enviarlos al modelo o a la lógica de la aplicación para que se realicen las operaciones pertinentes. El controlador actúa como intermediario entre los eventos que ocurren en la interfaz de usuario (vista) y la lógica subyacente o el estado de la aplicación (modelo).

Un claro ejemplo de controlador es el siguiente:

```
const location = useLocation();
const history = useHistory();
const isAuthenticated = useSelector(authenticatedSelector);

const loading = useSelector(authLoadingSelector);

const {
  control,
  handleSubmit,
  formState: { errors },
} = useForm({
  defaultValues: {
    email: "",
    password: "",
  },
});

const onSubmit = (data) =>
  dispatch(genericAction(LOGIN, ENUM_STATUS.FETCHING, { data }));

useEffect(() => {
  if (isAuthenticated) {
    console.log("isAuthenticated", location);
    const from = { ...location, pathname: PAGE_KEYS.HomePage };

    history.push(from);
  }
}, [isAuthenticated]);

return (
  <AuthLayout
    footer={
      <Stack spacing={6}>
        <Button
          isLoading={loading}
          onClick={handleSubmit(onSubmit)}
          colorScheme={"blue"}
```

Ilustración 20: Evento en Capa 1

La función onSubmit es el controlador que se ejecuta cuando ocurre el evento onSubmit. Al hacer clic en el botón "Iniciar sesión" o presionar "Enter" en un campo de entrada, el evento se activa y se llama a la función handleSubmit.

Dentro de onSubmit, recopilamos los datos ingresados por el usuario en un objeto llamado data, que contiene el nombre de usuario y la contraseña. A continuación, utilizamos la librería Axios para realizar una solicitud HTTP POST al servidor. Los datos del formulario se envían como parte del cuerpo de la solicitud para realizar el inicio de sesión.

Axios devuelve una promesa y manejamos la respuesta del servidor en caso de éxito o los errores que puedan ocurrir durante la solicitud.

3.7 – Capa 2: Servicio Rest

Un servicio REST es un conjunto de rutas y controladores que permiten a los clientes realizar operaciones en recursos utilizando métodos HTTP como GET, POST, PUT y DELETE. Cada recurso en el servicio REST tiene una URL única y representa una entidad o conjunto de datos específico.

Anteriormente analizamos de manera general cómo se lleva a cabo la comunicación entre el cliente y el servidor. Observamos que el cliente utiliza la librería Axios para enviar una solicitud HTTP a un endpoint ubicado en la parte del servidor.

Ahora, profundizaremos en cómo se recibe y procesa la información en el servidor REST.

Cuando el cliente envía la solicitud HTTP mediante Axios, esta solicitud llega al servidor, que se encarga de analizar la solicitud y direccionarla hacia la ruta correspondiente en nuestro servicio REST.

```
302   initializeRoutes() {
303     this._router.post(
304       `${this._path}/register`,
305       this.validateBeforeCreateAccount,
306       this.registerAccount
307     );
308     this._router.post(
309       `${this._path}/login`,
310       this.validateBeforeLogin,
311       this.login
312     );
```

Ilustración 21: End Point en servicio Rest

Las rutas en Express están definidas de manera que cada una de ellas se asocia a una URL específica y a un método HTTP particular, como GET, POST, PUT o DELETE.

Por ejemplo, podemos tener una ruta /login que se asocia al método POST para realizar la operación de login.

Cuando la solicitud llega a la ruta /login con el método POST, Express ejecuta la función controladora asociada a esta ruta.

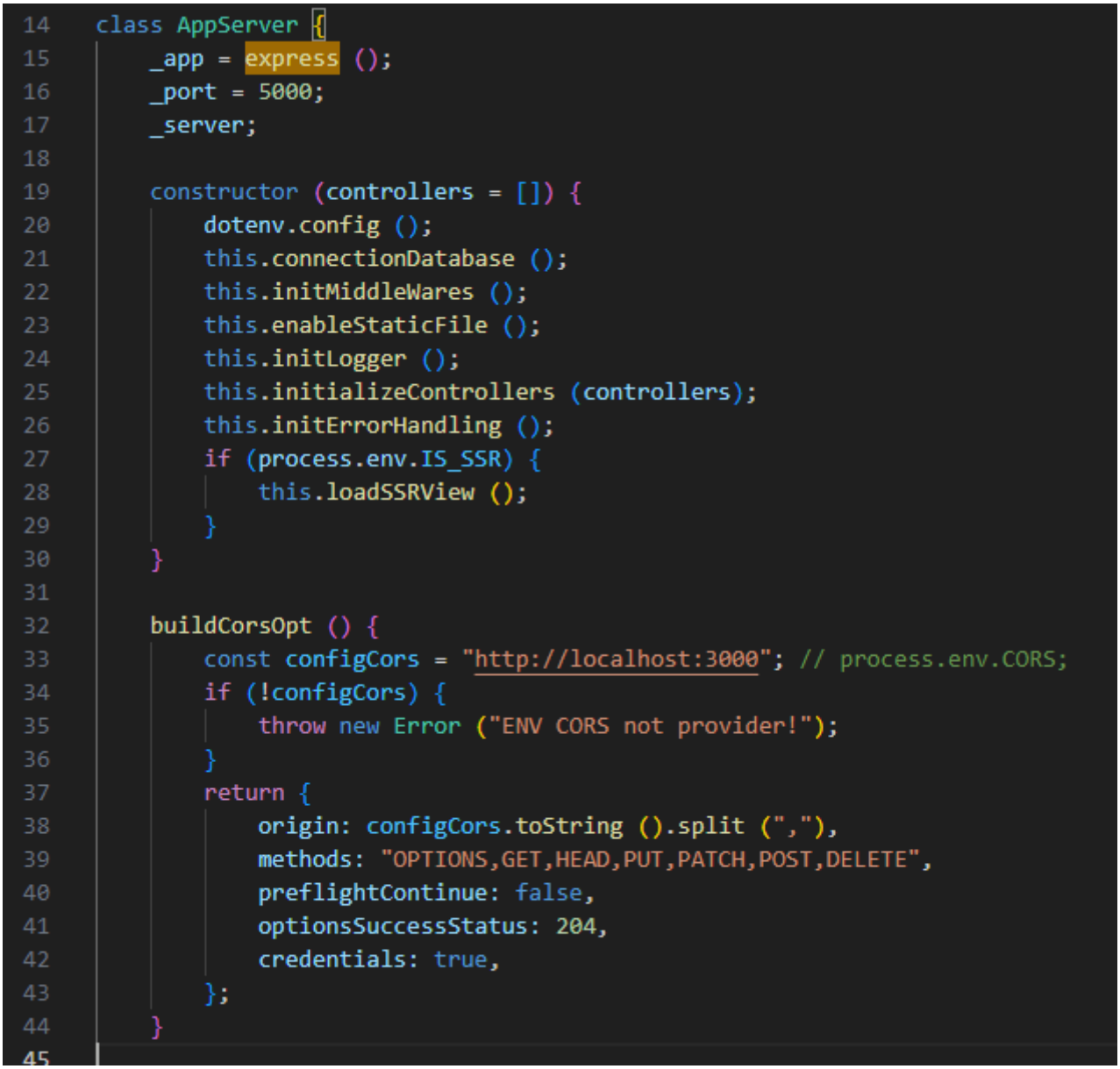
```
async login(req, res, next) {
  try {
    const { email, _id, isAdmin } = req.user;

    const payload = {
      id: _id,
      email,
      isAdmin
    };
    console.log(req);
    const token = await AuthService.generateToken(payload);
    const refreshToken = await AuthService.generateRefreshToken(payload);
    return res.json({
      status: 200,
      message: "success",
      data: {
        access_token: token,
        refresh_token: refreshToken,
        user: payload,
      },
    });
  } catch (error) {
    next(new ServerException(error.message));
  }
}
```

Ilustración 22: Evento asociado a un End Point

El controlador tiene la responsabilidad de manejar la solicitud y luego enviar la respuesta al cliente en formato JSON.

3.8 – Capa 3: Base de datos MongoDB



```
class AppServer {
    _app = express ();
    _port = 5000;
    _server;

    constructor (controllers = []) {
        dotenv.config ();
        this.connectionDatabase ();
        this.initMiddleWares ();
        this.enableStaticFile ();
        this.initLogger ();
        this.initializeControllers (controllers);
        this.initErrorHandling ();
        if (process.env.IS_SSR) {
            this.loadSSRView ();
        }
    }

    buildCorsOpt () {
        const configCors = "http://localhost:3000"; // process.env.CORS;
        if (!configCors) {
            throw new Error ("ENV CORS not provider!");
        }
        return {
            origin: configCors.toString ().split (","),
            methods: "OPTIONS,GET,HEAD,PUT,PATCH,POST,DELETE",
            preflightContinue: false,
            optionsSuccessStatus: 204,
            credentials: true,
        };
    }

```

Ilustración 23: Instanciación en AppServer

Como comentado anteriormente hemos hecho uso de Node.js y MongoDB para la creación del modelo en la 3ra capa.

Importamos las dependencias necesarias en el archivo principal y creamos una instancia de express además de configurarla como middleware para analizar las solicitudes entrantes en formato JSON utilizando express.json(). Esto nos permite recibir y procesar datos en formato JSON.

Luego, nos conectamos a la base de datos de MongoDB utilizando el cliente de MongoDB y la URI de conexión

```
connectionDatabase () {
    const mongoUrl = process.env.MONGO_URL;
    mongoose.connect (mongoUrl, {
        autoCreate: true,
        autoIndex: true,
    });
}
```

Ilustración 24: Conexión a la base de datos

La variable de entorno **MONGO_URL** contendrá la URI donde apuntará nuestro servidor de mongo, para agilizar el proceso hemos hecho uso de AtlasDB, un servicio de base de datos en la nube ofrecido por MongoDB, que proporciona una forma sencilla y conveniente de alojar, administrar y escalar bases de datos MongoDB sin la necesidad de configurar, lo que nos proporciona:

- **Escalabilidad y rendimiento**: AtlasDB está diseñado para escalar fácilmente tus bases de datos según tus necesidades. Puedes aumentar o disminuir la capacidad de almacenamiento y el rendimiento de tu base de datos de manera flexible, según las demandas de tu aplicación.

- **Alta disponibilidad**: AtlasDB proporciona redundancia y replicación automática para garantizar que tus bases de datos estén siempre disponibles. Los datos se replican en múltiples ubicaciones geográficas para mayor confiabilidad.

- **Seguridad**: AtlasDB ofrece características de seguridad integrales para proteger tus datos. Incluye autenticación, control de acceso basado en roles, encriptación en tránsito y en reposo, y auditoría de registros para cumplir con los estándares de seguridad y privacidad.

- Y sobre todo **administración simplificada**: AtlasDB se encarga de muchas tareas administrativas, como el monitoreo, la optimización del rendimiento, la aplicación de parches y actualizaciones de MongoDB. Esto nos ha permitido centrarnos en el desarrollo de la aplicación en lugar de preocuparnos por la gestión de la infraestructura de la base de datos.

En definitiva, en lugar de instalar y configurar MongoDB en un servidor propio o máquina local, AtlasDB te permite utilizar MongoDB en la nube, esta sería la visualización para la conexión de la base de datos.

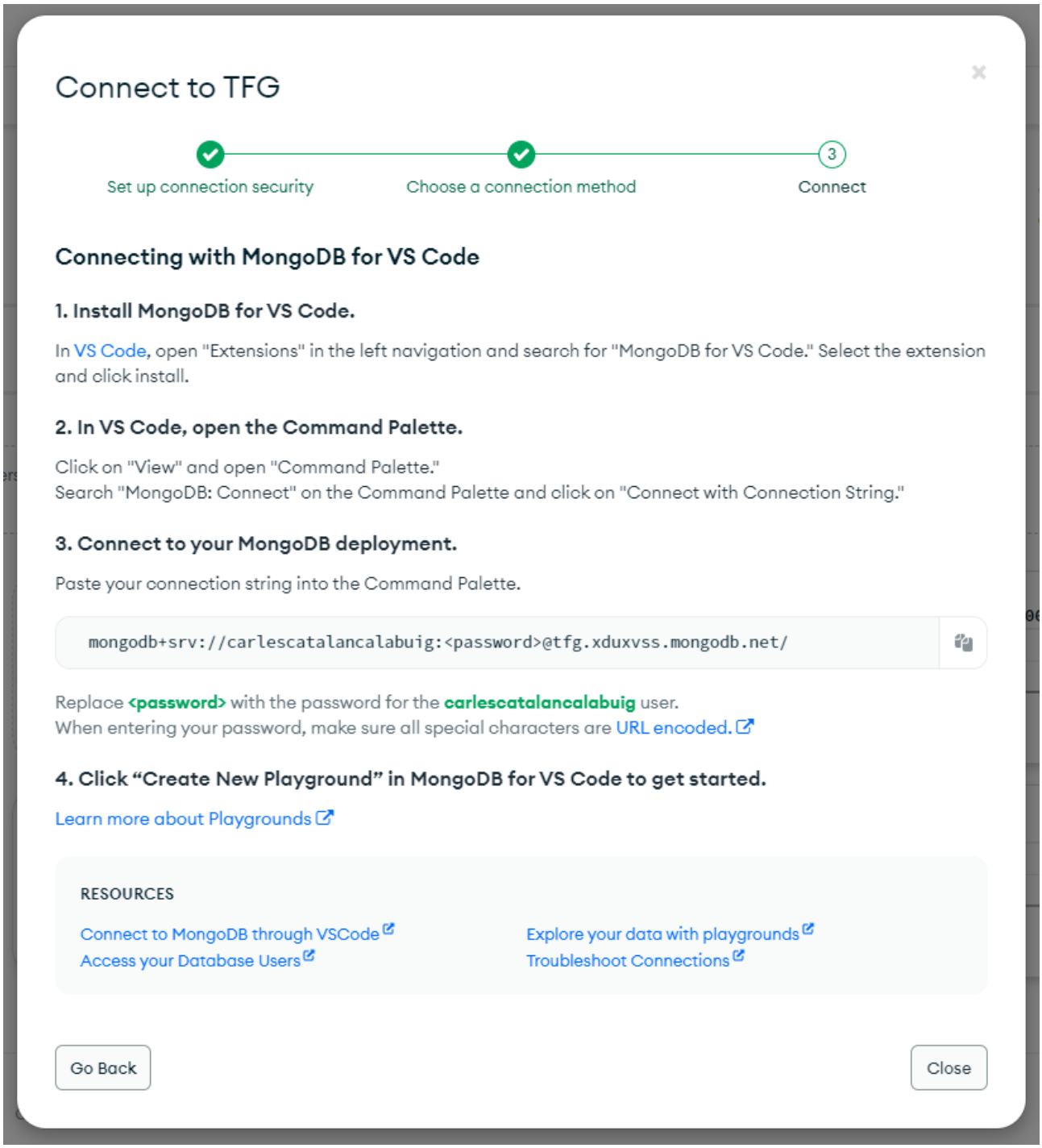


Ilustración 25: Conexión mediante AtlasDB

Además hemos utilizado una capa de abstracción de la base de datos para representar y trabajar con el esquema de la base de datos. Esta capa de abstracción se conoce como un Object-Document Mapper (ODM) o Object-Relational Mapper (ORM), dependiendo de si estás utilizando una base de datos NoSQL como MongoDB o una base de datos relacional.

