UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Diseño e implementación de un sistema de gestión de servicios para el sector automotriz

José Luis Santillán Calderón

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 7 de diciembre de 2023

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Diseño e implementación de un sistema de gestión de servicios para el sector automotriz

José Luis Santillán Calderón

Nombre del profesor, Título académico **Ricardo Flores, Ph. D.**

Quito, 7 de diciembre de 2023

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: José Luis Santillán Calderón

Código: 00211496

Cédula de identidad: 1725022634

Lugar y fecha: Quito, 7 de diciembre de 2023

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

En este trabajo se resalta la importancia de la transformación digital, específicamente en el sector automotriz. Se propone desarrollar una aplicación móvil multiplataforma utilizando Ionic Framework para los clientes de All Tires que mejore la experiencia del cliente en diferentes procesos. Los objetivos principales de esta aplicación es agilizar la revisión del estado actual de los mantenimientos de sus vehículos, mantener un historial para un mejor control vehicular y desarrollar un módulo de agendamiento de citas, todo esto con la finalidad de mejorar la gestión interna de la empresa. El proyecto busca beneficiar a la empresa y a sus clientes al brindar un servicio personalizado, reducir tiempos de espera y abrazar la transformación tecnológica en el sector automotriz.

Palabras clave: Transformación digital, Sector automotriz, Control vehicular, Mantenimiento vehicular, Innovación tecnológica.

ABSTRACT

This study underscores the significance of digital transformation, specifically within the automotive industry. It proposes the development of a cross-platform mobile application using the Ionic Framework for All Tires' customers, aimed at enhancing the customer experience across various processes. The primary objectives of this application are to streamline the status of current vehicle maintenance, maintain a history for better vehicle management, and create an appointment scheduling module, all with the aim of improving the company's internal operations. The project seeks to benefit both the company and its customers by providing a personalized service, reducing wait times, and embracing technological transformation within the automotive sector.

Key words: Digital transformation, Automotive sector, Vehicle control, Vehicle maintenance, Technological innovation.

Tabla de contenido

Tabla de imágenes

CAPÍTULO I – INTRODUCCIÓN

Antecedentes

La rápida evolución de las tecnologías ha desencadenado una serie de transformaciones significativas en las últimas décadas. La industria automotriz ha sido testigo de una transformación tecnológica continua, ya sea en maquinaria adquirida para brindar servicios o en software para la gestión interna. Con el desarrollo tecnológico, los vehículos han evolucionado de simples medios de movilización a complejas plataformas digitales.

En el mercado ecuatoriano, las empresas buscan satisfacer las necesidades de los consumidores modernos por lo que buscan soluciones que estén alineadas con el desarrollo tecnológico. Esta evolución se encuentra en proceso de desarrollo y solo los grandes concesionarios de vehículos o talleres con un capital económico alto pueden adquirir software que permite monitorear y gestionar de manera más eficiente el mantenimiento de los automotores.

All Tires ha reconocido la importancia de permanecer en la vanguardia de la tecnología para ofrecer una mejor experiencia a sus clientes, y de esta manera generar mayor fidelidad con su clientela y atraer a potenciales nuevos clientes.

Descripción de la empresa

All Tires – Tecnicentro Automotriz es una empresa que se encarga de ofrecer a sus clientes servicios de mecánica ligera en el norte de Quito. El negocio inició en el año 2003, y por 20 años ha crecido paulatinamente para ofrecer más servicios y productos a sus clientes, siempre con sus valores de empresa claros, los cuales son brindar un servicio de calidad y personalizado.

Misión.

El propósito fundamental y la razón de ser de *All Tires* como negocio es el siguiente: “Nuestra misión es proporcionar a nuestros clientes productos de la más alta calidad y tecnología de vanguardia. Buscamos promover el crecimiento a largo plazo y el bienestar económico tanto de nuestros inversionistas como de nuestros empleados”. Es importante destacar la misión, pues esto ayuda a generar una identidad corporativa que se puede ver reflejada en los productos asociados a la empresa.

Visión.

La declaración a largo plazo del tecnicentro automotriz se define de la siguiente manera: "Nos esforzamos por ser una empresa innovadora líder en nuestra zona de influencia, ampliamente reconocida y respetada por inversionistas, clientes y competidores. Aspiramos a ser un referente para otras empresas, inspirando la excelencia en todos los aspectos de nuestro negocio". Esta visión establece una base sólida para la planificación estratégica y el crecimiento de la empresa.

Contexto actual de la empresa

Como se mencionó con anterioridad, *All Tires* desea permanecer a la vanguardia de la tecnología y brindar un servicio personalizado y de calidad. Es por ello que actualmente el tecnicentro cuenta con un sistema de gestión de órdenes de trabajo denominado *All Tires Support*. Las órdenes de trabajos llevan un detalle del mantenimiento que se debe realizar a un vehículo, junto con información importante del propietario. En este sistema, se lleva un control de los

vehículos de los clientes, de esta forma en caso de que se necesite se puede revisar cuales han sido los servicios realizados en mantenimientos previos.

All Tires Support se conecta con una base de datos y almacena varias entidades. Sin embargo, las principales entidades son vehículos, clientes, y órdenes de trabajo. El software está implementado en un servidor local y está realizado con un enfoque de microservicios en *ASP .NET Core* para obtener independencia, modularidad y escalabilidad.

Justificación del proyecto

El software implementado actualmente es un paso para la transformación digital que está llevando a cabo *All Tires*, pues ha dejado de usar registros físicos para guardar un control de los mantenimientos de los vehículos de los clientes, y ahora son digitales. Sin embargo, este software es puramente de control interno. Es decir, que la administradora es la única persona capaz de acceder a dicha información.

En un caso hipotético en el que los clientes deseen conocer cuáles fueron los servicios realizados previamente en su vehículo, deberán dirigirse al tecnicentro, o contactar a la administradora para que verifique la información y se la transmita al propietario. Esto es una limitación en el sistema actual y podría resolverse con otro software complementario.

De esta manera, se propone la implementación de una aplicación móvil que este destinada a los clientes de *All Tires - Tecnicentro Automotriz*, que facilite y optimice procesos como estado del mantenimiento del vehículo, historial de servicios y opciones para agendar futuros mantenimientos. Este software beneficia al tecnicentro al permitirle proporcionar a sus clientes un servicio personalizado, se trata de un producto que añade un valor distintivo a la empresa y la diferencia de la competencia.

La optimización de la gestión interna es otra razón de peso para el desarrollo de esta aplicación, pues permitiría una administración eficiente. Comúnmente cuando se utiliza un gestor de citas, los tiempos de espera se reducen, y se gestionan de mejor manera los recursos. En ese sentido, el proyecto propone una solución original e innovadora en el sector automotriz ecuatoriano.

Objetivos

Generales.

- Modernizar la experiencia del cliente proporcionando un acceso conveniente y sencillo al historial de mantenimientos de sus vehículos.
- Implementar una solución tecnológica que optimice el agendamiento de mantenimientos vehiculares.
- Contribuir con la diferenciación y el servicio personalizado que busca brindar *All Tires* a sus clientes al ofrecer un servicio innovador y a la vanguardia de la tecnología.

Específicos.

- Desarrollar una aplicación intuitiva y amigable para los distintos usuarios, independientemente de su conocimiento tecnológico, y que sea acorde a la identidad visual de *All Tires*.
- Desarrollar un módulo de agendamiento de citas que permita a los usuarios seleccionar fechas y horarios disponibles de manera efectiva.
- Integrar de manera eficiente la base de datos y el sistema de gestión interna actual con la aplicación móvil.

- Diseñar e implementar una solución tecnológica escalable capaz de manejar un incremento en la cantidad de usuarios y actividades, sin que esto afecte el rendimiento y funcionamiento de la aplicación.
- Garantizar la seguridad de la información de los clientes mediante métodos de cifrado de datos y autenticación de usuarios.

CAPÍTULO II – ESTADO DEL ARTE

El área de estudio abordada en este proyecto se enfoca en el desarrollo de aplicaciones móviles y su aplicación en la industria, específicamente en el sector automotriz. El desarrollo de aplicaciones móviles implica la creación de software con interfaces intuitivas, programación eficiente, y adaptación a distintos dispositivos.

Adicionalmente, se ponen en práctica conceptos abstractos del área de la computación como lo son eficiencia, modularidad y escalabilidad. La integración con un sistema ya implementado supone un reto, pues hay que asegurar compatibilidad y una sinergia con el diseño de la arquitectura del software.

En relación con las mejores prácticas en el desarrollo de aplicaciones móviles, Pujari, Patil y Sutar (2020) exponen las mejores prácticas al momento de realizar una aplicación móvil, y ofrecen un análisis de las estrategias más importantes y efectivas en esta área, las cuales son tomadas en cuenta en la ejecución de este proyecto.

En el apartado de diseño y usabilidad, Kuusinen y Mikkonen (2014) señalan que los patrones de diseño de experiencia de usuario desempeñan un papel fundamental en la mejora de la usabilidad y la satisfacción del usuario en aplicaciones móviles. Se resalta la importancia de la

incorporación de patrones de diseño familiares que lleva a una interacción más efectiva y a una experiencia de usuario más positiva.

Análisis de la competencia

Tras una investigación de mercado en la tienda de aplicaciones de *Android Play Store*, se ha identificado que no existe una aplicación móvil de este estilo orientado a talleres. Si bien existen sistemas similares para ordenes de trabajo, o control de inventario y facturación, no se ha encontrado una aplicación orientada para los clientes, por lo que es una solución innovadora que ayuda a *All Tires* a diferenciarse de la competencia.

Existen aplicaciones en las que el usuario mismo pueda llevar un control de su vehículo, pero este no está vinculado a un taller como tal. *Drivvo* o *Simply Auto* son ejemplos de aplicaciones de gestión de vehículos en las que el usuario debe llevar su propio control, y en ciertos casos puede ser complejo. Es una solución válida pero una aplicación que gestionen los propios técnicos y administradora del taller proporciona una experiencia más completa, pues garantiza que tanto los clientes como el taller se beneficien de un servicio personalizado.

Metodología

Al realizar un proyecto que debe consumir datos de otro microservicio previamente realizado, es fundamental entender la estructura y la lógica de aquel software para lograr una integración eficiente y adecuada. Adicionalmente, se define junto a *All Tires* el comportamiento básico de la aplicación que se evidencia en la sección de diagrama de flujo.

Los diseños propuestos y la interfaz de usuario de la aplicación deben estar alineados con la identidad visual de *All Tires,* por lo que se usarán herramientas de diseño gráfico para lograr un

resultado profesional y personalizado. En este sentido, se usan herramientas como *Adobe Illustrator* y *Figma* para presentar bosquejos y diseños preliminares.

