UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

APLICACIÓN WEB PARA GESTIONAR INFORMACIÓN LOGÍSTICA EN UNA EMPRESA COURIER CON E-COMMERCE APLICANDO LA METODOLOGÍA SCRUM

SALCEDO ELIZALDE DANIEL MARCELO
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

APLICACIÓN WEB PARA GESTIONAR INFORMACIÓN LOGÍSTICA EN UNA EMPRESA COURIER CON E-COMMERCE APLICANDO LA METODOLOGÍA SCRUM

SALCEDO ELIZALDE DANIEL MARCELO
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TRABAJO TITULACIÓN
PROPUESTAS TECNOLÓGICAS

APLICACIÓN WEB PARA GESTIONAR INFORMACIÓN LOGÍSTICA EN UNA EMPRESA COURIER CON E-COMMERCE APLICANDO LA METODOLOGÍA SCRUM

SALCEDO ELIZALDE DANIEL MARCELO
INGENIERO DE SISTEMAS

VALAREZO PARDO MILTON RAFAEL

MACHALA, 20 DE SEPTIEMBRE DE 2022

MACHALA
2022

APLICACIÓN WEB PARA LA GESTIÓN DE INFORMACIÓN LOGÍSTICA EN UNA EMPRESA COURIER CON E-COMMERCE APLICANDO LA METODOLOGÍA SCRUM.

INFORME DE ORIGINALIDAD

3%	2%	0%	1%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	documentop.com Fuente de Internet	1%
2	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	<1%
3	tuprogramas.com Fuente de Internet	<1%
4	www.godan.info Fuente de Internet	<1%
5	Submitted to University of Debrecen / DEA Trabajo del estudiante	<1%
6	www2.deloitte.com Fuente de Internet	<1%
7	www.averlo.com Fuente de Internet	<1%
8	ftp.riken.jp Fuente de Internet	<1%

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN EL REPOSITORIO DIGITAL INSTITUCIONAL

El que suscribe, SALCEDO ELIZALDE DANIEL MARCELO, en calidad de autor del siguiente trabajo escrito titulado APLICACIÓN WEB PARA GESTIONAR INFORMACIÓN LOGÍSTICA EN UNA EMPRESA COURIER CON E-COMMERCE APLICANDO LA METODOLOGÍA SCRUM, otorga a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tiene potestad para otorgar los derechos contenidos en esta licencia.

El autor declara que el contenido que se publicará es de carácter académico y se enmarca en las dispociones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

El autor como garante de la autoría de la obra y en relación a la misma, declara que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asume la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

Machala, 20 de septiembre de 2022

SALCEDO ELIZALDE DANIEL MARCELO
0706363108

DEDICATORIA

Dedico este trabajo a Dios, mi inspiración y padre en todo momento, pilar principal de mi vida, forjador de mis metas y manifiesto de verdad.

Se lo dedico a mi hija, mi madre, mis hermanos, familiares, amigos y docentes que han puesto su voto de confianza en mis pasos, han estado pendiente de mis progresos y se han alegrado con mis triunfos; a todos ellos, este logro es también suyo.

También está dedicado a todos aquellos que aún no intentan, porque si yo pude, todos podemos. Y lo dedico a todas las personas que día a día buscan superarse obrando bien y buscando paz, ya que así empezó mi andar.

Sr. Salcedo Elizalde Daniel Marcelo

AGRADECIMIENTO

Agradezco a Dios por estar presente en mi vida, a mis padres por su educación y apoyo brindado, mis hermanos por estar siempre presentes y unidos en todo momento, mi hija por darme la actitud que necesito en el momento adecuado, a mis compañeros por ser parte de esta trayectoria tan importante para nuestras vidas, ya que sus observaciones, consejos y ayuda han repercutido de manera positiva en mi formación profesional, a mi novia Valeria Salas, por su carácter que inspiró en mí, fe, bondad, y lealtad.

Agradezco a mi tutor de tesis y amigo, Ing. Milton Valarezo por indicarme el camino correcto a seguir en mi formación profesional, y al Ing. Jimmy Molina por sus cátedras y su virtud que preserva una temática bien estructurada sobre la ingeniería de software.

Sr. Salcedo Elizalde Daniel Marcelo

RESUMEN

Las operaciones de logística en una empresa Courier con E-Commerce constituyen la base fundamental en el éxito de su modelo de negocios, ante la creciente demanda de este servicio en los últimos años es fundamental contar con un sistema software que ayude a gestionar la información de manera organizada, confiable, transparente y automatizada; esto enriquece el prestigio de la empresa y la posiciona sobre competidores que carecen de herramientas tecnológicas para mejorar la gestión de los servicios prestados. El presente proyecto está centrado en desarrollar un sistema web robusto y amigable haciendo uso de la metodología ágil scrum; por consiguiente contiene la información que detalla el product backlog en base a las historias de usuarios que recopila el product owner, el sprint backlog general que desarrolla el scrum master, y la descripción de las actividades que realiza el team developer en cada uno de los sprints necesarios para la construcción de la aplicación hasta el despliegue y puesta en producción de la misma. Para ello se lleva a cabo la implementación de una arquitectura cliente – servidor, haciendo uso de tecnologías las multiplataforma PostgreSQL para la base de datos, Angular 12 para la capa de presentación y .Net 5.0 para la programar la lógica del negocio, se implementan los patrones scoped y transient con el fin de administrar correctamente el tiempo de vida en las instancias que crea una petición REST, por consiguiente todos los controladores utilizan inyección de dependencia para instanciar la interfaz de uno o varios servicios, con la finalidad de usar los métodos que declara el contrato. Los servicios de persistencia se manejan a través del ORM Entity Framework Core, mismo que se encarga de administrar la capa de acceso de información. La capa de dominio concede autorización a sus métodos siempre que exista un Json web token en el header de las solicitudes HTTP, dichas solicitudes llegan de manera cifrada con formato JWT Bearer para ser analizadas por las políticas del negocio y acceso a la información. La aplicación cuenta con mapeadores para la conversión de objeto a objeto, inyección de dependencia para inicializar servicios, elasticidad en la implementación de sus métodos, entre otras prácticas de programación que ayudan al desarrollo de un código limpio, modular y fácil de interpretar. Se utiliza la arquitectura de tres capas para el desarrollo de la solución y se despliega todas las tecnologías mencionadas en un servidor con sistema operativo CentOS. Se utilizó la herramienta Postman para hacer test al backend, se evaluó el prototipo final mediante el cumplimiento funcionalidades importantes del proyecto haciendo uso de una herramienta creada por el scrum master, esta evaluación se realizó con el proyecto puesto a producción y se valoró los resultados del test a través de una métrica porcentual, la evaluación puso a prueba la implementación de la arquitectura de hardware y software, la comunicación entre capas y la construcción del proyecto con patrones de diseño; finalmente se determinó que el nivel de satisfacción respecto a las funcionalidades del sistema está sobre el 90%, lo que indica que se ha logrado construir y desplegar un producto altamente funcional.

Palabras clave: Net 5.0, Scrum, PostgreSQL, web development, SPA, Angular, JWT Bearer.

ABSTRACT

Logistics operations in a Courier company with E-Commerce constitute the fundamental basis for the success of its business model, given the growing demand for this service in recent years, it is essential to have a software system that helps manage the information an organized, reliable, transparent and automated manner; this enriches the prestige of the company and positions it against competitors that lack technological tools to improve the management of the services provided. This project is focused on developing a robust and friendly web system using the agile scrum methodology; therefore, it contains the information that details the product backlog based on the user stories collected by the product owner, the general sprint backlog developed by the scrum master, and the description of the activities carried out by the development team in each of the sprints necessary for the construction of the application, from its inception to its implementation and production. For this, the implementation of a client-server architecture is carried out, making use of the PostgreSQL multiplatform technologies for the database, Angular 12 for the presentation layer and .Net 5.0 for the programming of the business logic, the patterns "scoped and transient" to manage the lifetime of instances created by a REST request, therefore all controllers use dependency injection to instantiate the interface of one or more services and to use the methods that the contract declares. Persistence services are managed through the ORM Entity Framework Core, which is responsible for managing the information access layer. The domain layer grants authorization to its methods as long as there is a Json web token in the header of the HTTP requests, these requests arrive in encrypted form with JWT Bearer format to be analyzed by business policies and access to information. The application has mappers for object-to-object conversion, dependency injection to initialize services, elasticity in the implementation of its methods to generate different responses from the server, among other programming practices that help the development of clean, modular, reliable, and code easy to interpret. For the development of the solution, the three-tier architecture is used and all the mentioned technologies are deployed on a server with the CentOS operating system. The Postman tool was used to test the backend, the final prototype was evaluated fulfilling important functionalities of the project using a tool created by the scrum master, this evaluation was carried out with the project put into production and the results of the test through a metric percentage, the evaluation tested the implementation of the hardware and software architecture, the communication between layers and the construction of the project with design patterns; Finally, it was determined that the level of satisfaction with respect to the functionalities of the system is greater than 90%, which indicates that a highly functional product has been built and deployed.

Keywords: .Net 5.0, Scrum, PostgreSQL, web development, SPA, Angular, JWT Bearer.

Índice de Contenido

Índice de Tablas

Índice de Figuras

Índice de Anexos

Glosario de términos

Framework.- Marco o esquema de trabajo

DTO.- Data Transfer Object (Objeto para transferencia de datos)

DOM.- Document Object Model (Modelo de objeto de documento)

Paquete NuGet.- Son unidades de código reutilizable que otros desarrolladores ponen a su disposición para que los use en sus proyectos.

INTRODUCCIÓN

Un Courier se encarga de realizar importaciones bajo varias modalidades según describe el Comité de Comercio Exterior (Comex) en Ecuador, siendo la categoría B – 4x4 una de las más usadas por personas naturales que desean importar artículos para su uso personal sin fines comerciales; Esta categoría es la preferida por varios sistemas de E-Commerce como Amazon, Tienda Mia u otros negocios que residen en el exterior ya que eximen al cliente de pagar impuestos o derechos de importación siempre y cuando se sigan las normativas establecidas en la resolución 008-2022 COMEX [1] por consiguiente una persona puede comprar productos en varias tiendas de USA y esperar a recibirlas en la comodidad de su hogar por una tarifa fija de envío sin más complicaciones.

Desde 2020 a raíz de la pandemia ocasionada por el virus COVID-19 hubo una creciente demanda en este modelo de negocio ya que los ciudadanos deseaban no salir de sus hogares siempre que les fuera posible y con ello aparece la necesidad de tener un casillero postal en USA que consolide sus compras en una sola encomienda para enviar a Ecuador.

Por esta razón la empresa Envíos Ecuador decide expandir sus procesos de negocio y dar un mejor servicio a sus clientes implementando un sistema de casillero postal que permita gestionar de manera organizada el proceso de recepción y envío de encomiendas consolidadas para aquellos usuarios que requieran el servicio. El principal enfoque se encuentra en transparentar el trabajo realizado e inspirar confianza a los usuarios de tal manera que se optimice la gestión en atención al cliente y las operaciones logísticas de la empresa creando una sistema web con angular, .Net 6 y PostgreSQL que registre información de los clientes respecto a las encomienda recibidas en el casillero postal, sus productos, depósitos, deudas y despachos con un seguimiento de estos hasta el momento de la entrega final al cliente.

Para llevar a cabo el desarrollo del proyecto se ha establecido requerimientos específicos del sistema, actores que intervienen, políticas de autorización a la información y una interfaz amigable para el usuario con una arquitectura cliente – servidor utilizando tecnologías multiplataforma como es PostgreSQL, angular 12, .Net 5.0 y varios frameworks que garantizan un desarrollo robusto, elástico, escalable, altamente eficiente y mantenible.

Se ha optado por realizar este trabajo siguiendo la metodología scrum y evaluando el prototipo final terminado haciendo un test desde la capa de presentación a la capa de dominio; misma que está correctamente enlazada a la base de datos mediante la capa de acceso a la información, siguiendo la arquitectura de 3 capas y cliente - servidor la información devuelta es coherente, confiable y conforme a los requerimientos establecidos.

A continuación, se presenta la estructura del presente documento:

Capítulo I: Se expone la problemática, el origen y contexto de la investigación, la justificación del mismo y se listan los requerimientos principales del proyecto.

Capítulo II: Se define el prototipo a desarrollar, se expone un marco teórico de las tecnologías y técnicas utilizadas, así como la metodología de desarrollo.

Capítulo III: Se evalúa la funcionalidad del producto final en base a los requerimientos y se exponen los resultados obtenidos.

CAPÍTULO I
DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS.

1.1. ÁMBITO DE APLICACIÓN: DESCRIPCIÓN DEL CONTEXTO Y HECHOS DE INTERÉS.

El servicio de Courier se enfoca en llevar correspondencia o paquetería de un lugar a otro, las empresas con una alta demanda necesitan establecer una logística que satisfaga las actuales necesidades de los consumidores como información detallada de sus encomiendas, servicio de ruta y estado de su envió desde el inicio hasta la entrega. De la misma manera el E-Commerce va de la mano con los servicios de Courier ya que se ofrece un producto de manera online y se necesita de un agente Courier para asegurar que el cliente reciba su producto [2], [3].

El E-Commerce ha conseguido una alta creciente de demanda a raíz del COVID-19 dado que los consumidores al no salir de sus hogares por las prohibiciones de ley o simplemente por evitar aglomeraciones se vieron en la necesidad de realizar sus compras de manera online, es ahí donde el miedo de comprar en línea se fue perdiendo, gracias a esto actualmente es la manera favorita de compra para el promedio.

Varios consumidores actualmente compran online a una empresa en la cual ponen su confianza al azar esperando no ser estafados y aunque la mayoría son negocios serios siempre existen personas malintencionadas que esperan estafar, por otra parte, puede haber incidencias que ocasionen un retraso en sus entregas o respuestas tardías por parte del proveedor dado la alta demanda que se puede llegar a tener en sus líneas. Todo esto ocasiona el recelo del consumidor en volver a comprar dado que no se da una buena gestión a la información de manera transparente, puntual, masiva y en tiempo real [4].

Por todo lo mencionado se ha motivado el desarrollo de una solución tecnológica web que proporcione servicios de información en tiempo real sobre las incidencias en sus pedidos y entregas de tal manera que se desarrollará una base de datos en PostgreSQL que guarde toda la información de la empresa Courier, dos servicios API RESTFUL (administrador y cliente) desarrollado en .NET 5.0 para la comunicación de datos y permisos de usuarios al SGBD y 2 aplicaciones frontend (administrador y cliente) desarrolladas con el framework Angular 11 de manera robusta, flexible, escalable e intuitiva para asegurar una buena experiencia al usuario y a la gestión de la información.

1.2. ESTABLECIMIENTO DE REQUERIMIENTOS.

Para empezar con el desarrollo de esta solución se debe conocer a cabalidad los requerimientos del sistema y las necesidades de los usuarios, por este motivo se ha realizado varias entrevistas enfocadas en las diferentes actividades que se realizan en la empresa Courier a fin de poder automatizar los procesos involucrados y transparentar la información.

Así se ha identificado los siguientes requerimientos:

Crear un registro de clientes que guarde información personal de cada individuo como: Nombre completo, ciudad donde reside, fecha de nacimiento, número de cédula, email, entre otros; con el fin de poder identificarlos plenamente.

Crear un registro de empleados que identifique los colaboradores de la empresa y el establecimiento en que opera.

Creación automática de usuarios para clientes y empleados con la posibilidad de establecer permisos para cada uno de ellos y niveles de accesibilidad para la gestión de información.

Creación de un registro que almacene las transacciones de compra realizadas por los clientes, tales con información como productos comprados, tienda de origen, número de seguimiento, fecha de compra, imagen de evidencia, entre otros. Con el fin de almacenar un registro acerca del consumo de cada cliente.

Crear un registro que almacene las encomiendas recibidas en el casillero postal, juntamente con los productos que contiene cada encomienda.

Crear un sistema de etiquetado para cada producto con un código único y un historial de cambios desde su fecha de creación hasta la entrega al cliente final.

Crear un registro de depósitos, devoluciones y pagos en efectivo del cliente con el fin de llevar un historial detallado de su cuenta como acreditaciones, débitos, saldos a favor; saldos negativos y depósitos verificados, no encontrados o pendientes de revisión.

Crear un proceso de verificación automática aplicada a los depósitos según se carga un estado de cuenta específico.

Creación de guías para la movilización de mercadería por los diferentes puntos logísticos de la empresa Courier con el fin de poder identificar en qué guía se moviliza cada producto y su locación actual.

Creación de guías para la entrega de encomiendas desde una posición logística del Courier hasta el domicilio del cliente con la generación automática de un nuevo registro de débito en la cuenta del cliente.

Implementar un inicio y cierre de sesión que responda un token de autorización con el fin de permitir a los administradores, clientes y empleados comunicarse con el backend de la aplicación y sus diferentes puntos de ruta final.

Implementar Operaciones CRUD con niveles de permisos según corresponda al cliente, empleado o administrador del negocio.

Crear un tablero de información que corresponda a cada cliente donde se pueda visualizar sus movimientos de cuenta, compras, entregas y productos de manera individual, clara y amigable.

Crear formularios para editar el perfil de los clientes o empleados; editar productos, depósitos, órdenes de compra, productos recibidos, guías de movilización, guías de entrega y verificación de depósitos.

Desplegar las aplicaciones desarrolladas para la solución del problema en un servidor remoto, establecer sus respectivos DNS y levantar los servicios necesarios para dejar la aplicación corriendo lista para ser usada por múltiples usuarios desde cualquier navegador web.

1.3. JUSTIFICACIÓN DEL REQUERIMIENTO A SATISFACER.

Esta solución tecnológica basada en el desarrollo web de varias aplicaciones se realizará con el fin de mejorar la gestión de información de la empresa con el consumidor, por consecuencia se espera aumentar la confianza del cliente en la empresa, menos carga de trabajo para los colaboradores, transparencia en los procesos de logística y un fácil acceso a todos los procesos involucrados en la prestación de los servicios ofertados por la empresa Courier.

Se podrá acceder a toda esta información de manera remota desde cualquier parte del mundo, con tan solo tener acceso a internet, un navegador web y las claves de autenticación para la autorización de diferentes peticiones según el rol y permisos que se asignan a cada usuario.

Para llevar a cabo este proyecto se utilizará la metodología SCRUM con el fin de garantizar la calidad, escalabilidad y funcionalidad de toda la solución web que se va a realizar.

CAPÍTULO II
DESARROLLO DEL PROTOTIPO.

2.1 DEFINICIÓN DEL PROTOTIPO TECNOLÓGICO

Para el desarrollo tecnológico de la solución a los requerimientos planteados se implementan diversas tecnologías, frameworks, entornos de desarrollo y buenas prácticas estandarizadas con una arquitectura cliente – servidor.

El presente proyecto desarrolla un sitio web SPA con Angular para la ejecución del cliente, y una API para recibir solicitudes POST, PUT, UPDATE y PATCH.

Se configura el intercambio de datos a través del protocolo HTTP y HTTPS, asegurando la transferencia de información.

Se incorpora una función de autorización basada en tokens y políticas de negocio para mejorar la seguridad de la comunicación entre cliente – servidor y el acceso a los datos.

En la figura 1 podemos observar el esquema de comunicación con las diferentes funciones del software desarrollado en este proyecto.

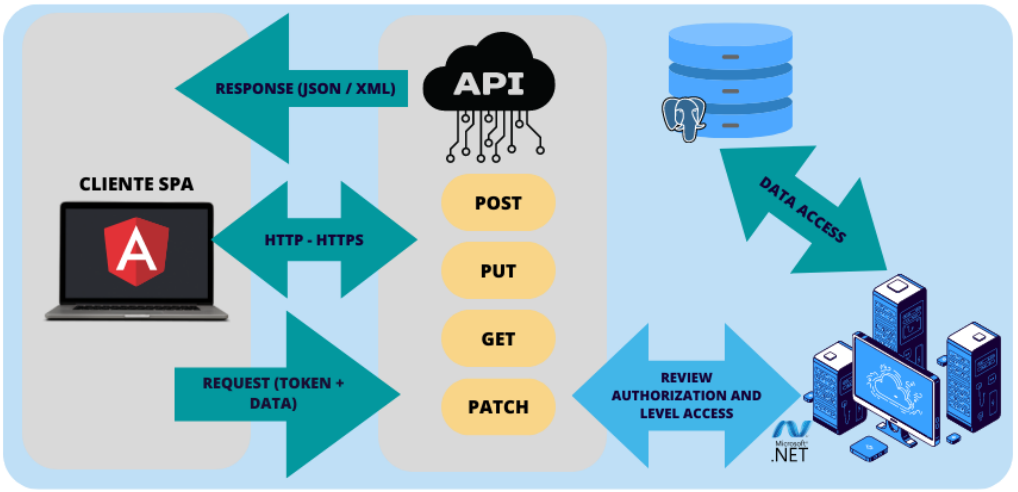


Figura 1: Prototipo tecnológico

Haciendo referencia la figura 1, podemos describir cómo funciona nuestro prototipo tecnológico, en primer lugar se observa un sitio web SPA con Angular para la ejecución del cliente, y una API para recibir solicitudes POST, PUT, UPDATE y PATCH.

Al realizar una solicitud desde el navegador web, se envía un token de autorización y la data que espera el end point a través del protocolo HTTP/HTTPS. Posteriormente la API se comunica con el servidor para revisar el nivel de acceso a los datos del usuario y las políticas del negocio a las que puede acceder; una vez validado que todo esté correcto, el servidor .NET accede a la base de datos para extraer o crear información, luego remite dicha información a la API donde se construye la respuesta en formato JSON o XML para ser enviada al cliente.

Cuando el usuario realiza una solicitud no autorizada según el nivel de acceso a los datos que tiene y las políticas del negocio, el servidor .NET no establece comunicación con la base de datos, no procesa la petición y niega el servicio.

Con la finalidad de mejorar la gestión de atención al cliente y las operaciones del negocio; este proyecto contará con la función de registrar usuarios, depósitos, productos, agencias, despachos, empleados, reembolsos y guías aéreas, generar cuentas por cobrar, cargar movimientos bancarios, hacer seguimiento a los productos registrados, escanear códigos de barra y códigos QR, administrar permisos y visualizar información del negocio en general, esto le permite a la empresa llevar un mejor control de sus clientes y las tareas que se realizan.

A continuación, se detallan las estructuras utilizadas en este proyecto.

2.1.1 Arquitectura de hardware

Se detalla la arquitectura de hardware que usa el prototipo tecnológico para su sostenibilidad; aquí listamos los componentes computacionales y la red de servidores que requiere el despliegue de la solución de este proyecto

2.1.1.1 Servidor VPS (HostGator)

Este es un servidor privado virtual que proporciona la compañía de HostGator, una opción muy flexible a la hora de realizar configuraciones propias para la solución de

nuestro proyecto ya que nos permite instalar o desinstalar funcionalidades a nuestro gusto, así como realizar las configuraciones que consideremos necesarias [5], similar a un servidor dedicado pero creado a través de virtualización.

Este servidor cuenta con sistema operativo CentOS, mismo que contiene la base de datos PostgreSQL y el compilado de nuestra API que se desarrolla con .Net 5.0, un servidor de correo electrónico y el servidor web que alimenta los diferentes navegadores para los clientes.

2.1.1.2 Arquitectura cliente – servidor

Se hace uso de esta arquitectura, dado su funcionamiento autónomo, lo cual permite trabajar con diferentes lenguajes de programación sin perder la compatibilidad de los datos; de la misma forma los recursos utilizados por cada actor son independientes en producción, mientras que en una primera etapa de desarrollo podrían trabajar desde un mismo computador hasta el despliegue.

Como se puede apreciar en la figura 2, el servidor web aloja el compilado para desplegar la interfaz de usuarios en los diferentes navegadores; desde esta interfaz web se pueden comunicar a través de peticiones PUT, POST, GET, PATCH, HEAD OPTIONS y DELETE con la API que se aloja el servidor KESTREL, misma que contiene el compilado desarrollado con .Net 5.0 y desde el servidor KESTREL se realiza la comunicación con el servidor de base de datos POSTGRES para realizar las diferentes operaciones CRUD según sea el caso, a partir de este punto se puede extender la arquitectura con microservicios [6].

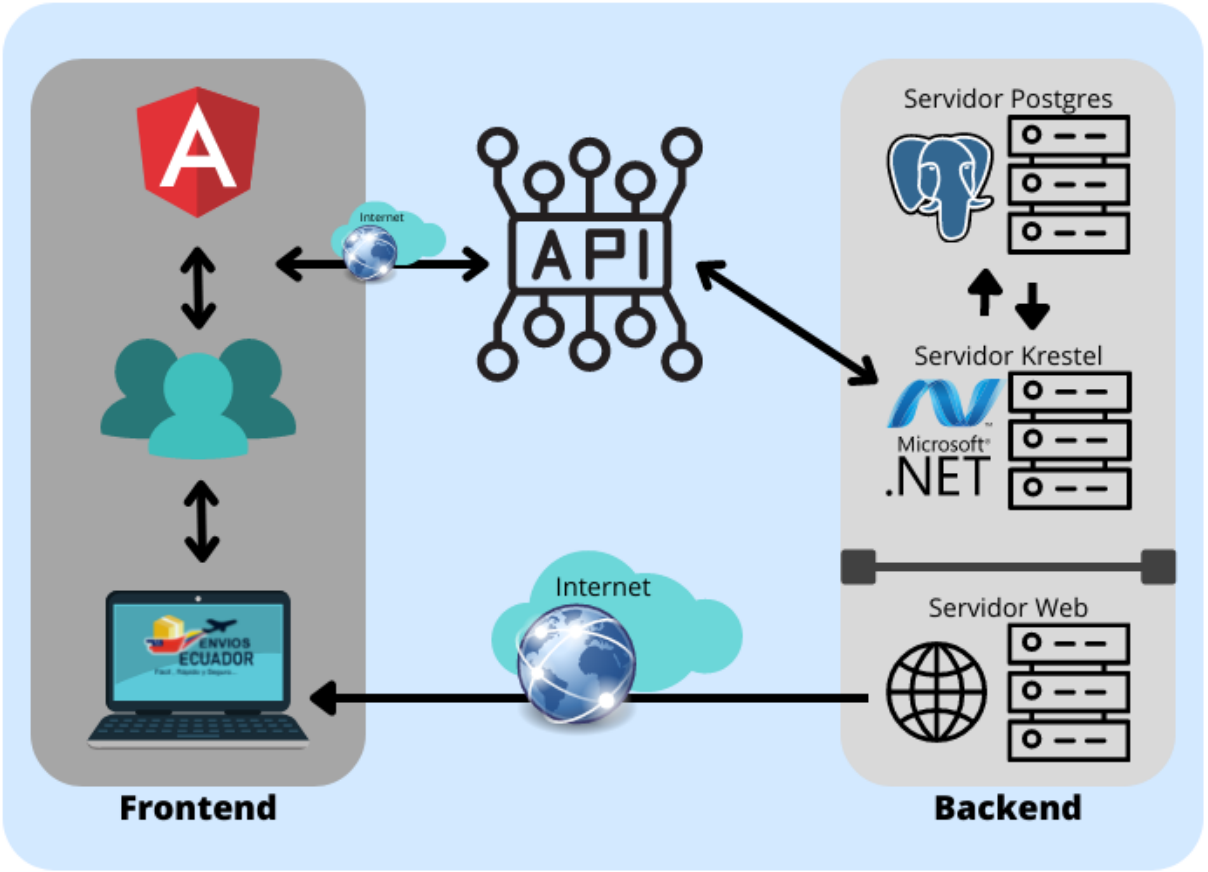


Figura 2: Arquitectura Cliente - Servidor

2.1.1.3 Implementación de la arquitectura

Para la correcta función de escalabilidad en el proyecto y un concepto de elasticidad bien definido se implementa una API RESTFUL para el intercambio de datos, así se asegura que el servidor pueda recibir data en formato XML, Json, texto plano, bits de imagen, vídeo y archivos.

2.1.2 Arquitectura de software

En esta sección se escoge las tecnologías adecuadas para poner en desarrollo el proyecto planteado, entre esto se destaca el lenguaje de programación, los frameworks que se utilizarán para dicho lenguaje, servicios a implementar, patrones de diseño que se incluirán en la codificación, herramientas de modelado, metodología de desarrollo y diseño del código en general.

El presente proyecto implementa una arquitectura de 3 capas, misma que es estructurada por el team developer y el scrum master, posterior al producto backlog recibido del product owner [7] según la metodología de desarrollo Scrum que utilizamos.

Una buena arquitectura influye directamente en el concepto de escalabilidad del proyecto y mantenibilidad del código [8], permitiendo un código limpio con buenas prácticas, cumpliendo estándares de programación, documentación y pruebas.

2.1.2.1 Arquitectura de tres capas

Es un tipo de arquitectura centrada en dividir el diseño de la aplicación de tal manera que la capa inferior solo pueda comunicarse con su capa superior sin depender de ella [9], mientras que la capa superior depende de la inferior para la entrada de datos, pero no para el procesamiento de los mismos.

En la figura 3 podemos ver como la comunicación es bidireccional desde uno o varios clientes hacia una capa de negocios centralizada (API) y desde la API a la base de datos. Así el cliente y los almacenes de datos no se conectan entre sí de manera directa.

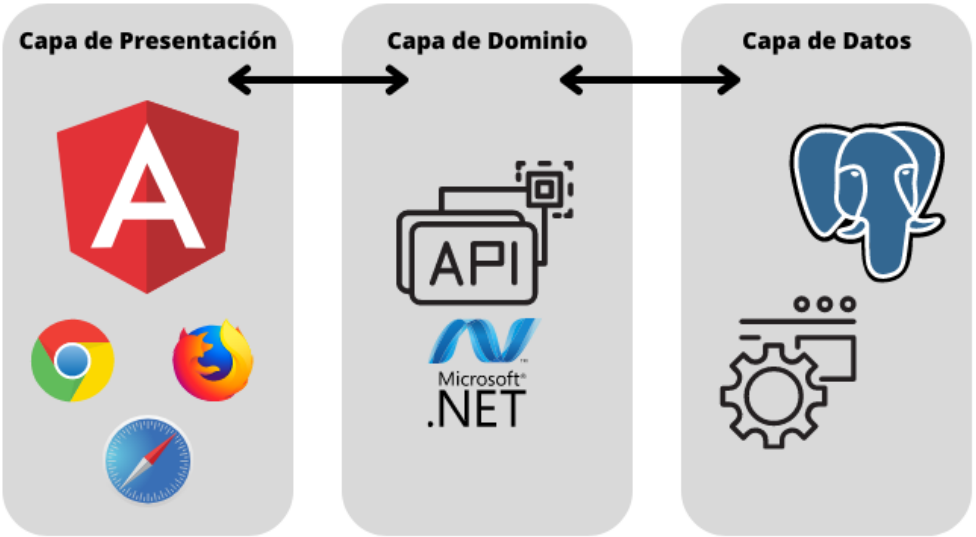


Figura 3: Arquitectura de tres capas

A continuación, detallaremos cada capa.

Capa de datos o acceso a datos (capa de persistencia)

Este nivel se encarga de conectarnos con uno o varios almacenes de datos para el correcto procesamiento de datos en la aplicación; estos pueden ser bases de datos relacionales y no relacionales como también archivos de texto plano, Json, entre otros. Es aquí donde se implementan los repositorios, y se especifica el contexto que interactúa con la base de datos.

Capa de dominio o capa de negocio

Es también conocido como la capa de nivel lógico o medio, es el núcleo de la aplicación [10], aquí se procesan los datos de entrada y salida de tal manera que esta capa interactúa con la capa de datos y la capa de presentación.

Esta capa contiene la lógica del negocio y es donde se especifica las reglas y políticas en las peticiones o respuestas según el nivel de acceso que posea un usuario, en ella usamos mapeadores, decodificadores de tokens, modelos Dtos para peticiones y comandos; también se encarga de validar los datos que vienen de la capa de presentación y evalúa los datos que se envían a sus diferentes capas respectivamente.

Esta capa contiene algoritmos y lógica que se ejecuta para cada petición, aquí se puede analizar si un usuario tiene el permiso para dicha petición, y el tipo de respuesta que se debe dar al cliente; dicho de una manera simplificada, en esta capa se establecen los contratos u interfaces y se declaran los patrones de diseño, para esta aplicación transient y scoped.

Capa de presentación

En esta capa implementamos la comunicación del cliente con el sistema a través de una aplicación web SPA desarrollada con angular. En este nivel usamos HTML, scss y typescript. Posteriormente se detalla cómo funcionan estas tecnologías en conjunto.

2.2 FUNDAMENTACIÓN TEÓRICA DEL PROTOTIPO

Este Proyecto es construido con tecnologías pioneras en el área de desarrollo de software, tales como .Net 5.0, un framework multiplataforma creado por Microsoft, Angular 11 creado por el equipo de Facebook y soportado por una gran comunidad de desarrolladores para la creación de sitios web SPA que implementan AJAX y una CLI potente para procesar código compilado en entornos de desarrollo y transformarlo en código interpretado para producción añadiendo de manera organizada y estandarizada una estructura que permite separar y vincular el hipertexto con sus funciones en componentes reutilizables y dinámicos; así como también PostgreSQL 12 una base de datos relacional open source de manejo profesional que permite disparadores, procedimientos almacenados, creación de vistas y tablas temporales entre otros [11].

Se pone a trabajar los mecanismos relacionados al servidor VPS, instalamos CentOS como sistema operativo, mismo que maneja la conectividad de la red, y permite crear los servicios necesarios para la puesta en producción de este proyecto [12].

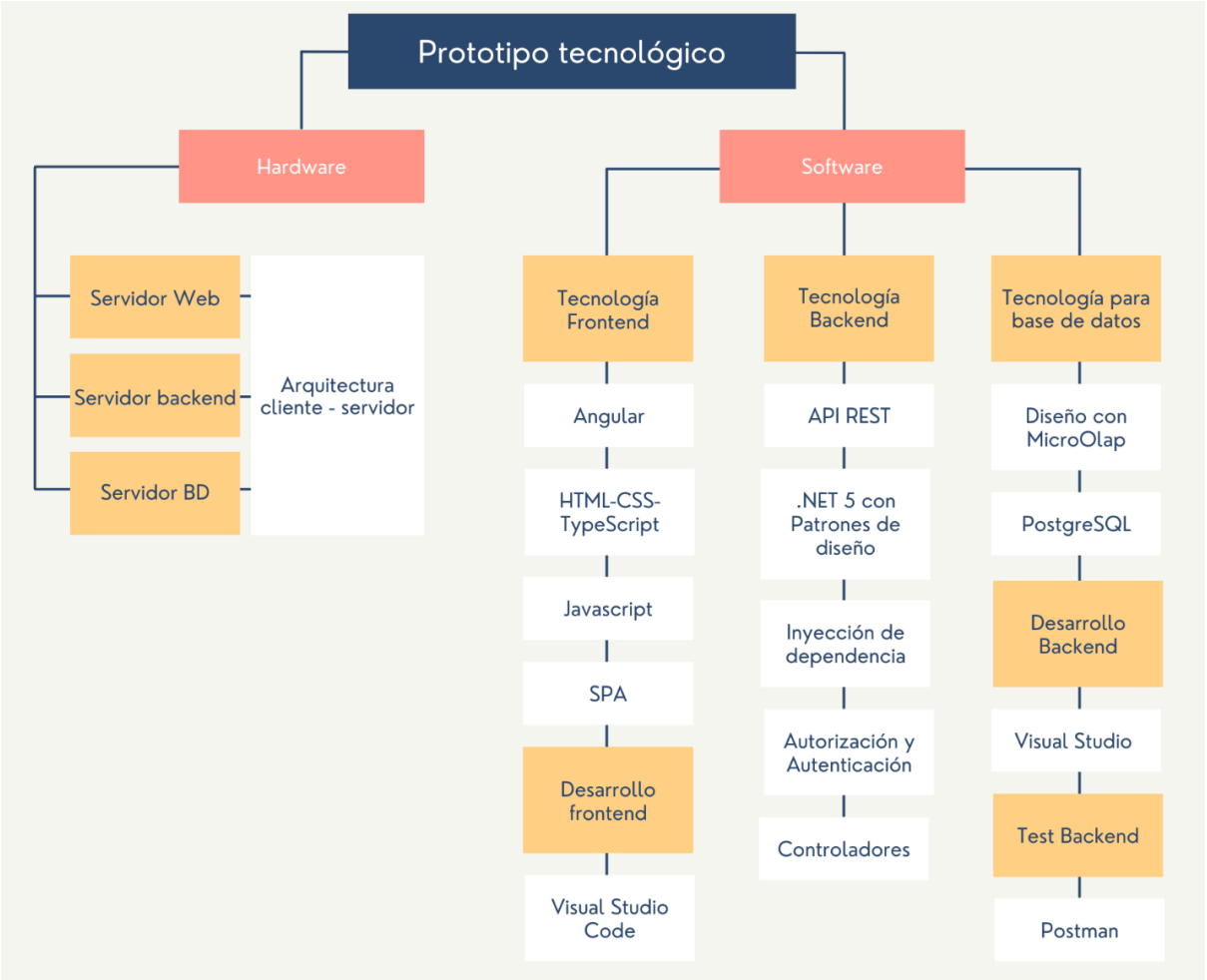


Figura 4: Organizador gráfico de tecnologías

En la figura 4 podemos ver el organizador gráfico que desglosa los conceptos usados para el desarrollo del proyecto y sus derivados.

2.2.1 Arquitectura cliente-servidor.

Es una de las arquitecturas más usadas para el desarrollo de aplicaciones informáticas en la actualidad, ya que separa la interacción del cliente con la base de datos, a más que se permite establecer reglas para el acceso a ellos de una manera controlada y segura [13].

Como su nombre lo indica se trata de establecer 2 actores principales que actúen entre sí de manera independiente; estos son el cliente que puede ser una aplicación web, móvil, una terminal de comandos u otros; y en el otro extremo se encuentra el servidor que responde a las solicitudes de un cliente dependiendo el tipo de requerimiento que le soliciten y la configuración interna del mismo el servidor puede responder a una solicitud a manera de un texto en HTML, XML, Json, formato imagen, mp3, video, etcétera [14].

Esto se puede resumir de la siguiente manera:

Cliente: Hace solicitudes.

Servidor: Responde a las solicitudes.

La principal utilidad en esta arquitectura es el hecho de centralizar las operaciones en el servidor, es decir si un cliente pide los datos de un usuario y estos se encuentran almacenados en una base de datos, es el servidor quien se encarga de comunicarse con el almacén de datos para extraer la información fidedigna que se solicita y transformarla en un formato de lectura válido para el cliente, dicho que sea capaz de interpretar de manera correcta, de esta manera todas las credenciales de acceso a los diferentes almacenes de datos que pueden existir son manejadas solo por el servidor de tal manera que el cliente no conoce el trabajo que se realiza por detrás de sus solicitudes[15].

Otra ventaja de este modelo es la escalabilidad que maneja el servidor, dado que todas las peticiones se manejan de manera centralizada el servidor está pensado para poder administrar las solicitudes de varios clientes en simultáneo [8], dicho servidor debe ser capaz de reconocer a todos los clientes que realizan la solicitudes y responderles a cada uno de manera independiente según el formato de lectura adecuado; por ejemplo si un cliente web pide la información de un usuario, este le responderá en formato Json; si un cliente móvil pide la misma información, este le responderá en formato XML, y así para los diferentes tipos de clientes que realicen solicitudes.

2.2.2 Tecnologías backend

2.2.2.1 .NET 5.0

“.NET le permite usar funcionalidades específicas de la plataforma, como las API del sistema operativo”.[16]

Lanzado en noviembre de 2020 es framework que ofrece Microsoft para el desarrollo de aplicaciones multiplataforma, integra todas las funcionalidades y librerías de C#, F# VB, así como todas las herramientas del ecosistema Microsoft, usa licencias de MIT y Apache2.

Entity Framework Core

Entity Framework (EF) Core es una extensión en código abierto y multiplataforma que puede servir como relacional de objetos (O/RM), que a su vez admite funciones .NET con una base de datos para usar objetos del entorno.

Este framework descarta la necesidad de escribir código complejo para el acceso a datos y se puede usar con diferentes gestores. De esta manera se puede acceder y manipular diferentes bases de datos usando el mismo código, pero con diferente cadena de conexión.

Microsoft Aspnetcore Authentication Jwtbearer

“La autenticación es el proceso de determinar la identidad de un usuario. La autorización es el proceso de determinar si un usuario tiene acceso a un recurso.” [17]

En un escenario de autenticación, una vez que un usuario ha iniciado sesión, cada solicitud posterior incluye un JWT, que se puede usar para verificar la identidad del usuario y el acceso a los métodos del controlador[18].

Esta librería se usa para autenticar un usuario, generalmente se la declara en el startup de la aplicación o en los constructores de clases que se desea usar. Para verificar el estado de un token en un controlador se puede usar la palabra reservada [Authorization] sobre los métodos que se vaya a pedir esta información con el propósito de permitir el acceso a esos puntos de la API sólo cuando la petición contenga un token válido.

Linq

LINQ es un conjunto de funciones de lenguaje y API que se puede usar para escribir varias consultas de manera unificada. Utiliza expresiones como argumento en la mayoría de sus métodos y altamente extensible con librerías externas, lo que permite construir métodos de consulta asíncronos cuando se requiere, esta librería almacena y recupera datos de diferentes fuentes de datos, como colección, diccionarios de datos o tuplas [19].

AutoMapper

Es una librería pequeña pero muy útil, ya que nos ahorra el trabajo de implementar código para mapear objetos, a más que se pueden aplicar diferentes reglas de mapeo para asegurar la calidad de la información. Los métodos y referencias a objetos se manejan de manera estandarizada, sin embargo no es un problema realizar todo tipo de ajustes en los mapeos ya que la librería implementa un stack de funciones y propiedades para resolverlo de manera sencilla [6].

2.2.2.2 Controllers

Los controllers o controladores son la parte fundamental en la construcción de una API, ya que es el punto de llegada para todas las peticiones REST que se implementan. El controlador se usa para definir y agregar una colección de acciones que proporcionan un vínculo con la solución lógica establecida en el backend, por ejemplo, enrutamiento, almacenamiento en caché, validación, autorización, mapeo de objetos entre otros.

Una buena estructura de programación nos indica que la lógica del backend no se debe describir en el método de acción del controlador, es mejor establecer subcapas que se encarguen de procesar la información y devolver los datos listos para enviar la respuesta de cada solicitud [20].

2.2.2.3 Inyección de dependencia

La inyección de dependencia es un patrón de diseño que se implementa generalmente en los constructores de clases para activar las funcionalidades de un servicio o contrato previamente establecido. Su función radica en proporcionar las dependencias que se producen entre objetos de alguna forma desde el exterior sin necesidad de escribirlas directamente en el código de cada clase.

2.2.2.4 Microsoft Aspnetcore Authorization

“La autorización se refiere al proceso que determina lo que un usuario puede hacer. Por ejemplo, un usuario administrativo puede crear una biblioteca de documentos,

agregar documentos, editar documentos y eliminarlos. Un usuario no administrativo que trabaje con la biblioteca sólo está autorizado para leer los documentos." [21]

Es una librería que forma parte extendida del ecosistema ASP NET CORE, esta nos ayuda a especificar reglas de negocio para el acceso a la información de tal manera que una acción del controlador puede responder las solicitudes de diferentes maneras, ya se denegando el servicio o mapeando los datos según el nivel de acceso que tenga el usuario en sesión [22].

JWT Bearer

Es un estándar abierto basado en JSON, *"sirve para la creación de tokens de acceso que permiten la propagación de identidad y privilegios".* [23]

Ayuda a la transmisión de información, con una estructura divida en tres partes: Header, payload y signatura. La información presentada es verificada y confiable ya que el algoritmo para generar dicho token y los métodos de la librería aseguran la calidad y robustez del mismo, es por ello que la tecnología JWT Bearer se utiliza para transmitir información cifrada de extremo a extremo.

La información proporcionada se debe enviar al servidor en cada petición, de esta manera el backend comprobará si es un token válido para dicha solicitud.

2.2.2.5 Microsoft Aspnetcore Authentication

Es una librería del ecosistema ASP NET CORE, su interfaz IAuthenticationService es el punto de entrada principal, este sigue un proceso dividido en cinco métodos u acciones de autenticación. El primero se basa en obtener datos de autenticación para una solicitud; también contiene un método para declarar solicitudes no autenticadas que no requieren autenticación, un método para prohibir o denegar una solicitud autenticada y los métodos para iniciar y cerrar sesión. *"En un alto nivel, la autenticación se basa en el concepto de esquemas, que se asignan a instancias específicas de IAuthenticationHandler que son responsables de manejar cada una de las acciones de autenticación".*[24]

La práctica de autenticación es adoptada por varias organizaciones tanto privadas como públicas, dado que simplifica la escalabilidad de sus funciones en cuanto al acceso sencillo y seguro que se implementa [25].

2.2.2.6 Patrones de diseño

Los patrones de diseño son colecciones de conocimientos de diseño probadas y estandarizadas que aplican indiferentemente del framework o lenguaje de programación que se utiliza. Es una solución probada por varios desarrolladores e implementada por una gran comunidad de personas que modelan su arquitectura basada en estándares de código limpio. Hay estructuras y comportamientos que a menudo aparecen en todos los desarrollos de software, es donde un patrón de diseño puede mitigar ese error y ayudar a la mantenibilidad, escalabilidad y elasticidad de un requerimiento implementado [26].

Patrón singleton

Singleton es uno de los patrones de diseño de clases en la programación orientada a objetos, y está diseñado para que la instancia de la clase sea siempre única en tiempo de ejecución [27].

Patrón transient

Transient, lo podemos interpretar como transitorio o fugaz, es una nueva instancia clase normal y corriente. Generamos un objeto que se crea, se utiliza, y finalmente se destruye. La duración de transient es igual a la duración de la ejecución del objeto en sí; suele tener un ciclo de vida corto, aunque este criterio está sujeto a la complejidad del proceso y de otras características.

Patrón scoped

Este patrón crea una instancia del objeto cuando se realiza una petición al servidor, mientras el ciclo de vida de la petición continúe existiendo, también lo hará la instancia del objeto. En este caso cada petición crea una nueva instancia en todos los casos.

2.2.2.7 API RESTFUL

La transferencia de estado representacional (REST) o los servicios web RESTFUL es la manera de dar interoperabilidad entre sistemas informáticos conectados a Internet por medio de métodos POST, PUT, DELETE, PATCH, GET principalmente [28].

Los servicios web que son compatibles con “REST”, dan paso a que los sistemas solicitantes puedan acceder y manipular representaciones textuales de recursos web utilizando un conjunto uniforme y predefinido de operaciones.

En otras palabras, una API RESTFUL, es un estilo de arquitectura para una interfaz de una aplicación “API”, la cual utiliza peticiones HTTP para lograr acceder y utilizar datos.

Post

“Envía información al servidor para que este la procese”. [29] Se especifican los parámetros en el cuerpo de la petición, gracias a este método es posible crear un nuevo recurso.

Put

“Envía un recurso al servidor”. [29] Esta petición sirve para editar un recurso, es decir, envía datos a uno que ya existe para su procesamiento

Delete

“Envía la petición al servidor de eliminar el recurso especificado”. [29] Por lo general la práctica de borrar un registro existente en una base de datos relacional se debe atender con la mayor precaución del caso, dado que, si otros registros son dependientes del mismo, se ocasiona un borrado en cascada lo que conlleva a la

pérdida de información. Si el borrado en cascada no está activado el servidor enviará un código de estado de error.

Get

"Solicita información o recurso al servidor, puede incluir parámetros en la url" [29]. En pocas palabras, la petición GET se utiliza para recuperar un recurso, a este se le puede agregar parámetros cuando se requiera.

2.2.3 Tecnologías frontend

2.2.3.1 HTML

HTML es una de las bases para la construcción de frontend, se lo conoce como lenguaje de hipertexto y es interpretado por todos los navegadores web con el fin de presentar de manera anidada el texto plano, permitiendo agregar condiciones y nombres de referencia a cada etiqueta de hipertexto utilizada [30].

2.2.3.2 CSS

CSS es un acrónimo de "Hojas de estilo en cascada", este lenguaje cumple la función de agregar estilo a los diseños estructurados en documentos creados con HTML. Esto sirve para mejorar la calidad de presentación web e implementar buenas prácticas de usabilidad [30].

2.2.3.3 Typescript

TypeScript es un lenguaje de programación desarrollado y proporcionado por Microsoft. Apareció en 2012 y es pensando para crear clases de JavaScript fuertemente tipadas aplicando modelos orientados a objetos y permitiendo el uso de herencia, interfaces, clases y más. De fondo el compilador se encarga de convertir este código en JavaScript para poder ser entendido por todos los entornos web [31].

2.2.3.4 JavaScript

JavaScript es un lenguaje de programación que permite implementar funciones u eventos a los sitios webs. generalmente se ejecuta en el navegador del lado del cliente. Usando JavaScript, además de manejar eventos, también es posible mostrar mapas y animaciones gráficas [32].

