**UNIVERSIDAD TÉCNICA DE AMBATO**

**Facultad de Ingeniería en Sistemas, Electrónica e Industrial**

**Título:** Informe del Desarrollo del Proyecto

**Carrera:**  Software

**Unidad de Organización Curricular:** Profesional

**Nivel y Paralelo:** 6to Software “A”

**Alumnos Participantes**: Carrasco Paredes Kevin Andres

Chimborazo Guaman William Andres

Quishpe Lopez Luis Alexander

Sailema Gavilanez Ismael Alexander

**Módulo y Docente:** Aplicaciones Distribuidas

Ing. Caiza Caizabuano Jose Ruben

**Fecha:** 05/05/2025



1. **PP**
2. **YY**
   1. **Objetivos**
   2. **General:**

Desarrollar y documentar un sistema distribuido de “Control de Combustible – Empresa XYZ”, basado en una arquitectura de microservicios que permita gestionar el consumo de combustible de maquinaria pesada y liviana, garantizando escalabilidad, alta disponibilidad y confiabilidad del sistema.

**1.2 Específicos:**

* Implementar microservicios independientes para la gestión de choferes, vehículos, rutas, el consumo de combustible y de autenticación, permitiendo una separación clara de responsabilidades y mantenimiento eficiente.
* Utilizar gRPC como protocolo de comunicación entre microservicios para asegurar una transmisión de datos eficiente, rápida y segura.
* Establecer una separación funcional entre la administración de maquinaria liviana y maquinaria pesada, adaptando la lógica del sistema a las particularidades de cada tipo de equipo.
* Diseñar una arquitectura distribuida que garantice la alta disponibilidad, escalabilidad y tolerancia a fallos del sistema.
  1. **Introducción**

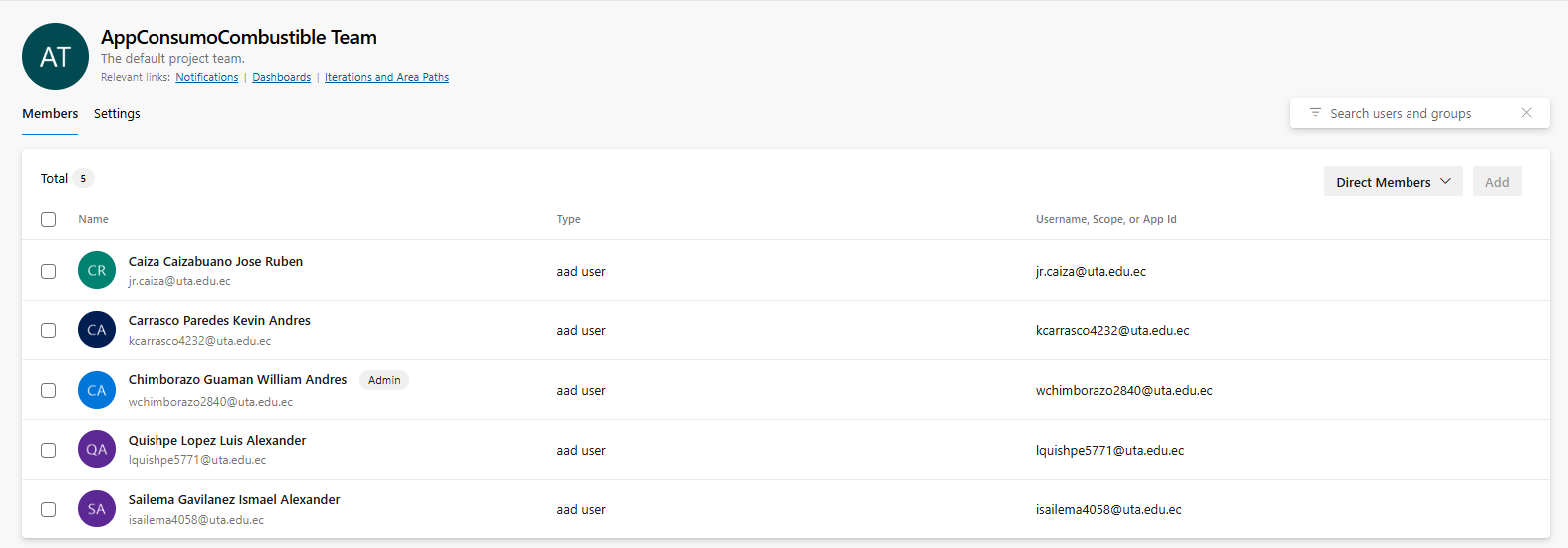
El presente informe documenta el proceso de desarrollo de un sistema distribuido para una empresa XYZ, la cual requiere gestionar y controlar el consumo de combustible tanto de la maquinaria pesada como liviana. La solución se basa en una arquitectura de microservicios separando distintas responsabilidades del sistema garantizando escalabilidad y alta disponibilidad. El sistema utiliza tecnologías modernas como ASP.NET Core, React, gRPC y Docker. El proyecto contiene los cinco microservicios principales —Gestión de choferes —Vehículos —Rutas —Consumo de combustible —Autenticación, esto interactuando con una API Gateway y se conecta con una base de datos MYSQL dockerizada. Este informe detalla la etapa de desarrollo de Login, el módulo de Vehículos, la base de datos, la comunicación con GRCP y el Azure DevOps. Además, incluye diagramas de casos de uso, secuencias y los Mockups de los módulos iniciales a presentar con el objetivo de proporcionar una visión integral del sistema y sus componentes.

