Propuesta de una mejora en el diseño de la arquitectura del sistema de gestión de la demanda WA Collaborative donde se solventen las deficiencias actuales y se mejoren los atributos de calidad de cara al usuario final

José Lover Daza Rojas
Efraín Trujillo Malaver
Walter Antonio Morales

Trabajo de grado presentado como requisito para optar al título de:
Especialista en Ingeniería de Software

Asesor(es)
Edwin Andrés Cubillos Vega

Instituto Tecnológico Metropolitano - ITM
Facultad de Ingenierías
Departamento Antioquía
Medellín, Colombia
2024

RESUMEN

En la sociedad de la información el software se ha hecho cada vez más imprescindible para todo tipo de actividades. De ahí se deriva una industria altamente competitiva que busca brindar a los consumidores productos que no solo cumplan con sus expectativas, sino que las superen. Esto genera una evolución constante de las tecnologías digitales y obliga a las empresas a mantenerse a la vanguardia tecnológica para permanecer relevantes en el mercado. Implementar buenas prácticas en el desarrollo del software es una estrategia esencial que permite a las empresas innovar, mejorar continuamente la experiencia del usuario y, con esto, asegurar una ventaja competitiva sostenible.

Aquellas empresas que dejan de lado estas necesidades corren el riesgo de quedarse en la obsolescencia, perdiendo espacio en el mercado, ya que los consumidores modernos buscan soluciones intuitivas, seguras y eficientes que se logran con un compromiso con la mejora continua.

Este enfoque es particularmente relevante para WA Solutions, empresa de consultaría en la planeación de la demanda, que con el desarrollo del software WaCollaborative, se enfrenta a deficiencias técnicas de desarrollo en su implementación, que comprometen la escalabilidad, mantenibilidad, la eficiencia en el desempeño, la seguridad, afectando la experiencia del usuario final y la adaptabilidad a las necesidades cambiantes del mercado.

Por tanto, se genera la necesidad de realizar una nueva arquitectura que busque superar las deficiencias técnicas de su software y desarrollar una aplicación móvil como complemento a una aplicación web, para mejorar la accesibilidad y flexibilidad de sus clientes, entendiendo que la movilidad es una necesidad en el proceso de planeación de la demanda. Este documento detalla los objetivos planteados y el proceso para abordar estos desafíos técnicos.

***Palabras clave*:** Aplicación Móvil, Arquitectura de Software, Planeación de la Demanda, Proceso Colaborativo.

RECONOCIMIENTOS

Agradecemos a Dios, Nuestra Luz en nuestra vida que guía nuestros caminos; agradecemos la formación en el ITM resultado del esfuerzo en adquirir conocimiento para ser aplicado en nuestra vida laboral, que engrandezca cada actividad que realicemos en el diseño e implementación de software, labor que hemos escogió para toda la vida. Gracias a todos los docentes en la que nos aportaron sus conocimientos a través de las clases y trabajos generados, en especial a Paola Noreña, Grissa Maturana, Juan Zuluaga, Juan Vallejo y Edwin Cubillos por su disposición a la solución de dudas y cordialidad.

Agradecemos a nuestros familiares pilares de nuestras vidas, que nos apoyan de una forma especializada en ITM, y nos da la fuerza necesaria para llevar a cabo nuestras metas de vida profesional en la evolución académica que aspiramos a cumplir realizando estudios superiores, mejorando los conocimientos y habilidades logrados en nuestros respectivos pregrados.

ACRÓNIMOS

DDMRP Es una metodología para realizar la planificación de necesidades de material o productos.

S&OP Sales and Operations Planning, o Planificación de Ventas y Operaciones, es un proceso integral de gestión empresarial que alinea los planes de ventas con las operaciones de producción para asegurar que las actividades de la empresa estén en sintonía con la demanda del mercado.

SMBD Sistema Manejador de Bases de Datos, permite gestionar la creación, modificar, eliminación de objetos de base de datos.

API Rest Aplicación desarrollada para intercambiar información a través de internet, donde se utiliza el protocolo HTTP para enviar y recibir datos.

Aplicación Móvil Aplicación desarrollada para que su ejecución sea en dispositivos móviles o Tablets.

Aplicación Web Aplicación desarrollada para que su ejecución sea en servidor web para visualización de información en Navegadores Web.

SQL SERVER Es un software para el almacenamiento de datos desarrollada y mantenida por Microsoft.

UNIT TEST Es una porción de código para comprobar que esté funcionando un método de forma adecuada con la simulación de datos precargados en base de datos en memoria o entornos de base de datos en tiempo real.

BUG Palabra que suele usarse para indicar un error de un programa informático, aplicación web, aplicación móvil.

CQRS Command Query Responsibility Segregation, es un patrón arquitectónico que separa las responsabilidades de lectura y escritura de una aplicación

TABLA DE CONTENIDO

Listado de tablas

Listado de figuras

1.INTRODUCCIÓN

WA Solutions es una compañía de consultoría y de desarrollo de soluciones informáticas enfocada en la metodología Demand Driven MRP (DDMRP) con énfasis en la gestión de la cadena de suministros en los entornos operacionales.

La compañía es propietaria del software WaCollaborative, el cual es una aplicación web que está implementada en varios de sus clientes para gestionar la planeación de la demanda siguiendo la metodología S&OP (Sales And Operation Planning).

Por parte de la empresa se requiere para su software una mejora en la arquitectura, ya que presenta deficiencias técnicas que impactan negativamente la mantenibilidad, la portabilidad, la eficiencia de desempeño, la escalabilidad, la seguridad y la usabilidad de cara al usuario final; razón por la que el sistema no puede adaptarse a las necesidades cambiantes de actuales clientes y mercado, lo que genera mala experiencia de usuario.

Reconociendo la importancia de la movilidad en la gestión de la demanda, la empresa ha notado la limitación que representa la ausencia de una aplicación móvil. Actualmente, los usuarios solo pueden acceder al sistema a través de dispositivos de escritorio o portátiles, lo que restringe la experiencia fuera de la oficina o el lugar de trabajo.

Con una aplicación móvil que funcione en conjunto con la aplicación web, WA Solutions ofrecerá a sus clientes una mejor experiencia de usuario, dando mayor flexibilidad al permitirle acceder a la plataforma de planeación de demanda desde cualquier momento y lugar. Aunque desde un dispositivo móvil se puede conectar a una aplicación web, la vista no está optimizada para este tipo de dispositivos. Con la aplicación móvil se tiene una interfaz diseñada específicamente para dispositivos móviles, aprovechando funciones táctiles, notificaciones y otras características que mejoran la usabilidad y la interacción con el usuario.

Este documento busca mostrar el desarrollo de los objetivos planteados, con lo que la compañía solucionará las necesidades técnicas con respecto al software que ofrece a sus clientes.

Para ello se usan técnicas de elicitación de requisitos para identificar las necesidades del cliente para dar una solución óptima; se implementa SCRUM como metodología ágil en la gestión de proyectos de software; se implementan buenas prácticas en el desarrollo de software como código limpio; y se aplican de patrones de diseño estratégicos para una arquitectura que supere las deficiencias técnicas identificadas.

1.1. Objetivos

1.1.1. General

Proponer una mejora en el diseño de la arquitectura del sistema de gestión de la demanda WA Collaborative por medio de la cual se solventen las deficiencias actuales y se mejoren los atributos de calidad de cara al usuario final.

1.1.2. Específicos

- Identificar las necesidades y requerimientos que tiene el software de gestión de la demanda WA Collaborative para alcanzar el nivel de satisfacción de sus clientes.
- Diseñar una arquitectura que asegure la mantenibilidad, portabilidad, eficiencia, escalabilidad, seguridad y usabilidad del software WA Collaborative.
- Proponer con la aplicación de patrones de desarrollo de software una mejora en la arquitectura.
- Elaborar prototipos móvil y web basado en las mejoras propuestas para el software WA Collaborative aplicando la Arquitectura diseñada.

1.2. Estructura del documento

El documento se estructura en 5 capítulos, capítulo 1, introducción, en este aparte se abordan los objetivos del proyecto; capítulo 2, marco teórico, aquí se da una noción al lector sobre los temas principales que abordara el documento; capítulo 3, metodología, en este capítulo se mencionarán maneta detallada todas las fases del proyecto, desde las técnicas de elicitación de requisitos usadas hasta el diseño de la solución, abordan etapas como la documentación de los diferentes diagramas, el diseño de la arquitectura del software, y los diferentes roles y artefactos usados.

2. MARCO TEÓRICO

2.1. Antecedentes de arquitectura de software

El uso del concepto de arquitectura de software es relativamente nuevo, desde los 60 se debe generar una estructura para construir sistemas de información antes de que se escriban las primeras líneas de código, luego en los 70 hasta los 90 solo se mencionaba la especificación detallada del software junto a su interfaz de usuario, ya para el final del siglo XX y comienzo del siglo XXI se inician técnicas, lineamientos, reglas, estrategias y estándares para elaborar arquitecturas de software creando Lenguajes Descriptivos (ADL).

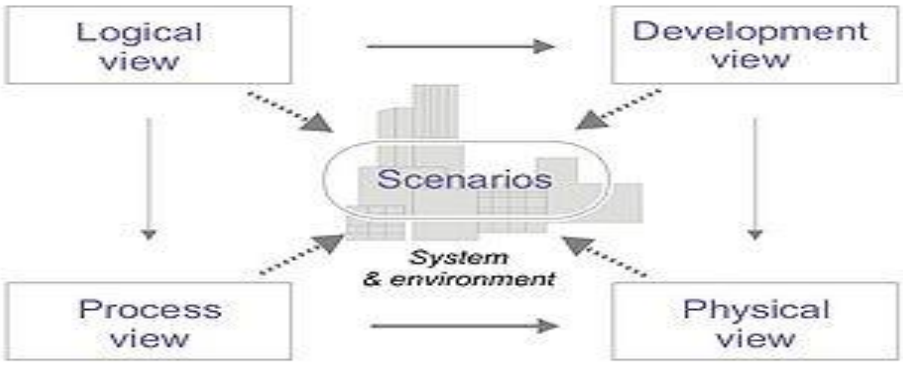
En la arquitectura de software se tiene tres visiones principales para tener en cuenta, las cuales son:

La Visión Estática: Se enfoca en describir los componentes.
La Visión Funcional: Se enfoca en describir lo que realiza cada componente.
La Visión Dinámica: Se enfoca en describir cómo se comportan los componentes a lo largo del tiempo.

El enfoque más utilizado es realizar una definición de una arquitectura de software mediante el modelo de Vistas de Arquitectura 4+1, estas vistas son:

Vista de Lógica: Se basa en realizar la descripción de la estructura y funcionalidad de un software.
Vista de Desarrollo: Se basa en la descripción de los paquetes y/o librerías que se utilizarán en los diferentes artefactos de un software.
Vista de Proceso: Se enfoca en el comportamiento del sistema en tiempo de ejecución, mostrar cómo es la comunicación de todos los componentes.
Vista Física: Se basa en mostrar la capa física donde se ejecuta cada componente, donde se indica el hardware necesario para su funcionamiento y ejecución.
Escenarios: Se enfoca en realizar las historias de uso, casos de uso y en general toda la especificación de las funcionalidades del software.

Figura 1. Vistas conceptuales de un Software



Fuente: Modelo de Vista 4 + 1 (2023) Modelo de Vistas de Arquitectura 4+1, obtenido [en línea] el 15 de febrero de 2025 en https://es.wikipedia.org/wiki/Modelo_de_Vistas_de_Arquitectura_4%2B1

2.2. Uso de software para planeación de la demanda

Las empresas interesadas en el software necesitan contar con un medio para la realización de la planificación a la demanda, dentro de la aplicación se permite que los colaboradores ingresen información y con esta se tome decisiones en los procesos relacionados. Esta solución informática permite la mejora de la comunicación de los colaboradores, estimula el trabajo colaborativo, mejora la toma de decisiones a partir de la información ingresada. Dentro de la industria existen diferentes programas de computador relacionados, los cuales ya fueron mencionados y definidos, y tienen un enfoque colaborativo para gestionar procesos dentro de una organización, algunos de ellos son Streamline, Skus cience, Forecast Pro, Datup, SAP APO, entre otros.

2.3. Calidad en el proceso de desarrollo de software

La calidad dentro del software, indica el grado que este satisface los requisitos o necesidades de sus usuarios. Su implementación aporta valor y forma de diferenciarse entre similares, la industria en apoyo de la academia y de lecciones aprendidas ha generado un modelo de calidad llamado ISO 25010 desde su aparición y definición en el año 2005, como un marco para definir atributos que debe tener un aplicativo como funcionalidad,

rendimiento, seguridad, mantenibilidad, etc. La ISO 25010 es parte de la familia de la ISO 25000, la cual es el resultado de la evolución de otras normas anteriores formuladas, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto informático y ISO/IEC 14598 que abordaba el proceso de evaluación de productos.

Las normas ISO las formularon o generó la Organización Internacional de Normalización, que aparecen según se ha estandarizado técnicamente la industria internacionalmente.

Figura 2. Calidad del Producto Software

Fuente: Platzi (2020) Atributos de Calidad, obtenido [en línea] el 15 de abril de 2024 en https://platzi.com/tutoriales/1248-pro-arquitectura/5498-atributos-de-calidad-de-un-producto-de-software/

Por tanto, como forma de diferenciación para Wa Collaborative, es importante que se haga énfasis en la mejora de los atributos de calidad de su aplicativo al proponer una arquitectura de software, al formularse gran cantidad de atributos, se van a seleccionar los más adecuados a mejorar, logrando aplicarlos a la definición de la arquitectura, los que se desean mejorar son los siguientes:

Seguridad: Es el atributo relacionado a la seguridad del software, donde solo se permite que un usuario tenga acceso mediante de un usuario y contraseña, cada módulo, petición y funcionalidad se valida que tenga autorización. La información se mantendrá protegida del acceso a personas externas al aplicativo, garantizar el rastreo de las consultas, creación y actualización de datos dentro del sistema almacenado logs de transacciones.

Mantenibilidad: Es el atributo enfocado a lograr que el software permita incrementar nuevas funcionalidades gracias a la fácil comprensión del código realizado por cualquier programador que haya participado o no en el proyecto. La reutilización de código permite que cualquier modificación sea efectiva y eficiente sin generar defectos o mal desempeño del aplicativo que se está modificando o extendiendo.

Portabilidad: El propósito de este atributo es la facilidad de instalación en diferentes entornos de ejecución sin afectar la funcionalidad y desempeño del producto.

Usabilidad: Es el atributo relacionado con un software fácil de aprender y usar, el usuario solo debe conocer el contexto para comprender cada funcionalidad. Cada módulo y formulario guía al usuario para su uso, con mensajes guía, componentes gráficos correctos, interfaces amigables e intuitivas.

Eficiencia: Es el atributo que relaciona el uso óptimo de recursos de hardware, para lo cual se debe utilizar programación con código limpio, código estructurado, con utilización de estándares de programación, documentar el código, diseñar correctamente cada componente dentro de una arquitectura limpia.

Escalabilidad: Este atributo hace referencia a que el software se pueda adaptar y dar respuesta con buen rendimiento a medida que este atiende a más cantidad de usuarios. La escalabilidad se puede realizar verticalmente o escalar al colocar más cantidad de hardware en una misma infraestructura o horizontal, que indica colocar más nodos o servidores e instalarse en estos para tener más instancias del software distribuidas en varios nodos o servidores dependiendo de la capacidad de compra de una empresa.

2.4. Acercamiento a la tecnología del software WaCollaborative

El software WA Collaborative se desarrolla utilizando las tecnologías más recientes, para la lógica de negocio Backend donde se organiza dentro de un proyecto en un estilo orientado a servicios, puesto se expondrán servicios API Rest para que sean consumidos por un proyecto de presentación de información Frontend realizado en Flutter como aplicación móvil, también se elabora un prototipo de apoyo una aplicación web en Blazor para realizar configuraciones de módulos, estos prototipos son realizados con el enfoque de usabilidad de las pantallas y/o páginas, de esta manera mantener separadas las responsabilidades de cada proyecto, se hace uso de .Net versión 7 como entorno de ejecución y el lenguaje de programación elegido es C#, como se ha indicado anteriormente es un entorno de una única plataforma que la empresa ya usa.

Para la base de datos se eligió SQL Server en su versión más reciente, una base de datos relacional que se puede gestionar con el software SQL Server Management Studio como SMBD, sirve para visualizar cada objeto de la base de datos, que indica que las tablas se relacionan con relaciones definidas para crear un modelo para almacenar datos. Como se indica, se realizan diferentes proyectos y capas para la implementación, pero se aplica diferentes técnicas y patrones de desarrollo de software, técnicas que se aplican dependiendo de sus características.

Para el diseño de la arquitectura, además de los atributos de calidad, se considera patrones de desarrollo de software, término manejado desde finales del siglo XIX, que consiste en aplicar soluciones planteadas a problemas comunes al implementar software y útiles si se usan en las situaciones adecuadas y por las razones correctas.

A continuación, se ilustra los patrones que se utiliza como referencia para la formulación de la arquitectura para el software Wa Collaborative:

Patrón Única Instancia de Objeto (Singleton): Este patrón indica que al usarlo e implementarlo, los objectos de una clase solo se permiten que se tenga una única instancia durante la ejecución de un software, este es aplicado mayormente para la creación del objeto de conexión a base de datos, para que este sea un único objeto y que se mantenga durante la ejecución.

Patrón Repositorio (Repository): Este patrón indica que se crea una clase repositorio asociado a una entidad o en forma genérica con los métodos que solo van a interactuar con la base de datos, aislando dentro de una capa la comunicación de la base de datos con el resto de una aplicación, dando seguridad e integridad a un software.

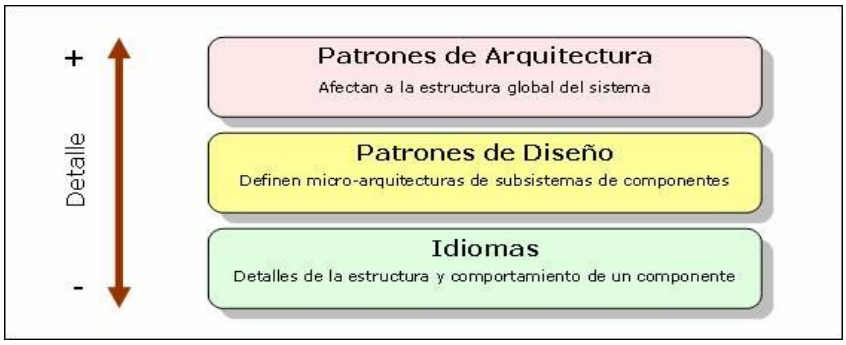
Patrón Unidad de Trabajo (Unit of Work): Este patrón hace referencia a que se crea una

clase como una unidad de trabajo, la cual realiza las peticiones o transacciones de una aplicación a la base de datos de forma que cuando estas se completan, se realiza la confirmación de las transacciones o acción que tiene como nombre Commit (confirmar o aprobar transacción), de lo contrario al generarse un error, se produce una eliminación de las transacciones en cadenas generadas, acción que tiene como nombre Rollback (Cancelar o eliminar transacción).

Patrón Adaptador (Adapter): Este patrón permite solucionar la incompatibilidad entre clases, es decir, que dos clases diferentes trabajen juntas mediante la homologación de campos y tipos. Este es utilizado cuando se tienen servicios web api en proyectos como componentes de software y aplicaciones de presentación escritas en diferentes lenguajes de marqueteado web.

Patrón Fachada (Facade): Este patrón se enfoca a que se realice la definición de los métodos necesarios para una entidad en interfaces y crear una clase que a través de ella se realice la implementación de los métodos definidos, esto sugiere tener el medio para poder implementar funcionalidades o métodos de una clase. Esto permite que se pueda implementar en otra entidad que necesite realizar el uso de las clases generando una simplicidad.

Figura 3. Patrones en el desarrollo de Software



Fuente: Patrones de Software (2014) Tópicos generales de Ingeniería de Software, obtenido [en línea] el 19 de marzo de 2024 en https://ingsoftwarei2014.wordpress.com/2014/05/28/patrones-de-software/

3. METODOLOGÍA

3.1. Generalidades

Para el desarrollo de un proyecto de software, se debe tener un código limpio donde se genera una producción de código que supla una necesidad y que sea entendible para cualquier ingeniero de software, basándose en la programación orientada a objetos, se tiene como principio de alta cohesión entre clases donde se le da una única tarea a una clase, tiene atributos y comportamiento especifico y coherente con el nombre de esta. Como otro principio se tiene el acoplamiento, indicando la dependencia o independiente que cada clase tiene de otras.

Para el diseño de software, se tiene un lenguaje denominado UML (Lenguaje Unificado de Modelado), Karoly Nyisztor, K.; Nyisztor, M. (2018) nos indica la importancia de familiarizarse en el diseño de un sistema a través de diferentes diagramas que sean entendibles por ingenieros en un único lenguaje que transmita cada detalle pensado por el autor de cada diagrama. UML permite diseñar y mostrar cómo está compuesta una arquitectura de software, indicar el comportamiento e interacción de cada clase, entre otros comportamientos diseñados de la interacción de un sistema mediante diagramas que se pueden crear.

Al diseñar una arquitectura de software permite que se entienda el comportamiento y comunicación entre componentes, además de que sus componentes sean mantenibles y que se puedan realizar pruebas independientes a cada componente o artefacto que la componen. Una de las arquitecturas más utilizadas recientemente es la arquitectura por microservicios (Backend), la cual permite separar la lógica y la vista o presentación en diferentes proyectos, es conformada por un proyecto que contiene los microservicios como componentes de una aplicación que ejecuta independientes a través de una interfaz denominada API Rest, cada microservicio realiza una tarea especifica e independiente de cada uno. En la industria existe varias tecnologías para realizar proyectos de microservicios como por ejemplo Spring Boot con programación en Java o Kotlin, Web Api de Microsoft con lenguaje de programación en C# entre otras tecnologías. Newman, S. (2021) indica que las organizaciones al pasar de aplicaciones monolíticas a microservicios más pequeños e independientes, los sistemas distribuidos se han vuelto más detallados.

Para la construcción de la capa de presentación donde se captura o se muestra la información, se construye para que consuma peticiones a los microservicios, realizando una petición y esperando el resultado de cada una. Para proyecto en tecnologías móviles existe varios lenguajes de programación como Flutter, Kotlin, Swift, entre otros para generar aplicaciones que se instalan y se ejecuta en entorno móvil, también en la industria existe varias tecnologías para realizar proyectos de presentación (Frontend) como Angular, Blazor,

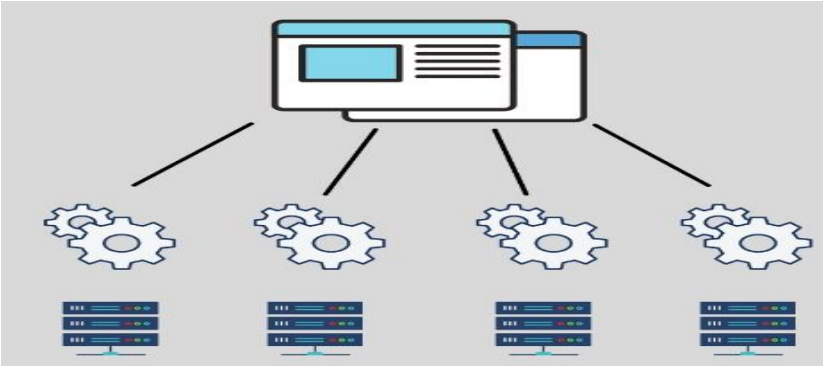
Razor, Ionic con programación en HTML, Typescript, Javascript, CSS entre otras tecnologías.

Como componentes principales están los servidores de infraestructura, donde se instalan los anteriores componentes de API Rest generados para el proyecto, que tiene la tarea del procesamiento aplicaciones, estos son maquinas robustas con procesadores y alta capacidad de memoria para instalar cualquier tipo de software. Dentro de cada servidor se puede instalar un software denominado servidor de aplicaciones, los cuales contienen un entorno de ejecución con todos los programas donde se pueda ejecutar proyectos de microservicios o presentación, dentro de la industria se tiene como por ejemplo IIS, Xamp, JBoss entre otros servidores de aplicaciones.

Para almacenar datos se tienen los motores de base de datos, diseñados para estar siempre disponibles para entregar información de las peticiones de datos que le llegan, además de insertar, actualizar o eliminar datos almacenados. En la industria existe varias tecnologías para realizar almacenamiento de datos como Oracle DB, SQL Server, PostgreSQL entre otros motores de base datos.