2.2.3.5 SPA

Single-Page Application (SPA) es un paradigma de arquitectura de aplicaciones web cuyo propósito es llevar el poder de las aplicaciones de escritorio al entorno delgado y multiplataforma de un navegador web. SPA ofrece una experiencia de usuario similar al de una aplicación nativa, mejorando sustancialmente el rendimiento de aplicaciones web mediante técnicas de diseño de cargue parcial de páginas, que además son compatibles tanto para PC como para dispositivos móviles [33], [34].

SPA es una tecnología muy potente para realizar web apps rápidas, fluidas y de éxito. Si la utilizamos para una web app que no necesita ser indexada, va perfecto. Si, por el

contrario, vamos a crear un producto, servicio o web que sí necesita estar indexada, la cosa se complica, aunque no es del todo imposible hacerla con SPA.

2.2.4 Base de datos (SGBD)

Una base de datos es un almacén de toda la información creada por un usuario o por el propio sistema, estos pueden ser relaciones o NoSQL [35].

2.2.4.1 PostgreSQL

Es un sistema de gestión de bases de datos relacional de objetos elaborado con código abierto. Soporta gran parte del estándar SQL y ofrece muchas características modernas [36].

Maneja altos volúmenes de información y permite administrar varias bases de datos. Aporta mucha flexibilidad a los diversos proyectos implementados en este gestor, además de ser multiplataforma, es decir, se puede utilizar en sistemas operativos como MAC OS X, Windows, Ubuntu, Debian, entre otros.

2.2.5 Entorno de desarrollo

2.2.5.1 Visual Studio

Es un conjunto completo de herramientas de desarrollo de Microsoft, que incluye la mayoría de las herramientas necesarias en todo el ciclo de vida del software, se usa principalmente para escribir código, testear, modelar, ejecutar y crear los compilados necesarios para la puesta en producción [37].

2.2.5.2 Visual Studio Code

Es un editor de código ligero, multiplataforma, gratuito, de código abierto y de alto rendimiento. Permite la instalación de extensiones para mejorar su rendimiento y funcionalidad dependiendo el lenguaje de programación con que se desea trabajar.

2.2.6 Entorno para pruebas

2.2.6.1 Postman

“Postman es una plataforma para probar, documentar y usar APIs. Postman simplifica cada paso del ciclo de vida de la API y agiliza la colaboración para realizar un backend API más rápido, seguro y confiable”. [38]

Usamos esta herramienta para comprobar el correcto funcionamiento de la implementación punto a punto realizada en el backend de la aplicación.

2.2.7 Metodología de desarrollo de software

Una metodología para el desarrollo de software está centrada en proporcionar una guía de implementación que puede incluir normas, métodos de trabajo, reuniones, evaluaciones de resultados y más. Escoger una metodología específica va a depender de los requerimientos del proyecto y la apreciación de la persona encargada del éxito del mismo [39].

2.2.7.1 Metodología ágil

Esta metodología está centrada en cumplir los objetivos del proyecto de manera rápida y eficiente, dejando a un lado la necesidad de una documentación rigurosa y el seguimiento de métricas específicas para alcanzar el éxito de un proyecto [40], [41].

2.2.7.2 Scrum

Scrum es una metodología ágil que se enfoca en el cumplimiento de los objetivos del negocio más que en una rigurosa documentación, estableciendo entregables fijos en plazos de 2 a 4 semanas con una retroalimentación sobre el desarrollo realizado y asignando nuevas tareas a cumplir en otro lapso de 2 a 4 semanas; este proceso se lo conoce como SPRINT y se realiza las veces que sea necesario hasta culminar el proyecto [42].

Para empezar con scrum el trabajo se divide entre 3 actores que son: El project owner, scrum master y team developer.

El project owner se encarga de recopilar los requerimientos del sistema y realiza un informe detallado sobre las funcionalidades del sistema basándose en las historias de los usuarios; a esto se conoce como product log.

El scrum master se encarga de revisar el product log para realizar un documento llamado backlog con el fin de asignar tareas al team developer, esto se conoce como sprint planning y debe tener una duración máxima de 8 horas.

El team developer debe desarrollar las tareas asignadas por el scrum master en lapsos de 2 a 4 semanas, esto se conoce como sprint, al final de cada sprint todo el equipo se reúne para discutir avances, estancamientos y temas de optimización, a esto se le conoce como sprint review y debe tener una duración máxima de 4 horas.

Todos los integrantes del equipo scrum deben realizar reuniones diarias para discutir temas relacionados con el desarrollo que se encuentran realizando y despejar dudas; se recomienda que dicha reunión tenga una duración máxima de 15 minutos.

Product backlog

Es una pila de todo el trabajo que se debe realizar para terminar el producto. Priorizando cada elemento que tiene más importancia en cada etapa.

Cada ítem debe agregar valor para el cliente y ordenados según criterios de prioridad.

No debe contener tareas de bajo nivel; debe contener funcionalidades, bugs e historias de usuario, tareas técnicas y trabajo de investigación.

Historias de usuarios

Son descripciones cortas y simples desde una perspectiva del usuario o cliente del sistema.

En la figura 3 podemos identificar la estructura propuesta por scrum para redactar una historia: Como usuario WEB quiero tal “OBJETIVO” para tal “MOTIVO”.

TITULO DE HISTORIA	
Rol o Persona:	**"Persona o rol de usuario con la necesidad"**
Quisiera:	*"<Accion u objetivo>"*
Para:	*"<Motivo, razon, valor>"*

Figura 5: Historia de usuario scrum

En la figura 5 apreciamos la estructura o características que tiene una historia de usuario, la cual se divide de la siguiente manera:

Independientes. - Para poder llevarlas a cabo en el orden que más nos convenga según las prioridades establecidas por el producto owner.

Negociables. - Se deben poder negociar con el product owner a fin de permitir el establecimiento de límites.

Valor para el usuario. - El “para” es fundamental siempre, de esta manera se va a entender la funcionalidad, esto es algo que debe tener claro todo el equipo de desarrollo.

Estimable. - El equipo de desarrollo debe ser vaya a realizar dicha tarea, debe ser capaz de estimar el esfuerzo que va a invertirse para realizarla.

Small. - Debe ser de un tamaño que el equipo de desarrollo pueda asumir en un sprint.

Testable. - Para asegurarnos que la tarea se ha implementado de forma correcta.

Sprint backlog

Lista de elementos en los que se debe trabajar durante los sprint, se trata de tareas técnicas pequeñas que permiten conseguir un incremento de software hasta terminarlo.

Permite visualizar aquellos elementos que aún no se han realizado, los ya terminados y quien se encarga de cada uno de ellos; basado en dicho concepto este documento se encarga de plasmar los elementos que se han terminado, elementos en proceso, elementos que aún no se empiezan y personas responsables de cada uno de ellos.

Este documento permite entender cuál es la evolución del trabajo durante el sprint y como hacer un análisis de riesgos.

2.2.8 Herramientas case

2.2.8.1 MicroOlap

MicroOLAP Database Designer es un modelador de base de datos práctica, proporciona una interfaz para el diseño, de un modelo relacional.

Permite gestionar usuarios, esquemas, vistas, relaciones, reglas de negocio, procedimientos almacenados, ingeniería inversa y generación de código SQL.

2.3 OBJETIVOS DEL PROTOTIPO

2.3.1 Objetivo general

Implementar un sistema web aplicando metodología scrum con angular, .Net 6 y PostgreSQL para la optimización de atención al cliente y procesos logísticos en una empresa Courier con E-Commerce.

2.3.2 Objetivos específicos

- Diseñar una arquitectura de tres capas con una API RESTFUL, una página web SPA y una base de datos relacional haciendo uso del framework Angular, lenguaje de programación .NET 5.0 y base de datos PostgreSQL.

- Implementar los patrones transient y scoped en los servicios de la aplicación, para el establecimiento de contratos e inyección de dependencia en el desarrollo del proyecto.

- Desarrollar una aplicación web haciendo uso de la metodología ágil scrum para conseguir una solución escalable y tecnologías multiplataforma para su despliegue en un servidor CentOS.

- Asegurar las credenciales de autorización mediante un token JWT Bearer para el acceso a la información según los permisos de usuario y políticas del negocio implementadas.

- Evaluar las funcionalidades del sistema para la estimación de satisfacción que cumple el proyecto para las mejoras de los procesos logísticos del negocio y la atención al cliente.

2.4 DISEÑO DEL PROTOTIPO

Para direccionar el desarrollo de la aplicación web, diseño de interfaces y despliegue del proyecto se aplicó la metodología SCRUM, que consta de 3 fases principalmente.

En la primera fase se recopila los requerimientos del usuario a través de la redacción de historias que describen requisitos funcionales del software.

La segunda fase establece un trabajo entre las historias de usuario y la experiencia del scrum master para crear una lista de tareas a realizar, se describen todas las funciones que va a tener el producto final.

La tercera y última fase se relaciona con la distribución del trabajo que realiza el equipo de desarrollo, se organiza un itinerario por sprints, que culmina en la revisión del mismo y una retroalimentación del trabajo que se realiza.

La cantidad de sprints necesarios para la culminación del proyecto depende del tamaño de la solución.

2.4.1 Definición de stakeholders

En una primera instancia el administrador de la empresa es la parte más interesada en la aplicación, ya que la utilidad y el éxito de la misma va a influenciar directamente en el control de su negocio y finanzas, él es quien mejor conoce el negocio por lo que se lo evalúa como la persona con más poder de conocimiento para el desarrollar una solución exitosa a los requerimientos del sistema.

También se incluye al área de atención al cliente y agentes de logística en bodega ya que se intenta aliviar la carga de trabajo que estos ejercen para las operaciones del negocio.

Tenemos al scrum master posicionado como otra de las personas que ejercen gran aporte en el éxito del proyecto, seguido del project owner y el team developer.

Figura 6: Matriz de interesados

La figura 6 plasma una matriz de stakeholders, en la cual podemos ver los actores involucrados directa e indirectamente en el desarrollo de la aplicación, estableciendo ejes de poder e interés en el desarrollo del proyecto siendo el centro lo más bajo y los extremos lo más alto.

2.4.2 Análisis del product owner

El e-commerce y las operaciones de importación son cada día más frecuentes en el entorno actual de Ecuador, llegado a una edad moderna donde instrumentos como internet, conectividad web, aplicaciones móviles y smartphones son de uso habitual es natural pensar que los negocios busquen migrar sus servicios al mundo digital; es por ello que la construcción de esta aplicación a más de automatizar las tareas del negocio simboliza prestigio para la empresa que se sirve de utilizar herramientas y tecnologías modernas dando una mejor impresión a los clientes inspirando confianza, seriedad y una gestión transparente para sus usuarios.

2.4.2.1 Historias de usuarios

A continuación, se detalla las historias de los usuarios para posteriormente realizar el product log y backlog según la metodología scrum.

Tabla 1: Historia de usuario - Políticas de seguridad en la información

Políticas de seguridad en la información	
Rol o Persona:	**Administrador**
Quisiera:	*"Perfilar los usuarios del sistema"*
Para:	*"Asignar niveles de acceso a la información"*

La historia de usuario descrita en la tabla 1 hace referencia a las políticas de seguridad en la información, misma que designa la funcionalidad de poder establecer diferentes niveles de acceso.

Tabla 2: Historia de usuario - Registro de clientes

Registro de clientes	
Rol o Persona:	**Administrador**
Quisiera:	*"Crear un registro de clientes"*
Para:	*"Dar seguimiento a sus compras, pedidos y transacciones"*

La historia de usuario descrita en la tabla 2 hace referencia al registro de clientes, mismos a los que posteriormente permite vincular diferentes compras, pedidos y transacciones que se realicen.

Tabla 3: Historia de usuario - Información de clientes

Información de clientes	
Rol o Persona:	**Atención al cliente**
Quisiera:	*"Que el registro de clientes almacene información como nombre, cédula, teléfono, email, Instagram"*
Para:	*"Facilitar el contacto con cada uno de ellos cada vez que sea necesario"*

La historia de usuario descrita en la tabla 3 hace referencia a la información de clientes que se registra en el sistema.

Tabla 4: Historia de usuario - Registro de empleados

Registro de empleados	
Rol o Persona:	**Administrador**
Quisiera:	*"Tener un registro de empleados"*

Para:	*" Poder saber quién está operando el sistema cuando se realiza una transacción"*

La historia de usuario descrita en la tabla 4 hace referencia al registro de empleados, mismos a los que posteriormente se encargará de registrar y revisar información acerca de la logística de envíos y sus clientes.

Tabla 5: Historia de usuario - Asignación de empleado a una agencia

Asignación de empleado a un agencia	
Rol o Persona:	**Administrador**
Quisiera:	*" Que cada empleado conste como trabajador de una sola agencia"*
Para:	*"Poder aplicar un filtro de movimientos realizados por empleado y ordenarlos por agencia"*

La historia de usuario descrita en la tabla 5 hace referencia a la información de empleados que se registra en el sistema.

Tabla 6: Historia de usuario - Compras de clientes

Compras de clientes	
Rol o Persona:	**Administrador**
Quisiera:	*" Crear un módulo que almacene las compras del cliente junto con los productos de esa compra"*
Para:	*"Tener una clasificación de compras por cliente"*

La historia de usuario descrita en la tabla 6 hace referencia a la información de compras que se registra en el sistema, tales que pertenecen a un cliente en particular.

Tabla 7: Historia de usuario - Llegada de encomiendas al casillero postal

Llegada de encomiendas al casillero postal	
Rol o Persona:	**Administrador**
Quisiera:	*Crear un módulo que registra todas las encomiendas que se reciben el en casillero postal y su contenido*
Para:	*"Poder vincular dicha encomienda al cliente que corresponda"*

La historia de usuario descrita en la tabla 7 hace referencia a la información que se registra en el casillero postal de cada cliente.

Tabla 8: Historia de usuario - Información clave de cada producto

Información clave de cada producto	
Rol o Persona:	**Administrador**
Quisiera:	*"Asignar un código único a cada producto recibido en el casillero postal"*
Para:	*“Escanear dicho código cuando se requiera información del producto o asignarlo en una guía de envió"*

La historia de usuario descrita en la tabla 8 hace referencia a una función que crea un código único para cada producto que se registra en el sistema.

Tabla 9: Historia de usuario - Información peso de cada producto

Información peso de cada producto	
Rol o Persona:	**Administrador**
Quisiera:	*"Asignar el peso a cada producto recibido en el casillero postal"*
Para:	*"Sumar los pesos de todos los productos asignados a una guía con el fin de calcular el cobro"*

La historia de usuario descrita en la tabla 9 hace referencia a un campo clave de cada producto, el peso del mismo que ayuda a realizar una estimación del cobro al cliente por el envío.

Tabla 10: Historia de usuario - Información de encomiendas recibidas

Información de encomiendas recibidas	
Rol o Persona:	**Administrador**
Quisiera:	*"Asignar número de seguimiento, fecha de llegada, imagen de evidencia por cada encomienda recibida en el casillero"*
Para:	*"Almacenar un registro de consumos y filtrar por aquellos campos cuando se requiera"*

La historia de usuario descrita en la tabla 10 hace referencia a los campos claves que debe tener cada encomienda recibida en una agencia.

Tabla 11: Historia de usuario - Información de compras

Información de compras	
Rol o Persona:	**Administrador**
Quisiera:	*" Agregar imagen de evidencia, fecha de compra y número de seguimiento a cada compra realizada"*

Para:	*"Poder revisar dicha información cuando se requiera"*

La historia de usuario descrita en la tabla 11 hace referencia a los campos claves que debe tener cada compra realizada.

Tabla 12: Historia de usuario - Movimientos de débito

Movimientos de débito	
Rol o Persona:	**Administrador**
Quisiera:	*"Registrar un movimiento de débito a la cuenta del cliente cada vez que se realiza un cobro o reembolso"*
Para:	*"Obtener un balance de cuenta del cliente y conocer sus saldos"*

La historia de usuario descrita en la tabla 12 hace referencia al registro de débitos en la cuenta del cliente, con el fin de poder tener un balance sobre su cuenta.

Tabla 13: Historia de usuario - Importar estados de cuenta

Importar estados de cuenta	
Rol o Persona:	**Administrador**
Quisiera:	*"Poder importar un estado de cuenta generado por banco Pichincha, Loja"*
Para:	*"Realizar una verificación de cada depósito que el cliente realizó, y que está verificación sea automática siempre que sea posible"*

La historia de usuario descrita en la tabla 13 hace referencia a implementar una función que permita importar el archivo generado por un banco que contiene los movimientos de cuenta.

Tabla 14: Historia de usuario - Guías de movilización

Guías de movilización	
Rol o Persona:	**Administrador**
Quisiera:	*"Un espacio para registrar guías de envió cuando se movilizan las encomiendas entre agencias"*
Para:	*"Poder tener un rastreo claro de donde se encuentra cada objeto"*

La historia de usuario descrita en la tabla 14 hace referencia al registro de guías generadas para la movilización de encomiendas entre agencias y los productos que contienen.

Tabla 15: Historia de usuario - Guías de despacho

Guías de despacho	
Rol o Persona:	**Atención al cliente**
Quisiera:	*" Crear órdenes de entrega con estado pendiente"*
Para:	*"El agente de bodega sepa que tiene una tarea pendiente "*

La historia de usuario descrita en la tabla 15 hace referencia al registro de guías generadas para la entrega final al cliente y los productos que contienen.

Tabla 16: Historia de usuario - Arquitectura 3 capas paso 1

Arquitectura 3 capas paso 1	
Rol o Persona:	**Scrum máster**
Quisiera:	*"Crear las conexiones entre el frontend y el backend de manera aislada a la base de datos"*
Para:	*"Separar la lógica del negocio de la capa de presentación"*

La historia de usuario descrita en la tabla 16 hace referencia a la implementación hardware que contiene el servidor web y el servidor backend.

Tabla 17: Historia de usuario - Arquitectura 3 capas paso 2

Arquitectura 3 capas paso 2	
Rol o Persona:	**Scrum máster**
Quisiera:	*"Crear la conexión entre el backend y el sistema gestor de base de datos correspondiente"*
Para:	*"Asegurar la conexión entre la capa de dominio y el acceso a datos aislando la capa de presentación"*

La historia de usuario descrita en la tabla 17 hace referencia a la implementación hardware que contiene la base de datos y el acceso a la información.

Tabla 18: Historia de usuario - Modelo relacional

Modelo relacional	
Rol o Persona:	**Scrum máster**

Quisiera:	*"Modelar la base de datos de la aplicación"*
Para:	*"Crear los esquemas y tablas correspondientes en el SGBD"*

La historia de usuario descrita en la tabla 18 hace referencia a la elaboración del modelo relacional que se implementa en la base de datos.

Tabla 19: Historia de usuario - Frontend

Frontend	
Rol o Persona:	**Scrum máster**
Quisiera:	*"Crear los formularios correspondientes a cada entidad"*
Para:	*"El envío de información al backend"*

La historia de usuario descrita en la tabla 19 hace referencia al desarrollo de formularios implementados en la interfaz para el envío de información a registrar en las tablas de cada entidad.

Tabla 20: Historia de usuario - Operaciones CRUD

Operaciones CRUD	
Rol o Persona:	**Scrum máster**
Quisiera:	*"Crear los repositorios con funcionalidades CRUD por cada tabla del sistema"*
Para:	*"Permitir la comunicación entre el servidor y la base de datos"*

La historia de usuario descrita en la tabla 20 hace referencia a los métodos CRUD que registran, solicitan, actualizan o borran información de cada entidad en las tablas de la base de datos.

Tabla 21: Historia de usuario - Deploy

Deploy	
Rol o Persona:	**Scrum máster**
Quisiera:	*“Crear los archivos reléase de la aplicación.”*
Para:	*"Subir al servidor web y backend"*

La historia de usuario descrita en la tabla 21 hace referencia a los compilados que se debe generar para el despliegue de la aplicación y puesta en producción.

Tabla 22: Historia de usuario - Login y logout

Login y Logout	
Rol o Persona:	**Administrador**
Quisiera:	*"Al momento de validar las credenciales de autenticación se debe generar un token de autorización"*
Para:	*"Poder realizar peticiones al servidor y que este responda según sus niveles de acceso"*

La historia de usuario descrita en la tabla 22 hace referencia al método de autenticación que maneja el sistema y la autorización basada en tokens.

Tabla 23: Historia de usuario - Creación de cuenta al cliente

Creación de cuenta al cliente	
Rol o Persona:	**Administrador**
Quisiera:	*"Al momento de realizar una guía de despacho se debe calcular el valor que adeuda el cliente y generar esta cuenta"*
Para:	*"Poder realizar el cobro y tener pendiente los valores generados"*

La historia de usuario descrita en la tabla 23 hace referencia a la función que crea una cuenta por cobrar del cliente cuando se genera un despacho de productos.

Tabla 24: Historia de usuario - Movimientos de débito y crédito por cliente

Movimientos de crédito y débito por cliente	
Rol o Persona:	**Administrador**
Quisiera:	*"Generar un listado de todos los pagos y deudas que ha generado un cliente en su trayectoria"*
Para:	*"Transparentar la información y visualizarla de manera ordenada"*

La historia de usuario descrita en la tabla 24 hace referencia a la lista de créditos y débitos que registra un cliente en su historial de cliente.

Tabla 25: Historia de usuario - Estado de guías de despacho

Estado de guías de despacho

Rol o Persona:	Agente de bodega
Quisiera:	*"Poder escanear los códigos de productos que se agregan en cada entrega"*
Para:	*"Que se descarguen los productos de la bodega y la orden de entrega cambie de estado a: COMPLETADO"*

La historia de usuario descrita en la tabla 25 hace referencia al estado de una guía que se debe despachar a un cliente. De tal manera que exista una función que descargue de bodega los productos entregados.

Tabla 26: Historia de usuario - Estado guías de movilización

Estado guías de movilización	
Rol o Persona:	**Administrador**
Quisiera:	*"Que el sistema indique las guías de movilización que ya se han receptado y las pendientes de recibir"*
Para:	*"Poder visualizar el trabajo que realiza cada agente e intermediario"*

La historia de usuario descrita en la tabla 26 hace referencia al estado de una guía que se moviliza de una agencia a otra. De tal manera que exista una función que descargue de una bodega los productos enviados y se carguen en la bodega que los recibe.

Tabla 27: Historia de usuario - Guías de despacho incluye productos

Guías de despacho incluye productos	
Rol o Persona:	**Administrador**
Quisiera:	*"Que todas las órdenes de entrega tengan los productos que se debe despachar en ellas"*
Para:	*"El agente de bodega se encargue de realizar dichos envíos"*

La historia de usuario descrita en la tabla 27 hace referencia a una interfaz que indique al agente de bodega los despachos que tenga pendientes de realizar.

2.4.2.1 Product backlog

Etapa 1: Modelado Entidad – Relación.

- Creación de las entidades del negocio.
- Establecer relaciones entre entidades.

- Establecer índices y operaciones de la base de datos.
- Generar código SQL para la creación de la base de datos.
- Ejecutar script para la creación del modelo entidad relación terminado.

Etapa 2: Creación del dominio de negocio

- Crear todas las clases de entidades de dominio descritas como tablas en la base de datos
- Creación de los modelos Dto. para mostrar data, actualizar e insertar por cada entidad de dominio.
- Establecer las reglas de mapeo para cada entidad y sus Dto. correspondiente.
- Crear los controladores para cada entidad y sus relaciones directas.
- Crear métodos de autenticación.
- Crear un método de autorización basado en tokens
- Crear los repositorios de cada entidad junto con sus interfaces o contratos.
- Crear un módulo para verificación automática de depósitos.
- Implementar patrón de diseño scoped para todos los repositorios que se conectan con los controladores de cada entidad respectivamente.
- Implementar patrón transient para los mapeadores de campos en cada entidad.

Etapa 3: Creación de la capa de acceso a los datos.

- Instalar Entity Framework Core y todas sus dependencias
- Crear la clase context para el acceso a los datos e incluir todas las entidades de dominio para relacionarlas con cada una de las tablas existentes en la base de datos.
- Especificar mediante FLUENT API las reglas de relación y restricción que tiene cada entidad de dominio según el modelado de la base de datos.
- Establecer la conexión de la capa de acceso a datos con el SGBD.
- Testear el correcto funcionamiento de la capa de acceso a datos.

Etapa 4: Modelado de interfaz web

- Diseño de la interfaz para login

- Diseño de un panel de navegación
- Diseño de pantalla de inicio.
- Diseño de formulario para clientes (Crear y actualizar)
- Diseño de la interfaz para listado de clientes.
- Diseño de formulario para compras y sus productos (Crear y actualizar)
- Diseño de la interfaz para listado de compras
- Diseño de formulario para recibir encomiendas y su contenido en el casillero postal (Crear y actualizar)
- Diseño de la interfaz para listado de encomiendas recibidas en el casillero postal y sus productos
- Diseño de formulario para depósitos (Crear y actualizar)
- Diseño de la interfaz para listar depósitos.
- Diseño de formulario para reembolsos (Crear y actualizar)
- Diseño de la interfaz para listado de reembolsos.
- Diseño de formulario para Empleados (Crear y actualizar)
- Diseño de la interfaz para listado de empleados.
- Diseño de formulario para creación de guías aéreas (Crear y actualizar)
- Diseño de la interfaz para listado de guías aéreas.
- Diseño de formulario para despachos a clientes (Crear y actualizar)
- Diseño de la interfaz para listado de despachos a clientes.
- Diseño de formulario para subir estados de cuenta (Crear y actualizar)
- Diseño de la interfaz para listar estados de cuenta.
- Diseño de la interfaz para listar agencias.
- Diseño de la interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

Etapa 5: Desarrollo de la capa de presentación

Establecer una hoja de estilos CSS estandarizada para la creación de todos los componentes web que se van a usar en el desarrollo.

Desarrollo de la interfaz para login

- Envío de petición POST para la autenticación
- Responder con el token de autorización.
- Guardar el token para adjuntarlo en todas las solicitudes que realice el usuario durante la sesión.

Desarrollo de un panel de navegación

- Con diseño responsive

Desarrollo de pantalla de inicio.

- Contiene botones de acción rápida para agregar cliente, recibir encomienda en casillero, agregar un nuevo depósito de clientes o crear una nueva compra.
- Agregar un espacio para leer códigos de productos que muestre en un modal toda la información relacionada.
- Agregar un espacio para listar todos los clientes con filtros para búsqueda y acceso rápido.
- Crear un espacio que liste las guías aéreas más próximas pendientes de recibir y los envíos a clientes pendientes de despachar.

Desarrollo de formulario para clientes (Crear y actualizar)

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.
- Ocultar la contraseña.
- Agregar un espacio para añadir el permiso de usuario correspondiente.
- Al Cliente únicamente se le puede dar permiso para ingresar al sistema y ver sus transacciones.

Desarrollo de la interfaz para listado de clientes.

- Listar los clientes con paginación agregando un botón para visitar el perfil del cliente y otro para realizar modificaciones en sus datos personales y permisos.

Desarrollo de formulario para compras y sus productos (Crear y actualizar).

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen de compra y los campos que contiene el formulario de compras.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un débito por la compra realizada.

- Posterior a esto se permite insertar los productos de compra con una imagen y su información referente.

Desarrollo de la interfaz para listado de compras y sus productos.

- Se lista todas las compras o las compras de un cliente específico.

- Cuando se listan las compras de todos los clientes, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 3 botones; uno para ver los detalles de la compra, otro para agregar uno o varios números de tracking y otro para agregar más productos a dicha compra.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Desarrollo de formulario para recibir encomiendas en el casillero postal y sus productos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen de la encomienda recibida y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un débito por algún servicio cobrado.

- Posterior a esto se permite insertar los productos que contiene dicha encomienda con una imagen y su información referente.

- Cada Producto debe tener una imagen, peso, seleccionar si es una compra del stock o una encomienda recibida solo por servicio de casillero, la ubicación donde se recibe el artículo y una descripción opcional.

Desarrollo de la interfaz para listado de encomiendas recibidas en el casillero postal y sus productos

- Se lista todas las encomiendas recibidas en el casillero, o todas las de un cliente específico.

- Cuando se listan las encomiendas recibidas de todos los clientes, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para ver los detalles del paquete y otro para agregar productos en la encomienda recibida.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Desarrollo de formulario para depósitos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen del depósito y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un crédito por el valor correspondiente al depósito luego de ser validado.

Desarrollo de la interfaz para listado de depósitos.

- Se lista todos los depósitos registrados, o los de un cliente específico.

- Cuando se listan todos los depósitos registrados, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para validar que el depósito se encuentra en el reporte del banco y otro para marcar el depósito como no encontrado en caso de que no exista.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

- Cada depósito debe pasar por una verificación antes de agregarlo como un saldo válido que el cliente puede usar; esta verificación puede terminar catalogando al depósito como válido o no encontrado.

Desarrollo de formulario para reembolsos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen del reembolso emitido y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla “Bills” del cliente generando un débito por el valor correspondiente al reembolso emitido.

Desarrollo de la interfaz para listado de reembolsos.

- Se lista todos los reembolsos registrados, o los de un cliente específico con paginación.

- Cuando se listan todos los reembolsos registrados, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para validar que el reembolso se realizó y otro para ver el detalle del reembolso.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Desarrollo de formulario para Empleados (Crear y actualizar)

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Ocultar la contraseña.

- Agregar un espacio para añadir el permiso de usuario correspondiente.

- Al Cliente únicamente se le puede dar permiso para ingresar al sistema y ver sus transacciones.

Desarrollo de la interfaz para listado de empleados.

- Listar los empleados con paginación.

- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para creación de guías aéreas (Crear y actualizar)

- Se debe realizar un formulario con los atributos de su entidad.

- Para agregar productos a una guía aérea se debe incluir un campo para escanear el código de cada producto.

- El programa debe analizar si el producto no ha sido agregado en ninguna otra guía.

- Se debe tener la opción de subir una imagen general de la guía aérea.

Desarrollo de la interfaz para listado de guías aéreas.

- Se debe listar todas las guías con opciones de paginación.

- Debe contener información que identifique el estado de la guía, y el contenido en ella.

- Cada registro debe contener 2 botones; uno para ver la información de dicha guía, y otro para recibir la guía con sus productos cuando ésta llega a su destino.

- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para despachos a clientes (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una dirección de envío, si no existe debe permitir crearla.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla “Bills” del cliente generando un débito por el valor correspondiente al valor del envío generado.

Desarrollo de la interfaz para listado de despachos a clientes.

- Se debe listar todas las guías con opciones de paginación.

- Debe contener información que identifique el estado de la guía, y el contenido en ella, el peso y la información de envío.

- Cada registro debe contener 2 botones; uno para ver la información de dicha guía, y otro marcar como enviado cada producto cuando se realiza el despacho.

- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para subir estados de cuenta (Crear)

- Se debe cargar una lista de todas las cuentas bancarias que se han creado.

- Se escoge subir el archivo generado por el banco a la cuenta bancaria que le corresponde.

- Se carga una vista previa de los movimientos bancarios que se van a subir y posteriormente se cargan los datos del archivo al servidor.

Desarrollo de la interfaz para listar estados de cuenta.

- Se debe cargar una lista de todas las cuentas bancarias que se han creado.

- Se escoge una cuenta bancaria para cargar el estado de cuenta subido hasta la fecha.

- Cada registro contiene un campo que indica si dicho movimiento se ha vinculado a algún depósito ingresado por el cliente.

- No se permite borrar ni editar nada de este registro.

Desarrollo de la interfaz para listar agencias.

- Se lista cada Agencia con el contenido de sus bodegas.

- Se permite ver el contenido de sus bodegas a detalle por cliente.

Desarrollo de la interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

- Este panel de información debe contener las acreditaciones y débitos que ha tenido un usuario a lo largo de su trayectoria.

- Cada registro debe contener un botón que vincule dicho registro con un depósito, envió, o compra según corresponda.

Etapa 6: Conexión de la capa de presentación con el dominio del negocio.

- Desarrollar los servicios de conexión http para realizar las peticiones a la API según corresponda.

Etapa 7: Test

- Test de autenticación y autorización

- Test de manipulación en el servicio de clientes

- Test de manipulación en el servicio para compras y sus productos (Crear y actualizar)

- Test de manipulación en el servicio para recibir encomiendas en el casillero postal y sus productos (Crear y actualizar)

- Test de manipulación en el servicio de depósitos (Crear y actualizar)

- Test de manipulación en el servicio de reembolsos (Crear y actualizar)

- Test de manipulación en el servicio de Empleados (Crear y actualizar)

- Test de manipulación en el servicio de guías aéreas (Crear y actualizar)

- Test de manipulación en el servicio de despachos a clientes (Crear y actualizar)

- Test de manipulación en el servicio para subir estados de cuenta

Etapa 8: Producción

Desplegar la aplicación web, el backend y la base de datos listo en producción.

2.4.3 Planeación del sprint

En esta sección se describe el orden a seguir para el desarrollo de la solución a través de un documento que realiza el scrum master con ayuda del team developer, contiene todas las interacciones que se van a realizar para culminar un entregable.

2.4.3.1 Sprint backlog

Según lo determinado por el team developer y scrum master, en base a las capacidades del equipo, se ha dividido el trabajo en 11 sprints que juntos dan solución al desarrollo completo de la aplicación puesta en producción.

Primer sprint

Diseño de la interfaz para login

Diseño de un panel de navegación

Diseño de pantalla de inicio.

Diseño de formulario para clientes (Crear y actualizar)

Diseño de la interfaz para listado de clientes.

Diseño de formulario para compras y sus productos (Crear y actualizar)

Agregar productos

Diseño de la interfaz para listado de compras

Diseño de formulario para recibir encomiendas y su contenido en el casillero postal (Crear y actualizar)

Agregar productos:

Diseño de la interfaz para listado de encomiendas recibidas en el casillero postal y sus productos

Creación de formulario para depósitos (Crear y actualizar)

Diseño de la interfaz para listar depósitos.

Diseño de formulario para reembolsos (Crear y actualizar)

Diseño de la interfaz para listado de reembolsos.

Diseño de formulario para Empleados (Crear y actualizar)

Diseño de la interfaz para listado de empleados.

Diseño de formulario para creación de guías aéreas (Crear y actualizar)

Diseño de la interfaz para listado de guías aéreas.

Diseño de formulario para despachos a clientes (Crear y actualizar)

Diseño de la interfaz para listado de despachos a clientes.

Diseño de formulario para subir estados de cuenta (Crear y actualizar)

Diseño de la interfaz para listar estados de cuenta.

Diseño de la interfaz para listar agencias.

Diseño de la interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

Segundo sprint

Creación de las entidades del negocio.

Establecer relaciones entre entidades.

Establecer índices y operaciones de la base de datos.

Generar código SQL para la creación de la base de datos.

Ejecutar script para la creación del modelo entidad relación terminado.

Crear todas las clases de entidades de dominio descritas como tablas en la base de datos

Creación de los modelos Dto. para mostrar data, actualizar e insertar por cada entidad de dominio.

Tercer sprint

Establecer las reglas de mapeo para cada entidad y sus Dto. correspondiente.

Instalar Entity Framework Core y todas sus dependencias

Crear la clase context para el acceso a los datos e incluir todas las entidades de dominio para relacionarlas con cada una de las tablas existentes en la base de datos.

Especificar mediante FLUENT API las reglas de relación y restricción que tiene cada entidad de dominio según el modelado de la base de datos.

Establecer la conexión de la capa de acceso a datos con el SGBD.

Testear el correcto funcionamiento de la capa de acceso a datos.

Cuarto sprint

Crear métodos de autenticación.

Crear un método de autorización basado en tokens

Crear los repositorios de cada entidad junto con sus interfaces o contratos.

Quinto sprint

Implementar patrón de diseño scoped para todos los repositorios que se conectan con los controladores de cada entidad respectivamente.

Implementar patrón transient para los mapeadores de campos en cada entidad.

Crear un módulo para verificación automática de depósitos.

Crear los controladores para cada entidad y sus relaciones directas.

Sexto sprint

Test de autenticación y autorización

Test de manipulación en el servicio de clientes

Test de manipulación en el servicio para compras y sus productos (Crear y actualizar)

Test de manipulación en el servicio para recibir encomiendas en el casillero postal y sus productos (Crear y actualizar)

Test de manipulación en el servicio de depósitos (Crear y actualizar)

Séptimo sprint

Test de manipulación en el servicio de reembolsos (Crear y actualizar)

Test de manipulación en el servicio de Empleados (Crear y actualizar)

Test de manipulación en el servicio de guías aéreas (Crear y actualizar)

Test de manipulación en el servicio de despachos a clientes (Crear y actualizar)

Test de manipulación en el servicio para subir estados de cuenta

Octavo sprint

Creación de la interfaz para login

Creación de un panel de navegación

Creación de pantalla de inicio.

Creación de formulario para clientes (Crear y actualizar)

Creación de la interfaz para listado de clientes.

Creación de formulario para compras y sus productos (Crear y actualizar)

Creación de la interfaz para listado de compras y sus productos

Creación de formulario para recibir encomiendas en el casillero postal y sus productos (Crear y actualizar)

Creación de la interfaz para listado de encomiendas recibidas en el casillero postal y sus productos

Creación de formulario para depósitos (Crear y actualizar)

Creación de la interfaz para listado de depósitos.

Creación de formulario para reembolsos (Crear y actualizar)

Creación de la interfaz para listado de reembolsos.

Creación de formulario para Empleados (Crear y actualizar)

Creación de la interfaz para listado de empleados.

Creación de formulario para creación de guías aéreas (Crear y actualizar)

Creación de la interfaz para listado de guías aéreas.

Creación de formulario para despachos a clientes (Crear y actualizar)

Creación de la interfaz para listado de despachos a clientes.

Creación de formulario para subir estados de cuenta (Crear y actualizar)

Creación de la interfaz para listar estados de cuenta.

Creación de la interfaz para listar agencias.

Creación de la interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

Noveno sprint

Establecer una hoja de estilos CSS estandarizada para la creación de todos los componentes web que se van a usar en el desarrollo.

Desarrollo de la interfaz para login

- Envío de petición POST para la autenticación
- Responder con el token de autorización.
- Guardar el token para adjuntarlo en todas las solicitudes que realice el usuario durante la sesión.

Desarrollo de un panel de navegación

- Con diseño responsive

Desarrollo de pantalla de inicio.

- Contiene botones de acción rápida para agregar cliente, recibir encomienda en casillero, agregar un nuevo depósito de clientes o crear una nueva compra.
- Agregar un espacio para leer códigos de productos que muestre en un modal toda la información relacionada.
- Agregar un espacio para listar todos los clientes con filtros para búsqueda y acceso rápido.
- Crear un espacio que liste las guías aéreas más próximas pendientes de recibir y los envíos a clientes pendientes de despachar.

Desarrollo de formulario para clientes (Crear y actualizar)

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.
- Ocultar la contraseña.
- Agregar un espacio para añadir el permiso de usuario correspondiente.
- Al Cliente únicamente se le puede dar permiso para ingresar al sistema y ver sus transacciones.

Desarrollo de la interfaz para listado de clientes.

- Listar los clientes con paginación agregando un botón para visitar el perfil del cliente y otro para realizar modificaciones en sus datos personales y permisos.

Desarrollo de formulario para compras y sus productos (Crear y actualizar).

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen de compra y los campos que contiene el formulario de compras.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un débito por la compra realizada.

- Posterior a esto se permite insertar los productos de compra con una imagen y su información referente.

Desarrollo de la interfaz para listado de compras y sus productos.

- Se lista todas las compras o las compras de un cliente específico.

- Cuando se listan las compras de todos los clientes, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 3 botones; uno para ver los detalles de la compra, otro para agregar uno o varios números de tracking y otro para agregar más productos a dicha compra.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Décimo sprint

Desarrollo de formulario para recibir encomiendas en el casillero postal y sus productos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen de la encomienda recibida y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un débito por algún servicio cobrado.

- Posterior a esto se permite insertar los productos que contiene dicha encomienda con una imagen y su información referente.

- Cada Producto debe tener una imagen, peso, seleccionar si es una compra del stock o una encomienda recibida solo por servicio de casillero, la ubicación donde se recibe el artículo y una descripción opcional.

Desarrollo de la interfaz para listado de encomiendas recibidas en el casillero postal y sus productos

- Se lista todas las encomiendas recibidas en el casillero, o todas las de un cliente específico.

- Cuando se listan las encomiendas recibidas de todos los clientes, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para ver los detalles del paquete y otro para agregar productos en la encomienda recibida.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Desarrollo de formulario para depósitos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen del depósito y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla Bills del cliente generando un crédito por el valor correspondiente al depósito luego de ser validado.

Desarrollo de la interfaz para listado de depósitos.

- Se lista todos los depósitos registrados, o los de un cliente específico.

- Cuando se listan todos los depósitos registrados, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para validar que el depósito se encuentra en el reporte del banco y otro para marcar el depósito como no encontrado en caso de que no exista.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

- Cada depósito debe pasar por una verificación antes de agregarlo como un saldo válido que el cliente puede usar; esta verificación puede terminar catalogando al depósito como válido o no encontrado.

Desarrollo de formulario para reembolsos (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.

- Se debe poder agregar una imagen del reembolso emitido y los campos que contiene el formulario.

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Al guardar el registro se debe actualizar la tabla “Bills” del cliente generando un débito por el valor correspondiente al reembolso emitido.

Desarrollo de la interfaz para listado de reembolsos.

- Se lista todos los reembolsos registrados, o los de un cliente específico con paginación.

- Cuando se listan todos los reembolsos registrados, el campo “nombre del cliente” debe ser un link que redirija al perfil de la persona.

- A cada registro se le debe agregar 2 botones; uno para validar que el reembolso se realizó y otro para ver el detalle del reembolso.

- Se debe agregar accesos directos para eliminar y editar junto con sus validaciones.

Desarrollo de formulario para Empleados (Crear y actualizar)

- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.

- Ocultar la contraseña.

- Agregar un espacio para añadir el permiso de usuario correspondiente.

- Al Cliente únicamente se le puede dar permiso para ingresar al sistema y ver sus transacciones.

Desarrollo de la interfaz para listado de empleados.

- Listar los empleados con paginación.

- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para creación de guías aéreas (Crear y actualizar)

- Se debe realizar un formulario con los atributos de su entidad.

- Para agregar productos a una guía aérea se debe incluir un campo para escanear el código de cada producto.

- El programa debe analizar si el producto no ha sido agregado en ninguna otra guía.

- Se debe tener la opción de subir una imagen general de la guía aérea.

Desarrollo de la interfaz para listado de guías aéreas.

- Se debe listar todas las guías con opciones de paginación.
- Debe contener información que identifique el estado de la guía, y el contenido en ella.
- Cada registro debe contener 2 botones; uno para ver la información de dicha guía, y otro para recibir la guía con sus productos cuando ésta llega a su destino.
- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para despachos a clientes (Crear y actualizar)

- Se debe poder seleccionar un cliente de la lista de clientes.
- Se debe poder agregar una dirección de envío, si no existe debe permitir crearla.
- Enviar las peticiones de crear y actualizar con el token en el header de la aplicación.
- Al guardar el registro se debe actualizar la tabla “Bills” del cliente generando un débito por el valor correspondiente al valor del envío generado.

Desarrollo de la interfaz para listado de despachos a clientes.

- Se debe listar todas las guías con opciones de paginación.
- Debe contener información que identifique el estado de la guía, y el contenido en ella, el peso y la información de envío.
- Cada registro debe contener 2 botones; uno para ver la información de dicha guía, y otro marcar como enviado cada producto cuando se realiza el despacho.
- Se debe agregar accesos directos para eliminar y editar.

Desarrollo de formulario para subir estados de cuenta (Crear)

- Se debe cargar una lista de todas las cuentas bancarias que se han creado.
- Se escoge subir el archivo generado por el banco a la cuenta bancaria que le corresponde.
- Se carga una vista previa de los movimientos bancarios que se van a subir y posteriormente se cargan los datos del archivo al servidor.

Desarrollo de la interfaz para listar estados de cuenta.

- Se debe cargar una lista de todas las cuentas bancarias que se han creado.

- Se escoge una cuenta bancaria para cargar el estado de cuenta subido hasta la fecha.

- Cada registro contiene un campo que indica si dicho movimiento se ha vinculado a algún depósito ingresado por el cliente.

- No se permite borrar ni editar nada de este registro.

Desarrollo de la interfaz para listar agencias.

- Se lista cada Agencia con el contenido de sus bodegas.

- Se permite ver el contenido de sus bodegas a detalle por cliente.

Desarrollo de la interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

- Este panel de información debe contener las acreditaciones y débitos que ha tenido un usuario a lo largo de su trayectoria.

- Cada registro debe contener un botón que vincule dicho registro con un depósito, envió, o compra según corresponda.

Décimo primer sprint

Desarrollar los servicios de conexión http para realizar las peticiones a la API según corresponda.

Desplegar la aplicación web, el backend y la base de datos listo en producción.

2.5. EJECUCIÓN Y/O ENSAMBLAJE DEL PROTOTIPO

Se realiza el desarrollo de los sprints planteados para construir la solución del proyecto.

2.5.1 FASE DE IMPLEMENTACIÓN

El team developer pone en marcha la elaboración y el cumplimento de cada tarea asignada en los sprints a fin de implementar todos los requerimientos del sistema

2.5.1.1 Revisión del primer sprint

A continuación se describe el detalle de los resultados obtenidos en el primer sprint.

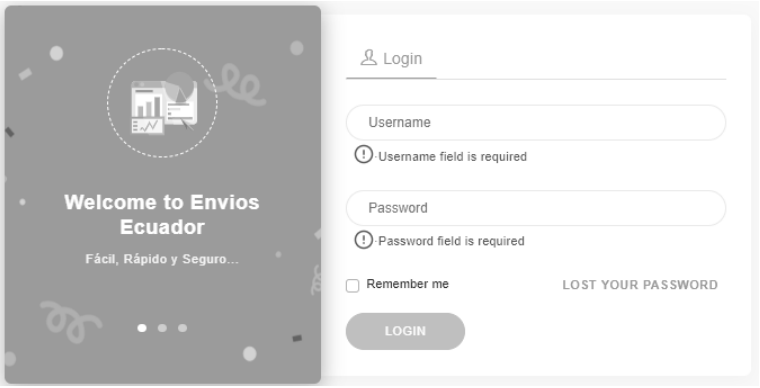


Figura 7: Diseño – Login

Se ha diseñado, en la figura 7, una interfaz para login con el objetivo de iniciar una sesión en la aplicación una vez aprobadas las credenciales de autenticación.

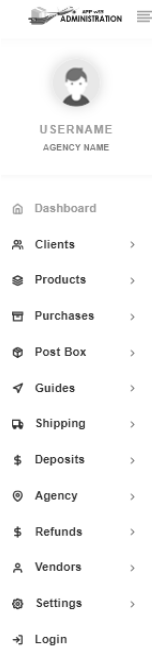


Figura 8: Diseño - Panel de navegación

Se ha diseñado, en la figura 8, un componente panel de navegación que contiene el acceso a las diferentes funcionalidades del sistema.

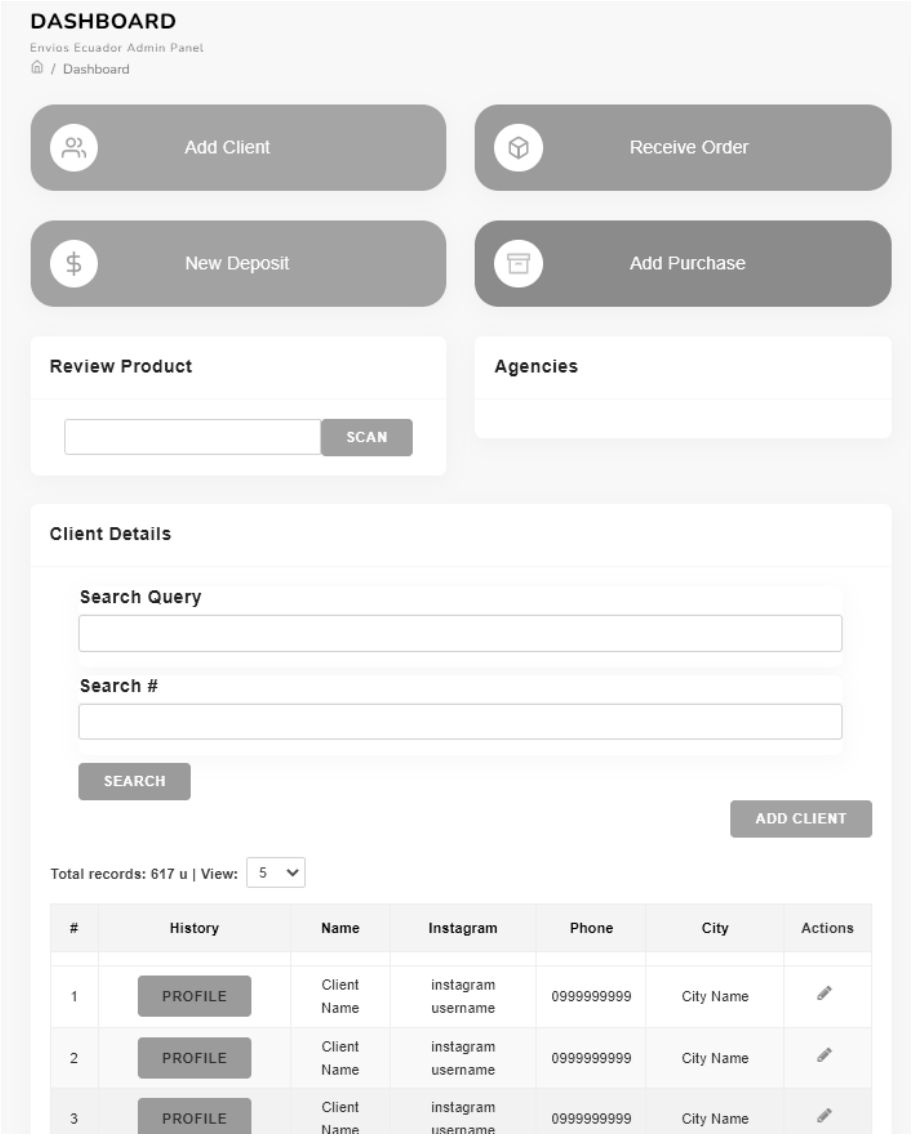


Figura 9: Diseño - Pantalla de inicio

Se ha diseñado, en la figura 9, una interfaz de inicio para la aplicación, el usuario será redirigido a esta interfaz luego de autenticarse, es un tablero con información y acceso a las funciones más usadas por los operadores del sistema.

