Tesinas de Grado

Lucas Edgardo Barrera

Migración de sistema de beneficios

2025

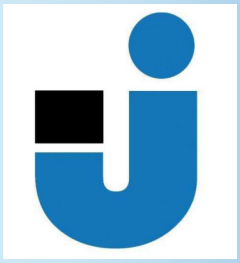
Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Migración de sistema de beneficios

Lucas Barrera

Florencio Varela, octubre 2024

Estudiante

Nombre y Apellido: Lucas Barrera

Correo electrónico: lucased.barrera@gmail.com

Organización donde se realiza la Práctica Profesional Supervisada

Nombre de la Empresa- Entidad- Institución: CONNECTIS ICT SERVICES S.A.

Dirección: Av. Luis María Campos 877 P10 1426

Teléfono: 011 15 5365-6900 extensión 6011

Sector: Servicios Profesionales

Tutor organizacional

Nombre y apellido: Weber Marta

Correo electrónico: marta.weber@getronics.com

Docente Supervisor

Nombre y apellido: Dr. Ing. Martín Morales

Correo electrónico: martin.morales@unaj.edu.ar

Docente tutor del Taller de Apoyo para la Producción de Textos Académicos

Nombre y apellido: Prof. Esp. Lia Lavigna

Correo electrónico: llavigna@unaj.edu.ar

Coordinador de la Carrera de Ingeniería en Informática

Nombre y apellido: Dr. Ing. Martín Morales

Correo electrónico: martin.morales@unaj.edu.ar

Resumen

El presente trabajo detalla la experiencia de una Práctica Profesional Supervisada realizada en el contexto laboral. El proyecto consistió en la migración de un sistema de beneficios, desde un sistema cerrado hacia una base de datos abierta, optimizando la accesibilidad y escalabilidad del sistema. La práctica tuvo como objetivo general diseñar, desarrollar, testear e implementar los microservicios y procedimientos de base de datos necesarios para la migración de un sistema de beneficios. Entre los objetivos específicos se incluyeron el relevamiento de las capacidades del sistema productivo, el desarrollo de procesos para la gestión de grupos y beneficios, la incorporación de nuevas características como la trazabilidad de las operaciones y la gestión de reintentos. Para ello, se empleó una metodología basada en el desarrollo ágil utilizando tecnologías ampliamente adoptadas en el mercado como Node.js, Express y Typescript para la creación de microservicios eficientes y escalables. Los resultados más significativos incluyen la consolidación del sistema migrado, la mejora en el rendimiento de las consultas y el procesamiento de datos, así como el despliegue exitoso de los microservicios en ambientes productivos.

Palabras clave: migración, API, Sql Server, Node.js.

Abstract

This work presents the design, the development, and the implementation of backend microservices using Agile methodologies, specifically the Scrum framework to enhance team collaboration and accelerate software delivery.

The integration of DevOps practices and tools like GitLab for CI/CD pipelines streamlined deployment and improved code management.

The backend architecture was built using Node.js with Express, following backend architecture patterns.

TypeScript was employed for typying safety and maintainability, while the Sequelize ORM facilitated interaction with SQL Server to manage complex database logic.

Key practices such as Test-Driven Development (TDD) were applied in this project using Jest for unit testing and Joi for schema validation.

The microservices exposed REST APIs with detailed documentation generated through Swagger, ensuring clear communication and usability across different teams.

Additionally, the project incorporated performance testing through JMeter and static code analysis using SonarQube to ensure code reliability.

Security vulnerabilities were assessed with HCL AppScan contributing to the overall robustness of the system.

This approach showcases a modern backend development process, leveraging microservices and continue integration to ensure scalability, maintainability and performance.

Keywords: API, Sql Server, Node.js.

Dedicatorias y agradecimientos

Esta práctica profesional supervisada está dedicada a mis padres, Nora y Eva, quienes me indicaron el camino y me inculcaron la cultura del esfuerzo, la dedicación y el trabajo. A mi esposa, Lore, por su amor y apoyo incondicional; a mis hijos, Mate y Ale, la motivación de mis logros. Su paciencia y sacrificios durante este proceso han sido fundamentales para alcanzar mis metas. Cada momento compartido con ellos me impulsa a seguir adelante, su sonrisa y alegría son mi inspiración para enfrentar cualquier desafío.

Agradezco a mis compañeros, con quienes recorrí mi carrera, y a mis profesores por su vocación y dedicación en la enseñanza. Especialmente, a mis tutores: Marta Weber, Lía Lavigna y Martín Morales, por su constante apoyo, orientación y confianza. Su experiencia y aportes fueron claves en esta etapa. Finalmente, a la Universidad Nacional Arturo Jauretche, por brindarme el espacio de aprendizaje enriquecedor que me permitió crecer tanto personal como profesionalmente.

Índice

Contenido

Índice de imágenes o figuras

1. Introducción

El presente trabajo, destinado a cumplir con la Práctica Profesional Supervisada (PPS), se realiza en el contexto de una empresa de consultoría. A tal fin, se solicita la migración de los procesos de gestión de cierto beneficio otorgado a los clientes de la empresa. La importancia del proyecto para la gestión de este beneficio viene dada por su objetivo de fidelización de clientes.

Actualmente, el procesamiento se realiza en un sistema cerrado con reglas predeterminadas, baja flexibilidad, funcionalidad limitada y escaso registro de datos, lo que se traduce en algunas limitaciones como: la falta de acceso a la información, poco control del proceso, largos tiempos de gestión, que impactan directamente en el cliente y afectan su percepción de la empresa.

La implementación actual permite realizar una solicitud, conocer las cuentas relacionadas de un cliente, el estado de asignación del beneficio para una cuenta determinada, la posibilidad de acceder al mismo y el estado de una solicitud pendiente. Teniendo presente que una persona física puede tener más de un cliente en la empresa y cada cliente puede tener varias cuentas, para poder contar con el beneficio, se presenta la limitación de que se obliga a la persona a migrar la cuenta bajo el mismo cliente.

En cuanto al beneficio, se otorga a un grupo o conjunto limitado de cuentas de un mismo cliente que él mismo selecciona. Para poder acceder al beneficio, el cliente debe gestionarlo en una aplicación móvil de la empresa o requerirlo vía telefónica, en ese caso, un representante de atención al cliente realizará el trámite en una aplicación web interna.

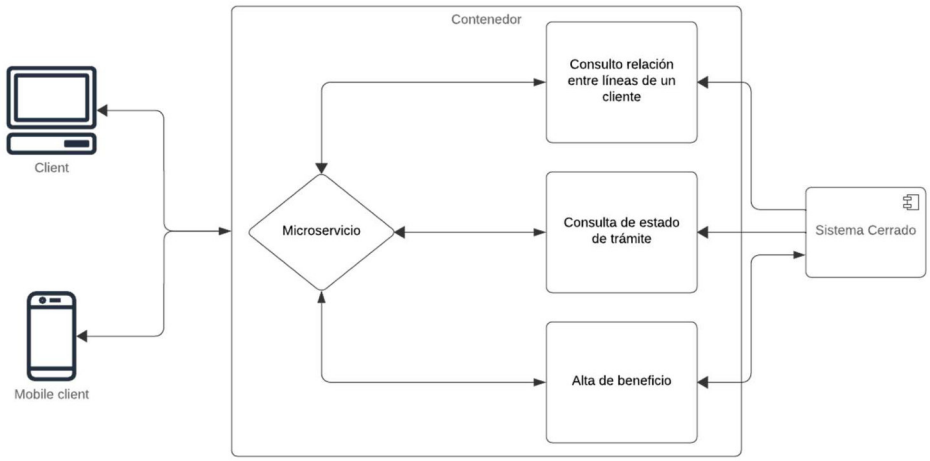


Figura 1. Diseño de arquitectura actual

Fuente: Elaboración propia, basada en la práctica.