Esto nos permite definir modelos y esquemas en la aplicación de Node.js que representan las entidades y relaciones de la base de datos. Estos modelos proporcionan una interfaz para interactuar con la base de datos de una manera orientada a objetos.

En el caso de MongoDB y AtlasDB, podemos utilizar el ODM llamado Mongoose anteriomente mencionado. Mongoose es una biblioteca de JavaScript que nos proporciona una forma sencilla de modelar los datos y realizar operaciones CRUD en MongoDB.

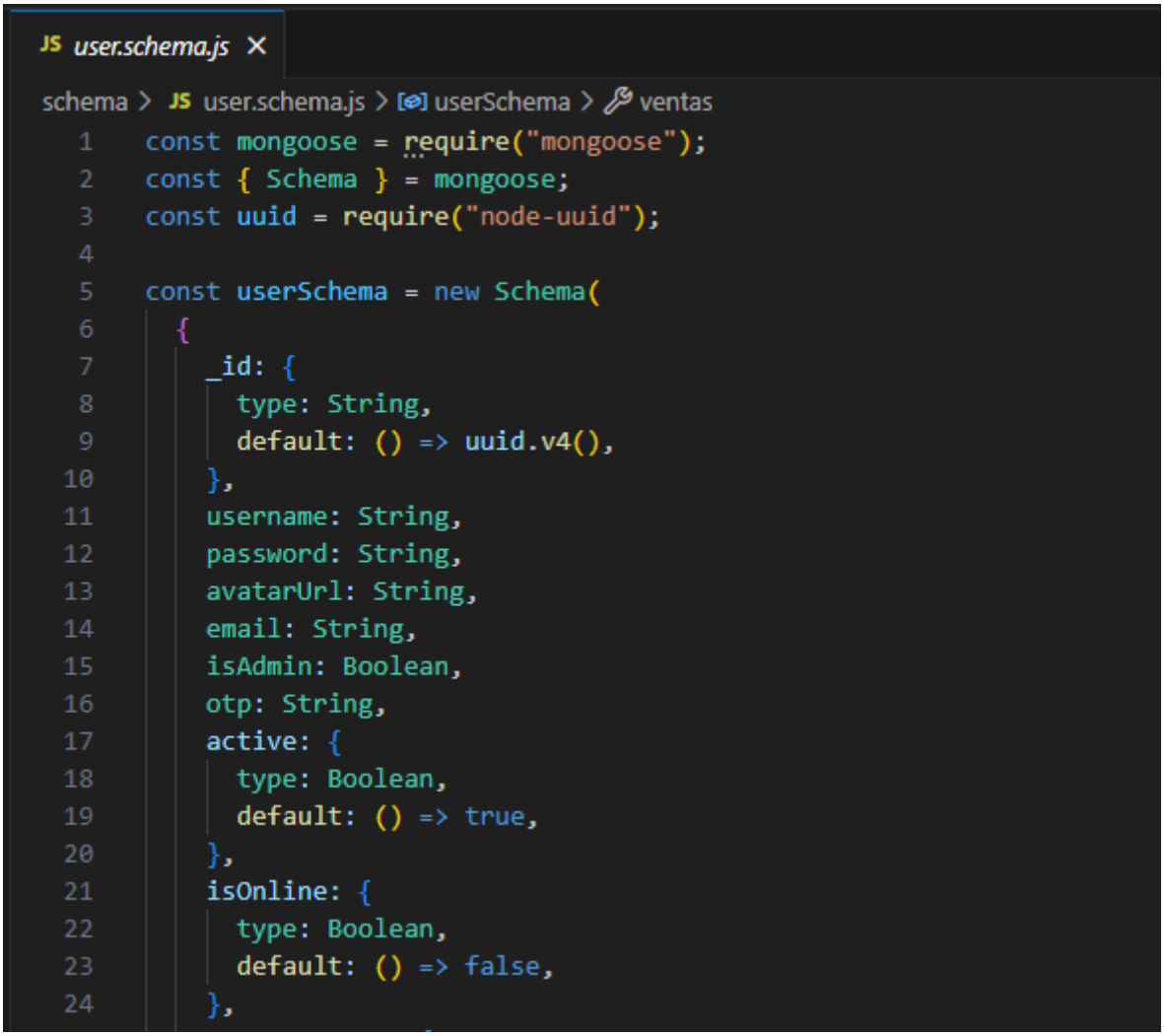


Ilustración 26: Modelo en Node.js .1

```
25      numeroVentas: {
26        type: Number,
27        default: () => 0,
28      },
29      gananciaVentas: {
30        type: Number,
31        default: () => 0,
32      },
33      clientesCaptados: {
34        type: Number,
35        default: () => 0,
36      },
37      cantidadProductosVendidos: {
38        type: Number,
39        default: () => 0,
40      },
41      ventas: {
42        type: Array,
43        default: () => [0,0,0,0,0,0,0,0,0,0,0,0],
44      },
45      objetivoVentas: {
46        type: Array,
47        default: () => [0,0,0,0,0,0,0,0,0,0,0,0],
48      },
49      hitos: {
50        type: Array,
51        default: () => [],
52      },
53      coordenadas: {
54        type: Array,
55        default: () => [],
56      },
```

Ilustración 27: Modelo en Node.js .2

Esta será una parte de la representación de las entidades y relaciones de la base de datos del modelo utilizado para la colección de MongoDB.

Una vez realizado el esquema es posible exportarlo.

```
const UserRepository = require("./user.schema");

module.exports = {
  UserRepository,
};
```

Ilustración 28: Exportación de esquema

Y a continuación nos será posible utilizar las acciones del modelo para interactuar con la base de datos utilizando las operaciones provistas por la biblioteca de MongoDB. Cada acción se utiliza para realizar una operación específica en la base de datos, como obtener datos, insertar nuevos registros, actualizar registros existentes o eliminar registros. Podemos ver como el esquema UserRepository es llamado en las siguientes funciones:

```
  async updateUser(_id, userUpdate) {
    try {
      return await UserRepository.findByIdAndUpdate(
        { _id },
        {
          ...userUpdate,
        }
      );
    } catch (e) {
      throw new Error(e.message);
    }
  }

  async getAllUsers() {
    try {
      return await UserRepository.find();
    } catch (e) {
      throw new Error(e.message);
    }
  }
}
```

Ilustración 29: Funciones haciendo uso del ORM

Las acciones del modelo mencionadas anteriormente están diseñadas para interactuar con una colección específica en una base de datos MongoDB, en este caso será la colección de Users.

En MongoDB, una colección es similar a una tabla en una base de datos relacional y almacena los documentos JSON. Cada acción del modelo se ejecuta en una colección en particular y afecta los documentos dentro de esa colección.

Estas operaciones se dirigen a la colección asociada y realizan la acción solicitada en los documentos de esa colección.

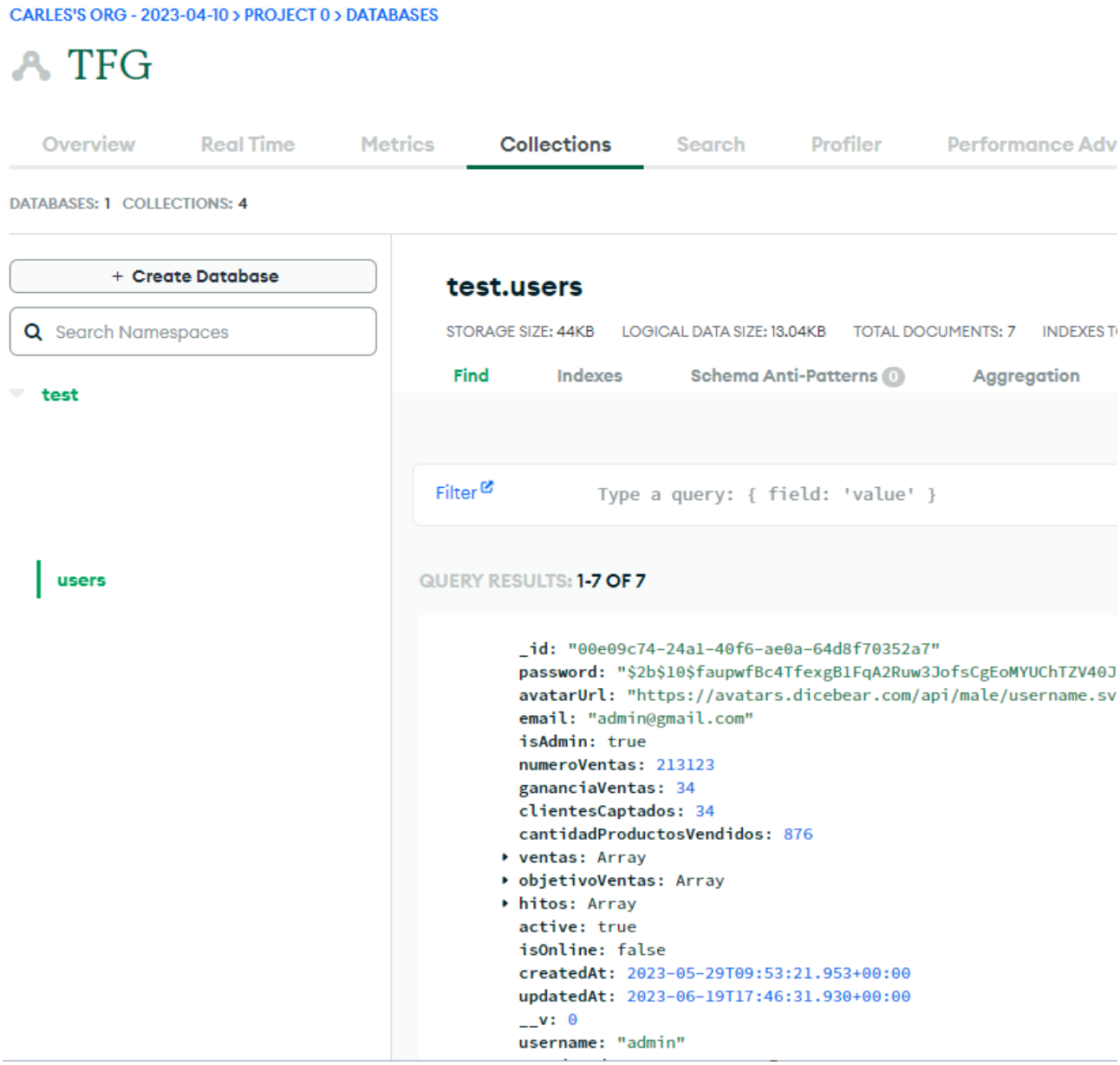


Ilustración 30: Vista general de una colección en MongoDB

3.9 – Control de versiones

El uso de Git y GitHub ha sido una elección acertada para el desarrollo de este proyecto, ya que estas herramientas proporcionan una infraestructura sólida y eficiente para el seguimiento de los cambios en el código.

- **Rama "Desarrollo"**: En esta rama, se realizan las integraciones de las características y mejoras desarrolladas.

- **Rama "Master"**: La rama "master" es la rama principal del proyecto y contiene la versión más estable y probada del código. Se considera como la versión de producción, lista para ser implementada en el entorno en vivo o entregada a los usuarios finales.

Una vez tenemos código funcional pasa a ser unido a la rama master.