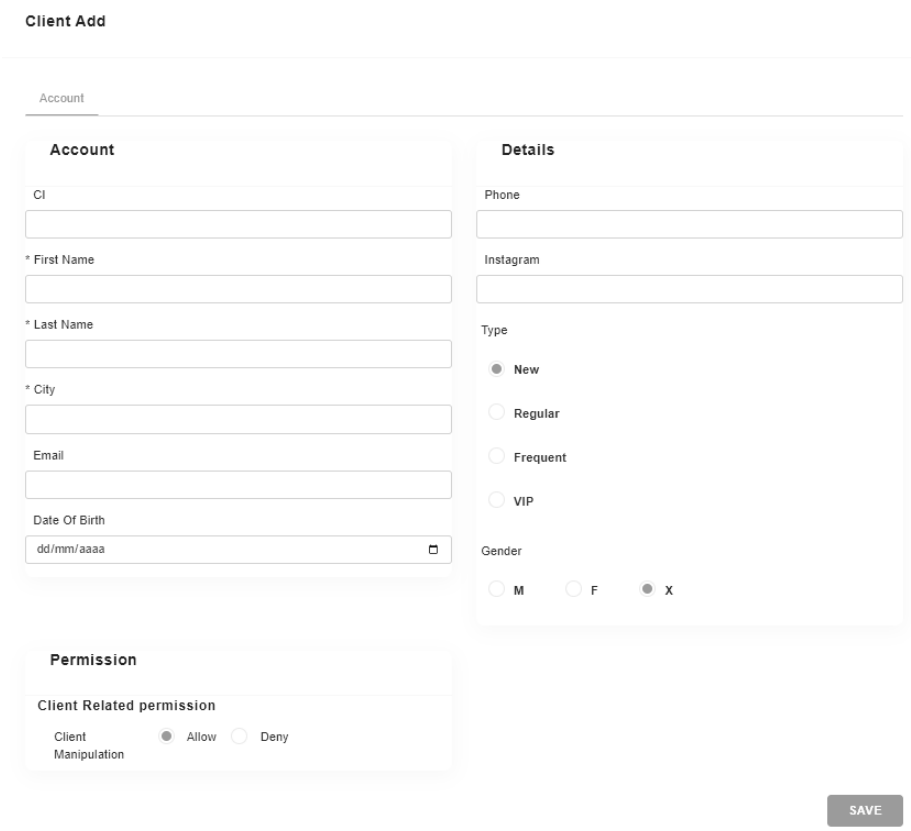


Figura 10: Diseño - Formulario clientes

Se ha diseñado, en la figura 10, el formulario para crear y actualizar clientes.

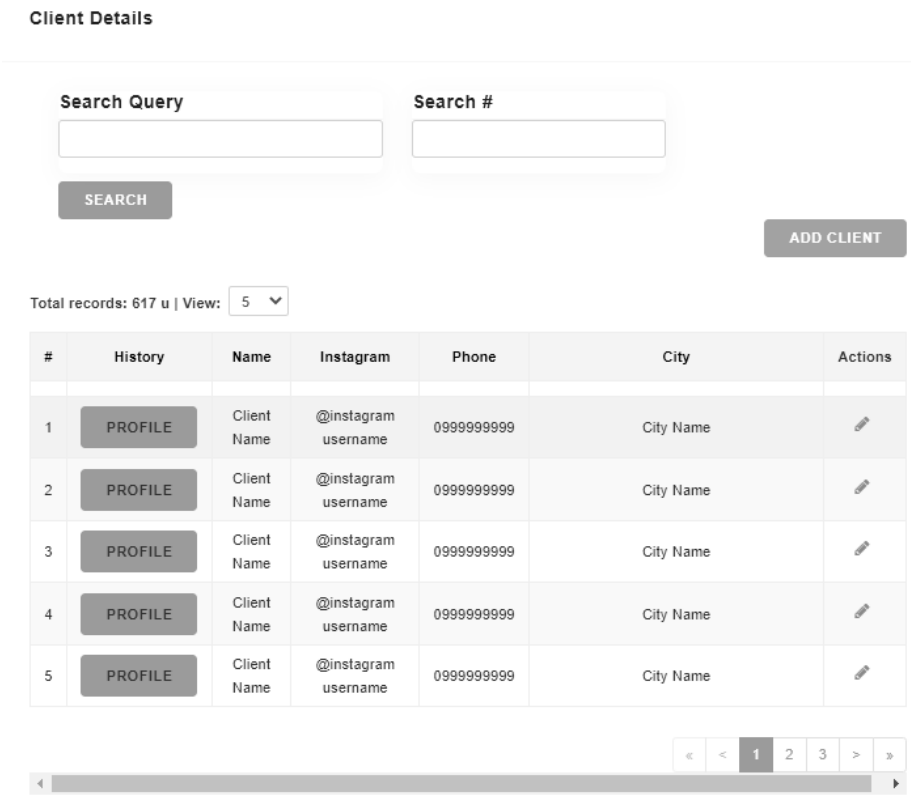


Figura 11 Diseño - Listar clientes

Se ha diseñado, en la figura 11, una interfaz para listar los clientes del negocio.

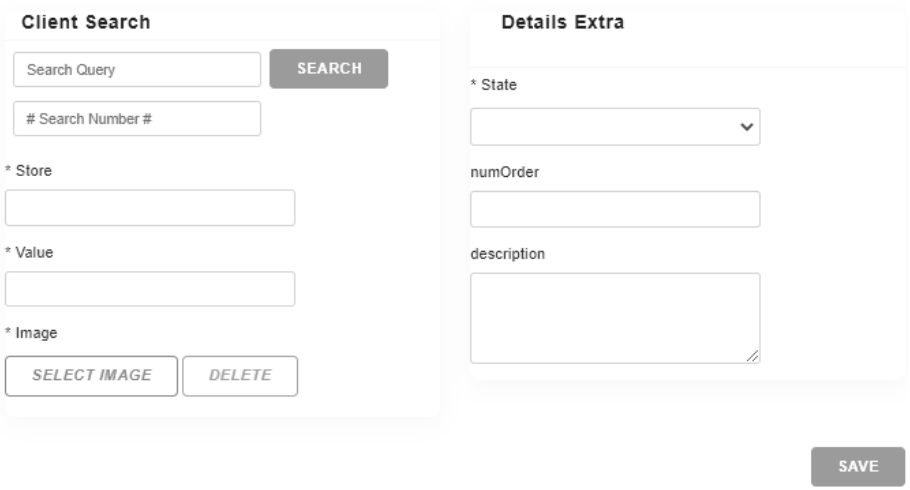


Figura 12: Diseño - Formulario compras

Se ha diseñado, en la figura 12, el formulario para crear y actualizar compras.

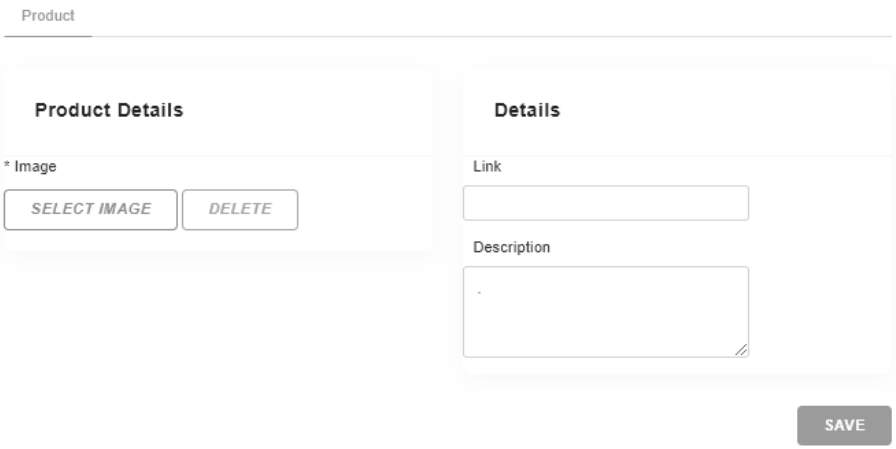


Figura 13: Diseño - Agregar productos a compra

Se ha diseñado, en la figura 13, el formulario para crear y actualizar productos de compras.

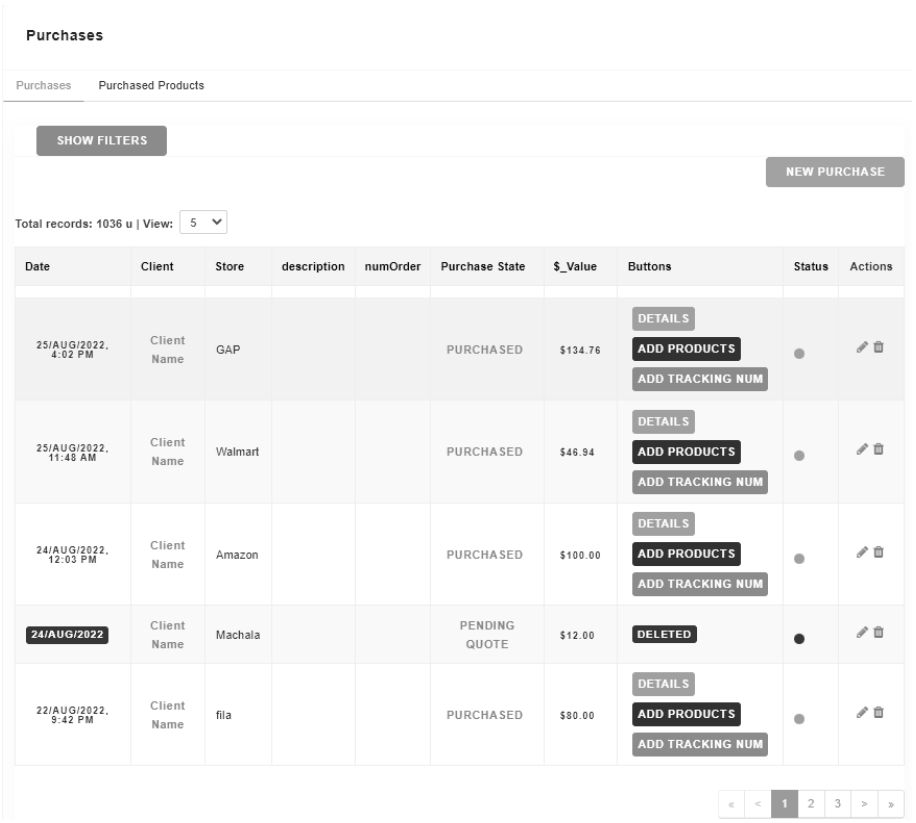


Figura 14: Diseño - Listar Compras

Se ha diseñado, en la figura 14, una interfaz para listar las compras realizadas.

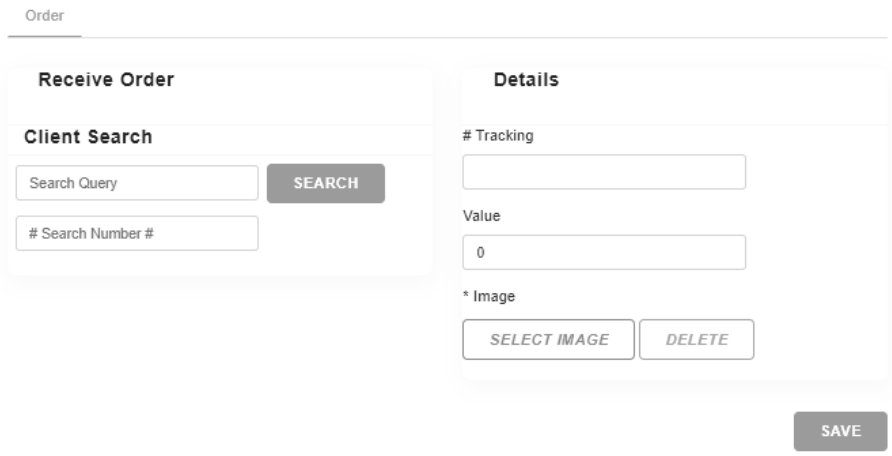


Figura 15: Diseño - Formulario recibir encomiendas

Se ha diseñado, en la figura 15, el formulario para crear y actualizar encomiendas recibidas en el casillero postal.

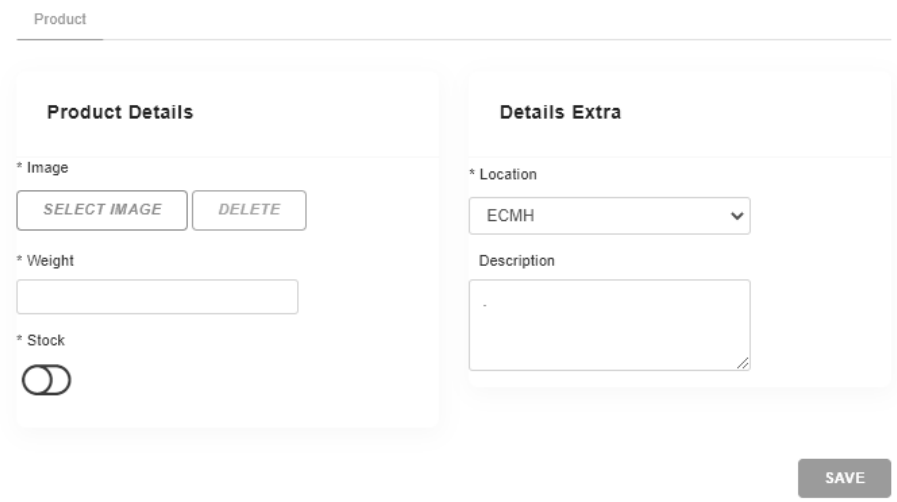


Figura 16: Diseño - Formulario agregar contenido de encomienda

Se ha diseñado, en la figura 16, el formulario para crear y actualizar el contenido de las encomiendas recibidas en el casillero postal.

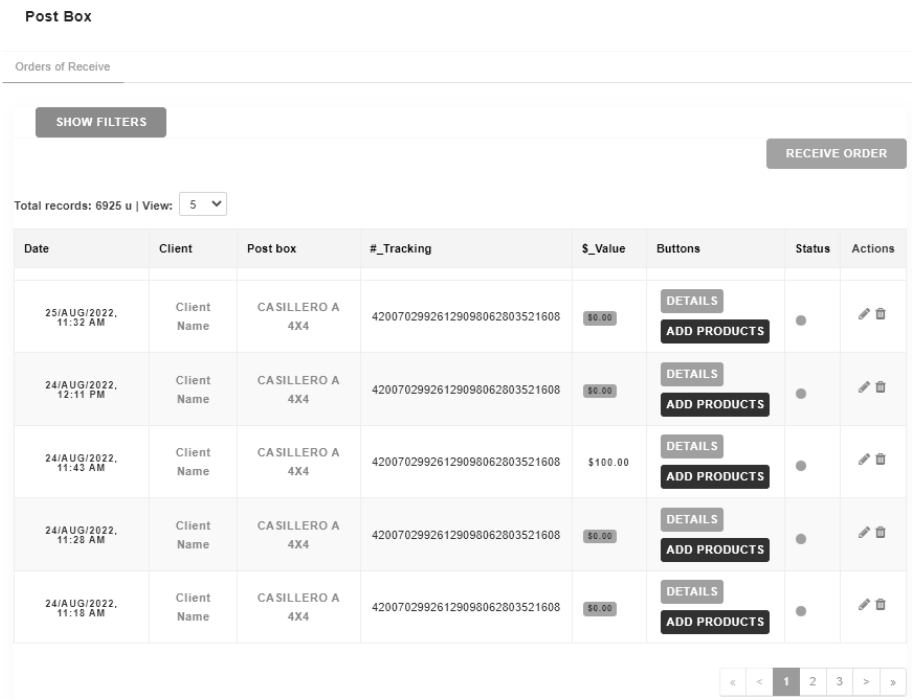


Figura 17: Diseño - Listar encomiendas recibidas

Se ha diseñado, en la figura 17, una interfaz para listar las encomiendas recibidas en el casillero postal y sus productos

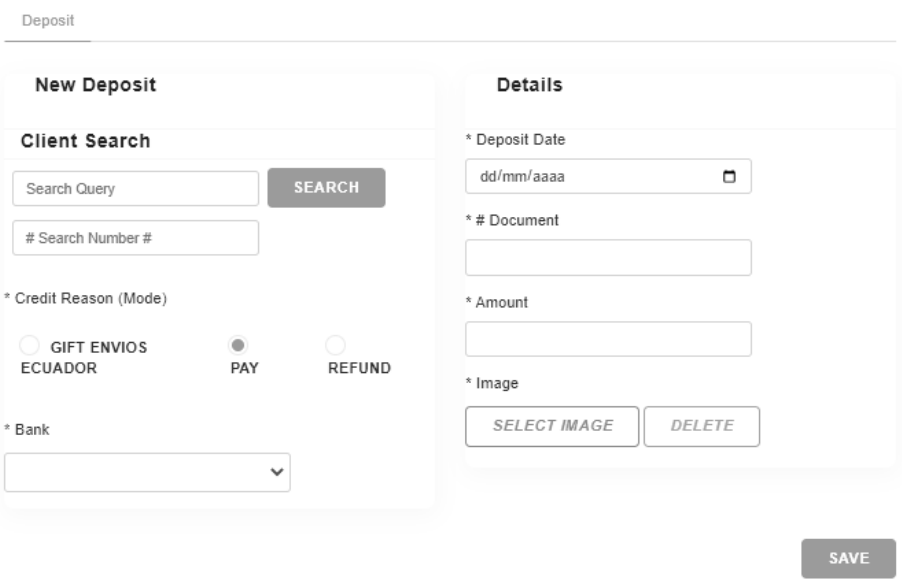


Figura 18: Diseño - Formulario para depósitos

Se ha diseñado, en la figura 18, el formulario para crear y actualizar depósitos de clientes.

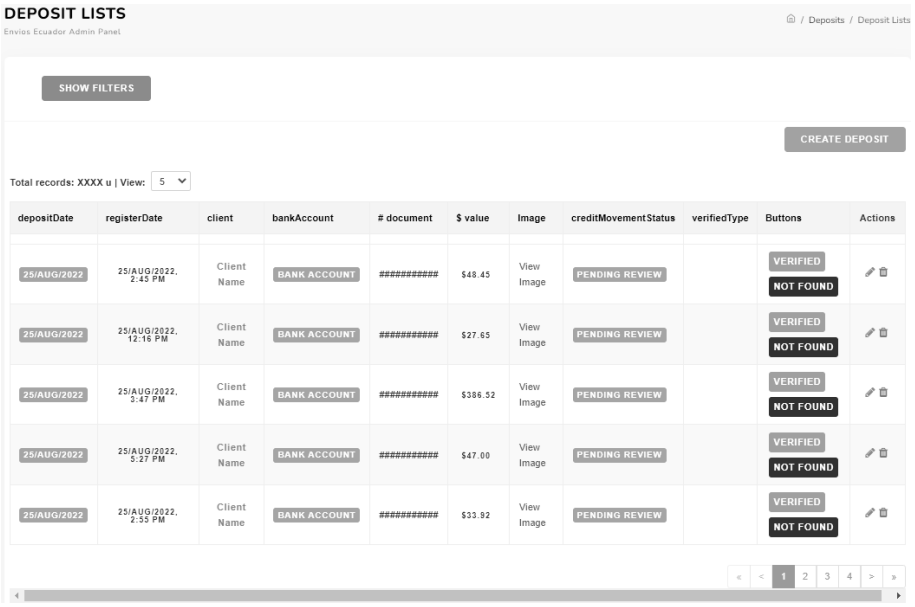


Figura 19: Diseño - Listar depósitos

Se ha diseñado, en la figura 19, una interfaz para listar los depósitos de clientes.

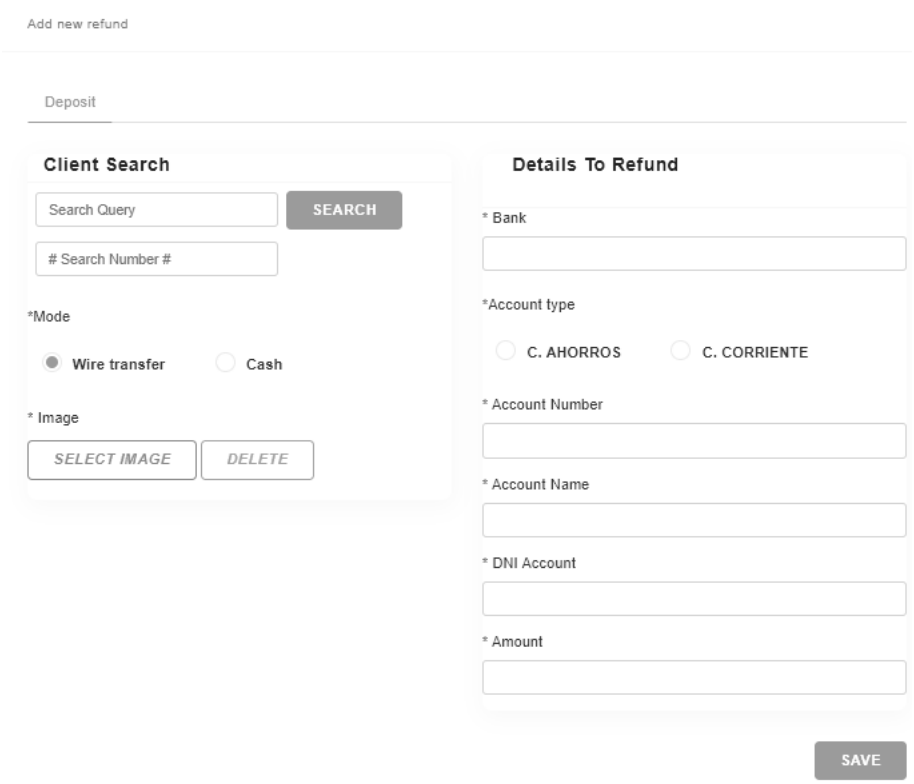


Figura 20: Diseño - Formulario reembolsos

Se ha diseñado, en la figura 20, el formulario para crear y actualizar reembolsos emitidos por la empresa.

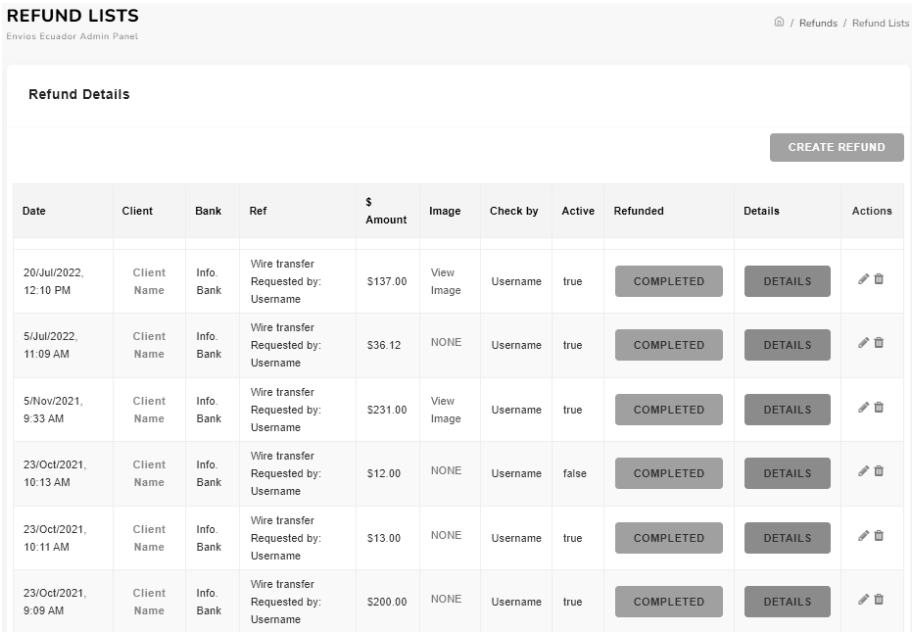


Figura 21: Diseño - Listar reembolsos

Se ha diseñado, en la figura 21, una interfaz para listar los reembolsos a clientes que realiza el negocio.

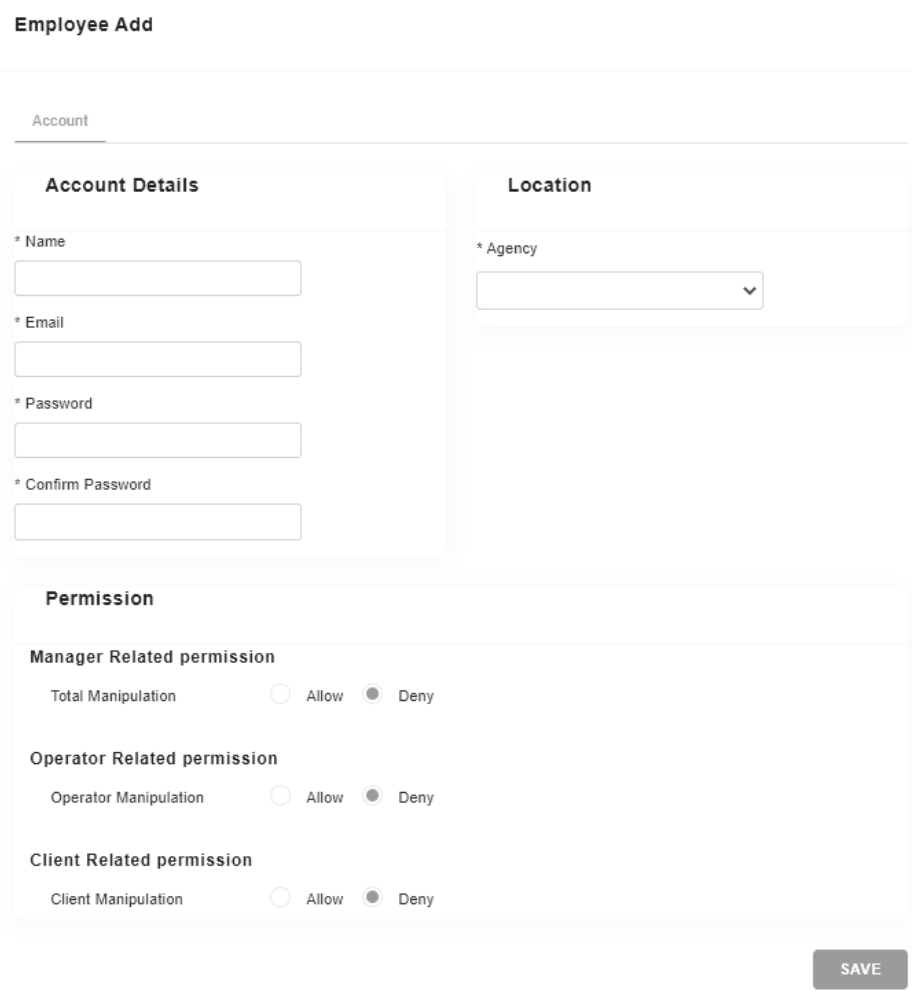


Figura 22: Diseño - Formulario empleados

Se ha diseñado, en la figura 22, el formulario para crear y actualizar los colaboradores del negocio.

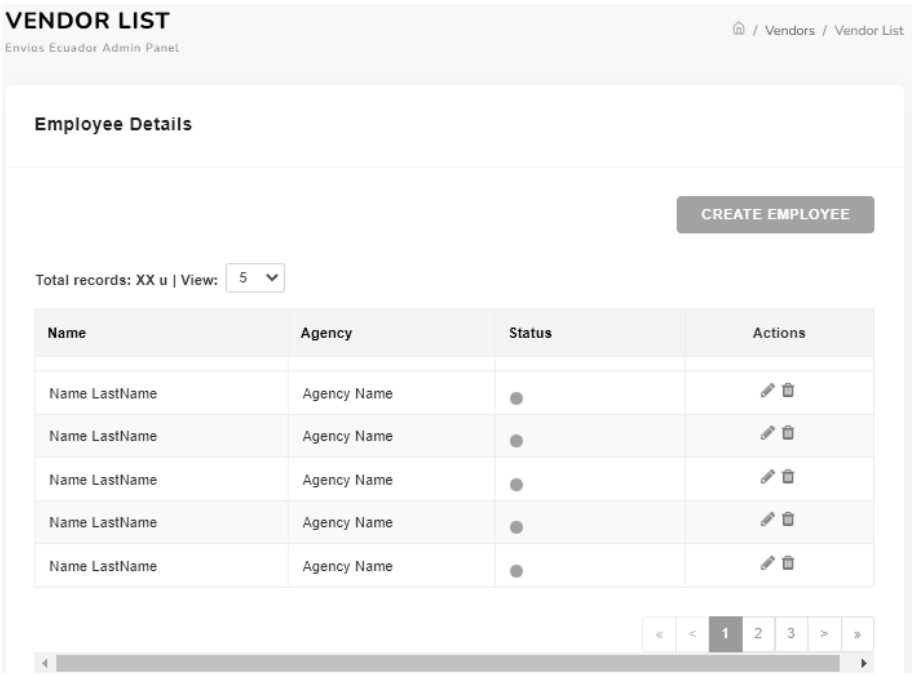


Figura 23: Diseño - Listar empleados

Se ha diseñado, en la figura 23, una interfaz para listar los empleados de la empresa.

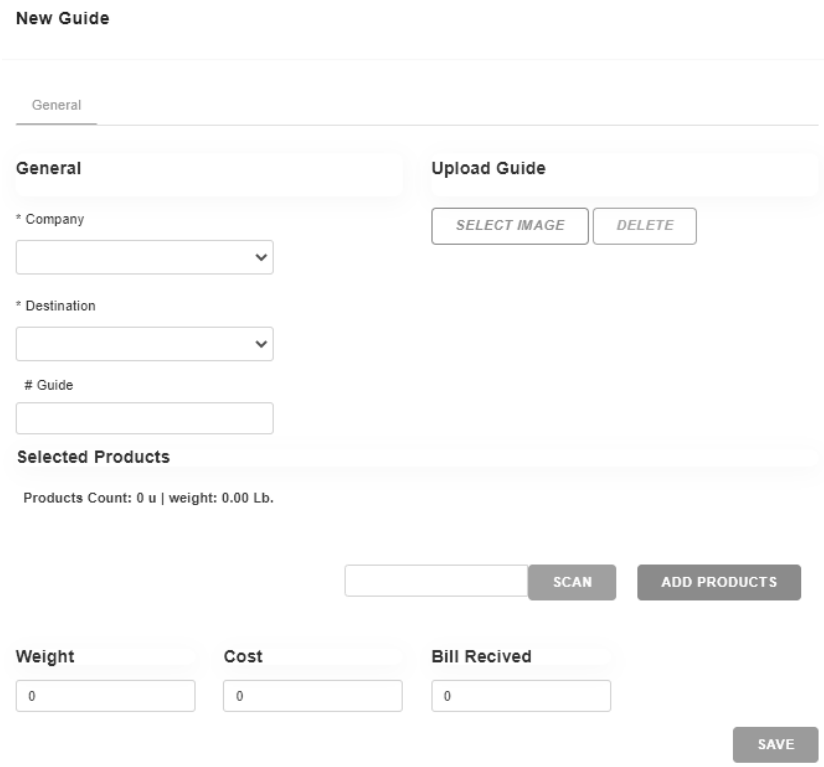


Figura 24: Diseño - Formulario guías aéreas

Se ha diseñado, en la figura 24, el formulario para crear y actualizar las guías aéreas de la empresa.

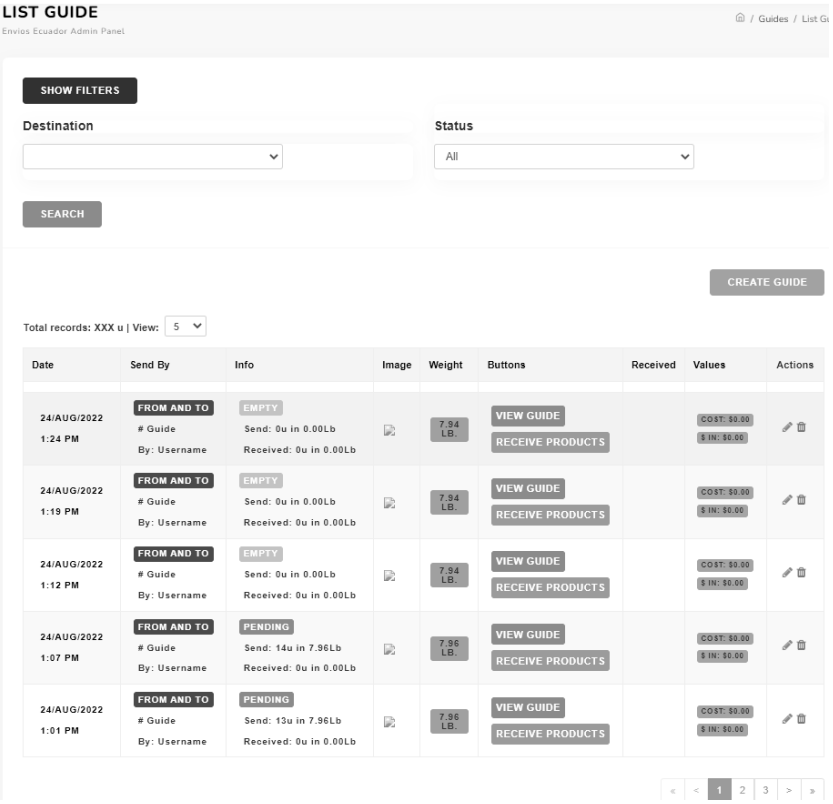


Figura 25: Diseño - Listar guías aéreas

Se ha diseñado, en la figura 25, una interfaz para listar las guías aéreas que sirven para la movilización de carga.

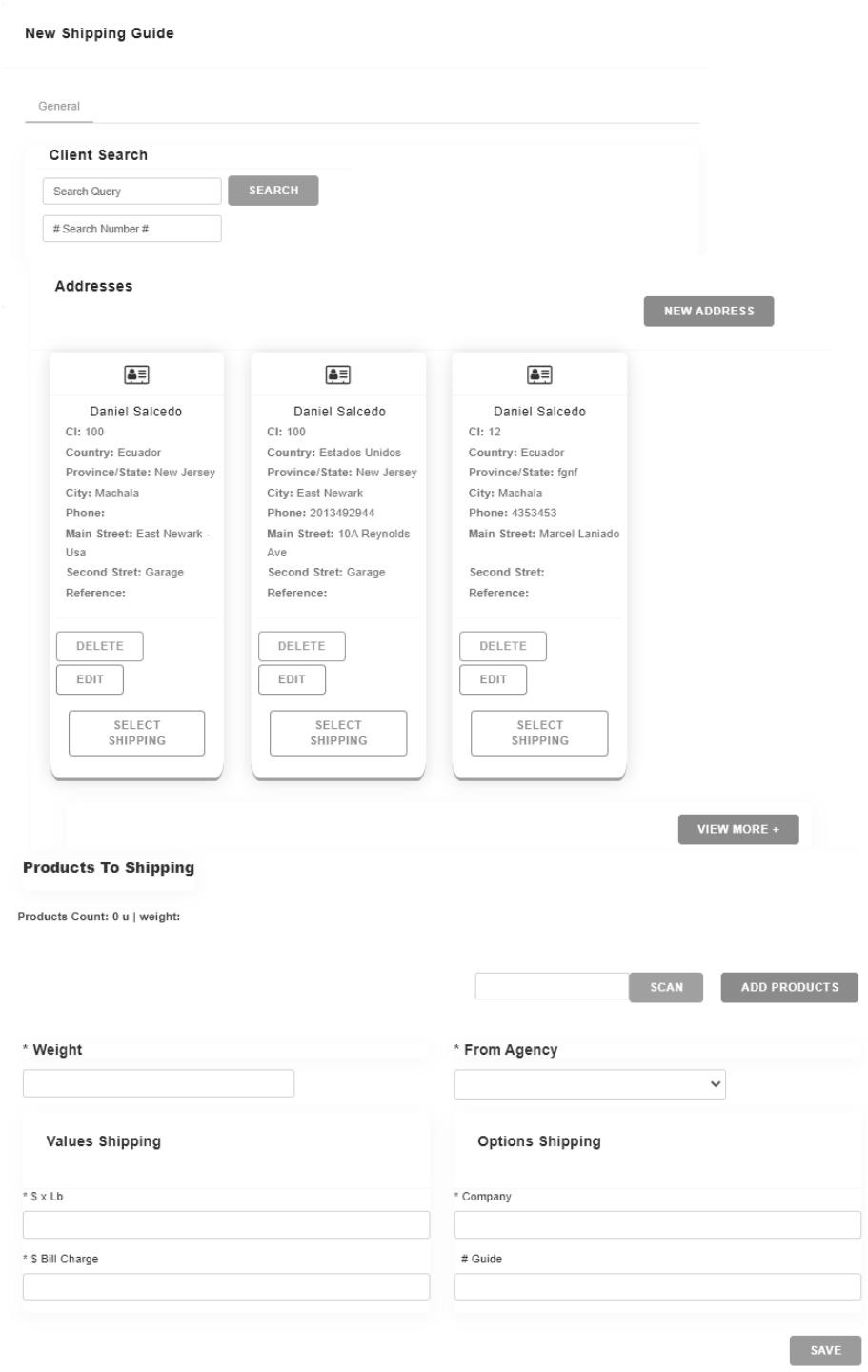


Figura 26: Diseño - Formulario despacho a cliente

Se ha diseñado, en la figura 26, el formulario para crear y actualizar las guías de despachos a clientes.

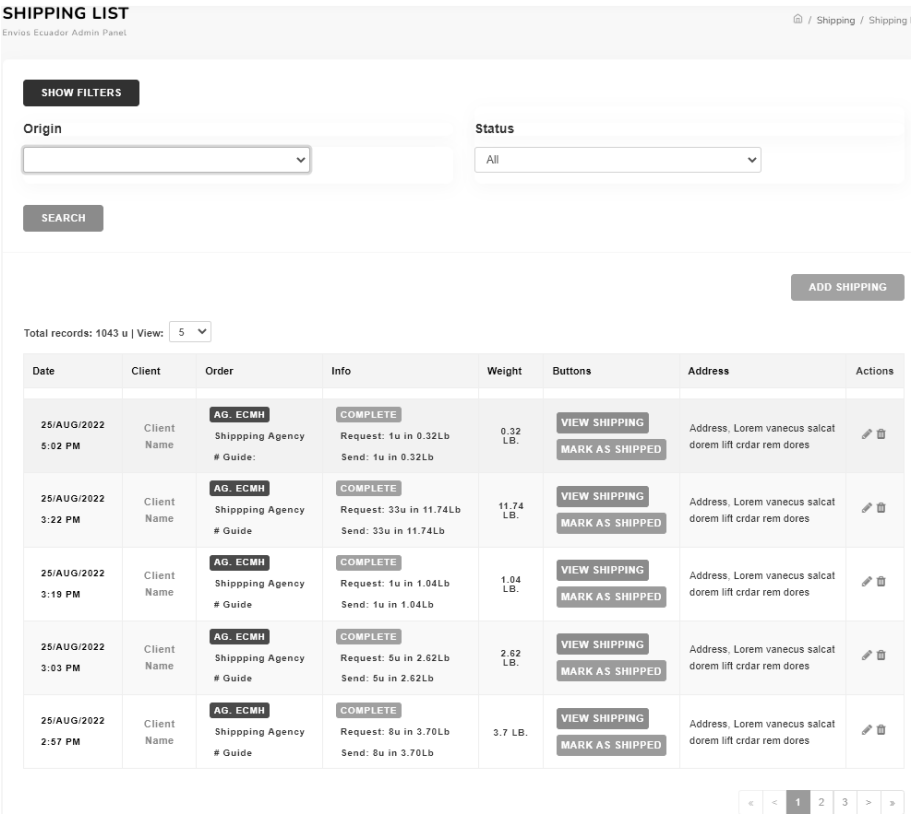


Figura 27: Diseño - Listar despachos a clientes

Se ha diseñado, en la figura 27, una interfaz para listar los despachos que se realizan al cliente final.

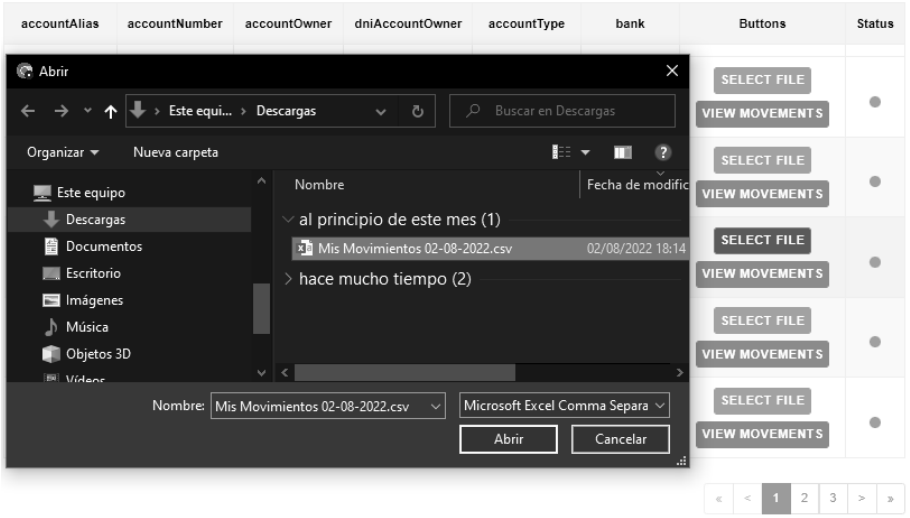


Figura 28: Diseño - Cargar estado de cuenta

Se ha diseñado, en la figura 28, el formulario para cargar los movimientos de una cuenta bancaria, con la finalidad de verificar si existen en la lista de depósitos que se registran.

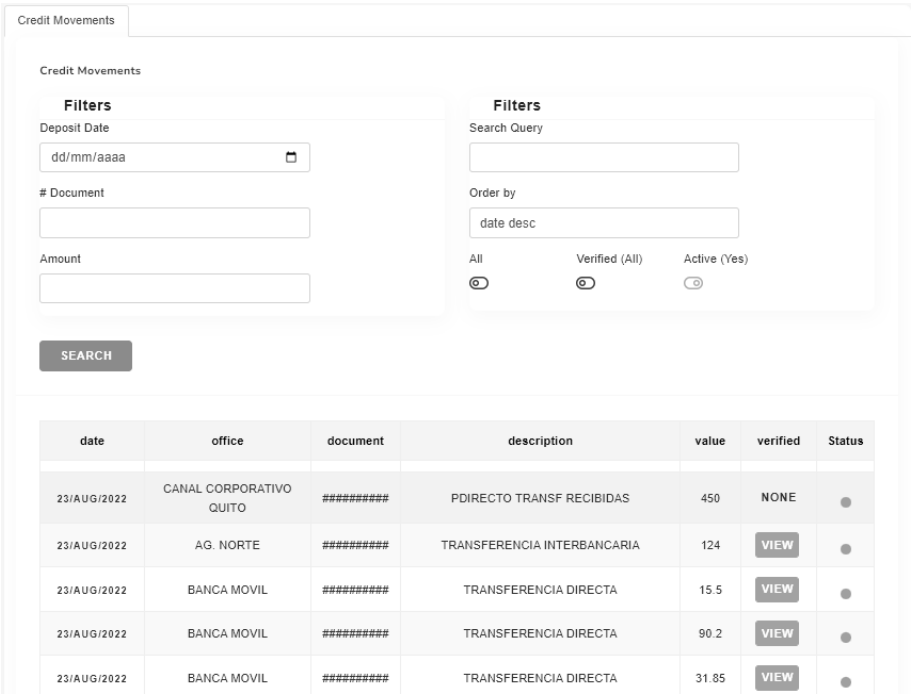


Figura 29: Diseño - Listar estados de cuenta

Se ha diseñado, en la figura 29, una interfaz para listar los estados de cuenta cargados por la empresa.

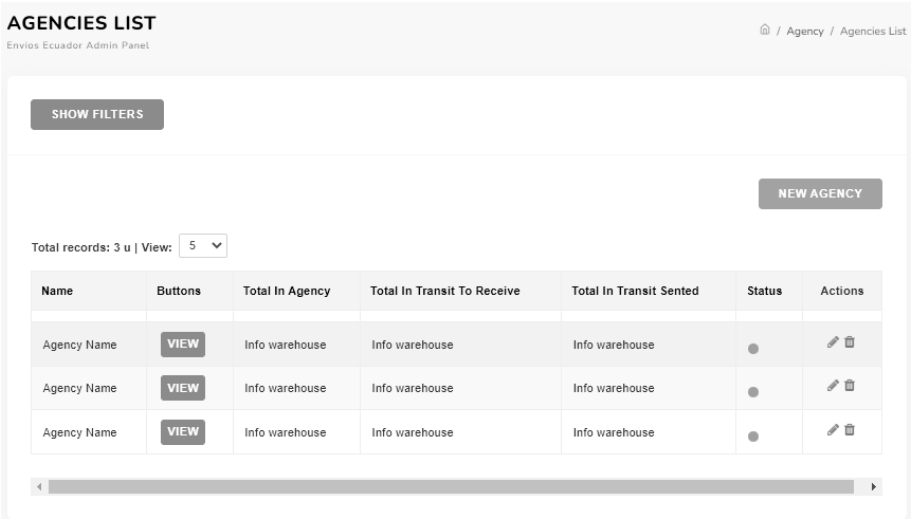


Figura 30: Diseño - Listar agencias

Se ha diseñado, en la figura 30, una interfaz para listar las agencias que forman parte de la empresa.

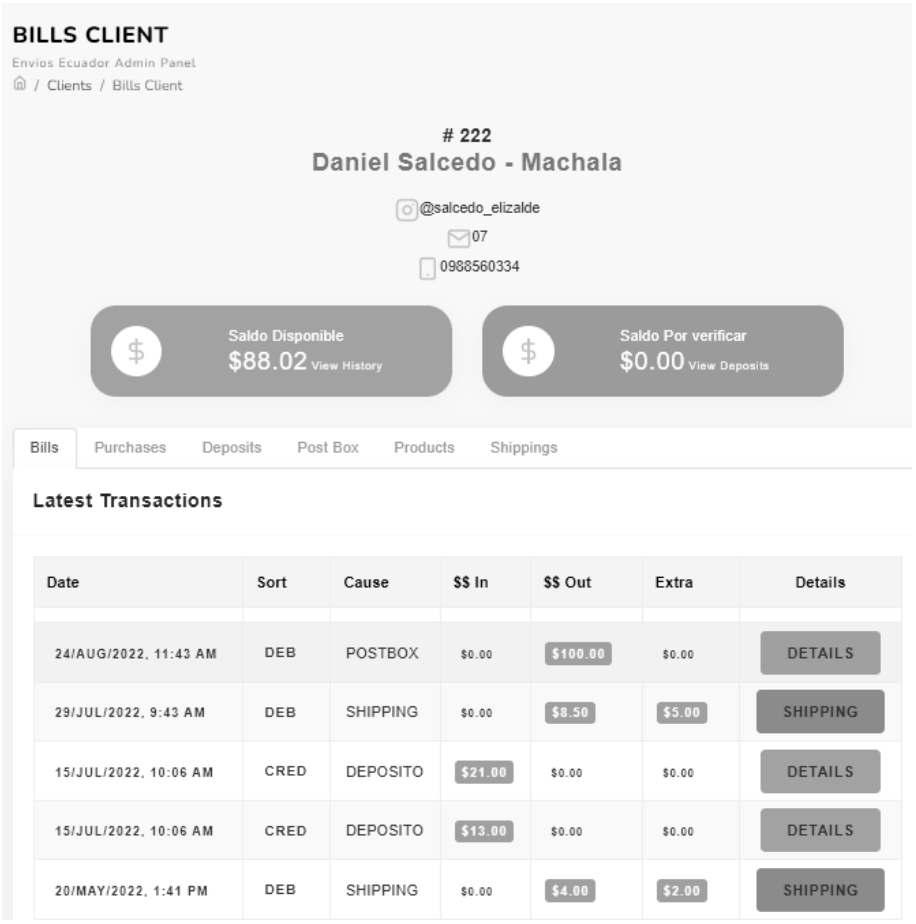


Figura 31: Diseño - Perfil del cliente

Se ha diseñado, en la figura 31, una interfaz para listar los movimientos de crédito y débito en la cuenta de un cliente con su balance de saldos y las transacciones del negocio que haya realizado.

