fidélitas
INGENIERÍA EN SISTEMAS DE COMPUTACIÓN

DN01 Documento de Arquitectura

Limitless

HOJA DE CONTROL

Proyecto	*Limitless*		
Entregable	*G3_SC603_M_Factibilidad*		
Autores	*José Araya, Adrián Cordero, Josué Carmona, Johannes Sequeira*		
Versión/Edición	0100	**Fecha Versión**	19/11/2025
		Nº Total de Páginas	*12*

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1	Creación del documento	Todo el equipo	19/11/2025

Contenido

1. Introducción

Este proyecto nace como una solución tecnológica diseñada para modernizar y optimizar la gestión de usuarios, clientes, vehículos y citas dentro de un entorno que requiere organización, precisión y acceso controlado a la información. La plataforma busca integrar en un solo sistema procesos que tradicionalmente se realizan de forma manual o aislada, brindando herramientas que facilitan la operación diaria tanto a administradores como a mecánicos y clientes.

Para lograrlo, se construyó sobre una arquitectura de microservicios que permite dividir las funciones en módulos independientes, cada uno con su propia lógica y base de datos. Esto hace posible un desarrollo más ordenado, una mayor capacidad de escalamiento y una administración más clara de los datos.

2. Propósito

El propósito de este proyecto es ofrecer una solución digital que facilite y optimice la gestión de procesos relacionados con usuarios, clientes, vehículos y citas. La plataforma busca centralizar información clave y automatizar tareas que normalmente requieren tiempo, coordinación manual y supervisión constante. Con ello se pretende mejorar la organización interna, reducir errores operativos y brindar una experiencia más práctica tanto para el personal administrativo y mecánico como para los propios clientes.

3. Descripción del sistema

Limitless es una plataforma web diseñada para administrar de forma integrada la información y los procesos relacionados con usuarios, clientes, vehículos y citas. El sistema permite que diferentes tipos de usuarios ingresen con roles específicos como administrador, mecánico y cliente y realicen únicamente las acciones que les corresponden según su perfil. Esto garantiza un manejo seguro y coherente de los datos, evitando que información sensible o procesos críticos se modifiquen sin autorización.

La aplicación funciona bajo una arquitectura de microservicios, donde cada módulo como Usuarios, Clientes o Citas que operan de manera independiente con su propia base de datos. Esta estructura permite que los componentes crezcan, se actualicen o se mantengan sin afectar al resto del sistema.

Limitless está diseñado para integrarse con servicios en la nube como Microsoft Azure, lo que permite desplegar la plataforma en entornos confiables, con alta disponibilidad y listos para escalar según la demanda. En conjunto, el sistema busca mejorar la experiencia de los usuarios finales, mantener la información organizada y brindar herramientas que hagan más eficiente el trabajo diario.

4. Objetivos y Restricciones arquitectónicas

Se busca en el sistema manejar y distribuir el tráfico mediante un sistema de microservicios que permita un escalado independiente entre los componentes, además de realizar una arquitectura que permita funcionalidad y permitir mecanismos de mensajes de errores, reintentos y timeouts.

Además, el sistema debe de ser mantenible y escalable para la incorporación de nuevas funcionalidades sin la necesidad de reprogramar los componentes ya existentes, por lo que la separación de los microservicios en sistemas separados y el uso de buenas prácticas son fundamentales.

También se debe dar una buena seguridad por medio de métodos de autentificación, autorización y protección de los datos tanto para las vistas del Front-End como las funcionalidades de los microservicios por lo que se deben agregar una gestión de roles y políticas de acceso para cada uno de estos roles.

4.1. Plataforma tecnológica

La plataforma consta de de un conjunto de tecnologías que permita la construcción de dos sistemas principales por separado, una vista Front-End que permita a los usuarios finales interactuar con el sistema, el cuál estaría bajo una arquitectura de microservicios y cada uno de estos microservicios, representaría una funcionalidad dentro del sistema, siendo independiente pero permitiendo la interoperabilidad, por medio del el uso de protocolos de comunicación de fácil entendimiento como REST.

4.1.1. Aplicación

Para la arquitectura de la aplicación Limitless, se decidió el optar por un modelo de Microservicios, donde el Front-End creado por medio de NodeJS, NextJS y JavaScript, va a consumir los distintos APIs por medio de AJAX. La conexión entre el Fron-End y los microservicios va a ser manejado por un API Endpoind, el cual va a redirigir las solicitudes para cada uno de los endpoints de desarrollados para cada funcionalidad de la aplicación, como lo sería el módulo de Usuarios, Clientes, entre otros.