Para lograr una solución modular y escalable, se decidió realizar un proyecto con arquitectura de dos capas. La capa del *back-end* que se encarga de la lógica, la gestión de datos, la seguridad y manejo de errores, y la capa de *front-end* que se centra en la interfaz de usuario y la experiencia del cliente.

El desarrollo simultáneo de ambas capas es esencial para sincronizar el avance de la lógica con el avance de la interfaz de usuario. Adicionalmente, esta estrategia de desarrollo permite la detección de errores o detección de requerimientos adicionales que puedan no haber sido definidos o contemplados en la planificación inicial del proyecto.

Justificación de elección de tecnología

Para el desarrollo de este proyecto se decidió elegir tecnologías actuales, pues es clave considerar factores como compatibilidad, escalabilidad y accesibilidad. A continuación, se detallan los principales motivos por lo que de seleccionó *Ionic Framework* para el desarrollo del *front-end, ASP .NET Core* para el desarrollo del *back-end*, y un servidor flexible de *Azure Database para Postgre SQL* como solución de base de datos. Estas tecnologías aseguran una solución integral para cumplir con los objetivos de la propuesta.

Ionic Framework.

El uso de *Ionic Framework* se justifica por su capacidad probada para el desarrollo de aplicaciones móviles multiplataforma, que asegura accesibilidad para un amplio público. Ionic cuenta con componentes con un diseño elegante y atractivo que facilitan el desarrollo. Cuenta con una línea de comando (CLI) que permite empaquetar el código para generar proyecto *IOS* y *Android* con un mismo código fuente (Román, 2015). En definitiva, simplifica el desarrollo en lo que respecta al *front-end* de una aplicación de estas características.

ASP .NET Core.

Para el *back-end* se seleccionó esta tecnología debido a su robustez, seguridad y capacidad transaccional. *ASP .NET Core* es un marco de alto rendimiento, de código abierto, que facilita la creación de *API's* eficientes para una comunicación con el *front-end*. Es de suma importancia garantizar una respuesta fluida para la experiencia de usuario, así como un software que sea estable, moderno y capaz de compilar en cualquier plataforma (Lock, 2023). Es una opción ideal para el desarrollo de *All Tires Connect*, pues asegura una solución sólida, es decir una solución compatible, eficiente, multiplataforma y de alto rendimiento.

Servidor flexible de Azure Database para Postgre SQL.

Al momento de desarrollar una aplicación, es imprescindible realizar una investigación previa sobre donde alojar la base de datos. En este contexto, el servidor flexible de *Microsoft Azure* es una opción viable. Como asegura la empresa Microsoft (2023), está diseñada para brindar una alta disponibilidad, seguridad de la información, un

monitoreo completo y flexibilidad al momento de configurar. Se elige esta tecnología ya que brinda confianza y un soporte continuo, además de un panel de configuración y administración completo.

CAPÍTULO III – DESARROLLO DEL PROTOTIPO

Arquitectura de la aplicación

Definir una estructura depende del enfoque de la aplicación, la estructura sobre la que se desarrolla es esencial para saber cómo se administra la seguridad, el mantenimiento del sistema y escalabilidad en futuras actualizaciones. En el presente capitulo, se expone la arquitectura de la aplicación *All Tires Connect* lo que facilita la apreciación y compresión de la robustez y escalabilidad de la solución.

Como arquitectura de toda la solución, se implementa una Arquitectura *REST* que proporciona comunicaciones cliente-servidor para aplicaciones web a través de protocolos *HTTP*. Esencialmente, se siguen tres principios de diseño para que se considere una arquitectura *REST*. Según Prayogi, Niswar, Indrabayu, y Rijal (2020) es direccionamiento, interfaz uniforme y sin estado. De manera concisa, en esta arquitectura, el *front-end* envía peticiones al *back-end*, que a su vez procesa y devuelve una respuesta que el *front-end* presenta al usuario.

Este tipo de arquitectura permite compartir la carga de trabajo para una mayor eficiencia. Como se mencionó en el capítulo anterior, se consigue de esta manera modularidad y escalabilidad. Adicionalmente, se usan otros patrones de diseño y buenas prácticas que aportan en el desarrollo como lo son inyección de dependencias, separación de responsabilidades y filtros de acción. Son

conceptos que se usan en los diferentes módulos de la aplicación, y en su respectiva sección se explica de manera más detallada cómo se implementan.

El resultado simplificado de la arquitectura de la aplicación se puede apreciar en la Imagen No. 1 en la que se observa como interactúan los principales módulos de la solución entre sí.

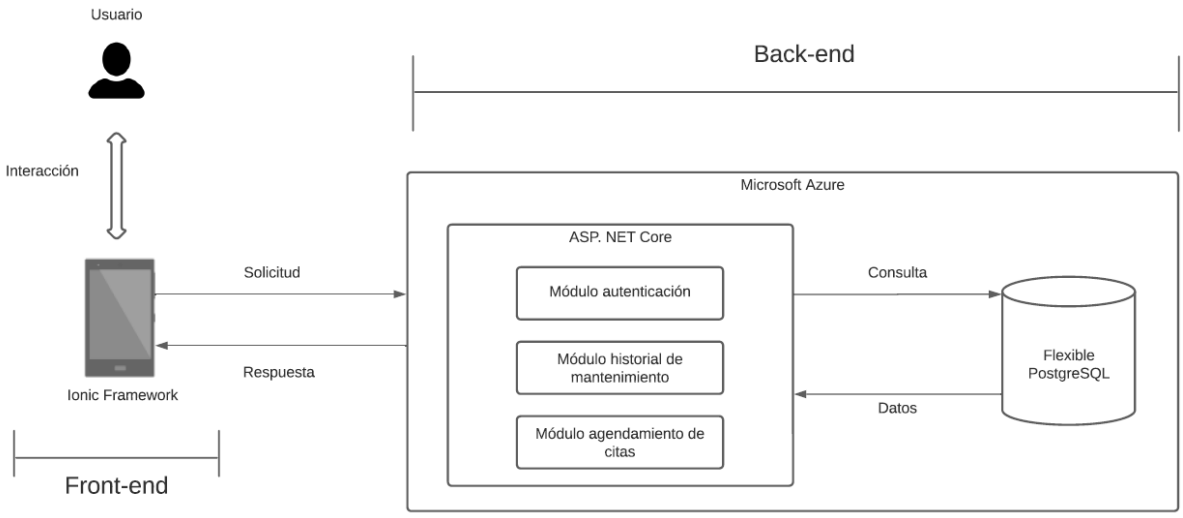


Imagen No. 1: Arquitectura de aplicación All Tires Connect

Es importante mencionar que dentro de *Ionic Framework* se adopta la arquitectura Modelo-Vista-Controlador *(MVC)*. Esta arquitectura es un patrón de diseño que separa la aplicación en tres componentes: el Modelo que se encarga de la lógica de negocio y datos; la Vista que es la interfaz de usuario; y el Controlador, que actúa como intermediario entre Modelo y Vista. Este diseño se ajustó para consumir de la *REST API* desarrollada en el *back-end* con *ASP .Net Core*, que a su vez tiene una arquitectura de tres capas para gestionar los procesos y peticiones.

Diagrama de flujo.

El diagrama de flujo ilustra la secuencia lógica de las operaciones que el usuario puede llevar a cabo en la aplicación y se expone en el Anexo No. 1. Al comenzar, el usuario

se enfrenta a la necesidad de iniciar sesión. Si aún no cuenta con credenciales, el sistema lo guía hacia un proceso de registro, tras el cual puede acceder sin inconvenientes. Una vez autenticado, podrá navegar por la aplicación según sus necesidades.

En la primera pantalla, el usuario tiene la capacidad de seleccionar su vehículo mediante la placa, permitiéndole visualizar las órdenes de mantenimiento que se encuentren en curso o que estén listas para entregar. La segunda pantalla ofrece un historial de todas las órdenes de trabajo relacionadas con su vehículo, proporcionando un registro detallado de los servicios e inventario utilizado.

Finalmente, en la tercera pantalla, el usuario puede programar una cita para futuros mantenimientos, eligiendo una fecha, un horario y especificando el motivo principal de la visita. Evidentemente, es capaz de navegar entre pantallas y de cerrar sesión cuando el usuario lo crea necesario. Las diferentes pantallas se pueden visualizar en el Anexo No. 5.

UML.

El diagrama UML expone la arquitectura de las tablas que se usan en la solución *All Tires Support*, y en la solución propuesta en este proyecto *All Tires Connect*. Entre las tablas más esenciales se encuentran *client, login, vehicle, reception, service, inventory* y *appointment* ya que tienen los principales datos que se mostraran en el *front-end*. En el Anexo No. 2 se observa el diagrama UML completo.

Un aspecto que es importante mencionar de este diagrama UML es que por pedido de la empresa *All Tires*, es necesario que el cliente exista en *All Tires Support* para que se pueda registrar en *All Tires Connect*. De esta forma, se creó la entidad *login* que tiene una relación uno a uno con la entidad *client*, y que llevará la información crucial para el módulo de autenticación.

Back-end

Configuración de base de datos.

Como se mencionó en el capítulo II, se usó *Microsoft Azure* para alojar la base de datos. Y se debió realizar una investigación para una configuración adecuada. Se creó un servidor denominado *alltiresdb* en la región *East US* para obtener una baja latencia en la conexión. Adicionalmente, se la configuró con la versión de *PostgreSQL 15*.

En la Imagen No. 2 se puede observar un resumen de la configuración realizada y descrita.

Imagen No. 2: Configuración servidor flexible en Microsoft Azure

A nivel de desarrollo se configuró *pgAdmin* para acceder a la base de datos y a sus registros. Aquí se configuró el *hostname*, el puerto y la contraseña como se observa en la Imagen No. 3.