La migración solicitada implica la investigación, diseño, desarrollo e implementación de un nuevo sistema que reemplace al actual, en el que se logre un mayor control sobre el proceso y se mejore la calidad de la información registrada sobre la gestión.

La propuesta para la migración consiste en:

- Diseñar un nuevo modelo de base de datos relacional en un servidor productivo SqlServer que registre lo siguiente: la gestión de los beneficios, la relación entre los grupos de cuentas de los clientes, las comunicaciones y la trazabilidad de la gestión.
- Desarrollar los procedimientos de base de datos necesarios para atender las solicitudes de altas, bajas y modificaciones de las entidades a modelar.
- Desarrollar los triggers para atender las respuestas de las solicitudes a una API (ya productiva), que realiza el alta y baja de beneficios.
- Desarrollar los procedimientos calendarizados o jobs para detectar cambios en las condiciones comerciales y gestionar reintentos de altas de beneficios.
- Desarrollar los microservicios necesarios, sobre la capa de base de datos, para permitir realizar las validaciones contra el sistema de gestión de clientes y consumir los procesos desarrollados en la base de datos.
- Los microservicios desarrollados serán consumidos por la aplicación móvil que es accedida directamente por los clientes y una aplicación web interna utilizada por los representantes de atención al cliente.

- El desarrollo de la solución será realizado bajo una metodología scrum en el marco de trabajo ágil.

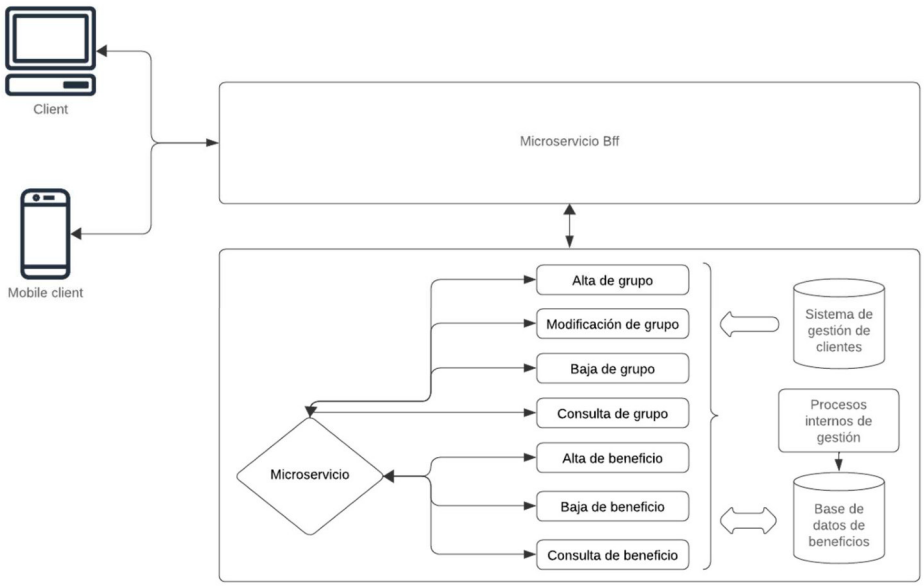


Figura 2. Diseño de arquitectura propuesta

Fuente: Elaboración propia, basada en la práctica.

Entre las mejoras a alcanzar por la migración, se espera: lograr mayor control de la relación entre los clientes, registrar la trazabilidad de la gestión, mejorar la cantidad y calidad de información disponible del modelo de datos, afianzar la relación con los clientes al gestionar las comunicaciones, acotar los tiempos de gestión de la relación y los beneficios.

Al momento de implementar la solución se debe realizar la migración de datos del sistema actual y apagado del mismo.

2. Objetivos

El objetivo general de esta PPS es diseñar, desarrollar, testear e implementar los microservicios y procedimientos de base de datos necesarios para realizar la migración de un sistema de beneficios. Para lograr esto, se deberán cumplir los siguientes objetivos específicos:

- Realizar un relevamiento acerca de las funcionalidades del sistema actualmente productivo.
- Desarrollar los procesos para la gestión de grupo y beneficio en un nuevo sistema que se alimentará de una nueva base de datos.
- Investigar herramientas y tecnologías para el desarrollo de microservicios.
- Desarrollar los procesos necesarios, bajo la arquitectura de microservicios, para consumir los procesos de base de datos desarrollados anteriormente.
- Registrar información del proceso para lograr la trazabilidad de la gestión.
- Registrar información valiosa para el negocio en el nuevo modelo de datos.
- Proveer información que ha sido registrada, para la generación de reportes.
- Lograr tener un mayor control de todo el proceso de gestión.
- Agregar funcionalidades no existentes en sistema actual como reintentos de altas y bajas de beneficios.
- Mejorar la percepción del cliente con la empresa, con el objetivo de transmitir pertenencia.
- Mejorar las comunicaciones con el cliente.
- Agilizar los tiempos de gestión.
- Exponer los microservicios en producción.
- Realizar una migración de datos del sistema actual al nuevo sistema desarrollado.
- Planificar y realizar el apagado del sistema actual.

3. Tareas por ejecutar

- Tarea 1: Completar el relevamiento iniciado con los usuarios para capturar su visión. El objetivo de esta tarea es poder obtener el listado de historias necesarias que se incluirán en el backlog para su posterior priorización y bajo la metodología scrum.

- Tarea 2: Diseñar el nuevo modelo de datos para registrar todas las necesidades de datos expuestas por el negocio obtenidas en la tarea 1. El objetivo de esta tarea es poder diseñar el modelo de datos, poder plasmarlo en un diagrama de base de datos y escribir las sentencias de SQL para la creación de las tablas.

- Tarea 3: Diseñar, desarrollar y probar los procedimientos de base de datos que serán interfaz con el microservicio, estos *stored procedures* serán invocados desde los microservicios y tendrán como finalidad consultar o gestionar los grupos y beneficios. En esta etapa, el diseño está intrínsecamente relacionado con el diseño y desarrollo de microservicios, que serán abordados en tareas posteriores. El objetivo de esta tarea es desarrollar los *stored procedures, triggers y jobs* necesarios para la gestión.

- Tarea 4: Desarrollar procedimientos internos de la base de datos. Además de los procesos descritos en el punto anterior, se deben desarrollar procesos para modificar los datos de los grupos y beneficios a partir de la detección de cambios en las condiciones comerciales.