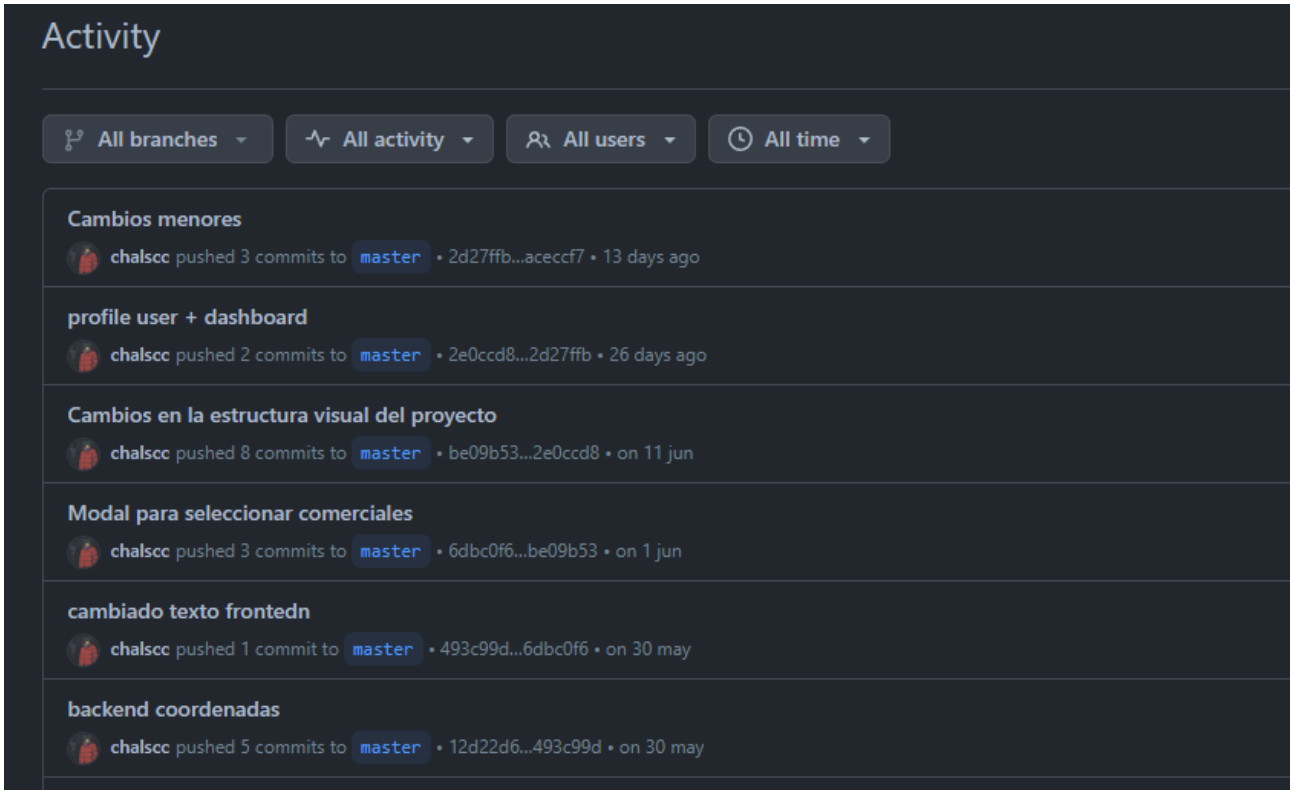


Ilustración 31: Commits realizados en GitHub

4 – Tour por la aplicación

4.1 – Login

Primero, nos encontramos con la pantalla de inicio de sesión (login), donde un usuario puede ingresar sus credenciales para acceder a la aplicación. Esta pantalla es la puerta de entrada a la plataforma y su función principal es autenticar y verificar la identidad del usuario antes de permitirle acceder a áreas protegidas, recordemos que contamos con dos tipos de usuarios diferentes, los usuarios administradores y los usuarios comerciales.

En la pantalla de login, se solicitan dos campos de entrada principales:

- **Correo electrónico**: Aquí el usuario debe proporcionar su identificador único en el sistema, que será su dirección de correo electrónico que previamente haya registrado.

- **Contraseña**: En este campo, el usuario debe ingresar su contraseña correspondiente al correo electrónico proporcionado. La contraseña se utiliza como un mecanismo de seguridad para verificar la identidad del usuario.

Una vez que el usuario ha completado los campos requeridos, habrá un botón "Sign in" para que el usuario realice la solicitud de inicio de sesión.

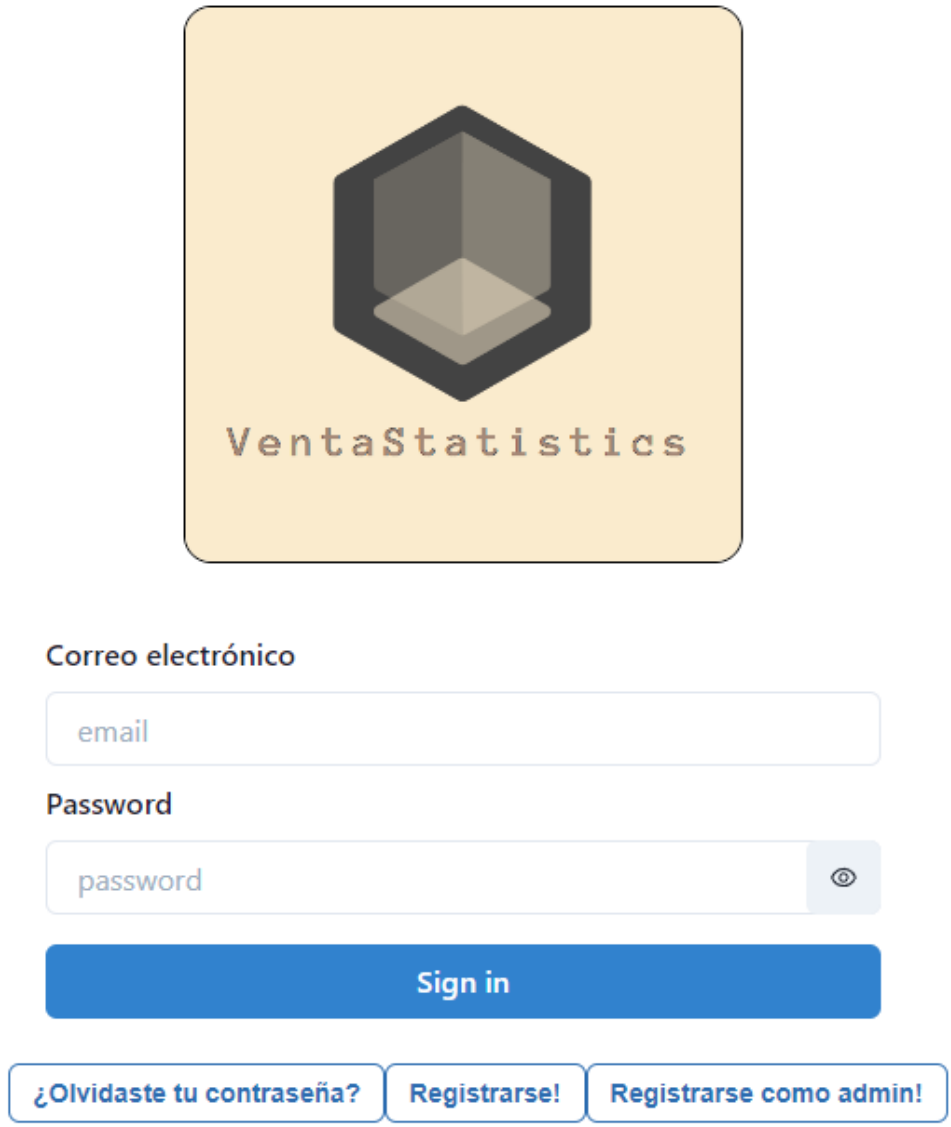


Ilustración 32: Pantalla Login

4.2 – Registro

La siguiente pantalla es la de registro, donde un usuario puede crear una nueva cuenta en la aplicación o en el sistema. Esta pantalla es fundamental para permitir que nuevos usuarios se registren y obtengan acceso a la plataforma.

En la pantalla de registro, se solicitan diversos campos de entrada para recopilar la información necesaria para crear la cuenta del usuario.

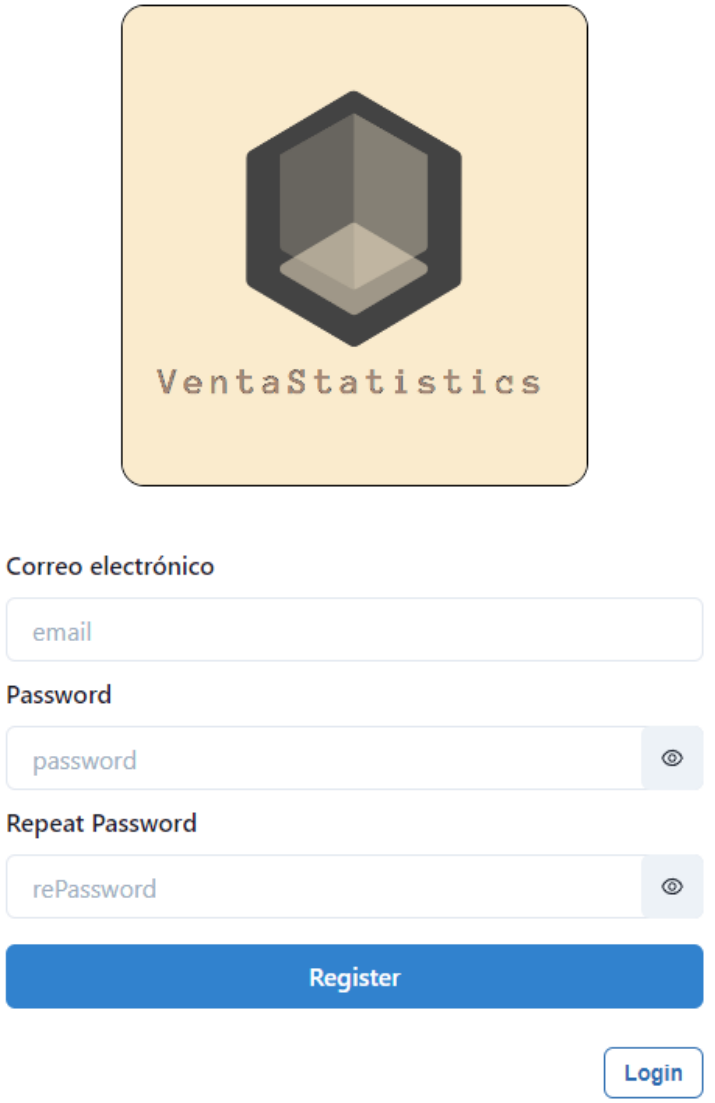


Ilustración 33: Pantalla de registro de usuario

También tendremos el registro como administrador, que como hemos comentado anteriormente pedirá un identificador único. Este identificador único es un código o número de identificación especial para los administradores.

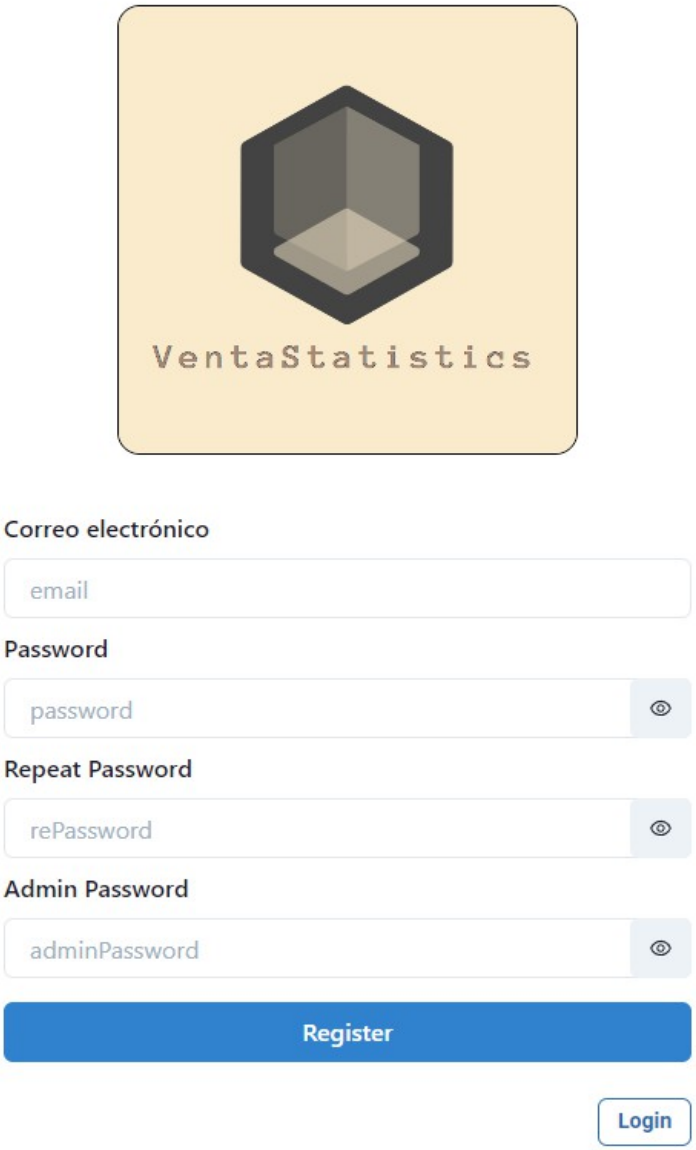


Ilustración 34: Pantalla de registro de administrador

4.3 – Menú de opciones

El menú de opciones en la aplicación consta de varias secciones que ofrecen diferentes funcionalidades para mejorar la experiencia del usuario. A continuación, se describe cada una de estas secciones:

- **Dashboard con gráficas de ventas**: Esta sección proporciona una vista general y visual de las ventas del negocio. Los datos de ventas se representan en gráficos y tablas para mostrar tendencias, estadísticas clave y comparativas de rendimiento.