* 1. **Desarrollo**

**Información Azure devops**

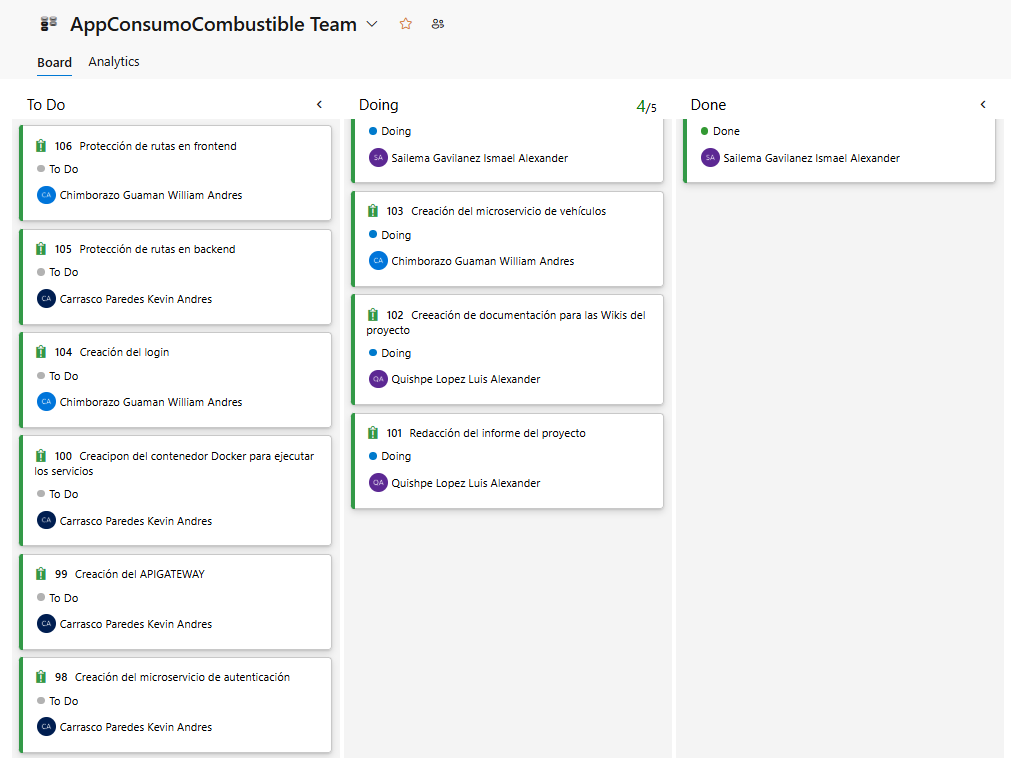
**Información de la organización:**

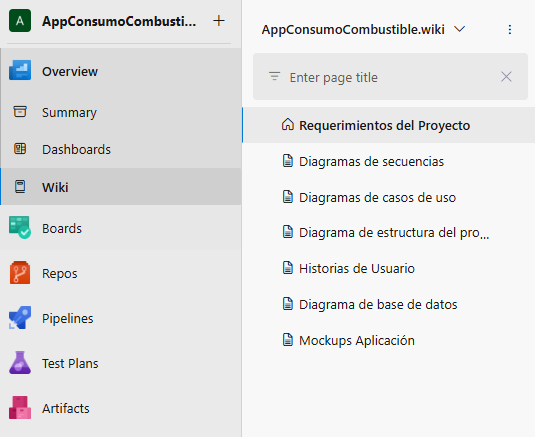
Tutor e integrantes que desarrollan el proyecto.

****

**Planificación de tareas**

Cada integrante del grupo fue designado una tarea dentro del Azure DevOps en la cual ya algunas están totalmente realizadas y otras en proceso de desarrollo. Como se puede observar en el Backlog.

****

**Estructura de wikis:**

**Especificación de requerimientos**

**Requerimientos funcionales**

Los requerimientos funcionales describen el comportamiento y capacidades específicas que el sistema debe proporcionar para satisfacer

las necesidades de los usuarios

**Servicio de Vehículos (VehículosService)**

* Clasificar vehículos como livianos o pesados.
* Gestionar el estado operativo de los vehículos.
* Asociar vehículos con choferes y rutas.
* CRUD de vehículos

**Servicio de Autenticación y Login (AutenticaciónService)**

* Implementar autenticación mediante JWT.
* Definir roles de usuario: Admin, Operador, Supervisor.
* Gestionar autorización por endpoints en gRPC (usando Interceptors o API Gateway si se expone REST).

**Interfaz de Usuario (IU) para Administrador y supervisor**

**Gestión de vehículos**

* Proporcionar una interfaz web en la que el administrador o supervisor puedan realizar la creación, edición y eliminación de vehículos los cuales serán relevantes e importantes en la etapa de designación de vehículos hacia los distintos choferes

**Interfaz de Usuario (IU) para Inicio de Sesión**

**Inicio de sesión**

* Proporcionar una interfaz web (desarrollada en React**)** para que el administrado, supervisor y operador pueda ingresar al sistema dependiendo de los roles asignados.
* Restingar el acceso a usuarios no autorizados o no registrados asegurando que solo accedan a los datos asignados a cada uno.

**Arquitectura de Microservicios**

Servicios Desacoplados:

* Implementar microservicios separados para la administración (gestión de entidades globales) y consultas médicas (gestión de consultas locales).
* Asegurar que cada microservicio pueda operar y escalar de forma independiente.

• Comunicación entre Servicios:

o Permitir la comunicación entre microservicios mediante APIs

RESTful para operaciones síncronas.

o Soportar mensajería asíncrona (por ejemplo, con RabbitMQ o

Kafka) para tareas como sincronización de datos o notificaciones.

**Requerimientos no Funcionales**

Los requerimientos no funcionales especifican los atributos de calidad y

restricciones del sistema, garantizando rendimiento, seguridad, escalabilidad y

mantenibilidad.

**Arquitectura y Escalabilidad**

* Utilizar una arquitectura de microservicios.
* Garantizar la disponibilidad y escalabilidad del sistema mediante una arquitectura distribuida.
* Implementar escalado horizontal por microservicio.
* Permitir orquestación futura con Kubernetes.

**Tecnologías**

* Desarrollar el backend con .NET 8 y gRPC.
* Utilizar contenedores Docker para el despliegue.
* Estructurar los repositorios en carpetas: /src/Services, /Protos, /Shared, /Gateway (opcional), /Infrastructure (docker-compose, DBs).

**Estructura de Microservicios**

* Estructurar cada microservicio en capas:
* Controllers (gRPC).
* Application (lógica de negocio).
* Domain (entidades, interfaces).
* Infrastructure (acceso a datos, gRPC clients).
* Persistence (base de datos).

**Comunicación**

* Usar gRPC como mecanismo de comunicación eficiente entre microservicios.
* Utilizar Protocol Buffers para la definición de mensajes.

**Usabilidad**

* Interfaces Intuitivas
* Proporcionar tooltips o documentación de ayuda para operaciones complejas (por ejemplo, generación de reportes).
* Asegurar que la interfaz de sea fácil de usar, con navegación clara y mínima capacitación requerida.

**Mantenibilidad**

* Estructurar los microservicios para permitir actualizaciones independientes sin afectar otros servicios.
* Usar APIs versionadas para soportar compatibilidad hacia atrás.
* Proporcionar documentación completa de la API usando Swagger para todos los endpoints.
* Incluir documentación técnica detallada en Azure DevOps sobre arquitectura, config

**Historias de Usuario para el sistema de Gestión de Combustible Empresa XYZ**

A continuación, se presentan las historias de usuario basadas en los requerimientos del proyecto descrito en el documento proporcionado. Estas historias están diseñadas para ser incluidas en la wiki de Azure DevOps y

**Historias de Usuario**

* 1. **Servicio de Vehículos (VehículosService)**

Como administrador,

quiero registrar un vehículo clasificándolo como liviano o pesado,

para gestionar el inventario de maquinaria.

**Prioridad:** Alta

**Criterios de aceptación**:

* El sistema permite ingresar detalles del vehículo (Placa, capacidad combustible, consumo combustible por km.).
* El vehículo se clasifica como liviano o pesado usando el campo TipoMaquinaria.
* El vehículo queda registrado, en estado operativo y visible en el sistema.
  1. **Actualizar y eliminar Vehículo**

Como administrador,

quiero actualizar o eliminar un vehículo,

para mantener la base de datos actualizada, corregir errores o dar de baja vehículos que ya no estén en operación.