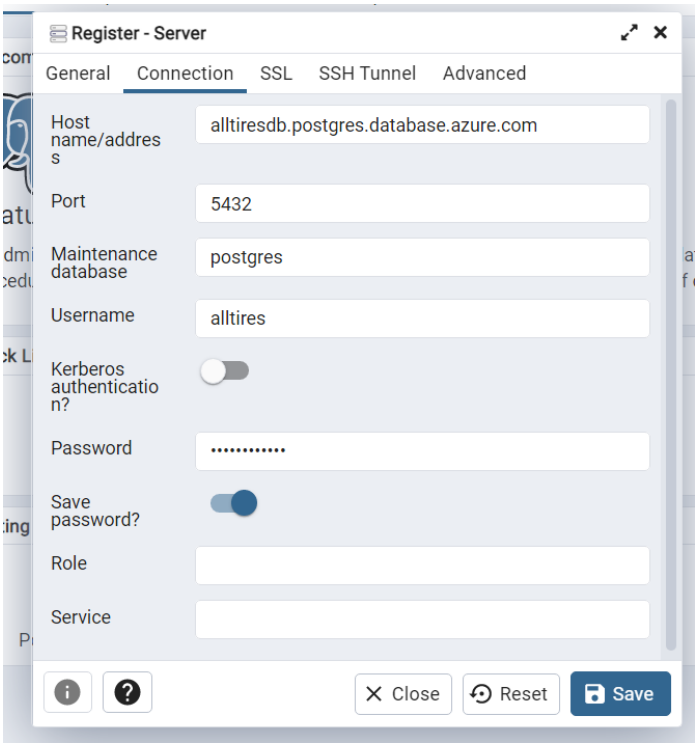


Imagen No. 3: Configuración en pgAdmin

Una vez que se realiza la conexión, se ejecutan los *queries* para la creación de las tablas juntos con sus relaciones respectivas. El resultado se puede observar en la Imagen No. 4 donde se aprecian todas las tablas de la solución, mismas que fueron expuesta en el diagrama UML.

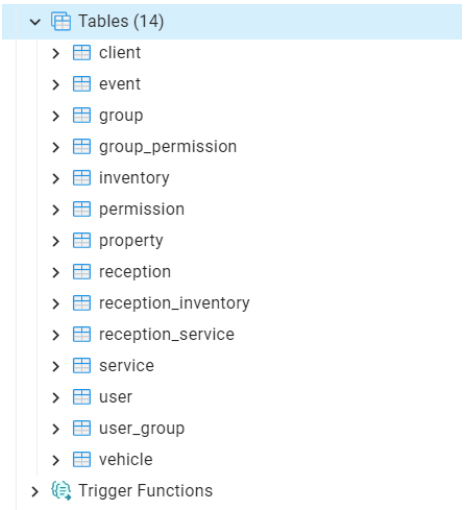


Imagen No. 4: Tablas de la base de datos

Módulo de autenticación.

Como se mencionó en la sección de arquitectura de la aplicación a nivel de *back-end* se implementó la solución con una arquitectura de tres capas. En este caso particular, se divide en repositorios, servicios y controladores. Según Gonzaga, Álvarez, y Gordillo (2006) los componentes distribuidos de esta manera para una arquitectura de *n* capas es esencial para la siguiente generación de aplicaciones que son altamente escalables y de alto rendimiento.

Para otorgar un nivel de seguridad en el proceso de registro y autenticación en la aplicación, así como para el posterior consumo de endpoints de la *API*, se implementó una autenticación basada en la tecnología *Jason Web Token*. Esta tecnología permite que posteriormente de un inicio de sesión se genere un token, con un tiempo de expiración, que contiene información esencial, y que permite el acceso a recursos solamente validados por el token (Bánáti, Kail, Karóczkai , y Kozlovszky, 2018).

La autenticación se realiza mediante mail y contraseña. Por tanto como primer paso se crean métodos necesarios para obtener la información de las entidades *client* y *login.* El

método *GetLoginClientByMail* recibe el parámetro de mail para poder realizar una búsqueda por mail en la entidad *login* como se observa en la Imagen No. 5.

```
3 references
public Login GetLoginClientByMail(string mail)
{
    var result = (from db in mS_AlltiresContext.Login
                  where db.Email == mail
                  select db).FirstOrDefault();
    return result;
}
```

Imagen No. 5: Método GetLoginClientByMail

Ahora con uso de la librería *JWT*, se recibe una clave y se debe verificar si coincide. Es importante mencionar que, para garantizar la seguridad de la información, la clave que ingresa el cliente es encriptada usando el algoritmo *SHA-256*. El proceso de verificar credenciales se lo realiza en el método denominado *verifyCredentials* y que se expone en la Imagen No. 6.

```
public async Task<TokenResponse> verifyCredentials(UserCredentials userCredentials)
{
    MicroservicesEcosystem.Models.Login login = iLoginRepository.GetLoginClientByMail(userCredentials.UserName);
    if (login==null) throw new ArgumentException(Errors.UserNameOrPasswordNoMatch.ToString());
    Boolean successLogin = login.VerifyPassword(userCredentials.Password, login.Password);
    if(successLogin == false) throw new ArgumentException(Errors.UserNameOrPasswordNoMatch.ToString());
    Client client = iClientRepository.GetClientById(login.ClientId);
    UserInformation userInformation = new UserInformation(client);
    return iJwtAuthenticationManager.Authenticate(userInformation);
}
```

Imagen No. 6: Método verifyCredentials

Adicionalmente, una vez que se verifica, extrae la información del cliente del repositorio con el método *GetClientById.* Esta información luego será usada para crear el token *JWT*. El método encargado de generar el token se denomina *Authenticate*, es un token de tipo *Bearer*, con un tiempo de expiración de una hora, y que se agrega información radical para el funcionamiento de la aplicación como lo son el identificador único del

cliente, el mail, y su nombre. Estas propiedades se aprecian en el código expuesto en la Imagen No. 7.

```
public TokenResponse Authenticate(UserInformation userInformation)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var tokenType = "Bearer";
    var expire = DateTime.UtcNow.AddSeconds(3600);
    var tokenKey = Encoding.ASCII.GetBytes(key);
    var tokenDescriptor = new SecurityTokenDescriptor
    {

        Claims = new Dictionary<string, object> {
            { "userId", userInformation.UserId },
            { "email", userInformation.Email },
            { "fullname", userInformation.FullName}
        },
        Subject = new ClaimsIdentity(new Claim[] {
            new Claim(ClaimTypes.AuthenticationMethod,"password"),
        }),
```

Imagen No. 7: Método Authenticate

Por último, se creó el *endpoint* en el controlador *ClientController* que encapsula todo el proceso de verificación de inicio de sesión, y una vez que se ejecuta se obtienen los resultados que se pueden apreciar en la Imagen No. 8. El *endpoint* para el login es */api/client/login* al cual hay que mandar como cuerpo un mail y una contraseña.

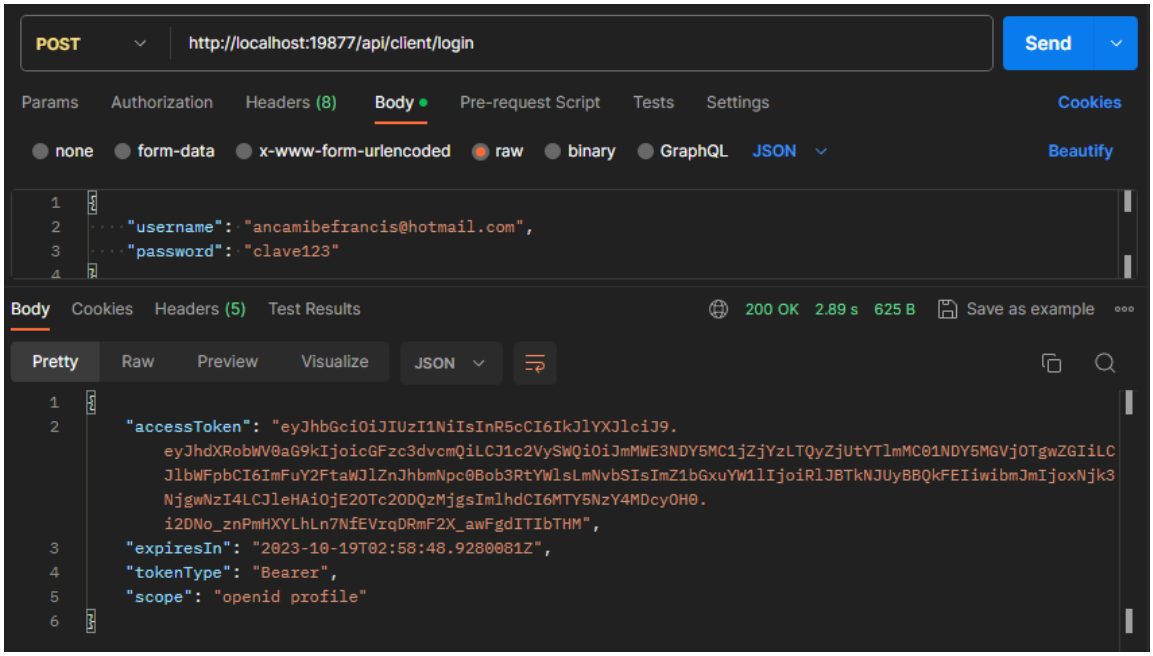


Imagen No. 8: Endpoint para inicio de sesión en Postman

Finalmente, para comprobar que todo el proceso de inicio de sesión es exitoso, se verifica el token generado. Esto se realizó en una herramienta que decodifica *JWT Tokens*. El resultado del token decodificado es el que se expone en la Imagen No. 9.