- **Perfil de usuario**: En esta sección, los usuarios pueden acceder y actualizar su perfil personal. Aquí, pueden cambiar su información de contacto, contraseña y preferencias.

- **Mapa de potenciales puntos de venta**: Esta funcionalidad ofrece un mapa interactivo que muestra los puntos de venta potenciales.

- **Calendario de hitos para comerciales**: En esta sección, los comerciales o representantes de ventas pueden ver un calendario con los hitos o eventos importantes relacionados con su trabajo.

- **Panel de datos para la inserción de ventas de los comerciales**: Esta sección permite a los comerciales ingresar las ventas y resultados de sus ventas.

- **Botón de cierre de sesión**: El cual será encontrado en la parte superior derecha de la interfaz, en la barra superior. Este botón nos permitirá cerrar la sesión protegiendo la información que se encuentra alojada con nuestro usuario.

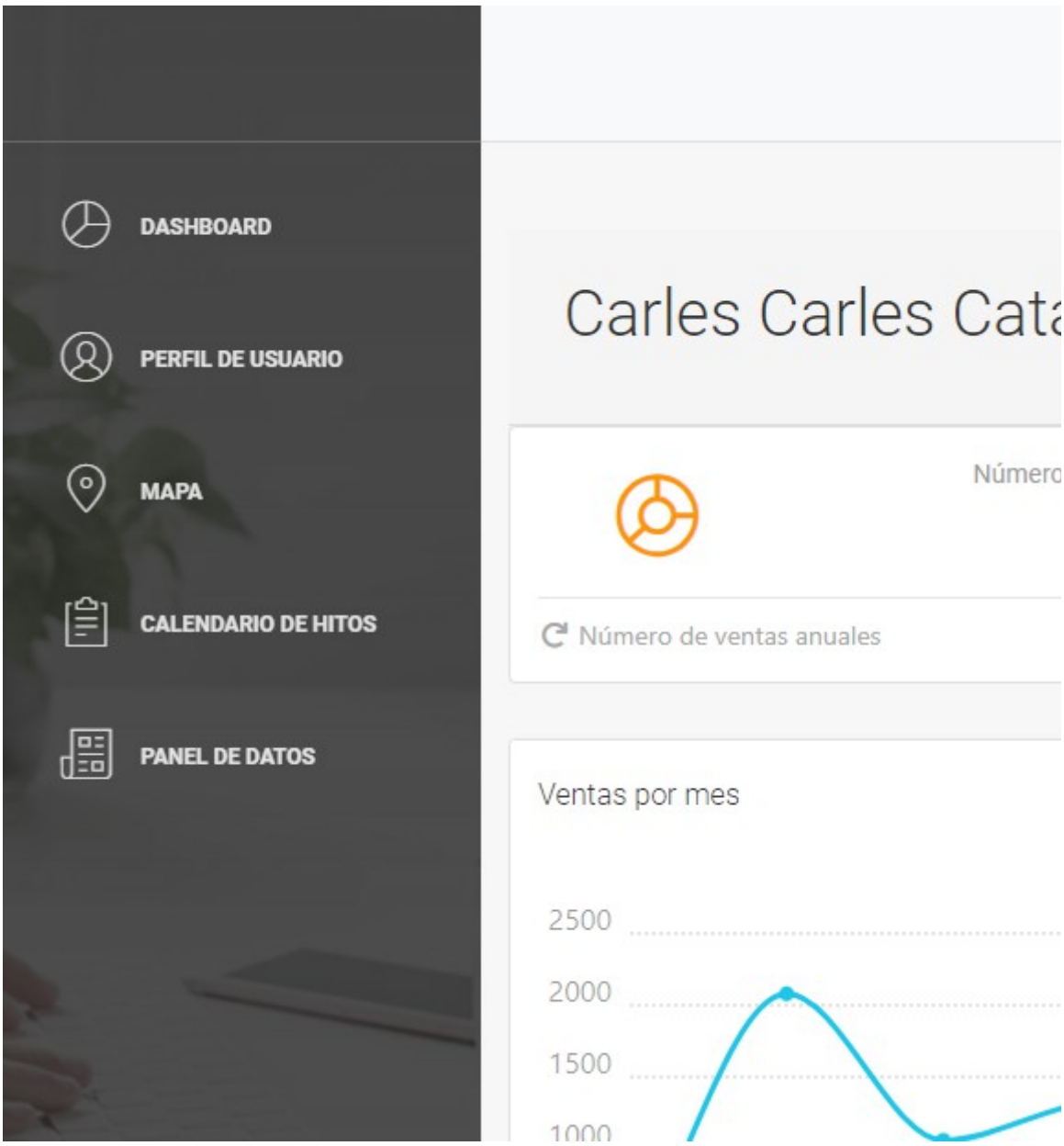


Ilustración 35: Menú de opciones

4.4 – Dashboard

La primera vista que encontramos es la del Dashboard, donde podemos ver las gráficas de los comerciales y otras métricas relevantes para nuestro negocio. Las gráficas proporcionan una visión general del rendimiento de las campañas publicitarias, mostrando datos como numero de ventas, clientes captados o ventas por mes.

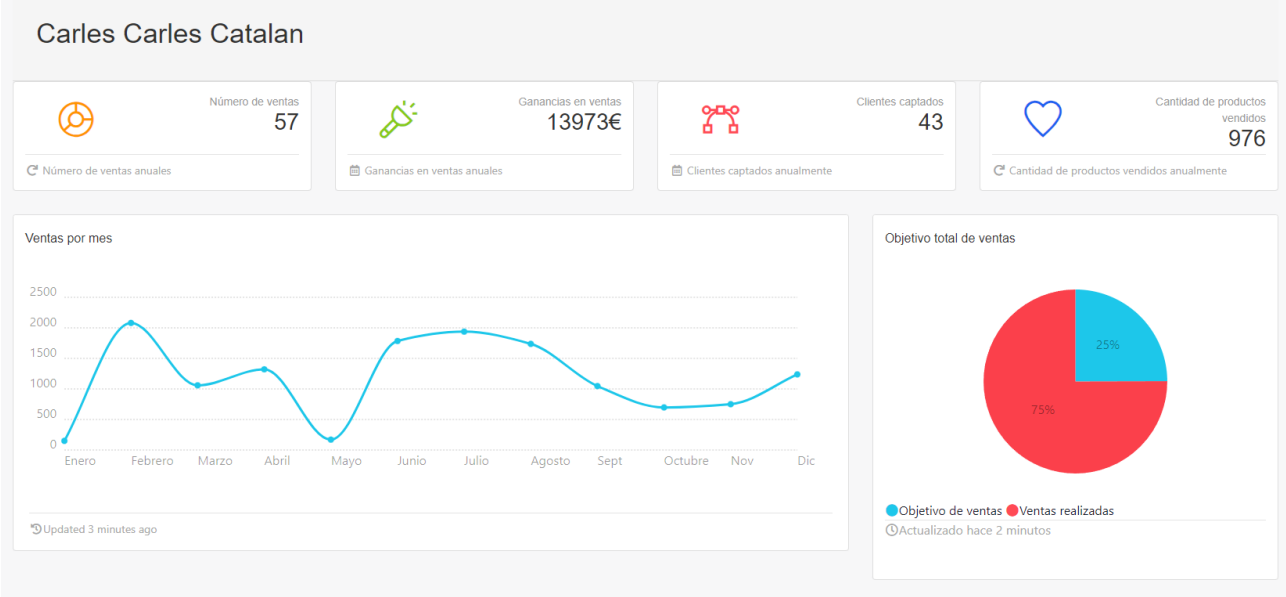


Ilustración 36: Gráficas dashboard 1

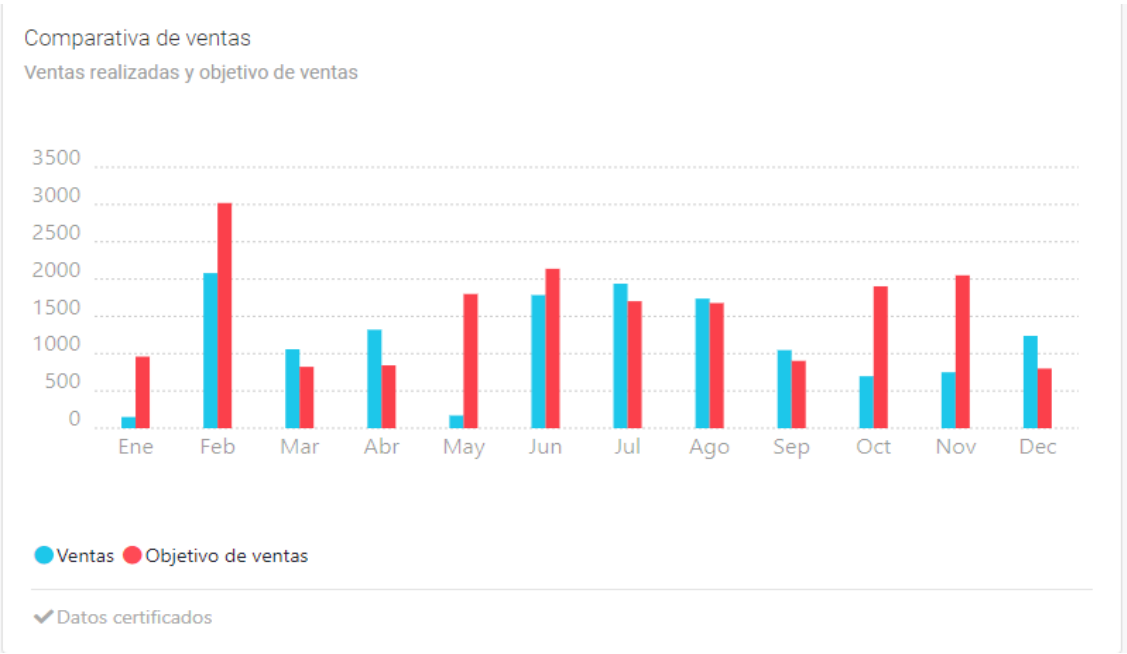


Ilustración 37: Gráficas dashboard 2

Además, también es posible ver los hitos realizados por el comercial en el dashboard.

Estos hitos proporcionan un seguimiento detallado del progreso de cada comercial en su ciclo de ventas. Al visualizar los hitos logrados, es posible evaluar la efectividad y el compromiso de cada vendedor en las diferentes etapas del proceso de ventas.

Es posible actualizar un hito como comercial o como usuario administrador cuando ha sido realizado, seleccionando la casilla alojada a la izquierda del hito.

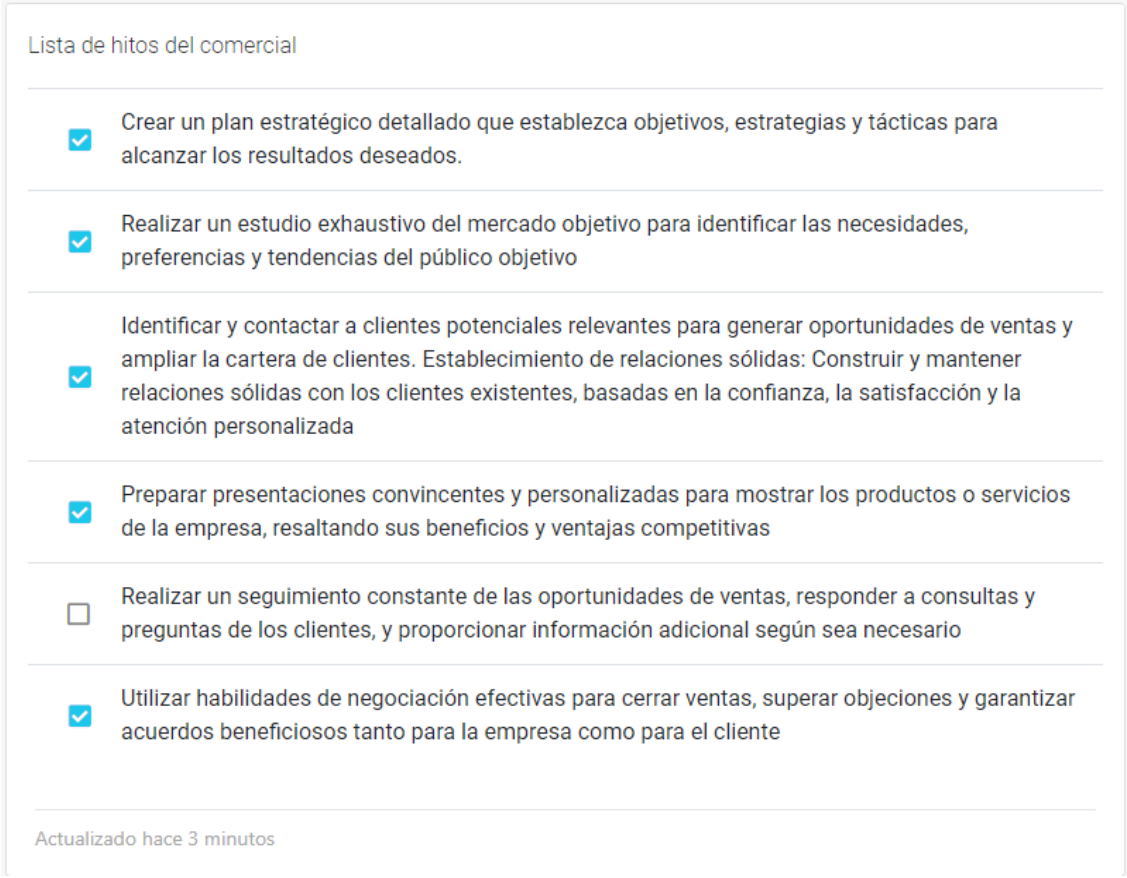


Ilustración 38: Lista de hitos comerciales

Es posible plegar y desplegar la información de los comerciales mediante un 'acordeon', esto nos proporciona una visión global de los comerciales que se encuentran a nuestro cargo.