Es útil conocer los diferentes artefactos o componentes que puede componerse una arquitectura, para lograr generar una estructura que satisfaga una necesidad en específico, el presente proyecto al basarse en la generación de una nueva arquitectura para la aplicación WA Collaborative que solucione el problema de obsolescencia de la tecnológica utilizada, se estudia las diferentes metodologías que se tiene disponible, donde se elige utilizar basada en microservicios dividiendo en dos proyectos, un proyecto de Microservicios realizados en C# Api Rest como Backend y un Aplicativo móvil como capa de presentación donde se realiza en un proyecto en Flutter como Frontend.

Figura 4. Arquitectura Microservicios (Backend - Frontend)



Fuente: Arrobasolutions (2023) Arquitectura de Microservicios, obtenido [en línea] el 11 de enero de 2024 en https://www.arrobasolutions.com/arquitectura-de-microservicios-que-es-y-cuales-son-sus-ventajas/

3.2. Análisis y especificación de requerimientos

3.2.1. Técnicas de elicitación de requerimientos

3.2.1.1. Entrevista 1

Conociendo al Stackeholder

Nombre del entrevistado: Walter Morales Gallego

Rol en la empresa: Líder técnico para el proyecto WA Collaborative

Conociendo la empresa

Nombre de la empresa: WA Solutions

A qué se dedica la empresa: WA Solutions es una compañía de consultoría y de desarrollo de soluciones informáticas enfocada en la metodología Demand Driven MRP (DDMRP) con énfasis en la gestión del flujo en los entornos de operacionales

Conociendo el software actual

Nombre del software actual: Wa Collaborative

¿Qué hace el software actual? : El software WaCollaborative es una herramienta web que permite al departamento comercial de nuestros clientes gestionar la planeación de su demanda en los lugares donde ellos operan, el proceso se hace de forma colaborativa, es decir el departamento comercial tiene diferentes roles como ejecutivos de ventas, gerentes regionales, gerentes nacionales, gerentes de marca, directores comerciales, y cada uno de ellos participa en la herramienta registrando de forma colaborativa las cantidades que ellos creen que van vender a los diferentes clientes del portafolio que ellos atienden.

¿En promedio cuántos clientes usan el software?: Actualmente tenemos 5 empresas que operan en América Latina.

¿En cuáles sectores económicos se usan el software?: puede ser cualquier sector económico que maneje una cadena de suministro y requiera planear la demanda lo cual le ayuda también.

¿Cuáles problemas ha identificado en el software actual?: Actualmente gran parte de la lógica de negocio que se encuentra en la base de datos genera un hueco de seguridad para la empresa ya que terceros podrían apropiarse fácilmente del conocimiento de cómo funciona el negocio, por lo cual el diseño planteará implementar la lógica de negocio en el Backend de la solución.

La solución está desarrollada bajo .NET Framework 4.5.2 el cual ya dejó de ser soportado por Microsoft propietario del Framework, lo que implica que por parte del proveedor no se recibirá más soporte para esta versión, ni habrá actualizaciones, por lo tanto, en caso de ocurrir algún error en el Framework no será posible obtener

una solución, lo que puede dejar el software fuera de servicio generando pérdidas para empresa al no cumplir con los contratos con los clientes.

Actualmente el software no tiene la posibilidad de ser escalado horizontal ni verticalmente para que pueda soportar muchas peticiones de los usuarios generando demoras en los tiempos de respuesta.

Conociendo los deseos del cliente frente al módulo que quiere migrar a una nueva arquitectura.

¿Cómo se llama el software que se quiere migrar?: se llama WA Collaborative.

¿Describa qué hace el software que se quiere migrar? : Permite a los usuarios de los diferentes roles de la estructura comercial registrar la demanda que piensan que van a tener los clientes que ellos atienden en un horizonte de tiempo determinado.

¿Mencione paso a paso el proceso que realiza un usuario dentro del sistema? Antes de iniciar el proceso, un usuario planeador de demanda ingresa al sistema para configurar la información necesaria para realizar un ciclo de colaboración durante un mes, las configuraciones son, registrar las fechas para cada papel en que ingresen en el mes a registrar las cantidades que corresponden a su colaboración; asignar el portafolio de clientes y productos que gestionará cada usuario.

Luego al iniciar el ciclo de colaboración, que se lleva a cabo durante el mes, cada usuario colaborador ingresa al sistema durante las fechas asignadas a registrar sus cantidades que proyecta vender, es decir, su colaboración para cada cliente durante los próximos meses. El usuario planeador también puede tener un portafolio asignado y registrar colaboraciones.

Al finalizar el ciclo de colaboración el equipo comercial tiene una reunión donde revisan el plan de demanda desde la herramienta para llegar a un consenso acerca de si la demanda que se está proyectando es con la que van a trabajar, si es así la aprueban, y si no pueden hacer modificaciones al plan, lo que llevaría a modificar las cantidades que ellos consideran según el consenso.

3.2.1.2. Entrevista 2

Introducción: En la primera entrevista hablamos de lo que hace el software, del proceso, pero en esta oportunidad vamos a detallar la información que es necesario registrar en el software, para esto le pedimos que nos diga puntualmente que información se registra para cada una de las siguientes cosas del software:
¿Cuál información se registra para la demanda colaborada?:

- Cantidad
- Fecha
- Producto
- Cliente

- Grupo de colaboración
- Tipo de demanda
- Ciclo de colaboración

¿Cuál información se registra para un producto?:

- Código
- Descripción
- Unidad de medida
- Factor de conversión

¿Cuál información se registra para el cliente?:

- Código
- Descripción
- Secuencia

¿Cuál información se registra para un grupo de colaboración?:

- Código
- Descripción

¿Cuál información se registra para un tipo de demanda?:

- Descripción
- Tipo de evento

¿Cuál información se registra para un tipo de evento?:

- Descripción

¿Cuál información se registra para un ciclo de colaboración?:

- Fecha inicial
- Fecha final

¿Cuál información se registra para el plan de demanda?:

- Fecha
- Mes
- Cantidad
- El plan de demanda tiene un estado que es en revisión y aprobado

¿Cuál información se registra para el portafolio?:

- Producto
- Cliente
- Rol

¿Cuál información se registra para los roles?:

- Nombre

¿Cuál información tiene un usuario?:

- Correo electrónico
- Nombre
- Contraseña

3.2.2. Reglas de negocio

Mediante la siguiente tabla se listan las reglas de negocio:

Tabla 1. Reglas de Negocio

Código	Nombre	Descripción	Formula	Fuente	Regla de negocio relacionada
BR001	Registrar, revisar y aprobar demanda colaborada	Solo se puede registrar y revisar la demanda colaborada cuando se encuentra en estado "En revisión".	DemandaColaborada. Estado == "En revisión"	Stakeholder	
BR002	Generar plan de demanda	EL plan de demanda solo se puede generar cuando la demanda colaborada ha sido aprobada, eso quiere decir que están en estado "Aprobado".	DemandaColaborada. Estado == "Aprobado"	Stakeholder	
BR003	Visualizar y descargar demanda colaborada	La demanda colaborada se puede visualizar y descargar en cualquier estado, lo puede hacer el usuario colaborador y el usuario planeador.		Stakeholder	

BR004	Usuario planeador	El usuario planeador es el único usuario que puede: gestionar usuarios planeadores y colaboradores, asignar roles, administrar clientes, administrar productos, administrar calendario de colaboración y administrar portafolio, también tiene los mismos permisos de un usuario colaborador.		Stakeholder	
BR005	Registrar demanda	la demanda colaborada solo se puede registrar dentro del tiempo estipulado en el calendario de colaboración.	Año, mes y día de calendario de colaboración <= año, mes y día actual	Stakeholder	BR001 BR006
BR006	Visualizar demanda únicamente de portafoli	Un usuario colaborador solo puede visualizar la demanda colaborada de clientes y productos que pertenecen a su		Stakeholder	BR001 BR005

	o asociado	portafolio y que estén en un calendario de colaboración con un año, día y mes superior al año, día y mes actual			
BR007	Usuario colaborador	El usuario colaborador únicamente puede: registrar, revisar, aprobar, visualizar y exportar la demanda colaborada.		Stakeholder	BR001 BR002 BR005 BR006 [1]

Fuente: propia

3.2.3. Diccionario de datos

A continuación, se muestra el diccionario de datos:

Tabla 2. Diccionario de datos

Nombre	Alias	Tipo	Proceso involucrado	Característica
A:0 Usuario planeador		Agente externo	P1: Crear usuario Planeador, P2: Convierte en colaborador, P3: Activa usuario, P4: Inactiva usuario, P5: Edita usuario, P24: Crear usuario colaborador, P25: Convierte en planeador, P26: Activa usuario colaborador, P27: Inactiva usuario colaborador, P28:	Id, Correo electrónico, Nombre, Rol, Ciudad, Portafolio

[1]

			Edita usuario colaborador, P6: Crea Cliente, P7: Elimina Cliente, P8: Edita Cliente, P9: Crea Producto, P10: Elimina Producto, P11: Edita Producto, P12: Crea Portafolio, P13: Elimina Portafolio, P14: Edita Portafolio, P15: Crea Calendario de colaboración, P16: Elimina Calendario de colaboración, P17: Edita Calendario de colaboración	
A:1 Usuario colaborador		Agente externo	P18: Visualiza Demanda colaborada, P19: Registra Demanda colaborada, P20: Exporta Demanda colaborada, P21: Revisa plan de demanda, P22: Aprueba plan de demanda, P23: Genera plan de demanda	Id, Correo electrónico, Nombre, Rol, Ciudad, Portafolio
B1: Rol		Bodega	P1: Crear usuario Planeador, P5: Edita usuario, P24: Crear usuario colaborador, P28: Edita usuario	Id, Nombre
B2: Usuario		Bodega	P1: Crear usuario Planeador, P2: Convierte en colaborador, P3: Activa usuario, P4: Inactiva usuario, P5: Edita usuario, P24: Crear usuario colaborador, P25: Convierte en	Id, Correo electrónico, Nombre, Rol, Ciudad, Portafolio

			planeador, P26: Activa usuario colaborador, P27: Inactiva usuario colaborador, P28: Edita usuario colaborador	
B3: Cliente		Bodega	P6: Crea Cliente, P7: Elimina Cliente, P8: Edita Cliente	Id, Código, Descripción, Punto de envío, ciudad, Canal de Distribución
B4: Producto		Bodega	P9: Crea Producto, P10: Elimina Producto, P11: Edita Producto	Id, Código, Descripción, Factor de conversión, Unidad de medida, Segmento, Categoría
B5: Portafolio		Bodega	P12: Crea Portafolio, P13: Elimina Portafolio, P14: Edita Portafolio	Id, Usuario, Producto, Cliente
B6: Calendario de colaboración		Bodega	P15: Crea calendario de colaboración, P16: Elimina Calendario de colaboración, P17: Edita Calendario de colaboración	Año, Mes, Día, Rol
B7: Demanda colaborada		Bodega	P18: Visualiza Demanda colaborada, P19: Demanda colaborada, P20: Exporta Demanda colaborada	Id, Cantidad, Fecha, Estado, Cliente, Producto, Grupo de Colaboración, Tipo de demanda, Plan de demanda
B8: Plan de demanda		Bodega	P21: Revisa plan de demanda, P22: Aprueba plan de demanda, P23: Genera plan de demanda	Id, Periodo, Estado

Fuente: propia

3.2.4. Requisitos funcionales

A continuación, se listan los requisitos funcionales de la aplicación web y móvil:

Aplicación Web

Tabla 3. Requisitos funcionales para la aplicación Web

Nombre	Descripción
Inicio de Sesión	La aplicación web debe permitir ingresar con un correo y contraseña, en caso de fallo sale un mensaje de notificación.
Gestión de Usuario	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) usuarios, con datos como nombre, rol, correo electrónico entre otros datos.
Gestión de Estados	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) estados.
Gestión de Unidades de Medidas	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) unidades.
Gestión de Países	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) países.
Gestión de Departamentos (Región)	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) departamentos.
Gestión de Ciudades	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) ciudades.
Gestión de Clientes	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) clientes.
Gestión de Canales de Distribución	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) canales de distribución.
Gestión de Segmentos	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) segmentos.

Gestión de Categorías	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) categorías.
Gestión de Tipos de Eventos	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) tipos de eventos.
Gestión de Productos	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) productos.
Gestión de Calendario de Colaboración	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) calendarios de colaboración.
Gestión de Colaboración de la Demanda	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) colaboración de la demanda.
Gestión de Portafolios	La aplicación web debe permitir que se pueda gestionar (listar, crear, editar) portafolios.
Exportar Archivo donde es enviado por Correo	La aplicación web debe permitir enviar un archivo con la demanda colaborada por correo.

Fuente: propia

Aplicación Móvil

Tabla 4. Requisitos funcionales para la aplicación Móvil

Nombre	Descripción
Registro de Usuario	La aplicación móvil debe permitir que los usuarios se registren creando una cuenta utilizando un usuario, correo electrónico, contraseña, entre otros datos.
Inicio de Sesión	La aplicación móvil debe permitir ingresar con un correo y contraseña, en caso de fallo sale un mensaje de notificación.
Filtros de Búsqueda	La aplicación móvil debe permitir tener filtro de búsqueda para encontrar y filtrar datos según sea pertinente por la funcionalidad a utilizar.

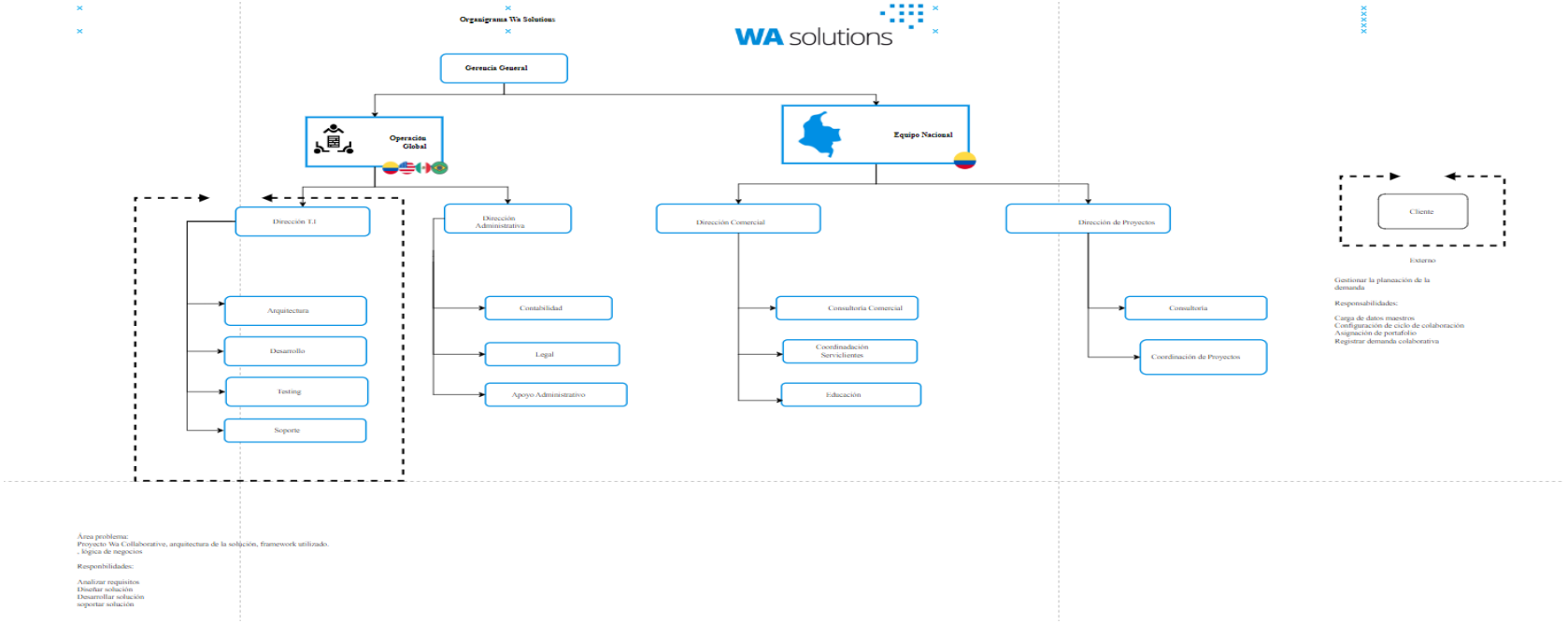
Aprobar Plan de Demanda	La aplicación móvil debe permitir que aprobar un plan de demanda, donde se cambia el estado del plan y se aprueban todas las colaboraciones realizadas.
Exportar Plan de Demanda	La aplicación móvil debe permitir que se exporte en archivo plano el plan de la demanda.
Visualizar Histórico	La aplicación móvil debe permitir la visualización de gráficos con la información histórica de venta para un determinado cliente.

Fuente: propia

3.2.5. Organigrama empresarial

El siguiente es el organigrama de la empresa WA Solutions, propietaria de software WA Collaborative

Figura 5. Organigrama Empresarial de WA Solutions

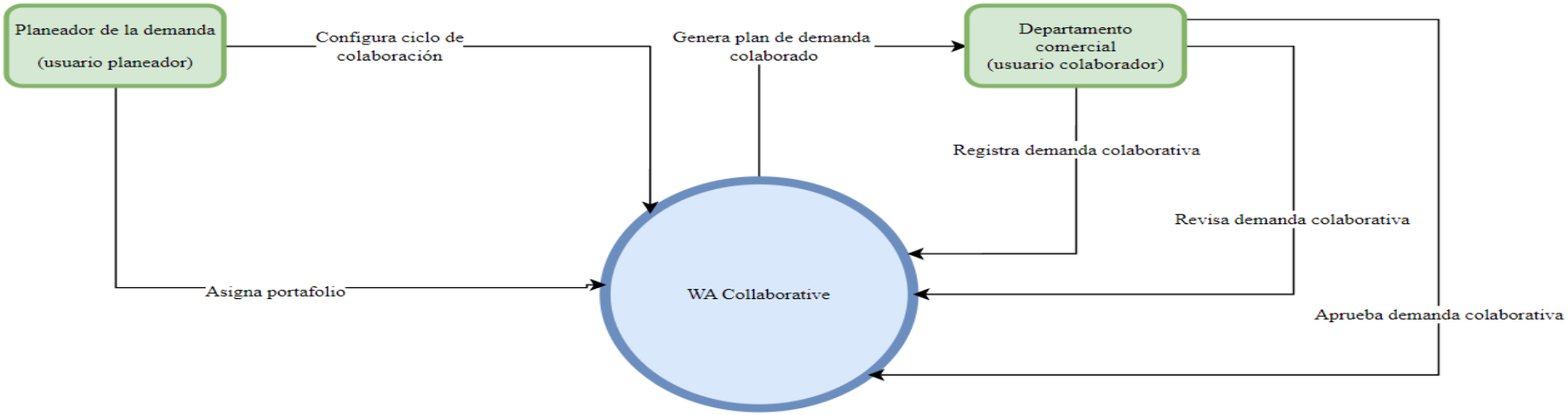


Fuente: Wa Solutions (2024) Organigrama Empresarial, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de enero de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%222oI27tSqnui_mLf_F38T%22%7D

3.2.6. Diagrama de contexto

El siguiente es el diagrama de contexto por medio del cual se muestra la interacción que tienen los usuarios con el software:

Figura 6. Diagrama del Contexto de la Planeación de la demanda en WA Solutions

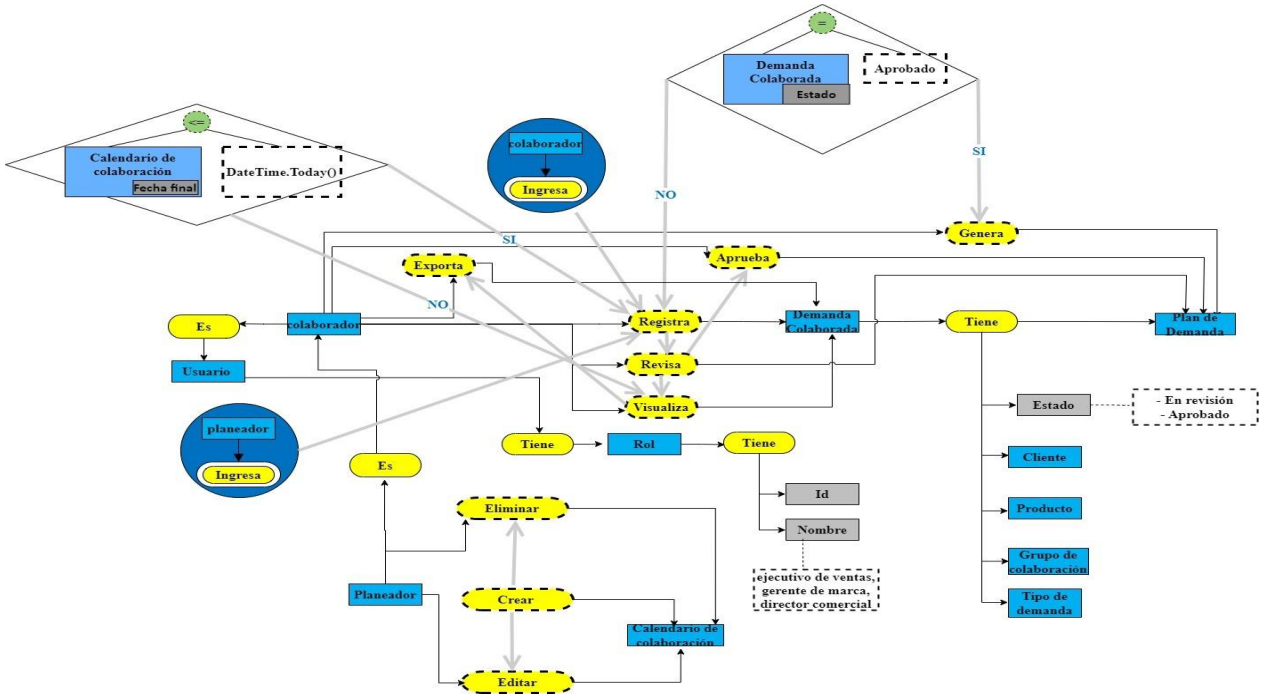


Fuente: Wa Solutions (2024) Organigrama Empresarial, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de enero de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%222oI27tSqnui_mLf_F38T%22%7D

3.2.7. Esquema preconceptual

El siguiente es el esquema preconceptual de WA Collaborative:

Figura 7. Diagrama del Esquema preconceptual de la planeación de la demanda en WA Solutions

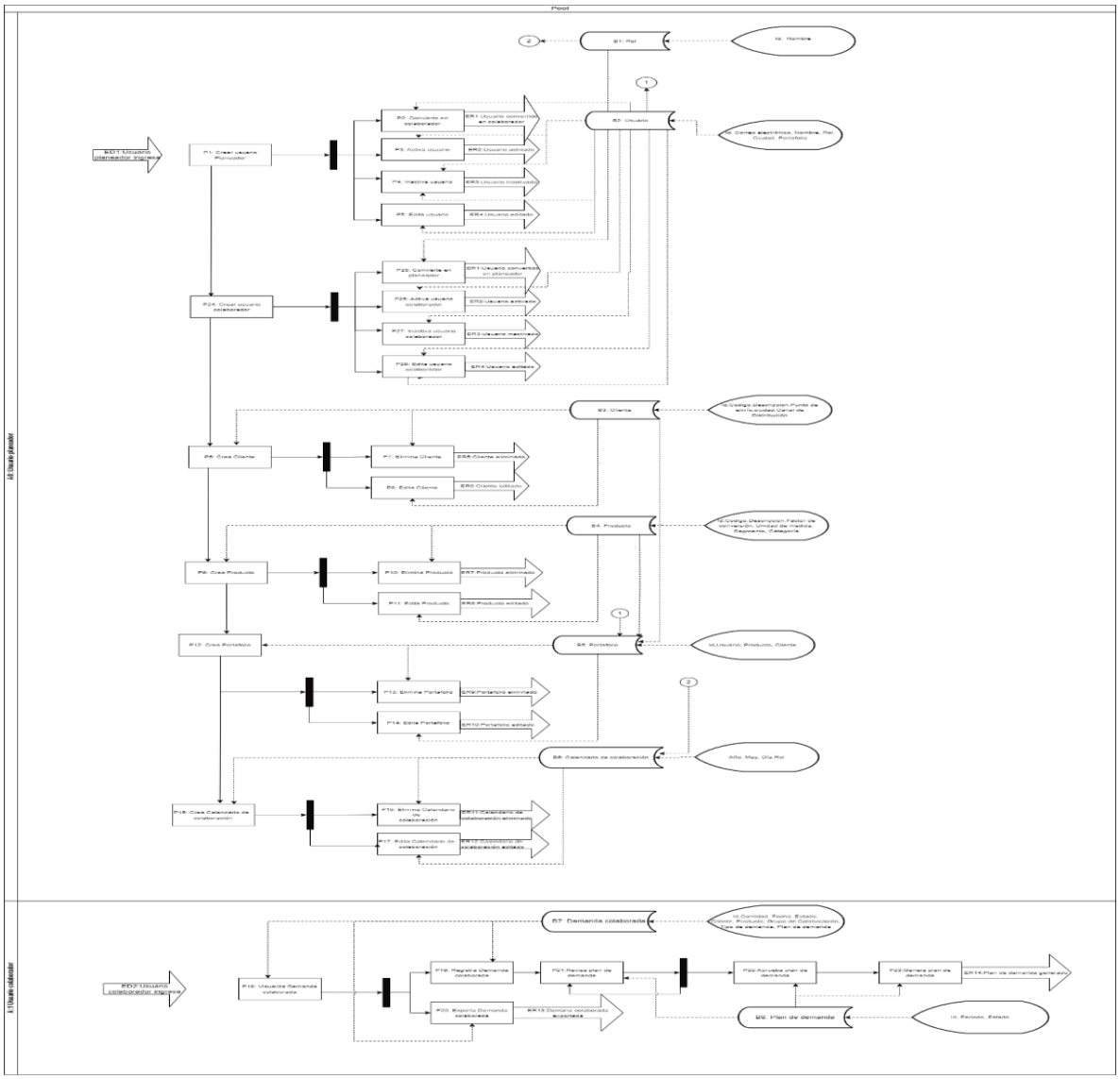


Fuente: Wa Solutions (2024) Organigrama Empresarial, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de enero de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%222oI27tSqnui_mLf_F38T%22%7D