**Prioridad:** Alta

**Criterios de aceptación**:

* + El sistema debe permitir editar o eliminar la información del vehículo
  + La eliminación del vehículo debe ser de manera lógica en caso de reportes
  1. **Seguridad y Autenticación**

Como usuario (Administrador, Supervisor, Operador),

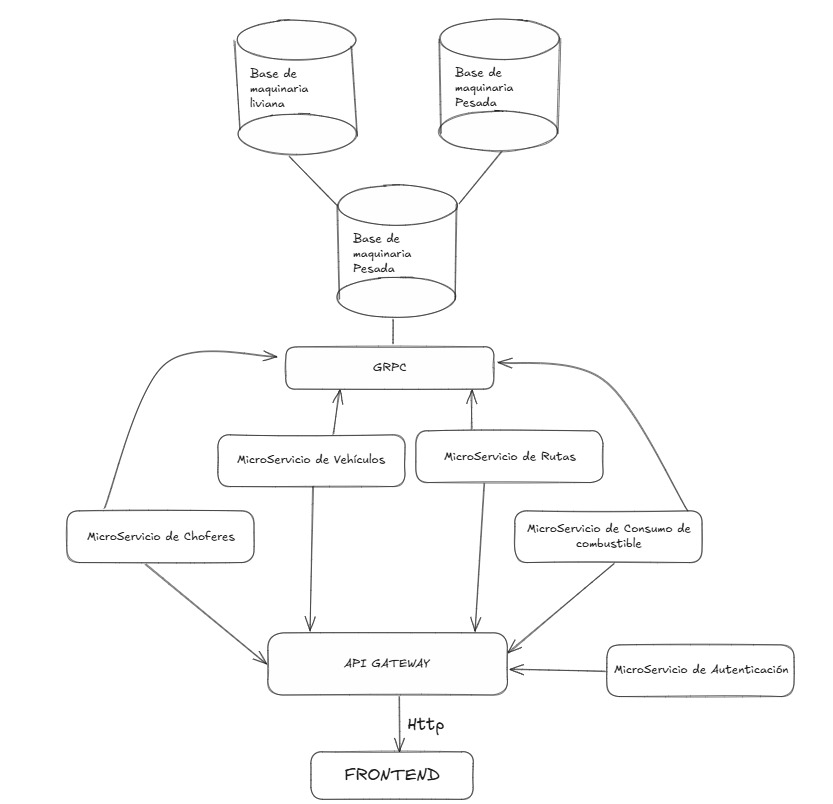
quiero autenticarme en el sistema con mis credenciales,

para acceder a las funcionalidades autorizadas.

**Prioridad:** Alta

**Criterios de aceptación**:

* + El sistema utiliza JWT para autenticar a los usuarios.
  + Los usuarios deben ingresar un nombre de usuario y contraseña válidos.
  + Se deniega el acceso si las credenciales son incorrectas.
  + Dependiendo de los roles asignados este tendrá una interfaz con operaciones diferentes dentro del sistema.

**Diagrama de estructura del proyecto**

El diagrama muestra una arquitectura basada en microservicios, donde

**Frontend (Cliente)**

Se comunica con el API Gateway mediante HTTP/REST (protocolo estándar para navegadores y aplicaciones móviles).

**API Gateway**

Actúa como punto único de entrada y enruta las solicitudes a los microservicioscorrespondientes:

* + Microservicio de Vehículos
  + Microservicio de Rutas
  + Microservicio de Choferes
  + Microservicio de Consumo de Combustible
  + Microservicio de Autenticación

**Comunicación entre Microservicios (gRPC)**

Los microservicios interactúan entre sí usando gRPC (alto rendimiento, bajo latency, ideal para comunicación interna). Ejemplo:

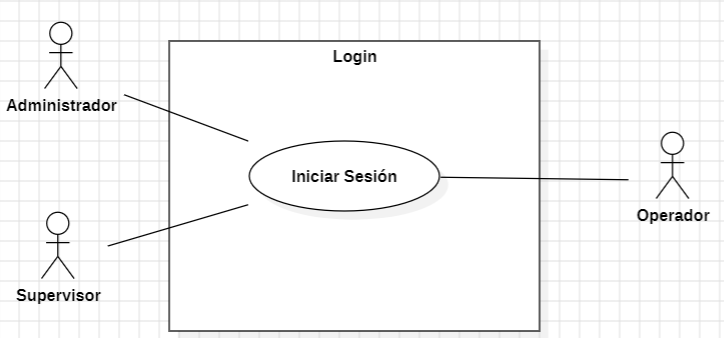
El Microservicio de Rutas podría consultar datos de vehículos al Microservicio de Vehículos mediante gRPC.

**Bases de Datos**

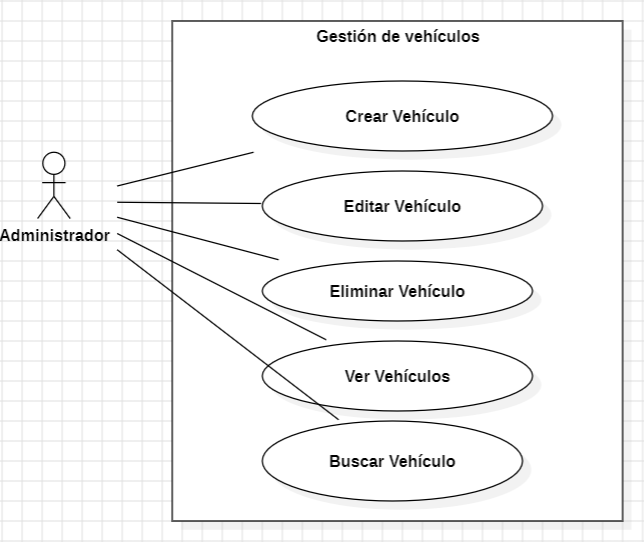
Maquinaria Liviana y Maquinaria Pesada (dos instancias, para segregar datos por tipo de vehículo).

Cada microservicio tiene su propia base de datos (ejemplo: el de Consumo de Combustible manejaría tablas específicas).