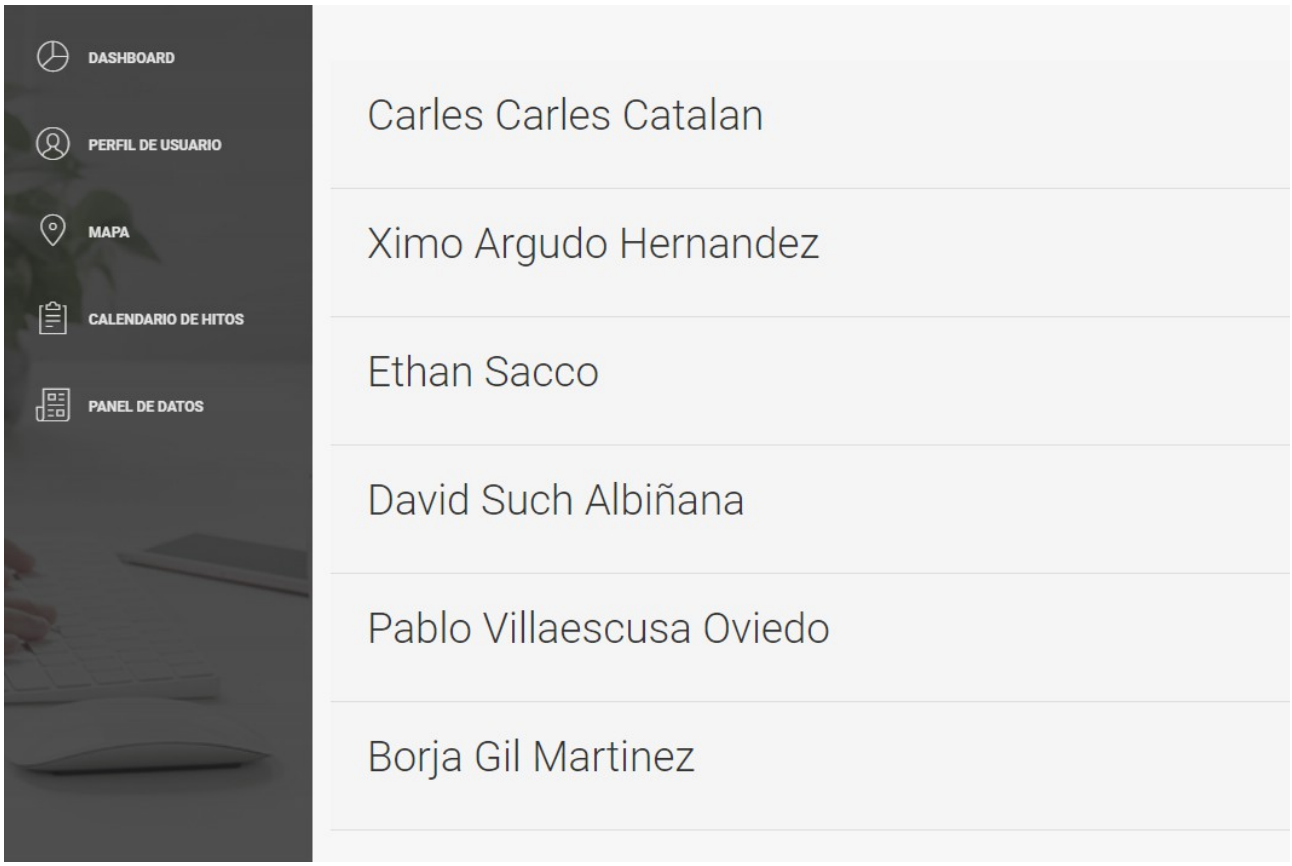


Ilustración 39: Dashboard colapsado

4.5 – Perfil de usuario

A continuación veremos la pantalla de Perfil de usuario, tenemos la imagen de usuario con una pequeña descripción y seguidamente los campos de perfil que permiten a los usuarios personalizar su información y preferencias. Estos campos incluyen datos como nombre, apellidos e información de contacto.

Ilustración 40: Perfil de usuario 1

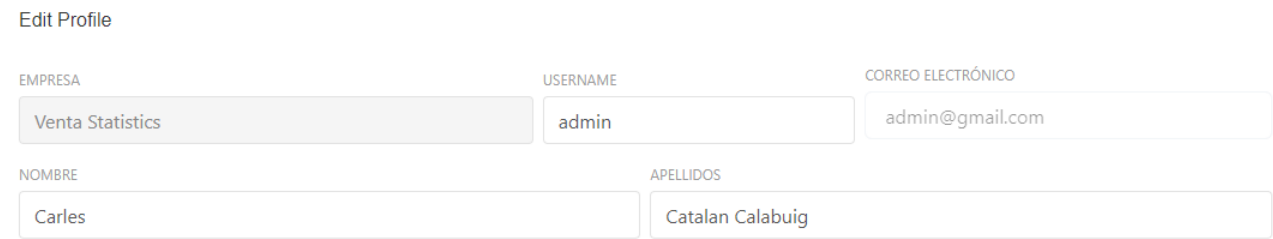


Ilustración 41: Perfil de usuario 2

Ilustración 42: Perfil de usuario 3

4.6 – Selección de comerciales

La vista de “selección de comerciales" es la sección que tiene como objetivo mostrar una lista de comerciales disponibles.

En esta vista, se mostrará una lista que enumera los diferentes comerciales existentes. Cada elemento de la lista mostrará el del comercial nombre.

Se podrá seleccionar uno de ellos haciendo clic en su nombre. Al hacerlo, dependiendo de la pantalla en la que nos encontremos podremos ver información asociada a ese comercial.

Para facilitar la búsqueda, también se dispondrá de un filtro (cuadro de búsqueda), donde será posible ingresar el nombre o parte del nombre del comercial que se esté buscando. Al activar el filtro, la lista se actualizará automáticamente para mostrar solo los comerciales que coincidan con el criterio de búsqueda.

El objetivo principal de esta vista es homogeneizar la selección de comerciales en toda la aplicación y proporcionar a los administradores una forma sencilla y conveniente de encontrar los comerciales disponibles.

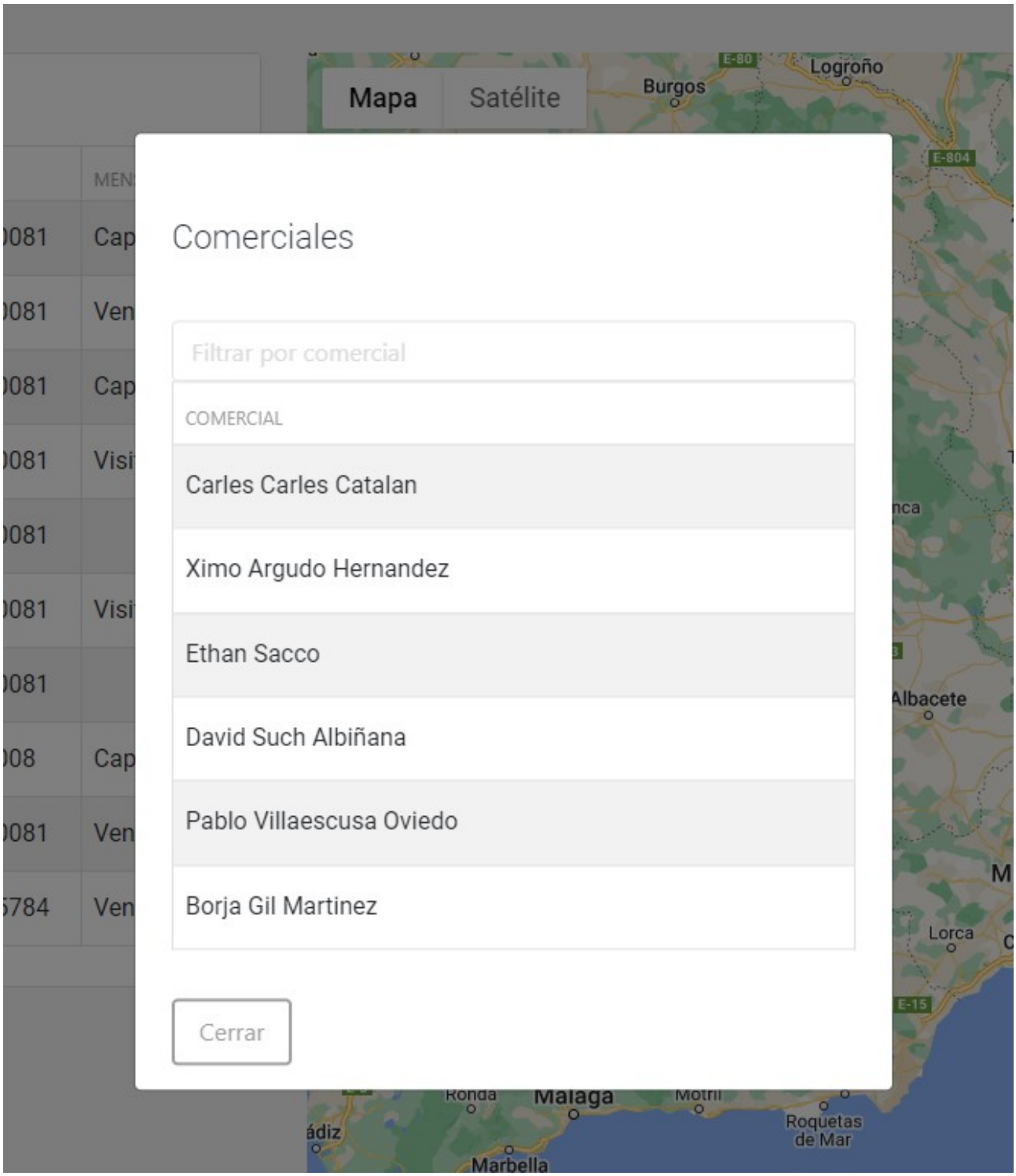


Ilustración 43: Selección en lista de comerciales

4.7 – Mapa interactivo

La vista de "Mapa interactivo" permite a un administrador asignar puntos de ventas potenciales a los comerciales de manera visual y geográfica tan solo dando clic en el mapa interactivo. Al dar clic emergerá una nueva ventana donde podremos asociar una descripción a este punto en el mapa.

La vista del mapa interactivo mostrará un mapa geográfico, basado en Google Maps, se ubicarán distintos markers o marcadores en las ubicaciones estratégicas para el comercio. Cada marcador representará un punto de venta potencial que un comercial puede atender.

El administrador tendrá el control total para agregar los marcadores en el mapa. Al asignar un punto de venta potencial a un comercial específico, simplemente deberá colocar el marcador en la ubicación deseada. De esta manera, se establece una relación entre el comercial y esa ubicación específica, lo que indica que ese comercial es responsable de atender ese punto de venta.

Además, la lista encontrada a la izquierda mostrará la información de todos los marcadores disponibles para ese comercial. Haciendo clic en cualquiera de las entradas hará zoom a la zona del mapa indicada.