3.2.8. Diagrama de procesos

Mediante el siguiente diagrama se describen cada uno de los procesos del sistema:

Figura 8. Diagrama del proceso del sistema

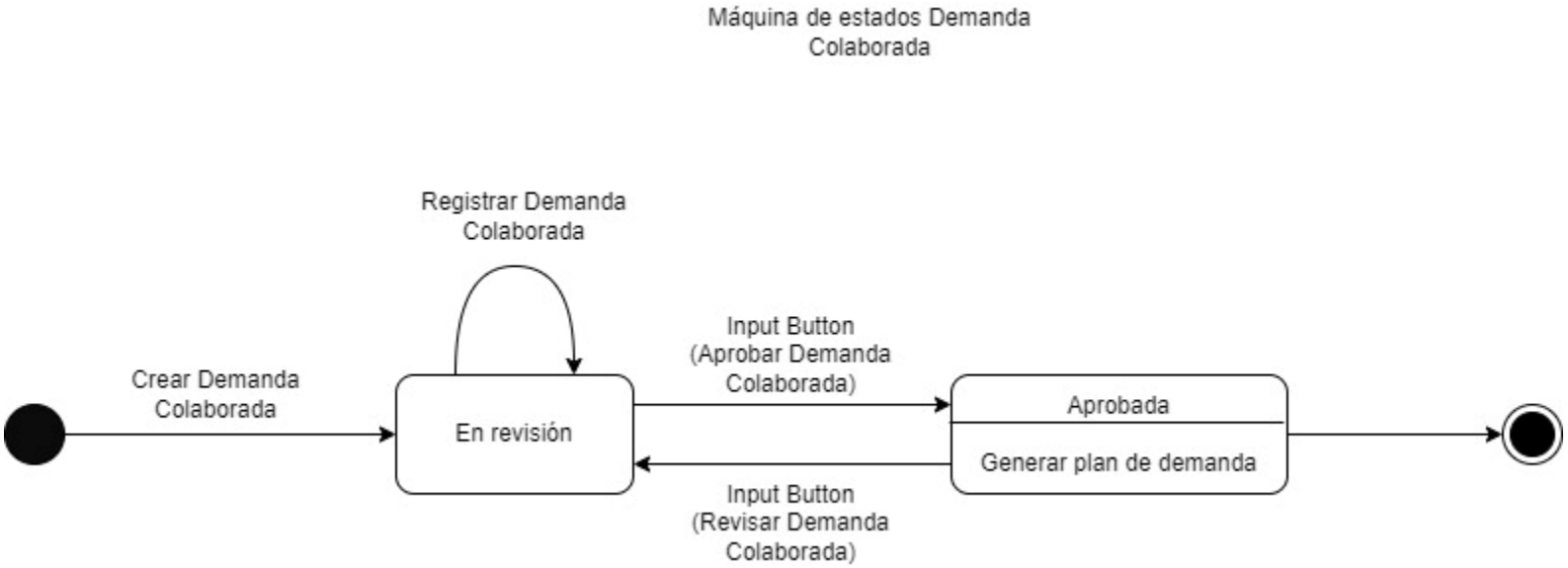


Fuente: Wa Solutions (2024) Diagrama de Procesos, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de enero de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.2.9. Máquina de estados

A continuación, se muestra el diagrama de estados:

Figura 9. Diagrama máquina de estados de la demanda colaborada

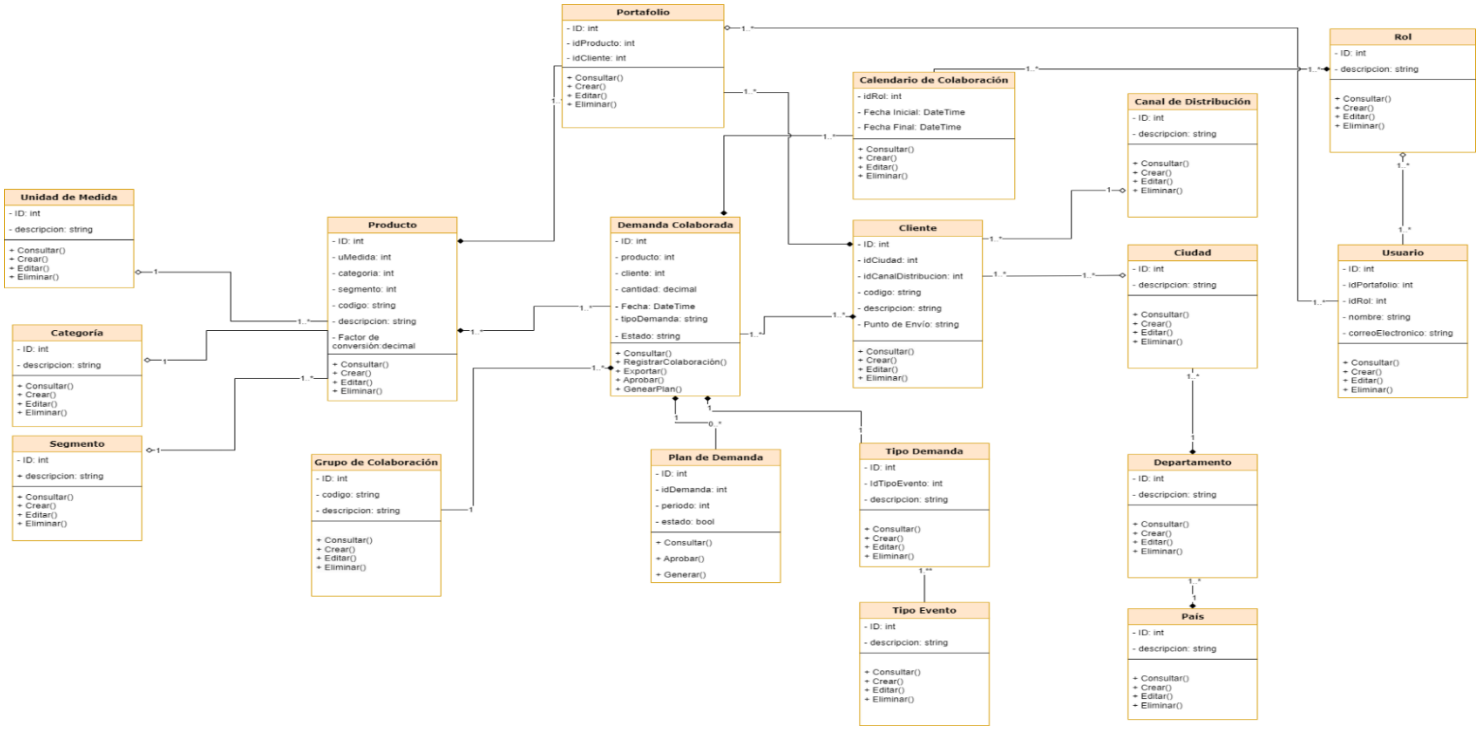


Fuente: Diagrama de Máquina de Estados, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de marzo de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.2.10. Diagrama de clases

A continuación, se muestra el diagrama de clases del software:

Figura 10. Diagrama de clases

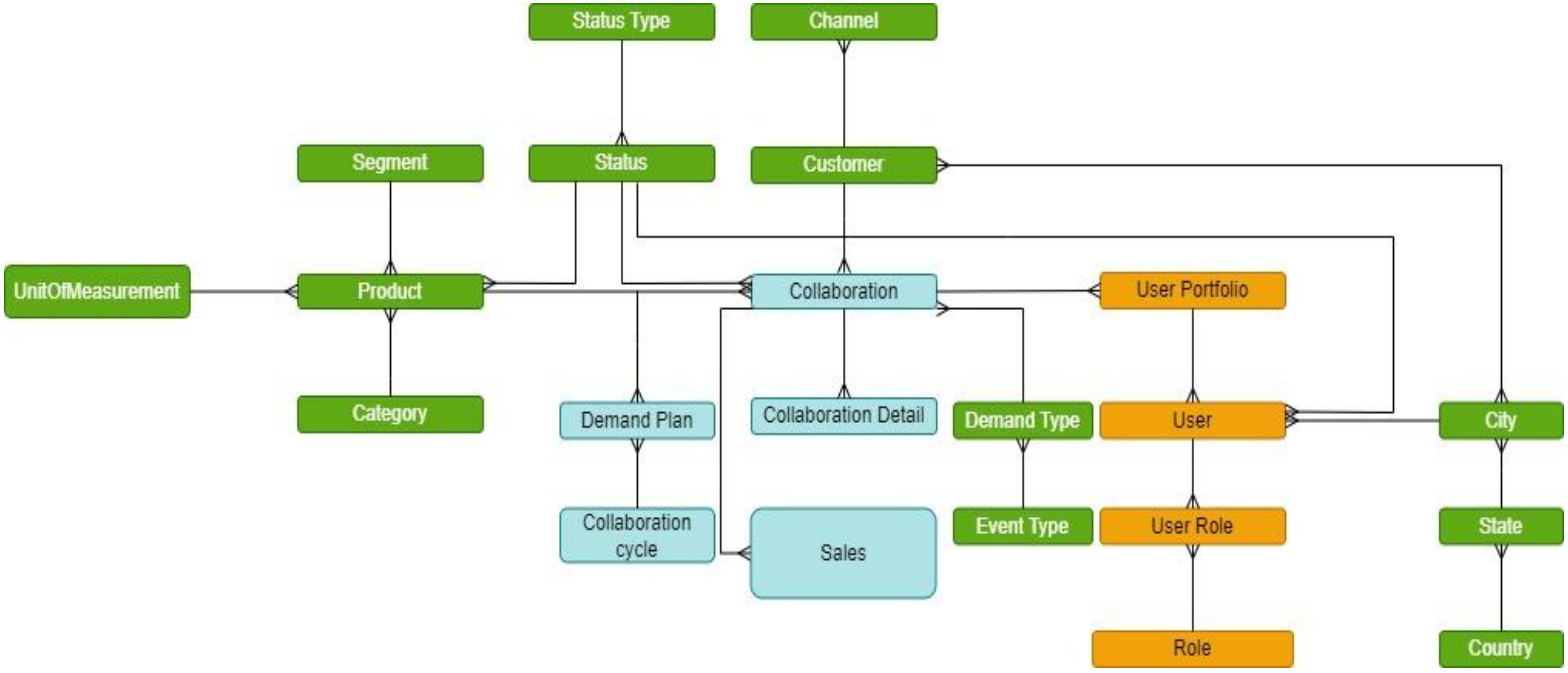


Fuente: Diagrama de Clases, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de marzo de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.2.11. Modelo entidad relación

A continuación, se muestra el modelo entidad relación donde se coloca los nombres de las tablas en traducción al inglés:

Figura 11. Modelo Entidad Relación



Fuente: Modelo Entidad Relación, **Nota:** Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 11 de marzo de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.2.12. Definición inicial del producto

3.2.12.1. Product Backlog

Aplicación WEB

A continuación, se ilustra la lista de funcionalidades que se van a implementar para el aplicativo web.

Tabla 5. Definición del Producto Backlog del Aplicativo WEB

Nro.	Historia de Usuario	Responsable	Sprint	Estado	Prioridad
1	HU00 - Crear Proyectos Backend-Front	Jose, Walter, Efraín	Sprint 1	Terminada	1
2	HU01-Crear Tipos Estados CRUD	Jose	Sprint 2	Terminada	3
3	HU02-Crear Unidades de Medida CRUD	Walter	Sprint 2	Terminada	3
4	HU03-Crear Países CRUD	Efraín	Sprint 2	Terminada	2
5	HU04-Crear Estados CRUD	Efraín	Sprint 2	Terminada	3
6	HU05-Crear Departamentos CRUD	Jose	Sprint 3	Terminada	2
7	HU06-Crear Clientes CRUD	Walter	Sprint 3	Terminada	4
8	HU07-Crear Usuarios CRUD	Jose	Sprint 3	Terminada	2
9	HU08-Crear Canales de Distribución CRUD	Efraín	Sprint 4	Terminada	2
10	HU09-Crear Ciudades CRUD	Walter	Sprint 4	Terminada	2

11	HU10-Crear Segmentos CRUD	Efraín	Sprint 4	Terminada	2
12	HU11-Crear Categoría CRUD	Walter	Sprint 5	Terminada	2
13	HU12-Crear Tipos de Demanda y Evento CRUD	Jose	Sprint 5	Terminada	2
14	HU13-Crear Productos CRUD	Jose	Sprint 5	Terminada	3
15	HU14-Crear Colaboración - Calendario CRUD	Walter	Sprint 6	Terminada	5
16	HU15-Crear Colaboración - Demanda CRUD	Efraín	Sprint 6	Terminada	5
17	HU16-Crear Portafolio CRUD	Jose, Walter, Efraín	Sprint 6	Terminada	5

Fuente: Propia

Aplicación Móvil

A continuación, se ilustra la lista de funcionalidades que se van a implementar para la aplicación móvil.

Tabla 6. Definición del Producto Backlog de la Aplicación Móvil

Nro.	Historia de Usuario	Responsable	Sprint	Estado	Prioridad
1	HU17 - Crear Proyectos en Flutter	Jose, Walter, Efraín	Sprint 7	Terminada	1
2	HU18-Login por email y contraseña	Jose	Sprint 8	Terminada	3
3	HU19-Recuperación de Contraseña	Walter	Sprint 8	Terminada	3
4	HU20-Crear Login para la App	Efraín	Sprint 8	Terminada	2

5	HU21-Ver Planes de Colaboración	Efraín	Sprint 9	Terminada	3
6	HU21-Editar Planes de Colaboración	Jose	Sprint 9	Terminada	2
7	HU21-Aprobar Planes de Colaboración	Jose	Sprint 9	Terminada	2
8	HU21-Exportar Planes de Colaboración	Efraín	Sprint 10	Terminada	2
9	HU22-Visualizaciónde Histórico de Ventas	Walter	Sprint 10	Terminada	2

Fuente: Propia

3.2.12.2. Historias de usuario Frontend y Backend

Aplicación WEB

A continuación, se muestran historias de usuario de funcionalidades como ejemplo, que se implementarán para el aplicativo web y que son repetitivas puesto se realiza CRUD para las entidades.

Tabla 7. Historia de usuario para listar los tipos de estados

Caso de Uso	**HU1.1 - Listar Tipos Estados CRUD - Listar.**
Actores	Planeador(a).
Propósito	Listar los tipos estados que tiene un proceso y/o maestro dentro del aplicativo. Listar los diferentes tipos estados que tiene un proceso y/o registro maestro. El **usuario** ingresa a la pestaña de los tipos de estados donde se muestra dicha lista
Precondición(es)	Que se inicie un browser y se digite la dirección del aplicativo web. Que el aplicativo web este instalado y compartido por la red local. Que el usuario este autenticado en el aplicativo.
Postcondición	Ninguna.
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Nombre**. Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100
Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo web digitando su usuario y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los **Tipos de Estados**.	4. Se muestra la página de los **Tipos de Estados** ya registrados y las operaciones posibles sobre estos.
Curso alterno	
5. Si no se tiene registros, se muestra un mensaje. **EX1**	

EX1. No se encuentra registros en la BD
Mockup

Fuente: Propia

Tabla 8. Historia de usuario para crear los tipos de estados

Caso de Uso	**HU1.2 - Crear Tipos Estados CRUD - Crear.**
Actores	Planeador(a).
Propósito	Crear un tipo de estado que tiene un proceso y/o maestro dentro del aplicativo. El usuario tiene la posibilidad de Crear un tipo de estado dentro del aplicativo web. El **usuario** ingresa a la pestaña de los tipos de estados donde se muestra dicha lista, se muestra un botón para crear un tipo de estado, dando clic en el botón y aparece un formulario.
Precondición(es)	Que se inicie un browser y se digite la dirección del aplicativo web. Que el aplicativo web este instalado y compartido por la red local. Que el usuario este autenticado en el aplicativo.
Postcondición	Almacenar el registro en la BD
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Nombre**. Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo web digitando su usuario y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los **Tipos de Estados**.	4. Se muestra la página de los **Tipos de Estados** ya registrados y las operaciones posibles sobre estos, dar clic en Nuevo Tipo Estado.
5. Se muestra el formulario con los campos requeridos	6. Se debe dar clic en el botón para guardar
Curso alterno	
7. Si no se digita los campos obligatorios, se muestra un mensaje. **EX1** 8. Se tiene la posibilidad de cancelar el guardado del registro y regresar al formulario de Listar, se muestra un mensaje. **EX2**	
EX1. Se crea exitosamente el registro en la BD	

EX2. Desea continuar con la creación/actualizar del registro o regresar a la anterior pantalla
Mockup

Fuente: Propia

Tabla 9. Historia de usuario para editar un tipo de estado

Caso de Uso	**HU01.3 - Editar Tipos Estados CRUD - Editar.**
Actores	Planeador(a).
Propósito	Editar un tipo de estado que tiene un proceso y/o maestro dentro del aplicativo. El usuario tiene la posibilidad de Editar un tipo de estado dentro del aplicativo. El **usuario** ingresa a la pestaña de los tipos de estados donde se muestra dicha lista, se muestra un botón para crear un tipo de estado, dando clic en el botón y aparece un formulario.
Precondición(es)	Que se inicie un browser y se digite la dirección del aplicativo web. Que el aplicativo web este instalado y compartido por la red local. Que el usuario este autenticado en el aplicativo.
Postcondición	Actualizar el registro en la BD
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Nombre**. Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo web digitando su usuario y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los **Tipos de Estados**.	4. Se muestra la página de los **Tipos de Estados** ya registrados y las operaciones posibles sobre estos, dar clic en Actualizar en la lista
5. Se muestra el formulario con los campos a editar	6. Se debe dar clic en el botón para actualizar
Curso alterno	
7. Si no se digita los campos obligatorios, se muestra un mensaje. **EX1** 8. Se tiene la posibilidad de cancelar la actualización del registro y regresar al formulario de Listar. **EX2**	
EX1. Se edita exitosamente el registro en la BD	

EX2. Desea continuar con la creación/actualizar del registro o regresar a la anterior pantalla
Mockup


Fuente: Propia

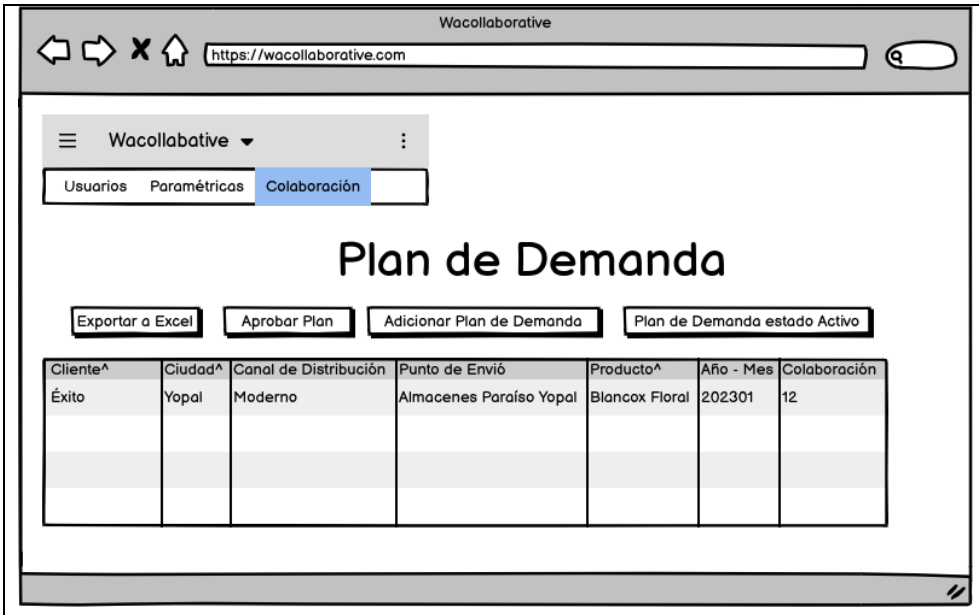
Tabla 10. Historia de usuario para Aprobar un plan de demanda

Caso de Uso	**HU15.4-Aprobar Plan de Colaboración - CRUD - Editar.**
Actores	Planeador(a)
Propósito	Listar los planes de demanda para la colaboración dentro del aplicativo. Aprobar los diferentes planes de demanda que tiene asociado un colaborador o usuario de aplicativo. El **usuario** ingresa a la pestaña de los planes de demanda donde se muestra dicha lista
Precondición(es)	Que se inicie un browser y se digite la dirección del aplicativo web. Que el aplicativo web este instalado y compartido por la red local. Que el usuario este autenticado en el aplicativo.
Postcondición	Ninguna.

Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Estado.** Campos numéricos obligatorio con longitud mínimo 1, máximo 10

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo web digitando su usuario y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los Planes de Demanda.	4. Se muestra la página de los **Planes de Demanda** ya registrados y las operaciones posibles sobre estos.
5. El usuario da clic en la opción de aprobar Plan	6. El sistema debe validar que se pueda validar el plan. Se muestra pantalla para aprobar o cancelar operación.
7. El usuario da clic en el botón de aceptar	8. El sistema cambia de estado del plan de la demanda y se envia notificación.