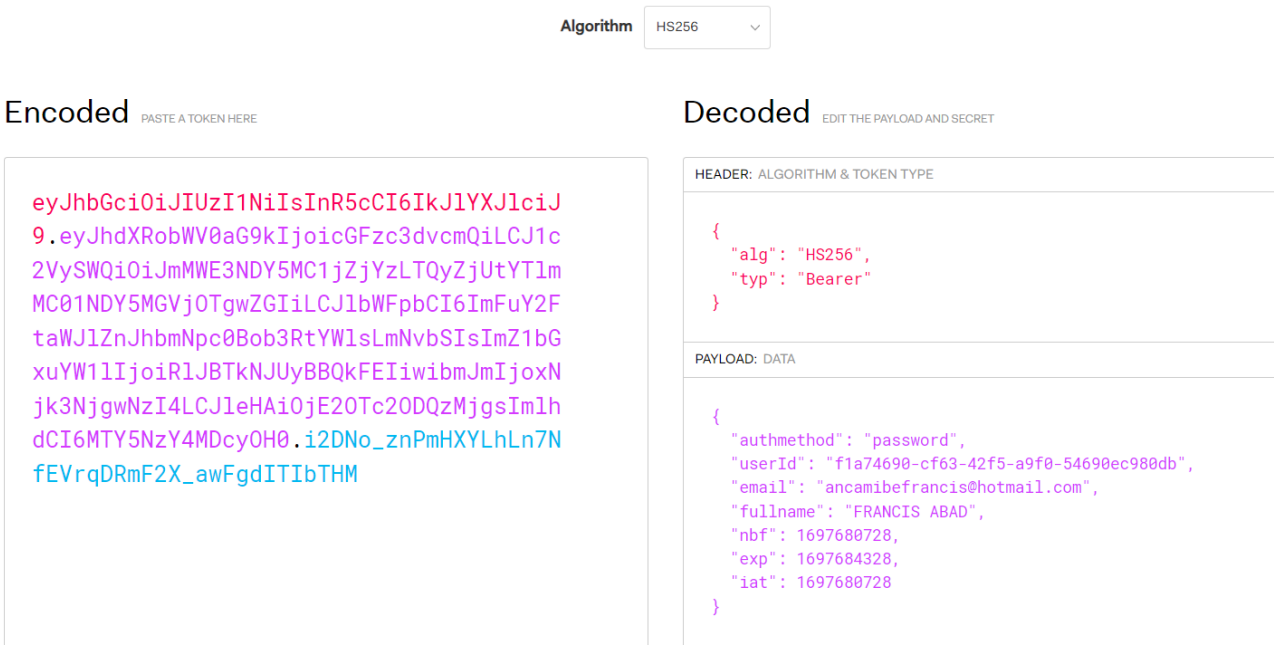


Imagen No. 9: Decodificación de JWT Token

Ahora bien, se explicó el proceso de inicio de sesión. Ahora hay que describir el proceso de un registro de un usuario en *All Tires Connect.* Es importante recordar que el usuario debe existir en la tabla *client*, para poder ser registrado. Eso se implementa en el repositorio para realizar la consulta a la base de datos con el método *GetLoginClientByIdentification* como se ve en la Imagen No. 10.

```
public Login GetLoginClientByIdentification(Guid uuid)
{
    var result = (from db in mS_AlltiresContext.Login
                  .Where(db => db.ClientId == uuid)
                  select db).FirstOrDefault();
    return result;
}
```

Imagen No. 10: Método GetLoginClientByIdentification

Este método es llamado en el servicio donde se implementa la lógica. Si el cliente no cuenta con un registro se procede a realizar una encriptación de la clave ingresada y añadirlo a la base de datos. Esto se hace en *ClientServices* y se encapsula en la función *RegisterLoginClient* expuesta en la Imagen No. 11.

```
public async Task RegisterLoginClient(Login loginRequest)
{
    Login login = loginRepository.GetLoginClientByIdentification(loginRequest.ClientId);
    if (login != null) throw new ArgumentException(Errors.ClientRegistered.ToString());
    loginRequest.HashPassword();
    loginRequest.SetCreatedAt();
    await this.loginRepository.Add(loginRequest).ConfigureAwait(false);
}
```

Imagen No. 11: Método RegisterLoginClient

El último paso es definir un *endpoint* que ejecute dicho método del servicio, y para este caso se definió la ruta */api/client* al que se le envía un cuerpo con la información

necesaria. El consumo del *endpoint* se probó en *Postman* [1]y la verificación de que funciona se observa haciendo una consulta a la base de datos. Esto se denota en la Imagen No. 12 e Imagen No. 13 respectivamente.

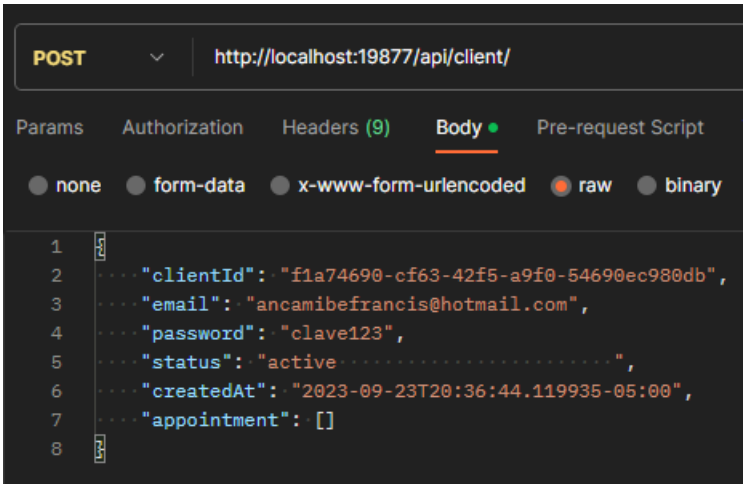


Imagen No. 12: Endpoint para registro de cliente en Postman

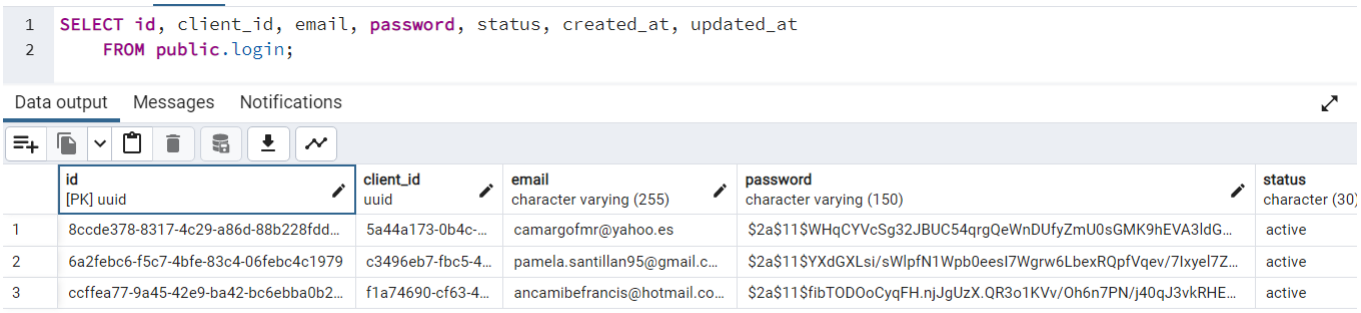


Imagen No. 13: Consulta de tabla login en base de datos

Módulo de historial de mantenimiento.

El módulo de historial de mantenimiento consiste en recopilar los datos de un mantenimiento realizado a un vehículo en particular. Este mantenimiento consta de campos

[1] Plataforma de colaboración para el desarrollo de API que permite a los usuarios diseñar, probar y monitorizar API's.

importantes como el número de la orden de trabajo, fecha, responsable del trabajo, los servicios realizados y el inventario ocupado en caso de que se haya requerido. El mantenimiento se encuentra asociado a la placa de un vehículo, y evidentemente en la aplicación *All Tires Connect*, una vez que se inicia sesión se podrá escoger entre los vehículos asociados al cliente.

Es importante mencionar que a nivel de usuario en el front-end, no solo se presenta el historial de los mantenimientos, sino que se puede observar los mantenimientos en curso o por entregar. Sin embargo, en el presente documento se proporciona una explicación del módulo del historial ya que es más completo y lleva la misma lógica que para mostrar los mantenimientos en curso o por entregar.

Dentro del repositorio *VehicleRepository* se crea el método *GetMaintenancesByLicensePlate* el cual es el encargado de realizar la consulta a la base de datos. Esta consulta devuelve una lista de *Maintenance* por placa de vehículo como se ve en la Imagen No. 14.

```
public List<Maintenance> GetMaintenancesByLicensePlate(string license_plate)
{
    var result = (from vehicle in mS_AlltiresContext.Vehicle
                  join property in mS_AlltiresContext.Property on vehicle.Id equals property.VehicleId into propertyGroup
                  from property in propertyGroup.DefaultIfEmpty()
                  join reception in mS_AlltiresContext.Reception on vehicle.Id equals reception.VehiculeId
                  join user in mS_AlltiresContext.User on reception.UserId equals user.Id
                  where vehicle.LicensePlate.Trim() == license_plate
                  select new Maintenance
                  {
                      ReceptionId = reception.Id,
                      ReceptionNumber = reception.ReceptionNumber.Trim(),
                      StartTime = LocalDateTimeNow.FormatDate(reception.CreatedAt),
                      Responsible = user.FirstName +" "+ user.LastName,
                      Mileage = property.Mileage.Trim()
                  }
        ).ToList();

    return result;
}
```

Imagen No. 14: Método GetMaintenancesByLicensePlate en repositorio de vehículos

El modelo de *Maintenance* se creó para esta solución en específico y se compone de los campos mencionados previamente, en la Imagen No. 15 se observa a nivel de código.

```
public class Maintenance
{
    9 references
    public Guid ReceptionId { get; set; }
    3 references
    public string ReceptionNumber { get; set; }
    3 references
    public string StartTime { get; set; }
    3 references
    public string Responsible { get; set; }
    3 references
    public string Mileage { get; set; }
    3 references
    public List<Service> Services { get; set; }
    3 references
    public List<Inventory> Inventory { get; set; }


}
```

Imagen No. 15: Modelo Maintenance

El modelo *Maintenance* se compone también de una lista de servicios y de inventario. Por lo que se creó dos métodos en sus repositorios respectivos que retornan los servicios e inventario asociados a dicha orden de trabajo. En la Imagen No. 16 se observa la implementación para obtener los servicios, mientras que en la Imagen No. 17 se observa la consulta para obtener el inventario, ambos asociados a un mantenimiento.

```
public List<Service> GetServicesByLicensePlateAndReceptionId(Guid uuid, string license_plate)
{
    var result = (from vehicle in mS_AlltiresContext.Vehicle
                  join reception in mS_AlltiresContext.Reception on vehicle.Id equals reception.VehiculeId
                  join reception_service in mS_AlltiresContext.ReceptionService on reception.Id equals reception_service.ReceptionId
                  join service in mS_AlltiresContext.Service on reception_service.ServiceId equals service.Id
                  where vehicle.LicensePlate == license_plate
                  && reception.Id == uuid
                  select service
        ).ToList();
    return result;
}
```

Imagen No. 16: Método GetServicesByLicensePlateAndReceptionId

```
public List<Inventory> GetInventoryByLicensePlateAndReceptionId(Guid uuid, string license_plate)
{
    var result = (from vehicle in mS_AlltiresContext.Vehicle
                  join reception in mS_AlltiresContext.Reception on vehicle.Id equals reception.VehiculeId
                  join reception_inventory in mS_AlltiresContext.ReceptionInventory on reception.Id equals reception_inventory.ReceptionId
                  join inventory in mS_AlltiresContext.Inventory on reception_inventory.InventoryId equals inventory.Id
                  where vehicle.LicensePlate.Trim() == license_plate
                  && reception.Id == uuid
                  select inventory
        ).ToList();
    return result;
}
```