- Tarea 5: Adquisición de conocimientos para el desarrollo de microservicios y diseño. La adquisición será de manera autónoma y también de los integrantes de la mesa del equipo de desarrollo. El objetivo de esta tarea es el de interiorizarse en la metodología de desarrollo de microservicios, diseñar los microservicios necesarios para cumplir con las necesidades de negocio y plasmar el diseño de la arquitectura en diagramas.

- Tarea 6: Desarrollar microservicios necesarios para responder las solicitudes de gestión de la aplicación móvil y la aplicación web. El objetivo de esta tarea es implementar los microservicios en ambientes bajos.

- Tarea 7: Realizar las pruebas en ambientes bajos, de los procesos de base de datos y microservicios. El objetivo de esta tarea es diseñar y ejecutar las pruebas y obtener la conformidad del sector de seguridad de datos de la empresa.

- Tarea 8: Implementación de los procesos de base de datos y microservicios. Migración de datos del sistema actual. El objetivo de esta tarea es desarrollar el proceso masivo para la migración de datos del sistema actual, realizar la migración y apagado del mismo.

- Tarea 9: Escritura del informe final.

4. Cronograma de trabajo

Tareas	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5
Tarea 1: historias	**X**				
Tarea 2: modelo, tablas	**X**				
Tarea 3: stored procedures, triggers, jobs	**X**				
Tarea 4: stored procedures, triggers, jobs		**X**			
Tarea 5: diseño ms, diagramas		**X**	**X**		
Tarea 6: ms en ambientes bajos			**X**	**X**	

Tarea 7: pruebas, seguridad				**X**	
Tarea 8: implementación, apagado y migración.					**X**
Tarea 9: Informe final PPS	**X**	**X**	**X**	**X**	**X**

5. Desarrollo

a. Conceptos, tecnologías y herramientas utilizadas.

i. Visual Studio Code

Visual Studio Code es un potente editor de código, gratuito y multiplataforma. Tiene funciones como autocompletado de texto inteligente, permite la depuración y corrección de código en una consola interactiva, está integrado con Git cubriendo toda funcionalidad directamente desde el editor, es personalizable mediante la instalación de extensiones que se ejecutan en procesos separados evitando ralentizar el editor (Visual Studio Code, 2024).

ii. Base de datos relacionales

Una base de datos relacional es un conjunto de datos organizados en tablas, a su vez, dentro de una tabla los datos se organizan en filas y columnas. Una fila es un registro individual, y cada columna representa un atributo al que se le asigna un valor. Esta organización facilita que se establezcan relaciones entre tablas y sea posible recuperar datos de diferentes maneras. Una base de datos relacional es una forma de almacenar datos con relaciones predefinidas y acceder a ellos. Esto conlleva que las bases de datos permiten la abstracción del almacenamiento físico de los datos (Base de datos Relacional, 2024).

iii. Sistema de administración de base de datos

Un sistema de administración de base de datos o RDBMS (*Relational Database Management System*) es un software que facilita una interfaz entre los usuarios o aplicaciones y la base de datos. El RDBMS permite al administrador de la base de datos, un mayor control sobre el acceso (Base de datos Relacional, 2024).

iv. Modelo de datos

En un modelo de datos las tablas representan entidades, es decir, un concepto real. Cada columna guarda un tipo de datos en particular, un atributo de la entidad que contiene el valor de este. En una tabla, las filas representan conjunto de valores relacionados y estas pueden identificarse unívocamente mediante el atributo “clave principal”. En cambio, una “clave externa” de una tabla, hace referencia a una “clave principal” de otra tabla, y permite establecer una relación lógica. En consecuencia, las filas de diferentes tablas pueden relacionarse mediante claves principales y externas (Base de datos Relacional, 2024).

v. SQL

El “*structured query lenguage*” (SQL) o “lenguaje de consulta estructurada” es un lenguaje de programación que permite almacenar, actualizar, buscar, procesar, eliminar y recuperar datos de una base de datos relacional (SQL, 2024).

vi. T-SQL

Transact-SQL es una extensión del lenguaje SQL, es un lenguaje de programación y como tal, permite la declaración y asignación de variables, realizar procesamiento de cadenas de texto, control de errores, transacciones, ejecución condicional, creación de procedimientos, pasaje de parámetros y control de flujo de programas (Introducción a la programación con Transact-SQL, 2024).

vii. Microsoft SQL server

Microsoft SQL Server es un sistema de administración de bases de datos relacionales o RDBMS, las aplicaciones o herramientas se conectan a una base de datos mediante T-SQL. El principal servicio de MS SQL Server es el motor de base de datos y su tarea es almacenar, procesar y proteger datos (SQL Server, 2024).

viii. SQL Server Management studio

SQL Server Management studio o SSMS provee un entorno de administración, configuración y desarrollo. De manera integral combina herramientas gráficas y editores de script para administradores y desarrolladores (SQL Server Management Studio, 2024).

ix. Stored procedure

Un procedimiento almacenado o *stored procedure* permite agrupar una o varias instrucciones T-SQL, acepta parámetros de entrada y puede devolver varios valores de salida al programa que lo convoca, también puede contener llamadas a otros *stored procedures*. El uso de *stored procedures* tiene distintas ventajas: menor tráfico de red al ejecutarse como un lote único sin necesidad de enviar el código a ejecutar, mayor seguridad al evitar otorgar permisos a usuarios y procesos para realizar una tarea, evita los ataques de inyección de código, permite la reutilización de código, mantenimiento más sencillo y rendimiento mejorado (Procedimientos almacenados, 2024).

x. Trigger

Un trigger es un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de la base de datos. En el caso de los triggers DML se disparan cuando se intenta modificar datos, es decir, cuando se ejecuta un INSERT, UPDATE o DELETE (Create Trigger (Transact-SQL), 2024).

xi. Jobs

El SQL Server Agent es un servicio de SQL Server que permite ejecutar *jobs*, los *jobs* contienen uno o varios *steps* y cada *step* tiene su propia tarea que puede ser ejecutar una sentencia SQL o el llamado a un *stored procedure*, entre otras. SQL Server Agent permite ejecutar un *job* mediante un *schedule*, en respuesta a un evento o a demanda (SQL Server Agent, 2024).

xii. Git y GitLab

Git es un sistema de control de versiones, que permite realizar un seguimiento de los cambios en el código, a través del tiempo y cada vez que se realizan modificaciones, se guarda una versión del código. Git es distribuido, lo que significa que se mantiene una copia local del proyecto, conocida como repositorio, permitiendo trabajar localmente y sin conexión. Los desarrolladores pueden confirmar cambios localmente y luego sincronizarlos con la copia del servidor. Al guardar el proyecto, Git crea una confirmación, que es una instantánea actualizada de todos los archivos del proyecto en un momento dado. Si un archivo no cambia entre dos confirmaciones, Git usa el archivo almacenado en la primera confirmación. Mediante un hash criptográfico, Git puede detectar cambios, pérdida de información o daño en los archivos. Un hash criptográfico es un algoritmo que, a partir de una entrada, genera

una cadena fija de caracteres y, por sus propiedades, se usa ampliamente para la validación de la integridad de datos. Git también proporciona herramientas para permitir el trabajo colaborativo, como las ramas. Estas permiten a los desarrolladores aislar sus cambios sobre una misma confirmación y, posteriormente, integrarlos a la rama principal.