Curso alterno
5. Si no se completó el plan de demanda no se puede aprobar, se muestra un mensaje. **EX1**
EX1. Se debe completar el plan de la demanda
Mockup



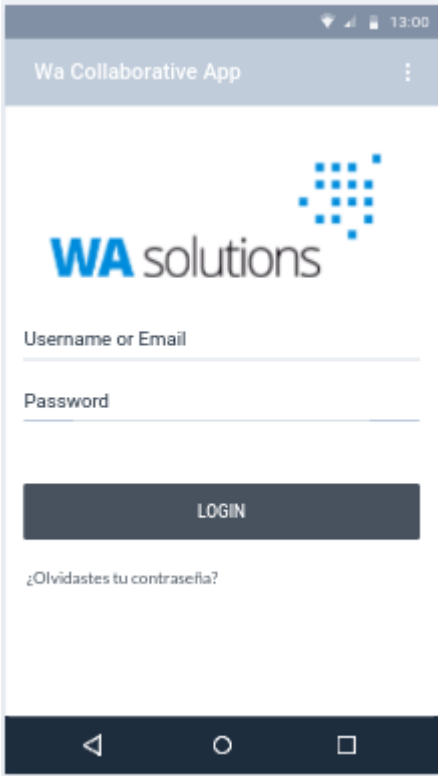
Fuente: Propia

3.2.12.3. Historias de usuario aplicación Móvil

Tabla 11. Historia de usuario para ver Planes de Colaboración

Caso de Uso	HU18.1-Login por email y contraseña
Actores	Planeador(a), Colaborador(a).
Propósito	Ingresar a la aplicación móvil. El usuario puede ingresar a aplicación móvil desde cualquier lugar. El **usuario** ingresa a la aplicación móvil digita sus credenciales, dando clic en el botón de login para ingresar.
Precondición(es)	Que el aplicativo móvil este instalado. Que se inicie a la aplicación móvil desde un dispositivo móvil Que el usuario este autenticado en la aplicación móvil.
Postcondición	Ingresar a la aplicación móvil
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Usuario o Email**. Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100 **Contraseña.** Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa a la aplicación móvil digitando su usuario o email y contraseña.	2. Se valida los datos y se muestra la **Página Principal** que tiene cada usuario dentro de la aplicación móvil.
3. El usuario escoge la opción de los **Tipos de Estados**.	4. Se muestra la página de los **Tipos de Estados** ya registrados y las operaciones posibles sobre estos, dar clic en Actualizar en la lista
5. Se muestra el formulario con los campos a editar	6. Se debe dar clic en el botón para actualizar
Curso alterno	
7. Si no se digita los campos obligatorios, se muestra un mensaje. **EX1** 8. Se tiene la posibilidad de cancelar la actualización del registro y regresar al formulario de Listar. **EX2**	
EX1. Se edita exitosamente el registro en la BD **EX2.** Desea continuar con la creación/actualizar del registro o regresar a la anterior pantalla	
Mockup	



Fuente: Propia

Tabla 12. Historia de usuario para ver Planes de Colaboración

Caso de Uso	**HU21.1-Ver Planes de Colaboración**
Actores	Planeador(a), Colaborador(a)
Propósito	Listar los planes de demanda para la colaboración dentro del aplicativo móvil. Listar los diferentes planes de demanda que tiene asociado un colaborador o usuario. El **usuario** ingresa a la pestaña de los planes de demanda donde se muestra dicha lista en la aplicación móvil
Precondición(es)	Que el aplicativo móvil este instalado. Que se inicie a la aplicación móvil desde un dispositivo móvil Que el usuario este autenticado en la aplicación móvil.
Postcondición	Ingresar a la aplicación móvil
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Cliente.** Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100

	Producto. Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100 **Estado.** Campos numéricos obligatorio con longitud mínimo 3, máximo 10

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo móvil digitando su usuario o email y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los **Planes de Demanda**.	4. Se muestra la página de los **Planes de Demanda** ya registrados y las operaciones posibles sobre estos.
Curso alterno	
5. Si no se tiene registros, se muestra un mensaje. **EX1**	
EX1. No se encuentra registros en la BD	
Mockup	

Fuente: Propia

Tabla 13. Historia de usuario para editar Planes de Colaboración

Caso de Uso	**HU21.2-Editar Planes de Colaboración**
Actores	Colaborador(a)
Propósito	Listar los planes de demanda para la colaboración dentro del aplicativo móvil. Listar los diferentes planes de demanda que tiene asociado un colaborador o usuario. El **usuario** ingresa a la pestaña de los planes de demanda donde se muestra dicha lista en la aplicación móvil
Precondición(es)	Que el aplicativo móvil este instalado. Que se inicie a la aplicación móvil desde un dispositivo móvil Que el usuario este autenticado en la aplicación móvil.
Postcondición	Ingresar a la aplicación móvil
Referencias Cruzadas	Satisface la **Seguridad, Eficiencia y desempeño, Usabilidad.**
Campos	**Cliente.** Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100 **Producto.** Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100 **Ciudad.** Campos alfanuméricos obligatorio con longitud mínimo 3, máximo100 **Año.** Campos numéricos obligatorio con longitud mínimo 4, máximo 4 **Mes.** Campos numéricos obligatorio con longitud mínimo 2, máximo 2 **Cantidad Colaboración.** Campos numéricos obligatorio con longitud mínimo 1, máximo 10
Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1. El usuario ingresa al aplicativo móvil digitando su usuario o email y contraseña.	2. Se muestra la **Página Principal** que tiene cada usuario.
3. El usuario escoge la opción de los **Planes de Demanda**.	4. Se muestra la página de los **Planes de Demanda** ya

	registrados y las operaciones posibles sobre estos.
5. El usuario elige el plan de la demanda donde se va a colaborar	6. Se muestra la página para agregar la colaboración.
7. El usuario digite la cantidad de colaboración al plan	8. Se valida la cantidad y se almacena en la BD con la actualización del registro.
Curso alterno	
5. Si no se tiene registros, se muestra un mensaje. **EX1**	
EX1. No se encuentra registros en la BD	
Mockup	

Fuente: Propia

3.2.12.4. Definición de Sprints

Para implementar el proyecto, se prevé que cada sprint dure 15 días, de 2 dos semanas, con un promedio de 2 o 3 historias de usuario realizadas entre el equipo Scrum.

Sprint 1. Para este sprint se realizará la creación de los proyectos, aplicación API Rest como Backend y aplicación Web como FrontEnd.

Sprint 2. Para este sprint se realizará de la implementación de los servicios Rest y pantallas de las historias de usuario HU01-Crear Tipos Estados, HU04-Crear Estados, Crear Unidades de Medida, HU03-Crear Países.

Sprint 3. Para este sprint se realizará de la implementación de los servicios Rest y pantallas de las historias de usuario HU05-Crear Departamentos CRUD, HU06-Crear Clientes CRUD, HU07-Crear Usuarios CRUD.

Sprint 4. Para este sprint se realizará de la implementación de los servicios Rest y pantallas de las historias de usuario HU08-Crear Canales de Distribución CRUD, HU09-Crear Ciudades CRUD, HU10-Crear Segmentos CRUD.

Sprint 5. Para este sprint se realizará de la implementación de los servicios Rest y pantallas de las historias de usuario HU11-Crear Categoría CRUD, HU12-Crear Tipos de Demanda y Evento CRUD, HU13-Crear Productos CRUD.

Sprint 6. Para este sprint se realizará de la implementación de los servicios Rest y pantallas de las historias de usuario HU14-Crear Colaboración - Calendario CRUD, HU15-Crear Colaboración - Demanda CRUD, HU16-Crear Portafolio CRUD.

Sprint 7. Para este sprint se realizará la creación del proyecto de la aplicación Móvil como Frontend.

Sprint 8. Para este sprint se realizará de la implementación de las pantallas en el aplicativo móvil de las historias de usuario HU18-Login por email y contraseña, HU19-Recuperación de Contraseña, HU20-Crear Login para la App.

Sprint 9. Para este sprint se realizará de la implementación de las pantallas en el aplicativo móvil de las historias de usuario HU21-Ver Planes de Colaboración, HU21-Editar Planes de Colaboración, HU21-Aprobar Planes de Colaboración.

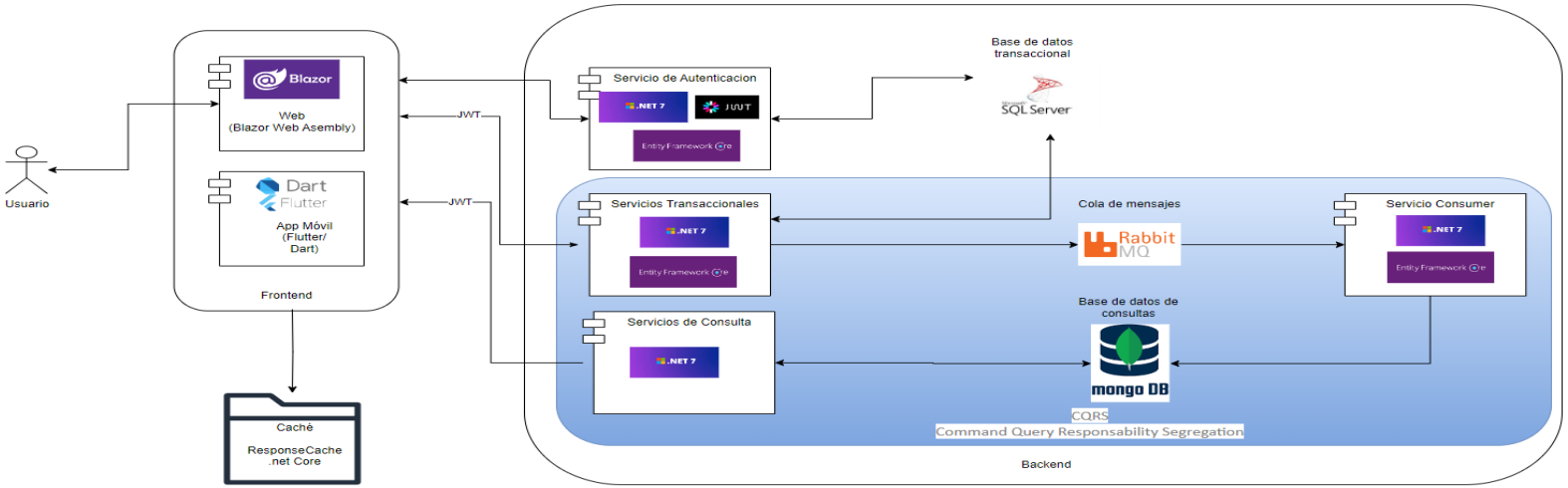
Sprint 10. Para este sprint se realizará de la implementación de las pantallas en el aplicativo móvil de las historias de usuario HU21-Exportar Planes de Colaboración, HU22-Visualizaciónde Histórico de Ventas.

3.3. Diseño de la Solución

3.3.1. Modelo de Arquitectura

A continuación, se muestra el modelo de arquitectura propuesto para WA Collaborative:

Figura 12. Modelo de Arquitectura



Fuente: Modelo de Arquitectura, Nota: Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 15 de abril de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.3.2. Descripción modelo de Arquitectura

La arquitectura anterior está compuesta por dos contenedores principales, el Frontend y el Backend, cada uno de ellos agrupa un conjunto de elementos que se pasa a describir de manera más detallada a continuación:

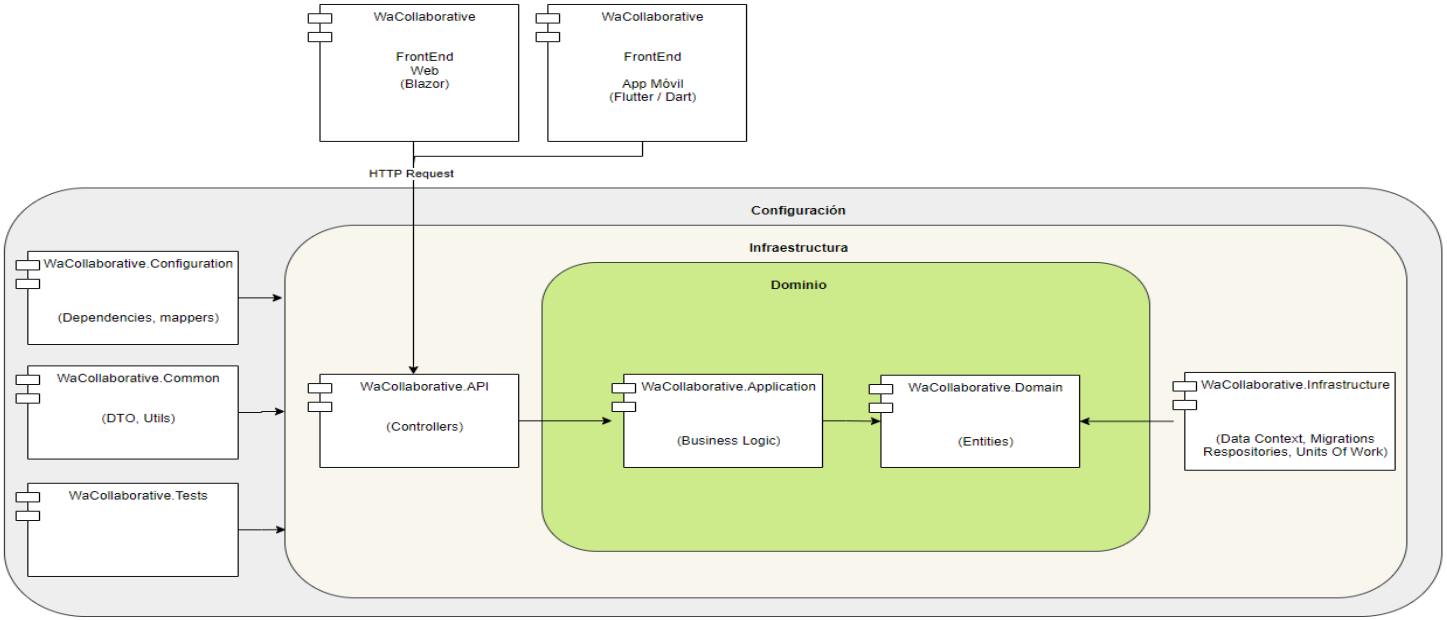
- **FrontEnd:** Este elemento se desarrollará en Blazor Webassembly, tecnología propia de Microsoft que permite hacer uso del lenguaje C# en su implementación, cuenta una comunidad activa, con el soporte por parte de Microsoft y con una amplia gama de librerías de las cuales se puede hacer uso para optimizar el desarrollo de aplicaciones.
- **BackEnd:** Se propone desarrollar el Backend en .NET Framework 7, lenguaje de programación C#, mediante el uso de Api Controllers para exponer servicios de tipo Api Rest. Se plantea una única solución Backend que agrupará los servicios que expone en cuatro grupos principales:
    - **Servicio de Autenticación:** Es el encargado de validar las credenciales del usuario que intenta acceder al sistema y con base en la validación aprobar o denegar el acceso. También hará uso de las tecnologías JWT por medio del cual se gestionará las autorizaciones a funcionalidades del sistema y de Entity Framework Core por medio del cual se manejará la validación de credenciales contra la base de datos.
    - **Servicios Transaccionales:** Son todos aquellos servicios que realicen las acciones de inserción, actualización o eliminación de registros de la base de datos. Usará Entity Framework Core para acceder la base de datos transaccional y también publicarán en Rabbit MQ, mediante colas de mensajes, la información del registro que se actualiza en la base de datos transaccional para que sincronice la base de datos de solo lectura en MongoDB.
    - **Servicio Consumer: C**on el ánimo de hacer uso del patrón de arquitectura CQRS, una vez los servicios transaccionales publiquen en RabbitMQ el mensaje queda en cola y lanza un evento para que el servicio consumer lea el siguiente elemento en cola y actualice en la base de datos de Mogo DB la información de lectura, esto para que la base de datos transaccional y la de solo lectura estén sincronizadas.
    - **Servicios de Consulta:** Son todos aquellos servicios que únicamente consultan información en la base de datos, lo harán directamente a la base de datos de solo lectura que estará en mogo DB con el fin de afectar la disponibilidad y la performance de la aplicación.
- Otros componentes de la arquitectura:

- **SQL Server:** Siguiendo con la línea de Microsoft, por el soporte, la madurez y la documentación que tiene SQL Server aquí se alojará la base de datos transaccional. SQL Server cuenta con un grupo de herramientas de administración y mantenimiento de base de datos muy completo y eso da garantía que el riesgo de pérdida de información es muy bajo.
- **RabbitMQ:** Es el componente donde se publicarán los registros actualizados en la base de datos para sincronizar la base de datos de solo lectura.
- **Mongo DB:** Es una base de datos NoSQL optimizada para el análisis de información y para entregar un buen performance sobre consultas de grandes volúmenes de datos por eso en ella se alojará la base de datos de solo lectura.
- **Response Cache:** Es una funcionalidad propia de .NET que permite mantener el caché en memoria durante determinado tiempo. Ya que el caché es en memoria allí se almacenarán pequeños volúmenes de datos, por ejemplo, datos de parametrización que no cambian mucho en el tiempo.

3.3.3. Arquitectura limpia Backend

El siguiente diagrama muestra el modelo de arquitectura limpia del Backend:

Figura 13. Modelo de Arquitectura Limpia



Fuente: Modelo de Arquitectura, Nota: Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 15 de abril de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.3.4. Descripción arquitectura limpia Backend

A continuación, se describen los elementos que componen el modelo de arquitectura limpia de Wa Collaborative:

- **WaCollaborative.Backend:** Está desarrollado en C#, .Net 7, contiene una arquitectura N Capas para garantizar la responsabilidad única de cada elemento de código.

  Las capas se abstraen en los siguientes proyectos dentro de la solución:

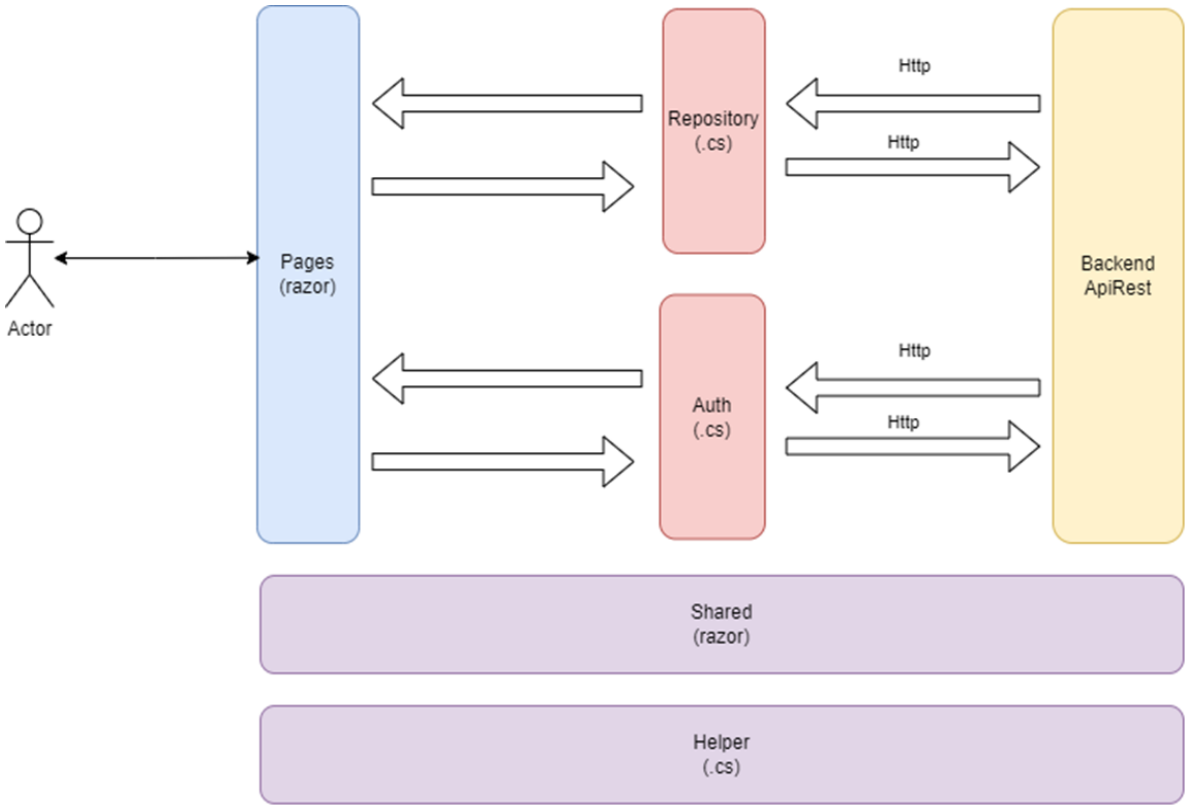
  - **WaCollaborative.API**: Es un proyecto web que contiene los servicios Rest Api de tipo ApiController, referencia al proyecto WaCollaborative.Application.
  - **WaCollaborative.Application**: Es un proyecto de tipo biblioteca de clases que contiene la lógica de negocio de la aplicación, referencia al proyecto WaCollaborative.Infrastructure.
  - **WaCollaborative.Infrastructure:** Es un proyecto de tipo biblioteca de clases que contiene todo lo relacionado con el acceso a bases de datos, contiene repositorios, unidad de trabajo, migraciones y data context. Hace uso del ORM Entity Core para el acceso a bases de datos.
  - **WaCollaborative.Domain**: Es un proyecto referenciado por WaCollaborative.Infrastructure y por WaCollaborative.Application, es de tipo biblioteca de clases y contiene todas las entidades del dominio de la aplicación.
  - **WaCollaborative.Configuration**: Es un proyecto transversal a todo el backend, de tipo biblioteca de clases que contiene todo lo relacionado con configuraciones como por ejemplo configuración del inyector de dependencias y mapeos.
  - **WaCollaborative.Common**: Es un proyecto transversal a todo el backend, de tipo biblioteca de clases que contiene las utilidades comunes y los DTO de la aplicación.

  - **WaCollaborative.Tests**: Este proyecto contiene todos los Tests unitarios de la aplicación por lo que es transversal a todo el Backend.

3.3.5. Arquitectura limpia Web

El siguiente diagrama es el correspondiente al modelo de arquitectura limpia del Frontend web, describe como interactúan cada uno de sus componentes:

Figura 14. Modelo de Arquitectura Limpia Web



Fuente: Modelo de Arquitectura Limpia Web, Nota: Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 15 de abril de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.3.6. Descripción de arquitectura limpia web

WaCollaborative.WEB: Es un proyecto desarrollado en Blazor que tiene las páginas junto a los formularios para crear y/o actualizar registros y mostrar información.
Se maneja un único proyecto y las capas se abstraen a una estructura de carpetas que garantizan responsabilidad única en los elementos que contiene cada una, las capas son las siguientes:

Pages: dentro de esta capa se crea una carpeta por cada módulo de la aplicación y este último nivel contiene los archivos .razor que son las páginas que accederán los usuarios cuando se publique el sitio.

Auth: esta capa contiene las clases encargadas de la lógica de autenticación. Contiene aspectos como login, logout, claims, Jwt.

Repositories: esta capa contiene la lógica de consumo de los servicios Api Rest expuestos por el Backend.

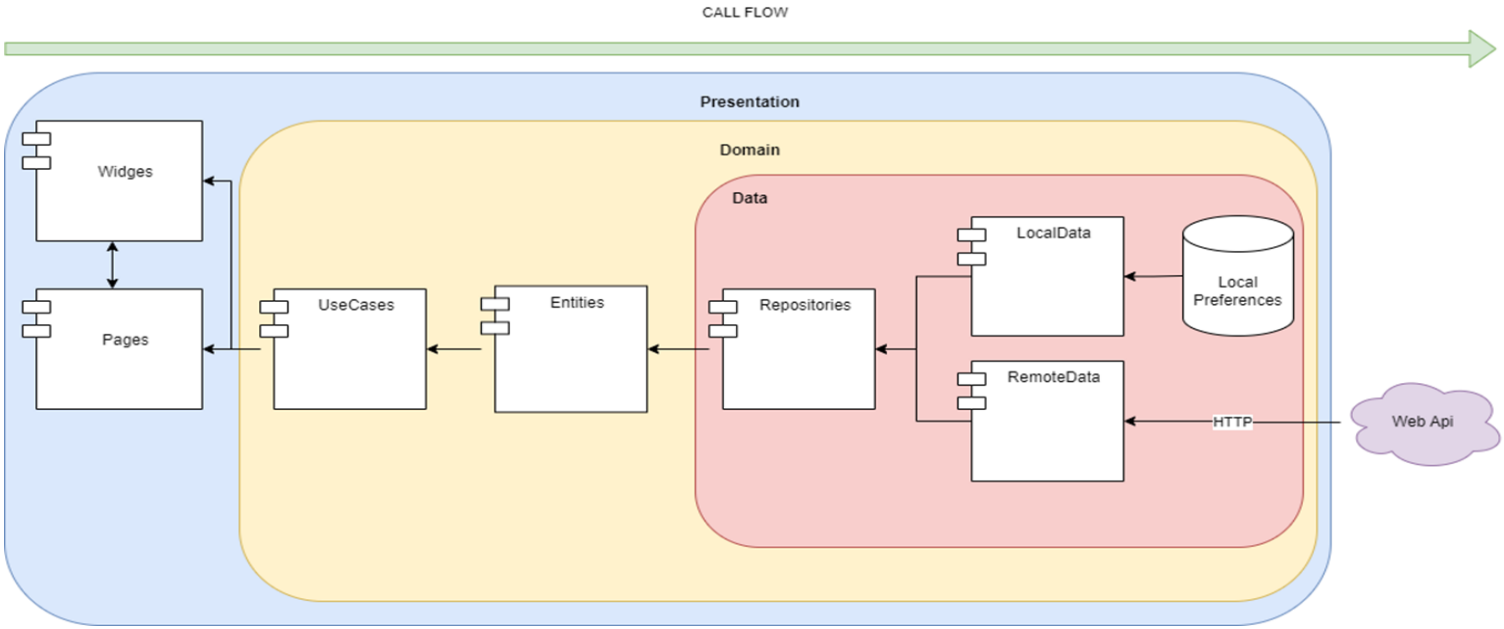
Helpers: es una capa transversal del proyecto que contiene el grupo de clases de utilidades que pueden ser usadas desde cualquier parte del Front.

Shared: esta capa contiene los archivos .razor que son compartidos por el Front.