Cada API va a consistir en un proyecto separado de Dotnet core, utilizando minimal API y mediante el uso de Entity Framework, se va a construir cada conexión con la respectiva base de datos utilizando una solución de Database first.

Cabe recalcar que al ser una arquitectura de microservicios, cada uno de los servicios va a tener su base de datos dedicada, segmentando los datos en distintos módulos.

4.2. Seguridad

El principal método de seguridad es mantener un sistema de autorización por medio de una cuenta obligatoria para el acceso al sistema, se busca alejar a personas no autorizadas y no habilitar el sistema tan abiertamente para identificar y brindar accesos dados por roles para limitar las funcionalidades dentro del sistema.

El sistema posee tres roles principales, los cuales son: Administrador, mecánico y cliente; donde el administrador posee la completa visibilidad de la página sin restricciones con la habilidad de ver, editar, agregar y eliminar en todos los componentes del sistema, El mecánico tiene acceso a visualizar citas, usuarios y vehículos, más solo puedo editar detalles dentro de las citas por lo que no puede agregar información nueva , si no actualiza el sistema con los datos brindados para el sistema.

Los clientes si pueden crear citas y además editar vehículos y su perfil, pero no todos los detalles serán editables, se limitará las opciones ya que se debe mantener una consistencia entre los datos editables por el mecánico y los datos del cliente, porque se busca evitar que se hagan conflictos entre discrepancias de datos entre el cliente y el empleado, por lo que detalles específicos relacionados al progreso de la cita serán editables por empleados, pero para editar deberá ponerse en contacto con un administrador.

Además, el manejo de credenciales será completamente tokenizado bajo un esquema JWT que permita la validación de credenciales correctas utilizando una comunicación segura entre las dos partes por medio de un objeto JSON (JWT, s.f.) y la denegación del acceso a terceros no deseados por lo que con ayuda de este EntityFramewok.

4.3. Tecnologías utilizadas

Detalle ampliamente los lenguajes de programación y tecnologías utilizadas en el proyecto

Entre las tecnologías a implementar se utilizará el lenguaje de programación C#, debido a que utilizar este lenguaje se puede implementar frameworks como .NET Core, el cual permite crear sistemas y microsistemas web y simplificar funcionalidades de JavaScript o TypeScript creando servicios con tiempos de respuesta muy rápidos y ofreciendo seguridad ante múltiples protocolos, además de ofrecer integración nativa para hosting en Microsoft Azure, siendo la plataforma en la cual se planea desplegar la aplicación (Microsoft, s.f.).

4.4. Herramientas utilizadas para el desarrollo

Durante el desarrollo del proyecto y con el fin de llegar a un final satisfactorio se utilizarán las siguientes herramientas

Detalle softwares e IDE utilizados en el desarrollo del proyecto

Nombre	Descripción
ASP.NET Core	Permite utilizar el lenguaje de programación C# para construir servicios web de manera sencilla moderna y segura.
Microsoft Visual Studio Code	Permite centralizar el desarrollo, la compilación, el debuging del código y además descargar plugins que extiendan las funcionalidades del editor de código para ofrecer una experiencia similar o incluso superior a la de un IDE.
Microsoft Azure	Permite subir la página web a servidores en la nube e implementar distintos tipos de automatización de procesos por medio de máquinas virtuales personalizables para instalar certificados y proveer un dominio público.
Microsoft SQL Server	Permite almacenar toda la información que se manejará dentro de la aplicación y ordenarla dependiendo de las distintas funcionalidades que los usuarios necesiten almacenar o consultar.

4.5. Estándares de desarrollo

El desarrollo de Limitless se realiza siguiendo prácticas que aseguran orden, coherencia y calidad en todo el sistema. Al trabajar con ASP.NET Core, se mantiene una estructura organizada por capas que separa la lógica de negocio, el acceso a datos y los controladores de las API. Esto facilita la mantenibilidad del código y garantiza que todos los servicios sigan el mismo estilo, utilizando convenciones REST y respuestas en formato JSON.

El uso de Visual Studio Code establece también lineamientos claros para la escritura del código, como el uso de PascalCase en clases y métodos, camelCase en variables internas y archivos estructurados de forma sencilla y comprensible. Se procura que el código sea limpio, legible y bien distribuido, evitando redundancias o prácticas que dificulten su mantenimiento.

En cuanto al manejo de datos, Microsoft SQL Server se utiliza bajo un modelo bien definido, con tablas y relaciones nombradas de forma coherente y con reglas claras para las claves primarias y foráneas. Todas las operaciones se gestionan desde ASP.NET Core de manera controlada, asegurando integridad, validaciones y protección de datos sensibles.