2.5.1.2 Revisión del segundo sprint

A continuación se describe el detalle de los resultados obtenidos en el segundo sprint.

- Se ha establecido las entidades del negocio, las relaciones entre ellas y sus índices dando paso a la creación del modelo relacional, mismo que se puede ver en el anexo 1 del trabajo.

- Se ha generado el código SQL para la creación de la base de datos, mismo que se puede ver en el anexo 2 del trabajo.

- Se ha ejecutado de manera exitoso el script para la creación de la base de datos, tal como se puede apreciar en la figura 32.

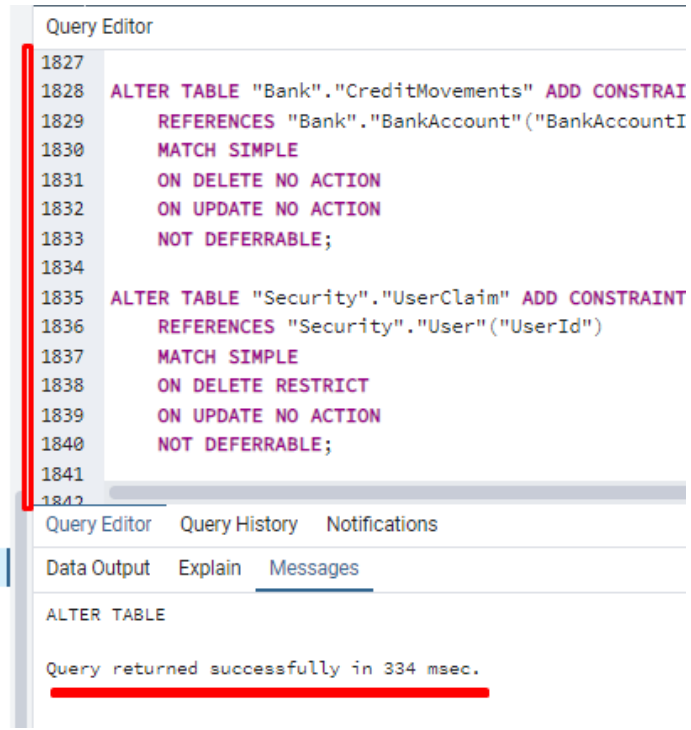


Figura 32: Ejecución de script SQL

- Se crearon todas las entidades de dominio que hace referencia a las tablas en la base de datos y las clases de tipo Dto que mapean las entidades de dominio, tal como se puede apreciar en la figura 33.

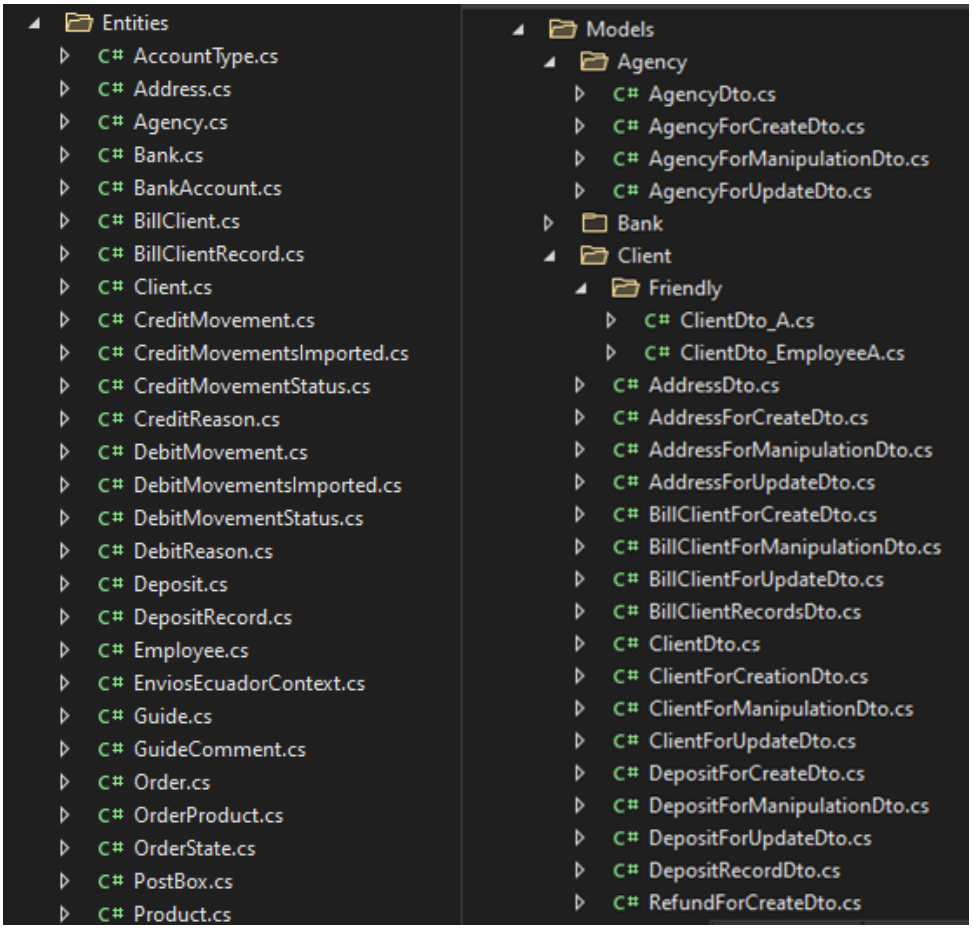


Figura 33: Entidades de dominio y Dtos.

2.5.1.3 Revisión del tercer sprint

A continuación se describe el detalle de los resultados obtenidos en el tercer sprint.

- Se establecen las reglas de mapeo para cada entidad y sus Dto. correspondiente.

Figura 34: Mapeo de entidades

Como indica la figura 34 se aplica esta regla de mapeo para todas las entidades, donde especificamos un dto para consulta, otro para creación, otro para la actualización de datos, cada uno de ellos se mapea a su entidad de dominio.

- Se instaló Entity Framework Core y todas las dependencias del proyecto indicadas en la figura 35.

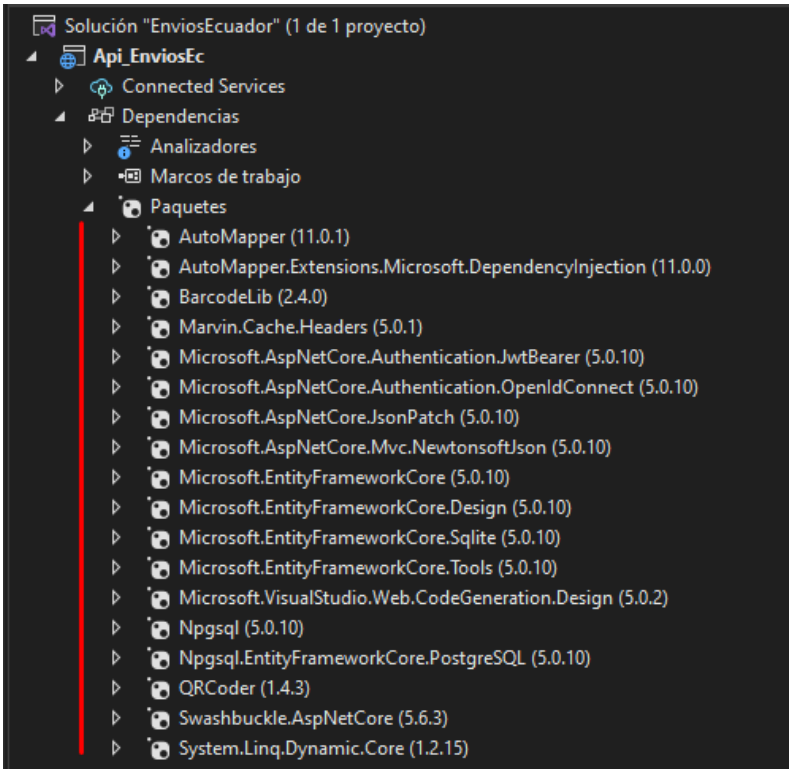
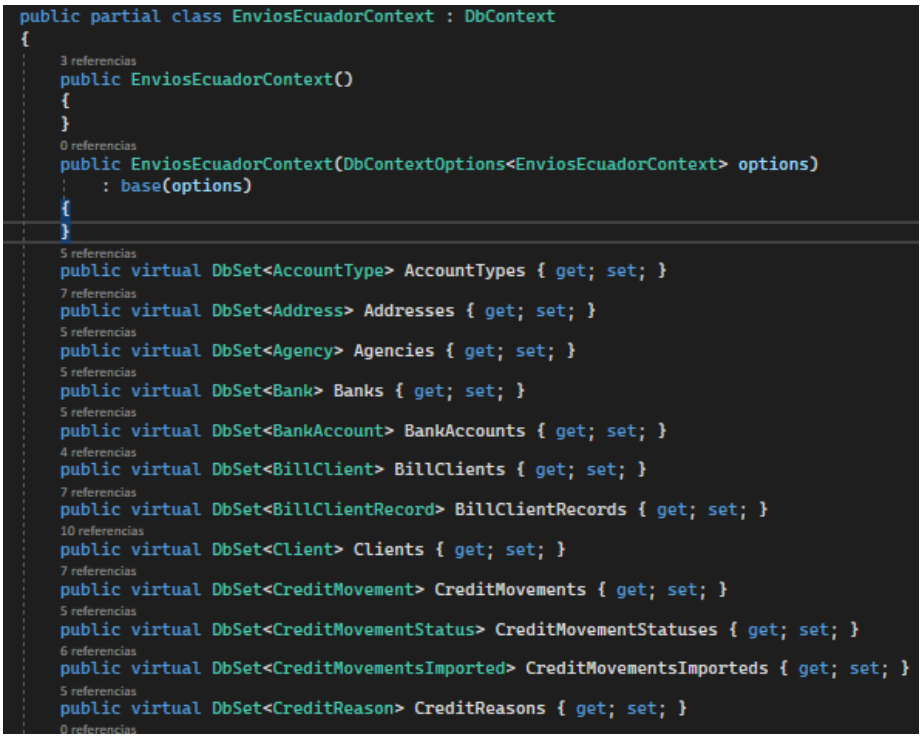


Figura 35: Paquetes NuGgets instalados

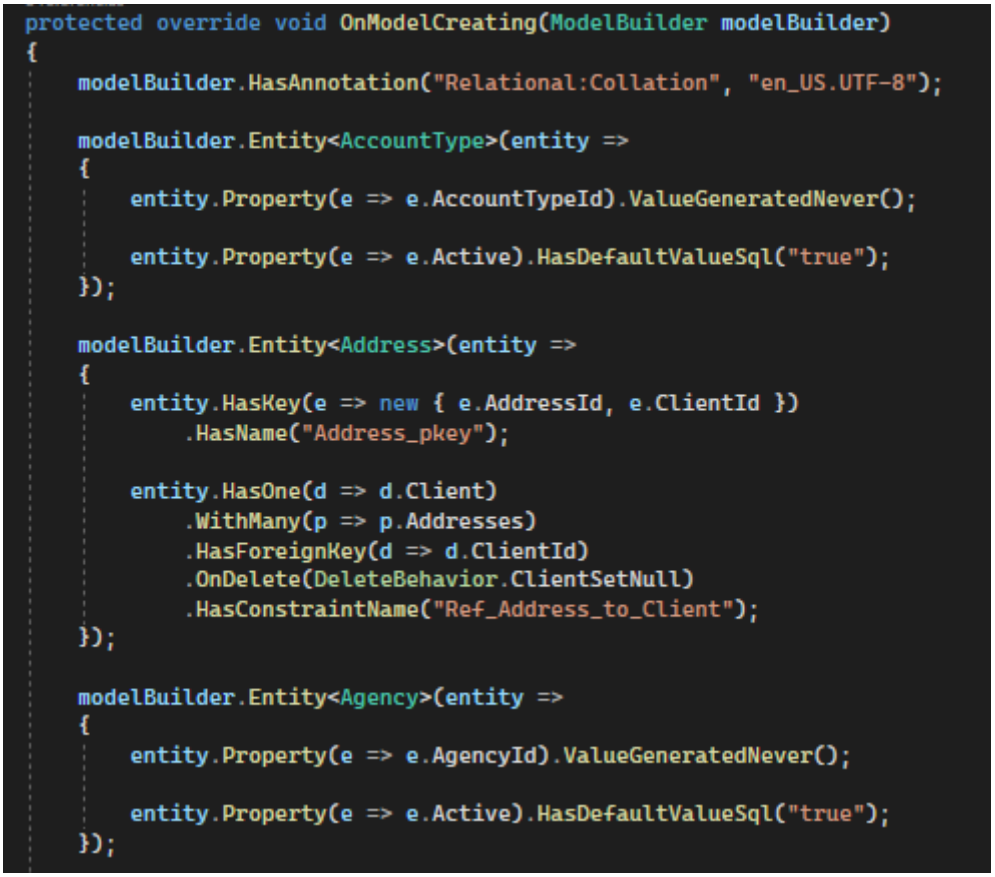
- Se creó la clase context para el acceso a los datos con todas las entidades de dominio tal como se ve en la figura 36.



```
public partial class EnviosEcuadorContext : DbContext
{
    3 referencias
    public EnviosEcuadorContext()
    {
    }
    0 referencias
    public EnviosEcuadorContext(DbContextOptions<EnviosEcuadorContext> options)
        : base(options)
    {
    }
    5 referencias
    public virtual DbSet<AccountType> AccountTypes { get; set; }
    7 referencias
    public virtual DbSet<Address> Addresses { get; set; }
    5 referencias
    public virtual DbSet<Agency> Agencies { get; set; }
    5 referencias
    public virtual DbSet<Bank> Banks { get; set; }
    5 referencias
    public virtual DbSet<BankAccount> BankAccounts { get; set; }
    4 referencias
    public virtual DbSet<BillClient> BillClients { get; set; }
    7 referencias
    public virtual DbSet<BillClientRecord> BillClientRecords { get; set; }
    10 referencias
    public virtual DbSet<Client> Clients { get; set; }
    7 referencias
    public virtual DbSet<CreditMovement> CreditMovements { get; set; }
    5 referencias
    public virtual DbSet<CreditMovementStatus> CreditMovementStatuses { get; set; }
    6 referencias
    public virtual DbSet<CreditMovementsImported> CreditMovementsImporteds { get; set; }
    5 referencias
    public virtual DbSet<CreditReason> CreditReasons { get; set; }
    0 referencias
```

Figura 36: Creación de DbContext

- Se establece mediante Fluent Api las reglas y relaciones de cada entidad, tal como se puede ver en la figura 37.



```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasAnnotation("Relational:Collation", "en_US.UTF-8");

    modelBuilder.Entity<AccountType>(entity =>
    {
        entity.Property(e => e.AccountTypeId).ValueGeneratedNever();

        entity.Property(e => e.Active).HasDefaultValueSql("true");
    });

    modelBuilder.Entity<Address>(entity =>
    {
        entity.HasKey(e => new { e.AddressId, e.ClientId })
            .HasName("Address_pkey");

        entity.HasOne(d => d.Client)
            .WithMany(p => p.Addresses)
            .HasForeignKey(d => d.ClientId)
            .OnDelete(DeleteBehavior.ClientSetNull)
            .HasConstraintName("Ref_Address_to_Client");
    });

    modelBuilder.Entity<Agency>(entity =>
    {
        entity.Property(e => e.AgencyId).ValueGeneratedNever();

        entity.Property(e => e.Active).HasDefaultValueSql("true");
    });
```

Figura 37:Implementación con Fluent Api

- Se estableció la conexión de la capa de acceso a datos con el SGBD haciendo uso del paquete Npgsql.EntityFrameworkCore.PostgreSQL, desde el startup de la aplicación como se indica en la figura 38.

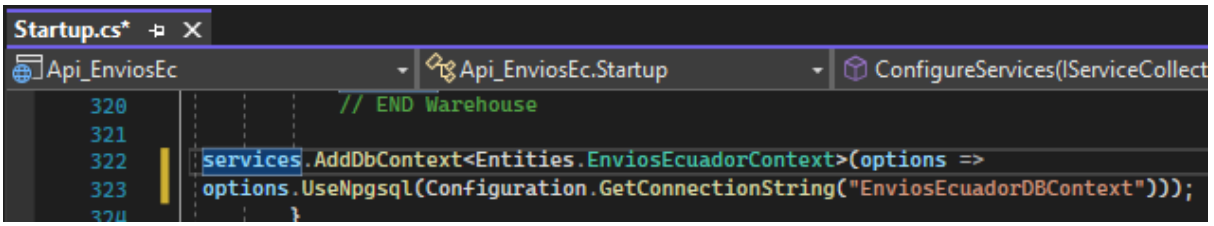


Figura 38: Conexión con el SGBD

2.5.1.4 Revisión del cuarto sprint

A continuación se describe el detalle de los resultados obtenidos en el cuarto sprint.

- Se crearon los métodos de autenticación y autorización basado en tokens, como se puede ver en la figura 39 tenemos una clase que administra la seguridad, y las clases que se usan para interactuar con ella.

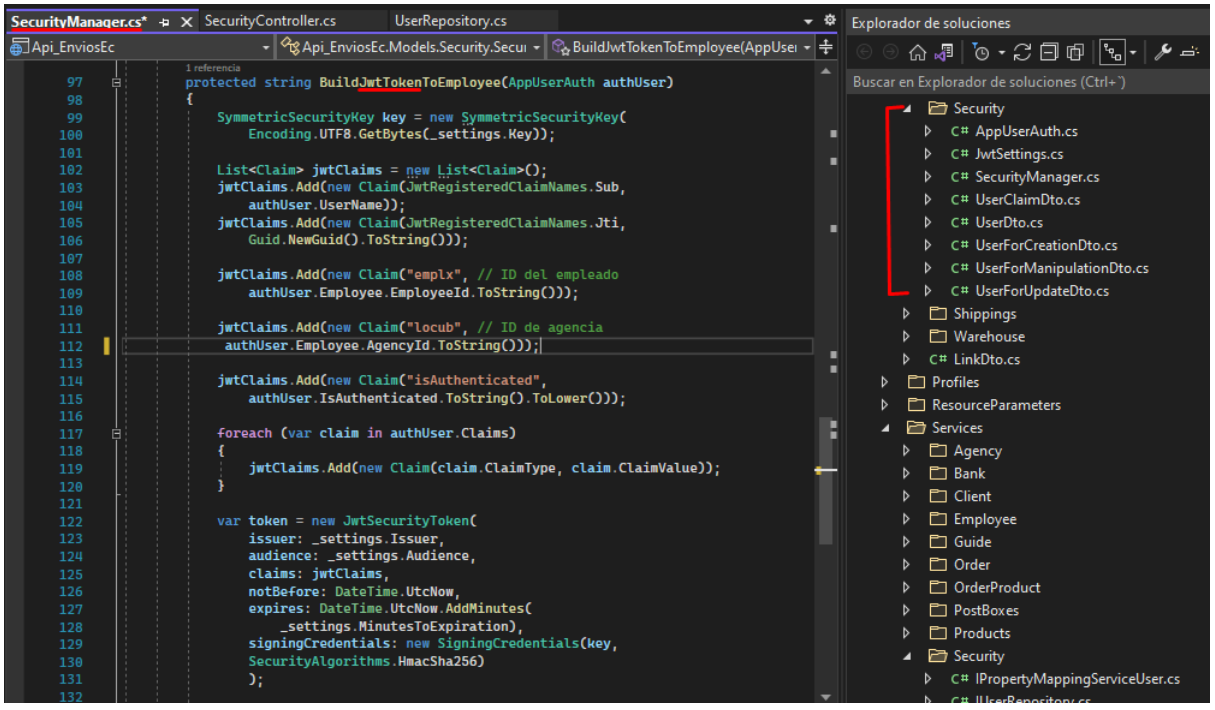


Figura 39: Autenticación y autorización

- Se han creado los repositorios de cada entidad junto con sus interfaces o contratos, tal como indica la figura 40.

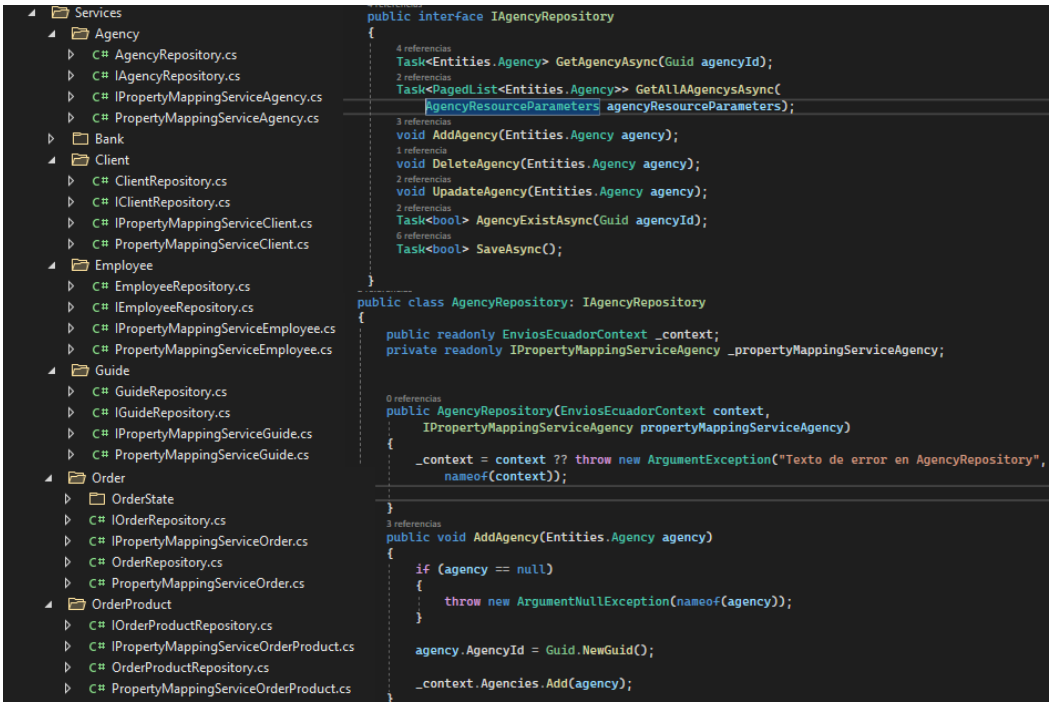


Figura 40: Creación de repositorios y contratos

El modelo de contrato definido en la figura 40 es usado para todos los contratos de la aplicación, al igual que su forma de implementar código en los repositorios.

2.5.1.5 Revisión del quinto sprint

A continuación se describe el detalle de los resultados obtenidos en el quinto sprint.

- Se implementa patrón scoped para todos los repositorios en función a sus contratos, mismos que conectan con los controladores de cada entidad esta configuración se realiza desde el startup de la aplicación como indica la figura 41.

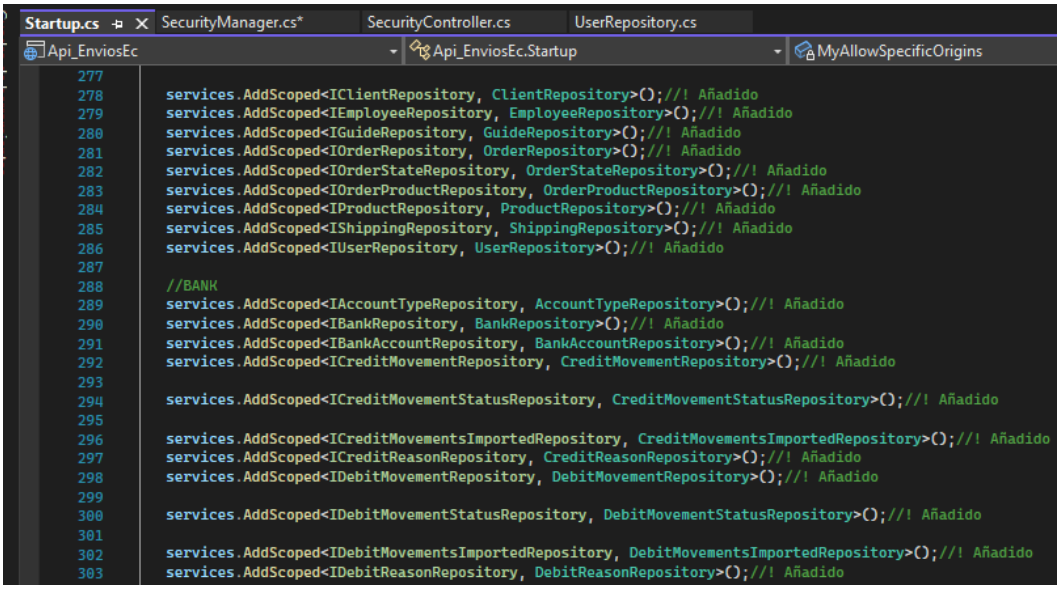


Figura 41: Implementación scoped

- Se Implementa transient para el servicio de mapeo con su respectivo contrato tal como se puede apreciar en la figura 42, mismo que se ocupan inyectan en los controladores de cada entidad.

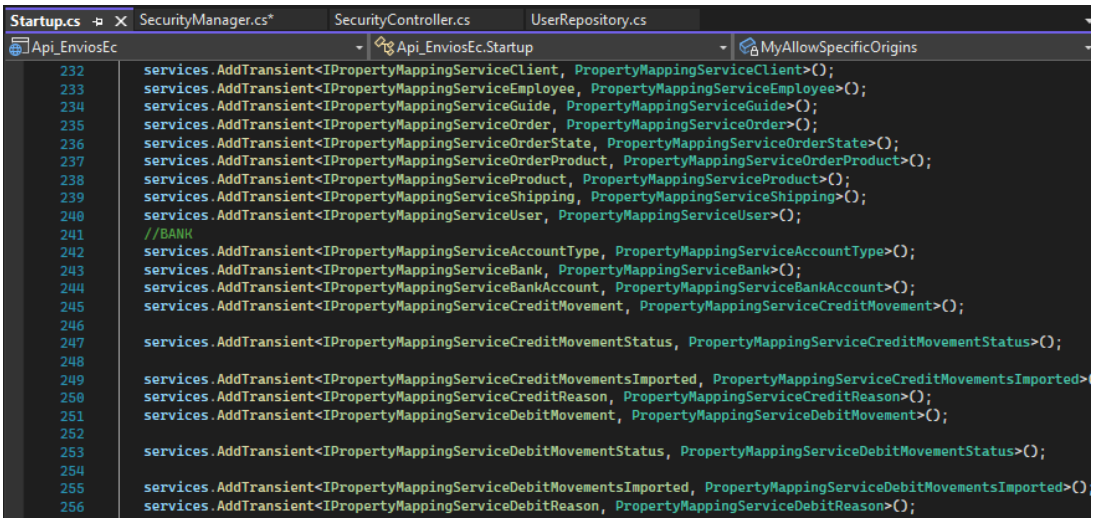


Figura 42: Implementación transient

- Se ha creado un disparador en la base de datos, aplicado a la tabla donde se guarda los registros que hacen referencia a movimientos bancarios importados desde el balance de una cuenta bancaria. Este disparador verifica si existe un registro en la tabla de depósitos que coincida con el número de transferencia y el valor declarado.

Figura 43: Disparador para verificación automática de depósitos

La figura 43 muestra el código del disparador aplicada a la tabla “Movimientos de Crédito Importados”, misma que cumple con la función de verificar automáticamente si un depósito registrado es válido.

- Se han creado las clases tipo controlador para cada entidad, misma que heredan de ControllerBase como indica la figura 44.

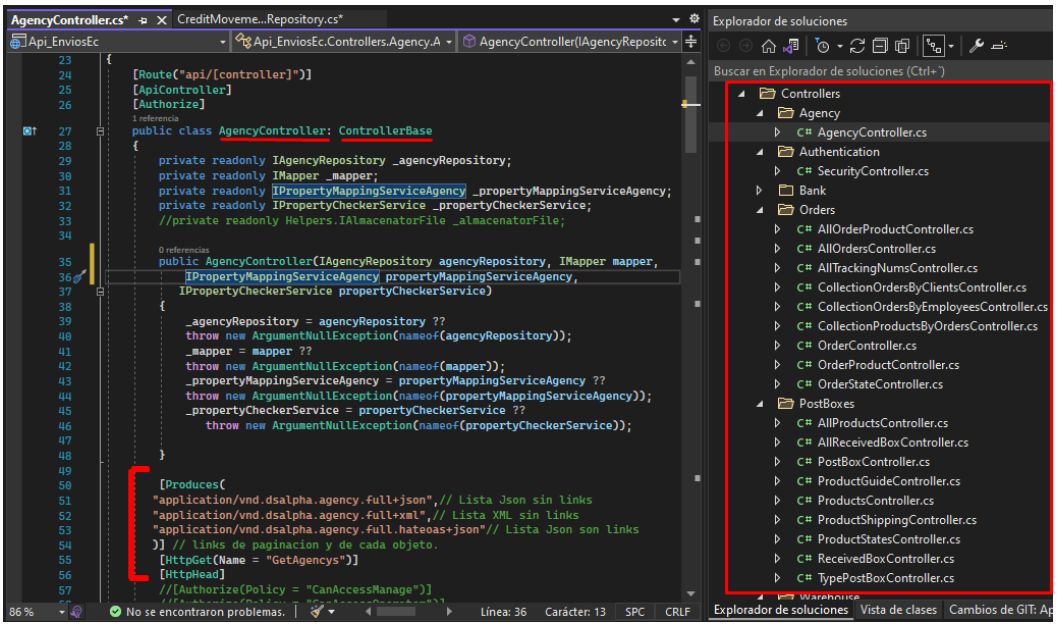


Figura 44: Creación de controladores

2.5.1.6 Revisión del sexto sprint y séptimo sprint

A continuación se describe el detalle de los resultados obtenidos en el sexto y séptimo sprint.

- Se ha creado un esquema completo de pruebas en Postman para comprobar el funcionamiento de los métodos creados para la API

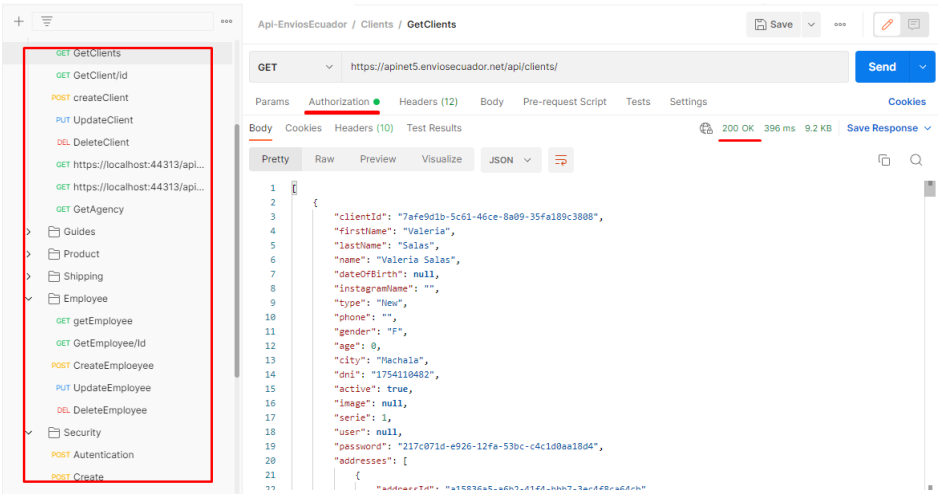


Figura 45: Test de API con Postman

Como se puede apreciar en la figura 45 desde la herramienta Postman se realizan las consultas apuntando a la ruta de acceso y el método de entrada, posteriormente se envía la solicitud a través del protocolo https incluyendo el token de autorización en la cabecera.

En la figura 45 podemos ver un marco de pruebas para cada punto de acceso a la API, esto representa el test de las actividades declaradas en el sexto y séptimo sprint de la aplicación.

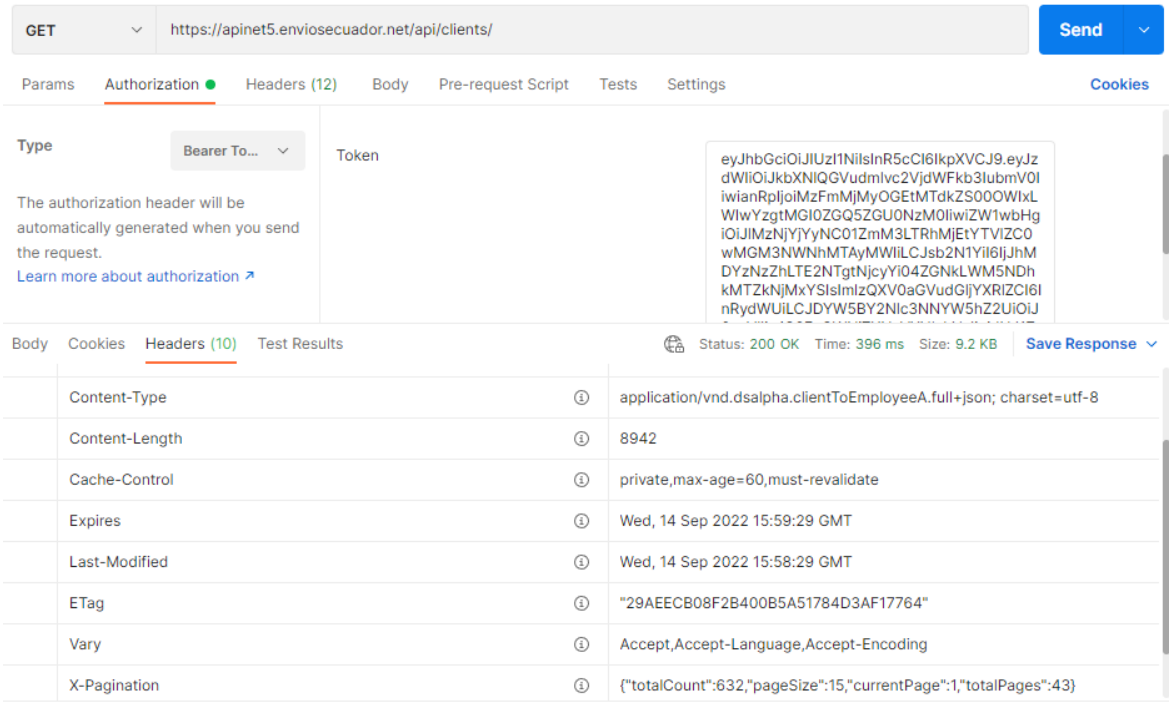


Figura 46: Test con token de autorización

En la figura 46 visualizamos el Bearer token que se incluye a la solicitud, y también se aprecia la cabecera de respuesta que el servidor envía; este especifica los parámetros de paginación, parámetros para el manejo de cache y conteo de registros.

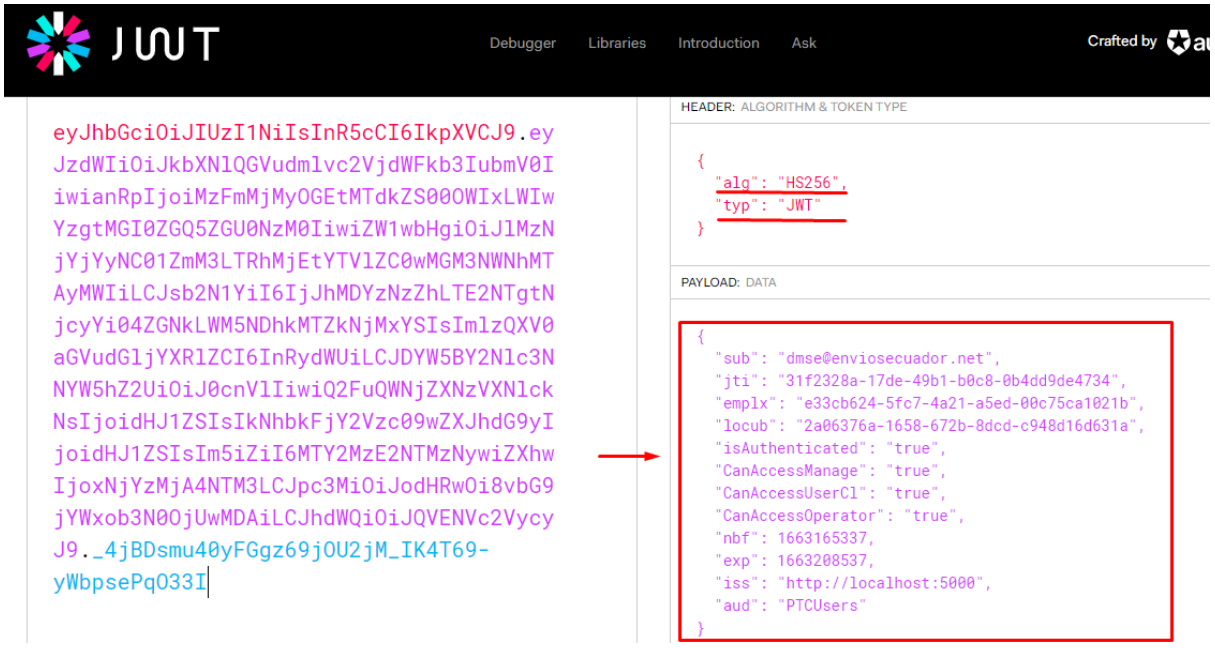


Figura 47: Contenido de token

La figura 47 denota una traducción a lenguaje natural del contenido que contiene un token, podemos apreciar la clase de algoritmo que usa el compilador para crear el código y la información codificada que lleva en su interior, las mismas que sirven para evaluar si cumple con las políticas del negocio implementadas en el sistema y el nivel de acceso a los datos que tiene el usuario.

```
services.AddAuthorization(cfg =>// especificamos las politicas de seguridad que tienen q ver con los claims
{
    cfg.AddPolicy("CanAccessOperator", p => // Nombre de la politica de seguridad
                        p.RequireClaim("CanAccessOperator", "true"));// Requiere q se cumpla esta caracteristica
    cfg.AddPolicy("CanAccessUserCl", p => // Nombre de la politica de seguridad
                        p.RequireClaim("CanAccessUserCl", "true"));// Requiere q se cumpla esta caracteristica
    cfg.AddPolicy("CanAccessManage", p => // Nombre de la politica de seguridad
                        p.RequireClaim("CanAccessManage", "true"));// Requiere q se cumpla esta caracteristica
});
```

Figura 48: Políticas del negocio

Como se puede evidenciar en la codificación plasmada de la figura 48, el servicio de autorización establece las políticas de acceso a los datos que debe cumplir un usuario previo a la comunicación con la capa de acceso a la información.

2.5.1.7 Revisión del octavo, noveno y décimo sprint

A continuación se describe el detalle de los resultados obtenidos en el octavo, noveno y décimo sprint.

- Establecer una hoja de estilos CSS estandarizada para la creación de todos los componentes web que se van a usar en el desarrollo.

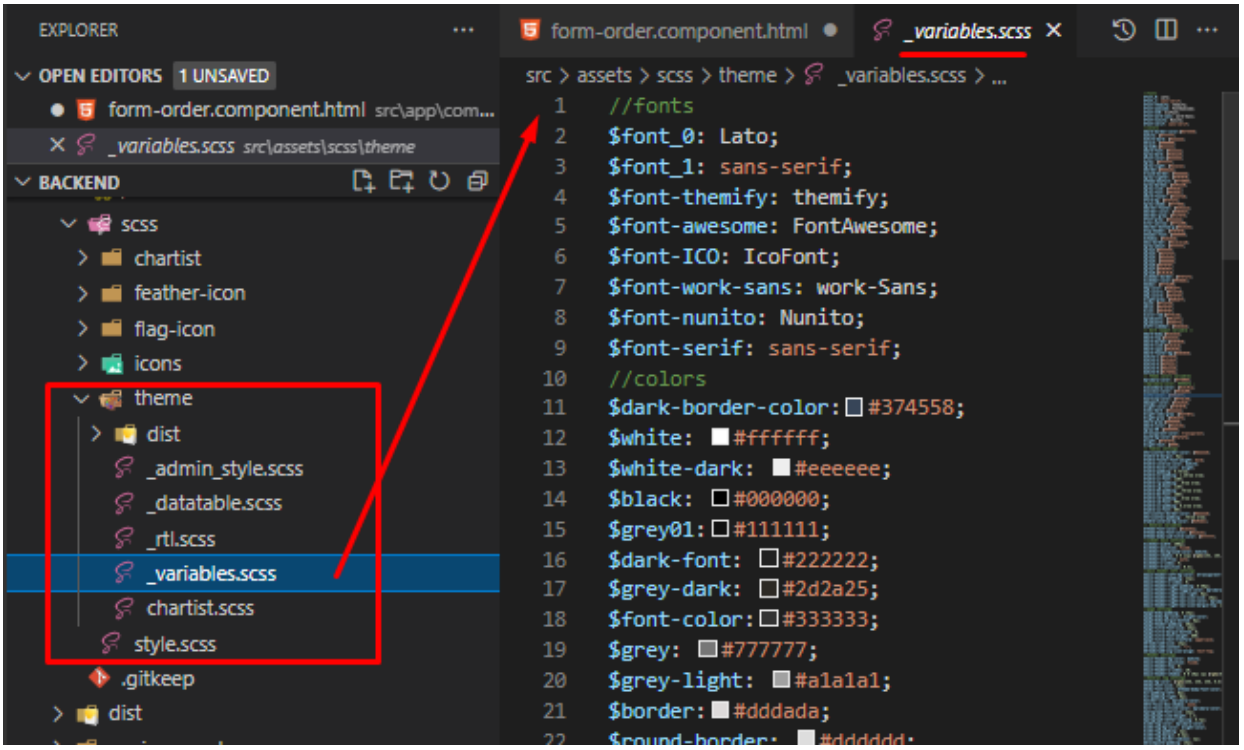


Figura 49:Contenido de estilo scss

Podemos ver en la figura 49 la carpeta creada para contener elementos de tipo scss, la cual establece estilos según el nombre de clases que lleva cada etiqueta declarada en los componentes HTML, desde esta colección de archivos se establecen las reglas de diseño responsive y se asegura una experiencia de usuario con usabilidad clara y concisa para todo el desarrollo de interfaces.

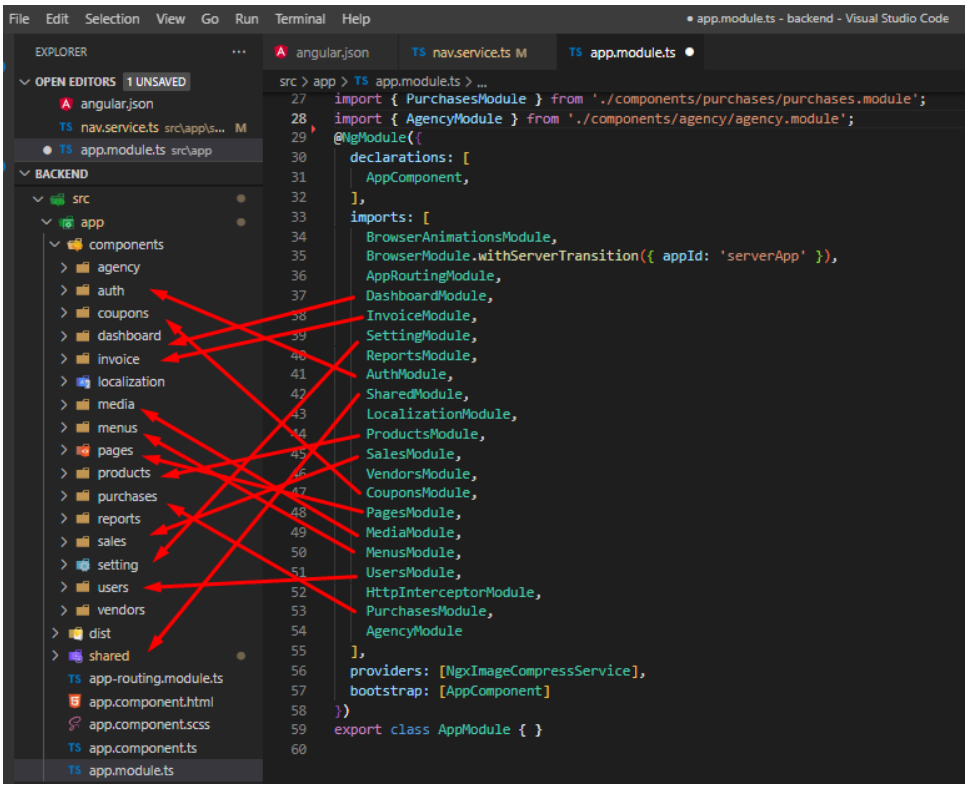


Figura 50: Declaración de módulos Angular

Manejando una arquitectura ordenada en el desarrollo de componentes Angular, se ha establecido módulos específicos que organicen sus dependencias de manera independiente en sus componentes. Por esta razón la figura 50 indica como cada módulo está relacionado con los componentes que se incluyan dentro de la carpeta a la que se hace referencia, cada carpeta contiene varios componentes como se muestra en la figura 51, estos componentes en conjunto crean una interfaz específica.

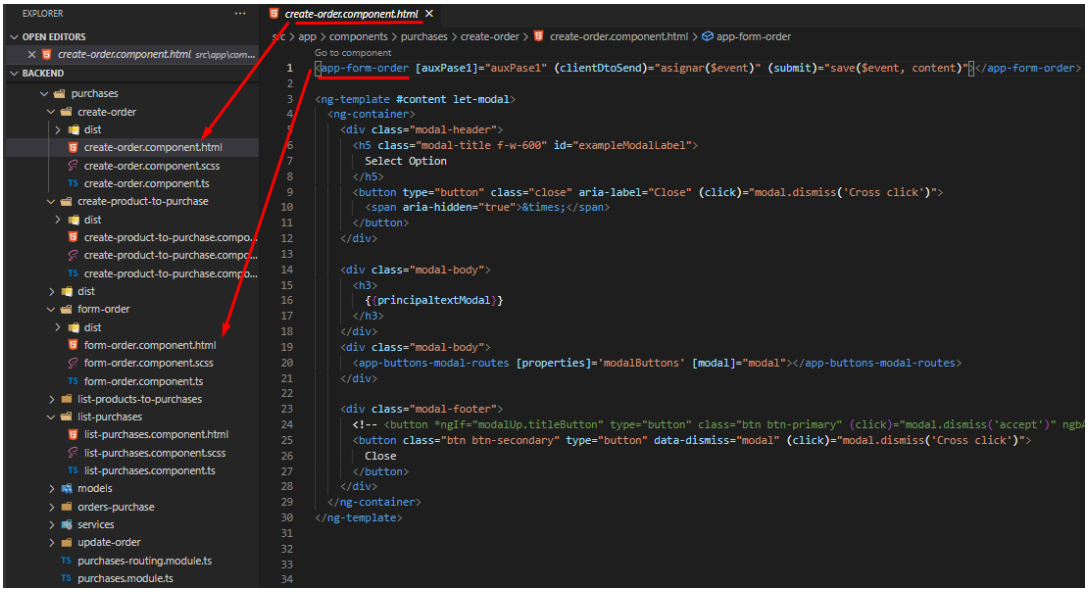


Figura 51: Componentes de módulos

Una vez desarrollados los componentes que satisfacen las necesidades de un módulo, se ha creado un componente para incluir sus referencias en el desarrollo de la interfaz, a fin de separar la lógica de la presentación. Como se puede apreciar en la figura 51 el

componente “Crear Orden” incluye al componente “formulario de orden”. de la misma manera el componente “Actualizar Orden” va a contener al componente “formulario de orden” pero la lógica que se especifica en el archivo TypeScript de cada componente va a ser diferente. Está mecanismo de trabajo nos permite modularizar y estandarizar el desarrollo de todas las interfaces del frontend.