Imagen No. 17: Método GetInventoryByLicensePlateAndReceptionId

Ahora bien, se procede a implementar el servicio *VehicleServices* que invoca a los métodos definidos en los repositorios, y establece la lógica. Obtiene los mantenimientos por la placa de un vehículo, y por cada mantenimiento, debe hacer una búsqueda para establecer los servicios e inventario correspondientes. Esto se observa en el método *GetMaintenancesByLicensePlate* en la Imagen No. 18.

```
public async Task<List<Maintenance>> GetMaintenancesByLicensePlate(string license_plate)
{
    List<Maintenance> maintenanceList = this.vehicleRepository.GetMaintenancesByLicensePlate(license_plate);
    foreach(Maintenance maintenance in maintenanceList)
    {
        List<Service> serviceList = this.serviceRepository.GetServicesByLicensePlateAndReceptionId(maintenance.ReceptionId, license_plate);
        List<Inventory> inventoryList = this.inventoryRepository.GetInventoryByLicensePlateAndReceptionId(maintenance.ReceptionId, license_plate);
        maintenance.Services = serviceList;
        maintenance.Inventory = inventoryList;
    }
    return maintenanceList;
}
```

Imagen No. 18: Método GetMaintenancesByLicensePlate

Para finalizar, se llama al método implementado del servicio en el controlador y se define un *endpoint*. La ruta *api/vehicle/maintenances/{license_plate}* será la encargada de devolver todos los mantenimientos asociados a una placa de un vehículo, siempre y cuando el usuario esté autenticado en la aplicación. Esto se hace mediante un filtro y se puede observar en la Imagen No. 19.

```
[HttpGet("maintenances/{license_plate}")]
[ServiceFilter(typeof(AuthenticateUserFilter))]
0 references
public async Task<List<Maintenance>> GetMaintenancesByLicensePlate(string license_plate)
{
    return await vehicleServices.GetMaintenancesByLicensePlate(license_plate).ConfigureAwait(false);
}
```

Imagen No. 19: Endpoint para mantenimientos

El resultado del consumo de este *endpoint* es un objeto *Maintenance.* Se lo puede observar en formato *JSON* en el Anexo No. 3. Donde se aprecia todos los campos y valores respectivos.

Módulo de agendamiento de cita.

Se implementa un sistema de agendamiento de cita simplificado según los requerimientos definidos por *All Tires*. Para optimizar el proceso de agendamiento, es esencial recopilar información detallada, incluyendo los datos del cliente, la placa del vehículo, la fecha y la hora de la cita, así como el motivo principal. Esta información permite preparar de manera eficiente y personalizada los servicios requeridos para cada cliente.

Las dos tablas esenciales para el desarrollo de este módulo son *Schedule* y *Appointment* las cuales se pueden observar en el Anexo No. 2. La primera de ellas contiene información relevante a los días disponibles para el agendamiento, mientras que la tabla *Appointment* almacena la información condensada de todo el agendamiento en sí con los campos mencionados previamente.

El primer paso de este módulo se inicia estableciendo cuáles son los horarios en los que trabaja el tecnicentro automotriz y que están a disposición del cliente. Se define el

intervalo entre citas, y por ahora solo se podrá realizar un agendamiento para un horario, así como un agendamiento por vehículo. Estos datos se observan en la Imagen No. 20.

Data output | Messages | Notifications

	id [PK] uuid	day character varying (15)	initial_schedule character varying (6)	final_schedule character varying (6)	interval character varying (6)	created_at timestamp with time zone
1	6711eb5c...	Lunes	8:30	17:00	30	2023-10-26 02:31:18.795...
2	2b8ceb4d...	Martes	8:30	17:00	30	2023-10-26 02:32:01.366...
3	b7967f95...	Miércoles	8:30	17:00	30	2023-10-26 02:32:24.669...
4	3ec9a0fa-...	Jueves	8:30	17:00	30	2023-10-26 02:32:38.870...
5	ee30dfff-...	Viernes	8:30	17:00	30	2023-10-26 02:32:54.705...
6	a663eeb2...	Sábado	8:30	14:00	30	2023-10-26 02:33:08.276...

Imagen No. 20: Datos de tabla Schedule

Con los horarios definidos en la base de datos, se define el repositorio que realiza la consulta. Se define el método *GetAvailableSchedule* cuyo objetivo principal es obtener una lista de horas disponibles para una fecha específica. El método recibe como parámetro una fecha, y realiza la gestión de horarios con las respectivas validaciones de datos. Si existe un horario disponible, calculas las horas de inicia y fin del horario laboral para el día enviado mediante el parámetro.

Se agrega a una lista todas las horas disponibles entre la hora de inicio y la de fin, incrementando en intervalos definidos en la tabla *Schedule.* Adicionalmente se obtiene las citas ya programadas por esa fecha, en caso de que exista, y elimina estas horas de la lista de horarios disponibles. Lo que devuelva la función es una lista de horas para nuevas citas en dicha fecha. Esto se aprecia en la Imagen No. 21.

```
Schedule schedule = scheduleRepository.GetScheduleByDay(day);
if (schedule != null)
{
    string initial_hour = LocalDateTimeNow.FormatDate(date) + " " + LocalDateTimeNow.FormatHour(DateTime.Parse(schedule.InitialSchedule));
    string final_hour = LocalDateTimeNow.FormatDate(date) + " " + LocalDateTimeNow.FormatHour(DateTime.Parse(schedule.FinalSchedule));
    DateTime initial_hour_parsed = DateTime.Parse(initial_hour);
    DateTime final_hour_parsed = DateTime.Parse(final_hour);
    while (initial_hour_parsed < final_hour_parsed)
    {
        hours.Add(LocalDateTimeNow.FormatHour(initial_hour_parsed));
        initial_hour_parsed = initial_hour_parsed.AddMinutes(Double.Parse(schedule.Interval.Trim()));
    }
    List<Appointment> appointments = appointmentRepository.GetAppointmentsByDate(LocalDateTimeNow.FormatDate(date));
    if (appointments.Count > 0)
    {
        foreach (Appointment appointment in appointments)
        {
            hours.Remove(appointment.Hour);
        }
    }
}

return hours;
```

Imagen No. 21: Método GetAvailableSchedule

El siguiente paso es definir un método que retorne el horario para un intervalo de fechas. El método *GetSchedule* recibe como parámetro una fecha inicial y final, de igual manera realiza las correctas validaciones de datos que obtiene, para cada día, el horario disponible usando el método *GetAvailableSchedule* explicado con anterioridad. Se crea un objeto *ScheduleHour* para ese día con la fecha y las horas disponibles como se observa en la Imagen No. 22.

```
while (initial_date_parsed < final_date_parsed.AddDays(1))
{
    List<string> hours = await GetAvailableSchedule(initial_date_parsed);
    if(hours.Count > 0)
    {
        ScheduleHour schedule = new ScheduleHour();
        schedule.Date = LocalDateTimeNow.FormatDate(initial_date_parsed);
        schedule.Hours = hours;
        scheduleHours.Add(schedule);
    }
    initial_date_parsed = initial_date_parsed.AddDays(1);
}
return scheduleHours;
```

Imagen No. 22: Método GetSchedule

Una vez definida la lógica del manejo de fechas y horarios, se implementan los métodos necesarios para las operaciones *CRUD* de *Appointment.* En la Imagen No. 22 se observa la implementación en *AppointmentRepository* de funciones fundamentales para la inserción de datos, la modificación y eliminación de registros.

```
2 references
public async Task<Appointment> Add(Appointment entity)
{
    entity.Id = Guid.NewGuid();
    mS_AlltiresContext.Appointment.Add(entity);
    await mS_AlltiresContext.SaveChangesAsync();
    return entity;
}
2 references
public async Task<Appointment> Update(Appointment entity)
{
    mS_AlltiresContext.Entry(entity).State = EntityState.Modified;
    await mS_AlltiresContext.SaveChangesAsync();
    return entity;
}
2 references
public async Task<Appointment> Delete(Guid id)
{
    var entity = await mS_AlltiresContext.Appointment.FindAsync(id);
    if (entity == null)
    {
        return entity;
    }

    mS_AlltiresContext.Appointment.Remove(entity);
    await mS_AlltiresContext.SaveChangesAsync();
    return entity;
}
```

Imagen No. 23: Métodos para CRUD en AppointmentRepository

Con el repositorio creado, se procede a llamar a los métodos en el servicio *AppointmentServices* que realizan las validaciones y la lógica respectiva. Estos servicios posteriormente serán usados en el controlador para construir los endpoints. Se aprecia en la Imagen No. 24 la implementación para el registro de un nuevo agendamiento. Se realiza primero una búsqueda, si existe un agendamiento arroja una excepción, caso contrario se añade el registro a la base de datos con los datos enviados mediante parámetro.

```
public async Task RegisterAppointment(Appointment appointmentRequest)
{
    Appointment appointment = this.appointmentRepository.GetAppointmentByLicensePlate(appointmentRequest.LicensePlate);
    if (appointment != null) throw new ArgumentException(Errors.VehicleHasAppointment.ToString());
    appointmentRequest.Status = "PENDING";
    appointmentRequest.SetCreatedAt();
    await this.appointmentRepository.Add(appointmentRequest).ConfigureAwait(false);
}
```

Imagen No. 24: Método RegisterAppointment

Para el caso de la modificación de datos se debe realizar una búsqueda en base al *Id* del agendamiento, traer los datos y editar los datos necesarios. De igual manera, en el servicio se maneja el posible error de que no exista un agendamiento con ese *Id.* Se puede observar la implementación en la Imagen No. 25.

```
public async Task UpdateAppointment(Appointment appointmentRequest)
{
    Appointment appointment = appointmentRepository.GetAppointmentById(appointmentRequest.Id);
    if (appointment == null) throw new ArgumentException(Errors.NoAppointmentForId.ToString());
    appointment.Date = appointmentRequest.Date;
    appointment.Hour = appointmentRequest.Hour;
    appointment.Reason = appointmentRequest.Reason;
    appointment.SetUpdatedAt();
    await this.appointmentRepository.Update(appointment).ConfigureAwait(false);

}
```

Imagen No. 25: Método UpdateAppointment

Por último, para el caso de eliminación de un registro de la tabla *Appointment* se realiza una búsqueda por *Id* del agendamiento, y se llama a al método *Delete* expuesto en previamente. La implementación se puede observar en la Imagen No. 26. Con esto, se finaliza las operaciones *CRUD* para el módulo de agendamiento de citas.

```
public async Task DeleteAppointmentById(Guid uuid)
{
    Appointment appointment = this.appointmentRepository.GetAppointmentById(uuid);
    if (appointment == null) throw new ArgumentException(Errors.NoAppointmentForId.ToString());
    await this.appointmentRepository.Delete(uuid).ConfigureAwait(false);
}
```