El despliegue se realiza mediante Microsoft Azure, donde se mantienen configuraciones seguras, entornos definidos y registros claros de cada actualización. Finalmente, el control de versiones se lleva a cabo en GitHub, utilizando un flujo organizado que permite rastrear cambios y coordinar el trabajo del equipo de forma eficiente.

4.5.1. Datos de diseño

Tipografías	
Títulos Primarios	DM Sans Bold
Títulos Secundarios	DM Sans Bold Italic
Texto Plano	Dm Sans Regular
Acentos y enlaces	Dm Sans Regular
Cromática	
Blanco (#F4F7FE) Gris (#909EBB) Morado (#462EFF) Cian (#7EDBFD) Celeste (#E4ECF9)	
Iconografía	
	

4.6. Componentes

4.6.1. Servidor de pruebas

En este servidor se ejecutaran al menos tres maquinas virtuales, donde en una se encontrara el Front-End en Node JS, en la segunda maquina se encontrara el API Gateway con cada microservicio y en la tercera maquina estarán las cinco bases de datos.

Para el servidor principal, donde va a estar corriendo cada maquina virtual, se piensa destinar aproximadamente 2 nucleos de procesador, 512 Gigabytes de almacenamiento y 16 GB de RAM. Esto para poder crear luego las tres maquinas virtuales con especificaciones optimas para probar el codigo. Cada maquina va a contar con 2 nucleos, 128 Gigabites de almacenamiento y 4 GB de memoria RAM.

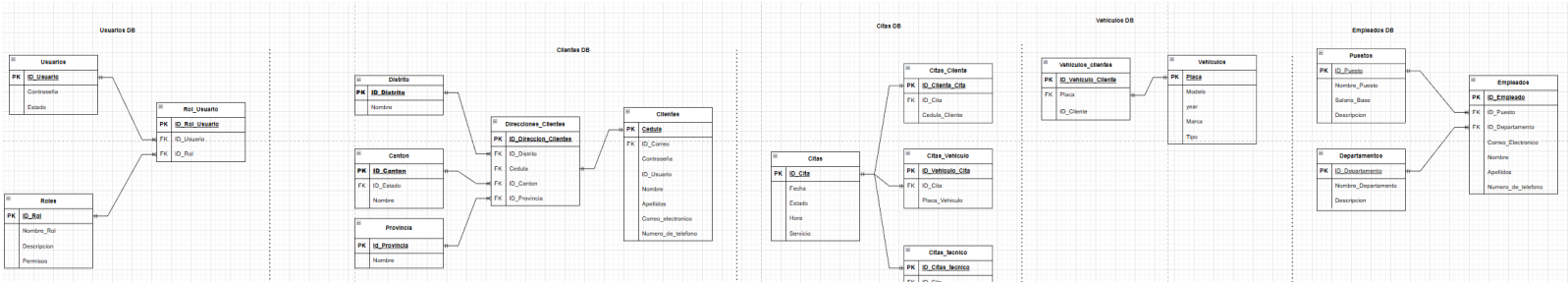
4.6.2. Servidor de producción

El servidor de producción se espera que este corriendo en Azure, ya sea en maquinas virtuales ya destinadas completamente a cada uno de los componentes de la aplicación. Cada maquina virtual contara con 256GB de almacenamiento, 8GB de memoria RAM y 4 núcleos de procesador.