**Diagrama de casos de uso**

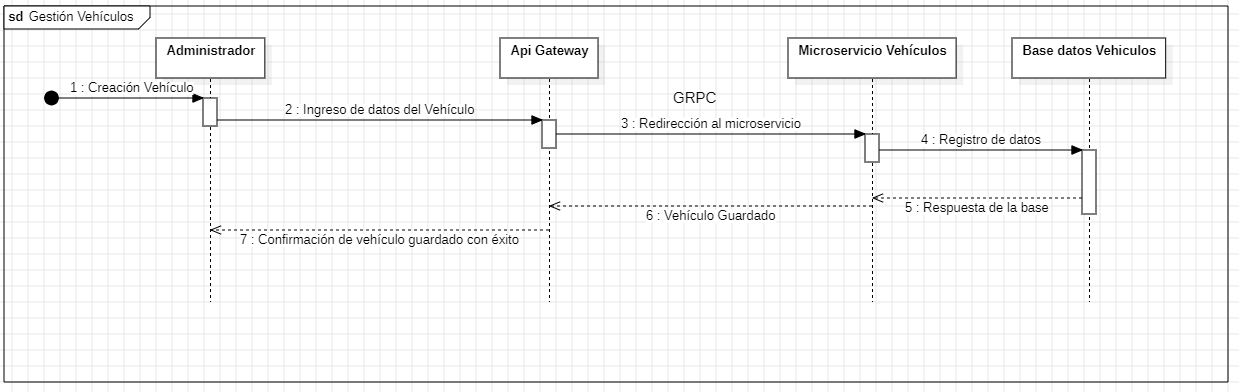
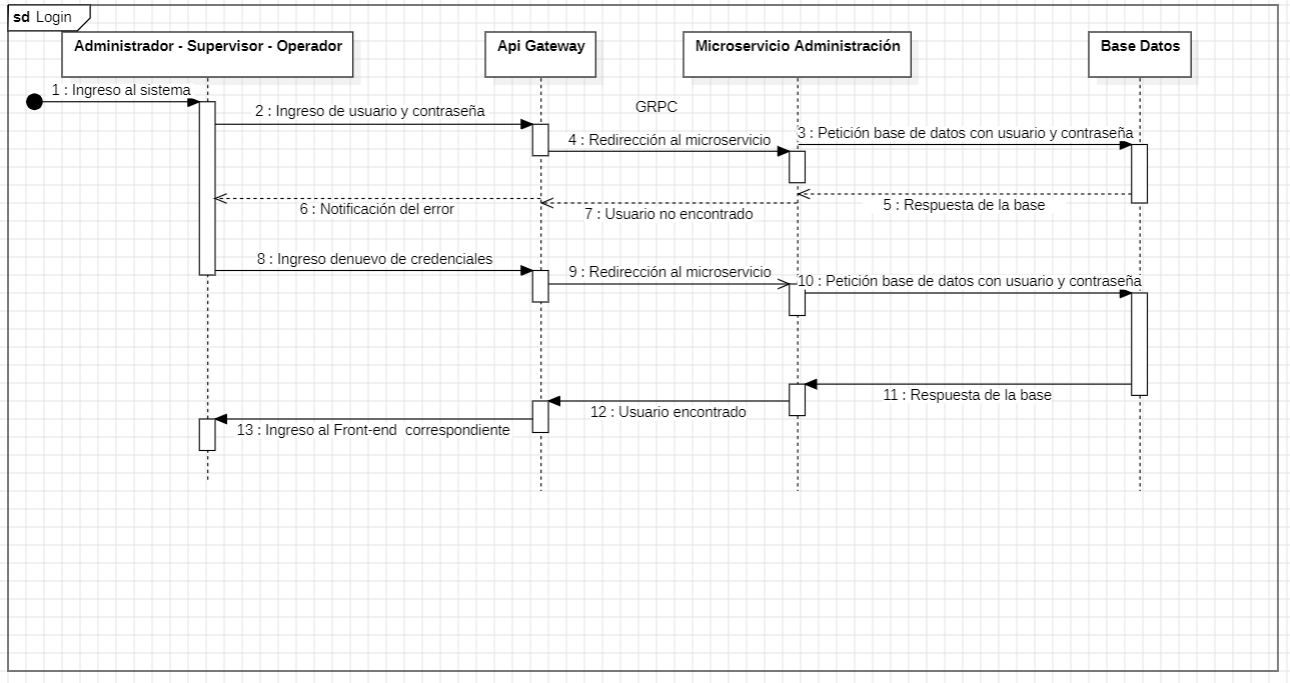
**Login:**

El diagrama de casos de uso del **Login** identifica las interacciones de tres actores, el Administrador, el Superviso y el Operador, con la funcionalidad de inicio de sesión. Funcionalidad principal para el acceso al sistema.

** Gestión de Vehículos:**

El diagrama de casos de uso de **Gestión de Vehículos** identifica la interacción del actor principal el Administrador, con las funcionalidades que tendrá en este módulo en este caso el de crear vehículo, editar vehículo, eliminar vehículo, buscar vehículo y ver vehículos.

**Diagramas de secuencia**

****

Los diagramas de secuencia ilustran las interacciones entre los componentes del

sistema (Usuario, API Gateway, Microservicios y Base de Datos) para dos escenarios principales: el ingreso al sistema (Microservicio de Administración) y la gestión de vehículos (Microservicio de Vehículos), utilizando comunicación gRPC.

* + **Ingreso al Sistema (Microservicio de Administración):**

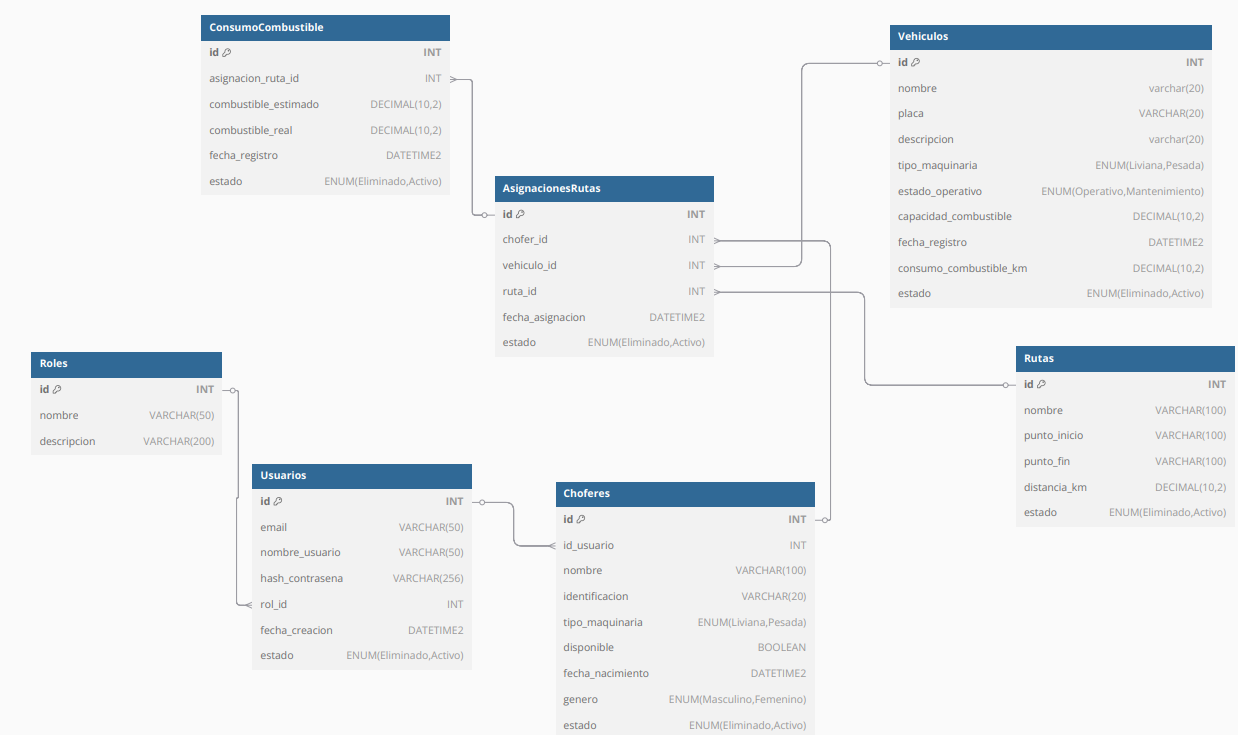
El usuario ingresa sus credenciales (usuario y contraseña) a través del frontend, enviando una solicitud al API Gateway. Este redirige la petición al Microservicio de Administración mediante gRPC. El microservicio busca las credenciales en la base de datos; si el usuario no es encontrado, retorna una respuesta negativa al API Gateway, que notifica al usuario. Si las credenciales son válidas, el microservicio retorna una respuesta positiva, y el API Gateway permite el ingreso del usuario al sistema.