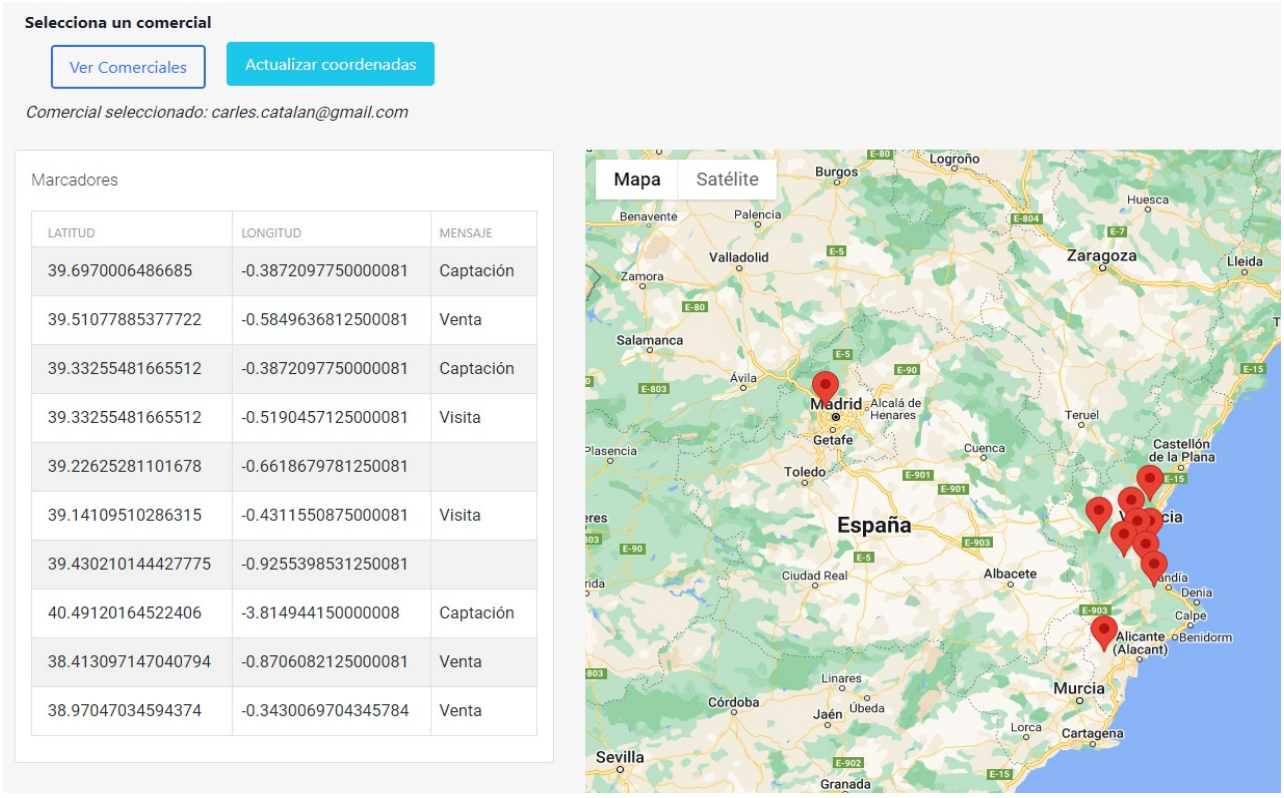


Ilustración 44: Mapa interactivo y listado de puntos en el mapa

4.8 – Calendario de hitos

En la vista 'Calendario de hitos', los usuarios podrán visualizar y hacer un seguimiento de los hitos previamente comentados.

En esta vista, los hitos estarán representados como eventos dentro de un calendario. Cada hito tendrá una fecha asociada para indicar cuándo se espera que ocurra o cuándo se ha alcanzado. Los usuarios podrán ver de manera clara y organizada cuáles son los hitos programados y cuáles ya han sido cumplidos.

Para cada hito en el calendario, se proporcione una descripción que explique su importancia.

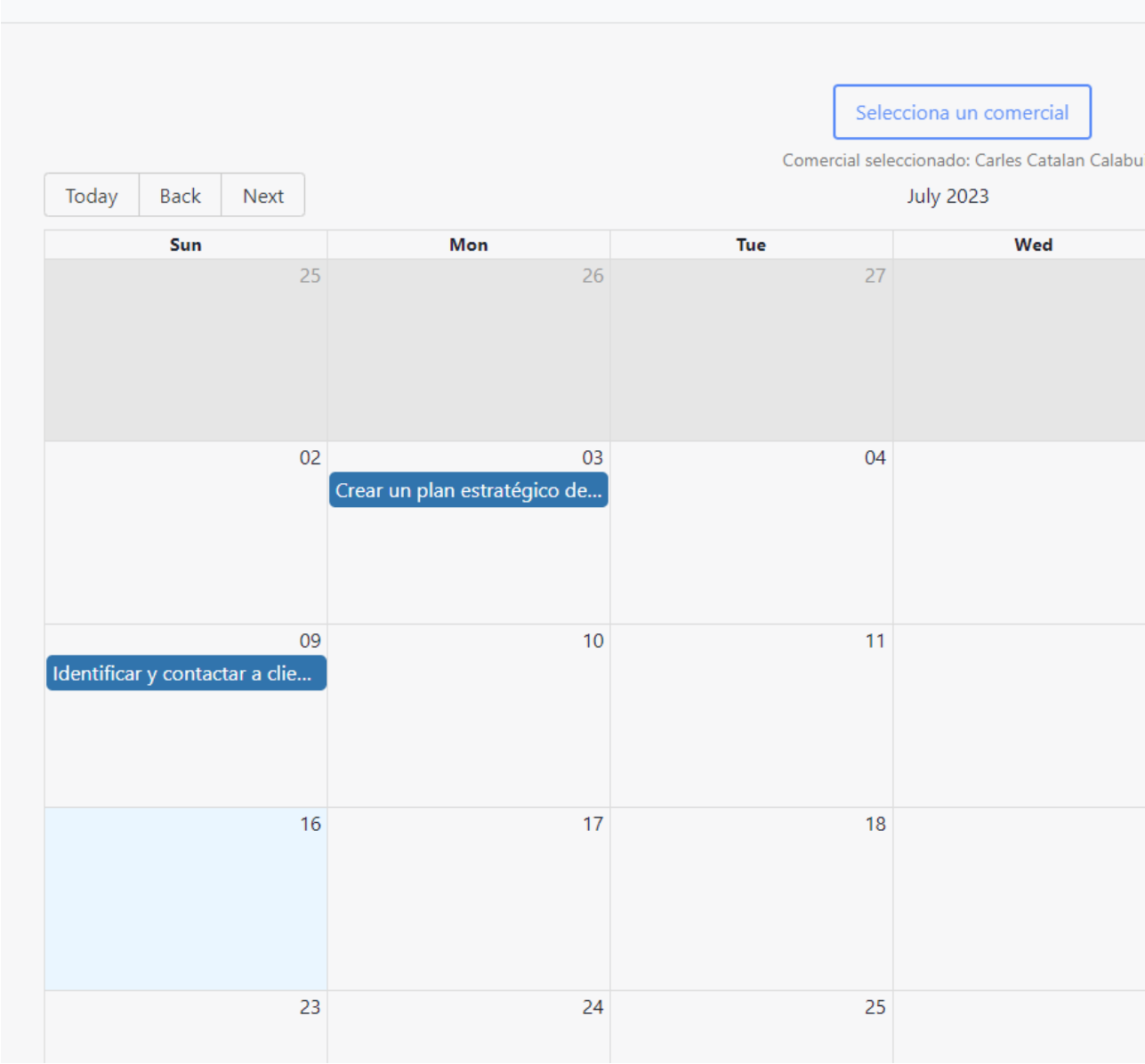


Ilustración 45: Calendario de hitos 1

Encontramos la vista descripción del hito, con el nombre del comercial, la descripción asociada al hito, la fecha que se espera para su cumplimiento y el estado del hito, el cual podía ser actualizado desde la pantalla de Dashboard.

Esta vista es mostrada cuando hacemos clic en alguno de los hitos del calendario.

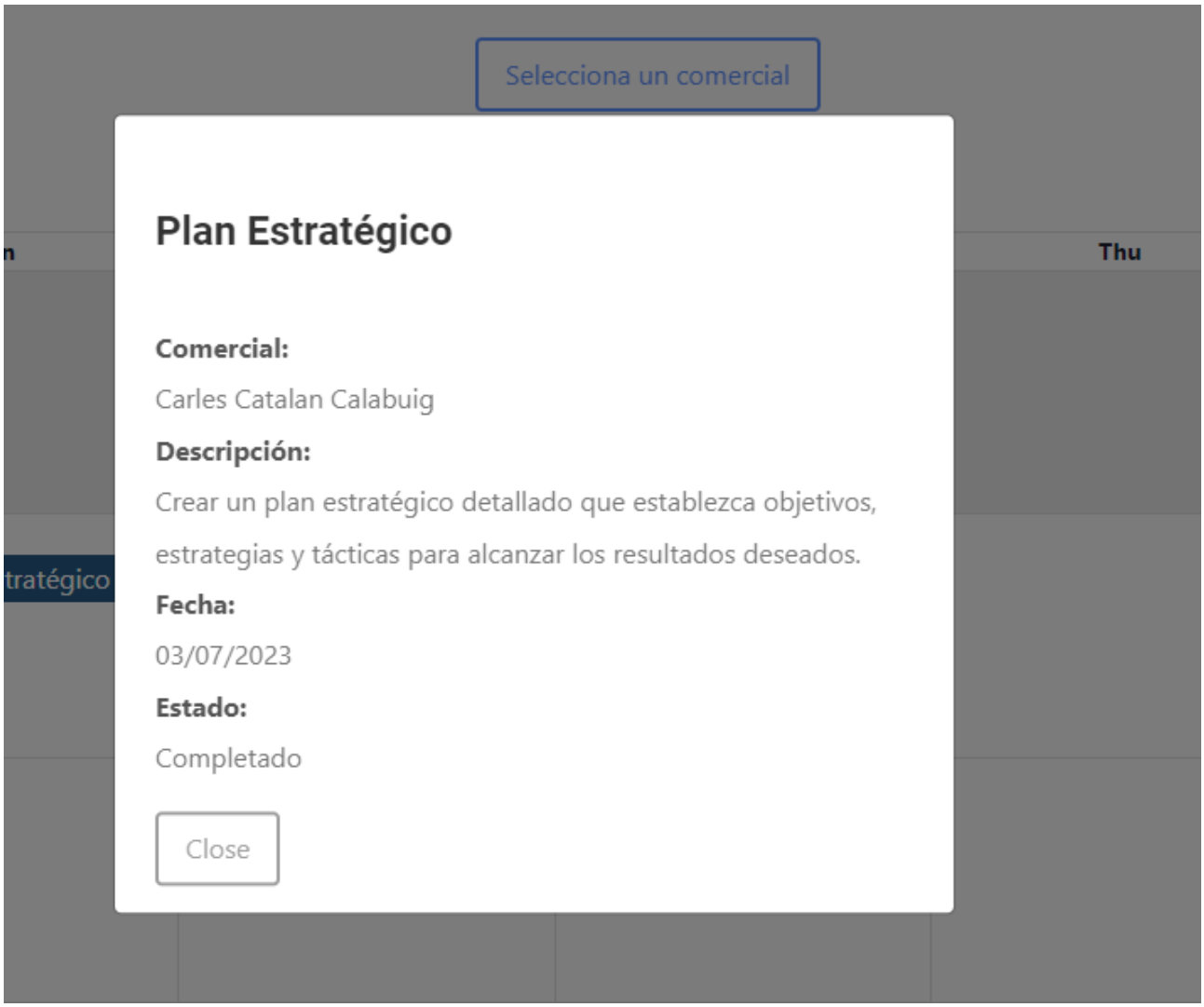


Ilustración 46: Vista informativa de hito

En caso que hagamos clic en alguna celda vacía, aparecerá la siguiente vista con datos ya establecidos como la fecha o el estado.

Esta es la vista de creación de un nuevo hito.

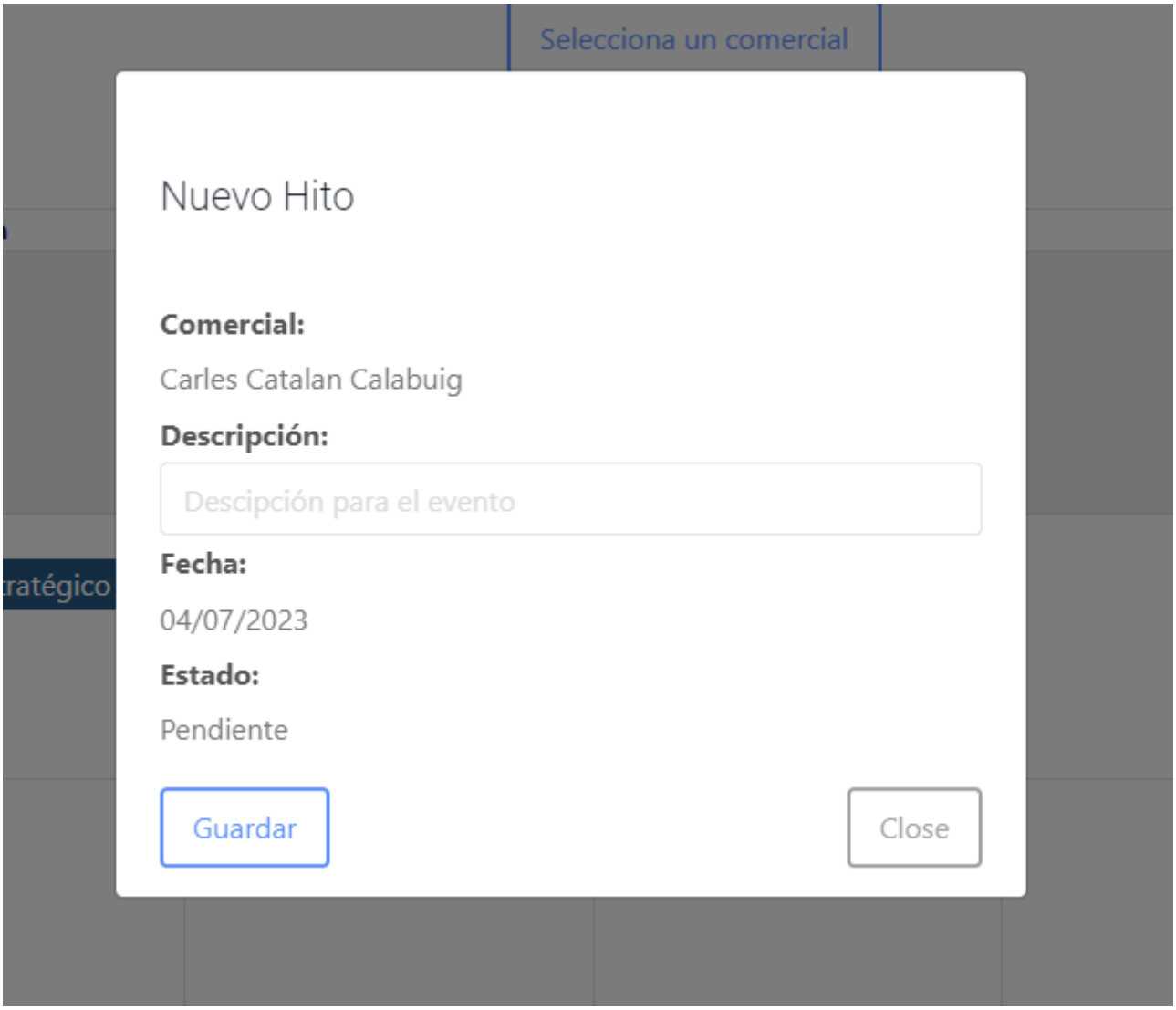


Ilustración 47: Creación de un nuevo hito

Existe la opción de establecer dicha vista como "vista agenda". Esta función permite a los usuarios cambiar la forma en que visualizan los eventos y hitos en el calendario, brindando una alternativa más específica para la gestión del tiempo y la planificación.