Las ventajas principales de Git son las siguientes:

- Desarrollo en paralelo: gracias a las versiones locales, es posible trabajar simultáneamente en diferentes partes del proyecto.
- Versiones de lanzamiento más rápidas: la rama principal contiene código estable, mientras que las otras ramas contienen el trabajo en curso, que solo se combina con la rama principal al estar finalizado. Al separar las ramas de desarrollo, es más fácil administrar el código estable.
- Integración incorporada: Git se integra con la mayoría de las herramientas, todos los IDE principales (entornos de desarrollo integrado) tienen compatibilidad con Git, lo que facilita el trabajo (Git, 2024).

GitLab, por su parte, es una plataforma web para la gestión del ciclo de vida del desarrollo de software utilizando Git como sistema de control de versiones. GitLab permite organizar y gestionar proyectos de desarrollo de software, facilitando la colaboración entre desarrolladores y otros equipos. Integra capacidades de CI (Integración Continua) y CD (Despliegue Continuo) automatizando la construcción, las pruebas y el despliegue de aplicaciones. Además, facilita la configuración de roles y niveles de acceso, proporciona acceso a información sobre cambios y acciones relacionadas en el repositorio, incluye herramientas de documentación y ofrece soporte para la metodología DevOps en el desarrollo de software. DevOps es una cultura y metodología que busca unificar el desarrollo (Dev) y las operaciones (Ops) para mejorar la eficiencia, reducir el tiempo de entrega y asegurar la calidad en la creación y mantenimiento de aplicaciones (GitLab, 2024).

xiii. Metodología ágil

El ciclo de vida en cascada incluye las fases de planificación, diseño, desarrollo, pruebas, implementación y mantenimiento. En los 90, era común que transcurrieran varios años entre la identificación de una necesidad empresarial y la distribución de una aplicación en funcionamiento. Debido a los rápidos cambios empresariales y de mercados en esa época, muchos proyectos sufrían cancelaciones significativas antes de poder distribuirse. Se hacía evidente la necesidad de un enfoque que permitiera adaptarse rápidamente a los cambios. La metodología ágil es un enfoque que promueve la colaboración y la eficiencia de los flujos de trabajo, guiado por un conjunto de valores fundamentales. Su objetivo principal es entregar

al cliente en cortos periodos de tiempo, pequeñas y funcionales piezas de software. Para lograrlo, se forman equipos pequeños y auto-organizados de desarrolladores y representantes de la empresa, quienes se reúnen regularmente durante el ciclo de vida del desarrollo de software. En lugar de resistirse a los cambios, los equipos ágiles los aceptan y se adaptan a ellos.

La metodología ágil surgió como respuesta a los enfoques en cascada de la gestión de proyectos, que se organizaban en secuencias lineales. Un grupo de desarrolladores redactó el Manifiesto para el desarrollo ágil de software. En este documento, que propone un nuevo enfoque para el desarrollo de software, se priorizan: las personas y las interacciones por encima de los procesos y las herramientas, el software en funcionamiento sobre la documentación exhaustiva, la colaboración con el cliente en lugar de la negociación contractual y la adaptación al cambio sobre la adherencia a un plan. El Manifiesto es una guía para pensar el desarrollo de software, no intenta imponer un conjunto de prácticas. Scrum es, probablemente, el marco ágil más utilizado (Metodología Ágil, 2024).

xiv. Aplicaciones Cliente – Servidor

El modelo cliente-servidor es una estructura de comunicación utilizada en redes de computadoras. Un servidor, que puede ser hardware o software, proporciona recursos a otros programas o computadoras, respondiendo a las peticiones de los clientes, procesándolas y devolviendo las respuestas adecuadas. Este modelo describe la interacción entre clientes y servidores, estableciendo una clara distribución de tareas entre ellos; es decir, el servidor ofrece servicios y el cliente los utiliza. Un servidor puede atender a múltiples clientes y, en esta relación, el servidor actúa de manera pasiva, mientras que el cliente es activo. Una computadora puede desempeñar el rol de servidor o cliente, dependiendo de si envía o recibe solicitudes de servicio.

xv. Backend

En el campo del desarrollo web, el Backend (BE) se encarga de gran parte de la lógica de una página web. También, asegura el correcto tratamiento de los datos, la seguridad y la optimización de los recursos, aunque estas funciones no sean visibles para el usuario.

Entre los diversos lenguajes de programación del lado del servidor, Node.js es uno de los más destacados. Las tareas realizadas desde el BE incluyen la implementación de la lógica del negocio, las conexiones con bases de datos, la seguridad de los sitios web y la optimización de recursos (Backend, 2024).

xvi. Node y Express

Node.js y Express son tecnologías complementarias que se utilizan comúnmente en el desarrollo de aplicaciones web del lado del servidor. Node.js es un entorno que opera de manera independiente de un navegador web y es de código abierto, multiplataforma, que permite desarrollar aplicaciones JavaScript del lado del servidor. Diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web, Node.js utiliza JavaScript como su lenguaje de programación. Además, cuenta con un gestor de paquetes llamado NPM (Node Package Manager), que facilita la resolución de dependencias y la automatización de compilación. Node.js es portable y está ampliamente soportado por servicios de hospedaje web con infraestructuras específicas para Node.

Por su parte, Express es el framework web más popular para Node que simplifica el proceso de configuración del servidor y definición de rutas. Un framework es un conjunto de herramientas, guías y estructuras predefinidas que se utilizan para desarrollar software de manera eficiente. Express proporciona mecanismos para escribir manejadores de peticiones HTTP en diferentes rutas, establecer ajustes web como el puerto a usar y procesar peticiones mediante middlewares. HTTP (Hypertext Transfer Protocol) es el protocolo utilizado para la transferencia de datos en la web, permitiendo la comunicación entre clientes (como navegadores web) y servidores. En el caso de Middleware, este se refiere a funciones que se ejecutan durante el ciclo de vida de una solicitud HTTP y que pueden modificar la solicitud, la respuesta o finalizar el ciclo (Node, 2024).

xvii. JavaScript

JavaScript es un lenguaje de programación o de secuencia de comandos y es una de las tres capas de tecnologías web junto con HTML y CSS. Por una parte, HTML es un lenguaje de marcado que se usa para estructurar y dar significado a una página web; por otra parte, CSS es un lenguaje de reglas de estilo y se usa para aplicar estilo al contenido HTML. En el desarrollo web, JavaScript permite crear contenido que se actualiza dinámicamente, controles multimedia, animación de imágenes, entre otras cosas (Javascript, 2024). JavaScript posee determinadas características que lo hacen potente y versátil del lado del servidor: mejora la eficiencia y rendimiento al atender gran cantidad de solicitudes utilizando un modelo de Entrada/Salida no bloqueante y basado en eventos, ejecución rápida y eficiente utilizando el motor V8 de Google al compilar JavaScript a código máquina nativo, permitiendo la reutilización de código y mejor organización con el sistema de módulos (CommonJS), acelerando el desarrollo mediante NPM al proveer herramientas y librerías, facilitando el

intercambio entre cliente y servidor al manejar JSON de manera nativa (Node.js documentation, 2024).