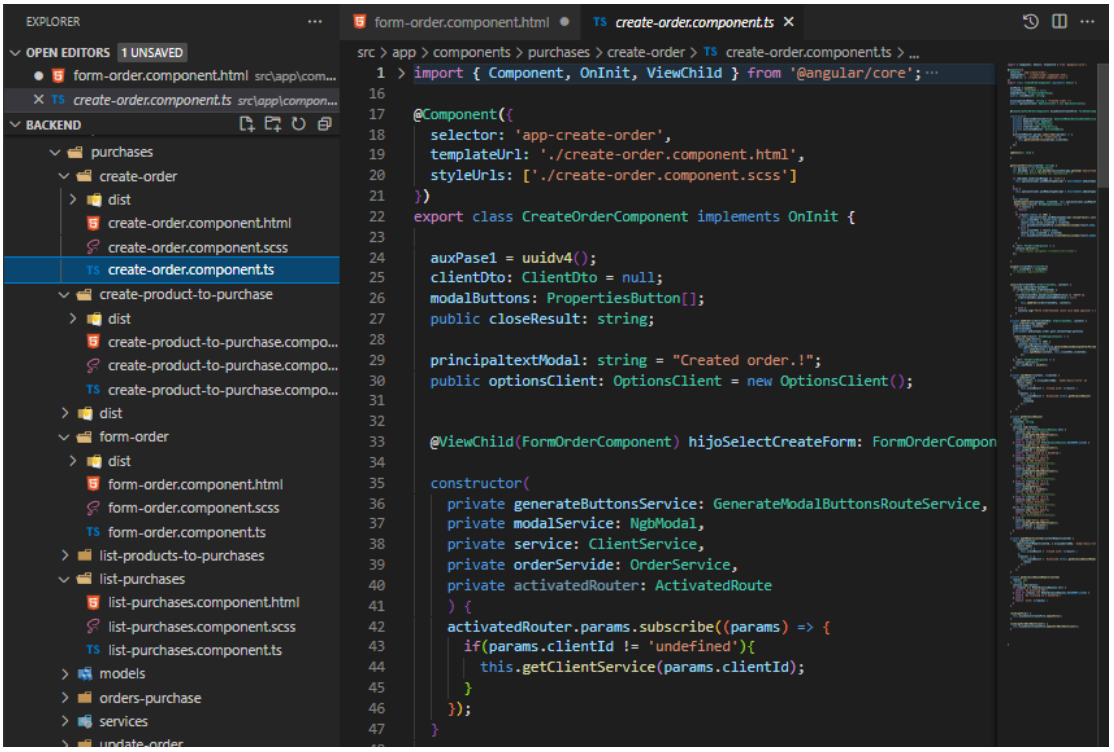


Figura 52: Typescript de componente Angular

- Se declara en todos los archivos TypeScript los componentes que infieren en la manipulación de datos y estilo como se ve en la figura 52, describiendo la ruta al archivo HTML y scss que se controla. Posterior a ello se inyectan los servicios que se van a utilizar a través del constructor de la clase, esto da paso a realizar la lógica que se considere necesario para dicho componente y la interacción de los eventos que incluya la interfaz, en la figura 53 se puede muestra el resultado visual que realiza la implementación descrita en las figuras 49 a 52.

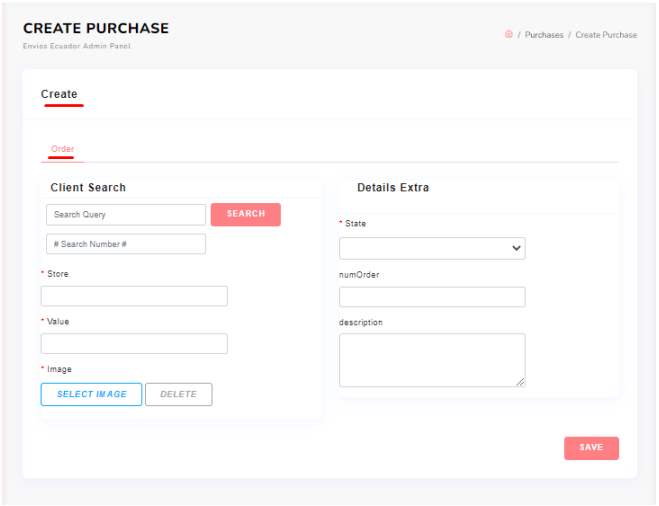


Figura 53: Formulario Angular

La figura 53 permite tener una visualización general de los formularios que se desarrollan para el envío de data al servidor. El estilo de este es general y se aplica a todos los formularios creados por la naturaleza de su diseño e implementación puede cambiar la ubicación de los objetos pero siempre se va a mantener el mismo diseño general para botones, campos de entrada, títulos, subtítulos y texto.

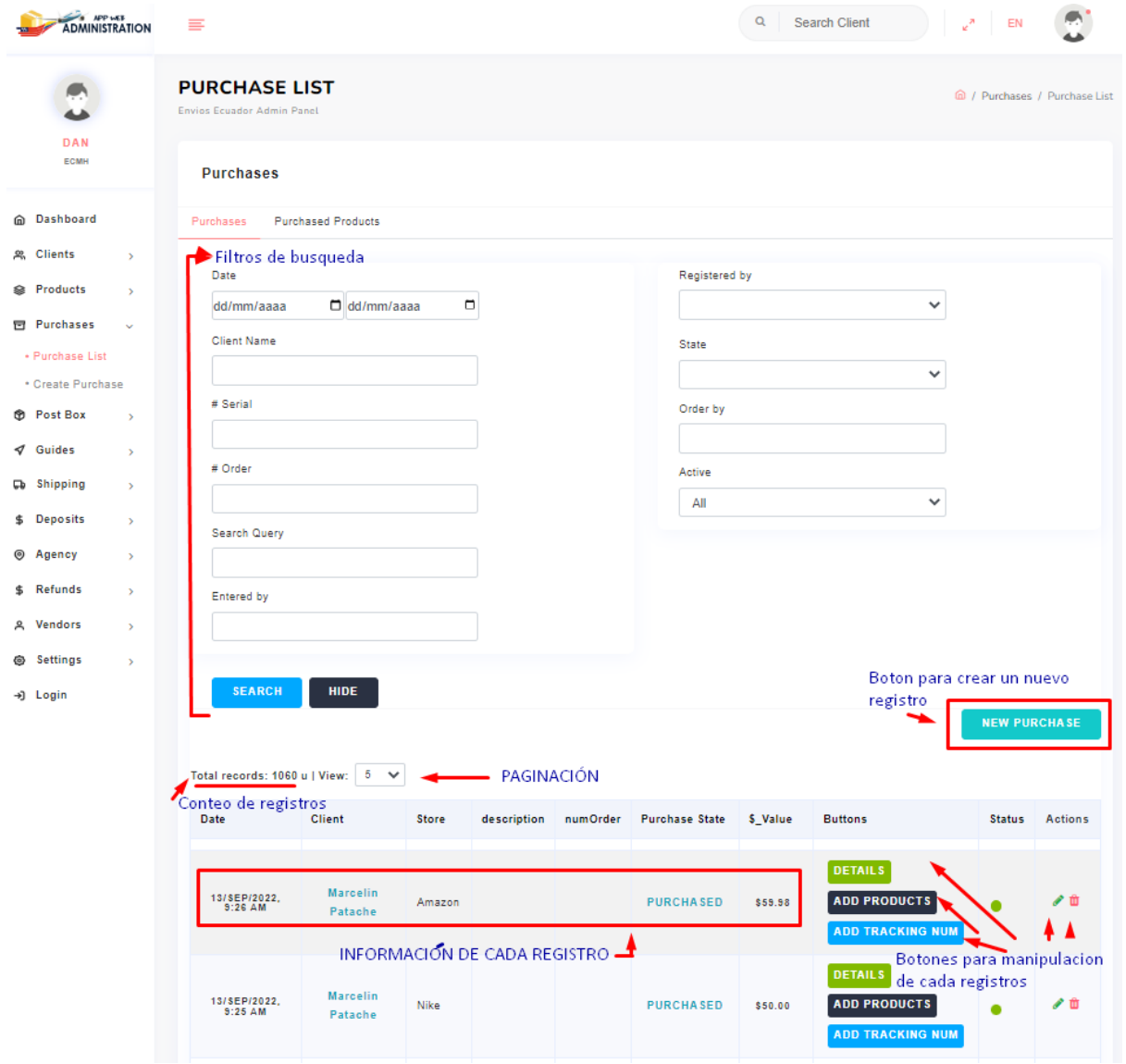


Figura 54: Estructura de interfaz Angular para listar

Para la interfaz de listar registros se desarrolló los componentes que se utilizan para la interacción de los eventos con el DOM, tales como se indica en la figura 54, filtros de búsqueda, botones de acceso, botones de manipulación de registros, elementos de paginación y la tabla donde se lista la información.

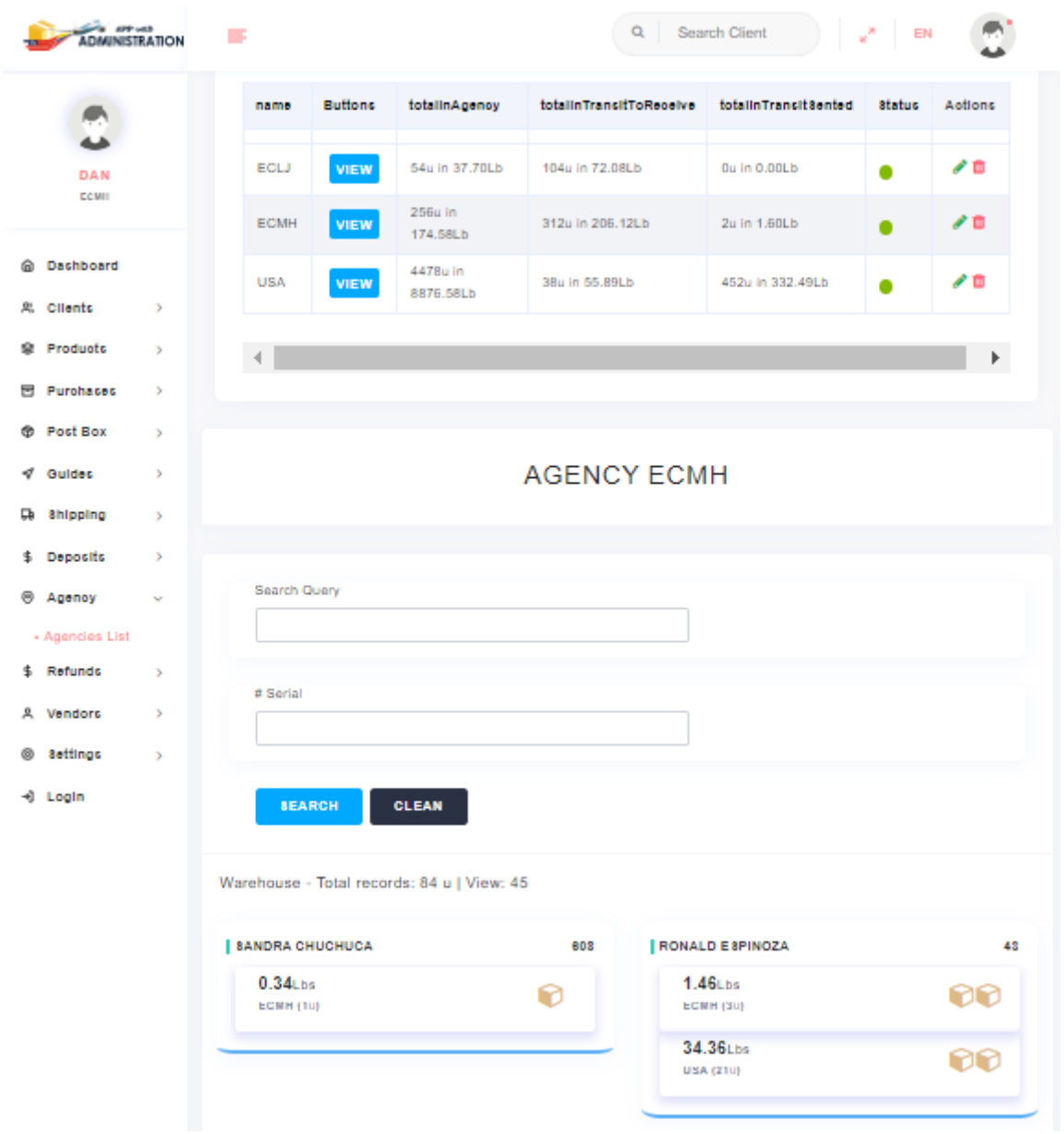


Figura 55: Interfaz Angular bodega de agencia

En la implementación de una interfaz para listar el contenido en bodega de una agencia se crearon los componentes que muestran dicha información por cliente a través de un arreglo multidimensional tal como se visualiza en la figura 55.

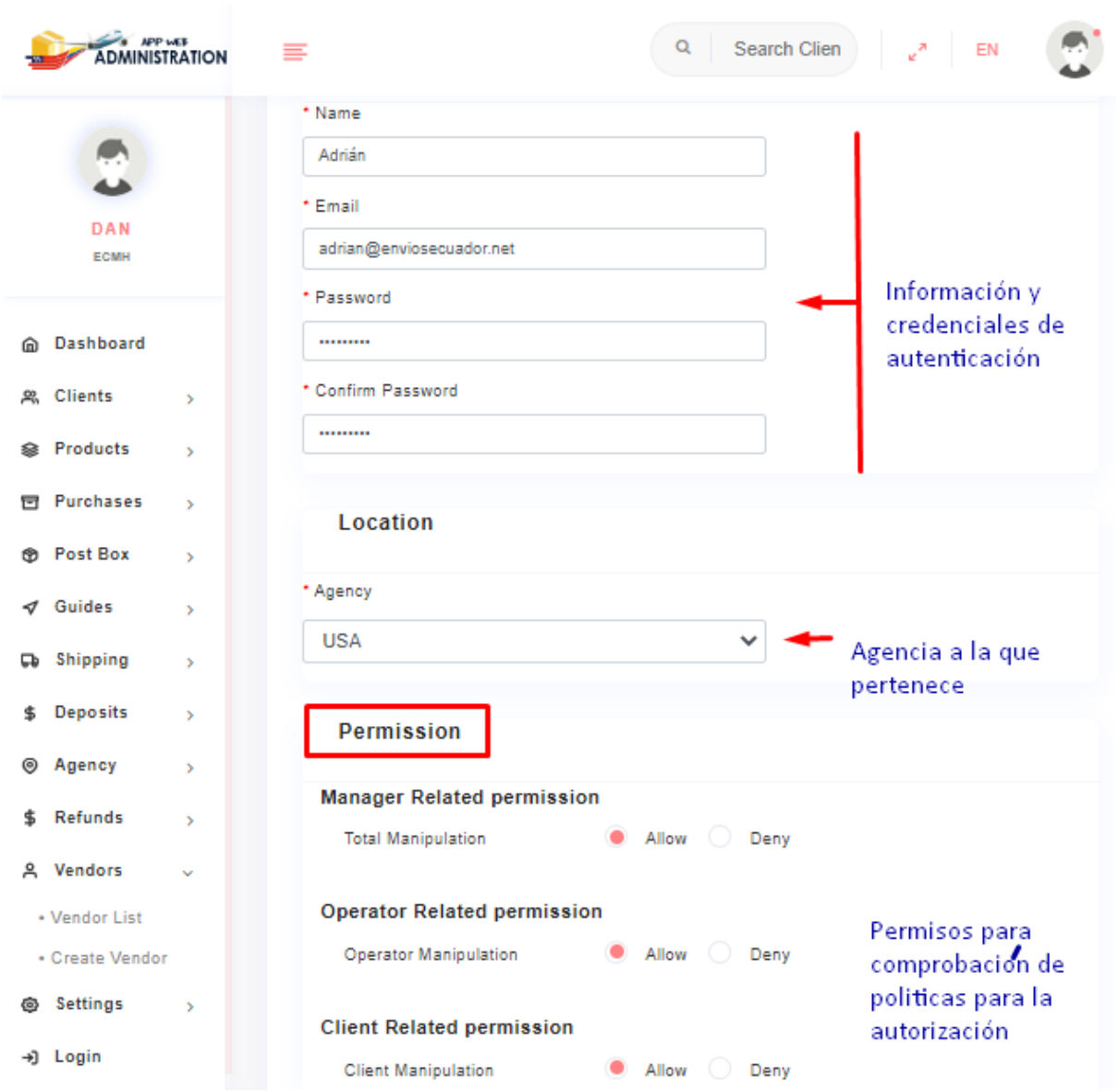


Figura 56: Interfaz Angular para administrar credenciales de autenticación y autorización

Para la administración de usuarios, permisos de autorización y credenciales de autenticación se ha creado una interfaz que oculta los caracteres del password, y manipula los permisos de usuario a través de botones de selección, tal como lo muestra la figura 56.

2.5.1.8 Revisión del décimo primer sprint

- Se desarrollaron los servicios que contiene la conexión a través del protocolo http para la comunicación con la API.

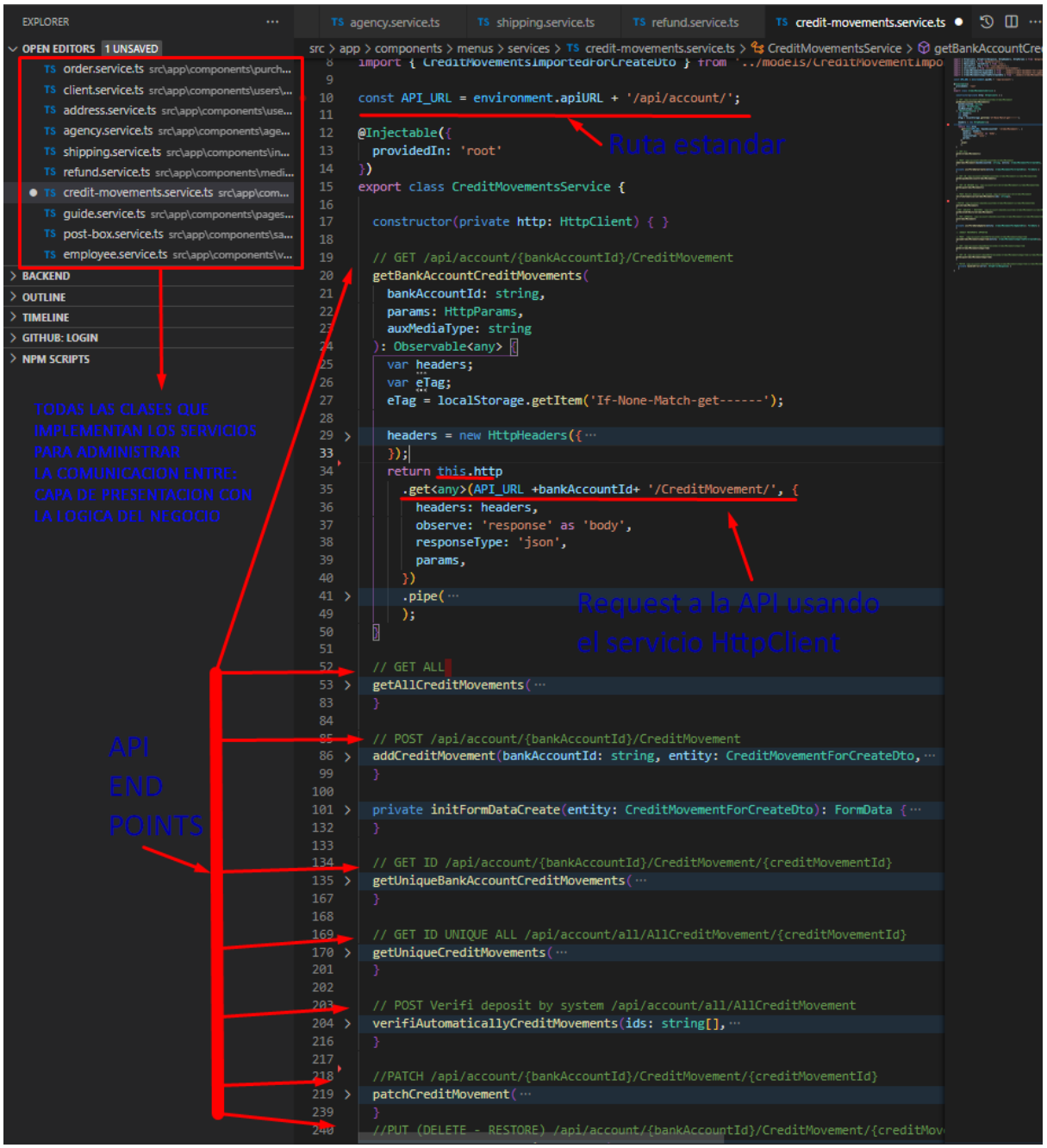


Figura 57: Servicios Angular para comunicación por protocolo http

Como se puede apreciar la figura 57 denota la estructura que sigue el desarrollo de todos los servicios para la comunicación de la capa de presentación con la lógica del negocio. En primer lugar se establece la ruta del endpoint de API al cual se hará referencia, posterior a esto se crea las funciones que realizaran solicitudes, mismas que deben incluir el método de comunicación, los parámetros de opciones para data y meta data que se envía y un observable que dispare una alerta cuando el servidor haya devuelto su respuesta o denegado el servicio.

- Se crearon los archivos a desplegar en el servidor para pasar nuestro proyecto a producción.

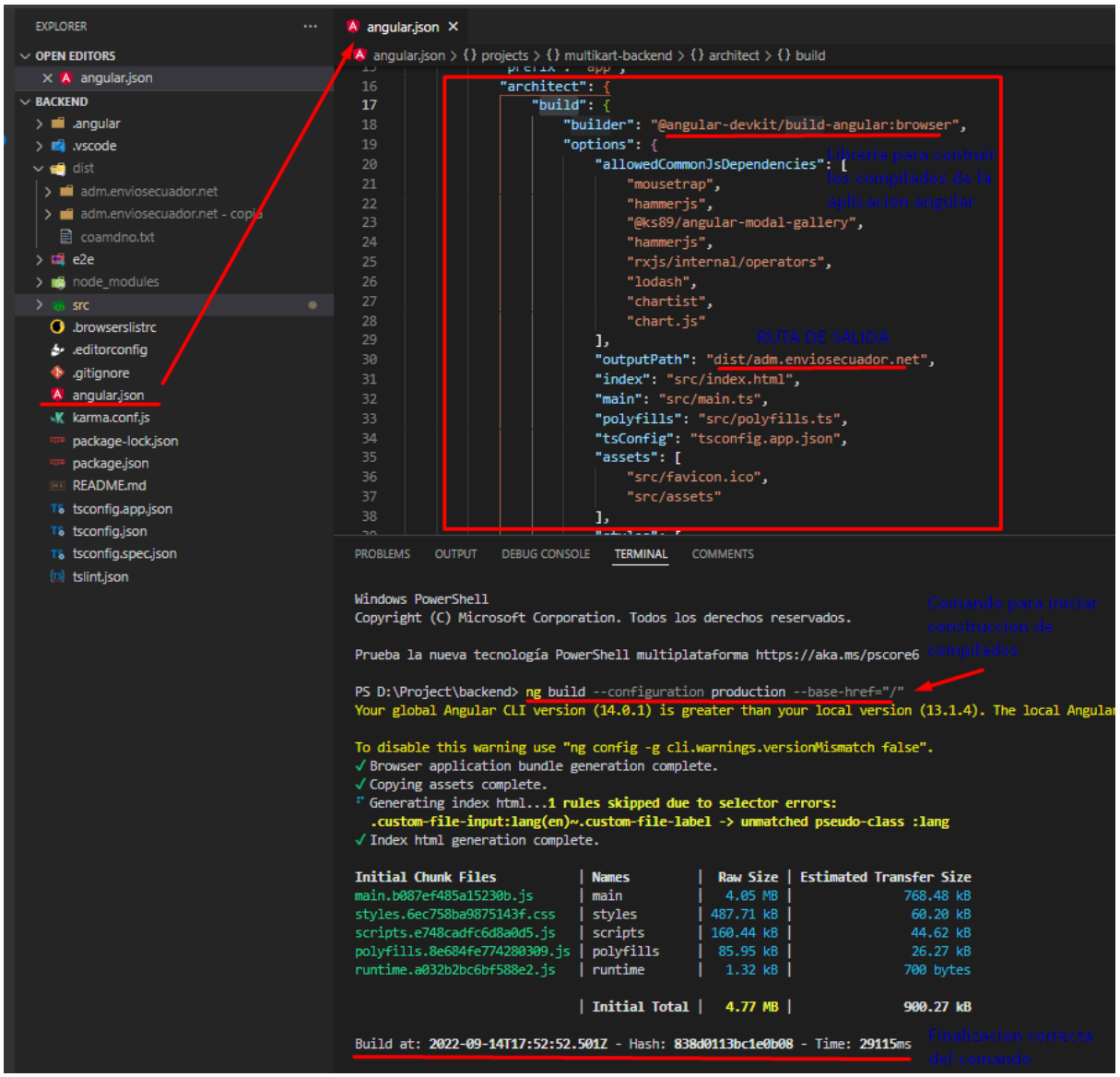


Figura 58: Configuración y generación de archivos Angular compilados para el servidor web

Se estableció la configuración del compilador para realizar las operaciones que generan los Archivos reléase para el despliegue de la aplicación tal como indica la figura 58.

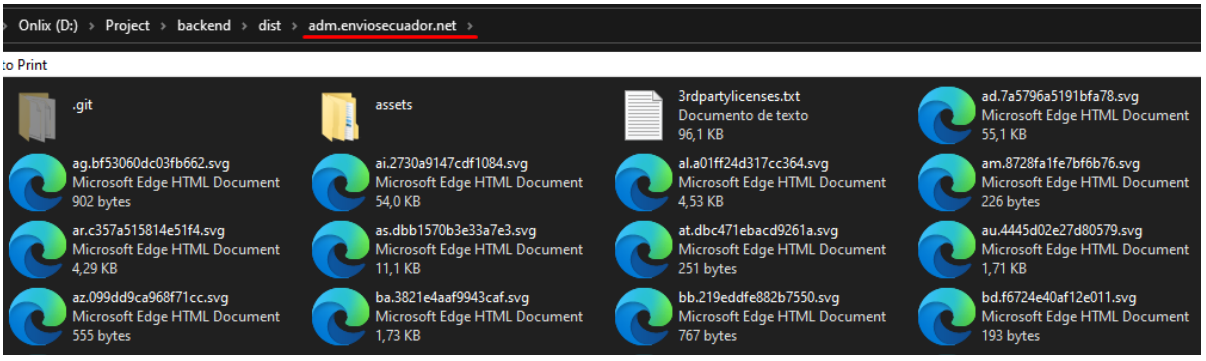


Figura 59: Archivos Angular compilados para el servidor web

La figura 59 muestra los archivos creados por el compilador de Angular para producción, mismos que deben ser cargados en el servidor web para dar por terminada la implementación de la capa de presentación.

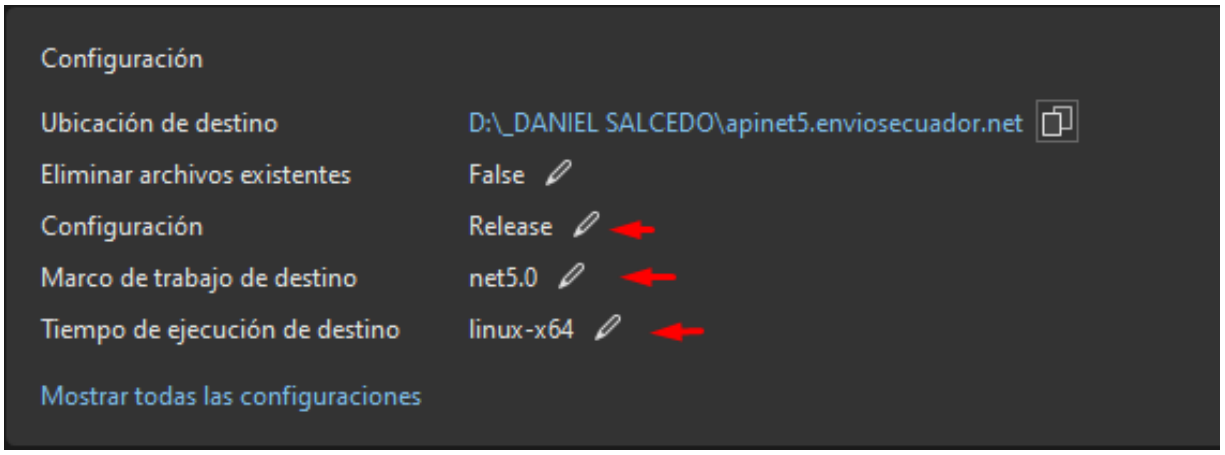


Figura 60: Configuración para la construcción de archivos compilados .NET 5

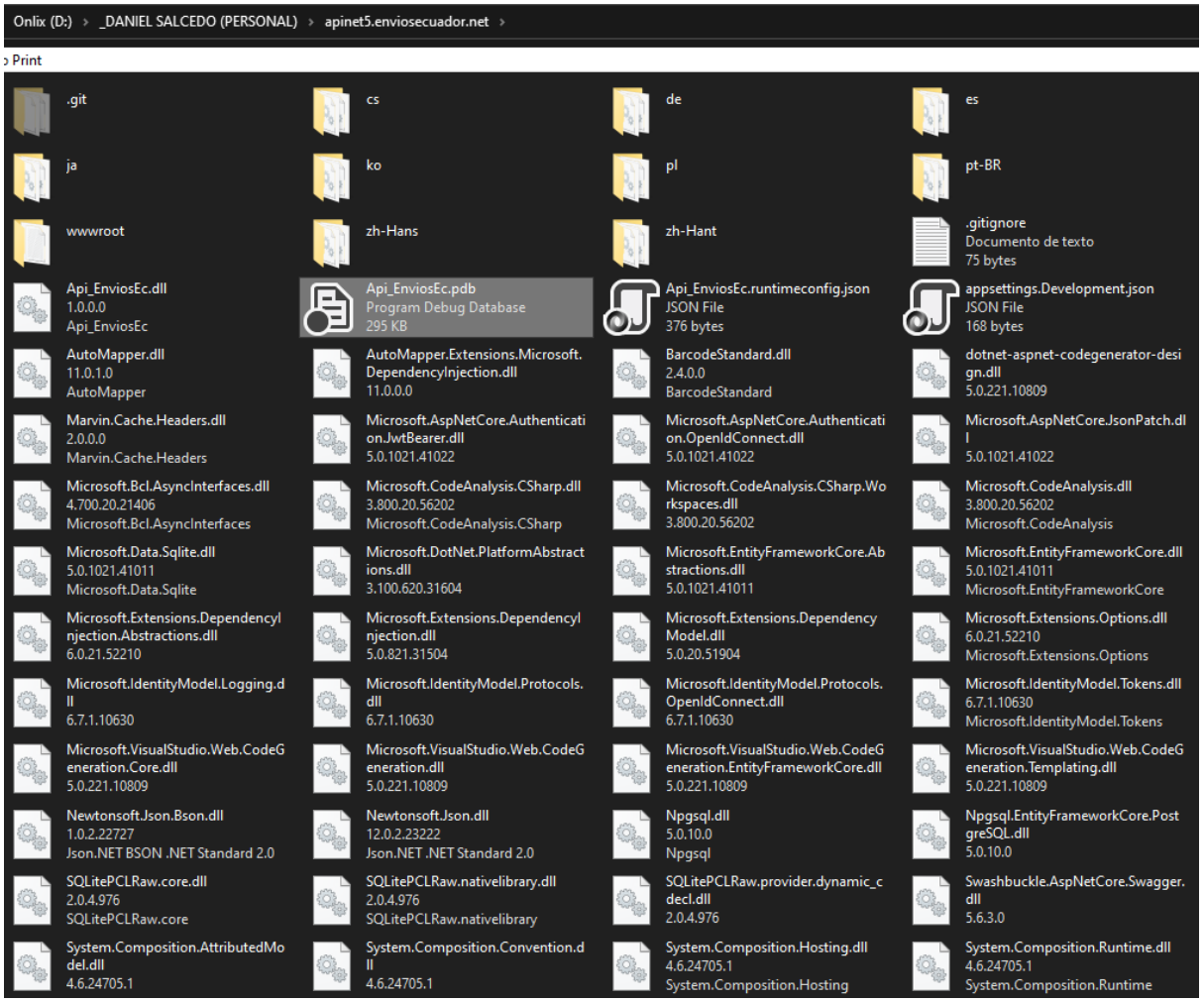


Figura 61: Archivos .NET 5 compilados para el servidor kestrel

La figura 61 muestra los archivos creados por el compilador de visual studio y Microsoft para producción, mismos que deben ser cargados en el servidor kestrel para dar por terminada la implementación de la lógica del negocio.

- Se creó la base de datos de producción, ejecutando el script SQL descrito en el anexo 2, a través de una conexión http entre el cliente PgAdmin y la base de datos del servidor, la conexión entre el cliente PgAdmin y el servidor se muestra en la figura 62.

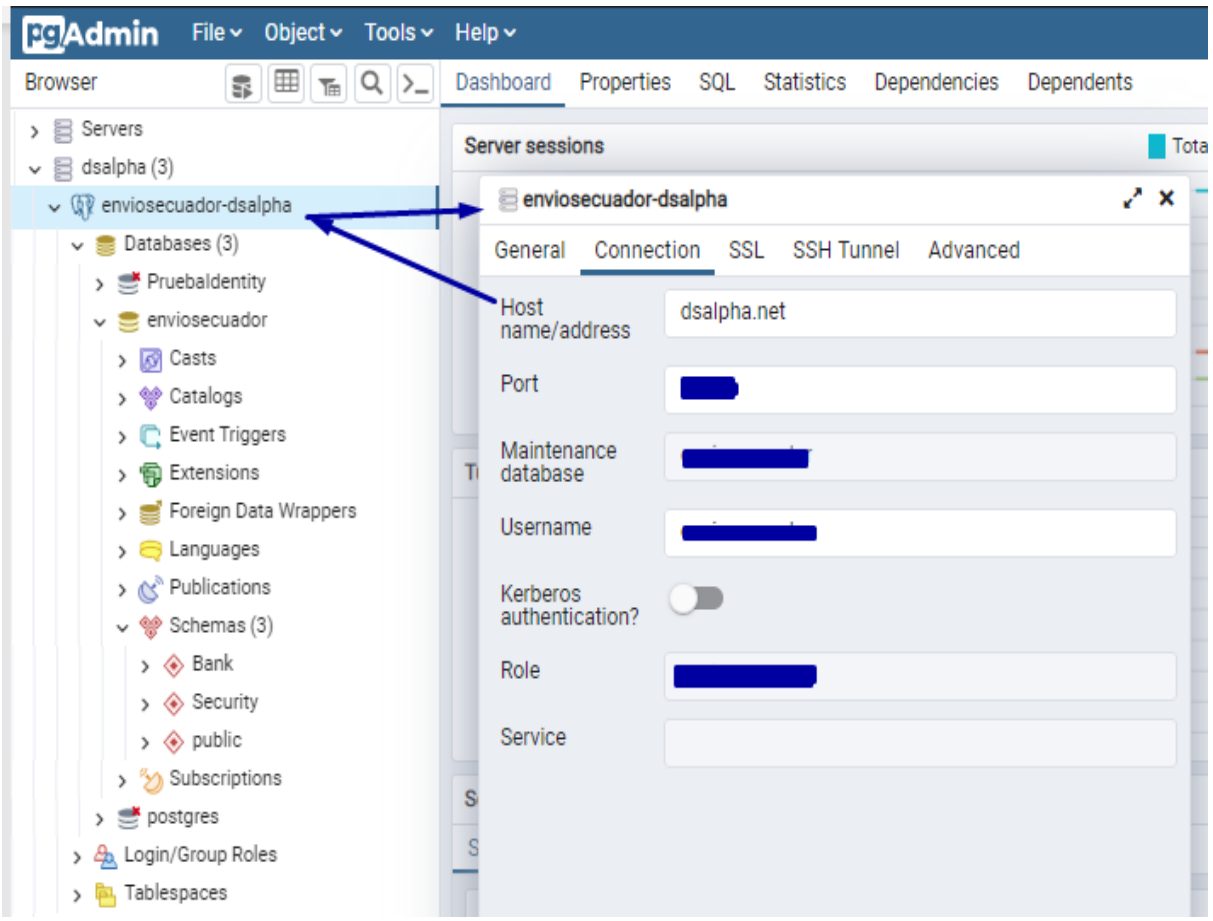


Figura 62: Conexión cliente - servidor PostgreSQL

Haciendo uso de Git y GitHub, se cargaron los archivos reléase del backend y frontend para la puesta a producción.

Comando utilizado para subir los archivos reléase a GitHub:

- git push origin (Desde la terminal local donde se encuentran los archivos compilados).

Comandos utilizados para cargar los archivos reléase al servidor:

- git fetch (Desde la terminal del servidor donde se desea cargar o actualizar los archivos compilados que previamente se subió a GitHub).

- git pull (Desde la terminal del servidor donde se desea cargar o actualizar los archivos compilados que previamente se subió a GitHub).

CAPÍTULO III.
EVALUACIÓN DEL PROTOTIPO.

3.1 PLAN DE EVALUACIÓN

La metodología Scrum es un marco de trabajo ágil que no describe ningún estándar o métrica de evaluación en su forma de trabajo, por la naturaleza ágil de su metodología, este centra todos los esfuerzos del equipo en cumplir con los requerimientos del sistema y se asegura su correcto funcionamiento mediante la revisión de todos los sprints planteados. Es por ello que el scrum master ha creado una tabla para realizar una evaluación objetiva con métricas porcentuales aplicada a las funcionalidades importantes del sistema, dicha tabla plantea verificar los resultados esperados y valora cada funcionalidad. El promedio de todas las valoraciones da como resultado el nivel de satisfacción que tiene el proyecto ante las expectativas planteadas.

3.2 RESULTADOS DE EVALUACIÓN

Toda esta etapa de evaluación se ha realizado con el proyecto en producción.

Tabla 28: Resultado de evaluación - Generación del token al autenticarse

Evaluación de funcionalidad del proyecto				
Funcionalidad 1:	Generación del token al autenticarse			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
En el login de la aplicación se ingresa las credenciales de autenticación (Usuario y contraseña)	Envío se data al servidor a través POST	100%	Se realiza la salida de data a través de un método POST	95%
	Status code 200	100%	El servidor responde un status code 200	
	Bearer Token	100%	Se genera un token JWT cuando las credenciales son correctas	
	Comunicación a través de https	100%	La solicitud se realiza a través del método POST	
	Datos del usuario autenticado	75%	La respuesta del servidor incluye los datos del usuario, agencia a la que pertenece y permisos. No se necesita retornar los permisos.	

La tabla 28 muestra el resultado de la evaluación a la funcionalidad 1, mismo que se evidencia en el anexo 3 y anexo 4 del documento.

Tabla 29: Resultado de evaluación funcionalidad 2- Denegar peticiones a usuarios no autorizados

Evaluación de funcionalidad del proyecto				
Funcionalidad 2:	Denegar peticiones a usuarios no autorizados			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza una petición, sin incluir el Bearer JWT a un end poin de la api que implementa métricas de autorización.	El servidor responde con un status code 401	100%	El servidor responde 401 Unauthorized	100%
	El servidor no devuelve data	100%	El servidor niega la entrada al servicio	

La tabla 29 muestra el resultado de la evaluación a la funcionalidad 2, mismo que se evidencia en el anexo 5 del documento.

Tabla 30: Resultado de evaluación - Gestionar permisos de clientes

Evaluación de funcionalidad del proyecto				
Funcionalidad 3:	Gestionar permisos de clientes			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza peticiones GET como cliente para revisar la información relacionada con su Id	Cuando el cliente tiene permiso para acceder, el servidor concede acceso al servicio	100%	Se devuelve data de la manera esperada, limitando la información	75%
	Cuando el cliente NO tiene permiso para acceder, el servidor niega el acceso al servicio	50%	El servidor kestrel niega las peticiones del usuario por no cumplir con las políticas del negocio, pero el servidor web le permite el acceso a la interfaz; El cliente no puede ver ninguna información del sistema pero debería no redirigir del login al home si no tiene permiso para extraer data del servidor.	

La tabla 30 muestra el resultado de la evaluación a la funcionalidad 3, mismo que se evidencia en el anexo 6, anexo 7 y anexo 8 del documento.

Tabla 31: Resultado de evaluación - Gestionar permisos de empleados

Evaluación de funcionalidad del proyecto				
Funcionalidad 4:	Gestionar permisos de empleados			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza peticiones al servidor con el permiso de manipulación total y manipulación de operador.	Con la manipulación total se espera tener todos los datos de clientes.	100%	Se cumple con las expectativas	100%
	Con la manipulación de tipo operador los datos como número telefónico, y red social se mantienen ocultos.	100%	Se cumple con las expectativas	

La tabla 31 muestra el resultado de la evaluación a la funcionalidad 4, mismo que se evidencia en el anexo 9, anexo 10 y anexo 11 del documento.

Tabla 32: Resultado de evaluación - Envío de token en el header de todas las peticiones

Evaluación de funcionalidad del proyecto				
Funcionalidad 5:	Envío de token en el header de todas las peticiones			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Asegurar la insercion del token en todas las peticiones del usuario	El token este presente en todas las solicitudes que realiza el cliente	75%	El token se insera en todas las solicitudes que realiza el cliente, sin embargo tambien realiza las peticiones cuando no existe un token, el servidor se encarga de negarle el servicio pero mejoraría el rendimiento si la solicitud sin token se negara en el cliente antes de lanzar la petición.	88%
	Que el token se inserte como cabecera de la solicitud	100%	Se cumple con la expectativa	

La tabla 32 muestra el resultado de la evaluación a la funcionalidad 5, mismo que se evidencia en el anexo 12 y 13 del documento.

Tabla 33: Resultado de evaluación - Respuestas en formato Json

Evaluación de funcionalidad del proyecto				
Funcionalidad 6:	Respuestas en formato Json			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza una petición GET indicando en el media type Accept que se espera una respuesta en formato JSON.	La data del servidor en formato JSON	100%	El servidor responde con la data esperada	100%
	El mismo media type que se incluye en la cabecera	100%	El media type no cambio en la cabecera de respuesta	
	Un mapeo correcto de las entidades anidadas en caso que existan relaciones en la solicitud	100%	Se mapean entidades sub relacionadas con normalidad	

La tabla 33 muestra el resultado de la evaluación a la funcionalidad 6, mismo que se evidencia en el anexo 14 y anexo 15 del documento.

Tabla 34: Resultado de evaluación - Respuestas en formato XML

Evaluación de funcionalidad del proyecto				
Funcionalidad 7:	Respuestas en formato XML			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza una petición GET indicando en el media type Accept que se espera una respuesta en formato XML.	La data del servidor en formato XML	100%	El servidor responde con la data esperada	67%
	El mismo media type que se incluye en la cabecera	100%	El media type no cambio en la cabecera de respuesta	
	Un mapeo correcto de las entidades anidadas en caso que existan relaciones en la solicitud	0%	No se mapean entidades sub relacionadas, el servidor dispara una excepción cuando una entidad contiene a otra entidad dentro.	

La tabla 34 muestra el resultado de la evaluación a la funcionalidad 7, mismo que se evidencia en el anexo 16 del documento.

Tabla 35: Resultado de evaluación - Conexión de capa presentación con capa de dominio

Evaluación de funcionalidad del proyecto				
Funcionalidad 8:	Conexión de capa presentación con capa de dominio			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se declara una ruta de acceso predeterminada en la capa de presentación para la comunicación con la API. Aislando el acceso a directo a la capa de acceso a los datos.	El cliente no conoce sobre credenciales de acceso al servidor de base de datos.	100%	Se cumple con la expectativa	100%
	El cliente desconoce la arquitectura del sistema que accede a la base de datos	100%	Se cumple con la expectativa	
	Se indica una ruta de acceso a través del protocolo https al host donde se ejecuta la API	100%	Se establece la ruta en el entorno de producción de la capa de aplicación indicando el protocolo https seguido de la dirección host donde se ejecuta el servidor kestrel	

La tabla 35 muestra el resultado de la evaluación a la funcionalidad 8, mismo que se evidencia en el anexo 17 y 18 del documento.

Tabla 36: Resultado de evaluación - Conexión de capa dominio con capa de acceso a datos

Evaluación de funcionalidad del proyecto				
Funcionalidad 9:	Conexión de capa dominio con capa de acceso a datos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se especifica en la programación de la capa de aplicación las credenciales de acceso a la base de datos.	Las credenciales de acceso a la base de datos están aisladas del cliente	100%	Se cumple con la expectativa	100%
	Las credenciales de acceso son manejadas por inyección de dependencia con entity framework core	100%	Se cumple con la expectativa	

La tabla 36 muestra el resultado de la evaluación a la funcionalidad 9, mismo que se evidencia en el anexo 19 del documento.

Tabla 37: Resultado de evaluación - Envío de archivo tipo File

Evaluación de funcionalidad del proyecto				
Funcionalidad 10:	Envío de archivo tipo File			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se carga un archivo desde la interfaz para ser enviado al servidor.	Se cargan archivos en la interfaz del cliente	50%	El cliente solo permite seleccionar un archivo a la vez. La carga masiva no está implementada.	90%
	Se transporta el archivo al servidor a través de una petición POST	100%	Se cumple con la expectativa	
	El servidor recibe un archivo binario y lo sube a la nube	100%	Se cumple con la expectativa	
	El servidor guarda en la base de datos la ruta donde almacenó el archivo	100%	Se cumple con la expectativa	
	El servidor responde con la ruta de acceso a la imagen almacenada en la nube	100%	Se cumple con la expectativa	

La tabla 37 muestra el resultado de la evaluación a la funcionalidad 10, mismo que se evidencia en el anexo 20, anexo 21 y anexo 22 del documento.

Tabla 38: Resultado de evaluación - Paginación en listado de registros

Evaluación de funcionalidad del proyecto				
Funcionalidad 11:	Paginación en listado de registros			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se realiza una petición GET que devuelve una colección de objetos.	El servidor hace una consulta limitando los registros que se pide a la base de datos, ya sea que el cliente lo especifique o no	100%	Se cumple con la expectativa	100%
	El servidor responde con una colección de datos limitada	100%	Se cumple con la expectativa	
	La cabecera de respuesta del servidor indica los registros existentes, los que entrega y los que se pueden solicitar según la página que se indica en el cliente.	100%	Se cumple con la expectativa	

La tabla 38 muestra el resultado de la evaluación a la funcionalidad 11, mismo que se evidencia en el anexo 23 y anexo 24 del documento.

Tabla 39: Resultado de evaluación - Filtros de búsqueda

Evaluación de funcionalidad del proyecto				
Funcionalidad 12:	Filtros de búsqueda			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Las peticiones incluyen parámetros de búsqueda.	El frontend tiene la facilidad de indicar los filtros que se desea aplicar	100%	Se cumple con la expectativa	100%
	El backend tiene la validación de los parámetros que se aceptan	100%	Se cumple con la expectativa	
	Los parámetros de filtro forman parte de la ruta cuando se realiza una petición.	100%	Se cumple con la expectativa	

La tabla 39 muestra el resultado de la evaluación a la funcionalidad 12, mismo que se evidencia en el anexo 25 y anexo 26 del documento.

Tabla 40: Resultado de evaluación - Carga masiva de movimientos bancarios

Evaluación de funcionalidad del proyecto				
Funcionalidad 13:	Carga masiva de movimientos bancarios			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Desde el frontend se selecciona un archivo .csv o .xml para realizar la importación de data. Este archivo debe ser un estado de movimientos bancarios (Banco Pichincha, Guayaquil y Loja)	El sistema filtra solo los movimientos de crédito para realizar la carga de datos	100%	El sistema descarta los movimientos que sean de débito.	80%
	El servidor carga los movimientos de cuenta generados por banco Pichincha	100%	Se cumple con la expectativa	
	El servidor carga los movimientos de cuenta generados por banco de Loja	100%	Se cumple con la expectativa	
	El servidor carga los movimientos de cuenta generados por banco Guayaquil	0%	No se cumple con este requerimiento, dado que el formato de banco guayaquil en sus reportes carece de una estructura sistemática.	
	El sistema descarta las transacciones que ya han sido cargadas antes	100%	Se cumple con la expectativa	

La tabla 40 muestra el resultado de la evaluación a la funcionalidad 13, mismo que se evidencia en el anexo 27 y anexo 28 del documento.

Tabla 41: Resultado de evaluación - Verificación automática de depósitos

Evaluación de funcionalidad del proyecto				
Funcionalidad 14:	Verificación automática de depósitos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Al cargar un estado de cuenta, se realiza la verificación de todos los depósitos registrados.	El sistema realiza una verificación del valor depositado y número de documento	100%	Se cumple con la expectativa	100%
	El sistema ingresa un texto en el registro, indicando que la verificación fue realizada por el sistema	100%	Se cumple con la expectativa	
	La base de datos se encarga de la verificación automática cuando se cargan registros nuevos.	100%	Se establece un disparador que cumple con la expectativa.	

La tabla 41 muestra el resultado de la evaluación a la funcionalidad 14, mismo que se evidencia en el anexo 29 del documento.

Tabla 42: Resultado de evaluación - Creación de débito por compras

Evaluación de funcionalidad del proyecto				
Funcionalidad 15:	Creación de debito por compras			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se debe crear un registro de débito a la cuenta del cliente, siempre que se genere una compra al cliente.	El sistema crea un débito cuando se hace una compra.	100%	Se han creado 1 disparador en la base de datos para registrar el débito a la cuenta de manera automática	100%
	El sistema borra dicho débito cuando se elimina el registro de compra.	100%	Se han creado 1 disparador en la base de datos para borrar el débito a la cuenta de manera automática	
	El sistema actualiza el valor de la cuenta, cuando se modifica el valor de la compra	100%	Se han creado 1 disparador en la base de datos para manejar el débito a la cuenta de manera automática	

La tabla 42 muestra el resultado de la evaluación a la funcionalidad 15, mismo que se evidencia en el anexo 30 y anexo 31 del documento.