* + **Gestión de Vehículos (Microservicio de Vehículos):**

El administrador, a través del front-end, realiza acciones relacionadas con la gestión de vehículos. Estas incluyen: ingresar datos del vehículo, agregar el vehículo al sistema, ingresar el vehículo y confirmar que se guarde.

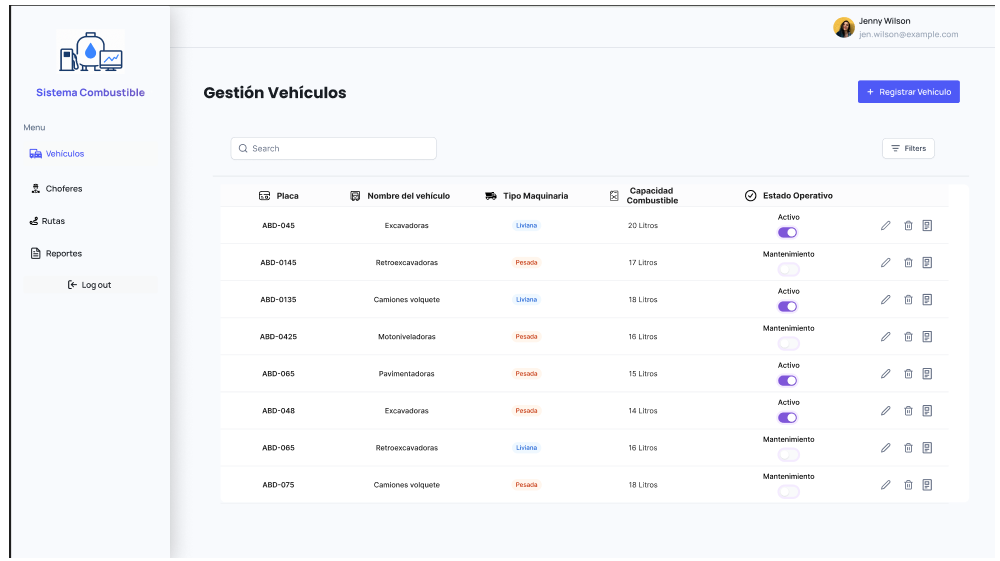
Cada acción se envía al API Gateway, que redirige la petición al Microservicio de Vehículos vía gRPC. El microservicio almacena los datos del vehículo, retornando una respuesta exitosa al API Gateway, que confirma al administrador el éxito de la operación.