xviii. TypeScript

TypeScript (TS) es un lenguaje de programación construido sobre JavaScript (JS), que proporciona características adicionales para escribir un código más limpio y robusto. Fue adoptado por Google en el desarrollo de Angular (framework de frontend), también en React (desarrollado originalmente en JS) para en el frontend y en Node.js para el backend. TS añade funcionalidades como tipado fuerte, que ayuda a evitar errores y mejora la legibilidad del código; anotaciones, que facilitan la documentación y comprensión de código al especificar claramente los tipos esperados y módulos, que organizan el código de manera que sea más mantenible y reutilizable. TS es un superconjunto de JS, lo que significa que todo el código en JS es válido en TS. Sin embargo, el código TS no puede ejecutarse directamente en un entorno JS, primero debe hacerse una transpilación a JS (Typescript vs Javascript, 2024).

xix. Microservicios

Una arquitectura monolítica es un modelo tradicional donde el software completo se compila de forma unificada e independiente de otras aplicaciones. La arquitectura monolítica puede ser práctica al inicio de un proyecto para simplificar la gestión del código y la implementación. Pero, a medida que la aplicación crece y se vuelve más compleja, resulta difícil escalarla, incorporar actualizaciones y gestionar la implementación continua. La arquitectura de microservicios descompone una aplicación en múltiples servicios independientes que pueden implementar de manera autónoma. Estos microservicios se comunican entre sí a través de API (Interfaces de Programación de Aplicaciones), que son conjuntos de definiciones y protocolos que permiten la interacción entre diferentes softwares. Esta metodología permite desplegar y escalar cada servicio por separado facilitando entregas rápidas y frecuentes de aplicaciones de alta complejidad.

Algunas características de los microservicios son:

- Desarrollo independiente: componentes que se pueden desarrollar, implementar, modificar y operar sin afectar a otros servicios.
- Experimentación y vuelta atrás: permiten experimentar con nuevas funcionalidades y revertirlas en caso de que no funcionen, lo que facilita la actualización del código y acelera la implementación de nuevas funcionalidades.

- Errores: simplifica el proceso de aislamiento y corrección de errores.
- Metodologías ágiles y DevOps: los equipos suelen crear un servicio dentro de un microservicio, fomentando el uso de metodologías ágiles y prácticas DevOps, lo que acorta el ciclo de desarrollo.
- Automatización de infraestructuras: los microservicios se suelen combinar con prácticas de automatización de infraestructuras, como Integración continua CI y CD, permitiendo crear e implementar los servicios sin afectar a otros equipos y gestionar diferentes versiones de servicios en paralelo (Microservicios, 2024).

xx. API REST o API RESTful

Como su nombre lo indica, una API REST es una interfaz de aplicaciones (API) que se encuadra dentro de los límites de la arquitectura REST, permitiendo la interacción con servicios web RESTful. Además, una API es un conjunto de definiciones y protocolos utilizados para diseñar e integrar el software de distintas aplicaciones. En una API, un *endpoint* es un punto final de comunicación en una API, específicamente, es una URL (o ruta) donde se puede acceder a un recurso específico de la API. Los endpoints definen dónde y cómo se pueden realizar las operaciones en una API, como obtener datos, enviar datos, actualizar información o eliminar recursos. Puede verse como el acuerdo entre el productor y el consumidor de la información, en él se establece lo que se necesita del consumidor para poder responder a una solicitud. La utilización de APIs permite compartir recursos o información, conservando la seguridad, el control de acceso y la autenticación, abstrayendo el proceso y el origen de la información.

REST es entendida como un conjunto de principios de arquitectura. Ante una solicitud a una API RESTful, se devuelve, mediante HTTP, un estado del recurso solicitado en formato JSON (JavaScript Object Notation), HTML, XML, Python, PHP o texto sin formato. Mediante HTTP, se transfiere información de metadatos, autorización, identificación uniforme de recursos (URI), cache, cookies, etc. Para que una API sea considerada RESTFul, debe cumplir con los siguientes principios:

- Arquitectura cliente-servidor: gestionada a través de solicitudes HTTP.
- Comunicación sin estado: entre cliente y servidor, sin relación entre diferentes llamadas.
- Datos almacenados en caché: con el objetivo de optimización, acumulando datos para requerimientos recurrentes.

- Interfaz uniforme entre los elementos: toda la información transferida está estandarizada. Los recursos solicitados son independientes de las representaciones devueltas, el cliente debe recibir información suficiente para poder manipular los recursos y los mensajes recibidos por el cliente le permiten decidir cómo procesar la información.
- Sistema de capas: que organiza los diferentes servidores que participan en la recuperación de la información, como seguridad o equilibrio de carga (Api Restful, 2024).

xxi. HTTP

HTTP o Protocolo de Transferencia de Hipertexto, permite enviar documentos a través de la red. Es un conjunto de reglas que establece qué mensajes se pueden intercambiar y qué respuestas se pueden enviar a otros mensajes. En los mensajes HTTP se pueden identificar dos partes: el encabezado y el cuerpo. Por un lado, el encabezado contiene metadatos, información importante como la codificación y los métodos HTTP, solo puede contener texto sin formato y suele ser más significativo que el cuerpo. Por otro lado, el cuerpo puede contener datos que se desean transmitir o puede estar vacío. Además, en el cuerpo se puede enviar texto sin formato, imágenes, HTML, XML. La respuesta HTTP debe especificar el tipo de contenido del cuerpo, lo que se especifica en el encabezado, en el campo *Content-Type*, por ejemplo “Content/Type: application/json”.

Los verbos HTTP le indican al servidor qué hacer con los datos identificados por la URL. Una URL proporciona la dirección de un recurso en la web y el formato típico es http://www.ejemplo.com/page, e incluye el protocolo (http o https), el nombre del dominio (www.ejemplo.com) y la ruta (/page).

La herramienta cURL (Client URL) es una herramienta de línea de comandos para transferir datos con sintaxis URL, permite realizar solicitudes, guardar la respuesta, adjuntar archivos JSON, añadir encabezados, entre otras funcionalidades. Cada solicitud HTTP, especifica el encabezado, en la cual la primera palabra, en mayúscula, es el verbo HTTP. Algunos de ellos son los siguientes: GET, POST, PATCH, PUT, HEAD y OPTIONS:

- GET: indica al servidor que transmita al cliente los datos identificados por la URL, los datos no serán modificados.
- POST: crea un nuevo recurso, contiene los datos necesarios para crear o actualizar el recurso.
- PUT: para actualizar un recurso identificado por la URL, contiene los datos necesarios para crear o actualizar el recurso.

- DELETE: eliminar un recurso identificado por la URL.
- PATCH: realiza actualizaciones parciales en un recurso. A diferencia de PUT, que reemplaza completamente el recurso, PATCH aplica cambios específicos.