Imagen No. 26: Método DeleteAppointment

Para finalizar la documentación del módulo del agendamiento de citas, se debe encapsular estos servicios en el controlador *AppointmentController* construyendo los *endpoints* que serán consumidos en el *front-end.*

En la Imagen No. 27 se puede observar la primera parte del controlador y se evidencia el método *POST* y *GET*.

```
[HttpPost(Order = 0)]
[ServiceFilter(typeof(AuthenticateUserFilter))]
0 references
public async Task RegisterAppointment([FromBody] Appointment appointment)
{
    await appointmentServices.RegisterAppointment(appointment).ConfigureAwait(false);
}

[HttpGet("{license_plate}")]
[ServiceFilter(typeof(AuthenticateUserFilter))]
0 references
public async Task<Appointment> GetAppointmentByLicensePlate(string license_plate)
{
    return await appointmentServices.GetAppointmentByLicensePlate(license_plate).ConfigureAwait(false);
}
```

Imagen No. 27: POST y GET de AppointmentController

En la Imagen No. 28 se evidencia la segunda parte del controlador que expone el método *PUT* y *DELETE*.

```
[HttpPut(Order = 0)]
[ServiceFilter(typeof(AuthenticateUserFilter))]
0 references
public async Task UpdateAppointment([FromBody] Appointment appointmentRequest)
{
    await appointmentServices.UpdateAppointment(appointmentRequest).ConfigureAwait(false);
}

[HttpDelete("{uuid}")]
[ServiceFilter(typeof(AuthenticateUserFilter))]
0 references
public async Task DeleteAppointment(Guid uuid)
{
    await appointmentServices.DeleteAppointmentById(uuid).ConfigureAwait(false);
}
```

Imagen No. 28: PUT y DELETE de AppointmentController

Front-end

El primer paso para el desarrollo del *front-end* es tener un bosquejo inicial de las vistas de la aplicación para entender como es el flujo de las vistas, y brindar la mejor experiencia de usuario posible. Los diseños aprobados por *All Tires* se muestran en Anexo No. 4, en el cual se procuró mantener la identidad visual de la empresa, crear un logo que sea sinérgico con el logo de la empresa, y ofrecer una experiencia de usuario intuitiva.

Es importante aclarar que el *back-end* se desarrolló en paralelo con el *front-end* por módulos, por lo que en el desarrollo existieron algunas modificaciones con respecto al bosquejo inicial. El resultado final de las vistas de la aplicación se puede observar en el Anexo No. 5, el cual representa el primer prototipo completamente funcional de *All Tires Connect.*

Como primer paso, se realiza todo lo que refiere a la estructura del proyecto, creación de directorios, creación de interfaces, sistema de rutas, menú de navegación y primeras codificaciones con textos fijos en las vistas. Adicionalmente se definen variables de color, importación de imágenes y edición de hojas de estilos.

Se crea servicios para consumir los *endpoints* definidos en los controladores respectivos del *back-end*. Para ello se utilizó la librería *Axios* que realiza solicitudes *HTTP* y la lógica respectiva para cada método. A continuación, en la Imagen No. 29 se observa el diagrama de secuencia para el módulo de autenticación e ilustra el proceso de ingreso de credenciales por parte del usuario, la solicitud de autenticación, verificación de credenciales y la respuesta con un token al usuario.

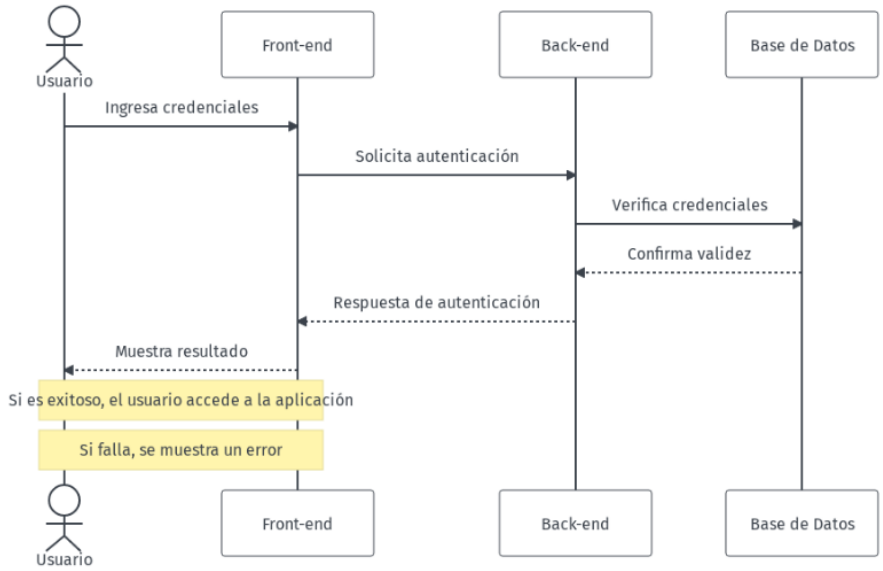


Imagen No. 29: Diagrama de secuencia de módulo de autenticación

A continuación, se expone en la Imagen No. 30 el diagrama de secuencia del módulo de historial de mantenimiento en la que se aprecia la interacción del *front-end* solicitando el historial al *back-end*, que a su vez consulta la base de datos por vehículo. Retorna la información al usuario de cada mantenimiento, incluyendo fechas, servicios realizados, inventario agregado y los responsables de dicho mantenimiento.

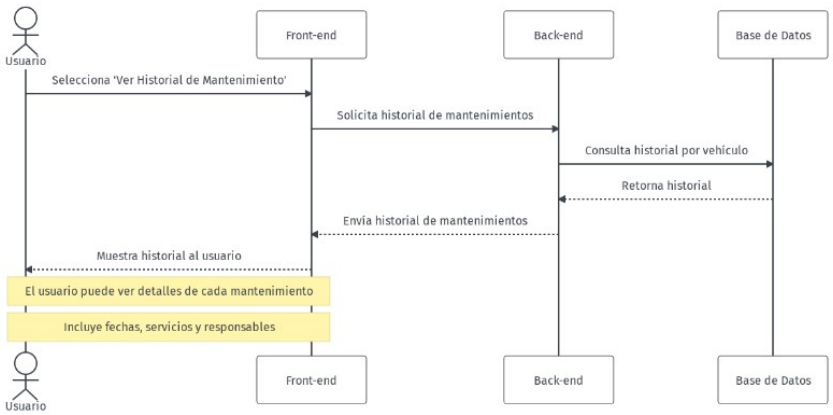


Imagen No. 30: Diagrama de secuencia de módulo de historial de mantenimiento

Para el caso del módulo de agendamiento de cita, se observa en la Imagen No. 31 su diagrama de secuencia correspondiente, en el que el usuario proporciona detalles como hora, fecha y motivo principal del agendamiento. Los datos se envían al *back-end*, verifica la disponibilidad en la base de datos, y si es posible confirma el agendamiento, culminando en la presentación al usuario con la confirmación de la cita.

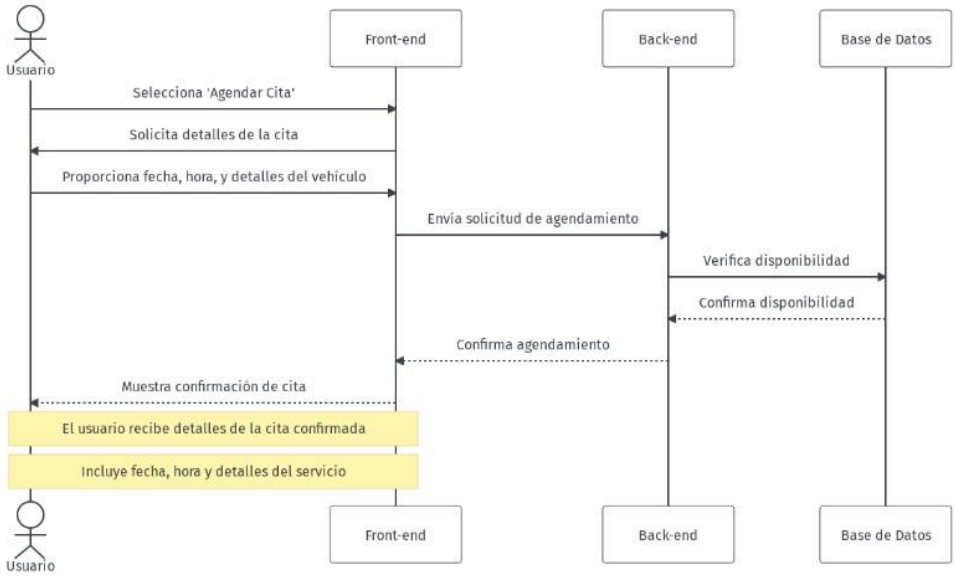


Imagen No. 31: Diagrama de secuencia de módulo de agendamiento de citas

En la etapa final del desarrollo del *front-end*, se implementa una lógica detallada para cada método, la cual invoca a los servicios ya definidos. Tras ejecutar las solicitudes *HTTP*, el archivo *Typescript* vinculado a cada interfaz de la aplicación activa estos métodos, facilitando el *data binding* con el *HTML* respectivo. Esta integración aprovecha los recursos construidos en el *back-end* para formar una aplicación robusta, intuitiva y con una experiencia de usuario coherente.

Esta fase culmina con la aplicación lista para ser presentada como un primer prototipo funcional, entrando en una etapa crucial de pruebas para la detección y corrección de errores, asegurando así la calidad y funcionalidad del producto final.

Pruebas

Esta sección se enfoca en evaluar la funcionalidad y usabilidad de la aplicación *All Tires Connect* considerando la perspectiva del comercio como de su clientela. El objetivo es que cumpla con los requisitos técnicos, pero también una experiencia de usuario óptima. Así, el proceso de pruebas se lleva a cabo en dos fases distintas.

La primera fase consiste en realizar pruebas internas de funcionalidad para verificar que los requerimientos del cliente se hayan cumplido. Por otra parte, se debe verificar que exista una integración correcta con el sistema *All Tires Support*. La segunda fase consiste en probar la usabilidad en situaciones reales de uso con aquellos clientes que estén dispuestos a otorgar retroalimentación o sugerencias acerca de la aplicación.