**Diagrama de base de datos**

****

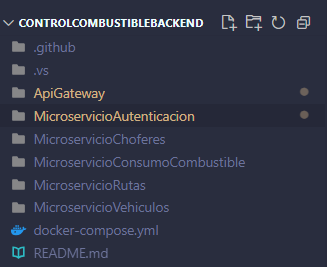
**Mockups**

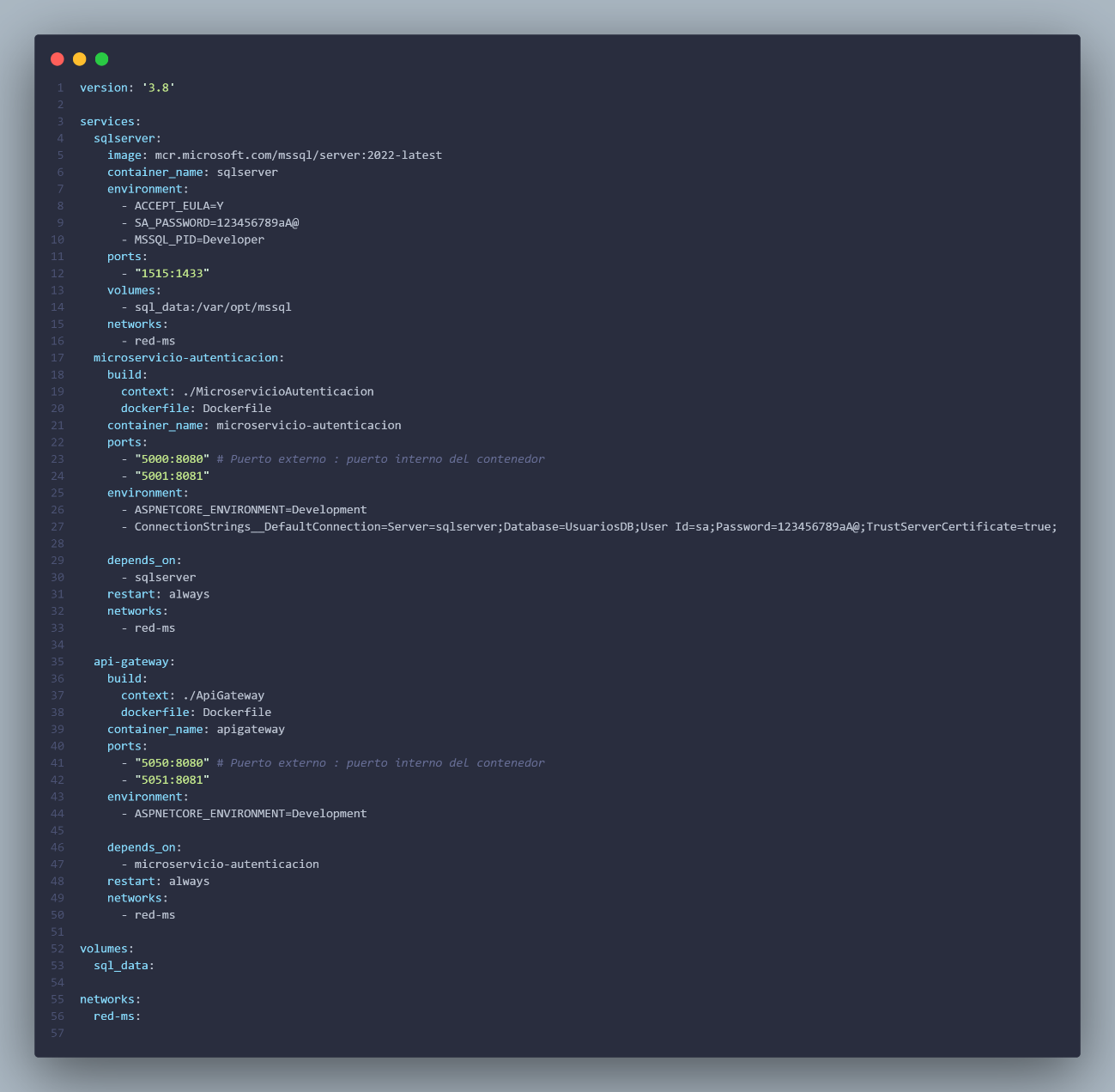
**Mockup de Login**

**Mockup de Gestión de Vehículos**

**Desarrollo del Sistema**

**Back-end**

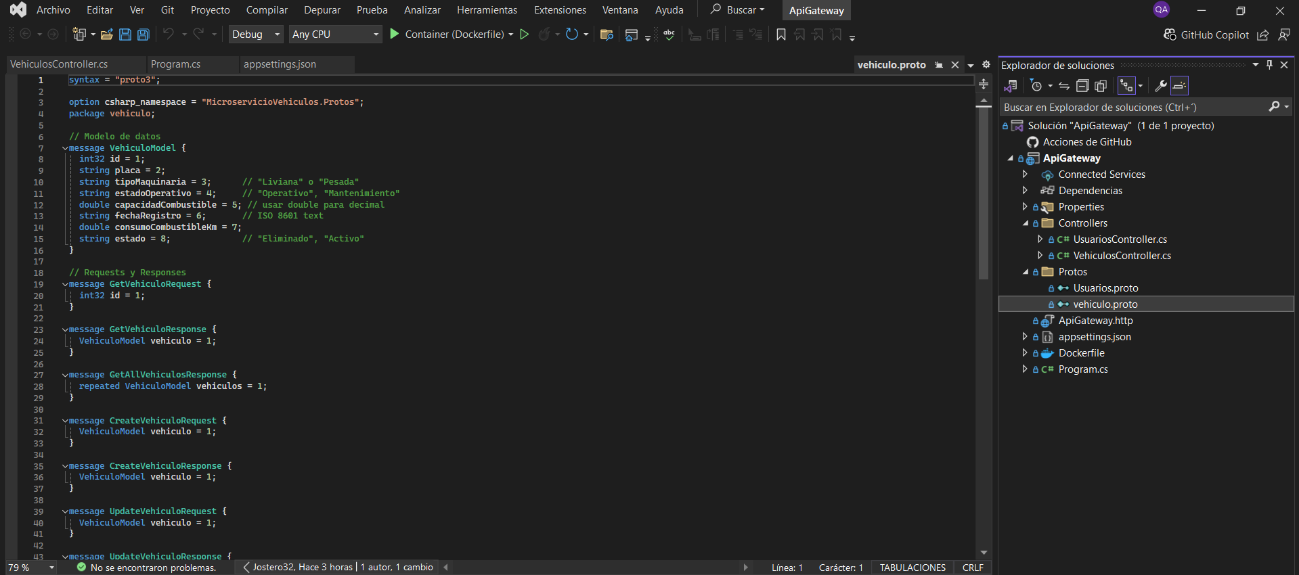
** Estructura del proyecto**

****

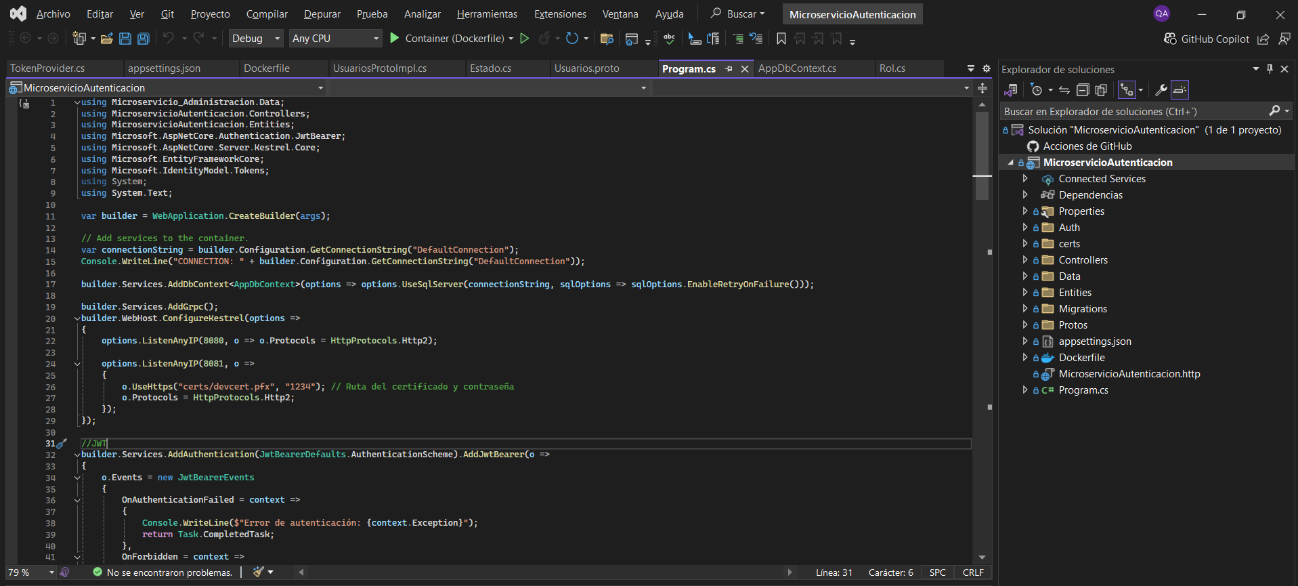
Cada microservicio está organizado en carpetas separadas dentro del proyecto Control\_Combustible\_Backend, lo que facilita una estructura ordenada y modular. Las carpetas incluyen ApiGateway, MicroservicioAutenticacion, MicroservicioChoferes,MicroservicioConsumoCombustible, MicroservicioRutas y MicroservicioVehiculos, cada una dedicada a un microservicio específico. Además, se incluye un archivo docker-compose.yml, esencial para la creación, configuración y distribución de contenedores, permitiendo una implementación eficiente de los microservicios en un entorno de contenedores. También se proporcionan archivos README.md para documentación adicional.

**APIs**

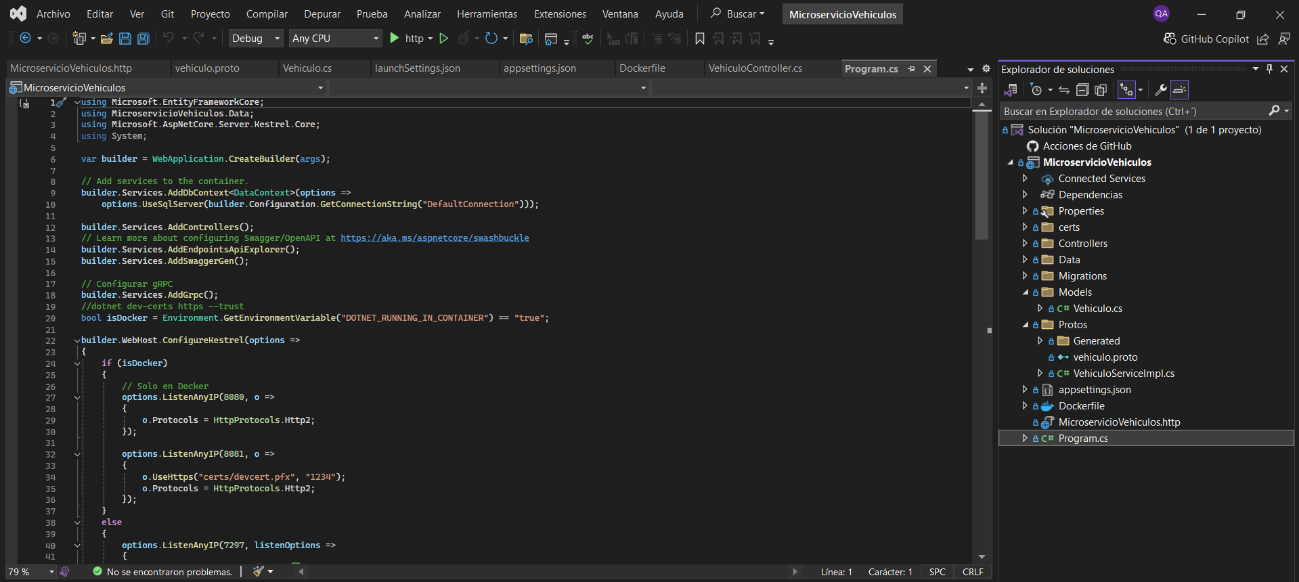
**Api Gateway**

****

El Api Gateway desarrollado en ASP .Net Core, tiene una estructura organizada con carpetas como Connected Services y Dependencies para servicios y referencias externas, Properties con appsettings.json para configuraciones, Controllers con UsuariosController.cs y VehiculosController.cs en C# para manejar usuarios y vehículos, Protos con archivos usuarios.proto y vehiculo.proto para contratos gRPC, un archivo ApiGateway.http para pruebas HTTP, un Dockerfile para contenedores Docker y Program.cs como punto de entrada, reflejando un microservicio diseñado como puerta de enlace que enruta solicitudes a otros microservicios mediante gRPC y HTTP.