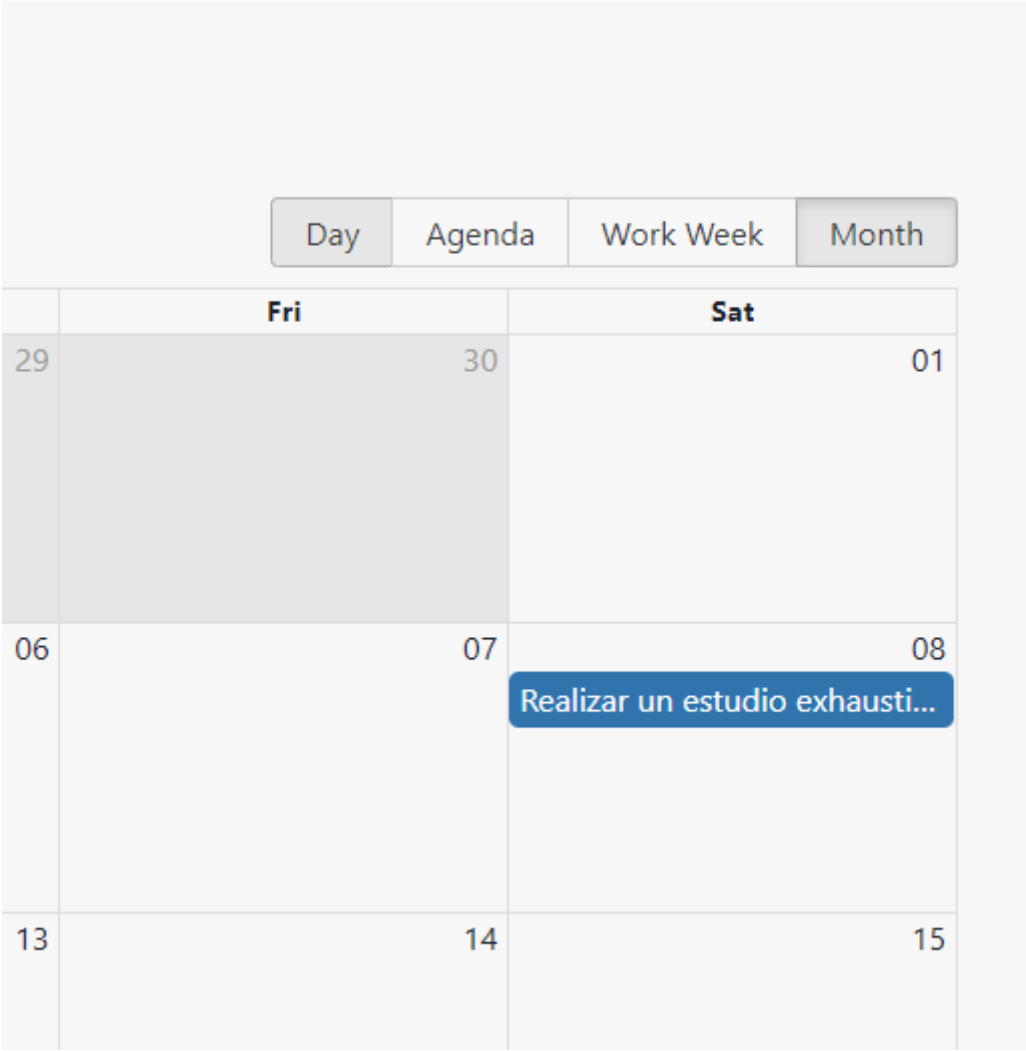


Ilustración 48: Botonera para la selección de vista del calendario

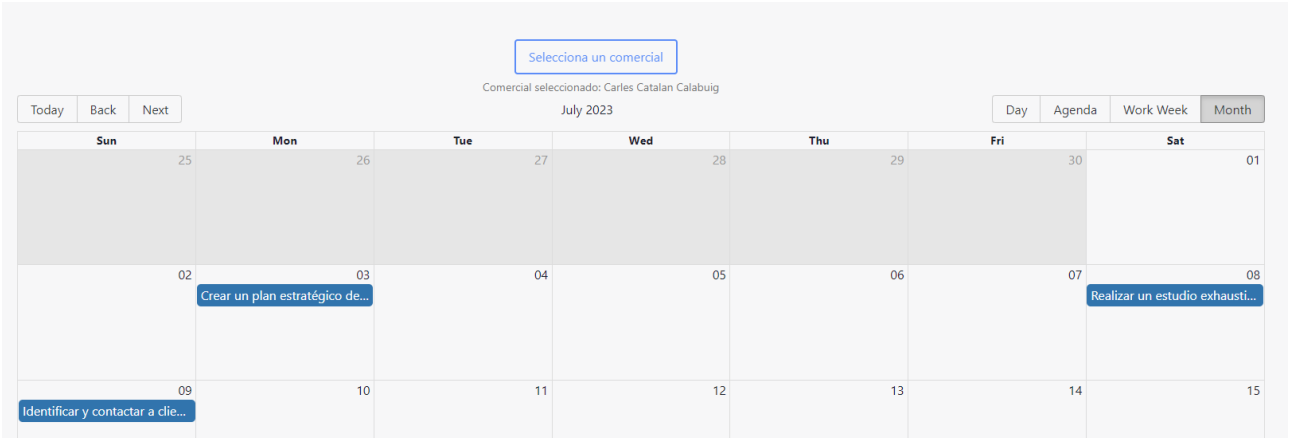


Ilustración 49: Vista general del calendario

Cuando se selecciona la "vista agenda", el calendario se mostrará en un formato de lista cronológica, en lugar de un formato de calendario tradicional con cuadrículas. En esta lista, se verán los eventos o hitos en orden cronológico, organizados por fecha y hora. Para cada evento, se mostrará la fecha en la que está programado y la descripción asociada a ese hito.

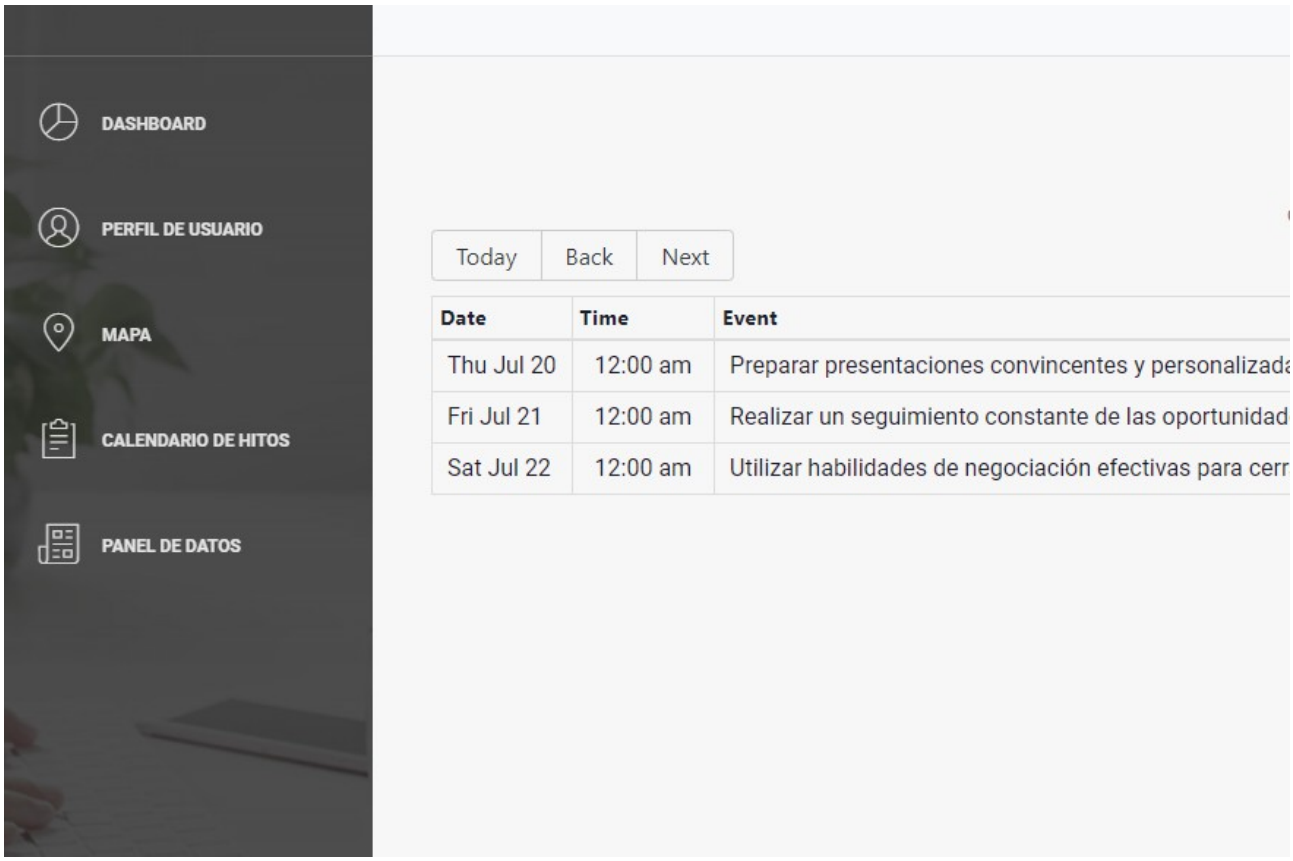


Ilustración 50: Vista calendario - agenda 1

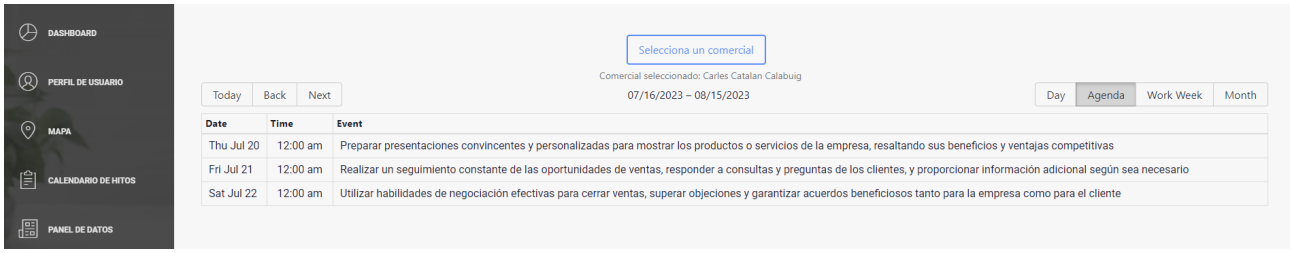


Ilustración 51: Vista calendario - agenda 2

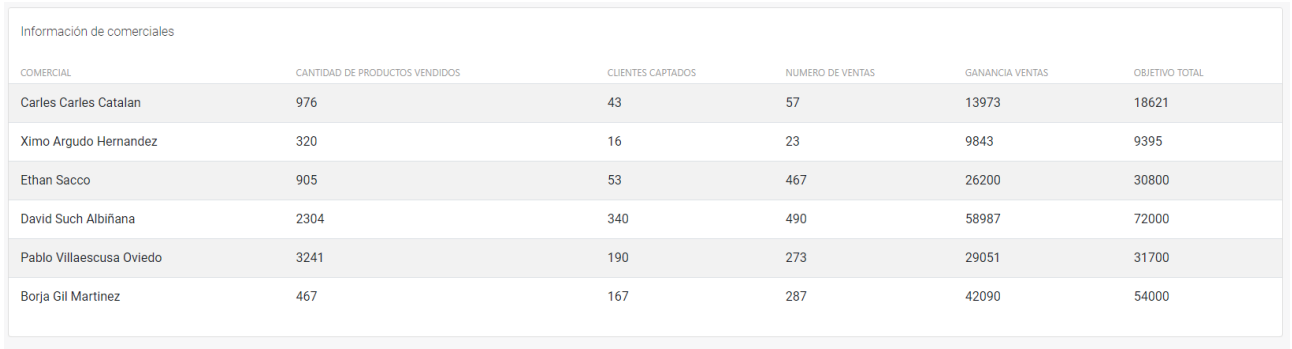
Por último, los administradores tendrán acceso a una tabla que proporciona una visión consolidada de las ventas realizadas por todos los comerciales en este año.