3.3.7. Arquitectura limpia Aplicación Móvil

Mediante el siguiente diagrama se muestra el modelo de arquitectura limpia propuesto para la aplicación móvil:

Figura 15. Modelo de Arquitectura Limpia Móvil



Fuente: Modelo de Arquitectura Limpia Web, Nota: Para visualizar mejor el diagrama ingresara al siguiente enlace [en línea] el 15 de abril de 2024 en https://app.diagrams.net/#G1I61mQ_HZr1MqamJ_jS0T3qP0xIEit8sZ#%7B%22pageId%22%3A%22kl5em6jxrjE1lnDLplB6%22%7D

3.3.8.Arquitectura Limpia Aplicación Móvil

La arquitectura de la aplicación móvil tendrá 3 capas. La primera capa es de presentación, esta tendrá los Widgets personalizados para el proyecto y los Pages que son las ventanas de la aplicación con las cuales van a interactuar los usuarios, la segunda capa es dominio y contendrá los casos de uso y las entidades del proyecto; la tercera es la capa de datos que contendrá los repositorios que accederán a fuentes de datos externas o locales

Esta arquitectura limpia propende por la responsabilidad única de cada componente, por el bajo acoplamiento y por la alta cohesión con el fin de garantizar un código más limpio, escalable y mantenible.

A continuación, se describe cada uno de los elementos que componen la arquitectura:

- **WACollaborative_app**: es una solución realizada en flutter el cual contendrá las siguientes capas y componentes:
    - **Presentation**: es una de las tres capas del proyecto y se representará dentro de la solución de código como una carpeta con el mismo nombre, dentro de ella se encuentran otras dos carpetas:
        - **CustomWidges**: si por necesidad o por facilidad se necesita personalizar un widget para ser usado en diferentes paginas entonces se creará dentro de esta carpeta.
        - **Pages**: tendrá cada página que se necesite para la aplicación móvil

    - **Domain**: es la segunda capa planteada en la arquitectura limpia de la aplicación móvil y también se representará de manera física en la solución como una carpeta con el mismo nombre. Tendrá dentro de sí las siguientes carpetas:
        - **UseCases**: Esta carpeta contendrá las clases que encapsulan la lógica de negocio de la aplicación. Dentro del proyecto este nivel de lógica es mínimo ya que el Backend contendrá la mayoría de la lógica de negocio.
        - **Entities**: son la representación abstracta de los objetos de dominio mediante clases.

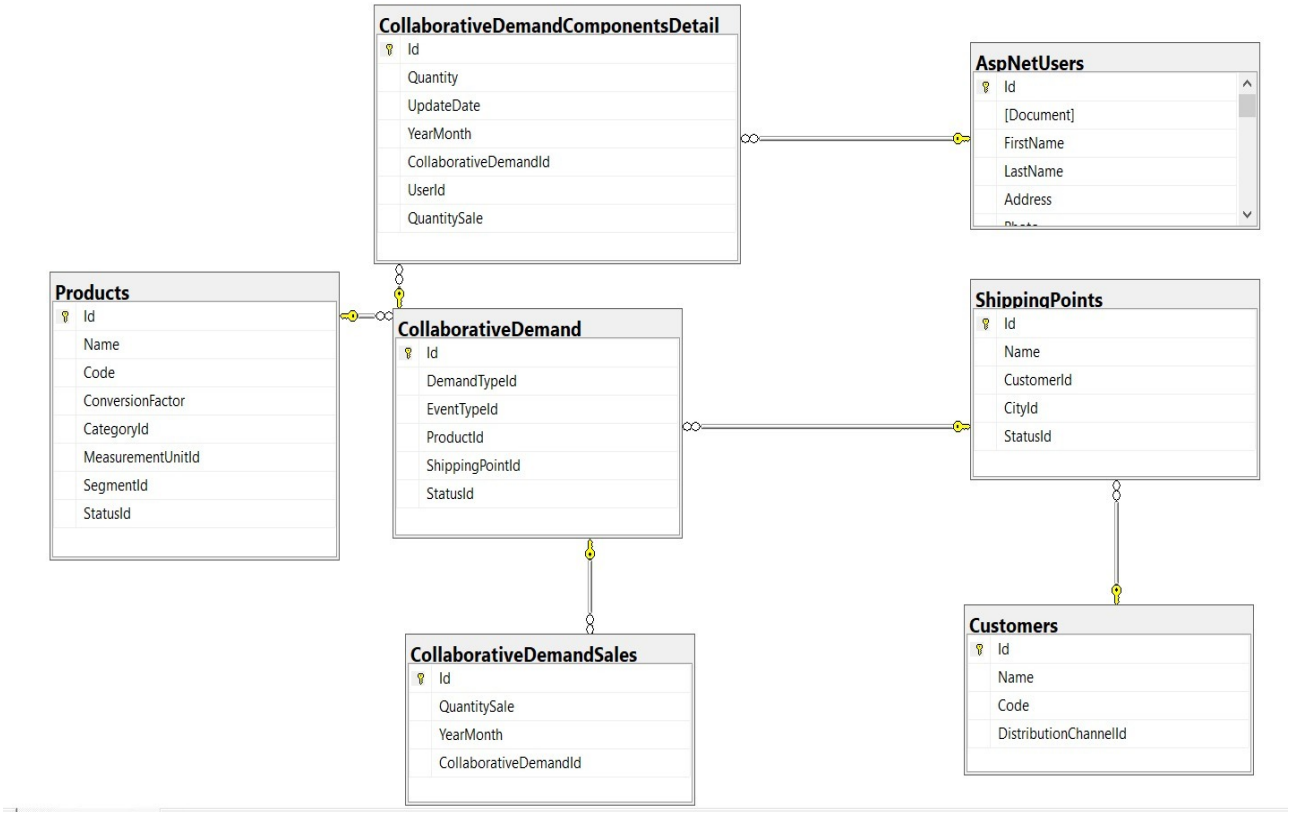
    - **Data**: es la tercera y última capa de la aplicación móvil, también representada por una carpeta que lleva su mismo nombre, contendrá todo lo referente a acceso a datos, sean locales, bases de datos o consumo de Api. Tendrá dentro de si las siguientes carpetas:
        - **Repositories**: contendrá los contratos necesarios para el acceso a datos, interfaces o componentes transversales

- **LocalData**: tendrá las clases que implementarán, para el acceso a datos local, los contratos establecidos en la carpeta Repositories.
- **RemoteData**: tendrá las clases que implementarán, para el acceso a datos mediante el consumo de Apis, los contratos establecidos en la carpeta Repositories.

3.3.9. Modelo de Base de Datos

El siguiente es el modelo de Base de Datos:

Figura 16. Modelo de Base de Datos



Fuente: Propio

3.3.10. Estilos arquitectónicos

A continuación, se describe el estilo arquitectónico utilizado en WA Collaborative:

Tabla 14. Estilos Arquitectónicos utilizado en WA Collaborative

Arquetipo	Web
Estilo/patrón arquitectónico y descripción	1. Domain Driven Design 2. Service Oriented Architecture (SOA) 3. Command query responsability segregation Justificación: Se escogen los patrones “Domain Driven Design” y “Layered Architecture” con el ánimo de crear una arquitectura centrada en el dominio y estructurada a nivel de capas lo que a su vez ayuda a cumplir el principio de responsabilidad única. También se escoge el patrón CQRS (Command query responsability segregation) para dividir la base de datos transaccional de la base de datos de solo lectura y poder escalar únicamente esta última en caso de requerirse ya que se prevé que el sistema realice consultas de grandes volúmenes de datos en el futuro cercano.
Lenguaje programación	1. C# Justificación: Se escoge el lenguaje de programación C# ya que es uno de los más populares a la fecha, tiene un alto grado de madurez, una comunidad activa muy amplia a nivel mundial y el respaldo de Microsoft.
Aspectos técnicos	1. Motor de base de datos sql server 2019 2. RabbitMQ 3. MongoDB Justificación: El software usará como motor de la base de datos transaccional Sql Server por su grado de madurez, por ser una base de datos relacional, por las herramientas de mantenimiento de base de datos con que cuenta, por ser muy segura, fácilmente migrable a la nube, por la comunidad activa mundialmente que aporta conocimiento en línea del producto. Se usará RabbitMQ para sincronizar las bases de datos de escritura y se solo lectura; se escoge como middelware o broker de mensajería por el manejo de colas con que cuenta,

	la posibilidad de poder suscribir más de un servicio consumidor (consumer) ya que esto facilita el escalamiento, también cuenta con una comunidad muy activa. MongoDB se usará como base de datos de solo lectura porque es un NoSql optimizado para grandes volúmenes de datos, al no ser relacional las consultas son más optimas ya que se pueden guardar en él documentos JSON exclusivamente con la información necesaria en cada consulta y así evitar inner joins sobre una base de datos transaccional; también tiene una amplia comunidad activa.
Frameworks	1. .Net Framework 7 2. Blazor Justificación: El sistema se desarrollará con .Net Framework 7 ya que provee una amplia gama de componentes de código que optimizan el proceso de codificación, es una suite con mucho reconocimiento a nivel mundial, cuenta con una comunidad virtual muy amplia y activa, tiene el respaldo de Microsoft y su versión 7 es una de la más recientes y estable a la fecha por su grado de madurez. Blazor se usará del lado del Frontend ya que al permitir usar C# para programar reduce la curva de aprendizaje del equipo de trabajo, también cuenta con el respaldo de Microsoft y con una comunidad en línea activa.

Fuente: Propio

El proyecto WA Collaborative se desarrolla, del lado Backend, con un estilo de arquitectura orientado a servicios ya que se expondrán servicios web Api Rest para que sean consumidos por el Front y de esta manera mantener separadas las responsabilidades del Backend de las responsabilidades del Frontend, se hace uso de .Net versión 7, lenguaje de programación C#.

.Net permite la creación de controladores que se exponen como servicios Api Rest al heredar de la clase Controller y hacer uso del Data Anotation [ApiController] y de esta manera también permite personalizar los end point de acceso a los servicios por medio del Data Anotation [Route(“api/name-end-point”)].

Del lado front end se usa Blazor. Blazor es un framework de microsoft que permite programar el front con lenguaje c#, por medio de blazor se consumirán los resvicios rest expuestos en por el backend.

3.3.11. Principios Solid

En la siguiente tabla se describe como se implementan los principios SOLID en WA Collaborative

Tabla 15. Principios Solid aplicados en WA Collaborative

Principios SOLID	
Single Responsibility	1. A nivel de clases cada de una de estas solo tienen una única responsabilidad 2. El Frontend está en una solución independiente al Backend 3. Con la arquitectura N capas se segregan las responsabilidades por proyecto
Open/Closed	1. Cada clase implementa una o más interfaces que no pueden ser extensas en cantidad de métodos para garantizar que las interfaces tienen bien definidas las responsabilidades, por ejemplo, un colaborador (Collaborator) y un planeador (Glider) son tipos de usuario y cuentan con funciones similares, aunque hay algunas que son diferentes, por eso el código se implementará de esta manera: CollaboratorService : IUserService, ICollaboratorService GliderService: IUserService, IGliderService
Liskov Substitution	Se implementa en la creación del repositorio genérico 1. Existe la interfase IBaseRepository 2. La clase BaseRepository implementa la interfase IBaseRepository

```
14 references
public interface IBaseRepository<TEntity>
{
    7 references
    Task<TEntity> GetAsync(int id);

    7 references
    Task<IEnumerable<TEntity>> GetAsync();

    3 references
    Task<TEntity> AddAsync(TEntity entity);

    3 references
    Task DeleteAsync(int id);

    3 references
    Task<TEntity> UpdateAsync(TEntity entity);
}
```

	 3. La clase CitiesRepository hereda de la clase BaseRepository y con eso puede ejecutar todos sus métodos, incluyendo los que BaseRepository implementó de IBaseRepository aunque CitiesRepository no haya implementado esta interfase.
Interface Segregation	1. El código está segregado en interfaces con responsabilidades claras respecto de la entidad de dominio sobre la cual trabajarán como de las responsabilidades a nivel logica de dominio (services) o de acceso a datos (repository), ejemplo: a. IBaseService b. IBaseRepository c. ICitiesService d. ICitiesRepository e. ICountriesService f. ICountriesRepository g. IStatesService h. IStatesRepository i. IUserService j. IUserRepository k. IGliderService l. IGliderRepository m. IcollaboratorService n. IcollaboratorRepository o. ICollaborationCalendarService p. ICollaborationCalendarRepository q. IBriefcaseService r. IBriefcaseRepository s. IRoleService t. IRoleRepository u. ICollaborativeDemandService v. ICollaborativeDemandRepository w. IProductService x. IProductRepository

	y. ICustomerService z. ICustomerRepository aa. ICollaborationGroupService bb. ICollaborationGroupRepository cc. IDemandTypeService dd. IDemandTypeRepository ee. IEventTypeService ff. IEventTypeRepository gg. IDemandPlanService hh. IDemandPlanRepository ii. ICategoryService jj. ICategoryRepository kk. ISegmentService ll. ISegmentRepsitory mm. IUnitOfMeasurementService nn. IUnitOfMeasurementRepository

Dependency Inversion	En el código se usar inyección de dependencias, por ejemplo, en el flujo de ciudades:
	`public class CitiesService : BaseService<CityDTO, City>, ICitiesServic`
	`{`
	`  private ICitiesRepository _citiesRepository;`
	`  private IMapper _mapper;`
	`  0 references`
	`  public CitiesService(`
	`  IBaseRepository<City> repository,`
	`  IMapper mapper,`
	`  ICitiesRepository citiesRepository)`
	`  : base(repository, mapper)`
	`  {`
	`  _citiesRepository = citiesRepository;`
	`  _mapper = mapper;`
	`  }`
	`}`

Fuente: Propio

3.3.12. Principios GRASP

Tabla 16. Principios GRASP aplicados en WA Collaborative

Principios GRASP	
Bajo Acoplamiento	Este principio se aplica en la arquitectura de Wa Collaborative en los siguientes casos: 1. El Frontend y el Backend no están relacionados entre sí, sino que la comunicación se establece a través de las

	peticiones http que realiza el Front al back y las respuestas de este. 2. Los controladores, las clases de la lógica de dominio (service) y las clases de la infraestructura (repository) se comunican por medio de inyección de dependencias.
Alta Cohesión	Este principio se aplica en la arquitectura de Wa Collaborative en los siguientes casos: 1. Cada elemento u objeto de negocio tiene su propia clase a nivel de Entidad de dominio, de lógica de dominio (service) y de acceso a datos (repository), ejemplo para el producto: se tiene la entidad Product, la clase ProductService que implementa la interfaz IProductService y la clase ProductRepository que implementa la interfaz ProductRepository. 2. El contrato de los servicios que expone el Backend son respetados por el Frontend al momento de hacer las peticiones, sucede lo mismo con las respuestas, ambos guardan coherencia.
Experto	Cada clase en la arquitectura de la aplicación tiene la responsabilidad sobre un único objeto de negocio y esa responsabilidad adicionalmente también se segrega según la lógica que implementa, si lógica de dominio (service) o lógica de acceso a datos (repository), ejemplo para producto: se tiene la entidad Product, la clase ProductService que implementa la interfaz IProductService y la clase ProductRepository que implementa la interfaz ProductRepository.
Creador	La instancia de nuevos objetos se hace mediante la inyección de dependencias y para eso se crea una clase inyector (DependecyInjection.cs) en el proyecto WaCollaborative.Configuration, este inyector se encarga de registrar cada interfaz con su clase correspondiente y administrar la creación de nuevas instancias de las clases.
Controlador	Se usa un proyecto web de tipo ApiController donde estas clases Controller reciben las peticiones del Frontend y mediante inyección de dependencias dirigen al flujo necesario para dar una respuesta lógica al cliente.
Polimorfismo	Se crean las clases BaseController que recibe recibirá genéricos de C#, esta clase implementa métodos de CRUD tradicionales y es heredada por los demás controladores así que dependiendo del tipo genérico realizara el CRUD sobre una entidad especifica:

	9 references
	public class BaseController<TDTO> : Controller where TDTO : class
	{
	private IBaseService<TDTO, TDTO> _service;
	
	4 references
	public BaseController(IBaseService<TDTO, TDTO> service)
	{
	_service = service;
	}
	
	[HttpGet]
	0 references
	public async Task<IActionResult> GetAsync()
	{
	return Ok(await _service.GetAsync());
	}
	
	Al igual que el BaseController también existirá un BaseService y un BaseRepository

Fabricación Pura	N/A
Indirección	N/A
Variaciones protegidas	El código fuente define un conjunto de interfaces con el ánimo de protegerlo frente a nuevos requisitos, las interfaces están listadas en la tabla de principios Solid, en el principio Interface Segregation

Fuente: Propio

3.3.13. Atributos de calidad – Requisitos no funcionales

Mediante la siguiente tabla se describen los requisitos no funcionales del software

Tabla 17. Atributos de Calidad – Requisitos no funcionales en WA Collaborative

Código	Tipo RNF	Nombre	Descripción	Criterios De Aceptación
HU_RNF 001	Seguridad	Registro y Autenticación	Yo como director de TI requiero que un usuario se pueda registrar y autenticar con el email y la contraseña	• Un usuario solo puede ser creado por un usuario planeador de la demanda. • El email que digita el usuario al momento de ser registrado en el sistema será el usuario

			para generar seguridad en la aplicación	con el que se autentique. • Cuando se haga un registro de un usuario nuevo o se modifique un email de un usuario, el sistema debe enviar un enlace a ese email registrado para que una vez el usuario de clic en él lo lleve a una página para crear su contraseña. • La contraseña que cree el usuario debe tener mínimo 8 caracteres de los cuales debe tener mínimo 1 letra mayúscula, 1 número y un carácter especial. • La aplicación debe controlar que el email con el que se registra un usuario no sea un email de prueba, ejemplo (mail@mailinator.com). • La aplicación debe validar que el email con el que se registra el usuario tenga la estructura de un correo electrónico, de lo contrario mostrar un mensaje claro del error. • La aplicación debe permitir configurar una opción, al administrador del sistema, para que en el campo email solo se permitan correos electrónicos de los dominios configurados y

				de esta manera descartar correos como gmail, hotmail, etc. • Al momento de autenticarse en el sistema el usuario solo tendrá la opción de tres intentos fallidos, una vez consuma sus tres intentos se debe bloquear su usuario para no permitirle acceso al sistema. • Los usuarios solo pueden ser desbloqueados por un usuario planeador de la demanda. • Una vez un usuario sea desbloqueado se le enviará a su email un enlace para que cree su contraseña. • Si un usuario olvida su contraseña se le enviará un enlace a su correo electrónico para que cree una nueva contraseña. • La contraseña de cada usuario debe expirar cada tres meses. • Si un usuario ingresa y su contraseña está expirada entonces el sistema debe enviarle un email con un enlace para que cree su nueva contraseña. • El sistema no debe permitir que a un usuario bloqueado se le envíe un enlace de creación de

					contraseña en ningún caso. • Los enlaces para crear una nueva contraseña deben tener una validez de 10 minutos máximo y una vez el usuario cree su nueva contraseña debe invalidarse.
HU_RNF 002	Seguridad	Trazabilidad	Yo como director de TI requiero que el sistema registre un log de trazabilidad para poder rastrear toda actividad que realice un usuario en la aplicación		• Cuando un usuario realice cualquier actividad de, inserción, modificación o eliminación de datos en la aplicación entonces el sistema debe registrar esa actividad en un log que puede ser consultado únicamente por el usuario planeador, los datos que debe guardar el sistema son los siguientes: email de usuario, fecha y hora de la acción, tipo de acción (ejemplo, registro de colaboración, aprobación de demanda, etc.), valores afectados por la acción.
HU_RNF 003	Eficiencia y desempeño	Respuesta rápida del sistema	Yo como cliente requiero que el sistema sea rápido en responder para optimizar el tiempo de los usuarios		• El Grid de colaboración donde se carga la demanda que puede registrar un colaborador debe responder en máximo 10 segundos. • Cuando un colaborador registre la demanda el guardado no

				debe tardar más de 5 segundos. • La aprobación de la demanda no debe tardar más de 10 segundos. • La generación del plan de demanda no debe tardar más de 1 minuto.
HU_RNF 004	Usabilidad	Facilidad de uso	Yo como cliente requiero que el sistema provea fluidez y facilidad de uso para el usuario con el fin de que la curva de aprendizaje sea corta	• El menú con las opciones que tiene cada usuario sobre el sistema debe estar al lado izquierdo de la aplicación y se debe poder ocultar para proveer de mayor espacio al área de trabajo. • En el Grid de colaboración el usuario debe poder navegar entre las columnas de una misma fila con la tecla Tab y con las teclas flecha a la izquierda y flecha a la derecha. • En el grid de colaboración el usuario debe poder navegar entre diferentes filas con las teclas flecha arriba y flecha abajo. • El software debe ser responsive ajustándose a los tres tamaños de pantallas más usados.

Fuente: Propio

3.3.14. Escenarios

Los siguientes son los escenarios relacionados a los requisitos no funcionales:

Tabla 18. Escenario de Seguridad para WA Collaborative

Escenario de Seguridad HU_RNF 001	**Fuente/origen:**	Usuarios del aplicativo
	Estímulo:	Registro de usuarios, roles y permisos para el acceso a la aplicación
	Artefacto:	Módulo de Usuario
	Entorno:	Acceso protegido
	Respuesta:	Se debe indicar si el usuario tiene acceso a la aplicación y sus diferentes módulos
	Medida/Métrica:	Que cada petición se valide si el usuario tiene permisos para realizarla

Fuente: Propio

Tabla 19. Escenario de Eficiencia y Desempeño para WA Collaborative

Escenario de Eficiencia y desempeño	Fuente/origen:	Usuarios del aplicativo
	Estímulo:	Que en la navegación del aplicativo sea rápida sin interrupciones
	Artefacto:	Aplicativo
	Entorno:	Ejecución del aplicativo
	Respuesta:	Cada petición es rápida en su respuesta
	Medida/Métrica:	Cada petición debe demorar máximo 5 segundos

Fuente: Propio

Tabla 20. Escenario de Usabilidad para WA Collaborative

Escenario de Usabilidad	Fuente/origen:	Usuarios del aplicativo
	Estímulo:	Navegación del aplicativo
	Artefacto:	Aplicativo
	Entorno:	Ejecución del aplicativo
	Respuesta:	Que el aplicativo sea amigable en su uso, que este siempre disponible sin interrupciones y que se tenga mensajes guías para su uso
	Medida/Métrica:	Cada petición al aplicativo debe tener una respuesta de éxito o fallo

Fuente: Propio

3.3.15. Tácticas de arquitectura

Por medio de la siguiente tabla se detallan las tácticas de arquitectura que se aplicarán para abordar los requisitos no funcionales:

Tabla 21. Tácticas de Arquitectura implementadas en WA Collaborative

RNF	Tipo RNF	Tácticas	Descripción
HU_RNF 001 - Registro y Autenticación	Seguridad	Seguridad de Autorización basada en JWT	El usuario se autentica con usuario y contraseña y el servidor retornará un token de tipo JWT que será enviado por el front en cada petición que haga al back y por medio del cual se autorizará al usuario solo a los recursos a que tiene acceso.
		Principio de menor privilegio	los usuarios y componentes de la aplicación solo deben tener acceso a los recursos y datos

			que necesitan para realizar sus funciones
		Cifrado de datos	La contraseña se almacenará cifrada con el algoritmo SHA-2 .
HU_RNF 002 - Trazabilidad	Seguridad	Registro de acciones	Se realizará el registro de las acciones de inserción, modificación y eliminación de datos.
		Metadatos en acciones	email de usuario, fecha y hora de la acción, tipo de acción (ejemplo, registro de colaboración, aprobación de demanda, etc.), datos originales (antes de ejecutar la acción), nuevos datos (después de ejecutar la acción)
		Generación de informes de trazabilidad	El sistema proveerá un informe de trazabilidad al cual solo puede acceder un usuario planeador
HU_RNF 003 - Respuesta rápida del sistema	Eficiencia y desempeño	Paginación de datos listados en tablas	Todas las tablas que listen información de la base de datos deben estar paginadas y las consultas a la base de datos debe hacerse de manera paginada.
		Caché	Las consultas que no requieran estar actualizadas en tiempo real deben ser guardadas en caché, por ejemplo, los informes de trazabilidad.
HU_RNF 004 - Facilidad de uso	Usabilidad	Diseño centrado en el usuario	
		Navegación sencilla	La navegación entre funcionalidades no debe ser compleja, el usuario debe encontrar todos los accesos a las funcionalidades a las que tiene permiso en el menú izquierdo.

		Retroalimentación inmediata	Cada vez que el usuario ejecute una acción el sistema debe mostrarle un spinner de loading hasta que la solicitud responda y un mensaje de la acción que acabó de ejecutar donde indique si fue satisfactoria o generó un error.