La empresa *All Tires* otorgó comentarios mayormente positivos, según la administradora “es una aplicación simple y directa, que coloca a nuestro tecnicentro automotriz a la vanguardia tecnológica que demanda el mercado actual. Da a nuestra clientela información oportuna sobre los mantenimientos de sus vehículos, y por ende seguridad al momento de conducir”.

La clientela también proporcionó valiosos comentarios enfocados en la funcionalidad de la aplicación. Sugirieron mejoras en la navegabilidad y presentaron ideas para futuras implementaciones, como son sistema de notificaciones y la opción de añadir servicios adicionales a través de la aplicación, que podrían realizarse mientras el vehículo se encuentra en el taller. Los comentarios se realizaron sobre el primer prototipo de *All Tires Connect* que se encuentra expuesto en el Anexo No. 5.

Los comentarios recopilados durante la fase de pruebas son fundamentales para el desarrollo continuo de *All Tires Connect.* La aplicación ha demostrado ser una herramienta eficaz y directa para la empresa y su clientela, con detalles por pulir y mejorar, como es de costumbre en el desarrollo de software. Sin embargo, demuestra ser una base sólida y robusta para el desarrollo futuro de la aplicación.

CAPÍTULO IV - CONCLUSIONES

Conclusiones

El proyecto ha sido de vital importancia para la transformación digital en el sector automotriz, y sobre todo para la empresa *All Tires.* La aplicación *All Tires Connect* desea otorgar beneficios significativos a la empresa y a su clientela. Mediante esta aplicación, los clientes del tecnicentro son capaces de acceder a la información de sus vehículos de una manera eficiente y modernizada. Ofrece un servicio personalizado y reduce tiempos de espera mediante su módulo de agendamiento de citas.

La aplicación contribuye en el proceso de fidelización de la clientela, y en la atracción a nuevos clientes, posicionando a *All Tires* en la vanguardia tecnológica del sector. Además, permite una gestión interna mejorada, pues la facilidad de acceso al historial de mantenimientos ayuda a los técnicos a planificar con anticipación los mantenimientos futuros. En algunos casos, esta herramienta resulta esencial para identificar y anticipar posibles averías.

Las tecnologías implementadas fueron un pilar fundamental en el objetivo de desarrollar una aplicación intuitiva, pues *Ionic Framework* fue esencial para el desarrollo del *front-end,*

mientras que otras herramientas como *Figma* apoyaron en el proceso de diseño y definición de la experiencia de usuario.

En lo que se refiere al desarrollo del *back-end*, la tecnología *ASP .NET Core* jugó un papel esencial para el éxito del proyecto, pues ofrece un entorno de desarrollo robusto y flexible. Para el diseño y modelado de la base de datos se usó *Visual Paradigm* que permite crear un diseño *UML* preciso. Además, la elección de Microsoft Azure como plataforma de alojamiento garantiza una infraestructura confiable y escalable, lo que es decisivo para un rendimiento optimo.

Finalmente, se consiguió una solución tecnológica que sea escalable y modular, estableciéndose como un primer prototipo funcional para la empresa *All Tires*. Esta solución no solo cumple con los requisitos actuales de la empresa, sino que también ofrece una base sólida para futuras expansiones y mejoras. *All Tires Connect* representa un avance significativo en el camino hacia la transformación digital, posicionándose a la vanguardia de las evoluciones tecnológicas en el sector automotriz.

Trabajo a futuro

El ámbito de las aplicaciones móviles se caracteriza por su constante innovación y evolución, siendo crucial mantenerse a la par con el rápido avance de las nuevas tecnologías y las cambiantes demandas de los usuarios. La versión actual de *All Tires Connect* es la base sobre la cual se realizarán distintas mejoras con el fin de que la aplicación se mantenga a la vanguardia en un mercado competitivo.

La primera fase del trabajo a futuro consiste en el despliegue de la aplicación móvil en la Play Store. Esto ayudará a aumentar la accesibilidad y facilidad para descarga la solución por parte de los usuarios finales. El despliegue además de incrementar la accesibilidad, facilitará la

distribución de actualizaciones y mejores, asegurando que los usuarios siempre tengan acceso a la última versión de la aplicación.

Un aspecto importante por desarrollar es la implementación de notificaciones *push*, las cuales permiten informar al usuario sobre cambios en la base de datos. Esencialmente, las notificaciones indicarían a los usuarios que su vehículo está listo para ser retirado, alertar que se ha añadido un nuevo mantenimiento en la orden de trabajo en curso, o un recordatorio para un próximo mantenimiento. Esta funcionalidad mejorará la comunicación entre el taller y los clientes, incrementando la transparencia y la confianza.

En lo que respecta al módulo de agendamiento de citas, se planea un desarrollo para facilitar la gestión y visualización en el tecnicentro automotriz. Sin embargo, es importante aclarar que esto formará parte de la solución *All Tires Support*. Con ello, las personas a cargo podrán coordinar y realizar un seguimiento de las citas de todos los clientes.

Además de las mejoras técnicas, también se planea una revisión constante de la interfaz de usuario y de la experiencia de usuario. Esta estrategia no solo se enfocará en el diseño estético, sino también en la funcionalidad y navegabilidad de la aplicación. Se llevarán a cabo pruebas de usabilidad para recoger retroalimentación de los usuarios y realizar los ajustes pertinentes en la interfaz.

El objetivo a futuro es perfeccionar *All Tires Connect* en todos sus aspectos, asegurando que no solo cumpla, sino que supere las expectativas de los usuarios. Se aspira que la aplicación se convierta en un referente en el mercado de aplicaciones de Quito destinadas al sector automotriz, destacándose por su innovación y facilidad de uso. Estos esfuerzos son fundamentales para asegurar que *All Tires Connect* sea una solución efectiva en el presente y un aliado indispensable para los usuarios en el futuro.

Referencias

Bánáti, A., Kail, E., Karóczkai , K., & Kozlovszky, M. (2018). Authentication and authorization orchestrator for microservice-based software architectures. *41st International Convention on Information and Communication Technology* (págs. 1180-1184). Opatija, Croacia: Electronics and Microelectronics (MIPRO).

Gonzaga, E., Álvarez, J., & Gordillo, A. (2006). *Arquitecturas en n-Capas: Un Sistema Adaptivo.* Ciudad de México, México: Polibits.

Kuusinen, K., & Mikkonen , T. (2014). On Designing UX for Mobile Enterprise Apps. *40th EUROMICRO Conference on Software Engineering and Advanced Applications*, (págs. 221-228). Verona, Italia.

Lock, A. (2023). *ASP.NET Core in Action, Third Edition.* Nueva York, Estados Unidos: Manning.

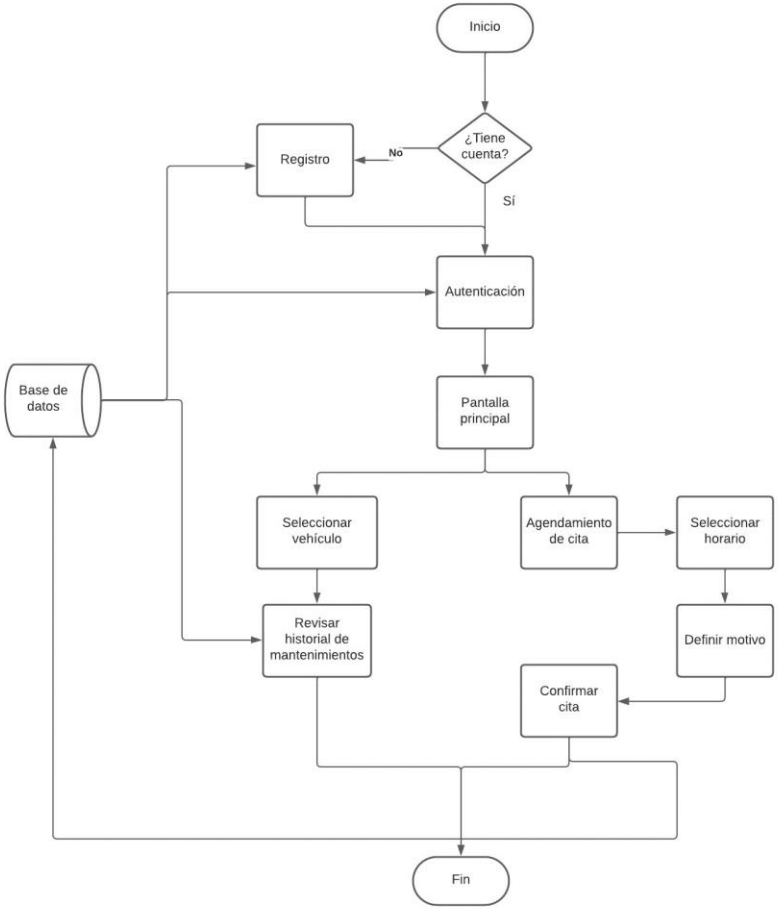
Microsoft. (2023). *Azure Database for PostgreSQL - Flexible Server*. Obtenido de Azure Documentation: https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/overview

Prayogi, A., Niswar, M., Indrabayu, A., & Rijal, M. (2020). Design and Implementation of REST API for Academic Information System. *IOP Conference Series: Materials Science and Engineering*, 12-47. doi:10.1088/1757-899X/875/1/012047