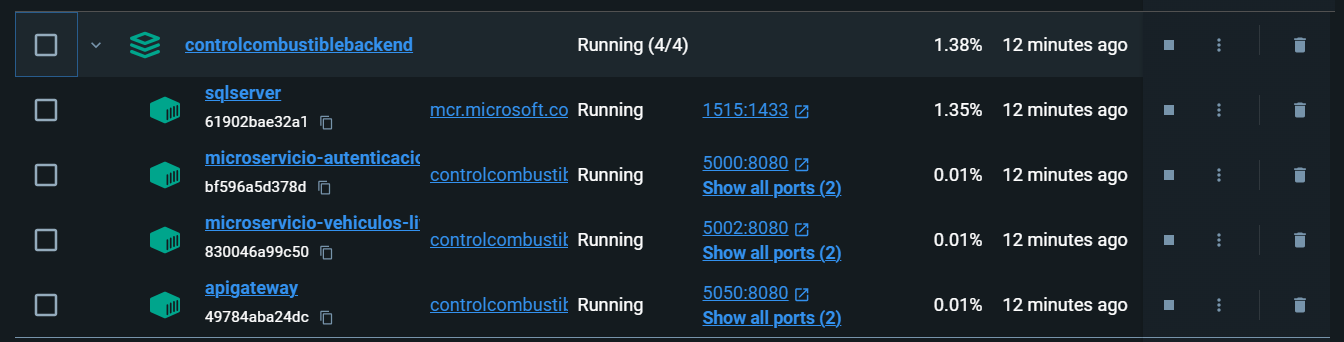
** Api Autenticación y Seguridad**

El Api de Microservicio de Autenticación y Seguridad es más amplio, está estructurada con carpetas como Connected Services y Dependencies para servicios y referencias externas, Properties con appsettings.json para configuraciones, Auth y Certs para autenticación y certificados, Controllers para manejar lógica de endpoints, Data y Entities para gestión de datos y entidades de la base de datos, Migrations para cambios en la base de datos, un Dockerfile para contenedores Docker, MicroservicioAutenticacion.http para pruebas HTTP y Program.cs como punto de entrada, diseñada para gestionar la autenticación en el sistema de microservicios.

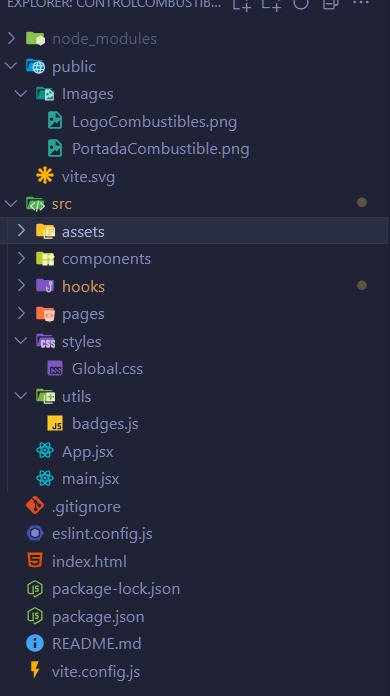
** Api de Vehículos**

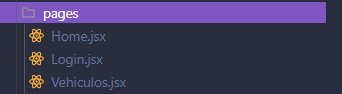
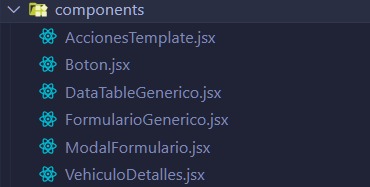
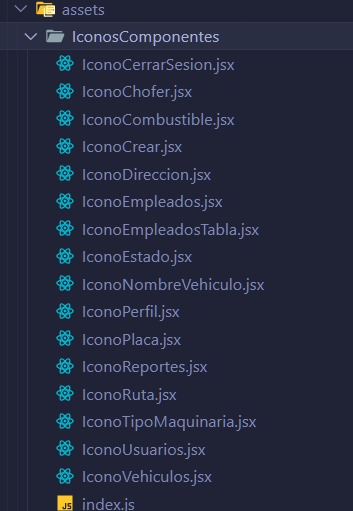
El Api de Microservicio de Vehículos, está organizada con carpetas como Connected Services y Dependencies para servicios y referencias externas, Properties con appsettings.json para configuraciones, Certs para certificados, Controllers y Data para lógica de endpoints y gestión de datos, Migrations para cambios en la base de datos, Models con Vehiculos.cs para definir entidades, Protos con vehiculo.proto para contratos gRPC, Generated para archivos generados automáticamente, un Dockerfile para contenedores Docker, MicroservicioVehiculos.http para pruebas HTTP y Program.cs como punto de entrada, diseñada para gestionar información de vehículos en el sistema de microservicios.

**Contenedores de Microservicios**

****

Los contenedores de sqlserver para la base de datos, microservicios autenticación, microservicio de Vehículos livianos y el api gateway están corriendo en distintos puertos. Cada contenedor tiene un consumo de CPU del 0.01%-1.38% reflejando una configuración distribuida y operativa para la aplicación.