Fuente: Propio

3.3.16. Estrategia de Clean Code

La siguiente estrategia de Clean Code se aplicará en la construcción del código fuente con el fin de aplicar buenas prácticas en desarrollo de software:

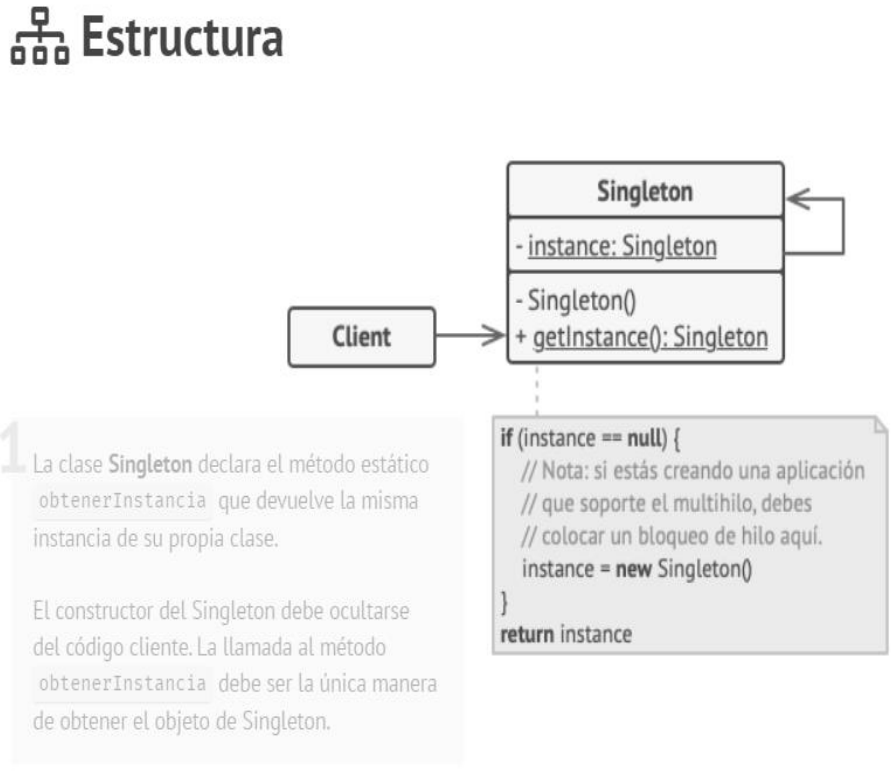
- Generalidades
  - Todo debe ser codificado en inglés.
  - Eliminar cualquier código no utilizado o comentariado.
  - No se debe usar valores numéricos o literales, en lugar de esto se debe guardar esos valores en constantes y hacer uso de ellas.
  - No se debe mantener comentarios innecesarios, por ejemplo, comentar la clase Product con el comentario //Product class ya que el nombre de la clase por sí sola da a entender que es la representación lógica de un producto de la vida real relacionado al dominio de la aplicación.
- Métodos
  - Los nombres de los métodos deben describir la funcionalidad del método por sí solos.
  - Los métodos asíncronos deben finalizar con la palabra Async (NombreDelMetodoAsync())
  - Un método no debe tener más de 20 líneas.
  - El nombre de los métodos se hace mediante el estándar Pascal case (la primera letra en mayúscula), ejemplo GetUserByIdAsync(id).
  - Nunca pueden existir más de dos ciclos anidados (uno dentro del otro), ejemplo: no puede existir un for dentro de otro for que a su vez está dentro de un tercer for.
  - El nombre de ningún método puede usar underscore, ejemplo GetUser_Async, en vez de eso debe ser GetUserAsync salvo en las pruebas unitarias.
  - Los parámetros de los métodos deben ser nombrados con camel case (la primera letra en minúscula).
  - Todo método público debe tener pruebas unitarias

- Variables:
  - Para nombrarlas se debe usar camel case, o sea la primera letra en minúscula, ejemplo productName.
  - No se debe usar el tipo dinámico “var” para declarar una variable, sino que se debe usar el tipo explicito, ejemplo: en vez de var amount usar int amount.
  - El nombre de la variable debe describir lo que guarda la variable por sí sola, ejemplo: para guardar el usuario que retorna un método la variable no se debe llamar User result = GetUser(id), sino User user = GetUser (id).
  - No se deben declarar variables que no se vayan a usar.
- Constantes
  - Los nombres de las constantes deben ser descriptivos
  - Los nombres de las constantes deben ser en mayúsculas sostenida y acepta underscore, ejemplo: NOMBRE_DE_CONSTANTE
- Clases
  - Los using deben estar en orden alfabético y se deben borrar los que no se usan
  - El nombre de la clase debe tener nomenclatura pascal case (La primera letra en mayúscula)
  - Las clases deben tener una única responsabilidad
  - Los nombres de las clases deben ser descriptivos

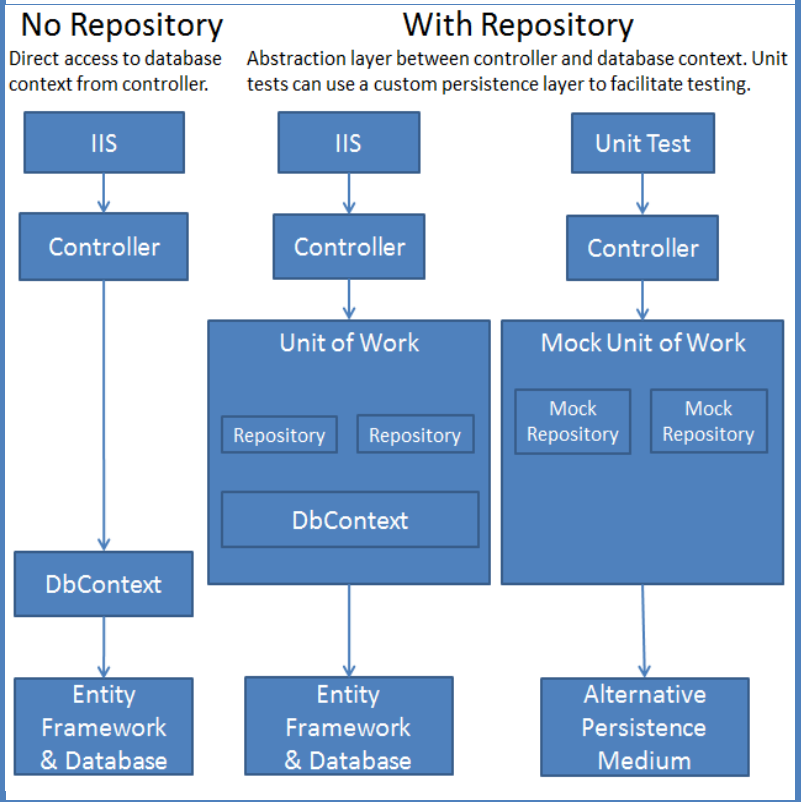
3.3.17. Patrones de diseño

Los siguientes patrones de diseño se implementarán en el desarrollo de la aplicación:

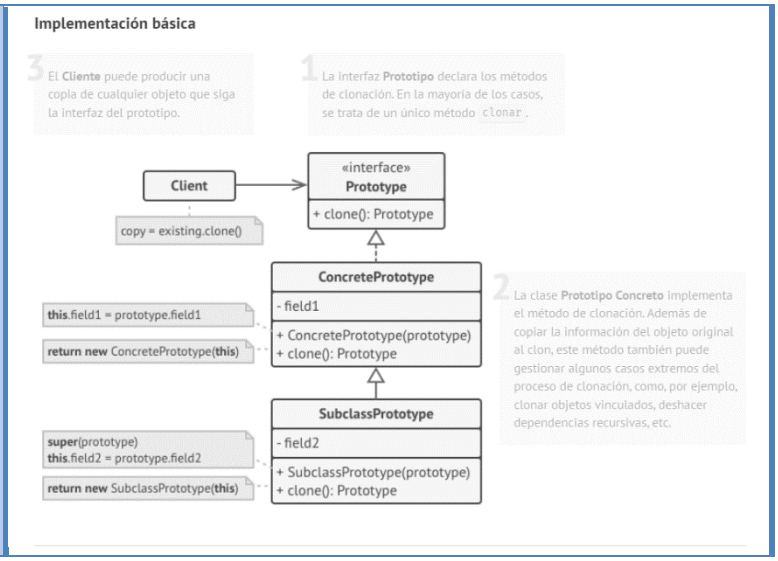
Tabla 22. Patrones de Diseño implementados en WA Collaborative

Nombre del patrón	Singleton
Descripción	Su función principal es crear una única instancia de un objeto y poder compartirla a través de la aplicación
Problema (pretende resolver con el patrón)	El Backend se busca crear una única instancia de la conexión a la base de datos transaccional y a la base de datos de solo lectura. En el Frontend se busca mantener una única instancia del http client con que se consumen los servicios del Backend.
Estructura	

Nombre del patrón	Repository and Unit Of Work
Descripción	Su función principal es separar la lógica de acceso a datos para que sea independiente (repository) y mantener las

	transacciones a la base de datos en una única unidad de trabajo para mantener la consistencia de estas (unit of work)
Problema	Responsabilidad unida de la capa de acceso a datos y consistencia en la información que se envía a la base de datos
Estructura	

Nombre del patrón	Prototype
Descripción	Permite copiar objetos existentes sin que el código dependa de sus clases.
Problema	Al momento copiar un producto para crear uno igual al original, pero poderle cambiar únicamente algunos pocos atributos

Estructura	

Fuente: Propio

3.4. Calidad de software

3.4.1. Matriz de Riesgo de Software

La siguiente es la matriz de riesgos del producto:

Nomenclatura para matriz de riesgos:

AC = Atributo de calidad

RF = Requisito funcional

Tabla 23. Matriz de Riesgo para WA Collaborative

	Tipo de requisito	**Consecuencia**	**Causa (raíz)**	**Probabilidad**	**Impacto**	**Mitigación**
1	AC	Funcionalidad: Funcionalidad limitada o inexistente	Requisitos mal entendidos o no documentados	Alta	Alto	Realizar una revisión exhaustiva de los requisitos con los stakeholders

2	AC	Fiabilidad: Fallas frecuentes o errores en el software	Falta de pruebas exhaustivas	Media	Alto	Implementar pruebas automatizadas y realizar pruebas de estrés
3	AC	Usabilidad: Interfaz de usuario confusa o difícil de usar	Falta de diseño centrado en el usuario	Baja	Bajo	Realizar pruebas de usabilidad con usuarios reales y recopilar retroalimentación
4	AC	Eficiencia en Desempeño: Rendimiento lento o recursos excesivamente utilizados	Codificación ineficiente o algoritmos complejos	Media	Medio	Realizar pruebas de rendimiento y verificar que ante altos niveles de estrés el software siga respondiendo correctamente.
5	AC	Mantenibilidad: Dificultad para realizar cambios o agregar nuevas funcionalidades	Código mal estructurado o documentación insuficiente	Alta	Medio	Verificar y validar que el software sea modular
6	AC	Portabilidad: Incompatibilidad con ciertas plataformas o sistemas operativos	Dependencia de características específicas de la plataforma	Baja	Medio	Verificar y validar el correcto funcionamiento del software en los navegadores chrome, opera y edge.
7	AC	Compatibilidad:	Falta de pruebas de	Baja	Medio	Realizar pruebas de integración con sistemas externos y

		Interoperabilidad limitada con otros sistemas o software	integración adecuadas			seguir estándares de interoperabilidad
8	AC	Seguridad: Vulnerabilidades de seguridad o brechas de datos	Falta de prácticas de desarrollo seguro	Media	Alto	Verificar y validar que el software siga las recomendaciones de OWASP
9	RF	Un usuario colaborador pueda aprobar el plan de demanda	No se manejan adecuadamente los permisos de usuario ya que el único que puede aprobar la demanda es el usuario planeador	Baja	Alto	Hacer pruebas exhaustivas a los permisos que tiene un cada rol de usuario en cada módulo de la aplicación
10	RF	Los usuarios puedan colaborar en la demanda por fuera de las fechas del calendario de colaboración	No se hacen correctamente las validaciones de fecha inicio y fecha final que estipula el calendario de colaboración para poder colaborar	Media	Alto	Crear test unitarios que identifiquen cuando la validación de las fechas está mal. Generar un set de pruebas que incluyan la validación de las fechas.

11	RF	Los usuarios puedan colaborar en una demanda que fue aprobada	No se realizan las respectivas validaciones en el código ya que una demanda no se puede colaborar cuando ya está aprobada	Baja	Alto	Crear test unitarios para alertar cuando algún cambio en el código permita la colaboración de una demanda en estado aprobado. Generar un set de pruebas que incluyan la validación de estados de la demanda.
12	RF	Los usuarios puedan colaborar en productos y clientes que no estén dentro de su portafolio	No se realizan las validaciones necesarias para que los usuarios solo puedan colaborar en productos y clientes de su portafolio.	Baja	Alto	Crear test unitarios para alertar cuando algún cambio en el código permita la colaboración en clientes y productos no relacionados al portafolio del usuario. Generar un set de pruebas que incluyan esta validación.
13	RF	Un usuario colaborador pueda crear un portafolio y asignarlo a un usuario	Falta de las respectivas validaciones a nivel de código y falta de entendimiento de los requerimient	Baja	Alto	Crear test unitarios que blinden el código. Generar un set de pruebas que incluyan esta validación.

			os ya que solo el usuario planeador puede crear el portafolio			
14	RF	Un usuario colaborador pueda crear un calendario de colaboración	Falta de las respectivas validaciones a nivel de código y falta de entendimiento de los requerimientos ya que solo el usuario planeador puede crear el calendario de colaboración	Baja	Alto	Crear test unitarios que blinden el código. Generar un set de pruebas que incluyan esta validación.
15	RF	Un usuario inactivo pueda autenticarse	Mal manejo de autenticaciones y autorizaciones	Baja	Alto	Set de pruebas exhaustivas para la funcionalidad
16	RF	Un usuario colaborador pueda crear registros en tablas de datos maestros como	Falta de entendimiento de los requisitos funcionales. Mal manejo de autorizacione	Baja	Alto	Claridad y especificad en los criterios de aceptación de las historias de usuario respecto al nivel de autorización de

		productos, clientes, etc. Ya que para esto solo está autorizado el usuario planeador.	s a nivel de código.			cada elemento del software. Set de pruebas enfocado al nivel de autorización.

Fuente: Propio

3.4.2. Proceso de no conformidad

Cuando se identifique una no conformidad durante cualquier etapa del proyecto, se seguirá un proceso formalizado para su manejo:

- Identificación: Cualquier miembro del equipo puede identificar una no conformidad, ya sea durante la ejecución de pruebas, revisiones de código, o cualquier otra actividad relacionada con el desarrollo del software.

- Registro: La no conformidad será registrada en un sistema de seguimiento de problemas o en una herramienta de gestión de calidad designada, junto con detalles específicos sobre la naturaleza de la no conformidad, su ubicación en el producto o proceso, y cualquier otra información relevante.

- Investigación: Se llevará a cabo una investigación exhaustiva para determinar las causas subyacentes de la no conformidad. Esto puede implicar la revisión de registros de actividad, entrevistas con miembros del equipo involucrados y análisis de datos relevantes.

- Acciones Correctivas: Se desarrollarán y ejecutarán acciones correctivas para abordar la no conformidad identificada. Estas acciones pueden incluir la corrección de errores en el código, la actualización de documentación, la revisión de procesos o la capacitación del personal, según sea necesario.

- Verificación: Una vez implementados las acciones correctivas, se verificará su efectividad para garantizar que la no conformidad se resuelva satisfactoriamente. Esto puede implicar la realización de pruebas adicionales, revisiones de código o cualquier otra actividad de verificación pertinente.

- Cierre: Una vez verificado que la no conformidad se ha abordado satisfactoriamente, se cerrará el registro correspondiente en el sistema de seguimiento de problemas.

Se documentarán todas las acciones tomadas y los resultados de la verificación para futuras referencias y lecciones aprendidas.

3.5. Scrum

3.5.1. Roles

A continuación, se define los roles que se elaboran para la implementación del proyecto bajo la metodología Scrum.

Tabla 24. Definición de Roles para la Metodología Scrum

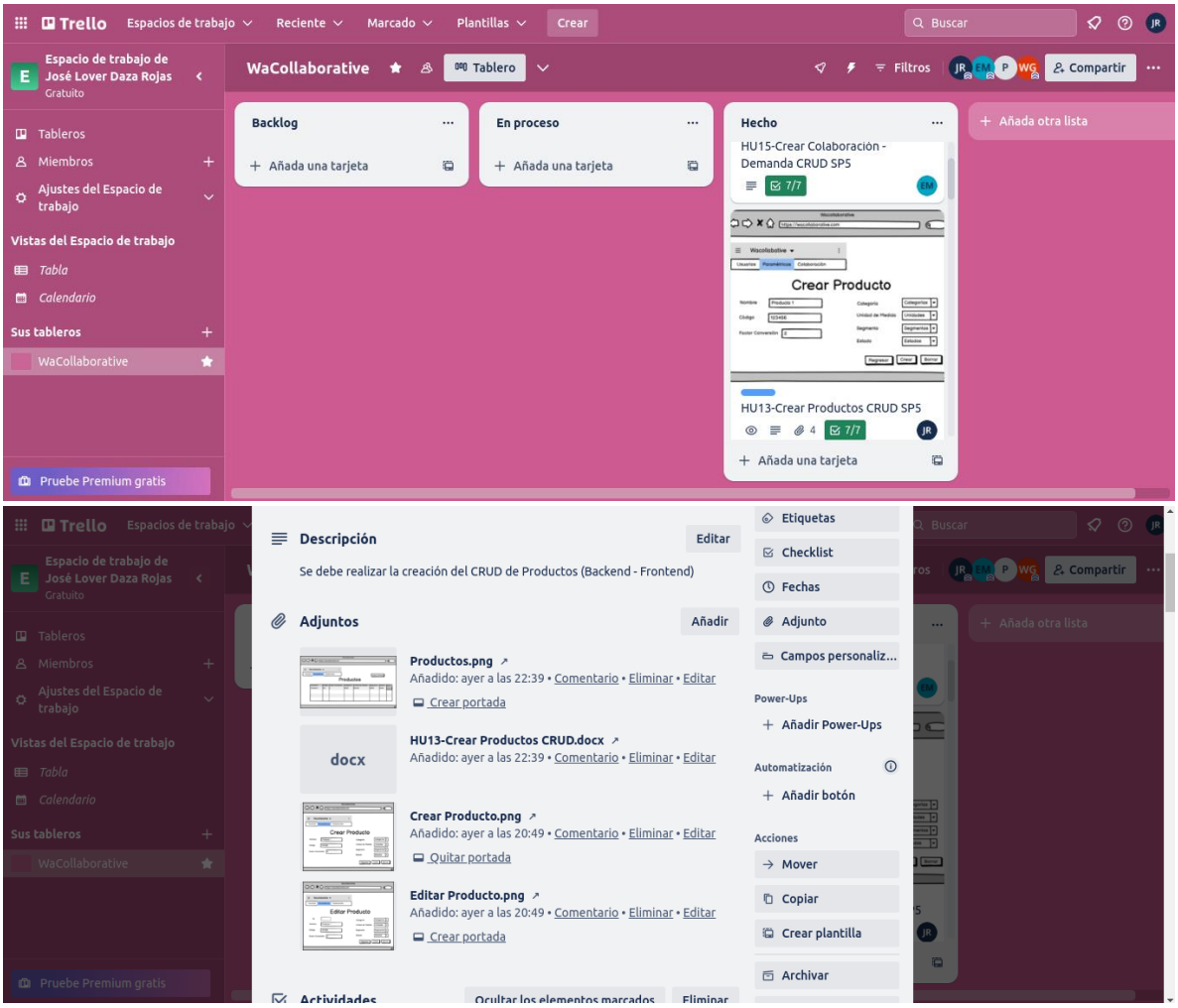
Persona	Contacto	Rol
Walter Antonio Morales Gallego	waltermorales6186@correo.itm.edu.co	Coordinador/ Scrum Master
Jose Lover Daza Rojas	josedaza1121437@correo.itm.edu.co	Desarrollador / Analista de Requerimientos
Efraín Trujillo Malaver	efraintrujillo1125059@correo.itm.edu.co	Desarrollador / Tester

Fuente: Propia

3.5.2. Cronograma y tablero scrum

A continuación, se muestra el tablero en Trello, donde se crearon las actividades para cada sprint, el cual se defino que tiene una duración de dos semanas por sprint.

Figura 17. Tablero Scrum para gestión por Sprint



Fuente: Propia

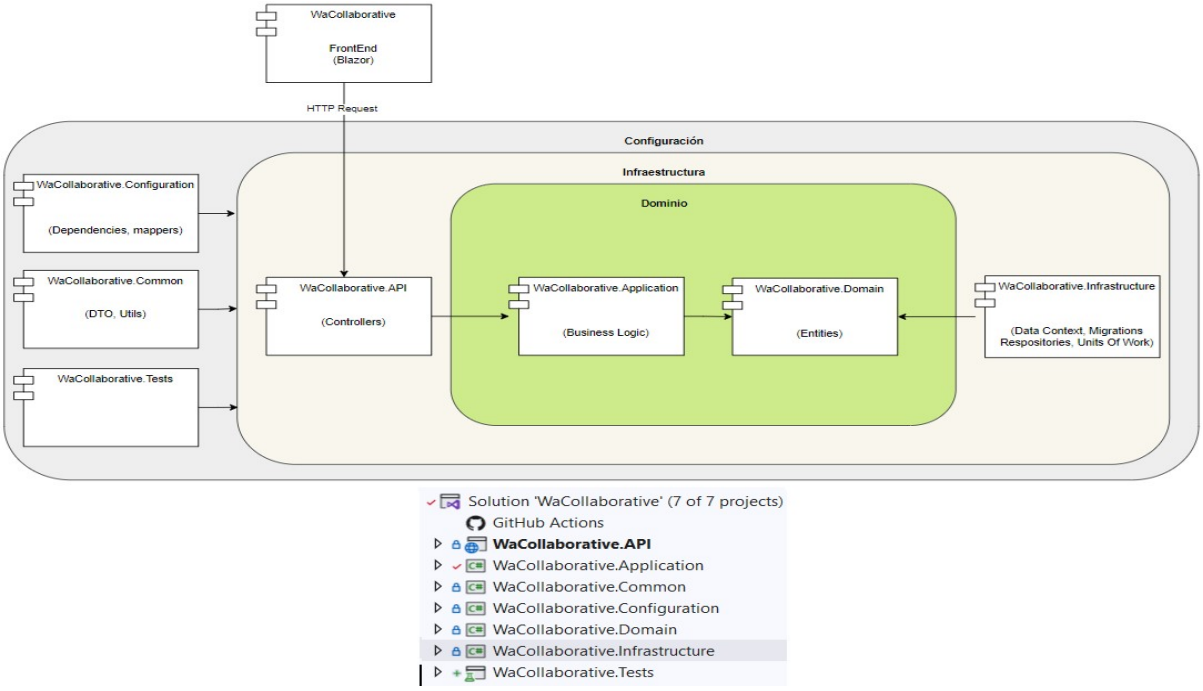
4. RESULTADOS Y DISCUSIÓN

4.1. Diseño de la Implementación

4.1.1. Implementación de arquitectura limpia Backend

En el siguiente comparativo se evidencia como se aplica la arquitectura limpia al proyecto del Backend

Figura 18. Implementación de arquitectura Limpia para el Backend



Fuente: Propia

A través de la siguiente imagen se evidencia que el código fuente está estructurado para que un controlador consuma un service y este un repository, por ejemplo, el ProductsController consume el ProductService y este consume el ProductRepository, todo mediante el uso de inyección de dependencias a través del controlador, esto lleva a los siguientes puntos muy importantes dentro de la arquitectura:

- Se hace uso de los principios SOLID responsabilidad única, abierto/cerrado, segregación de interfaces, inversión de dependencias; esto se describe más detalladamente en la tabla “Principios SOLID” de este documento.
- Se hace uso de los principios GRASP bajo acoplamiento, alta cohesión, experto, creador, controlador, variaciones protegidas; esto se describe más detalladamente en la tabla “Principios GRASP” de este documento.

- Se hace uso de la arquitectura orientada a servicios SOA, lo cual se evidencia cuando se hace uso de la data anotation "[ApiController]" el cual expone el controlador como un servicio de tipo Api Rest., lo cual va acorde a lo expuesto en los estilos arquitectónicos del presente documento

Figura 19. Ejemplo de Implementación de código en Backend

```
namespace WaCollaborative.API.Controllers.Parameters
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class ProductsController : BaseController<ProductDTO>
    {
        private IProductsService _ProductsService;

        0 references
        public ProductsController(IProductsService ProductsService) : base(ProductsService)
        {
            _ProductsService = ProductsService;
        }
    }
}
```

```
namespace WaCollaborative.Application.Services
{
    2 references
    public class ProductsService : BaseService<ProductDTO, Product>, IProductsService
    {
        private IProductsRepository _ProductsRepository;
        private IMapper _mapper;

        0 references
        public ProductsService(
            IBaseRepository<Product> repository,
            IMapper mapper,
            IProductsRepository ProductsRepository)
            : base(repository, mapper)
        {
            _ProductsRepository = ProductsRepository;
            _mapper = mapper;
        }
    }
}
```

```
namespace WaCollaborative.Infrastructure.Repositories
{
    2 references
    public class ProductsRepository : BaseRepository<Product>, IProductsRepository
    {
        0 references
        public DataContext Context
        {
            get
            {
                return (DataContext)_context;
            }
        }

        0 references
        public ProductsRepository(DataContext Context) : base(Context)
        {
        }
    }
}
```

Fuente: Propia

La siguiente imagen muestra la creación de un repositorio genérico, que a su vez es heredado por el repositorio de la entidad "Product" lo cual va acorde con los siguientes puntos de la arquitectura:

- Se hace uso del principio SOLID Liskov; esto se describe más detalladamente en la tabla "Principios SOLID" de este documento.