Los códigos de respuesta HTTP estandarizan el resultado del estado de la solicitud, su significado no es muy preciso, lo que se debe a la generalidad de HTTP. Algunos códigos de respuesta HTTP son los que se presentan a continuación:

- 200 OK: indica que la solicitud se ha realizado correctamente.
- 201 Created: indica que la solicitud tuvo éxito y se creó el recurso, es una confirmación del éxito de PUT o POST.
- 400 Bad Request: la solicitud está mal realizada, entonces los datos no pasan la validación o están en formato incorrecto.
- 401 Unauthorized: indica que debe realizar la autenticación antes de acceder al recurso.
- 404 Not Found: indica que no se pudo encontrar el recurso solicitado.
- 405 Method Not Allowed: indica que el método HTTP especificado no es compatible con ese recurso.
- 409: indica un conflicto, por ejemplo, que está utilizando una solicitud PUT para crear el mismo recurso dos veces.
- 500: indica imprevistos del lado del servidor (HTTP y API REST, 2024).

xxii. TDD

El Test Driven Development (TDD) es un enfoque de desarrollo de software en el cual las pruebas unitarias se escriben antes del desarrollo del código. Consiste en la creación de los casos de prueba para cada funcionalidad que se quiere desarrollar. Después de escribir las pruebas, se desarrolla el código de manera incremental para que cumplan con las pruebas escritas. Si alguna prueba falla, el código se modifica hasta que pase todas las pruebas. Este ciclo de desarrollo iterativo garantiza que el código sea limpio, robusto y simple. TDD pretende que el desarrollo sea más rápido, eliminando duplicación de código y permitiendo escribir código nuevo solo cuando la prueba falle. Las fases de TTD incluyen: desarrollo y escritura de la prueba, donde se define de forma clara cuáles son los requerimientos necesarios para finalizar la escritura de código, se la conoce como fase red por los errores que se destacan en ese color; la validación de pruebas consiste en verificar que los test se cumplan, se la conoce como fase *green* y la refactorización para verificar que se cumpla con las buenas prácticas y se verifique un código limpio. Algunas ventajas de TDD son las siguientes: reducción de errores, detección de requisitos no especificados, eliminación de código

duplicado, menor coste de mantenimiento y redundancia, mayor productividad y documentación, etc. (TDD, 2024).

xxiii. Modelo de 3 niveles

En la arquitectura de tres niveles se organiza la aplicación en tres niveles informáticos lógicos y físicos: el nivel de presentación o interfaz de usuario, el nivel de aplicación donde se procesan los datos y el nivel de datos donde se almacenan y gestionan los datos asociados con la aplicación. En cuanto a responsabilidad, desarrollo, escalabilidad y mantenimiento, cada nivel es independiente del resto.

- Nivel de presentación: corresponde a la interfaz con la que el usuario puede interactuar con la aplicación. Debe mostrarle información y recopilar datos de él. Se puede ejecutar en un navegador web. Se encarga del manejo de solicitudes y respuestas HTTP, validación de la entrada y formateo de salida. Los controladores (*controllers)* corresponden al nivel de presentación, gestionan las rutas y endpoints de la API generando las respuestas en formato JSON o XML.
- Nivel de aplicación: también conocido como nivel lógico de negocio, es el núcleo de la aplicación. En este nivel se procesa, utilizando la lógica empresarial, la información recopilada en el nivel de presentación contra otra información obtenida del nivel de datos. También puede añadir, suprimir o modificar datos en el nivel de datos. Los servicios (services) corresponden al nivel de aplicación, gestionan transacciones y controlan los flujos de trabajo.
- Nivel de datos: es donde se almacena y gestiona la información procesada por la aplicación, interactúa con bases de datos u otros sistemas de almacenamiento para leer, crear, actualizar o eliminar datos. Los repositorios (*repository*) o DAOs (*Data Access Object*) corresponden al nivel de datos (Arquitectura de tres niveles, 2024).

xxiv. Patrones en la arquitectura BE

A continuación, se detallan conceptos de la arquitectura BE sobre la que se desarrollará la aplicación: FCD, Service Layer, Repository Pattern, DAO. Los patrones de diseño buscan solucionar problemas recurrentes de diseño, aportan una forma estandarizada de solucionarlos. Proveen un lenguaje común de comunicación entre programadores y logran mejorar la mantenibilidad y flexibilidad de los proyectos (Patrones de diseño, 2024).

- FCD: Facade es un patrón de diseño que tiene como objetivo ser intermediario entre consumidores y distintos sistemas. Esto permite simplificar el código del consumidor, ocultar los componentes del sistema reduciendo las dependencias, facilitar el cambio gracias al bajo acoplamiento (Facade, 2024).

- Service Layer: es un patrón arquitectónico que separa la lógica empresarial de la infraestructura subyacente; además, es una capa de abstracción que contiene los componentes de la aplicación que proporcionan servicios a la capa superior y permite concentrarse en la lógica de negocio y la reutilización de código. Puede modificarse o sustituirse sin afectar el código de los demás componentes (Capa de Servicio, 2024).

- Repository Pattern: es un patrón de diseño que proporciona una forma de administrar la lógica de acceso a los datos. Busca simplificar el código de acceso a los datos y separarlo de la lógica empresarial, reduciendo la duplicación de código y su mantenimiento (Repository Pattern, 2024).

- DAO: Data Access Object es un patrón de diseño que se utiliza para separar la lógica de persistencia de datos. El servicio se mantiene independiente de cómo se realizan las operaciones de bajo nivel para acceder a las bases de datos (DAO, 2024).

xxv. Jest

Jest es un framework de pruebas unitarias para JavaScript, especialmente popular en entornos Node.js. Las pruebas unitarias son un tipo de pruebas de software, que se centran en verificar el funcionamiento de componentes individuales de código, generalmente funciones o métodos de manera aislada para asegurar que cada unidad funcione correctamente. Jest está bien documentado, requiere poca configuración para funcionar de inmediato y puede ampliarse para adaptarse a requisitos y ejecuta los test en paralelo en sus propios procesos. Además, ejecuta primero las pruebas que fallaron antes y las reorganiza en función de cuánto tiempo tardaron. Por su parte, permite evaluar la cobertura de código de proyectos completos sin configuraciones adicionales y provee integración directa con mocks, lo que facilita la simulación de cualquier objeto fuera del alcance de la prueba.