Tabla 43: Resultado de evaluación - Creación de débito por despachos

Evaluación de funcionalidad del proyecto				
Funcionalidad 16:	Creación de debito por despachos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se debe crear un registro de débito a la cuenta del cliente, siempre que se genera un despacho de encomiendas.	El sistema crea un débito cuando se realiza una entrega o envío de encomienda.	100%	Se han creado 1 disparador en la base de datos para registrar el débito a la cuenta de manera automática	100%
	El sistema borra dicho débito cuando se elimina la entrega.	100%	Se han creado 1 disparador en la base de datos para borrar el débito a la cuenta de manera automática	
	El sistema actualiza el valor de la cuenta, cuando se modifica el valor por el servicio prestado	100%	Se han creado 1 disparador en la base de datos para manejar el débito a la cuenta de manera automática	

La tabla 43 muestra el resultado de la evaluación a la funcionalidad 16, mismo que se evidencia en el anexo 32 del documento.

Tabla 44: Resultado de evaluación - Creación de débitos por reembolsos

Evaluación de funcionalidad del proyecto				
Funcionalidad 17:	Creación de débitos por reembolsos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se debe crear un registro de débito a la cuenta del cliente, siempre que se realice un reembolso de dinero a su cuenta personal.	El sistema crea un débito cuando se confirma un reembolso emitido.	100%	Se han creado 1 disparador en la base de datos para registrar el débito a la cuenta de manera automática	100%
	El sistema borra dicho débito cuando se elimina el reembolso.	100%	Se han creado 1 disparador en la base de datos para borrar el débito a la cuenta de manera automática	
	El sistema actualiza el valor de la cuenta, cuando se modifica el reembolso	100%	Se han creado 1 disparador en la base de datos para manejar el débito a la cuenta de manera automática	

La tabla 44 muestra el resultado de la evaluación a la funcionalidad 17, mismo que se evidencia en el anexo 33 del documento.

Tabla 45: Resultado de evaluación - Creación de créditos por depósitos

Evaluación de funcionalidad del proyecto				
Funcionalidad 18:	Creación de créditos por depósitos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se debe crear un registro de crédito a la cuenta del cliente, siempre que se verifique un depósito.	Se crea un registro en la cuenta del cliente cuando se verifican los depósitos que ha ingresado	100%	Se han creado 1 disparador en la base de datos para registrar la acreditación a la cuenta de manera automática	100%
	Se elimina el registro de la cuenta cuando se elimina la verificación del depósito.	100%	Se han creado 1 disparador en la base de datos para borrar la acreditación a la cuenta de manera automática	
	Se puede manipular los datos de un depósito siempre que este no haya sido verificado.	100%	Se han creado 1 disparador en la base de datos para manejar la acreditación a la cuenta de manera automática	

La tabla 45 muestra el resultado de la evaluación a la funcionalidad 18, mismo que se evidencia en el anexo 34 del documento.

Tabla 46: Resultado de evaluación - Perfil de cliente con sus movimientos del negocio

Evaluación de funcionalidad del proyecto				
Funcionalidad 19:	Perfil de cliente con sus movimientos del negocio			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
La interfaz tiene un componente que indica todos los registros de un cliente relacionados con el negocio.	Se listan todos los movimientos de crédito y débito del cliente, así como su balance de cuenta	100%	Se cumple con la expectativa	100%
	Se lista todas las compras que ha realizado el cliente	100%	Se cumple con la expectativa	
	Se lista todas las encomiendas y productos recibidos en el casillero postal del cliente	100%	Se cumple con la expectativa	
	Se lista todos los depósitos que ha realizado el cliente.	100%	Se cumple con la expectativa	
	Se lista todos los despachos que se le ha realizado al cliente.	100%	Se cumple con la expectativa	

La tabla 46 muestra el resultado de la evaluación a la funcionalidad 19, mismo que se evidencia en el anexo 35 del documento.

Tabla 47: Resultado de evaluación - Funcionalidad de accesos rápidos en página de inicio

Evaluación de funcionalidad del proyecto				
Funcionalidad 20:	Funcionalidad de accesos rápidos en página de inicio			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
El tablero inicial para el usuario que inicia sesión debe tener accesos rápidos para las funciones las recurrentes	Botón para agregar nuevo cliente	100%	Se cumple con la expectativa	100%
	Botón para recibir una encomienda en el casillero postal	100%	Se cumple con la expectativa	
	Botón para agregar un nuevo depósito	100%	Se cumple con la expectativa	
	Botón para registrar una compra.	100%	Se cumple con la expectativa	

La tabla 47 muestra el resultado de la evaluación a la funcionalidad 20, mismo que se evidencia en el anexo 36 del documento.

Tabla 48: Resultado de evaluación - Vista de bodega por cada agencia

Evaluación de funcionalidad del proyecto				
Funcionalidad 21:	Vista de bodega por cada agencia			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se debe listar el contenido de cada agencia en sus bodegas.	Existe un indicador de lo que se tiene en agencia	100%	Se puede visualizar el total de productos y el peso total en libras de lo que se tiene en bodega.	60%
	Existe un indicador de lo que se espera recibir en una agencia	100%	Se puede visualizar el total de productos y el peso total en libras de lo que se espera recibir en una agencia.	
	Existe un indicador de lo que se ha enviado desde una agencia y aun no ha sido recibido.	100%	Se puede visualizar el total de productos y el peso total en libras de lo que se ha enviado a otra agencia y aún no ha sido recibido.	
	Funcionamiento de botón para modificar los datos de una agencia	0%	No se ha implementado el evento	
	Funcionamiento de botón para borrar una agencia	0%	No se ha implementado el evento	

La tabla 48 muestra el resultado de la evaluación a la funcionalidad 21, mismo que se evidencia en el anexo 37 del documento.

Tabla 49: Resultado de evaluación - Vista de artículos en bodega por cliente

Evaluación de funcionalidad del proyecto				
Funcionalidad 22:	Vista de artículos en bodega por cliente			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se presenta un listado de los artículos que se encuentran en la bodega de una agencia.	Se agrupa la información por clientes	100%	Se cumple con la expectativa	100%
	Se visualiza lo que un cliente tiene en bodega, lo que se esta esperando recibir y lo que almacena en otras bodegas.	100%	Se cumple con la expectativa	
	Se muestra la cantidad de productos que tiene un cliente y el peso total de estos.	100%	Se cumple con la expectativa	

La tabla 49 muestra el resultado de la evaluación a la funcionalidad 22, mismo que se evidencia en el anexo 38 del documento.

Tabla 50: Resultado de evaluación - Vista de envíos con sus productos

Evaluación de funcionalidad del proyecto				
Funcionalidad 23:	Vista de envíos con sus productos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
se debe listar todos los envíos realizados y organizados por fecha, con un indicador de cumplimiento y un botón para visualizar el contenido de cada envío	Filtro por agencia	100%	Se cumple con la expectativa	100%
	Indicadores de estado	100%	Se cumple con la expectativa	
	Botones de manipulación	100%	Se cumple con la expectativa	
	Información relevante	100%	Se cumple con la expectativa	

La tabla 50 muestra el resultado de la evaluación a la funcionalidad 23, mismo que se evidencia en el anexo 39 del documento.

Tabla 51: Resultado de evaluación - Creación de etiquetado de productos

Evaluación de funcionalidad del proyecto				
Funcionalidad 24:	Creación de etiquetado de productos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se crea un código por cada producto registrado, posteriormente se debe imprimir dicho código y pegarlo en el producto, esto ayuda a gestionar el estado y ubicación del producto de manera rápida y eficaz.	Se crea un código por cada producto	100%	En la base de datos existe un disparador para crear un código único por cada registro que se inserta en la tabla de productos.	100%
	Se permite la impresión de códigos con estilo código de barra y QR	100%	Se cumple con la expectativa	

La tabla 51 muestra el resultado de la evaluación a la funcionalidad 24, mismo que se evidencia en el anexo 40 y anexo 41 del documento.

Tabla 52: Resultado de evaluación - Campo para escanear códigos de productos

Evaluación de funcionalidad del proyecto				
Funcionalidad 25:	Campo para escanear códigos de productos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Componente especializado para la lectura de códigos de barra y escaneo de códigos QR	Diseño simple	100%	Se cumple con la expectativa	83%
	Diseño intuitivo	50%	Hace falta un tooltip o una descripción del campo input	
	Luego de escanear un código debe indicar toda la información relacionada al mismo.	100%	Se cumple con la expectativa	

La tabla 52 muestra el resultado de la evaluación a la funcionalidad 25, mismo que se evidencia en el anexo 42, anexo 43 y anexo 44 del documento.

Tabla 53: Resultado de evaluación - Escaneo de productos para generación de guías aéreas

Evaluación de funcionalidad del proyecto				
Funcionalidad 26:	Escaneo de productos para generación de guías aéreas			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
El componente de escaneo implementado en la interfaz para creación de guías aéreas debe buscar el código escaneado y posteriormente agregarlo a la guía.	Solicitud automática a la capa de aplicación luego de escaner un código	100%	Se cumple con la expectativa	100%
	Inserción automática en la guía luego cuando el servidor responde con el producto solicitado.	100%	Se cumple con la expectativa	

La tabla 53 muestra el resultado de la evaluación a la funcionalidad 26, mismo que se evidencia en el anexo 45 del documento.

Tabla 54: Resultado de evaluación - Escaneo de productos para generación de guías de entrega

Evaluación de funcionalidad del proyecto				
Funcionalidad 27:	Escaneo de productos para generación de guías de entrega			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
El componente de escaneo implementado en la interfaz para creación de despachos a clientes, debe buscar el código escaneado en la agencia donde se encuentra el agente y posteriormente agregarlo a la guía.	Solicitud automática a la capa de aplicación luego de escaner un código	100%	Se cumple con la expectativa	100%
	Validación de existencia de productos disponibles en bodega	100%	Se cumple con la expectativa	
	Inserción automática en la guía luego cuando el servidor responde con el producto solicitado.	100%	Se cumple con la expectativa	

La tabla 54 muestra el resultado de la evaluación a la funcionalidad 27, mismo que se evidencia en el anexo 46 del documento.

Tabla 55: Resultado de evaluación - Módulo para recibir encomiendas en el casillero postal y agregar productos

Evaluación de funcionalidad del proyecto				
Funcionalidad 28:	Modulo para recibir encomiendas en el casillero postal y agregar productos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se registra la información referente a la encomienda recibida en el casillero postal y todo su contenido es asignado al cliente propietario del mismo	Opción para cargar una imagen de evidencia	75%	Permite cargar una sola imagen, se debería poder hacer una carga masiva	92%
	Opción para seleccionar al cliente	100%	Se filtran los clientes según los caracteres que vamos ingresando en el input	
	Campo para agregar el número de seguimiento del paquete	100%	Se cumple con la expectativa	

La tabla 55 muestra el resultado de la evaluación a la funcionalidad 28, mismo que se evidencia en el anexo 47 y anexo 48 del documento.

Tabla 56: Resultado de evaluación - Módulo para registrar compras de clientes con sus productos

Evaluación de funcionalidad del proyecto				
Funcionalidad 29:	Modulo para registrar compras de clientes con sus productos			
TEST DEL REQUERIMIENTO				
Manifiesto	**Resultados esperados**	**% Cumplimiento**	**Descripción**	**Valoración**
Se registra una compra realizada por el cliente para tener en cuenta la encomienda que se espera recibir y su contenido	Se debe poder indicar el valor de la compra	100%	Cuando la compra se realiza con el dinero del negocio, se crea un debito en la cuenta del cliente.	100%
	Se debe ingresar la tienda donde se realiza el pedido	100%	Se cumple con la expectativa.	
	Se debe poder adjuntar una imagen de evidencia que corrobore que la compra se realizó	100%	Se cumple con la expectativa.	

La tabla 56 muestra el resultado de la evaluación a la funcionalidad 29, mismo que se evidencia en el anexo 49 del documento.

Se ha promediado los resultados de la evaluación para determinar el nivel de satisfacción que tiene el proyecto ante las expectativas planteadas.

Figura 63: Resultado de evaluación

La figura 63 muestra un gráfico de pastel con el promedio de la evaluación, dicho dato da por concluido el proyecto con un grado de satisfacción mayor a 90% por lo que se considera un producto altamente funcional.

4. CONCLUSIONES

- Se estableció la estructura hardware que soporta la implementación de una arquitectura de tres capas para la solución del proyecto, esto genera valor a la flexibilidad y escalabilidad del mismo.

- Se desarrolló una API REST con .Net 5, misma que es el núcleo de la capa de aplicación, una página web SPA con Angular para para la capa de presentación y una base de datos relacional con PostgreSQL para almacenar la información del negocio; con el fin de tener independencia en los componentes principales de la arquitectura.

- Se estructuró módulos para registrar usuarios, depósitos, productos, agencias, despachos, empleados, reembolsos, guías aéreas, cuentas por cobrar, cargar movimientos bancarios, seguimiento a producto, escaneo de códigos de barra y códigos QR y administración de permisos para la automatización de las tareas que la empresa maneja con el fin de mejorar la gestión de atención al cliente y logística del negocio.

- Se implementó la capa de acceso a los datos haciendo uso de los patrones scoped y transient para asegurar una ciclo de comunicación con la base de datos de manera controlada y estable.

- Se usó la metodología SCRUM para la dirección del proyecto, misma que nos permitió alcanzar los objetivos planteados con 11 sprints.

- Se configuró políticas del negocio y permisos de usuario con el fin de conceder o denegar servicios a partir del nivel de acceso a la información que tiene un usuario autenticado.

- Se agregó la funcionalidad al cliente para interceptar todas las peticiones que realiza a la API con el fin de insertar en la cabecera de la solicitud el Bearer JWT, mismo que contiene de manera cifrada las credenciales del usuario. Esto ayuda a la confidencialidad de la información compartida.

- Se realizó la evaluación del prototipo en producción para determinar el nivel de satisfacción que posee el proyecto; el resultado de la evaluación indica que las funcionalidades del sistema se satisfacen en un 91%, por lo que se puede asegurar que la gestión de procesos logísticos y atención al cliente se ha mejorado.

5. RECOMENDACIONES

- Primero establecer los requisitos del negocio y posterior a ello elegir las tecnologías de desarrollo e implementación con el fin de asegurar la compatibilidad entre todo el ecosistema tecnológico.

- Utilizar clases genéricas, contratos y patrones de diseño en la construcción del backend con el propósito de tener un desarrollo limpio, fácil de administrar, corregir e implementar cambios; esto nos ayuda a mantener un buen concepto de elasticidad y escalabilidad.

- Basar la construcción de clases en estándares de programación a fin de mantener la compatibilidad con otras tecnologías de manera sencilla.

- Crear una hoja de estilo responsive para toda la aplicación con el fin de implementar un diseño homogéneo y amigable a más de aumentar la capacidad de realizar cambios generales a todos los componentes del frontend.

- Asegurarse que todo el equipo scrum entienda las necesidades del negocio a la perfección, de esta manera se evitará retrasos en el desarrollo de la solución y será más fácil identificar fallas u opciones de optimización para todos los miembros del equipo.

6. BIBLIOGRAFÍA

[1] «Envíos Courier o Postal – Servicio Nacional de Aduana del Ecuador», *Servicio Nacional de Aduana del Ecuador*. https://www.aduana.gob.ec/envios-courier-postal/ (accedido 2022).

[2] L. Wang, Y. Chai, y Y. Liu, «Analysis of e-commerce transaction system's division of labor based on essential services quantity», *Int. J. Crowd Sci.*, vol. 1, n.º 3, pp. 197-209, sep. 2017, doi: 10.1108/IJCS-08-2017-0015.

[3] W. Yu, C. Yan, Z. Ding, C. Jiang, y M. Zhou, «Analyzing E-Commerce Business Process Nets via Incidence Matrix and Reduction», *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 48, n.º 1, pp. 130-141, ene. 2018, doi: 10.1109/TSMC.2016.2598287.

[4] M. Wang, Z. Ding, P. Zhao, W. Yu, y C. Jiang, «A Dynamic Data Slice Approach to the Vulnerability Analysis of E-Commerce Systems», *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 50, n.º 10, pp. 3598-3612, oct. 2020, doi: 10.1109/TSMC.2018.2862387.

[5] «VPS Hosting - VPS with SDD Storage | HostGator». https://www.hostgator.com/vps-hosting/?utm_source=google&utm_medium=brandsearch&gclid=CjwKCAjwpKyYBhB7EiwAU2Hn2Xld9yAbBNUP2G-XT6xhlSK0VG5NFcDzN1J1njMVDHqtPbuVrPmfehoCjFYQAvD_BwE&gclsrc=aw.ds#compare-plans (accedido 2022).

[6] E. Maya y D. López, «Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web», presentado en Gestión delas TICs para la Investigación y la Colaboración, San José, jul. 2018. [En línea]. Disponible en: https://www.researchgate.net/publication/328887426_Arquitectura_de_Software_basada_en_Microservicios_para_Desarrollo_de_Aplicaciones_Web

[7] J. Hernández, P. Velasco-Elizondo, G. Sobrevilla, y S. Soriano, «Aplicando Scrum y Prácticas de Ingeniería de Software para la Mejora Continua del Desarrollo de un Sistema Ciber-Físico», *ReCIBE Rev. Electrónica Comput. Informática Bioméd. Electrónica*, vol. 6, n.º 1, pp. 1-15, 2017.

[8] Y. T. Casañola, Y. B. Benítez, y Y. M. Lombida, «Buenas prácticas de experiencia de usuario ante los factores críticos: tiempo, experiencia y presupuesto», *Rev. Cuba. Cienc. Informáticas*, vol. 15, pp. 297-313, 2021.

[9] Y. Viveros Martínez, E. López Domínguez, Y. Hernández Velázquez, S. Domínguez Isidro, M. A. Medina Nieto, y J. De La Calleja, «Layered Software Architecture for the Development of Third-Generation Video Surveillance Systems», *IEEE Access*, vol. 7, pp. 98507-98521, 2019, doi: 10.1109/ACCESS.2019.2930401.

[10] C. Li, J. Ge, Z. Li, L. Huang, H. Yang, y B. Luo, «Monitoring Interactions Across Multi Business Processes with Token Carried Data», *IEEE Trans. Serv. Comput.*, vol. 12, n.º 6, pp. 941-954, nov. 2019, doi: 10.1109/TSC.2016.2645690.

[11] S. E. Cebeci, K. Nari, y E. Ozdemir, «Secure E-Commerce Scheme», *IEEE Access*, vol. 10, pp. 10359-10370, 2022, doi: 10.1109/ACCESS.2022.3145030.

[12] X. Ma *et al.*, «One Host with So Many IPs! On The Security Implications of Dynamic Virtual Private Servers», *IEEE Commun. Mag.*, vol. 59, n.º 2, pp. 64-69, feb. 2021, doi: 10.1109/MCOM.001.2000602.

[13] E. Pateromichelakis *et al.*, «End-to-End Data Analytics Framework for 5G Architecture», *IEEE Access*, vol. 7, pp. 40295-40312, 2019, doi: 10.1109/ACCESS.2019.2902984.

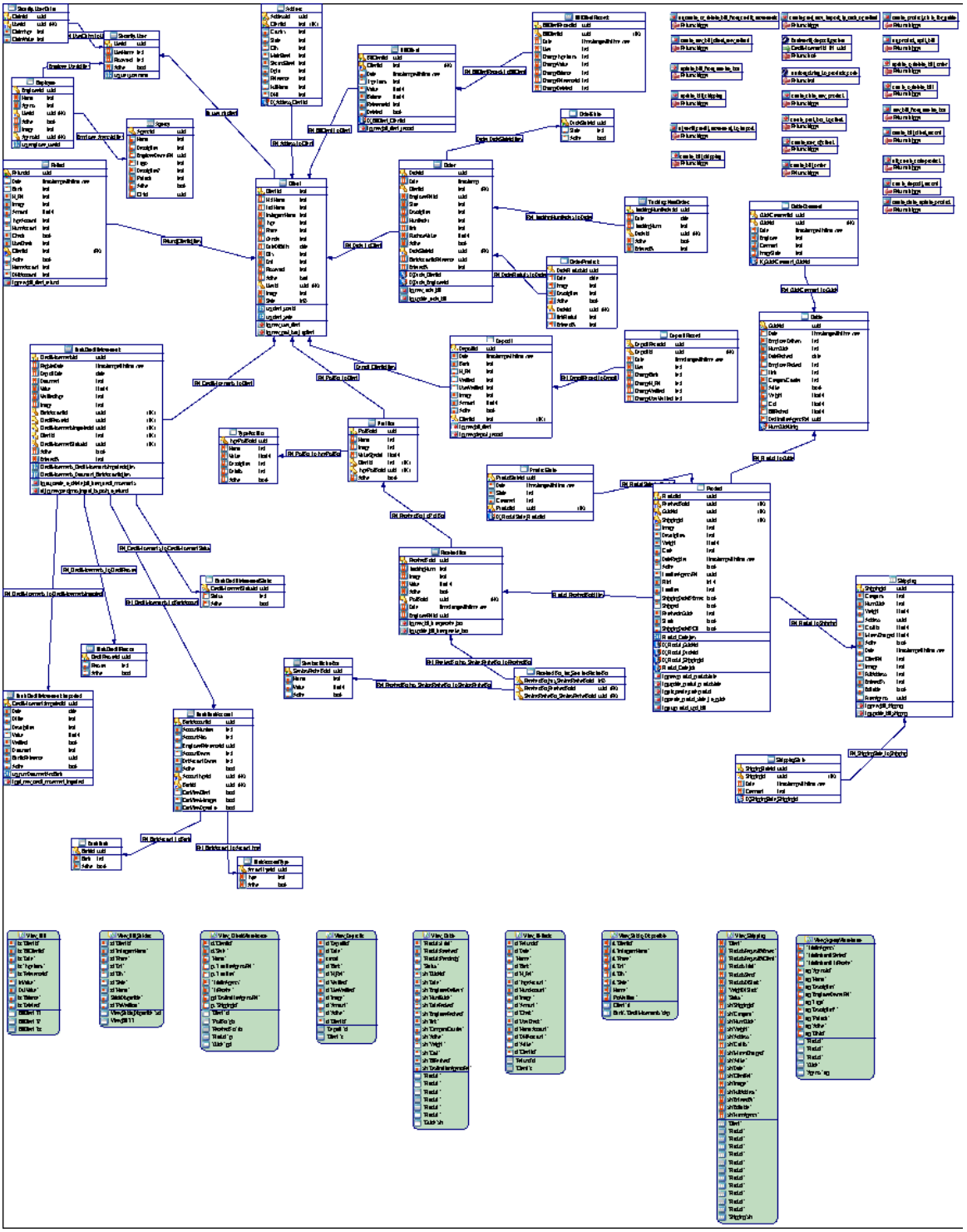
[14] M. Lim, «C2CFTP: Direct and Indirect File Transfer Protocols Between Clients in Client-Server Architecture», *IEEE Access*, vol. 8, pp. 102833-102845, 2020, doi: 10.1109/ACCESS.2020.2998725.

[15] L. Masing, F. Lesniak, y J. Becker, «A Hybrid Prototyping Framework in a Virtual Platform Centered Design and Verification Flow», *IEEE Embed. Syst. Lett.*, vol. 13, n.º 1, pp. 1-4, mar. 2021, doi: 10.1109/LES.2020.2995084.
[16] gewarren, «Introducción a .NET Framework - .NET Framework». https://docs.microsoft.com/es-es/dotnet/framework/get-started/ (accedido 2022).
[17] mjrousos, «Overview of ASP.NET Core Authentication». https://docs.microsoft.com/en-us/aspnet/core/security/authentication/ (accedido 2022).
[18] J. K. Musyimi, *A SAML 2.0 Authentication Middleware for ASP.NET Core*. 2018. [En línea]. Disponible en: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2502567/20071_FULLTEXT.pdf?sequence=1
[19] BillWagner, «Language-Integrated Query (LINQ) (C#)». https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (accedido 2022).
[20] H. Wang y F. Fang, «Research on E-Commerce Supply Chain Design Based on MVC Model and Virtual Image Technology», *IEEE Access*, vol. 8, pp. 98295-98304, 2020, doi: 10.1109/ACCESS.2020.2996675.
[21] Rick-Anderson, «Introduction to authorization in ASP.NET Core». https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction (accedido 2022).
[22] D. Chandra Jadala, «Authentication and Authorization Mechanism for Cloud Security», vol. 8, n.º 6, sep. 2019, doi: 10.35940/ijeat.F8473.088619.
[23] davidmu1, «Tokens de acceso de la Plataforma de identidad de Microsoft - Microsoft Entra». https://docs.microsoft.com/es-es/azure/active-directory/develop/access-tokens (accedido 2022).
[24] dotnet-bot, «Microsoft AspNetCore Authentication». https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication (accedido 2022).
[25] R. F. Olanrewaju, B. U. I. Khan, M. A. Morshidi, F. Anwar, y M. L. B. M. Kiah, «A Frictionless and Secure User Authentication in Web-Based Premium Applications», *IEEE Access*, vol. 9, pp. 129240-129255, 2021, doi: 10.1109/ACCESS.2021.3110310.
[26] L. T. Espinosa y N. S. Martínez, «Aplicación de un proceso para la gestión de la mantenibilidad en el desarrollo de software», *Rev. Cuba. Cienc. Informáticas*, vol. 15, pp. 350-365, 2021.
[27] D. Nesteruk, «Singleton», en *Design Patterns in Modern C++20*, 2022, pp. 101-120. doi: 10.1007/978-1-4842-7295-4_5.
[28] G. Vega-Gorgojo, «CRAFTS: Configurable REST APIs for Triple Stores», *IEEE Access*, vol. 10, pp. 32426-32441, 2022, doi: 10.1109/ACCESS.2022.3160610.
[29] H.-W. Deng, M. Rahman, M. Chowdhury, M. S. Salek, y M. Shue, «Commercial Cloud Computing for Connected Vehicle Applications in Transportation Cyberphysical Systems: A Case Study», *IEEE Intell. Transp. Syst. Mag.*, vol. 13, n.º 1, pp. 6-19, 2021, doi: 10.1109/MITS.2020.3037314.
[30] Y. Muñoz, M. A. Alonso-Lavernia, I. Castillo-Pérez, V. Martínez-Lazcano, y F. Gálvez-González, «Desarrollo de un Sitio Web con Metodologías de Diseño Hipermedial y de Diseño Dinámico», *Ingenio Concienc. Bol. Científico Esc. Super. Ciudad Sahagún*, vol. 7, n.º 13, Art. n.º 13, ene. 2020, doi: 10.29057/escs.v7i13.5263.
[31] S. Uzayr, «TypeScript Architecture», en *TypeScript for Beginners*, 2022, pp. 151-174. doi: 10.1201/9781003203728-5.

[32] M. Segovia y S. Roche, «Evaluación de la facilidad de aprendizaje de frameworks JavaScript: Backbone, Angular y Ember», *Rev. Tecnol. Cienc. Educ.*, pp. 67-83, ene. 2019, doi: 10.51302/tce.2019.242.
[33] A. Arraiz, «Framework de desarrollo de Proyectos Sociotecnológicos basado en la notación de Metamodelos de procesos de Ingeniería de Software (spem 2.0)», *Rev. Arbitr. Interdiscip. Koinonía*, vol. 3, n.º 6, pp. 79-95, 2018.
[34] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, y P.-E. Gaillardon, «OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs», *IEEE Micro*, vol. 40, n.º 4, pp. 41-48, jul. 2020, doi: 10.1109/MM.2020.2995854.
[35] S. L. Torres, «Componente de revisión de estándar de arquitectura de datos para el gestor de bases de datos SQLite», *Innov. Softw.*, vol. 2, n.º 1, p. 14, 2021.
[36] W. de O. Silva y A. F. T. Carneiro, «SUBSURFACE UTILITY NETWORK CADASTRE PROPOSAL, BASED ON LADM (ISO / FDIS 19152)», *Bol. Ciênc. Geodésicas*, vol. 26, n.º 2., p. e2020006, 2020, doi: 10.1590/s1982-21702020000200006.
[37] Y.-F. Kung, S.-W. Liou, G.-Z. Qiu, B.-C. Zu, Z.-H. Wang, y G.-J. Jong, «Home monitoring system based internet of things», en *2018 IEEE International Conference on Applied System Invention (ICASI)*, abr. 2018, pp. 325-327. doi: 10.1109/ICASI.2018.8394599.
[38] «Postman API Platform | Sign Up for Free», *Postman*. https://www.postman.com/ (accedido 2022).
[39] C. Alarcón y S. Rojas, «Comparacion Prince2 y OpenUp para desarrollo de software | Tecnología Investigación y Academia», *Tecnol. Investig. Acad.*, vol. 6, n.º 1, pp. 77-83, 2018.
[40] H. Vite Cevallos, K. Montero, y J. Cuesta, «Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software», *Espirales Rev. Multidiscip. Investig.*, vol. 2, jun. 2018, doi: 10.31876/re.v2i17.269.
[41] J. F. Herrera-Cubides, P. A. Gaona-García, y K. Gordillo-Orjuela, «Una Mirada a la Web de los Datos. Caso de Estudio: Consumo de Servicios CKAN», *Ingeniería*, vol. 22, n.º 1, p. 111, ene. 2017, doi: 10.14483/udistrital.jour.reving.2017.1.a07.
[42] M. Sihuay, A. Dávila, y M. Pessoa, «Factors Models of Scrum Adoption in the Software Development Process: A Systematic Literature Review», *ReCIBE Rev. Electrónica Comput. Informática Bioméd. Electrónica*, vol. 7, n.º 1, pp. 23-44, 2018.

ANEXOS

Anexo 1: Diagrama de Modelo Relacional



Anexo 2: Código SQL para la generación de la base de datos

```
CREATE SCHEMA "Bank" AUTHORIZATION "enviosecuador";
CREATE SCHEMA "Security" AUTHORIZATION "postgres";

CREATE TABLE "Address" (
    "AddressId" uuid NOT NULL,
    "ClientId" text NOT NULL,
    "Country" text NOT NULL,
```

```
    "State" text,
    "City" text NOT NULL,
    "MainStreet" text NOT NULL,
    "SecondStreet" text,
    "Dpto" text,
    "Reference" text,
    "FullName" text NOT NULL,
    "DNI" text,
    CONSTRAINT "Address_pkey" PRIMARY KEY("AddressId","ClientId")
);

CREATE INDEX "IX_Address_ClientId" ON "Address" USING BTREE (
    "ClientId"
);


ALTER TABLE "Address" OWNER TO "postgres";

CREATE TABLE "Agency" (
    "AgencyId" uuid NOT NULL,
    "Name" text NOT NULL,
    "Description" text,
    "EmployeeOwnerRef" uuid,
    "Logo" text,
    "Description2" text,
    "Portada" text,
    "Active" bool NOT NULL DEFAULT true,
    "CityId" uuid,
    CONSTRAINT "Agency_pkey" PRIMARY KEY("AgencyId")
);

ALTER TABLE "Agency" OWNER TO "enviosecuador";

CREATE TABLE "BillClient" (
    "BillClientId" uuid NOT NULL,
    "ClientId" text NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "TypeTrans" text NOT NULL,
    "Value" float4 NOT NULL,
    "Balance" float4 NOT NULL,
    "ReferenceId" text NOT NULL,
    "Deleted" bool NOT NULL DEFAULT false,
    CONSTRAINT "PK_BillClient" PRIMARY KEY("BillClientId")
);

CREATE INDEX "IX_BillClient_ClientId" ON "BillClient" USING BTREE (
    "ClientId"
);


CREATE TRIGGER "tg_new_bill_client_record" AFTER UPDATE
    ON "BillClient" FOR EACH ROW
    EXECUTE PROCEDURE "create_bill_client_record"();


ALTER TABLE "BillClient" OWNER TO "postgres";

CREATE TABLE "BillClientRecords" (
    "BillClientRecordsId" uuid NOT NULL,
    "BillClientId" uuid,
    "Date" timestamp with time zone DEFAULT now(),
    "User" text,
    "ChangeTypeTrans" text,
    "ChangeValue" text,
```

```
    "ChangeBalance" text,
    "ChangeReferenceId" text,
    "ChangeDeleted" text,
    CONSTRAINT "BillClientRecords_pkey" PRIMARY
KEY("BillClientRecordsId")
);

ALTER TABLE "BillClientRecords" OWNER TO "postgres";

CREATE TABLE "Client" (
    "ClientId" text NOT NULL,
    "FirstName" text NOT NULL,
    "LastName" text NOT NULL,
    "InstagramName" text,
    "Type" text,
    "Phone" text,
    "Gender" text,
    "DateOfBirth" date,
    "City" text NOT NULL,
    "Dni" text,
    "Password" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "UserId" uuid,
    "Image" text,
    "Serie" BIGSERIAL NOT NULL,
    CONSTRAINT "PK_Client" PRIMARY KEY("ClientId"),
    CONSTRAINT "uq_client_userid" UNIQUE("UserId"),
    CONSTRAINT "uq_client_serie" UNIQUE("Serie")
);

CREATE TRIGGER "tg_new_user_client" AFTER INSERT
    ON "Client" FOR EACH ROW
    EXECUTE PROCEDURE "create_user_of_client"();


CREATE TRIGGER "tg_new_post_box_to_client" AFTER INSERT
    ON "Client" FOR EACH ROW
    EXECUTE PROCEDURE "create_post_box_to_client"();


ALTER TABLE "Client" OWNER TO "postgres";

CREATE TABLE "Deposit" (
    "DepositId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "Bank" text NOT NULL,
    "N_Ref" text,
    "Verified" text DEFAULT 'FALSE'::text,
    "UserVerified" text DEFAULT 'SYSTEM'::text,
    "Image" text,
    "Amount" float4 NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "ClientId" text NOT NULL,
    CONSTRAINT "Deposit_pkey" PRIMARY KEY("DepositId")
);

CREATE TRIGGER "tg_new_bill_client" AFTER UPDATE
    ON "Deposit" FOR EACH ROW
    EXECUTE PROCEDURE "create_o_delete_bill"();


CREATE TRIGGER "tg_new_deposit_record" AFTER UPDATE
    ON "Deposit" FOR EACH ROW
    EXECUTE PROCEDURE "create_deposit_record"();
```

```
ALTER TABLE "Deposit" OWNER TO "postgres";

CREATE TABLE "DepositRecord" (
    "DepositRecordId" uuid NOT NULL,
    "DepositId" uuid,
    "Date" timestamp with time zone,
    "User" text,
    "ChangeBank" text,
    "ChangeN_Ref" text,
    "ChangeVerified" text,
    "ChangeUserVerified" text,
    CONSTRAINT "DepositRecord_pkey" PRIMARY KEY("DepositRecordId")
);

ALTER TABLE "DepositRecord" OWNER TO "postgres";

CREATE TABLE "Employee" (
    "EmployeeId" uuid NOT NULL,
    "Name" text NOT NULL,
    "Agency" text NOT NULL,
    "UserId" uuid,
    "Active" bool NOT NULL DEFAULT true,
    "Image" text,
    "AgencyId" uuid NOT NULL,
    CONSTRAINT "PK_Employee" PRIMARY KEY("EmployeeId"),
    CONSTRAINT "uq_employee_userid" UNIQUE("UserId")
);

ALTER TABLE "Employee" OWNER TO "postgres";

CREATE TABLE "Guide" (
    "GuideId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "EmployeeDelivery" text NOT NULL,
    "NumGuide" text,
    "DateRecived" date,
    "EmployeeRecived" text,
    "Link" text,
    "CompanyCourrier" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "Weight" float4 NOT NULL,
    "Cost" float4 NOT NULL,
    "BillRecived" float4 NOT NULL,
    "DestinationAgencyRef" uuid NOT NULL,
    CONSTRAINT "PK_Guide" PRIMARY KEY("GuideId")
);

CREATE UNIQUE INDEX "NumGuideUniq" ON "Guide" USING BTREE (
    "NumGuide"
);


ALTER TABLE "Guide" OWNER TO "postgres";

CREATE TABLE "GuideComment" (
    "GuideCommentId" uuid NOT NULL,
    "GuideId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "Employee" text NOT NULL,
    "Comment" text,
    "ImageState" text,
    CONSTRAINT "PK_GuideComment" PRIMARY KEY("GuideCommentId")
```

```
);

CREATE INDEX "IX_GuideComment_GuideId" ON "GuideComment" USING BTREE (
    "GuideId"
);


ALTER TABLE "GuideComment" OWNER TO "postgres";

CREATE TABLE "Order" (
    "OrderId" uuid NOT NULL,
    "Date" timestamp NOT NULL,
    "ClientId" text NOT NULL,
    "EmployeeRefId" uuid,
    "Store" text,
    "Description" text,
    "NumOrder" text,
    "Link" text,
    "PurchaseValue" float4 NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "OrderStateId" uuid,
    "BankAccountIdReference" uuid,
    "EnteredBy" text NOT NULL DEFAULT 'ENVIOS ECUADOR'::text,
    CONSTRAINT "PK_Order" PRIMARY KEY("OrderId")
);

CREATE INDEX "IX_Order_ClientId" ON "Order" USING BTREE (
    "ClientId"
);


CREATE INDEX "IX_Order_EmployeeId" ON "Order" USING BTREE (
    "EmployeeRefId"
);


CREATE TRIGGER "tg_new_order_bill" AFTER INSERT
    ON "Order" FOR EACH ROW
    EXECUTE PROCEDURE "create_bill_order"();


CREATE TRIGGER "tg_update_order_bill" AFTER UPDATE
    ON "Order" FOR EACH ROW
    EXECUTE PROCEDURE "update_o_delete_bill_order"();


ALTER TABLE "Order" OWNER TO "postgres";

CREATE TABLE "OrderProducts" (
    "OrderProductsId" uuid NOT NULL,
    "Date" date NOT NULL,
    "Image" text,
    "Description" text,
    "Active" bool NOT NULL DEFAULT true,
    "OrderId" uuid NOT NULL,
    "LinkProduct" text,
    "EnteredBy" text NOT NULL DEFAULT 'ENVIOS ECUADOR'::text,
    CONSTRAINT "OrderProducts_pkey" PRIMARY KEY("OrderProductsId")
);

ALTER TABLE "OrderProducts" OWNER TO "enviosecuador";

CREATE TABLE "OrderState" (
    "OrderStateId" uuid NOT NULL,
```

```
    "State" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "PK_OrderState" PRIMARY KEY("OrderStateId")
);

ALTER TABLE "OrderState" OWNER TO "postgres";

COMMENT ON COLUMN "OrderState"."State" IS 'Todos los productos llegaron
Todos los productos enviados
Faltan productos por llegar
Faltan productos por enviar
Valores Pendientes de pagar';

CREATE TABLE "PostBox" (
    "PostBoxId" uuid NOT NULL,
    "Name" text,
    "Image" text,
    "ValueSpecial" float4,
    "ClientId" text NOT NULL,
    "TypePostBoxId" uuid,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "PostBox_pkey" PRIMARY KEY("PostBoxId")
);

ALTER TABLE "PostBox" OWNER TO "enviosecuador";

COMMENT ON TABLE "PostBox" IS 'Pueden Haber diferentes tipos de
casilleros:
1. VIP
2. DIRECTOS - SIN REVISION (lb $ )
3. NORMAL';

CREATE TABLE "Product" (
    "ProductId" uuid NOT NULL,
    "ReceivedBoxId" uuid NOT NULL,
    "GuideId" uuid NOT NULL,
    "ShippingId" uuid NOT NULL,
    "Image" text,
    "Descriptions" text NOT NULL,
    "Weight" float4 NOT NULL,
    "Code" text,
    "DateRegister" timestamp with time zone NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "LocationAgencyRef" uuid,
    "Print" int4 NOT NULL DEFAULT 0,
    "Location" text,
    "ShippingOrderByEnvec" bool NOT NULL DEFAULT false,
    "Shipped" bool NOT NULL DEFAULT false,
    "ReceivedInGuide" text NOT NULL DEFAULT 'NONE'::text,
    "Stock" bool NOT NULL DEFAULT false,
    "ShippingOrderByCli" bool NOT NULL DEFAULT false,
    CONSTRAINT "PK_Product" PRIMARY KEY("ProductId"),
    CONSTRAINT "Product_Code_key" UNIQUE("Code")
);

CREATE INDEX "IX_Product_GuideId" ON "Product" USING BTREE (
    "GuideId"
);


CREATE INDEX "IX_Product_OrderId" ON "Product" USING BTREE (
    "ReceivedBoxId"
);
```

```
CREATE INDEX "IX_Product_ShippingId" ON "Product" USING BTREE (
    "ShippingId"
);


CREATE UNIQUE INDEX "Product_Code_idx" ON "Product" USING BTREE (
    "Code"
);


CREATE TRIGGER "tg_new_product_productstate" AFTER INSERT
    ON "Product" FOR EACH ROW
    EXECUTE PROCEDURE "create_state_new_product"();


CREATE TRIGGER "tg_update_product_productstate" AFTER UPDATE
    ON "Product" FOR EACH ROW
    EXECUTE PROCEDURE "create_state_update_product"();


CREATE TRIGGER "tg_afc_create_codeproduct" AFTER INSERT
    ON "Product" FOR EACH ROW
    EXECUTE PROCEDURE "afc_create_codeproduct"();


CREATE TRIGGER "tg_create_product_state_for_guide" AFTER UPDATE
    ON "Product" FOR EACH ROW
    EXECUTE PROCEDURE "create_product_state_for_guide"();


CREATE TRIGGER "tg_au_product_updt_bill" AFTER UPDATE
    ON "Product" FOR EACH ROW
    EXECUTE PROCEDURE "au_product_updt_bill"();


ALTER TABLE "Product" OWNER TO "postgres";

CREATE TABLE "ProductState" (
    "ProductStateId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "State" text NOT NULL,
    "Comment" text,
    "ProductId" uuid NOT NULL,
    CONSTRAINT "PK_ProductState" PRIMARY KEY("ProductStateId")
);

CREATE INDEX "IX_ProductState_ProductId" ON "ProductState" USING BTREE
(
    "ProductId"
);


ALTER TABLE "ProductState" OWNER TO "postgres";

COMMENT ON COLUMN "ProductState"."State" IS 'Todos los productos
llegaron
Todos los productos enviados
Faltan productos por llegar
Faltan productos por enviar
Valores Pendientes de pagar';

CREATE TABLE "ReceivedBox" (
    "ReceivedBoxId" uuid NOT NULL,
```

```
    "TrackingNum" text,
    "Image" text,
    "Value" float4 NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "PostBoxId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "EmployeeRefId" uuid NOT NULL,
    CONSTRAINT "ReceivedBox_pkey" PRIMARY KEY("ReceivedBoxId")
);

CREATE TRIGGER "tg_new_bill_from_receive_box" AFTER INSERT
    ON "ReceivedBox" FOR EACH ROW
    EXECUTE PROCEDURE "new_bill_from_receive_box"();


CREATE TRIGGER "tg_update_bill_from_receive_box" AFTER UPDATE
    ON "ReceivedBox" FOR EACH ROW
    EXECUTE PROCEDURE "update_bill_from_receive_box"();


ALTER TABLE "ReceivedBox" OWNER TO "enviosecuador";

CREATE TABLE "ReceivedBox_has_ServicesReciveBox" (
    "ReceivedBox_has_ServicesReciveBoxId" BIGSERIAL NOT NULL,
    "ReceivedBox_ReceivedBoxId" uuid NOT NULL,
    "ServicesReciveBox_ServicesReciveBoxId" uuid NOT NULL,
    CONSTRAINT "ReceivedBox_has_ServicesReciveBox_pkey" PRIMARY
KEY("ReceivedBox_has_ServicesReciveBoxId")
);

ALTER TABLE "ReceivedBox_has_ServicesReciveBox" OWNER TO
"enviosecuador";

CREATE TABLE "Refund" (
    "RefundId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "Bank" text NOT NULL,
    "N_Ref" text NOT NULL,
    "Image" text,
    "Amount" float4 NOT NULL,
    "TypeAccount" text,
    "NumAccount" text,
    "Check" bool DEFAULT false,
    "UserCheck" text,
    "ClientId" text,
    "Active" bool NOT NULL DEFAULT true,
    "NameAccount" text,
    "DNIAcccount" text,
    CONSTRAINT "Refund_pkey" PRIMARY KEY("RefundId")
);

CREATE TRIGGER "tg_new_bill_client_refund" AFTER UPDATE
    ON "Refund" FOR EACH ROW
    EXECUTE PROCEDURE "create_new_bill_client_new_refund"();


ALTER TABLE "Refund" OWNER TO "postgres";

CREATE TABLE "ServicesReciveBox" (
    "ServicesReciveBoxId" uuid NOT NULL,
    "Name" text NOT NULL,
    "Value" float4 NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
```

```
    CONSTRAINT "ServicesReciveBox_pkey" PRIMARY
KEY("ServicesReciveBoxId")
);

ALTER TABLE "ServicesReciveBox" OWNER TO "enviosecuador";

COMMENT ON TABLE "ServicesReciveBox" IS 'Los servicios son:
Abrir el paquete
Declaracion Aduanera
Declaracion de impuestos

El momento en q recibe la guia con este producto es cuando se agrega
esta cuenta al cliente.';

CREATE TABLE "Shipping" (
    "ShippingId" uuid NOT NULL,
    "Company" text NOT NULL,
    "NumGuide" text,
    "Weight" float4 NOT NULL,
    "Address" uuid,
    "CostLb" float4 NOT NULL,
    "MoneyCharged" float4 NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "Date" timestamp with time zone,
    "ClientRef" text NOT NULL,
    "Image" text,
    "FullAddress" text,
    "EnteredBy" text NOT NULL DEFAULT 'ENVIOS ECUADOR'::text,
    "Editable" bool NOT NULL DEFAULT true,
    "FromAgency" uuid NOT NULL,
    CONSTRAINT "PK_Shipping" PRIMARY KEY("ShippingId")
);

CREATE TRIGGER "tg_new_bill_shippng" AFTER INSERT
    ON "Shipping" FOR EACH ROW
    EXECUTE PROCEDURE "create_bill_shipping"();


CREATE TRIGGER "tg_update_bill_shippng" AFTER UPDATE
    ON "Shipping" FOR EACH ROW
    EXECUTE PROCEDURE "update_bill_shipping"();


ALTER TABLE "Shipping" OWNER TO "postgres";

CREATE TABLE "ShippingState" (
    "ShippingStateId" uuid NOT NULL,
    "ShippingId" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    "Comment" text NOT NULL,
    CONSTRAINT "PK_ShippingState" PRIMARY KEY("ShippingStateId")
);

CREATE INDEX "IX_ShippingState_ShippingId" ON "ShippingState" USING
BTREE (
    "ShippingId"
);


ALTER TABLE "ShippingState" OWNER TO "postgres";

CREATE TABLE "TrackingsNumOrders" (
    "TrackingsNumOrdersId" uuid NOT NULL,
    "Date" date NOT NULL,
```

```
    "TrackingNum" text NOT NULL,
    "OrderId" uuid NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "EnteredBy" text NOT NULL DEFAULT 'ENVIOS ECUADOR'::text,
    CONSTRAINT "TrackingsNumOrders_pkey" PRIMARY
KEY("TrackingsNumOrdersId")
);

ALTER TABLE "TrackingsNumOrders" OWNER TO "enviosecuador";

CREATE TABLE "TypePostBox" (
    "TypePostBoxId" uuid NOT NULL,
    "Name" text NOT NULL,
    "Value" float4 NOT NULL,
    "Description" text NOT NULL,
    "Details" text,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "TypePostBox_pkey" PRIMARY KEY("TypePostBoxId")
);

ALTER TABLE "TypePostBox" OWNER TO "enviosecuador";

COMMENT ON COLUMN "TypePostBox"."Name" IS '4x4
Comercial
Categoria G';

CREATE TABLE "Bank"."AccountType" (
    "AccountTypeId" uuid NOT NULL,
    "Type" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "AccountType_pkey" PRIMARY KEY("AccountTypeId")
);

ALTER TABLE "Bank"."AccountType" OWNER TO "enviosecuador";

CREATE TABLE "Bank"."Bank" (
    "BankId" uuid NOT NULL,
    "Bank" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "Bank_pkey" PRIMARY KEY("BankId")
);

ALTER TABLE "Bank"."Bank" OWNER TO "enviosecuador";

CREATE TABLE "Bank"."BankAccount" (
    "BankAccountId" uuid NOT NULL,
    "AccountNumber" text,
    "AccountAlias" text NOT NULL,
    "EmployeeReferenceId" uuid NOT NULL,
    "AccountOwner" text,
    "DniAccountOwner" text,
    "Active" bool NOT NULL DEFAULT true,
    "AccountTypeId" uuid NOT NULL,
    "BankId" uuid NOT NULL,
    "CanViewClient" bool NOT NULL DEFAULT false,
    "CanViewManager" bool NOT NULL DEFAULT false,
    "CanViewOperator" bool NOT NULL DEFAULT false,
    CONSTRAINT "BankAccount_pkey" PRIMARY KEY("BankAccountId")
);

ALTER TABLE "Bank"."BankAccount" OWNER TO "enviosecuador";
```

```
COMMENT ON COLUMN "Bank"."BankAccount"."AccountAlias" IS 'Aqui
ingresamos un alias (identificativo en lenguaje natural) para la
cuenta';

CREATE TABLE "Bank"."CreditMovementStatus" (
    "CreditMovementStatusId" uuid NOT NULL,
    "Status" text NOT NULL,
    "Active" bool NOT NULL,
    CONSTRAINT "CreditMovementStatus_pkey" PRIMARY
KEY("CreditMovementStatusId")
);

ALTER TABLE "Bank"."CreditMovementStatus" OWNER TO "enviosecuador";

COMMENT ON COLUMN "Bank"."CreditMovementStatus"."Status" IS 'EL estado
puede ser:

1. No Encontrado
2. Verificado
3. Pendiente
';

CREATE TABLE "Bank"."CreditMovements" (
    "CreditMovementsId" uuid NOT NULL,
    "RegisterDate" timestamp with time zone,
    "DepositDate" date,
    "Document" text,
    "Value" float4 NOT NULL,
    "VerifiedType" text,
    "Image" text,
    "BankAccountId" uuid NOT NULL,
    "CreditReasonId" uuid,
    "CreditMovementsImportedId" uuid,
    "ClientId" text NOT NULL,
    "CreditMovementStatusId" uuid NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    "EnteredBy" text NOT NULL DEFAULT 'ENVIOS ECUADOR'::text,
    CONSTRAINT "CreditMovements_pkey" PRIMARY KEY("CreditMovementsId"),
    CONSTRAINT "CreditMovements_CreditMovementsImportedId_key"
UNIQUE("CreditMovementsImportedId"),
    CONSTRAINT "CreditMovements_Document_BankAccountId_key"
UNIQUE("Document")
);

CREATE TRIGGER "tg_au_create_or_delete_bill_from_credit_movements"
AFTER UPDATE
    ON "Bank"."CreditMovements" FOR EACH ROW
    EXECUTE PROCEDURE
"au_create_or_delete_bill_from_credit_movements"();


CREATE TRIGGER "ai_tg_new_cred_mov_import_by_cash_or_refund" AFTER
INSERT
    ON "Bank"."CreditMovements" FOR EACH ROW
    EXECUTE PROCEDURE "create_cred_mov_import_by_cash_or_refund"();


ALTER TABLE "Bank"."CreditMovements" OWNER TO "enviosecuador";

COMMENT ON COLUMN "Bank"."CreditMovements"."RegisterDate" IS 'Este
campo de llena automaticamente e indica la fecha que se ingreso la fila
de registro';
```

```
COMMENT ON COLUMN "Bank"."CreditMovements"."DepositDate" IS 'Aqui se
indica la fecha del depósito segun el documento de depósito.';

COMMENT ON COLUMN "Bank"."CreditMovements"."Document" IS 'Este campo
indica el número de documento con que se realizo la transaccion';

COMMENT ON COLUMN "Bank"."CreditMovements"."Value" IS 'Este campo
indica la cantidad exacta de acreditacion segun el documento de
depósito';

COMMENT ON COLUMN "Bank"."CreditMovements"."VerifiedType" IS 'El tipo
de verificacion puede ser por:
1. El sistema (el algoritmo lo asocia)
2. Manual (El usuario lo asocia)';

CREATE TABLE "Bank"."CreditMovementsImported" (
    "CreditMovementsImportedId" uuid NOT NULL,
    "Date" date NOT NULL,
    "Office" text NOT NULL,
    "Description" text NOT NULL,
    "Value" float4 NOT NULL,
    "Verified" bool NOT NULL DEFAULT false,
    "Document" text NOT NULL,
    "BankIdReference" uuid NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "CreditMovementsImported_pkey" PRIMARY
KEY("CreditMovementsImportedId"),
    CONSTRAINT "uq_numDocumentAndBank"
UNIQUE("Document","BankIdReference")
);

CREATE TRIGGER "tg_ai_new_credit_movement_imported" AFTER INSERT
    ON "Bank"."CreditMovementsImported" FOR EACH ROW
    EXECUTE PROCEDURE "ai_verifi_credit_movement_to_import"();


ALTER TABLE "Bank"."CreditMovementsImported" OWNER TO "enviosecuador";

COMMENT ON TABLE "Bank"."CreditMovementsImported" IS 'Toda esta
informacion se importa automaticamente ';

CREATE TABLE "Bank"."CreditReason" (
    "CreditReasonId" uuid NOT NULL,
    "Reason" text NOT NULL,
    "Active" bool NOT NULL DEFAULT true,
    CONSTRAINT "CreditReason_pkey" PRIMARY KEY("CreditReasonId")
);

ALTER TABLE "Bank"."CreditReason" OWNER TO "enviosecuador";

CREATE TABLE "Security"."User" (
    "UserId" uuid NOT NULL,
    "UserName" text NOT NULL,
    "Password" text NOT NULL,
    "Active" bool NOT NULL,
    CONSTRAINT "User_pkey" PRIMARY KEY("UserId"),
    CONSTRAINT "uq_user_username" UNIQUE("UserName")
);

ALTER TABLE "Security"."User" OWNER TO "postgres";

CREATE TABLE "Security"."UserClaim" (
    "ClaimId" uuid NOT NULL,
    "UserId" uuid NOT NULL,
```

```
    "ClaimType" text NOT NULL,
    "ClaimValue" text NOT NULL,
    CONSTRAINT "UserClaim_pkey" PRIMARY KEY("ClaimId")
);

ALTER TABLE "Security"."UserClaim" OWNER TO "postgres";


ALTER TABLE "Address" ADD CONSTRAINT "Ref_Address_to_Client" FOREIGN
KEY ("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "BillClient" ADD CONSTRAINT "Ref_BillClient_to_Client"
FOREIGN KEY ("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "BillClientRecords" ADD CONSTRAINT
"Ref_BillClientRecords_to_BillClient" FOREIGN KEY ("BillClientId")
    REFERENCES "BillClient"("BillClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Client" ADD CONSTRAINT "fk_user_pk_client" FOREIGN KEY
("UserId")
    REFERENCES "Security"."User"("UserId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Deposit" ADD CONSTRAINT "Deposit_ClientId_fkey" FOREIGN
KEY ("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "DepositRecord" ADD CONSTRAINT
"Ref_DepositRecord_to_Deposit" FOREIGN KEY ("DepositId")
    REFERENCES "Deposit"("DepositId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Employee" ADD CONSTRAINT "Employee_UserId_fkey" FOREIGN
KEY ("UserId")
    REFERENCES "Security"."User"("UserId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;
```

```
ALTER TABLE "Employee" ADD CONSTRAINT "Employee_AgencyId_fkey" FOREIGN
KEY ("AgencyId")
    REFERENCES "Agency"("AgencyId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "GuideComment" ADD CONSTRAINT "Ref_GuideComment_to_Guide"
FOREIGN KEY ("GuideId")
    REFERENCES "Guide"("GuideId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Order" ADD CONSTRAINT "Ref_Order_to_Client" FOREIGN KEY
("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Order" ADD CONSTRAINT "Order_OrderStateId_fkey" FOREIGN
KEY ("OrderStateId")
    REFERENCES "OrderState"("OrderStateId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "OrderProducts" ADD CONSTRAINT "Ref_OrderProducts_to_Order"
FOREIGN KEY ("OrderId")
    REFERENCES "Order"("OrderId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "PostBox" ADD CONSTRAINT "Ref_PostBox_to_Client" FOREIGN
KEY ("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "PostBox" ADD CONSTRAINT "Ref_PostBox_to_TypePostBox"
FOREIGN KEY ("TypePostBoxId")
    REFERENCES "TypePostBox"("TypePostBoxId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Product" ADD CONSTRAINT "Ref_Product_to_Guide" FOREIGN KEY
("GuideId")
    REFERENCES "Guide"("GuideId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;
```

```
ALTER TABLE "Product" ADD CONSTRAINT "Ref_Product_to_Shipping" FOREIGN
KEY ("ShippingId")
    REFERENCES "Shipping"("ShippingId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Product" ADD CONSTRAINT "Product_ReceivedBoxId_fkey"
FOREIGN KEY ("ReceivedBoxId")
    REFERENCES "ReceivedBox"("ReceivedBoxId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "ProductState" ADD CONSTRAINT "Ref_ProductState_to_Product"
FOREIGN KEY ("ProductId")
    REFERENCES "Product"("ProductId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "ReceivedBox" ADD CONSTRAINT "Ref_ReceivedBox_to_PostBox"
FOREIGN KEY ("PostBoxId")
    REFERENCES "PostBox"("PostBoxId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "ReceivedBox_has_ServicesReciveBox" ADD CONSTRAINT
"Ref_ReceivedBox_has_ServicesReciveBox_to_ReceivedBox" FOREIGN KEY
("ReceivedBox_ReceivedBoxId")
    REFERENCES "ReceivedBox"("ReceivedBoxId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "ReceivedBox_has_ServicesReciveBox" ADD CONSTRAINT
"Ref_ReceivedBox_has_ServicesReciveBox_to_ServicesReciveBox" FOREIGN
KEY ("ServicesReciveBox_ServicesReciveBoxId")
    REFERENCES "ServicesReciveBox"("ServicesReciveBoxId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Refund" ADD CONSTRAINT "Refund_ClientId_fkey" FOREIGN KEY
("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "ShippingState" ADD CONSTRAINT
"Ref_ShippingState_to_Shipping" FOREIGN KEY ("ShippingId")
    REFERENCES "Shipping"("ShippingId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
```

```
    NOT DEFERRABLE;

ALTER TABLE "TrackingsNumOrders" ADD CONSTRAINT
"Ref_TrackingsNumOrders_to_Order" FOREIGN KEY ("OrderId")
    REFERENCES "Order"("OrderId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."BankAccount" ADD CONSTRAINT
"Ref_BankAccount_to_AccountType" FOREIGN KEY ("AccountTypeId")
    REFERENCES "Bank"."AccountType"("AccountTypeId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."BankAccount" ADD CONSTRAINT
"Ref_BankAccount_to_Bank" FOREIGN KEY ("BankId")
    REFERENCES "Bank"."Bank"("BankId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."CreditMovements" ADD CONSTRAINT
"Ref_CreditMovements_to_Client" FOREIGN KEY ("ClientId")
    REFERENCES "Client"("ClientId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."CreditMovements" ADD CONSTRAINT
"Ref_CreditMovements_to_CreditMovementsImported" FOREIGN KEY
("CreditMovementsImportedId")
    REFERENCES
"Bank"."CreditMovementsImported"("CreditMovementsImportedId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."CreditMovements" ADD CONSTRAINT
"Ref_CreditMovements_to_CreditReason" FOREIGN KEY ("CreditReasonId")
    REFERENCES "Bank"."CreditReason"("CreditReasonId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."CreditMovements" ADD CONSTRAINT
"Ref_CreditMovements_to_CreditMovementStatus" FOREIGN KEY
("CreditMovementStatusId")
    REFERENCES "Bank"."CreditMovementStatus"("CreditMovementStatusId")
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Bank"."CreditMovements" ADD CONSTRAINT
"Ref_CreditMovements_to_BankAccount" FOREIGN KEY ("BankAccountId")
    REFERENCES "Bank"."BankAccount"("BankAccountId")
```

```
    MATCH SIMPLE
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE;

ALTER TABLE "Security"."UserClaim" ADD CONSTRAINT
"Ref_UserClaim_to_User" FOREIGN KEY ("UserId")
    REFERENCES "Security"."User"("UserId")
    MATCH SIMPLE
    ON DELETE RESTRICT
    ON UPDATE NO ACTION
    NOT DEFERRABLE;


CREATE OR REPLACE VIEW "View_AgencyWarehouse" AS
    SELECT ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."LocationAgencyRef" = ag."AgencyId") AND
("Product"."Location" !~~ '%IN TRANSIT%'::text) AND
("Product"."ShippingId" =
'60000005-7004-8003-9002-100000000001'::uuid))) AS "TotalInAgency",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."LocationAgencyRef" = ag."AgencyId") AND
("Product"."Location" ~~ '%IN TRANSIT%'::text))) AS
"TotalInTransitSented",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM ("Product"
             JOIN "Guide" USING ("GuideId"))
          WHERE (("Guide"."DestinationAgencyRef" = ag."AgencyId") AND
("Product"."Location" ~~ '%IN TRANSIT%'::text))
          GROUP BY "Guide"."DestinationAgencyRef") AS
"TotalInTransitToReceive",
    ag."AgencyId",
    ag."Name",
    ag."Description",
    ag."EmployeeOwnerRef",
    ag."Logo",
    ag."Description2",
    ag."Portada",
    ag."Active",
    ag."CityId"
   FROM "Agency" ag;

ALTER TABLE "View_AgencyWarehouse" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_Bill" AS
    SELECT bc."ClientId",
    bc."BillClientId",
    bc."Date",
    bc."TypeTrans",
    bc."ReferenceId",
    COALESCE(( SELECT t1."Value"
           FROM "BillClient" t1
          WHERE ((t1."TypeTrans" = 'CRED'::text) AND (t1."BillClientId"
= bc."BillClientId"))
```

```
          ORDER BY bc."Date"), (0)::real) AS "InValue",
    COALESCE(( SELECT t2."Value"
           FROM "BillClient" t2
          WHERE ((t2."TypeTrans" = 'DEB'::text) AND (t2."BillClientId"
= bc."BillClientId"))
          ORDER BY bc."Date"), (0)::real) AS "OutValue",
    bc."Balance",
    bc."Deleted"
   FROM "BillClient" bc
  WHERE (bc."Deleted" = false);

ALTER TABLE "View_Bill" OWNER TO "postgres";



CREATE OR REPLACE VIEW "View_Bill_Saldos" AS
    SELECT sd."ClientId",
    sd."InstagramName",
    sd."Phone",
    sd."Dni",
    sd."City",
    sd."Serie",
    sd."Name",
    COALESCE((sum(t1."InValue") - sum(t1."OutValue")), (0)::real) AS
"SaldoDisponible",
    sd."PorVerificar"
   FROM ("View_Saldo_Disponible" sd
     LEFT JOIN "View_Bill" t1 USING ("ClientId"))
  GROUP BY sd."ClientId", sd."PorVerificar", sd."Serie",
sd."InstagramName", sd."Phone", sd."Dni", sd."City", sd."Name";

ALTER TABLE "View_Bill_Saldos" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_ClientsWarehouse" AS
    SELECT cl."ClientId",
    cl."Serie",
    concat(cl."FirstName", ' ', cl."LastName") AS "Name",
    pr."LocationAgencyRef",
    pr."Location",
        CASE
            WHEN ((pr."Location" !~~ '%IN TRANSIT%'::text) AND
(pr."ShippingId" = '60000005-7004-8003-9002-100000000001'::uuid)) THEN
concat('["', count(pr."ProductId"), '","',
round((COALESCE(sum(pr."Weight"), (0)::real))::numeric, 2), '"]')
            ELSE NULL::text
        END AS "TotalInAgency",
        CASE
            WHEN (pr."Location" ~~ '%IN TRANSIT%'::text) THEN
concat('["', count(pr."ProductId"), '","',
round((COALESCE(sum(pr."Weight"), (0)::real))::numeric, 2), '"]')
            ELSE NULL::text
        END AS "ToReceive",
    gd."DestinationAgencyRef",
    pr."ShippingId"
   FROM ((((("Client" cl
     JOIN "PostBox" pb ON ((pb."ClientId" = cl."ClientId")))
     JOIN "ReceivedBox" rb ON ((rb."PostBoxId" = pb."PostBoxId")))
     JOIN "Product" pr ON ((pr."ReceivedBoxId" = rb."ReceivedBoxId")))
     JOIN "Guide" gd ON ((gd."GuideId" = pr."GuideId")))
  WHERE (pr."ShippingId" =
'60000005-7004-8003-9002-100000000001'::uuid)
```