- Se hace uso del principio GRASP polimorfismo; esto se describe más detalladamente en la tabla "Principios GRASP" de este documento.

- Se hace uso de los patrones de diseño “Singleton” y “Repository and Unit Of Work”; esto se describe más detalladamente en el aparte de “Patrones de diseño” de este documento.

Figura 20. Ejemplo de Implementación de código genérico en Backend

```
namespace WaCollaborative.Infrastructure.Repositories
{
    22 references
    public interface IBaseRepository<TEntity>
    {
        7 references
        Task<TEntity> GetAsync(int id);

        7 references
        Task<IEnumerable<TEntity>> GetAsync();

        3 references
        Task<TEntity> AddAsync(TEntity entity);

        3 references
        Task DeleteAsync(int id);

        3 references
        Task<TEntity> UpdateAsync(TEntity entity);
    }
}
```

```
namespace WaCollaborative.Infrastructure.Repositories
{
    using Microsoft.EntityFrameworkCore;
    using WaCollaborative.Infrastructure.Data;

    18 references
    public class BaseRepository<TEntity> : IBaseRepository<TEntity> where TEntity : class
    {
        public DataContext _context;
        protected DbSet<TEntity> _entity;

        8 references
        public BaseRepository(DataContext context)
        {
            _context = context;
            _entity = context.Set<TEntity>();
        }

        3 references
        public virtual async Task<TEntity> AddAsync(TEntity entity)
        {
            _context.Add(entity);

            await _context.SaveChangesAsync();
            return entity;
        }
```

```
using WaCollaborative.Domain.Entities;
using WaCollaborative.Infrastructure.Data;

namespace WaCollaborative.Infrastructure.Repositories
{
    2 references
    public class ProductsRepository : BaseRepository<Product>, IProductsRepository
    {
        0 references
        public DataContext Context
        {
            get
            {
                return (DataContext)_context;
            }
        }

        0 references
        public ProductsRepository(DataContext Context) : base(Context)
        {
        }
    }
}
```

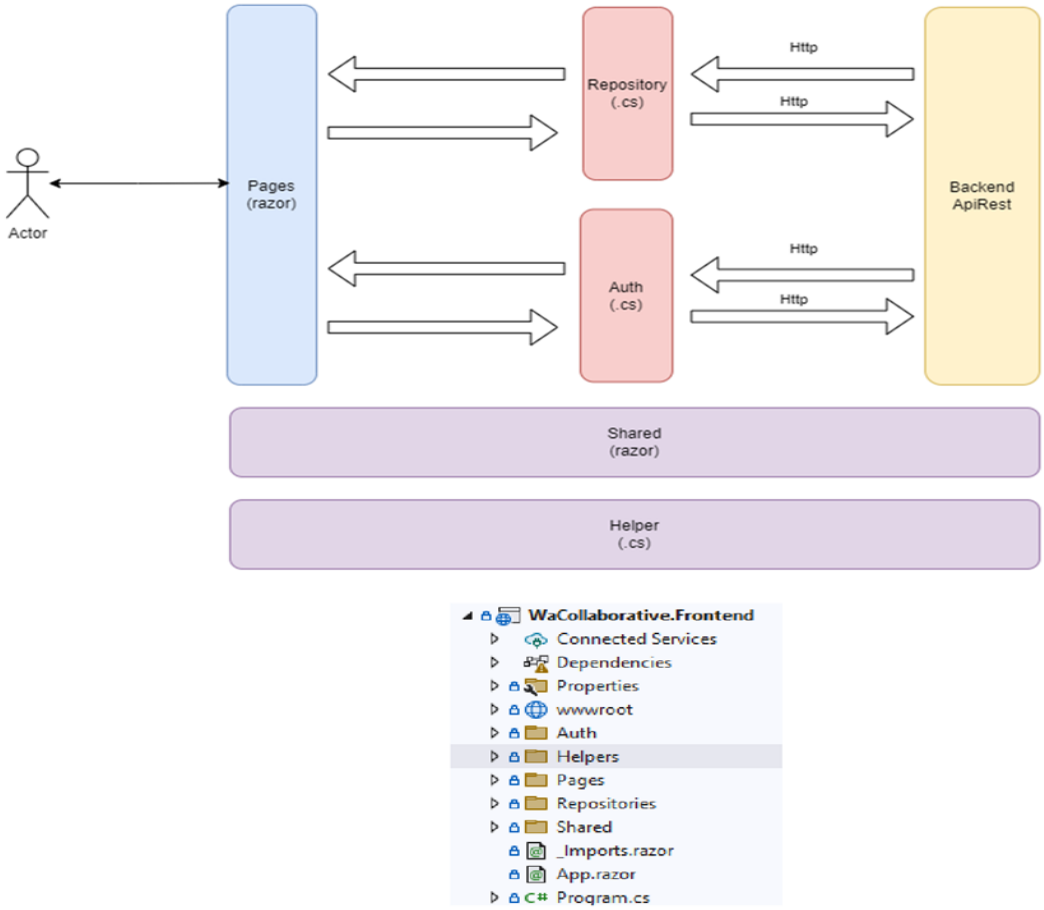
Fuente: Propia

Las anteriores imágenes representan los aspectos más importantes para tener en cuenta en el CRUD de “Product”.

4.1.2. Implementación de arquitectura limpia Frontend

En las siguientes imágenes se evidencia la implementación de la arquitectura limpia del proyecto Frontend realizado en Blazor, está la estructura de carpetas que garantiza el modelo de arquitectura.

Figura 21. Implementación de Arquitectura en Frontend

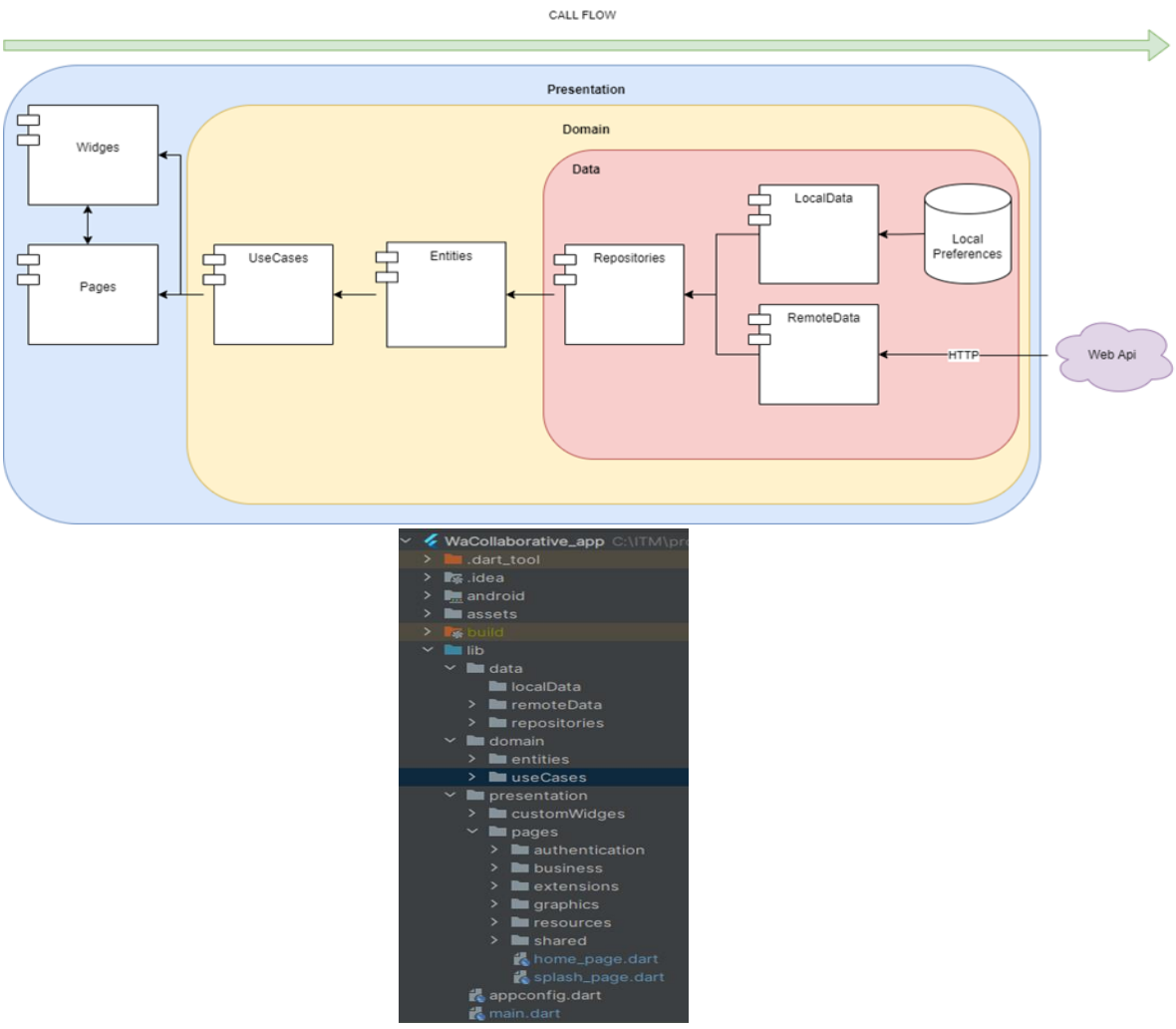


Fuente: Propia

4.1.3. Implementación arquitectura limpia aplicación móvil

En las siguientes imágenes se evidencia la implementación de la arquitectura limpia del proyecto de la aplicación móvil realizado en Flutter, está la estructura de carpeta que garantiza el modelo de arquitectura.

Figura 22. Implementación de Arquitectura en Frontend como Aplicación Móvil

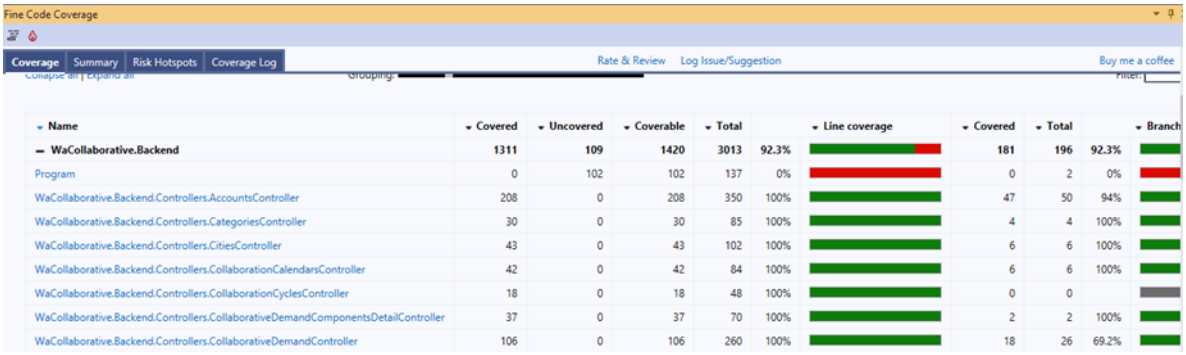


Fuente: Propia

4.1.4. Resultado pruebas unitarias

Se obtuvo un 92.3% del código que se puede probar con una cobertura por las pruebas unitarias:

Figura 23. Resultados de las pruebas unitarias



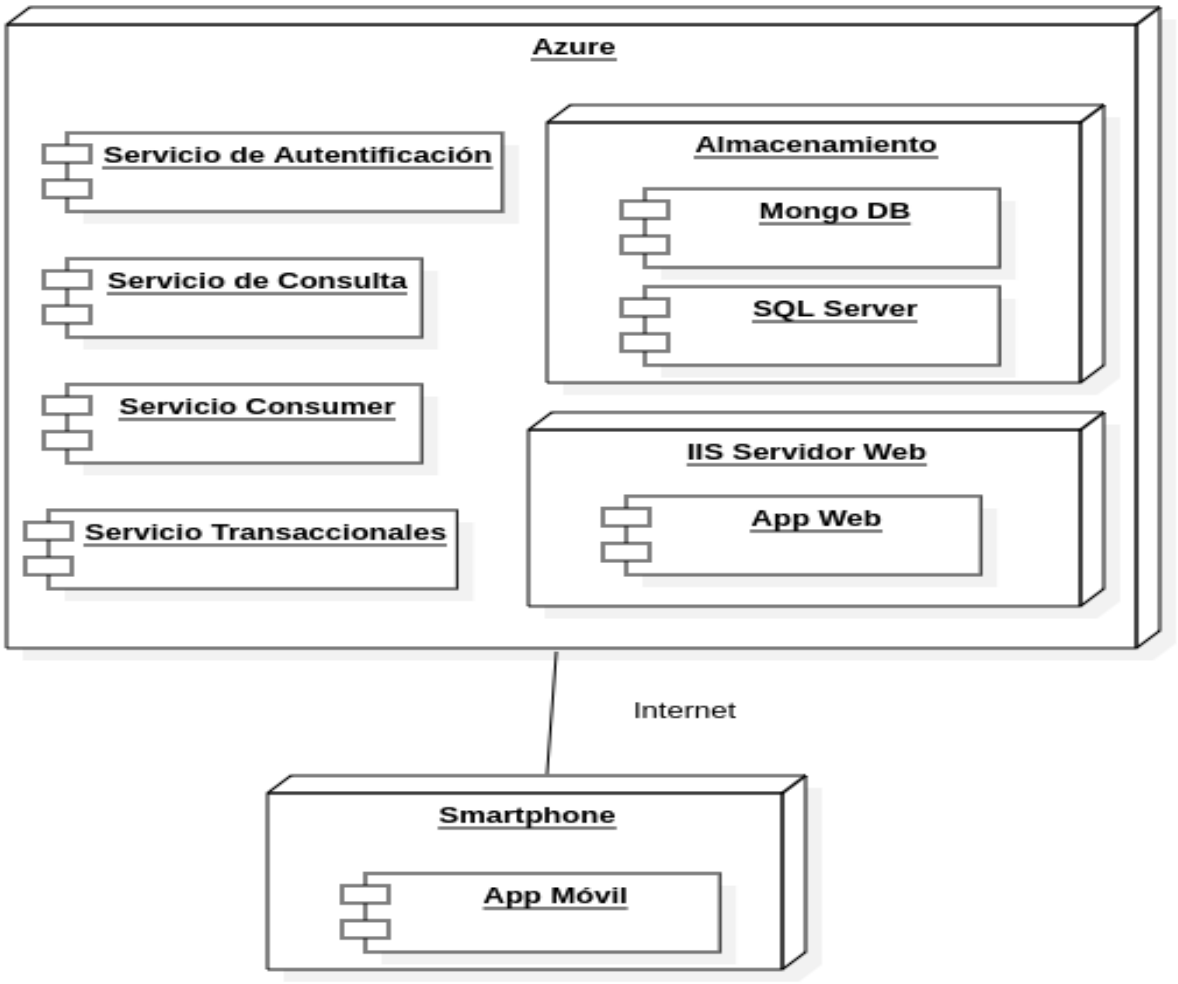
Name	Covered	Uncovered	Coverable	Total		Line coverage	Covered	Total		Branch
WaCollaborative.Backend	1311	109	1420	3013	92.3%		181	196	92.3%	
Program	0	102	102	137	0%		0	2	0%	
WaCollaborative.Backend.Controllers.AccountsController	208	0	208	350	100%		47	50	94%	
WaCollaborative.Backend.Controllers.CategoriesController	30	0	30	85	100%		4	4	100%	
WaCollaborative.Backend.Controllers.CitiesController	43	0	43	102	100%		6	6	100%	
WaCollaborative.Backend.Controllers.CollaborationCalendarsController	42	0	42	84	100%		6	6	100%	
WaCollaborative.Backend.Controllers.CollaborationCyclesController	18	0	18	48	100%		0	0		
WaCollaborative.Backend.Controllers.CollaborativeDemandComponentsDetailController	37	0	37	70	100%		2	2	100%	
WaCollaborative.Backend.Controllers.CollaborativeDemandController	106	0	106	260	100%		18	26	69.2%	

Fuente: Propia

4.1.5. Diagrama de Despliegue para el Backend

Para la implementación del proyecto se utilizó como infraestructura en la nube Azure, para alojar los componentes desarrollados, servicio en la nube que brinda mantenimiento, respaldo y soporte automatizado.

Figura 24. Diagrama de despliegue Backend – Frontend



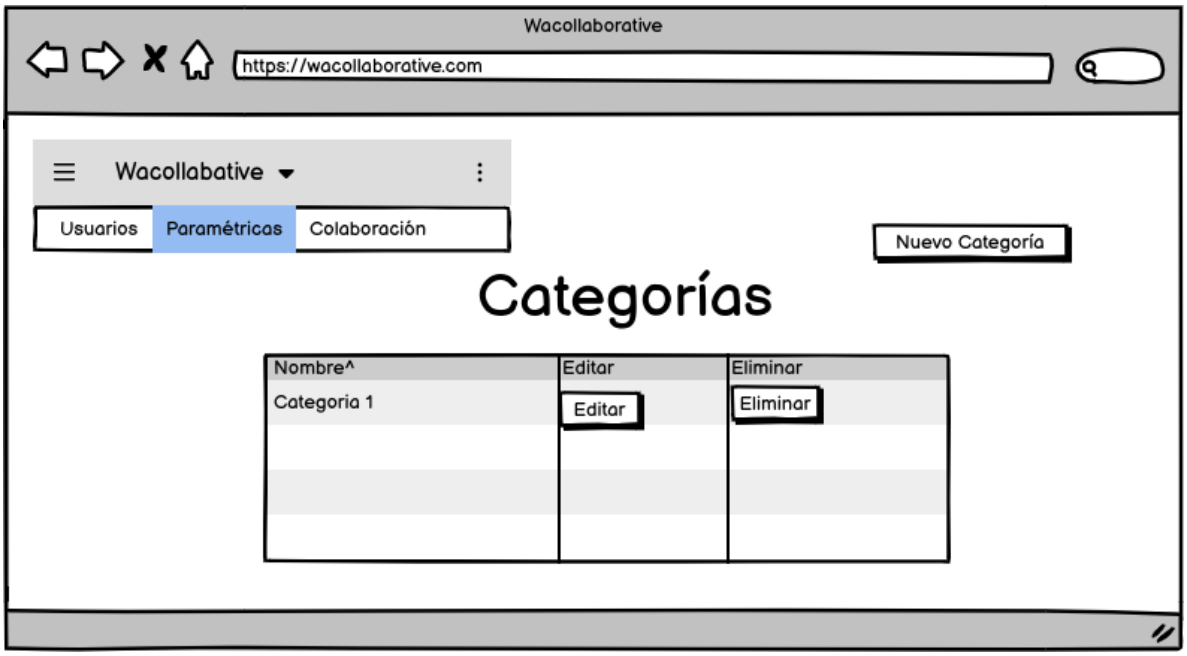
Fuente: Propia

4.1.6. Mockup de Pantallas

Aplicación WEB

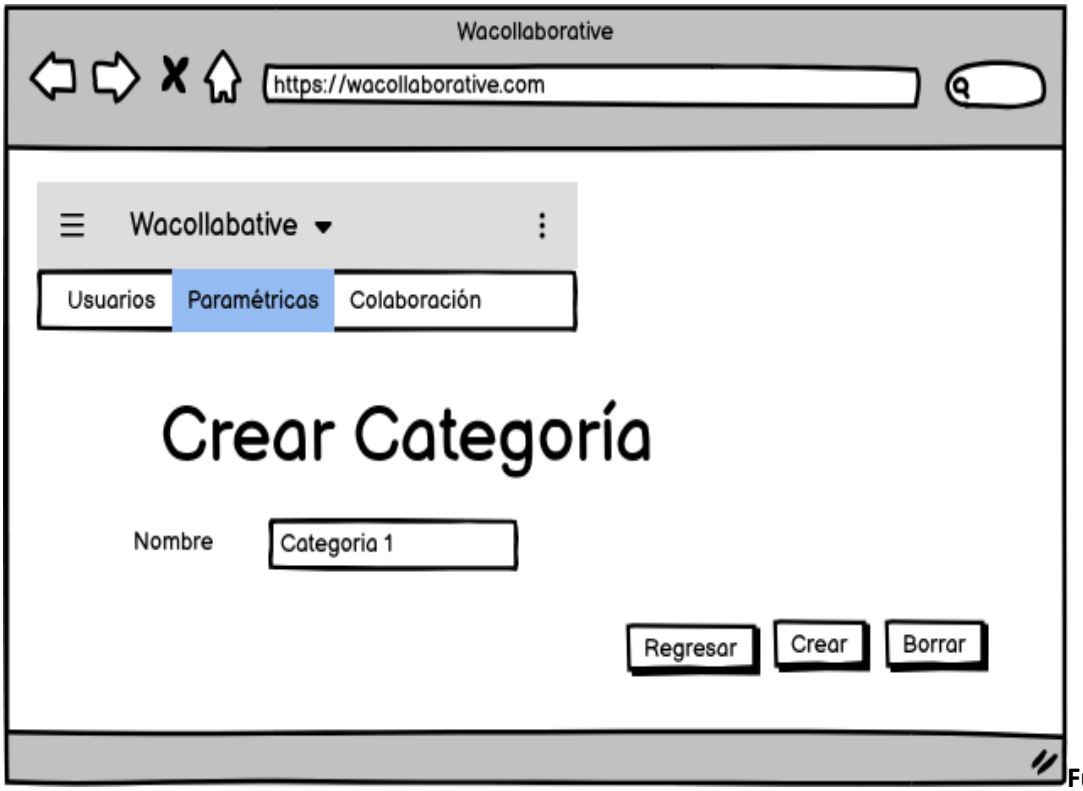
A continuación, e muestra mockup de pantallas para el aplicativo WEB, dando una visión de la pantalla final.

Figura 25. Mockup para el diseño de la Pantalla de Gestionar Categorías



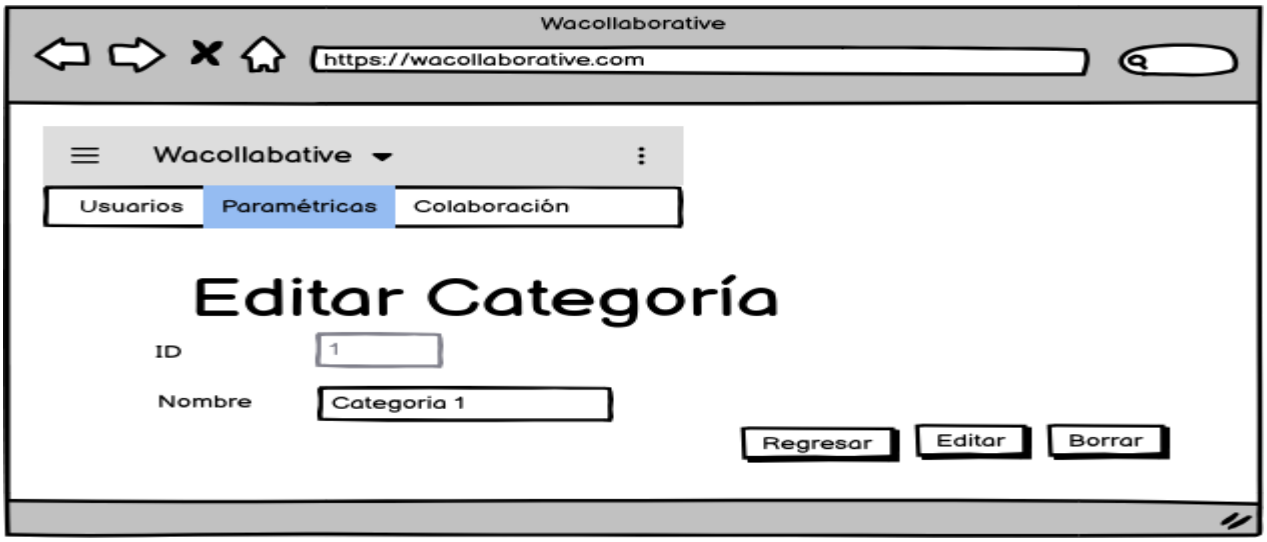
Fuente: Balsamiq

Figura 26. Mockup para el diseño de la Pantalla de Crear Categoría



Fuente: Balsamiq

Figura 27. Mockup para el diseño de la Pantalla de Editar Categoría



Fuente: Balsamiq

Figura 28. Mockup para el diseño de la Pantalla de Gestionar Productos

Fuente: Balsamiq

Figura 29. Mockup para el diseño de la Pantalla de Crear Productos

Fuente: Balsamiq

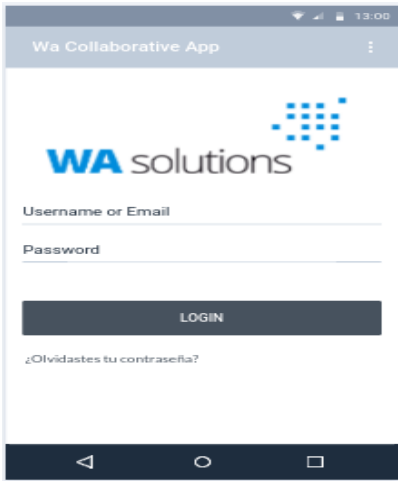
Figura 30. Mockup para el diseño de la Pantalla de Editar Productos

Fuente: Balsamiq

Aplicación Móvil

A continuación, se muestra mockup de pantallas para el aplicativo Movil, dando una visión de la pantalla final.