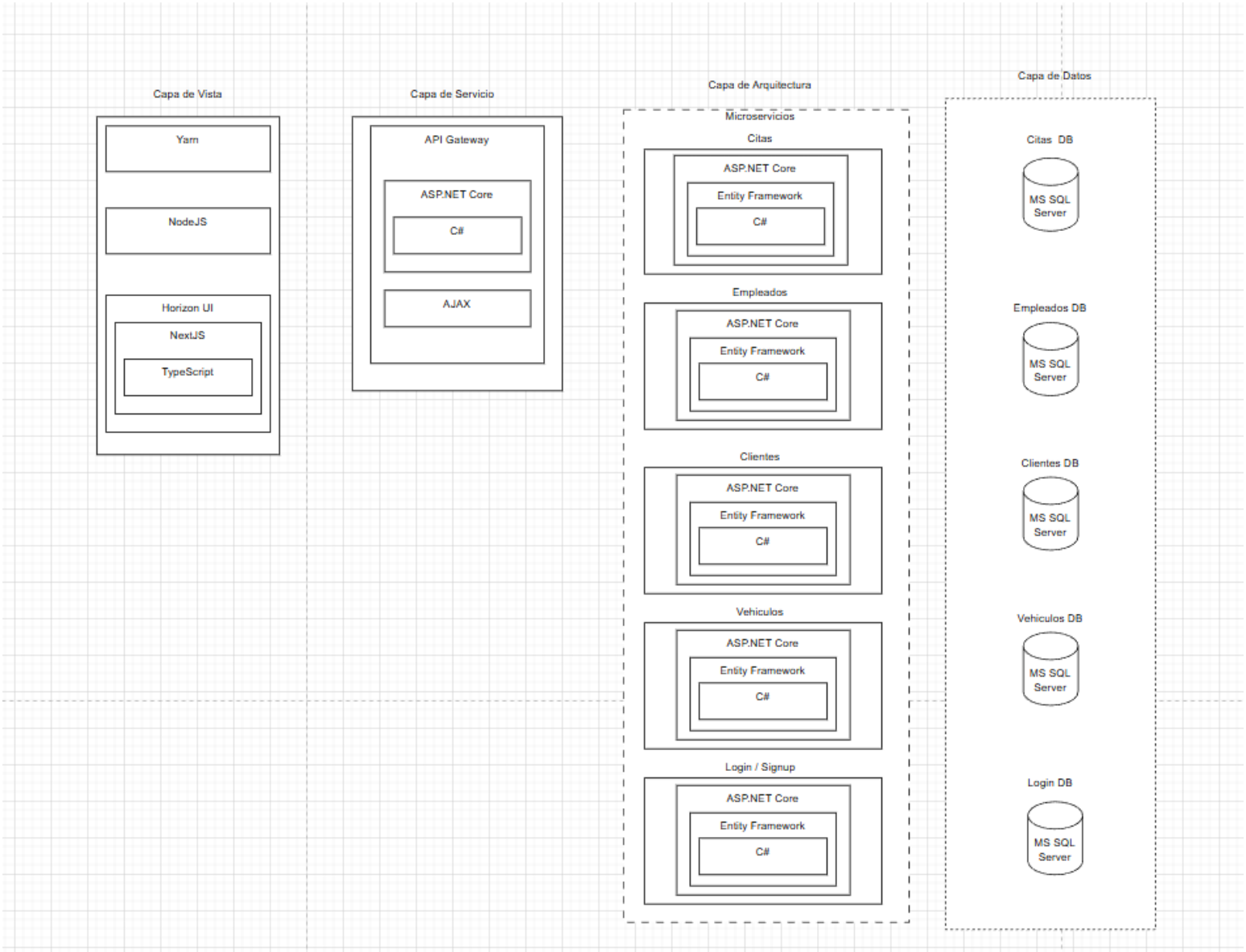
4.6.3. Base de datos

Herramientas de Construcción	
Entorno de Desarrollo	SQL Server Management Studio 18.4
Lenguaje	SQL Transact-SQL
Sistema Gestor de Base de Datos	SQL Server 2022
Justificación	
Para las bases de datos del sistema, se escogió SQL Server, la cual es una base de datos ligera, y sencilla para empezar el desarrollo de aplicación. Sin embargo, cuando la aplicación ya esté disponible para los consumidores, la idea es migrar a bases de datos mucho más robustas en sistemas más adaptados al nivel de carga como Azure SQL o PosgreDB.	

4.6.4. Diagrama entidad-relación (si aplica)



4.6.5. Diagrama de infraestructura



5. Control de Cambios y Versiones

Para un desarrollo apropiado del codigo del proyecto, se planea no solamente utilizar GitHub como el principal software de control de versiones, si no que, sumado a eso, se van a incorporar estrategias para asegurar que el codigo sea desarrollado de manera fluida. Se va a crear una rama del codigo en el repositorio por cada historia de usuario para poder desarrollarla acordemente y una vez completada, la rama será implementada a la rama principal(main).

5.1.1. Github

Enlace del repositorio de Github del proyecto:

https://github.com/pgn3/SinLimiteUI

6. Referencias

JWT (s.f.). *Introduction to JSON Web Tokens*. https://www.jwt.io/introduction#what-is-json-web-token

Microsoft (s.f.). *Create web APIs with ASP.NET Core.* https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-10.0

Microsoft (s.f). *Overview of ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/overview?view=aspnetcore-10.0*

C. Richarson, «Microservice Architecture,» Microservices.io. Available: https://microservices.io/patterns/data/database-per-service.html.