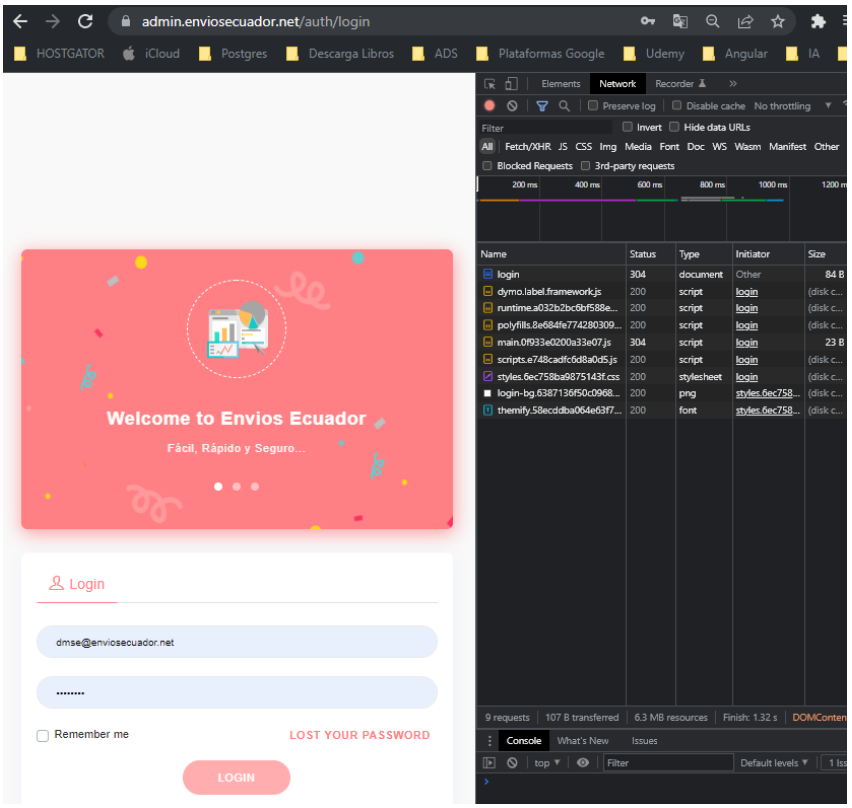
```
  GROUP BY cl."ClientId", pr."LocationAgencyRef", pr."Location",
pr."ShippingId", gd."DestinationAgencyRef"
  ORDER BY cl."ClientId";

ALTER TABLE "View_ClientsWarehouse" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_Deposits" AS
    SELECT d."DepositId",
    d."Date",
    concat(c."FirstName", ' ', c."LastName") AS concat,
    d."Bank",
    d."N_Ref",
    d."Verified",
    d."UserVerified",
    d."Image",
    d."Amount",
    d."Active",
    d."ClientId"
   FROM ("Deposit" d
     JOIN "Client" c USING ("ClientId"));

ALTER TABLE "View_Deposits" OWNER TO "postgres";



CREATE OR REPLACE VIEW "View_Guide" AS
    SELECT ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE ("Product"."GuideId" = sh."GuideId")) AS
"ProductsTotal",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."GuideId" = sh."GuideId") AND
("Product"."ReceivedInGuide" ~~ 'YES,%'::text))) AS "ProductsReceived",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."GuideId" = sh."GuideId") AND
("Product"."ReceivedInGuide" !~~ 'YES,%'::text))) AS
"ProductsPendindg",
        CASE
            WHEN (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS concat
               FROM "Product"
              WHERE ("Product"."GuideId" = sh."GuideId")) =
(0)::numeric) THEN concat('EMPTY')
            WHEN (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS concat
               FROM "Product"
              WHERE (("Product"."GuideId" = sh."GuideId") AND
("Product"."ReceivedInGuide" !~~ 'YES,%'::text))) = (0)::numeric) THEN
concat('COMPLETE')
            WHEN (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS concat
               FROM "Product"
```

```
            WHERE (("Product"."GuideId" = sh."GuideId") AND
("Product"."ReceivedInGuide" ~~ 'YES,%'::text))) = (0)::numeric) THEN
concat('PENDING')
            ELSE concat('INCOMPLETE')
        END AS "Status",
    sh."GuideId",
    sh."Date",
    sh."EmployeeDelivery",
    sh."NumGuide",
    sh."DateRecived",
    sh."EmployeeRecived",
    sh."Link",
    sh."CompanyCourrier",
    sh."Active",
    sh."Weight",
    sh."Cost",
    sh."BillRecived",
    sh."DestinationAgencyRef"
   FROM "Guide" sh
  WHERE (sh."GuideId" <> '60000005-7004-8003-9002-100000000001'::uuid)
  ORDER BY sh."Date" DESC;

ALTER TABLE "View_Guide" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_Refunds" AS
    SELECT d."RefundId",
    d."Date",
    concat(c."FirstName", ' ', c."LastName") AS "Name",
    d."Bank",
    d."N_Ref",
    d."TypeAccount",
    d."NumAccount",
    d."Image",
    d."Amount",
    d."Check",
    d."UserCheck",
    d."NameAccount",
    d."DNIAcccount",
    d."Active",
    d."ClientId"
   FROM ("Refund" d
     JOIN "Client" c USING ("ClientId"));

ALTER TABLE "View_Refunds" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_Saldo_Disponible" AS
    SELECT cl."ClientId",
    cl."InstagramName",
    cl."Phone",
    cl."Dni",
    cl."City",
    cl."Serie",
    concat(cl."FirstName", ' ', cl."LastName") AS "Name",
    COALESCE(sum(
        CASE
            WHEN ((dep."CreditMovementStatusId" =
'd1fb5e22-f539-140d-8508-d8bb837d5f40'::uuid) AND (dep."Active" = true)
AND (dep."ClientId" = cl."ClientId")) THEN dep."Value"
            ELSE (0)::real
        END), (0)::real) AS "PorVerificar"
```

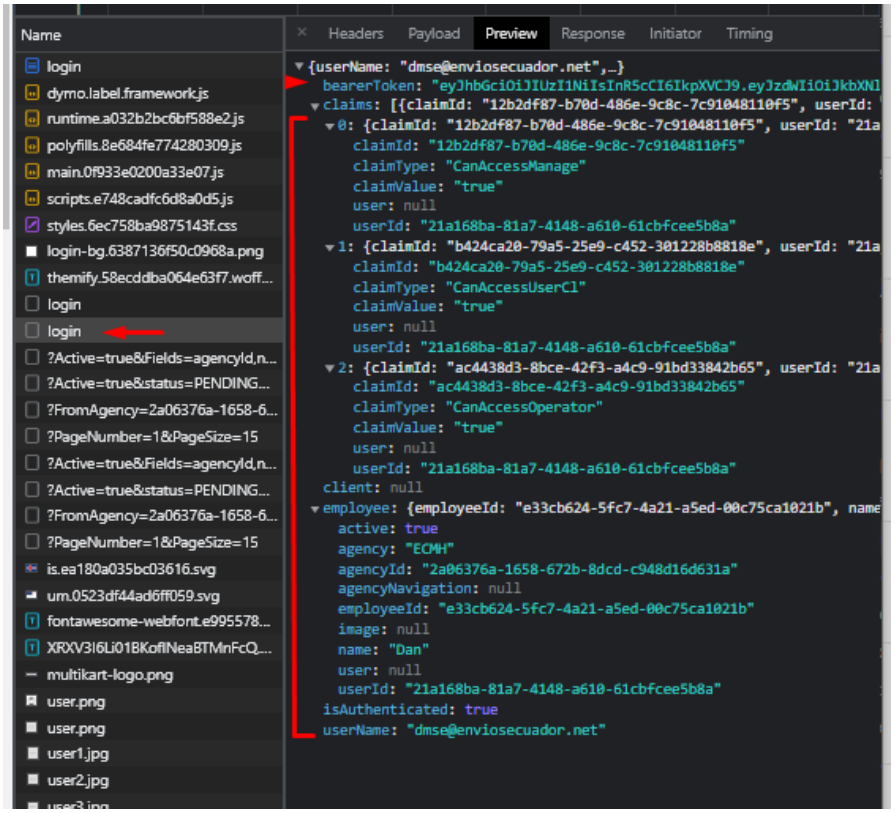
```
    FROM ("Client" cl
      LEFT JOIN "Bank"."CreditMovements" dep USING ("ClientId"))
  GROUP BY cl."ClientId";

ALTER TABLE "View_Saldo_Disponible" OWNER TO "enviosecuador";



CREATE OR REPLACE VIEW "View_Shipping" AS
    SELECT ( SELECT concat(btrim("Client"."FirstName"), ' ',
btrim("Client"."LastName")) AS "Client"
           FROM "Client"
          WHERE ("Client"."ClientId" = sh."ClientRef")) AS "Client",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."ShippingOrderByEnvec" = true))) AS
"ProductsRequestByEnvec",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."ShippingOrderByCli" = true))) AS "ProductsRequestByClient",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE ("Product"."ShippingId" = sh."ShippingId")) AS
"ProductsTotal",
    ( SELECT concat(count("Product"."ProductId"), 'u in ',
round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric, 2),
'Lb') AS concat
           FROM "Product"
          WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."Shipped" = true))) AS "ProductsSend",
    ( SELECT count("Product"."ProductId") AS count
           FROM "Product"
          WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."Stock" = true))) AS "ProductsOfStock",
    ( SELECT COALESCE(sum("Product"."Weight"), (0)::real) AS "coalesce"
           FROM "Product"
          WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."Stock" = true))) AS "WeightOfStock",
        CASE
            WHEN (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS concat
                   FROM "Product"
                  WHERE ("Product"."ShippingId" = sh."ShippingId")) =
(0)::numeric) THEN concat('EMPTY')
            WHEN (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS round
                   FROM "Product"
                  WHERE ("Product"."ShippingId" = sh."ShippingId")) = (
SELECT round((COALESCE(sum("Product"."Weight"), (0)::real))::numeric,
2) AS round
                   FROM "Product"
                  WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."Shipped" = true)))) THEN concat('COMPLETE')
            WHEN ((( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS round
                   FROM "Product"
```

```
                WHERE ("Product"."ShippingId" = sh."ShippingId")) >
(0)::numeric) AND (( SELECT round((COALESCE(sum("Product"."Weight"),
(0)::real))::numeric, 2) AS round
                 FROM "Product"
                WHERE (("Product"."ShippingId" = sh."ShippingId") AND
("Product"."Shipped" = true))) = (0)::numeric)) THEN concat('PENDING')
              ELSE concat('INCOMPLETE')
          END AS "Status",
      sh."ShippingId",
      sh."Company",
      sh."NumGuide",
      sh."Weight",
      sh."Address",
      sh."CostLb",
      sh."MoneyCharged",
      sh."Active",
      sh."Date",
      sh."ClientRef",
      sh."Image",
      sh."FullAddress",
      sh."EnteredBy",
      sh."Editable",
      sh."FromAgency"
     FROM "Shipping" sh
    WHERE (sh."ShippingId" <>
'60000005-7004-8003-9002-100000000001'::uuid)
    ORDER BY sh."Date" DESC;

ALTER TABLE "View_Shipping" OWNER TO "enviosecuador";
```

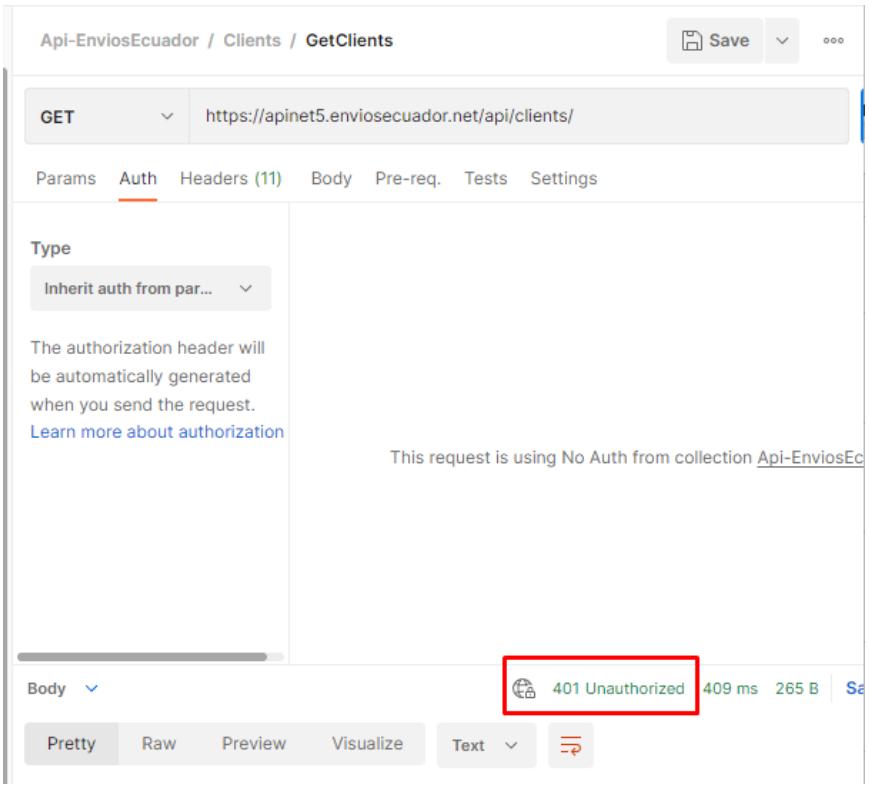
Anexo 3: Evaluación de prototipo - Generación del token al autenticarse 1



Anexo 4: Evaluación de prototipo - Generación del token al autenticarse 2

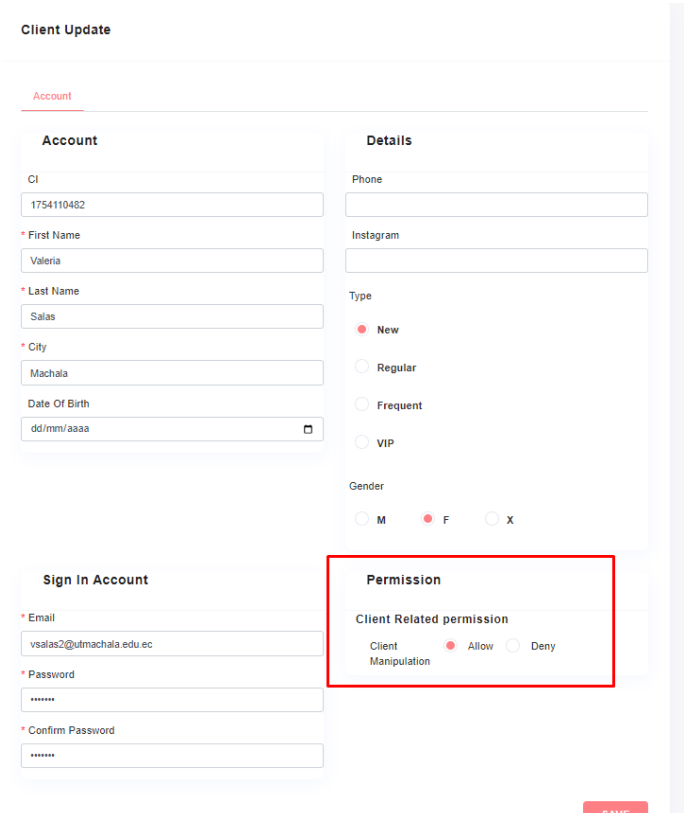


Anexo 5: Evaluación de prototipo – Denegar peticiones a usuarios no autorizados



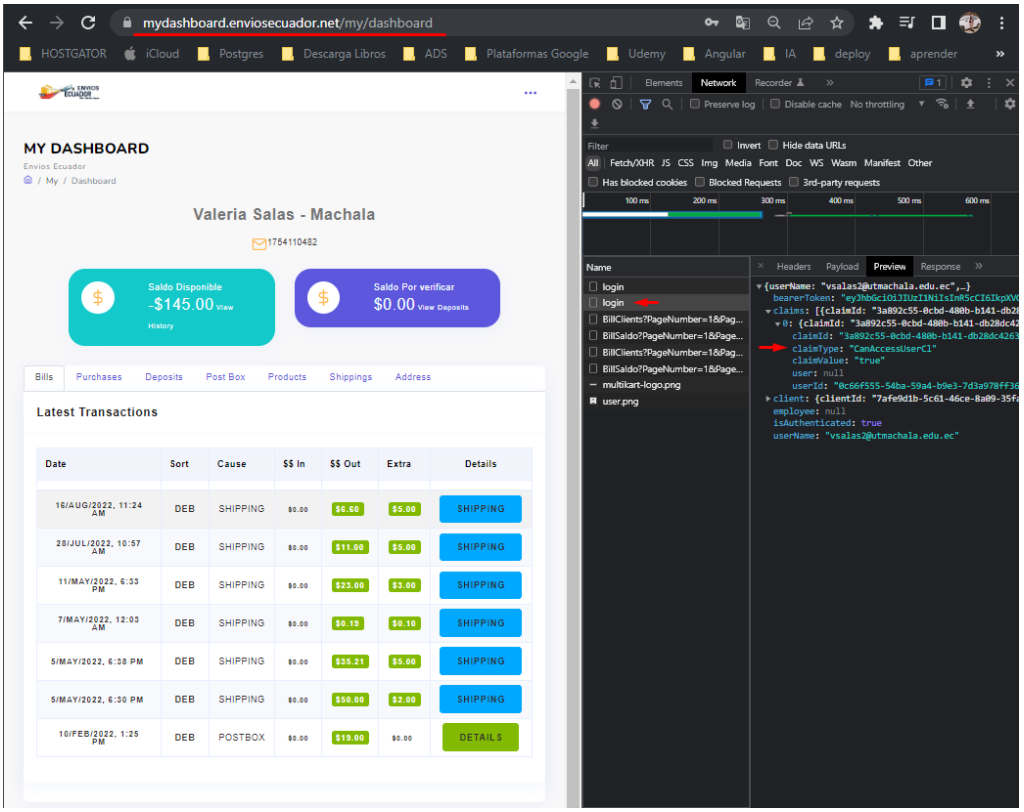
Anexo 6: Evaluación de prototipo – Gestionar permisos de clientes 1

Se le permite el acceso a la información relacionada con su Id de cliente:



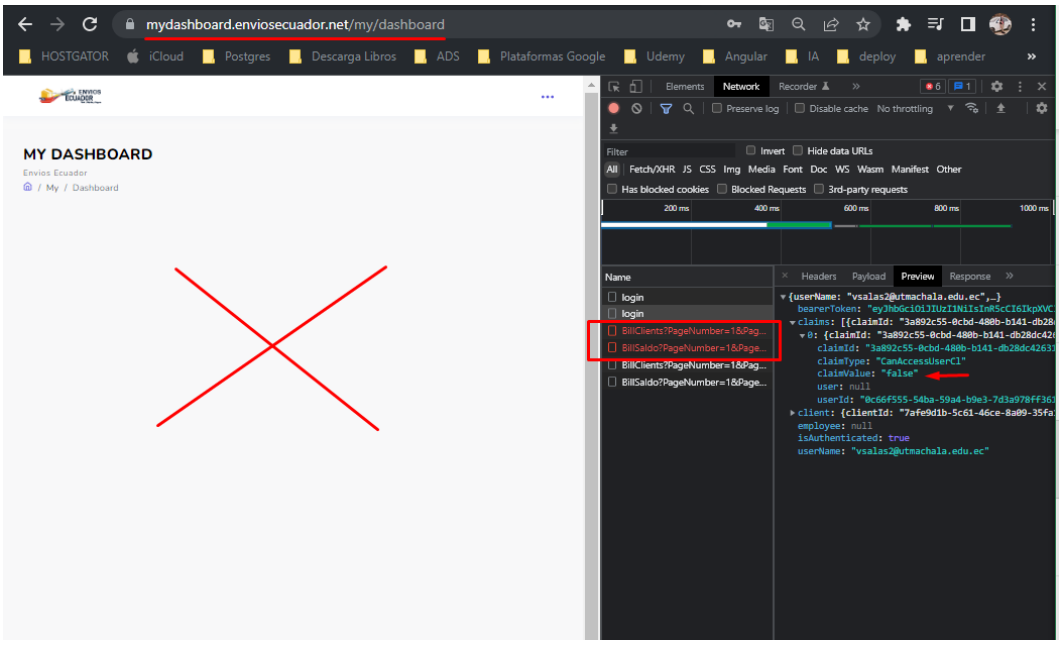
Anexo 7: Evaluación de prototipo – Gestionar permisos de clientes 2

El punto de entrada le concede acceso a los servicios que está autorizado (la información relacionada con su Id de cliente).



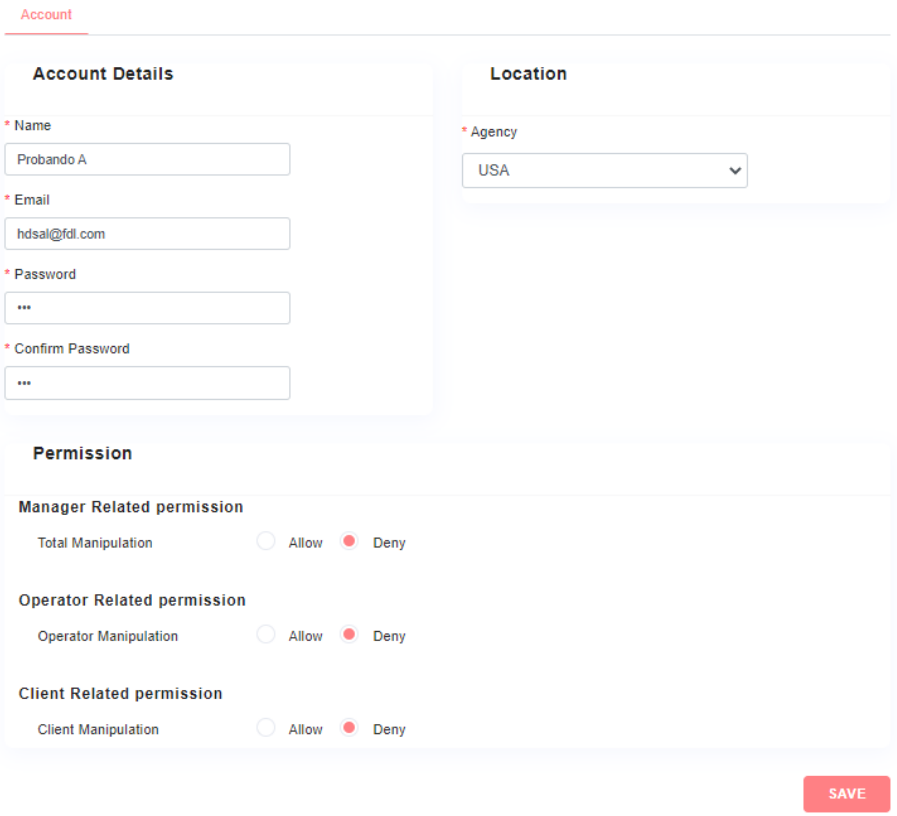
Anexo 8: Evaluación de prototipo – Gestionar permisos de clientes 3

Cuando se retira el permiso de acceso a su información, el servicio le niega la entrada a los métodos.



Anexo 9: Evaluación de prototipo – Gestionar permisos de empleados 1

Con el acceso a toda la información en clientes se puede ver toda la información personal de cada uno



Anexo 10: Evaluación de prototipo – Gestionar permisos de empleados 2

Se puede ver los campos Instagram y teléfono como usuario con todos los permisos.

Anexo 11: Evaluación de prototipo – Gestionar permisos de empleados 3

Ahora le quitaremos el permiso de administrador, entonces podemos ver que el servidor solo entrega la información que su nivel de acceso le permite

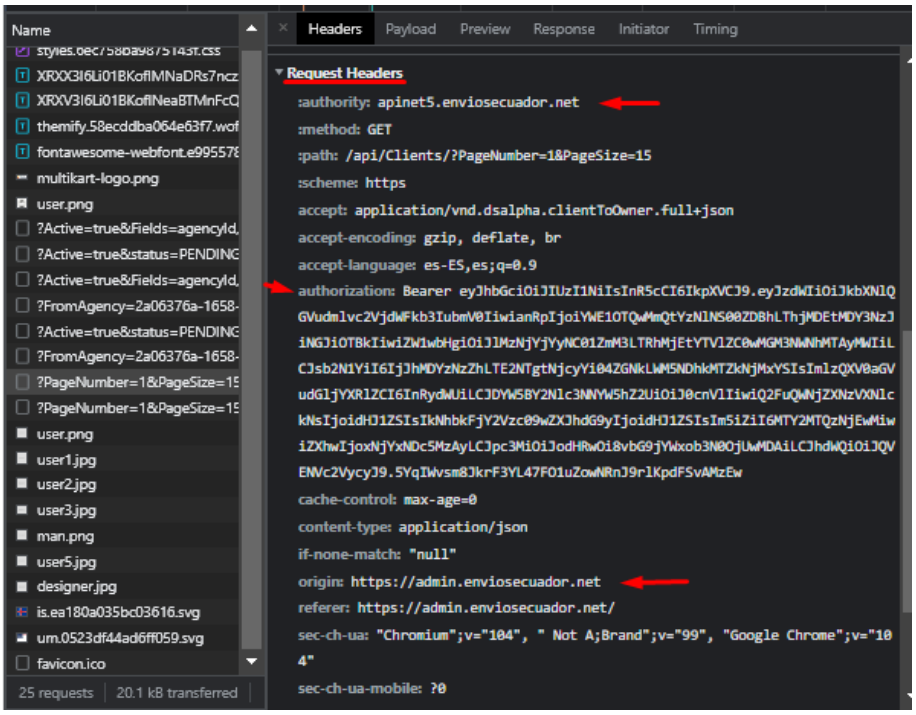
Anexo 12: Evaluación de prototipo – Envío de token en el header de todas las peticiones 1

Para asegurar el correcto funcionamiento respecto a la inserción del token en todas las peticiones https, se ha creado del lado del cliente una clase que implementa la interfaz HttpInterceptor, el cual vincula a todas las peticiones salientes el token almacenado en la sesión del cliente como vemos en la siguiente imagen.

src > app > shared > service > TS http-interceptor.model.ts > ...

```
import { Injectable, NgModule } from '@angular/core';
import { Observable, throwError } from 'rxjs'
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest, HttpErrorResponse, HttpStatusCode } from '@angular/common/http';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { SecurityService } from './security.service';
import { catchError, map } from 'rxjs/operators';


@Injectable()
export class HttpRequestInterceptor implements HttpInterceptor {

    constructor(private authenticationService:
        SecurityService) {}


    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        var token = sessionStorage.getItem('bearerToken');
        if(token == ''){
            token = '0'
        }
        if (token) {

            const newReq = req.clone(
                {
                    headers: req.headers.set('Authorization', 'Bearer ' + token)
                }
            )
            return next.handle(newReq).pipe(
                map((event: HttpEvent<any>) => {
                  return event;
                }),
                catchError(
                  (
                    httpErrorResponse: HttpErrorResponse,
                    _: Observable<HttpEvent<any>>
                  ) => {
                    if (httpErrorResponse.status === HttpStatusCode.Unauthorized) {
                      this.authenticationService.redirectToLogin();
                    }
                    return throwError(httpErrorResponse);
                  }
                )
            );
        }
        else {
            return next.handle(req);
        }
    }
}

@NgModule({
    providers: [{
        provide: HTTP_INTERCEPTORS,
        useClass: HttpRequestInterceptor,
        multi: true
    }]
})
export class HttpInterceptorModule {}
```

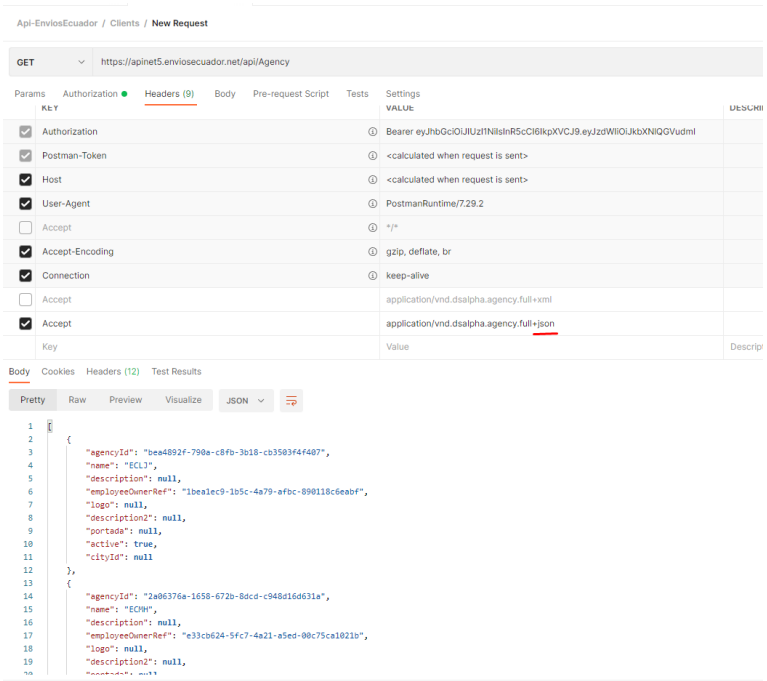
Anexo 13: Evaluación de prototipo – Envío de token en el header de todas las peticiones 2



Anexo 14: Evaluación de prototipo – Respuestas en formato Json 1

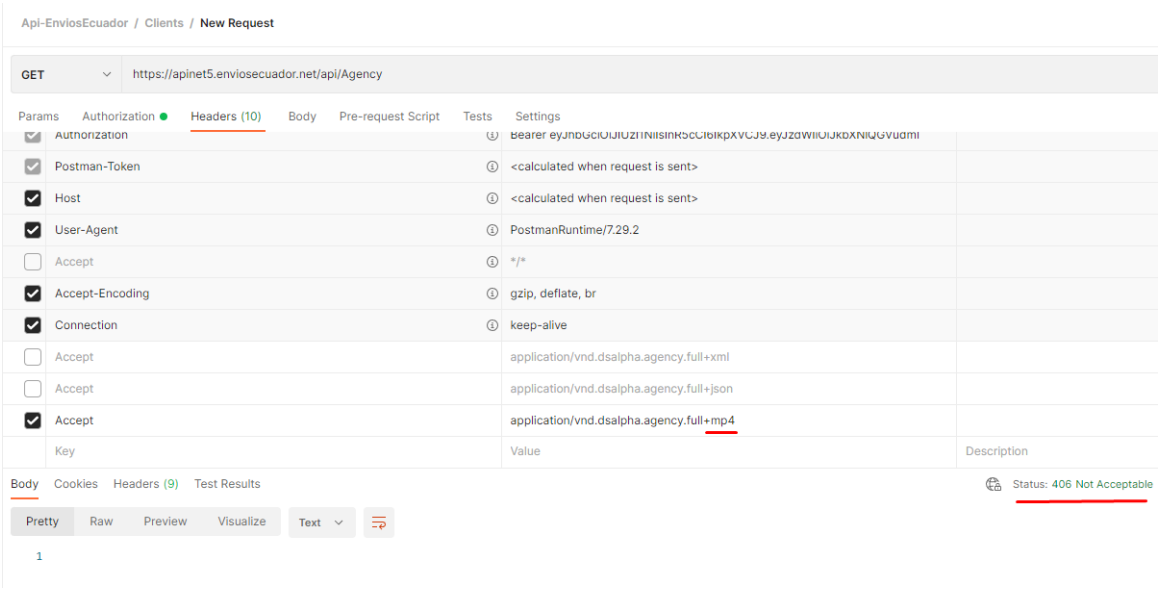
En la siguiente imagen podemos ver como el header del request en su accept indica el tipo de respuesta que espera del servidor

Respuesta en formato Json.



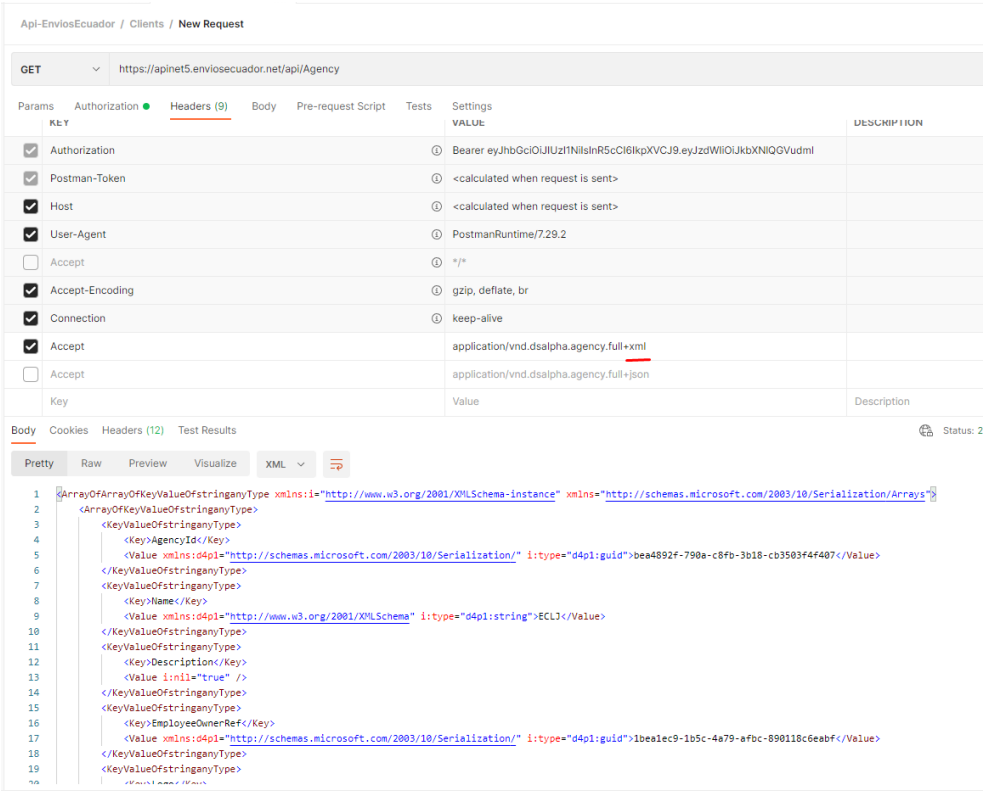
Anexo 15: Evaluación de prototipo – Respuestas en formato Json 2

Enviando una accept no configurada para la petición el servidor responde un error 406 indicando que el cliente está realizando una petición que no puede aceptar.