Figura 31. Pantalla del ingreso a la aplicación móvil

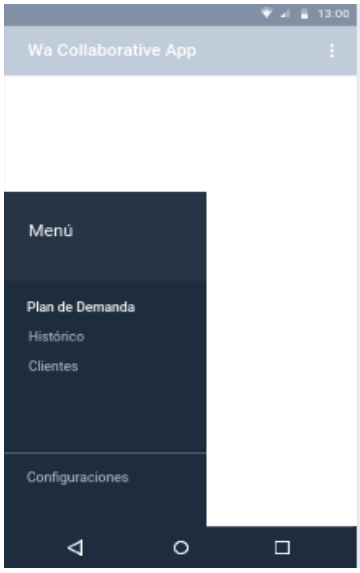


Fuente Marvel App

Figura 32. Pantalla de recuperación de contraseña

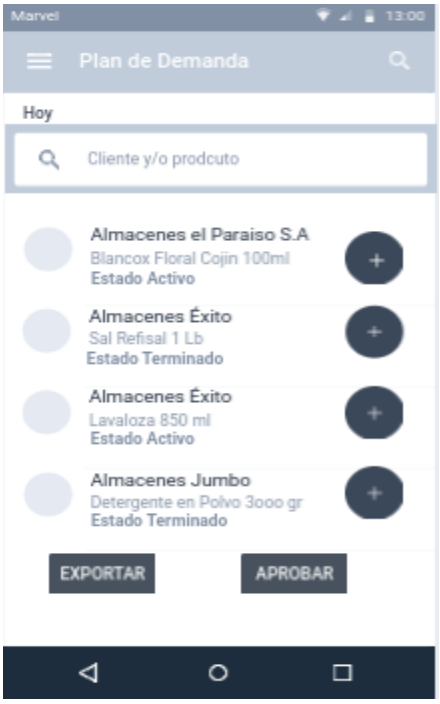
Fuente Marvel App

Figura 33. Pantalla de Menú



Fuente Marvel App

Figura 34. Pantalla de Plan de Demanda



Fuente Marvel App

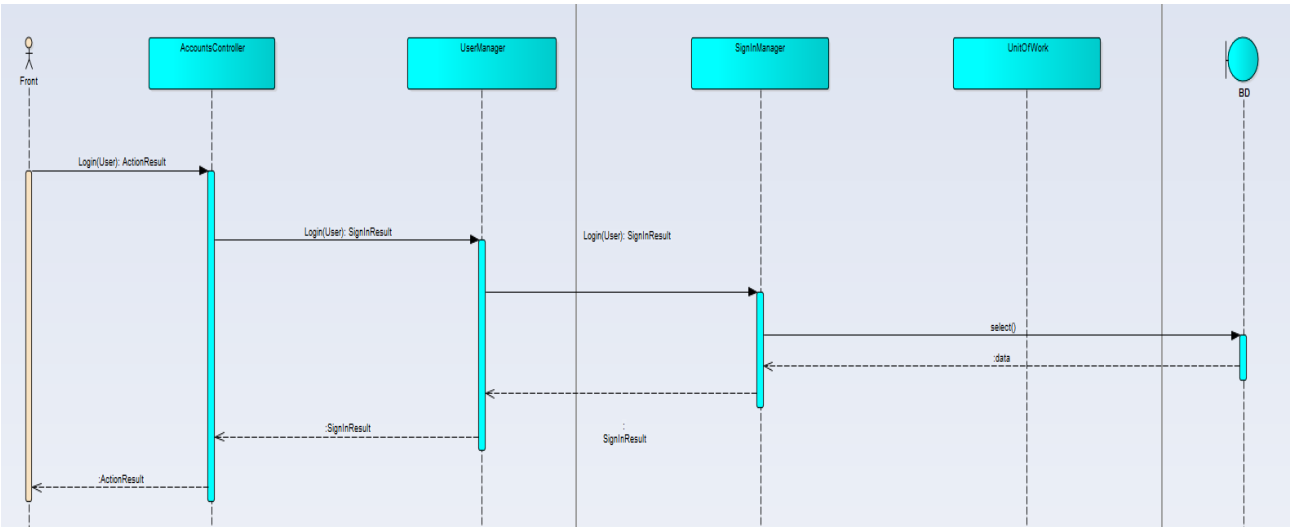
Figura 35. Pantalla de Colaboración

Fuente Marvel App

4. 1.2 Diagramas de Secuencia

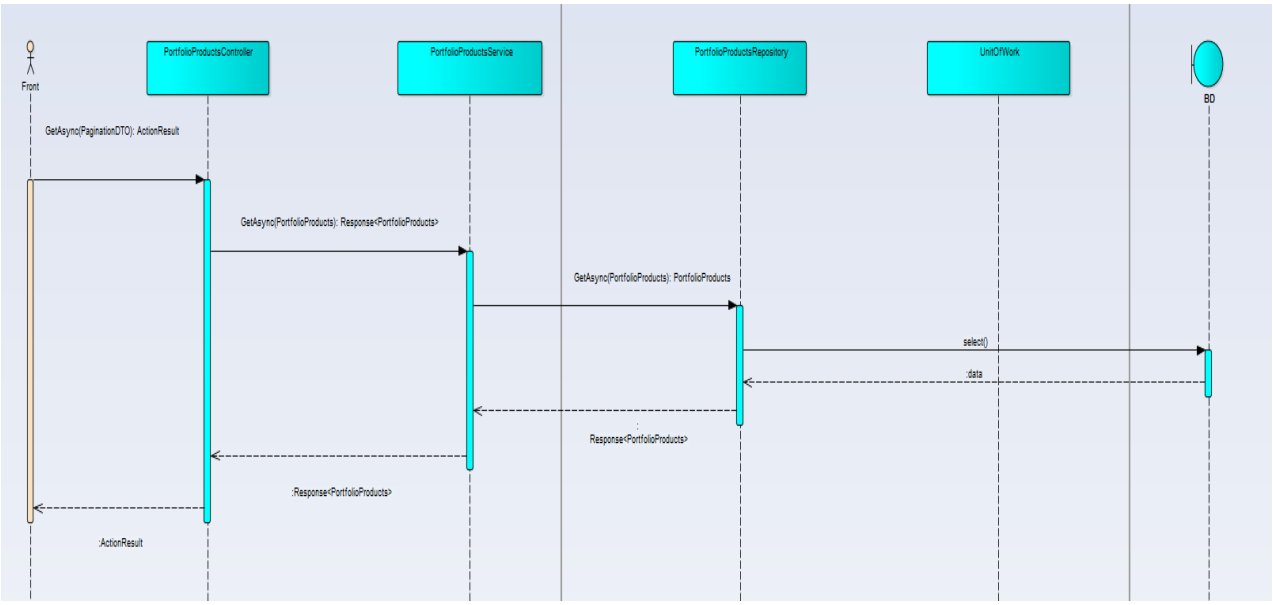
A continuación, se muestra algunos diagramas de secuencia de funcionalidades que se implementaron.

Figura 36. Diagrama de Secuencia Login



Fuente: Propia

Figura 37. Diagrama de Secuencia Portafolios por Productos



Fuente: Propia

4. 2 Implementación

A continuación, se muestra algunas pantallas del prototipo de la aplicación móvil y web de funcionalidades que se implementaron.

Figura 38. Pantalla de Login

Fuente: Propia

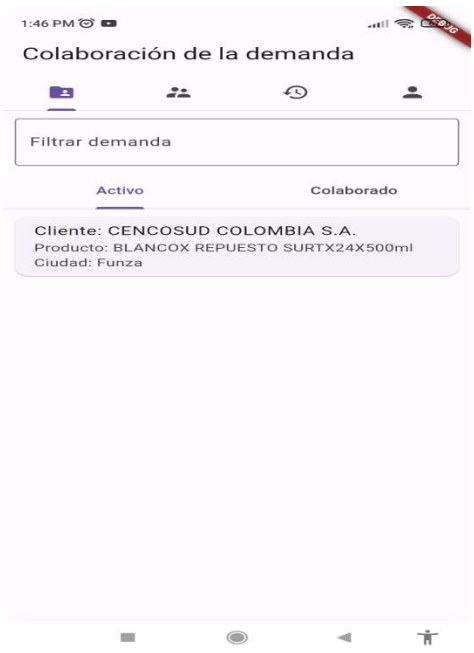
Figura 39. Pantalla de Recuperación de Contraseña

Fuente: Propia

Figura 40. Pantalla de Cambio de contraseña

Fuente: Propia

Figura 41. Pantalla de Colaboraciones asignadas a un Usuario



Fuente: Propia

Figura 42. Pantallazo de Historial de ventas de un Usuario

Fuente: Propia

Figura 43. Registrar demanda colaborada

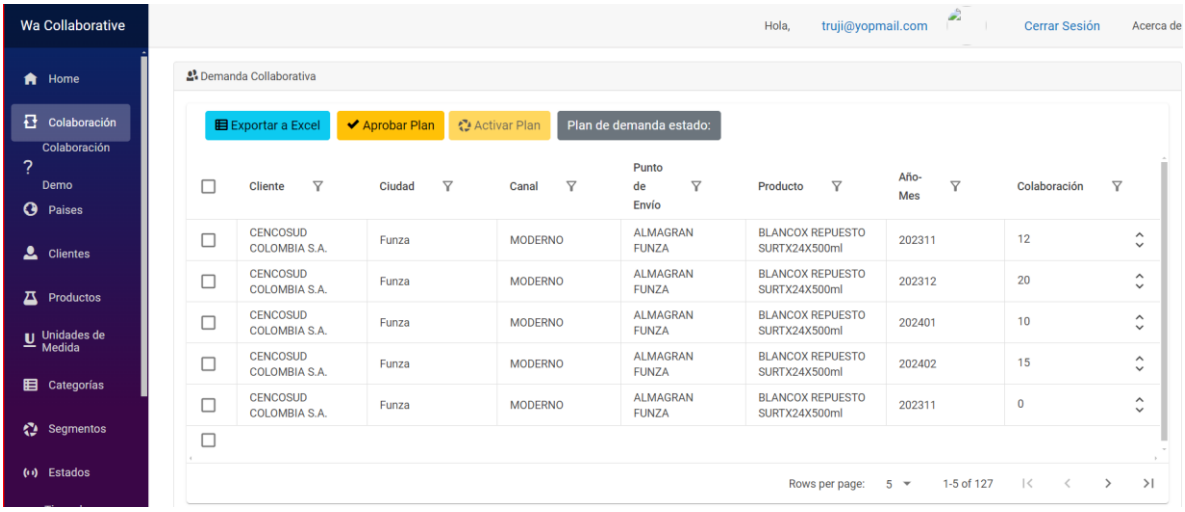


Figura 44. Menú y pantalla de inicio

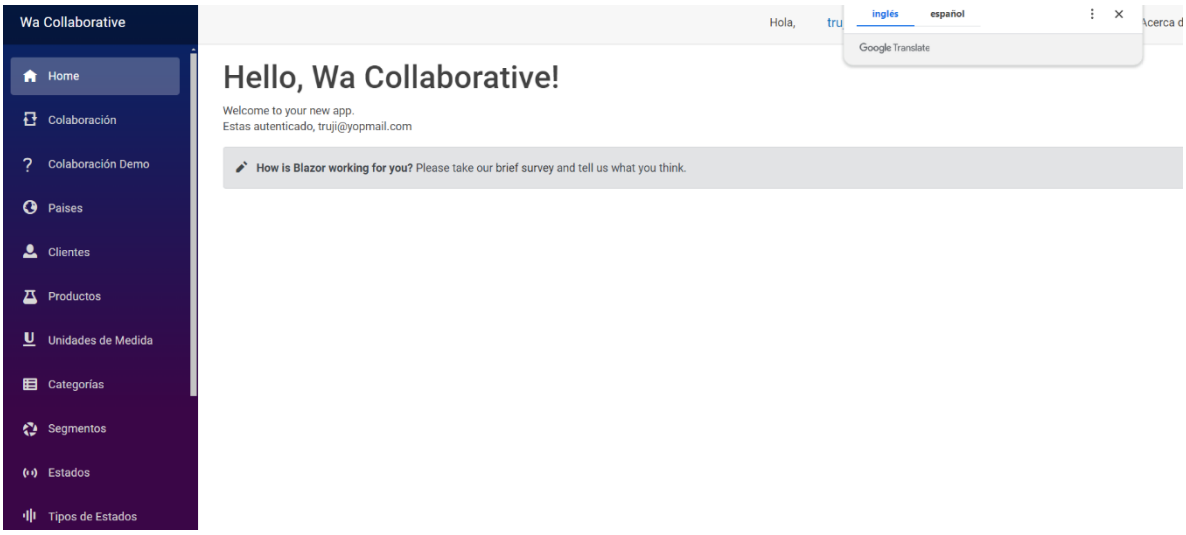
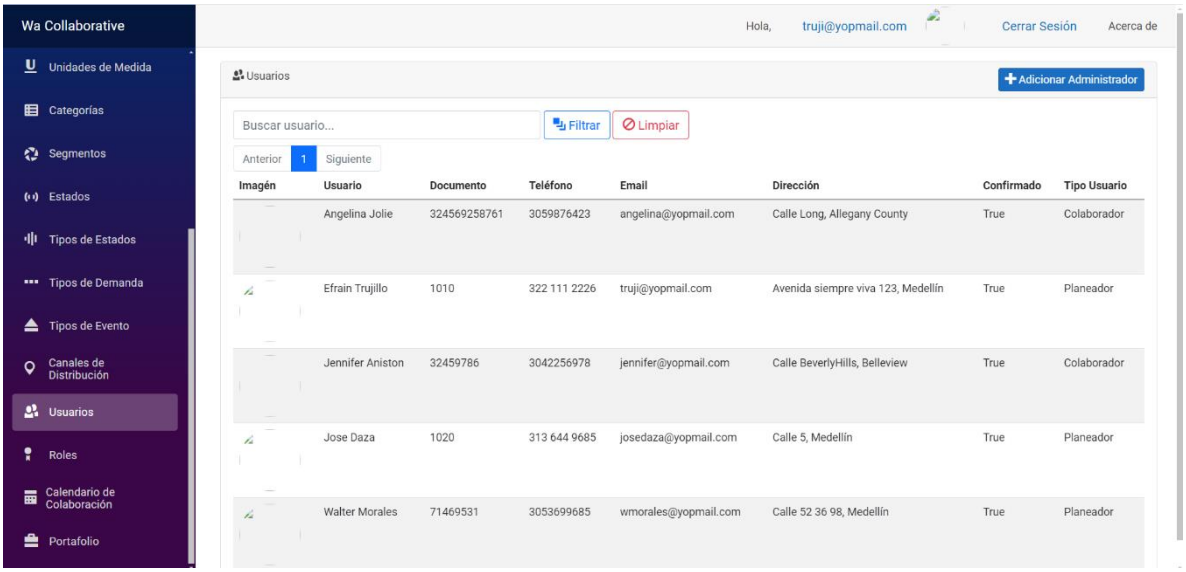


Figura 45. Administración de usuarios



5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

5. 1 Conclusiones

Mediante el desarrollo del proyecto, se entendió la importancia del diseño arquitectónico que integre diferentes componentes de software que se utilizan para la implementación de un conjunto de aplicaciones, como es el caso se integró una aplicación web y una aplicación móvil, que son aplicaciones que presentan información al usuario final (Frontend). Para el Backend se realizó la integración de un proyecto API Rest que realiza la tarea de la interacción con la base de datos para registrar, crear, modificar o eliminar datos bajo validación de autorización en cada operación.

La definición de la arquitectura cumple con la necesidad de la empresa, entendiendo que el diseño anterior posee fallas técnicas y falta de estabilidad, generando la inviabilidad de disminuir los tiempos para acoger nuevos clientes que utilicen la aplicación web. Además, de poseer problemas de seguridad al tener la lógica por Base de Datos, generando el riesgo de que los clientes tenga conocimiento del negocio junto a reglas de negocio, donde puede ser modificada, eliminada o copiada. Con una arquitectura limpia se detalla el interés de

aplicar las mejores técnicas para la construcción de software, disminuir los problemas que genera la interacción de la actual arquitectura que se posee, mejorando su estabilidad, adaptabilidad y mantenimiento. La aplicación móvil como una nueva herramienta que apoye a los colaboradores para capturar información desde cualquier parte apoya a las estrategias organizacionales, permitiendo generar un impacto positivo en el nicho de mercado donde está la empresa.

Con el resultado del diseño de una arquitectura y aplicación móvil como un prototipo generado del proyecto, se da una herramienta a la fuerza comercial para gestionar sus actividades diarias y eliminando el retraso de la captura de información, logrando el posicionamiento de la empresa en el mercado, incorporando tecnología para automatizar procesos disminuyendo la carga laboral de los colaborares de la empresa para la toma de decisiones en la gestión de la demanda colaborada.

5. 2 Recomendaciones y Trabajos Futuros

Una vez que se finalizó la implementación de este proyecto y habiendo definido sus conclusiones, se plantean las siguientes consideraciones:

Para trabajos futuros, se debe implementar la arquitectura propuesta en este documento, para ello se entregan los diseños arquitectónicos correspondientes, para facilitarlo. Adicionalmente, se suministra el código fuente del Backend, aplicación web, aplicación móvil y diseño de base de datos que incluyen un desarrollo parcial de la solución. Esto permitirá que la empresa avance con el proyecto y realice los ajustes que considere necesarios.

Como recomendación, se debe validar la correcta implementación de la arquitectura conforme se realice nuevas funcionalidades. Con la adición de funciones se puede cometer el error de desviarse de las pautas de la arquitectura, comprometiendo el uso de buenas prácticas. También es aconsejable mantener actualizas las versiones de los Frameworks y librerías, evitando el uso de software obsoleto.

En cuanto a la aplicación móvil, se recomienda finalizar con el desarrollo utilizando buenas prácticas e implementar una arquitectura de acuerdo con los criterios de la empresa utilizando patrones de diseño adecuados.

Además de las recomendaciones previamente mencionadas, es crucial que la empresa adopte y siga un conjunto de lineamientos de desarrollo estandarizados. Esto incluye la implementación de un proceso de integración y entrega continuas (CI/CD) para automatizar la prueba y despliegue de aplicaciones. También es importante establecer un protocolo de revisión de código que asegure la calidad y coherencia del código a través de revisiones por pares. Se sugiere la adopción de metodologías ágiles para mejorar la colaboración y eficiencia del equipo de desarrollo, así como para responder de manera más efectiva a los cambios en los requisitos del proyecto. Finalmente, se recomienda fomentar una cultura de documentación exhaustiva para facilitar la mantenibilidad y escalabilidad de las aplicaciones desarrolladas. Estos lineamientos ayudarán a mantener un alto estándar de calidad en el desarrollo de software y contribuirán al éxito a largo plazo de los proyectos de la empresa.

Por último, teniendo en cuenta que la ingeniería de software evoluciona cada vez más rápido, se recomienda que se mantenga actualizada la arquitectura con nuevas tendencias de la industria.

REFERENCIAS

1. Anderson, D. (2010) Kanban: Successful Evolutionary Change for Your Technology Business. Estados Unidos de América: Blue Hole Press.
2. Arrobasolutions (2023) Arquitectura de Microservicios, obtenido [en línea] el 11 de septiembre de 2023 en https://www.arrobasolutions.com/arquitectura-de-microservicios-que-es-y-cuales-son-sus-ventajas/
3. Cetindamar, D.; Phaal, R. (2016) Technology Management: Activities and Tools. Ed. Estados Unidos de América: Red Globe Press.
4. Carrión, J. (2011) Estrategia de la visión a la acción. 3° ed. España: ESIC.
5. Cobb, C. (2015) Project Manager's Guide to Mastering Agile. Estados Unidos de América: Wiley.
6. Debrauwer, A.; Heyde, F. (2014) UML 2.5 Iniciación, ejemplos y ejercicios corregidos. Ed. 5. ENI: España.
7. Debrauwer, A.; Heyde, F. (2022). Patrones de diseño en C# Los 23 modelos de diseño: descripción y soluciones ilustradas en UML 2 y C#. Ed. 2. ENI: España.
8. DeMarco, T. & Lister, T. (2016) Peopleware: Productive Projects and Teams. 3° ed. Estados Unidos de América: Dorset House.
9. Fowler, M. (2002) Patterns of Enterprise Application Architecture. Addison-Wesley Professional: Estados Unidos de Norteamérica.
10. Genett, D. M. (2005) Delega: un modelo para crear equipos de alto rendimiento. España: Empresa Activa.
11. Guérin, B-A. (2018) Gestión de proyectos informáticos. 3° ed. España: ENI.
12. Hermida, A. (2022) Por qué es relevante tener un proceso de S&OP, obtenido [en línea] el 13 de septiembre de 2023 en https://www.evaluandoerp.com/relevante-proceso-sop/
13. Iger, R. A. (2020) Lecciones de Liderazgo Creativo. España: Conecta.
14. Karoly Nyisztor, K.; Nyisztor, M. (2018) UML and Object-Oriented Design Foundations: Understanding Object-Oriented Programming and the Unified Modeling Language. Independently published: Estados Unidos de Norteamérica.
15. Leal, V. (2020) Liderazgo: Adquiere Poderosos Hábitos y Habilidades de Liderazgo Rápidamente.
16. Lencioni, P. (2011) Las Cinco Disfunciones de un Equipo. España: Editorial Empresa Activa.
17. Microsoft (2023) ciclo de vida: .NET Framework, obtenido [en línea] el 19 de agosto de 2023 en https://learn.microsoft.com/es-es/lifecycle/faq/dotnet-framework
18. Martin, R. (2017) Arquitectura limpia: Guía de un artesano para la estructura y el diseño del software. Ed. 1. Pearson: Estados Unidos de Norteamérica.
19. Microsoft (2022) Documentación .Net 7.0, obtenido [en línea] el 19 de agosto de 2023 en https://dotnet.microsoft.com/es-es/download/dotnet/7.0
20. Microsoft (2022) Mejoras de rendimiento en .NET 7, obtenido [en línea] el 19 de agosto de 2023 en https://devblogs.microsoft.com/dotnet/performance_improvements_in_net_7/
21. Newman, S. (2021) Building Microservices: Designing Fine-Grained Systems. Ed. 2. O'Reilly Media: Estados Unidos de Norteamérica.
22. Redacción ADP, (2022) ¿Qué es el entorno VUCA y cómo afecta a la supervivencia de las empresas?, obtenido [en línea] el 13 de septiembre de 2023 en https://www.apd.es/que-es-el-entorno-vuca-y-como-afecta-a-la-supervivencia-de-las-empresas/

23. Sommerville, Ian. (2016) Ingeniería de Software. Ed. 10. Pearson: Estados Unidos de Norteamérica.
24. Sutherland, J. (2018) Scrum El revolucionario método para trabajar el doble en la mitad de tiempo. España: Ariel.
25. WA Solutions (2023) Pagina web de la empresa WA Solutions, obtenido [en línea] el 19 de agosto de 2023 en https://wasolutions.co/es/contacto/
26. Wallace, T. (2004) Sales & Operations Planning: The How-To Handbook Steelwedge Software: Estados Unidos de Norteamérica.

FIRMA ESTUDIANTES _____

_____

_____

FIRMA ASESORES _____

_____

FECHA ENTREGA: _____