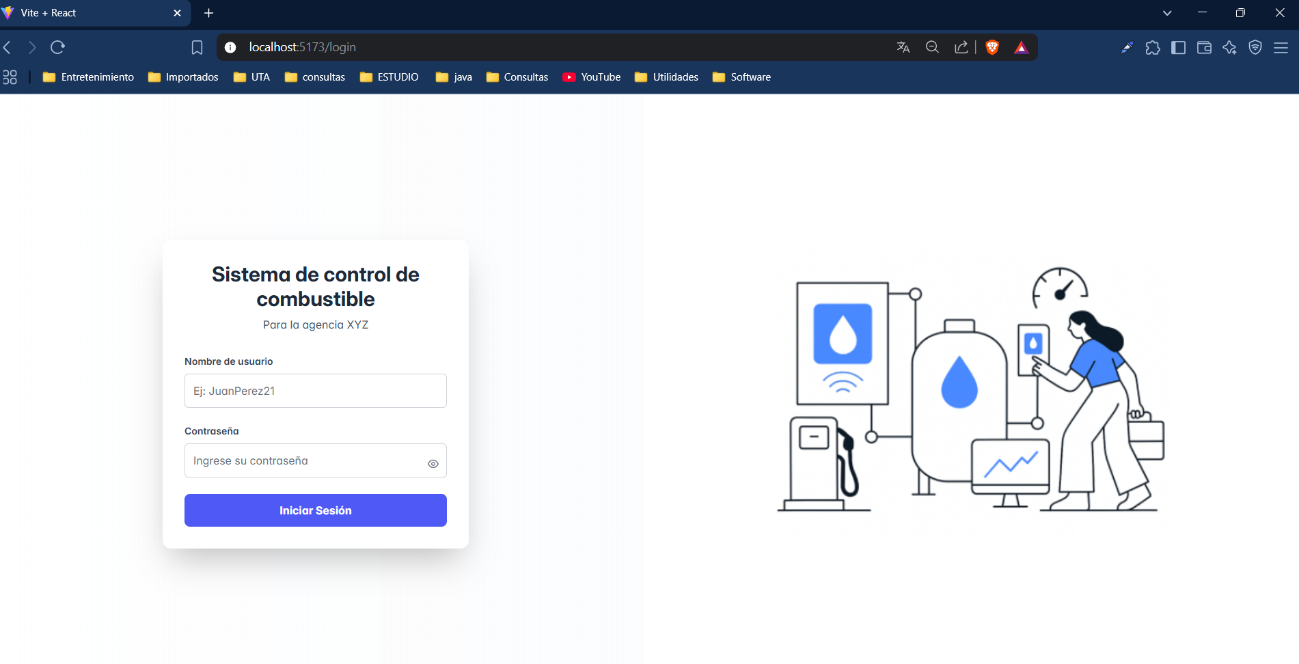
**Front-end**

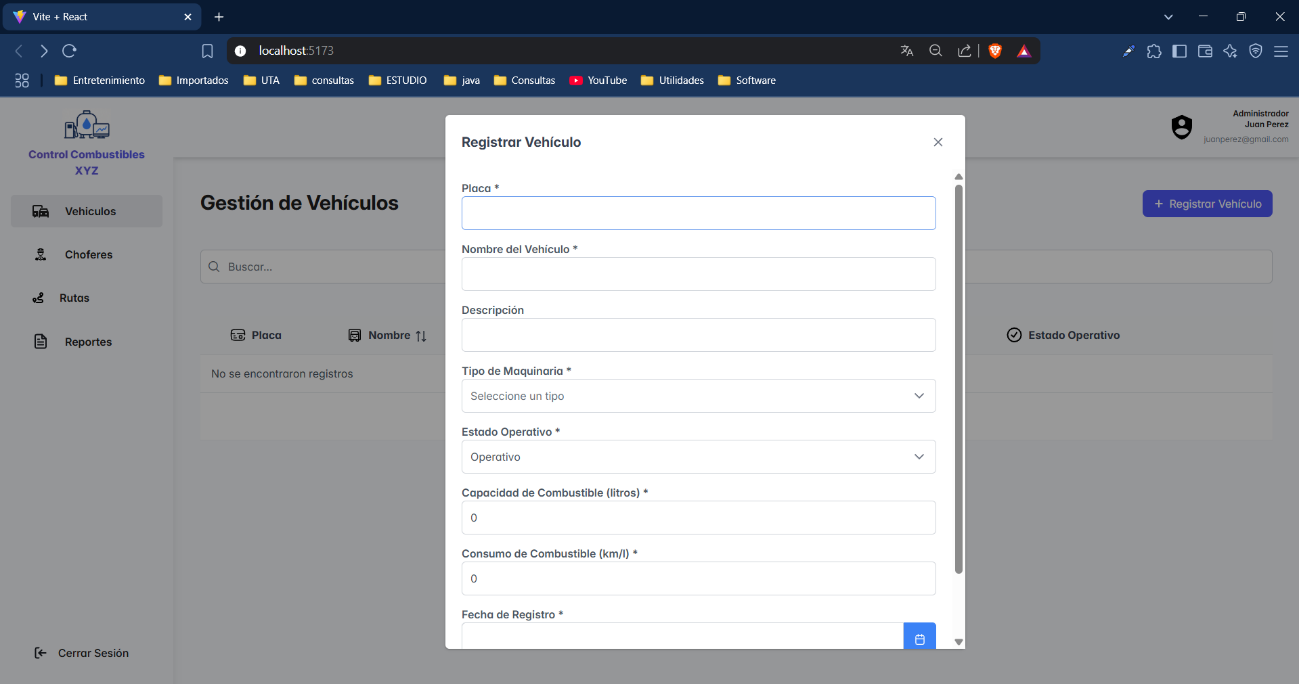


El desarrollo del proyecto en la parte del **front-end** tiene una estructura basada en JavaScript y construido con Vite + React. Contiene distintas carpetas incluyendo archivos clave para el funcionamiento como: la carpeta public contiene recursos estáticos como imágenes (LogoCombustibles.png, PortadaCombustible.png), mientras que la carpeta src alberga el código fuente, incluyendo assets, components contiene imágenes svg pero en forma de componentes, hooks contiene recursos donde se maneja toda la lógica, pages contiene las interfaces principales como el login, home y la gestión de vehículos y donde se llaman los componentes, styles (con Global.css), utils, y archivos clave como badges.js, App.jsx, y main.jsx. El proyecto también incluye archivos de configuración como. gitignore, eslint.config.js, index.html, package-lock.json, package.json, README.md, y vite.config.js, lo que indica un entorno bien organizado con soporte para linting, gestión de dependencias y documentación, diseñado para una aplicación frontend moderna y modular.

**Aplicación Desarrollada y Levantada Localmente**

**Login**



 **Gestión de vehículos**

Finalmente, se exhibe la aplicación totalmente desarrollada, con todas las interfaces solicitadas implementadas de manera funcional, incluyendo una pantalla de **login** para autenticación de usuarios y un módulo de **gestión de vehículos** que permite registrar y administrar datos como placa, nombre, descripción, tipo de maquinaria, estado operativo, capacidad de combustible, consumo por kilómetro y fecha de registro. La aplicación, levantada en un entorno local en la dirección `localhost:5173`, opera de forma exitosa, evidenciando un rendimiento estable y una interacción fluida con las funcionalidades diseñadas, lo que confirma la correcta integración de los componentes frontend y backend del sistema de control de combustible XYZ.

* 1. **Conclusiones**
* En conclusión, el proyecto está utilizando una arquitectura de microservicios bien estructurada, con componentes como ApiGateway, MicroservicioAutenticacion, y MicroservicioVehiculos organizados en carpetas separadas, cada uno con su propio Dockerfile y configuraciones específicas. Esto, junto con el uso de docker-compose.yml, permite una implementación modular y escalable, ideal para manejar diferentes funcionalidades (autenticación, gestión de vehículos, consumo de combustible) de forma independiente.
* La integración de gRPC mediante archivos .proto (como vehiculo.proto y usuarios.proto) y el uso de certificados (Certs) en los microservicios reflejan un enfoque en una comunicación eficiente y segura entre servicios. Además, ApiGateway actúa como un punto central para enrutar solicitudes, optimizando la interacción entre los microservicios y mejorando la seguridad al centralizar el acceso.
* Para el frontend de la aplicación, esta desarrollado con Vite y React, destaca por su estructura modular y organizada, con una clara separación de componentes, estilos globales, y utilidades que facilitan el mantenimiento y la escalabilidad. Las interfaces de usuario, como el login y la gestión de vehículos, ofrecen una experiencia intuitiva y funcional, permitiendo a los usuarios autenticarse y administrar datos de vehículos de manera eficiente