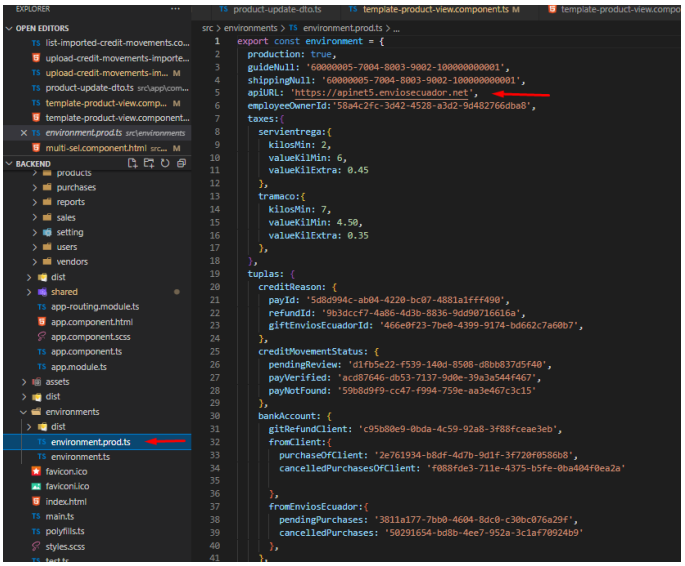
Anexo 16: Evaluación de prototipo – Respuestas en formato XML

Respuesta en formato XML



Anexo 17: Evaluación de prototipo – Conexión de capa presentación con capa de dominio 1

Es la capa de presentación quien almacena la URL base de todos los puntos de acceso a la API, es la única conexión que tiene para comunicarse con el servidor y esta llega directamente a la capa de dominio.



Anexo 18: Evaluación de prototipo – Conexión de capa presentación con capa de dominio 2

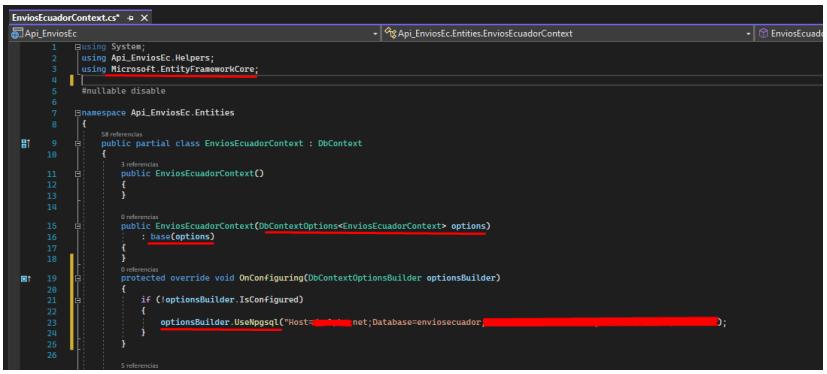
Podemos ver en la imagen como se ha codificado el servicio de peticiones a través de http únicamente con la URL que conecta con capa de dominio.

```
const API_URL = environment.apiURL + '/api/';

@Injectable({
  providedIn: 'root'
})
export class AgencyService {

  constructor(private http: HttpClient) { }

  // Agency
  // get all /api/Agency
  getAllAgencys(
    params: HttpParams,
    auxMediaType: string
  ): Observable<any> {
    var headers;
    var eTag;
    eTag = localStorage.getItem('If-None-Match-get------');

    headers = new HttpHeaders({
      'Content-Type': 'application/json',
      Accept: `${auxMediaType}`,
      'If-None-Match': `"${eTag}"`,
    });

    return this.http
      .get<any>(API_URL + 'Agency/', {
        headers: headers,
        observe: 'response' as 'body',
        responseType: 'json',
        params,
      })
      .pipe(
        tap((resp) => {
          if (resp.status == 200) {
            //   this.eTag = JSON.parse(resp.headers.get("ETag"));
            //    localStorage.setItem('If-None-Match-get-orders', this.eTag);
          }
        }),
        catchError(this.handleError)
      );
  }
```

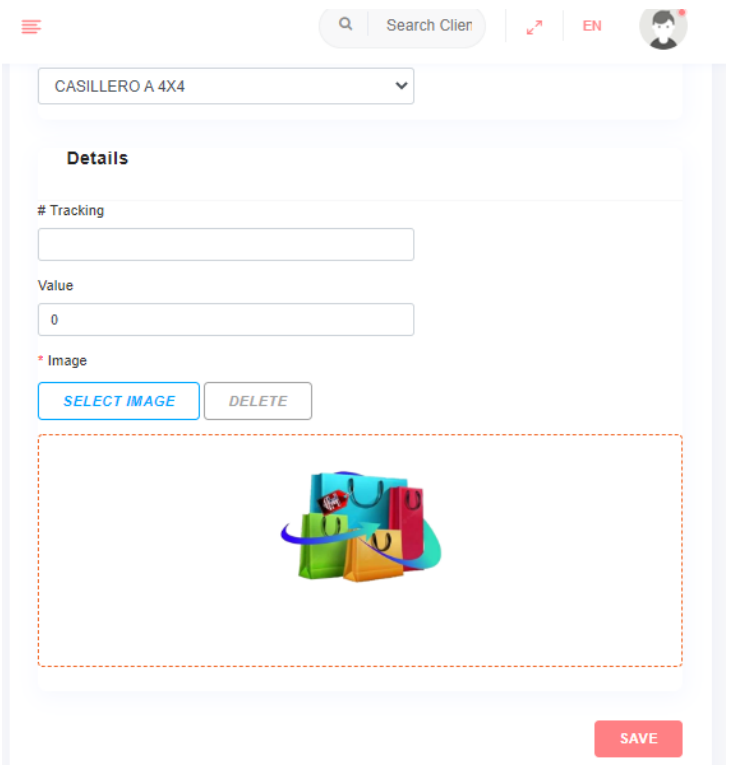
Anexo 19: Evaluación de prototipo – Conexión de capa dominio con capa de acceso a datos

Podemos ver la utilización del ORM Entity Framework Core para crear el vínculo de acceso a datos desde la capa de aplicación mediante inyección de dependencia utilizando el constructor del DbContext.

Esta es la única línea de conexión que usa la capa de aplicación.

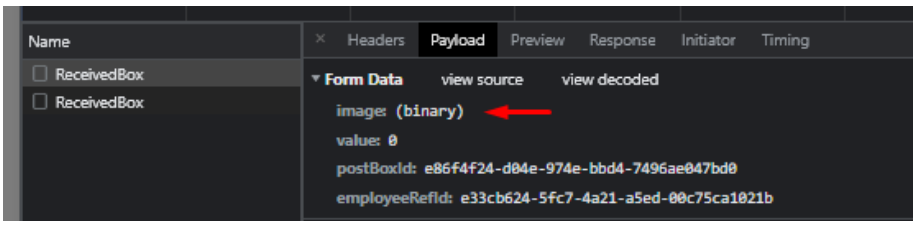


Anexo 20: Evaluación de prototipo – Envío de archivo tipo File 1



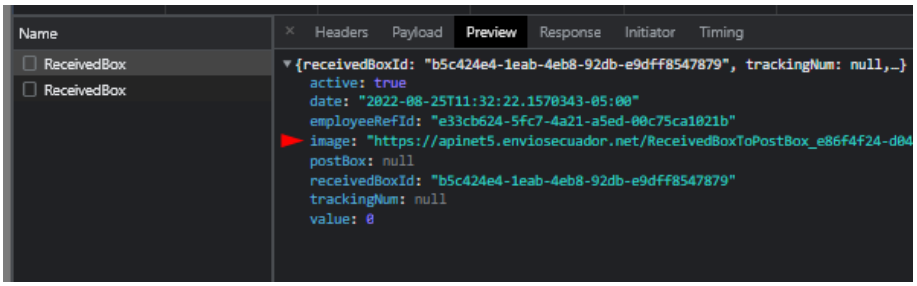
Anexo 21: Evaluación de prototipo – Envío de archivo tipo File 2

Podemos ver que se envía un archivo binario dentro de un formulario de datos.



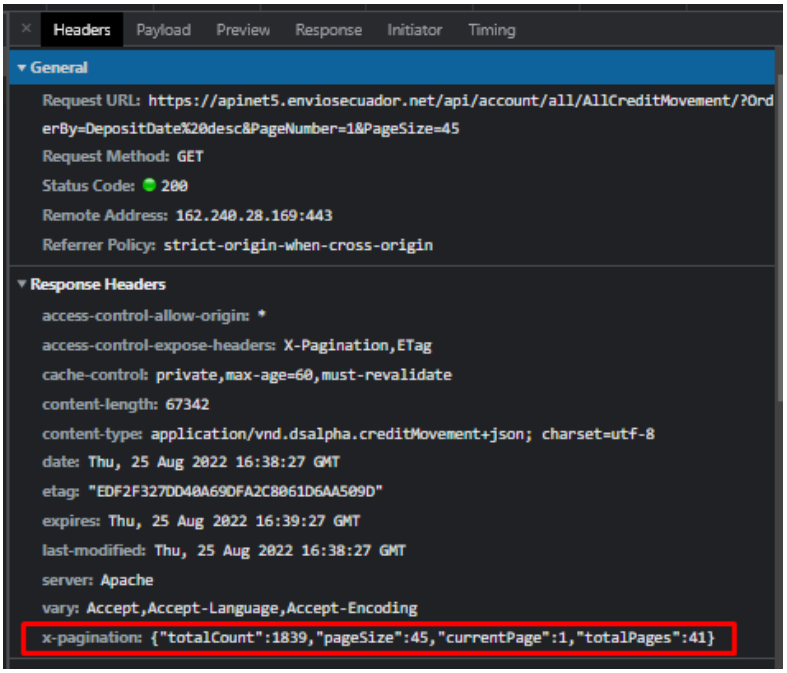
Anexo 22: Evaluación de prototipo – Envío de archivo tipo File 3

El servidor recibe el formato imagen y lo guarda en un espacio reservado del servidor, luego devuelve el link de acceso a la imagen.

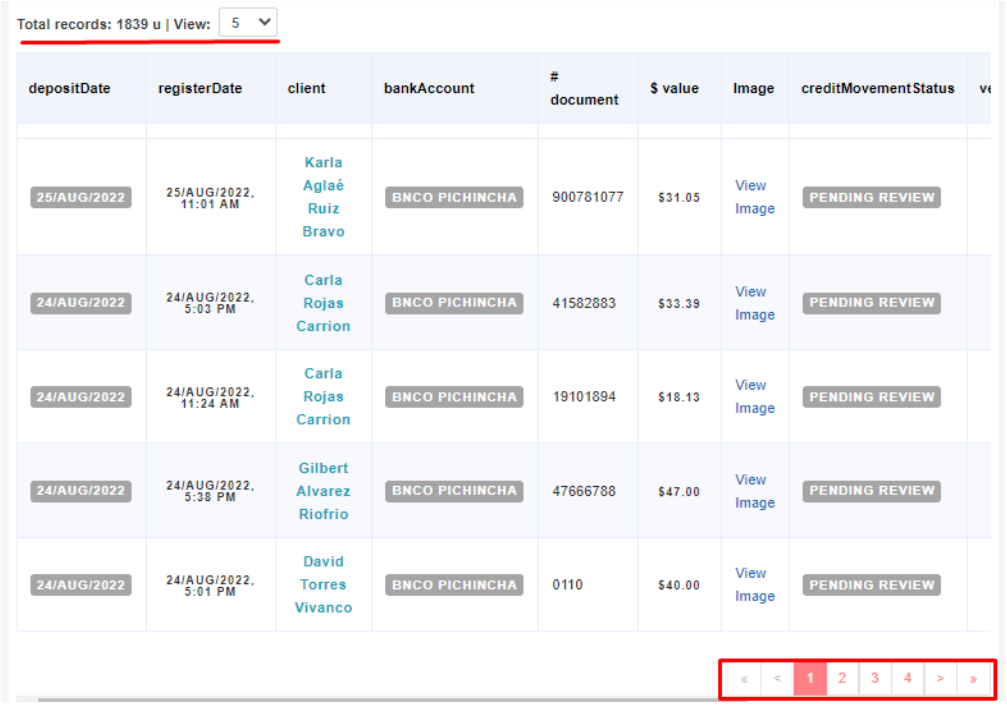


Anexo 23: Evaluación de prototipo – Paginación en listado de registros 1

Podemos ver los detalles de paginación que responde el servidor cuando se realiza una petición que devuelve una lista de registros.

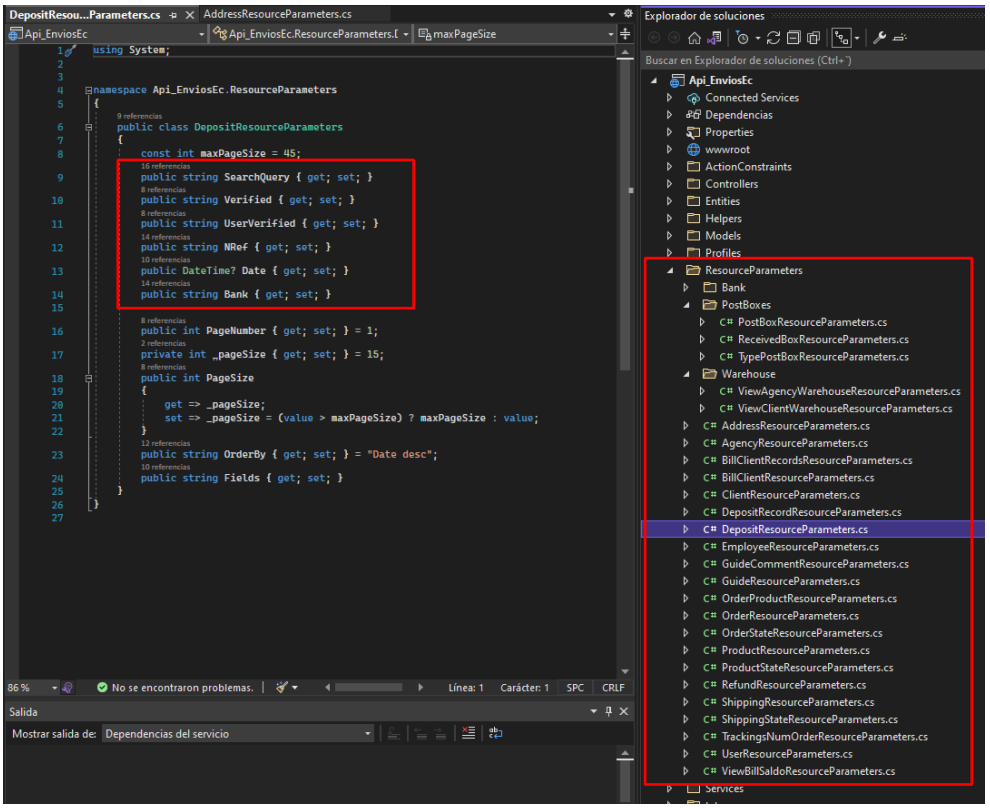


Anexo 24: Evaluación de prototipo – Paginación en listado de registros 2



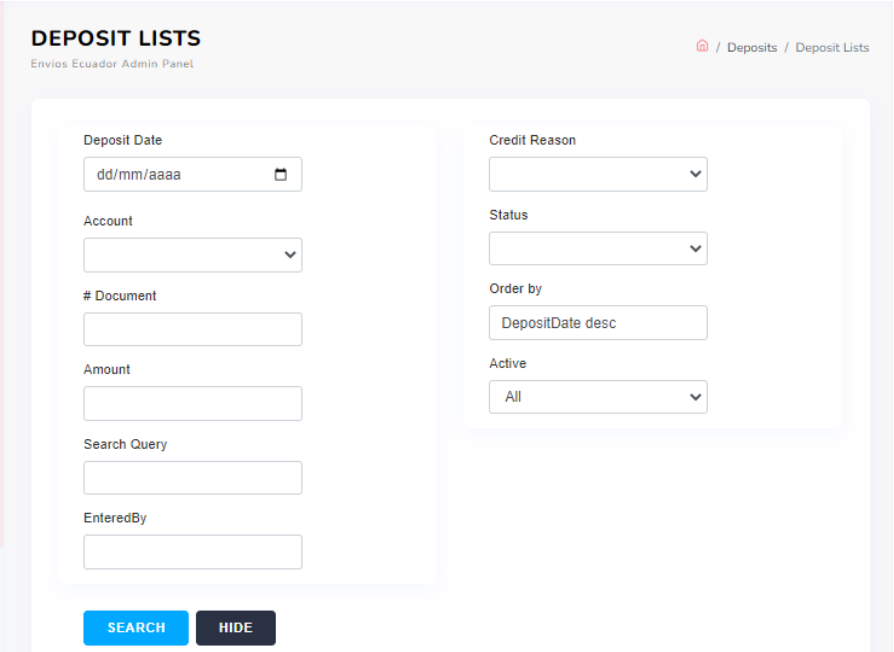
Anexo 25: Evaluación de prototipo – Filtros de búsqueda 1

El backend está preparado para recibir parámetros de búsqueda descritos en las peticiones que recibe.

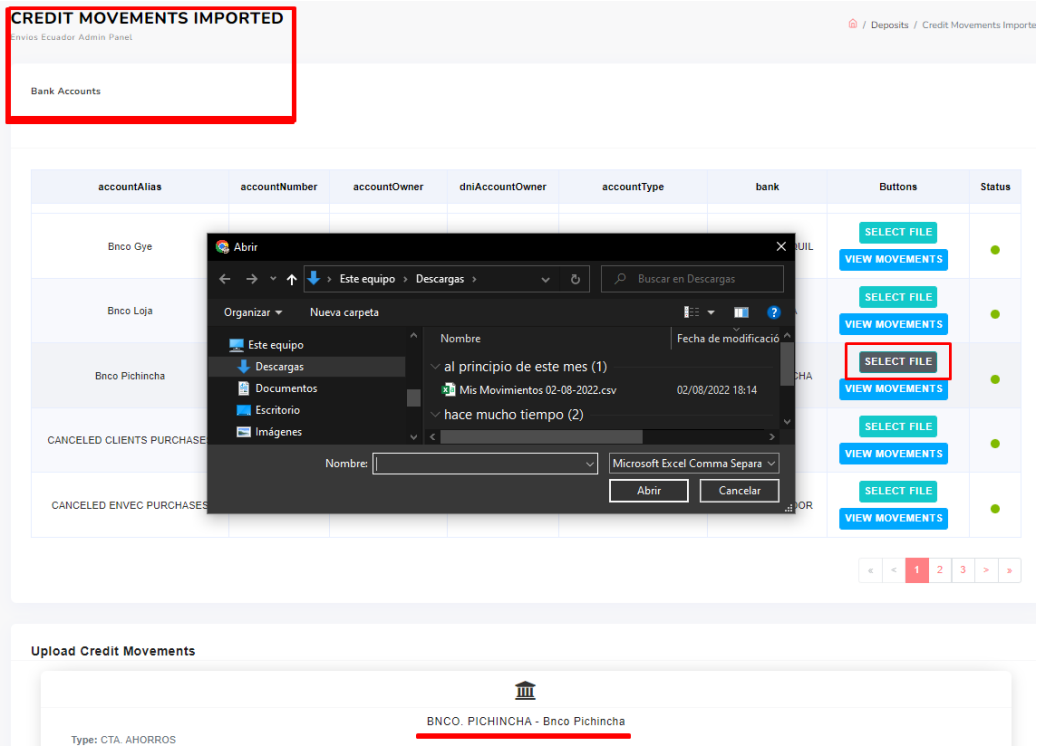


Anexo 26: Evaluación de prototipo – Filtros de búsqueda 2

El Frontend implementa campos de filtro para realizar búsquedas específicas en una petición.

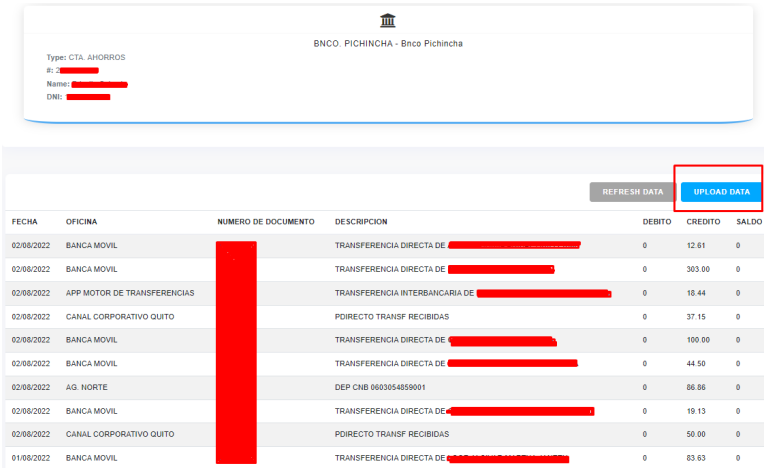


Anexo 27: Evaluación de prototipo – Carga masiva de movimientos bancarios 1



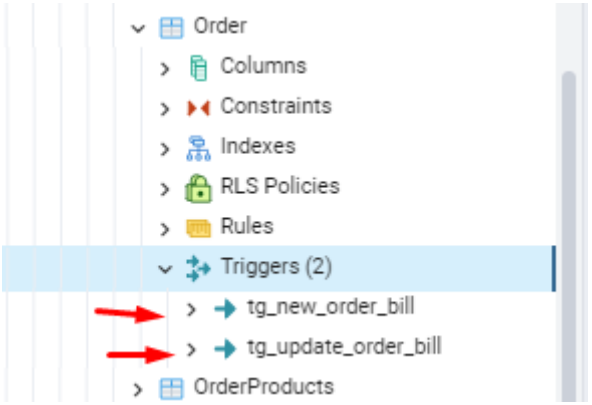
Anexo 28: Evaluación de prototipo – Carga masiva de movimientos bancarios 2

Podemos subir el reporte de cuenta generado por el banco para cargar dicha información al sistema y posteriormente se pueda realizar una verificación de los depósitos de clientes, siempre y cuando este exista en la lista importada que genera la entidad bancaria.



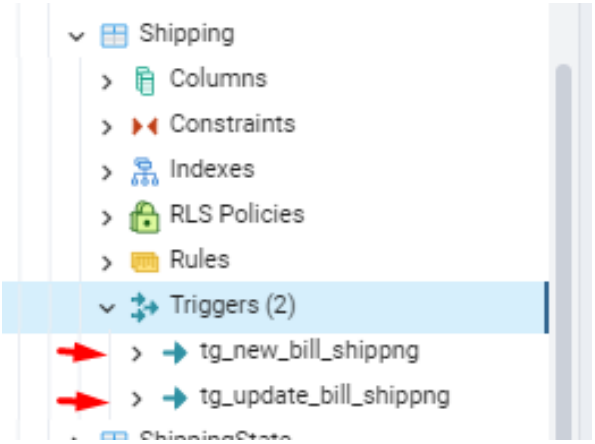
Anexo 29: Evaluación de prototipo – Verificación automática de depósitos

Anexo 30: Evaluación de prototipo – Creación de débito por compras 1

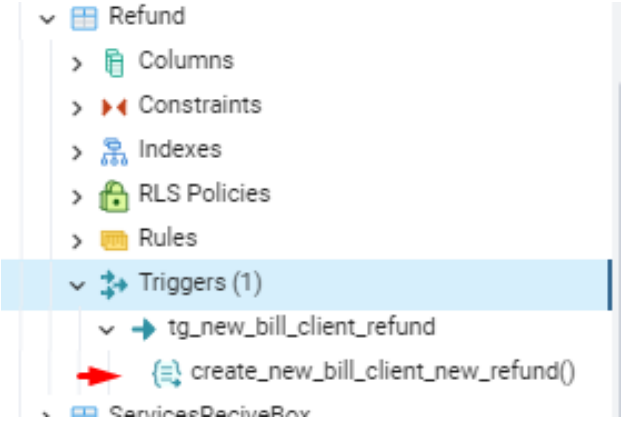


Anexo 31: Evaluación de prototipo – Creación de débito por compras 2

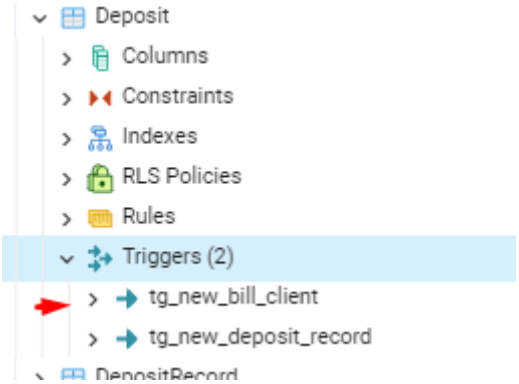
Anexo 32: Evaluación de prototipo – Creación de débito por despachos



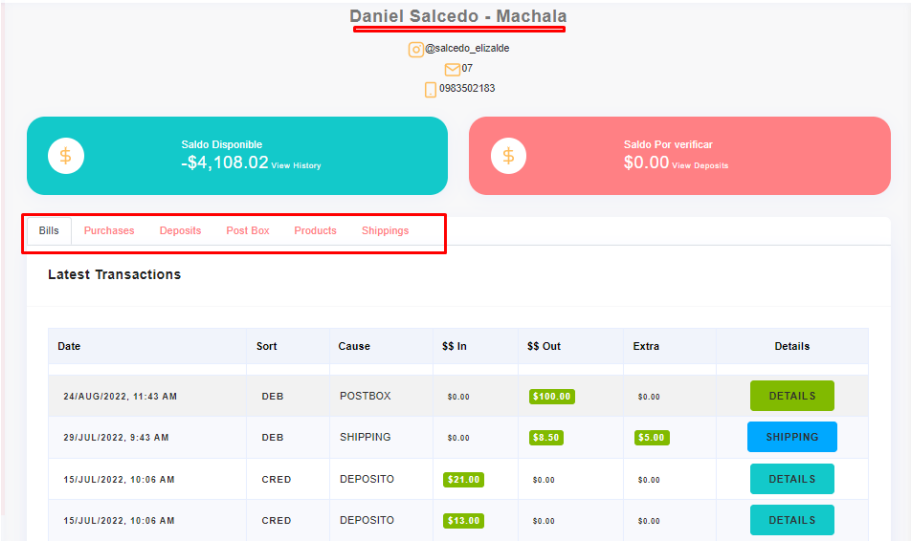
Anexo 33: Evaluación de prototipo – Creación de débitos por reembolsos



Anexo 34: Evaluación de prototipo – Creación de créditos por depósitos

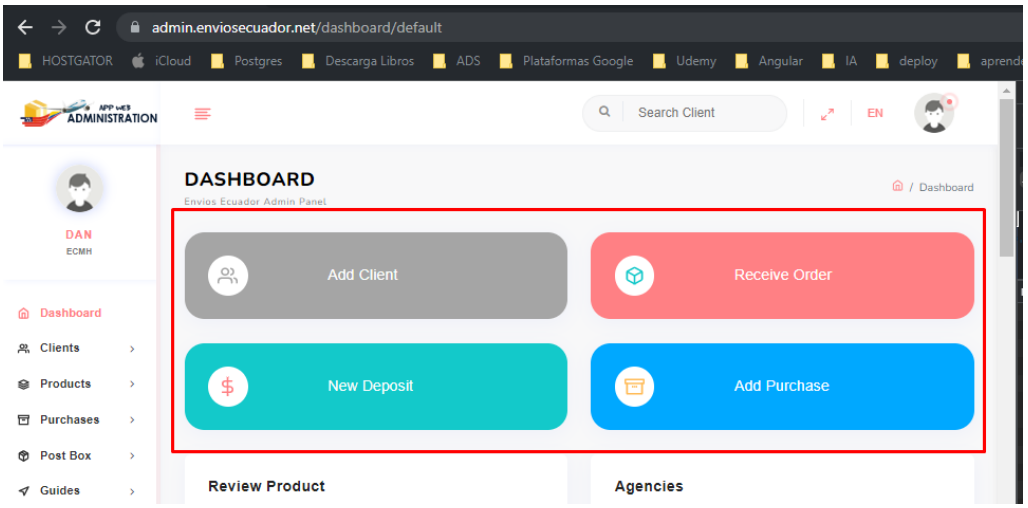


Anexo 35: Evaluación de prototipo – Perfil de cliente con sus movimientos del negocio



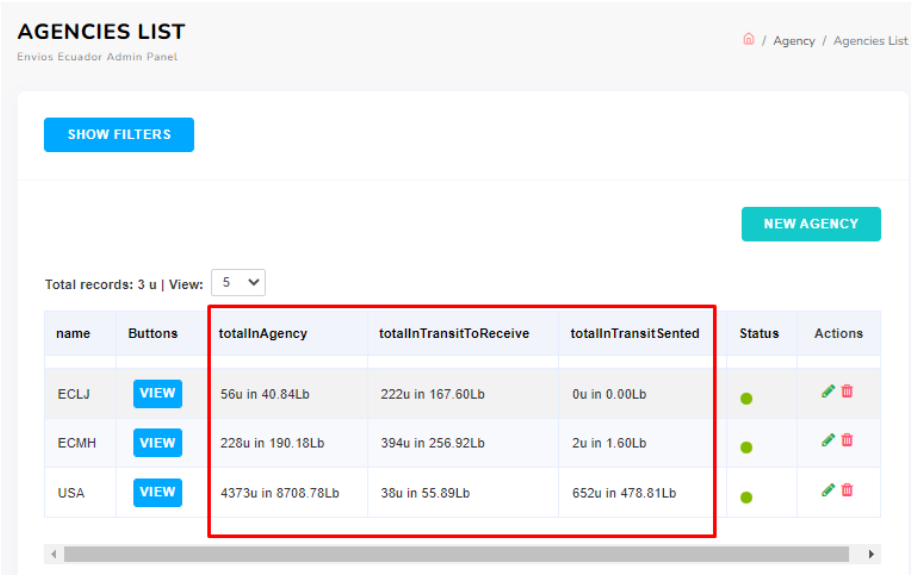
Anexo 36: Evaluación de prototipo – Funcionalidad de accesos rápidos en página de inicio

Se ha probado cada botón verificando la redirección de manera correcta.



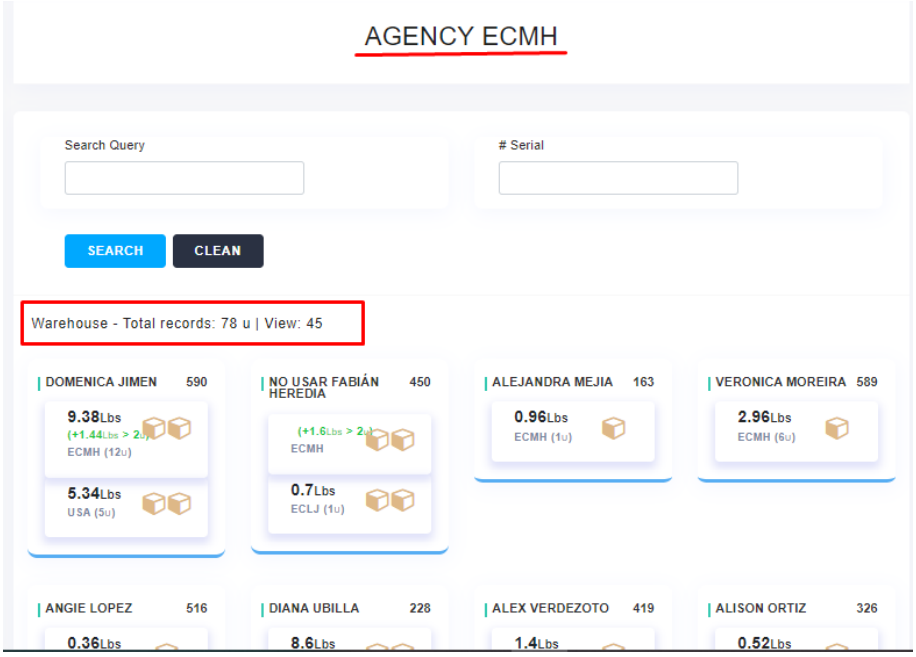
Anexo 37: Evaluación de prototipo – Vista de bodega por cada agencia

Se evidencia que la lista de agencias cuenta con un conteo individual del total almacenado en cada bodega.



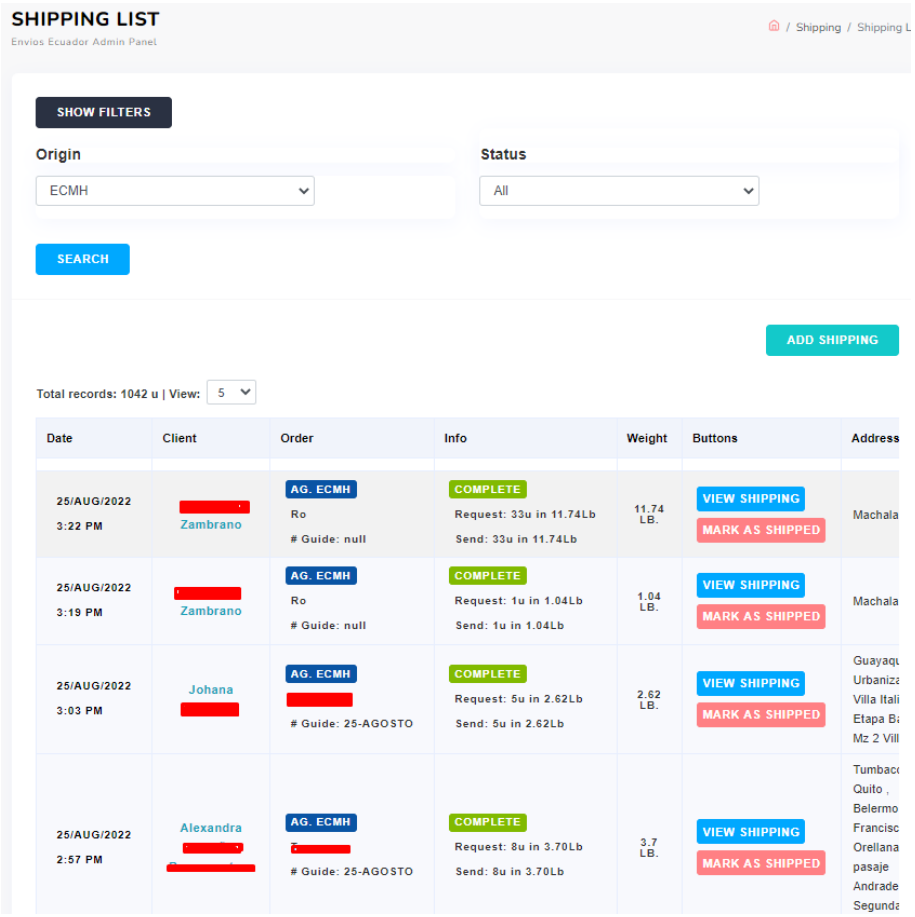
Anexo 38: Evaluación de prototipo – Vista de artículos en bodega por cliente

Se puede apreciar una lista detallada de todo el contenido almacenado en una bodega, agrupado por clientes.

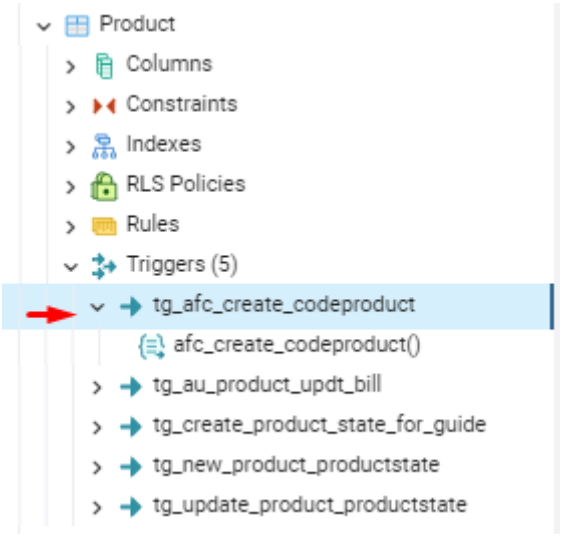


Anexo 39: Evaluación de prototipo – Vista de envíos con sus productos

Se puede apreciar un listado de todos los envíos realizados y organizados por fecha, con un indicador de cumplimiento y un botón para visualizar el contenido de cada envío.



Anexo 40: Evaluación de prototipo – Creación de etiquetado de productos 1



Anexo 41: Evaluación de prototipo – Creación de etiquetado de productos 2

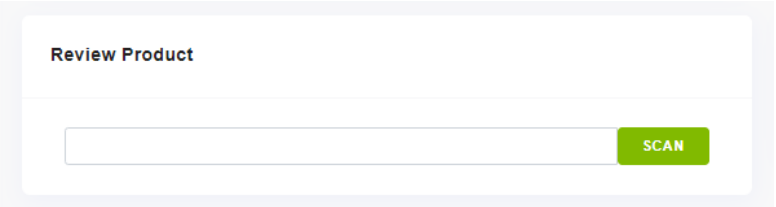
Se puede evidenciar que en la base de datos existe un disparador para crear un código único por cada registro que se inserta en dicha tabla.

tg_afc_create_codeproduct

General Definition Events Transition Code SQL

```
1
2   DECLARE
3    new_code TEXT;
4   BEGIN
5
6     LOOP
7         new_code := random_string_to_products_code();
8         BEGIN
9         update public."Product" set "Code"=new_code
10        where "ProductId" = new."ProductId";
11            EXIT;
12        EXCEPTION WHEN unique_violation THEN
13
14        END;
15    END LOOP;
16    return new;
17  END;
18
```

Anexo 42: Evaluación de prototipo – Campo para escanear códigos de productos 1

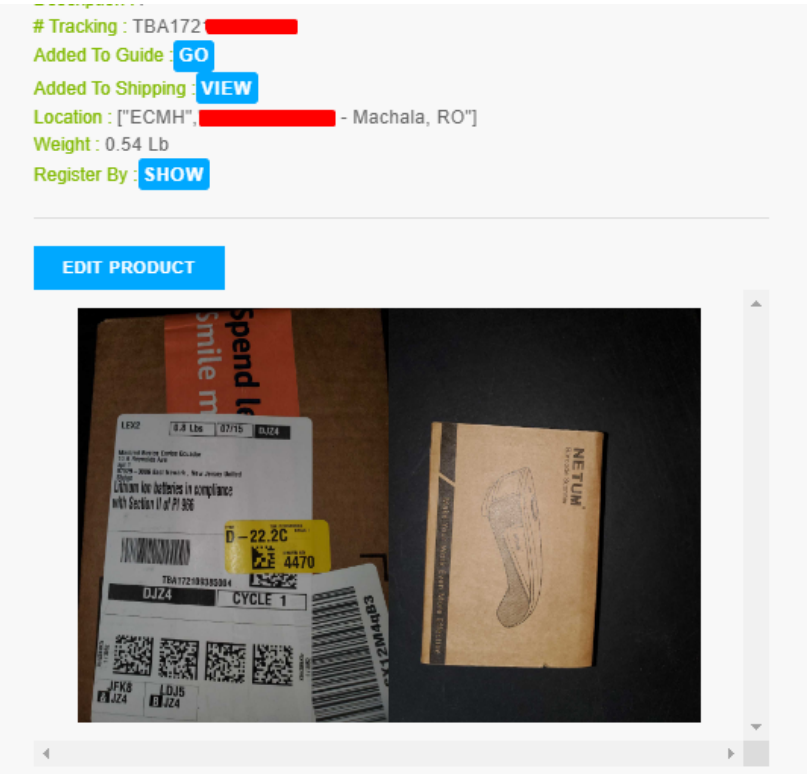


Anexo 43: Evaluación de prototipo – Campo para escanear códigos de productos 2

Este campo está listo para recibir un código insertado por un lector de códigos de barra o para escanear un código QR desde la cámara del dispositivo.

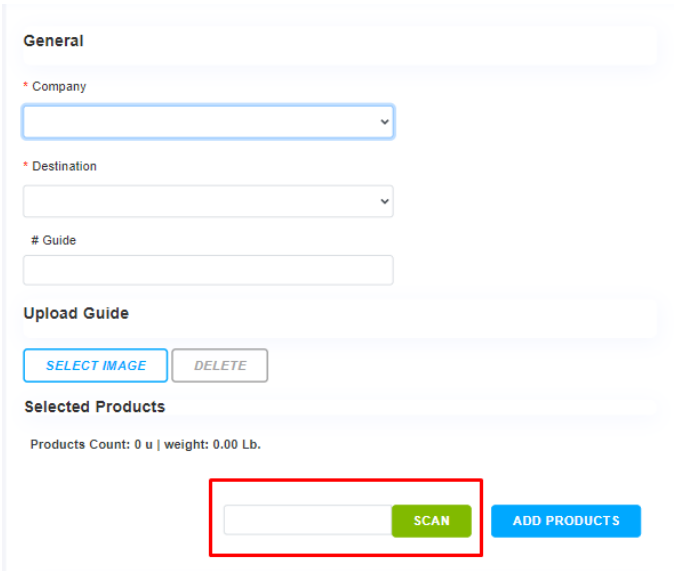
Anexo 44: Evaluación de prototipo – Campo para escanear códigos de productos 3

Una vez leído el código, se abre un modal indicando toda la información referente a dicho producto.



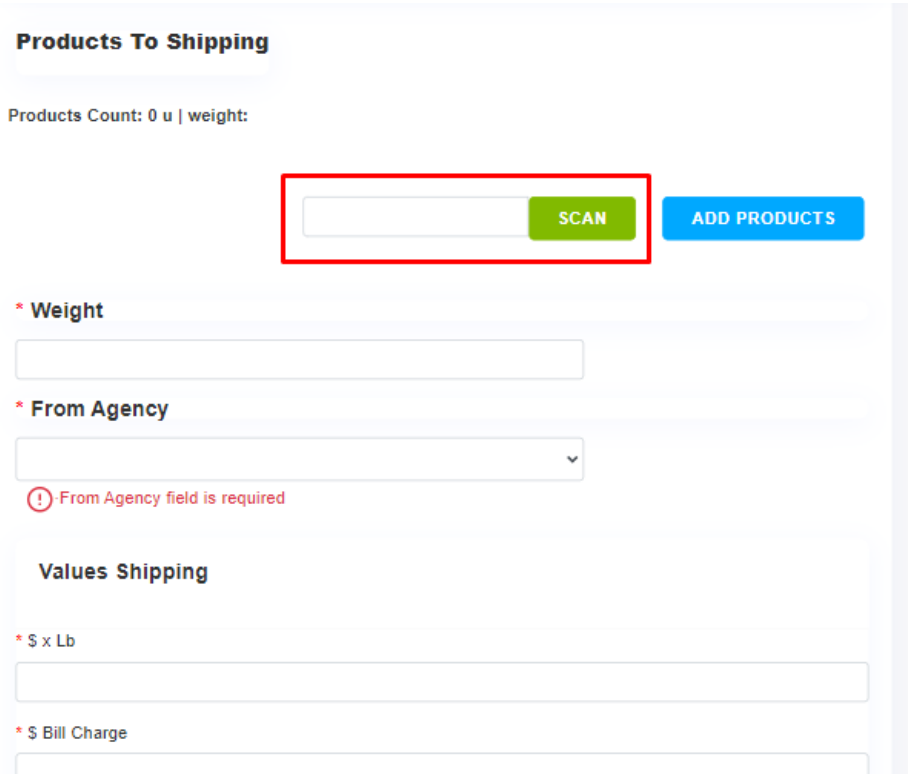
Anexo 45: Evaluación de prototipo – Escaneo de productos para generación de guías aéreas

Para la creación del contenido en una guía aérea se puede escanear el código del producto para agregarlo a la misma.

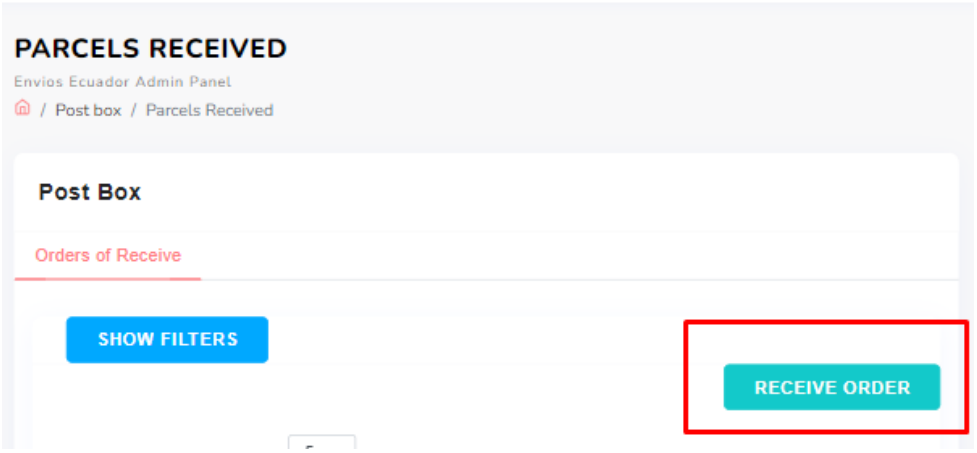


Anexo 46: Evaluación de prototipo – Escaneo de productos para generación de guías de entrega

Se permite agregar los productos en un envió con el escaneo de códigos o seleccionándolos de manera manual.

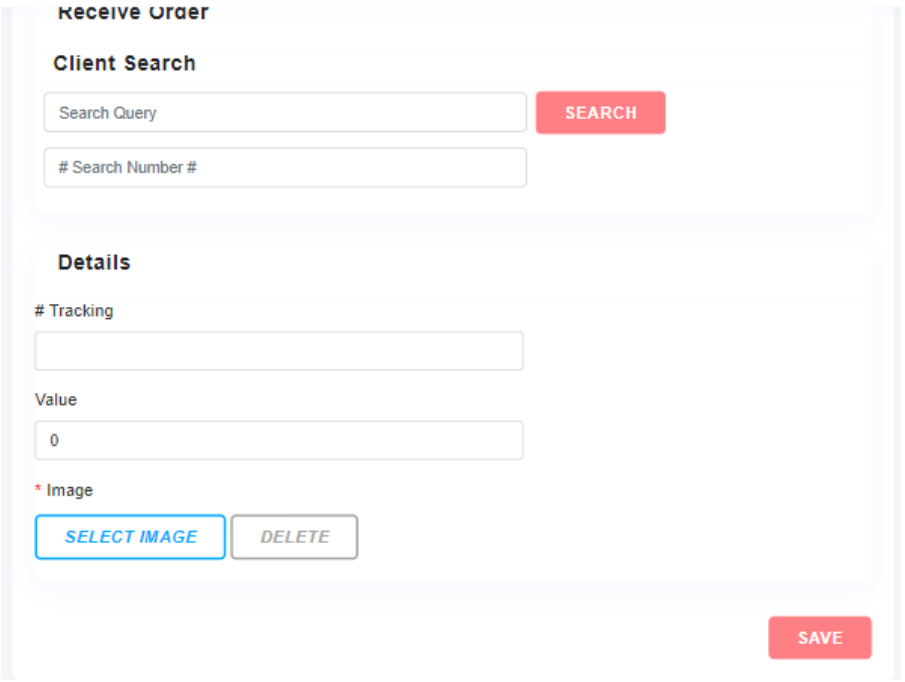


Anexo 47: Evaluación de prototipo – Modulo para recibir encomiendas en el casillero postal y agregar productos 1



Anexo 48: Evaluación de prototipo – Modulo para recibir encomiendas en el casillero postal y agregar productos 2

En este módulo registramos la información referente a la encomienda recibida en el casillero postal y todo su contenido es asignado al cliente propietario del mismo.



Anexo 49: Evaluación de prototipo – Módulo para registrar compras de clientes con sus productos

Se registra una compra realizada por el cliente para tener en cuenta la encomienda que se espera recibir y su contenido.