Los mocks son objetos falsos que imitan el comportamiento de objetos reales, permitiendo probar el código en aislamiento sin depender de componentes externos. Jest se puede complementar con la API “Mock Functions” para espiar llamadas a funciones con

código fácilmente legible. Cuando las pruebas fallan, Jest provee una explicación detallada, mostrando los valores esperados y obtenidos (Jest, 2024).

xxvi. Joi

La validación de datos y estructuras en aplicaciones de software maximiza la integridad, seguridad y eficiencia de los sistemas. La validación garantiza que los datos proporcionados cumplan con ciertos criterios y que las estructuras de datos sean conforme a las expectativas del sistema. Joi es una biblioteca para JavaScript que facilita la definición y aplicación de reglas para la validación de datos. Joi simplifica la validación de datos de entrada permitiendo definir reglas claras y específicas que deben cumplir los datos antes de ser procesados. Asegura que los datos y objetos estén en el formato correcto, definiendo y validando los mismos. Además, simplifica el manejo de errores devolviendo información detallada y respuestas adecuadas sobre reglas no cumplidas. Para entender el funcionamiento de Joi se deben comprender algunos conceptos claves:

- Un *Schema* define la estructura de datos esperada y sus reglas de validación, campos permitidos, tipos de datos y condiciones a cumplir.
- *Types* son los diferentes tipos de datos (cada uno con reglas específicas) que se pueden validar, como strings, números, objetos, arrays, etc.
- *Rules* son condiciones o restricciones que se aplican a un campo específico en un esquema, como longitud, patrones de coincidencia, valores permitidos, etc.
- *Modifiers* son funciones que se aplican a un campo antes y después de una validación, como conversiones u operaciones adicionales.
- *Validation* consiste en validar que los datos proporcionados cumplen con las reglas definidas en el schema.
- *Errors proporcionan* información detallada sobre qué reglas no se cumplieron y en qué datos se encuentran los problemas.
- *Defaults* son los valores predeterminados que se asignan a un campo si no se proporciona un valor, Joi permite definirlo en el esquema.
- *Labels* permiten asignar nombres personalizados a los campos del esquema.
- *Examples* sirven para ilustrar como deben ser los datos que cumplan con el esquema (Joi, 2024).

xxvii. Sequelize

Sequelize es un ORM (Object Relational Mapper) para Node.js, que facilita el trabajo con base de datos relacionales. Un ORM realiza un mapeo entre los registros de base de datos y objetos para poder gestionar la información de la base de datos en Node.js. Sequelize proporciona soporte para sincronización de base de datos, asociaciones, transacciones, migraciones de base de datos a medida que reduce el tiempo de desarrollo y evita ataques de SQL Injections (Sequelize, 2024).

xxviii. Swagger

Swagger es una herramienta para documentar, diseñar y probar APIs de forma interactiva y visual. Es una documentación online de una API en la que se pueden ver, de manera detallada, todos los endpoints desarrollados. Swagger hace visibles los datos de entrada y permite probar los endpoints directamente en su interfaz. Esto es posible, haciendo clic en el botón “try it” de cada endpoint, ingresar los datos y hacer clic en el botón “execute”, lo que retornará el cURL y la respuesta a la acción ejecutada (Swagger, 2024).

xxix. Apache Jmeter

Jmeter es un software de código abierto, que tiene como objetivo probar el comportamiento funcional y medir el rendimiento. Puede simular una carga pesada en un servidor, red u objeto para probar su resistencia o analizar el rendimiento general bajo diferentes tipos de carga. Jmeter puede cargar y probar el rendimiento de diferentes aplicaciones, servidores y protocolos. Además, provee un IDE de prueba que permite guardar los datos de pruebas, una interfaz de línea de comandos, la generación de informes detallados y ofrece portabilidad completa. JMeter es ampliamente utilizado para probar aplicaciones web, servicios RESTful, bases de datos, servicios FTP, y más (Jmeter, 2024).

xxx. Sonarqube

Sonarqube es una plataforma de código abierto, que tiene como objetivo mejorar la calidad del código desarrollado proporcionando métricas que ayuden a detectar errores y vulnerabilidades de seguridad. Es una herramienta que realiza análisis estático de código fuente, lo que es ideal en la fase de testing. Sonarqube se integra perfectamente dentro de un contexto DevOps en el flujo de integración continuo. (SonarQube, 2024).

xxxi. AppScan

Hcl AppScan es un conjunto de herramientas diseñadas para identificar y remediar vulnerabilidades en aplicaciones mediante el análisis de seguridad, tanto estático como dinámico. Además, tiene la capacidad de analizar el código fuente, análisis de vulnerabilidades y detección de amenazas en tiempo real. El software se integra fácilmente en herramientas de desarrollo IDEs, así como con plataformas de integración y entrega continua (CI/CD), facilitando la automatización y mejora de seguridad durante todo el ciclo de desarrollo. (HCL AppScan, 2024)

6. Metodología de desarrollo

i. Scrum

Se puede pensar en Scrum como un marco de trabajo ágil en el que el equipo se enfoca en pequeñas partes a la vez de un problema complejo, con experimentación continua y ciclos de retroalimentación que tienen como objetivo mejorar y aprender a lo largo del camino. Scrum ayuda al equipo a lograr valor de forma incremental y colaborativa, proveyendo la estructura necesaria para que el equipo integre su forma de trabajo optimizando las necesidades específicas. El equipo Scrum está formado por un Product Owner (PO), un Scrum Master (SM) y Developers (Devs), cada uno tiene responsabilidades específicas. El Scrum Team participa de cinco eventos y produce tres artefactos. Además, en Scrum, los incrementos de trabajo valioso se entregan en ciclos cortos de un mes o menos, que se denominan Sprints y durante el ciclo Sprint, se produce retroalimentación continua, lo que permite la inspección y adaptación del proceso y de lo que se entregará. El Scrum Team y otros miembros de su organización, negocio, usuarios o base de clientes conocidos como stakeholders, inspeccionan los resultados del Sprint y los ajustan para el siguiente (What is Scrum?, 2024).

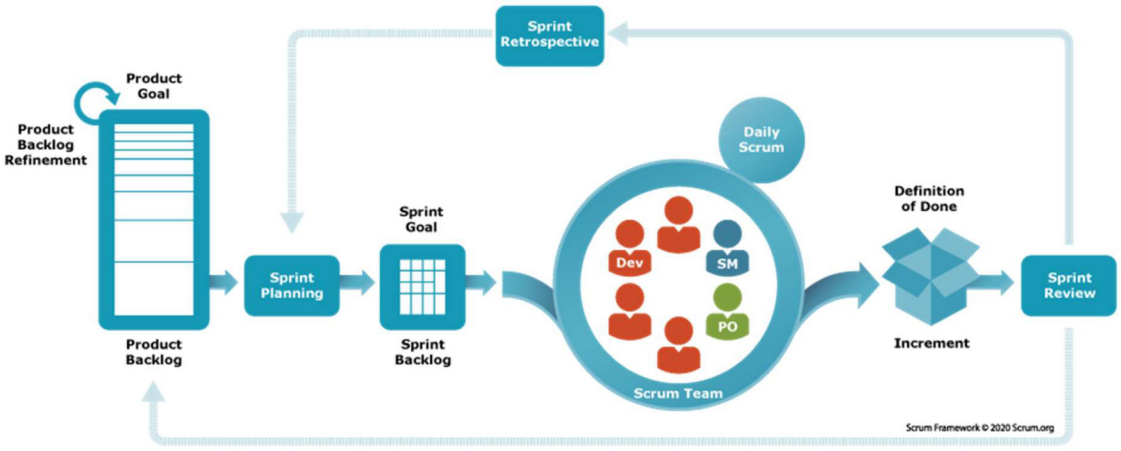


Figura 3. ¿Qué es Scrum?

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/2023-09/scrum-framework-9.29.23.png

ii. Scrum Team

El Scrum Team (equipo de Scrum) es multifuncional, sus miembros tienen todas las habilidades necesarias para crear valor. También es un equipo autogestionado, deciden internamente quién hace qué, cuándo y cómo. El equipo es responsable de todas las actividades relacionadas con el producto: lograr la colaboración, la verificación, operación, experimentación, investigación y desarrollo. De la misma forma, es responsable de lograr un incremento valioso en cada sprint (The Scrum Team, 2024).