Esta tabla tiene como objetivo brindar una perspectiva completa y resumida del rendimiento de los comerciales en términos de ventas.

Información de comerciales

COMERCIAL	CANTIDAD DE PRODUCTOS VENDIDOS	CLIENTES CAPTADOS
Carles Carles Catalan	976	43
Ximo Argudo Hernandez	320	16
Ethan Sacco	905	53
David Such Albiñana	2304	340
Pablo Villaescusa Oviedo	3241	190
Borja Gil Martinez	467	167

Ilustración 52: Vista calendario - tabla 1



Información de comerciales

COMERCIAL	CANTIDAD DE PRODUCTOS VENDIDOS	CLIENTES CAPTADOS	NUMERO DE VENTAS	GANANCIA VENTAS	OBJETIVO TOTAL
Carles Carles Catalan	976	43	57	13973	18621
Ximo Argudo Hernandez	320	16	23	9843	9395
Ethan Sacco	905	53	467	26200	30800
David Such Albiñana	2304	340	490	58987	72000
Pablo Villaescusa Oviedo	3241	190	273	29051	31700
Borja Gil Martinez	467	167	287	42090	54000

Ilustración 53: Vista calendario - tabla 2

4.9 – Panel de datos

Por último encontramos la pantalla de 'Panel de datos' donde podremos asignar objetivos de venta a los comerciales en nuestra plataforma.

Primero tendremos que seleccionar un comercial desde el botón 'Ver comerciales'.

Ilustración 54: Vista panel de datos 1

Ilustración 55: Vista panel de datos 2

Contaremos con un paginador por años el cual nos permite consultar información a lo largo del tiempo.

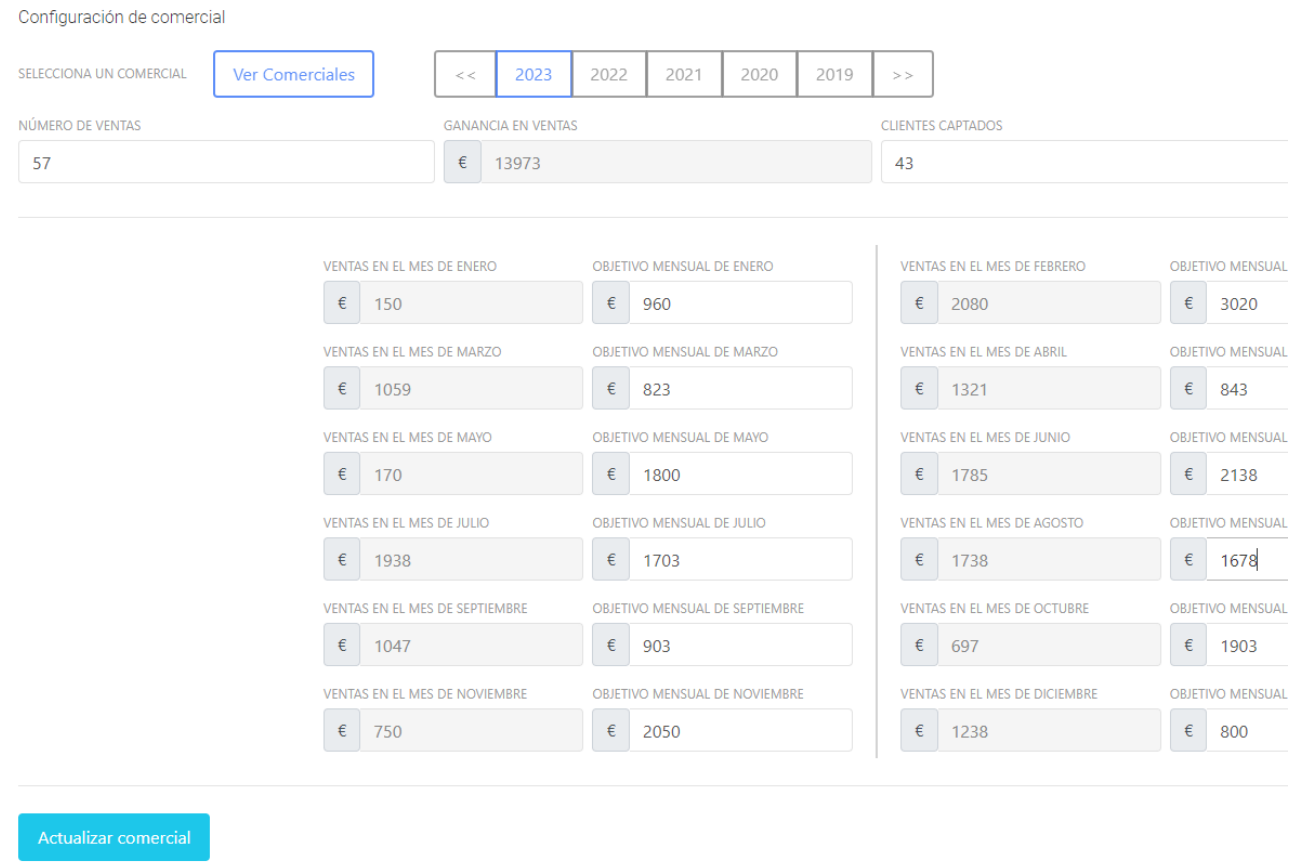


Ilustración 56: Vista panel de datos general

5 – Conclusiones y trabajos futuros

Estoy realmente satisfecho con el trabajo realizado en este proyecto. Al principio, tenía algunas dudas sobre qué tecnologías y temática elegir, pero al final, optar por el stack MERN ha resultado ser una buena decisión.

El stack MERN, con MongoDB, Express, React y Node.js, me ha dado la oportunidad de aumentar mis conocimientos en JavaScript, un lenguaje fundamental en el desarrollo web.

Lo más importante es que trabajar con este stack me ha permitido indagar aun más en la arquitectura cliente-servidor, un concepto esencial para construir aplicaciones web modernas y eficientes. Además, he podido profundizar en el patrón de diseño MVC (Modelo-Vista-Controlador), que ha mejorado significativamente mi habilidad para estructurar y organizar el código de manera más clara y sostenible.

El proceso de desarrollo ha sido enriquecedor y gratificante, y me ha proporcionado una valiosa experiencia en la creación de aplicaciones web completas y funcionales.

Además, he aprendido mucho sobre nuevas librerías, su usabilidad e implementación.

Una de las principales mejoras que podrían realizarse con este proyecto es la automatización de la entrada de datos, permitiendo que la información provenga directamente desde un ERP (Enterprise Resource Planning).

Para implementar esta mejora, se deberían establecer interfaces de conexión entre el proyecto y el ERP, asegurando que los datos se transmitan de manera segura y confiable.

La sincronización periódica o en tiempo real garantizaría que los datos estén siempre actualizados en la plataforma.

Aun así, sin duda, seria una mejoría compleja ya que en el mercado actual hay una ingente cantidad de sistemas ERP y habría que desarrollar conectores ad hoc para cada uno de ellos.

6 – Bibliografía

1. *MongoDB documentation*. (s. f.-b). MongoDB Documentation. https://www.mongodb.com/docs/
2. *Express Web Framework (Node.js/JavaScript) - Learn Web Development | MDN*. (2023, 3 julio). https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
3. *Getting started – react*. (s. f.). React. https://legacy.reactjs.org/docs/getting-started.html
4. *Run JavaScript everywhere.* (s. f.). Run JavaScript Everywhere. https://nodejs.dev/en/
5. *MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la web | MDN*. (s. f.). https://developer.mozilla.org/es/docs/Glossary/MVC
6. colaboradores de Wikipedia. (2021). Modelo–Vista–Controlador. *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
7. *W3Schools online web tutorials*. (s. f.). https://www.w3schools.com/
8. Google Maps Platform. (s. f.). *Google for Developers*. https://developers.google.com/maps?hl=es-419
9. *GitHub: Let's build from here*. (s. f.). GitHub. https://github.com/
10. *Mongoose ODM V7.4.0*. (s. f.). https://mongoosejs.com/
11. *Express - Infraestructura de aplicaciones web Node.js*. (s. f.). https://expressjs.com/es/
12. *Chakra UI - a simple, modular and accessible component library that gives you the building blocks you need to build your REACT applications.* (s. f.). Chakra UI: Simple, Modular and Accessible UI Components for your React Applications. https://chakra-ui.com/
13. Contributors, M. O. J. T. A. B. (s. f.). *Bootstrap*. https://getbootstrap.com/
14. *Empezando | Axios Docs*. (s. f.). https://axios-http.com/es/docs/intro
15. *Visión general Cliente-Servidor - Aprende Desarrollo Web | MDN*. (2023, 9 marzo). https://developer.mozilla.org/es/docs/Learn/Server-side/First_steps/Client-Server_overview
16. *Get started with Atlas — MongoDB Atlas*. (s. f.). https://www.mongodb.com/docs/atlas/getting-started/
17. *NPM: React-scripts*. (s. f.). npm. https://www.npmjs.com/package/react-scripts

7. Anexos

Para la realización de este proyecto ha sido necesario preparar el entorno de trabajo con la instalación de Node.js, NPM y GIT. A continuación haremos un breve repaso para la instalación de las herramientas.

Visitaremos el sitio web oficial de Node.js (https://nodejs.org) y descargaremos la versión LTS (Long-Term Support) adecuada para nuestro sistema operativo.

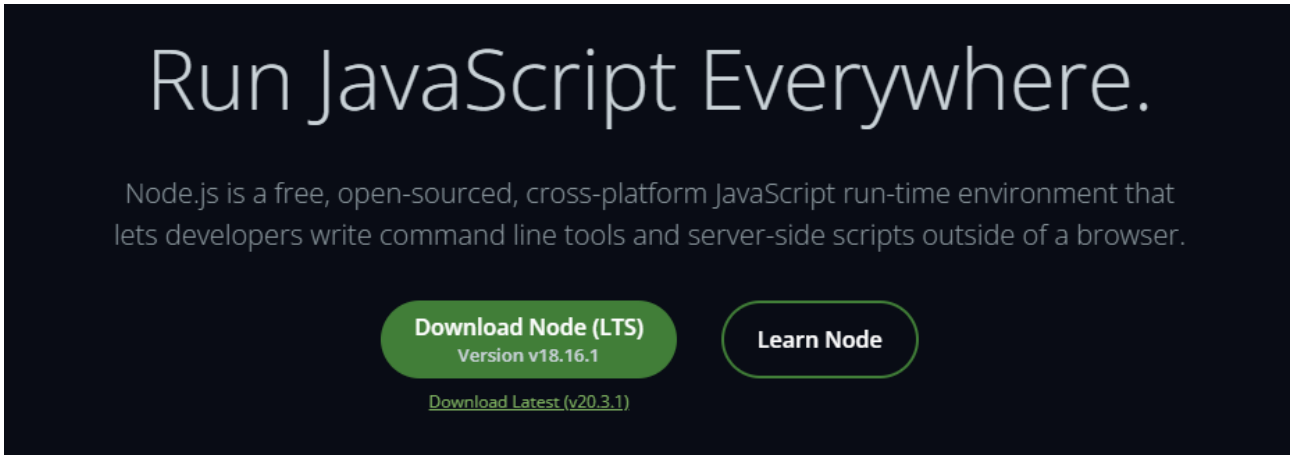


Ilustración 57: NodeJS LTS

Ejecutaremos el instalador y seguiremos las instrucciones para completar la instalación de Node.js y NPM (Node Package Manager).

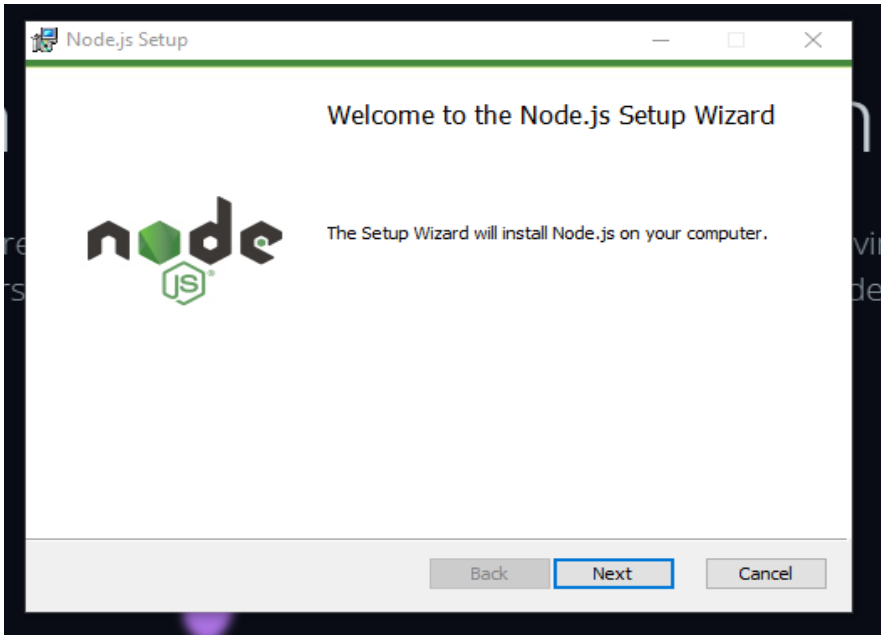


Ilustración 58: Instalación de NodeJS LTS

Para la instalación de Git Bash, navegaremos al sitio oficial e instalaremos la versión optima y seguiremos los pasos del instalador.

Ilustración 59: Instalación Git Bash