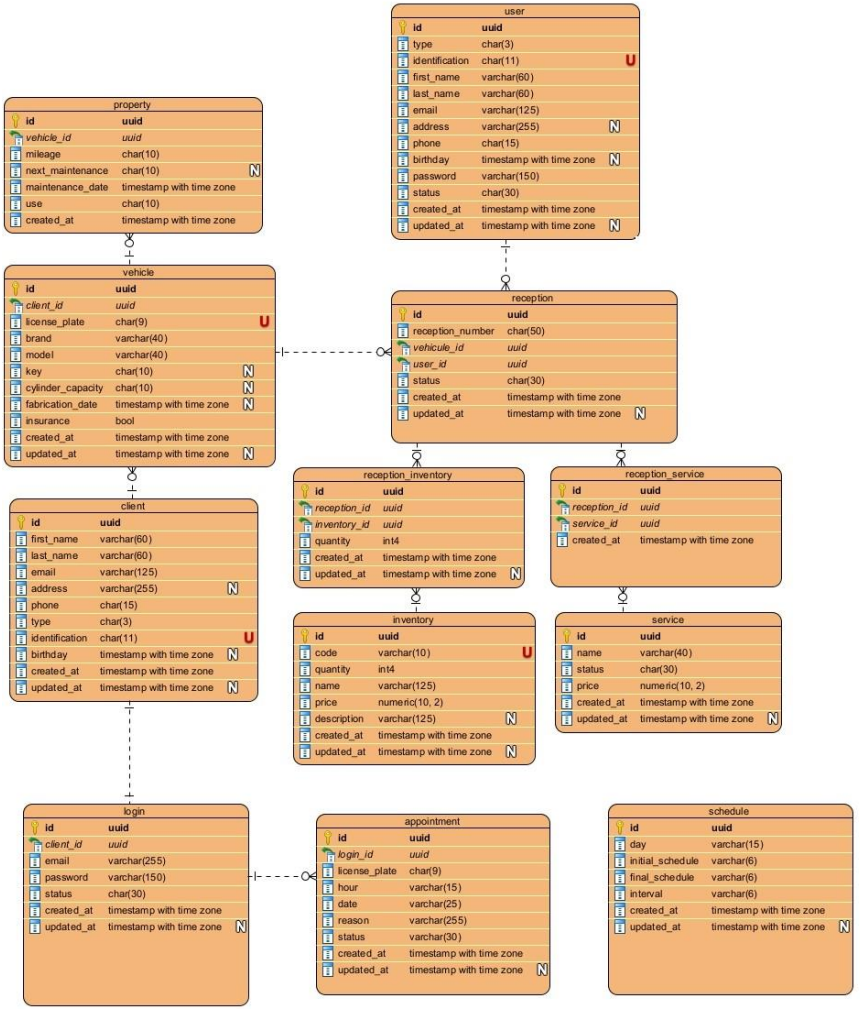
Pujari, V., Patil, R., & Sutar, S. (2020). A Review on Best Practices in Mobile Application Development. *National Seminar on "Trends in Geography, Commerce, IT And Sustainable Development.* Khed, India.

Román, A. (2015). *Aplicación móvil multiplataforma para el aprendizaje de lenguajes de programación.* Madrid, España: Escuela Tecnica Superior de Ingeniería y Sistemas de Telecomunicación.

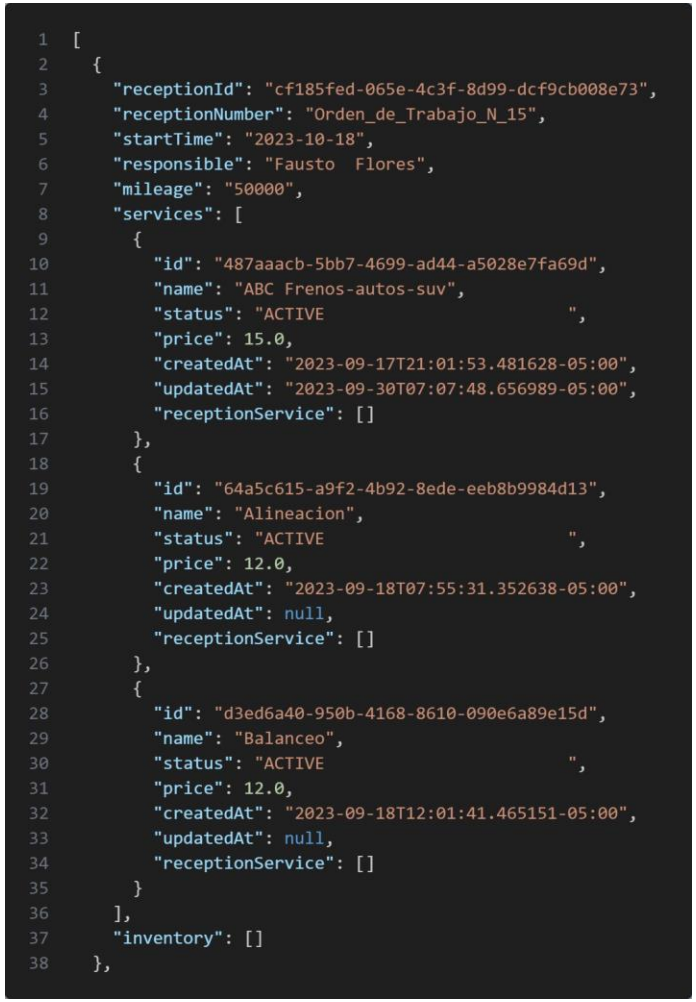
ANEXOS



Anexo No. 1: Diagrama de flujo de All Tires Connect



Anexo No. 2: Diagrama UML de All Tires Connect



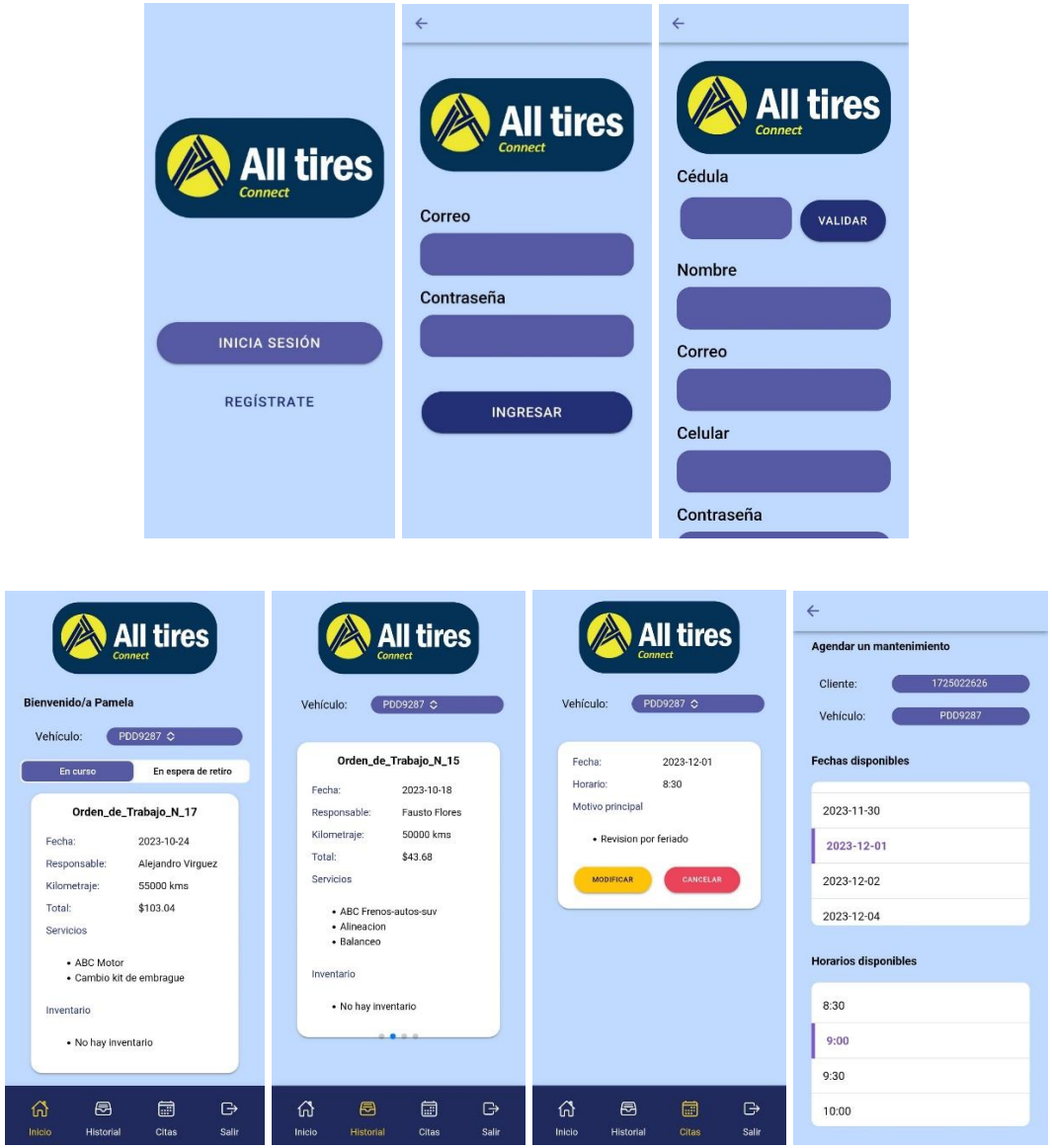

```
[
  {
    "receptionId": "cf185fed-065e-4c3f-8d99-dcf9cb008e73",
    "receptionNumber": "Orden_de_Trabajo_N_15",
    "startTime": "2023-10-18",
    "responsible": "Fausto  Flores",
    "mileage": "50000",
    "services": [
      {
        "id": "487aaacb-5bb7-4699-ad44-a5028e7fa69d",
        "name": "ABC Frenos-autos-suv",
        "status": "ACTIVE                          ",
        "price": 15.0,
        "createdAt": "2023-09-17T21:01:53.481628-05:00",
        "updatedAt": "2023-09-30T07:07:48.656989-05:00",
        "receptionService": []
      },
      {
        "id": "64a5c615-a9f2-4b92-8ede-eeb8b9984d13",
        "name": "Alineacion",
        "status": "ACTIVE                          ",
        "price": 12.0,
        "createdAt": "2023-09-18T07:55:31.352638-05:00",
        "updatedAt": null,
        "receptionService": []
      },
      {
        "id": "d3ed6a40-950b-4168-8610-090e6a89e15d",
        "name": "Balanceo",
        "status": "ACTIVE                          ",
        "price": 12.0,
        "createdAt": "2023-09-18T12:01:41.465151-05:00",
        "updatedAt": null,
        "receptionService": []
      }
    ],
    "inventory": []
  },
```



```
{
    "receptionId": "47543cdf-4738-4148-b972-81eaba00bf9a",
    "receptionNumber": "Orden_de_Trabajo_N_17",
    "startTime": "2023-10-24",
    "responsible": "Alejandro Virguez",
    "mileage": "55000",
    "services": [
      {
        "id": "7f508b9b-5fac-4eee-b2fb-5bba08512dac",
        "name": "ABC Motor",
        "status": "ACTIVE                          ",
        "price": 22.0,
        "createdAt": "2023-09-18T12:02:07.509945-05:00",
        "updatedAt": null,
        "receptionService": []
      },
      {
        "id": "65e88058-9ab9-46f7-98fc-e35490d2b5e9",
        "name": "Cambio kit de embrague",
        "status": "ACTIVE                          ",
        "price": 70.0,
        "createdAt": "2023-09-18T12:03:08.696516-05:00",
        "updatedAt": null,
        "receptionService": []
      }
    ],
    "inventory": []
  }
]
```


Anexo No. 3: Respuesta de endpoint con lista de objetos Maintenance

Anexo No. 4: Diseño preliminar en Figma



Anexo No. 5: Vistas de primer prototipo de All Tires Connect en dispositivo