1. Scrum Master (SM)

El Scrum Master es el responsable de establecer Scrum, ayudando a todos a comprender la teoría y la práctica de Scrum. En primer lugar, el SM ayuda al Scrum Team entrenándolo en autogestión y funcionalidad cruzada, además, es responsable de ayudar a sus equipos a tener éxito. También, como parte de su función, debe eliminar impedimentos del equipo, asegurar que se cumplan los eventos, y que dichos eventos estén dentro del plazo establecido. En segundo lugar, ayuda al PO proveyendo técnicas para la definición de objetivos y gestión del backlog del producto, estableciendo una planificación empírica, facilitando la colaboración de las partes interesadas (stakeholders). En tercer lugar, ayuda a la organización, liderándolos, capacitándolos, y entrenándolos en su adopción de Scrum,

eliminando barreras entre las partes interesadas y los equipos Scrum. En cualquier situación, un Scrum Master utiliza habilidades sociales para actuar como líder, facilitador, entrenador, gerente, mentor, maestro, eliminador de impedimentos o agente de cambio, dependiendo de esa situación (What is a Scrum Master?, 2024).

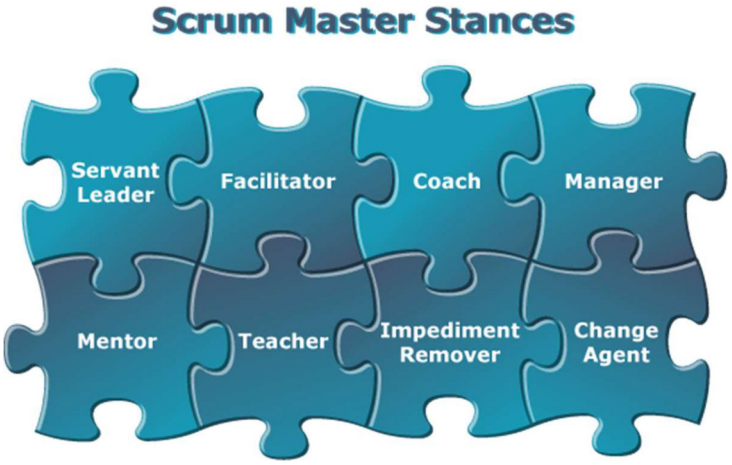


Figura 4. Posturas de Scrum Master

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/2022-11/asset_1scrum_framework.png

2. Product Owner (PO)

Un Product Owner (PO - dueño del producto) es el responsable de maximizar el valor del producto resultante del trabajo del equipo, brindar claridad al equipo sobre la visión y el objetivo de un producto, finalmente, es lo que determina la priorización para ofrecer valor a todas las partes interesadas (stakeholders). También, identifica, mide y maximiza el valor durante todo el ciclo de vida del producto. El PO es responsable de la gestión eficaz del Product Backlog ya que debe desarrollar y comunicar explícitamente el objetivo del producto, crear y comunicar claramente y ordenar los elementos del Product Backlog, finalmente, asegurarse de que el Product Backlog sea transparente, visible y comprendido. Además, de la gestión del backlog, el PO debe obtener apoyo de la organización para respaldar las decisiones que tome, a través del incremento de trabajo compartido en la Sprint Review. El PO puede adoptar varias posturas frente a diferentes situaciones para lograr su objetivo final de maximizar el valor. Estas posturas incluyen el visionario, el colaborador, el representante del cliente, el tomador de decisiones, el experimentador y el influyente (What is a Product Owner?, 2024).

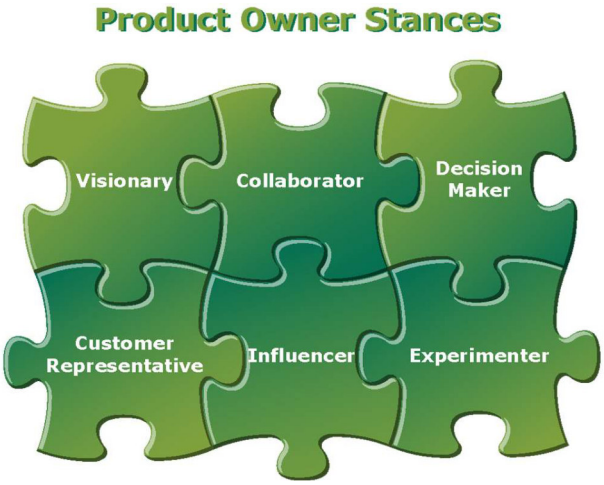


Figura 5. Posturas del Product Owner

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/2022-12/product-owner-stances-graphic-v2_0.png

3. Developer (Dev)

Los Developers (Desarrolladores) son las personas del equipo Scrum que se comprometen a crear cualquier aspecto de un incremento utilizable en cada Sprint. No son, necesariamente, desarrolladores de software, dado que pueden ser parte del diseño, la construcción, la prueba o el envío del producto. Un desarrollador es el responsable de la generación del Sprint Backlog, de inculcar calidad adhiriéndose a una definición de hecho (Definition of Done), adaptar su plan cada día para cumplir con el Sprint y responsabilizarse entre miembros, como profesionales. Los desarrolladores, en ocasiones, deben adoptar roles como facilitadores, tutores y coaching (What is a Developer?, 2024).

iii. Scrum Events

El primero de los Scrum Events (eventos de Scrum) que se describe es el Sprint. El Sprint es el contenedor para todos los demás eventos y cada evento es una oportunidad para inspeccionar y adaptar los artefactos de Scrum. Estos eventos están diseñados para permitir la transparencia requerida, se utilizan para crear regularidad y minimizar las reuniones no definidas en Scrum (The Scrum Events, 2024).

1. Sprint

Un Sprint es un periodo de trabajo de duración fija. En la búsqueda de minimizar riesgos, lograr coherencia y garantizar iteraciones breves de retroalimentación acerca de

cómo se realiza el trabajo y en qué se está trabajando, un Sprint suele durar a lo sumo, un mes. No obstante, un nuevo Sprint comienza inmediatamente luego de la finalización del Sprint anterior y dentro de este se realiza el trabajo necesario para lograr el objetivo del producto, la planificación del Sprint, los Scrum diarios, la revisión del Sprint y la retrospectiva del Sprint. Durante su desarrollo, no se realizan cambios que puedan poner en peligro su propio objetivo, la calidad no disminuye, se realiza el refinamiento del Product Backlog y el alcance se puede aclarar y renegociar con el PO. Los Sprints permiten previsibilidad al poder adaptar:

- **Product Goal** (Objetivo del Producto): describe un estado futuro del producto y puede servir para como objetivo para que el equipo Scrum planifique, está en el Product Backlog, el resto del Producto Backlog surge para definir el "qué" cumplirá el Objetivo del Producto.

- **Sprint Goal** (Objetivo del Sprint): es un único objetivo y es un compromiso de los desarrolladores, proporciona flexibilidad en términos de trabajo exacto, es decir, se pueden redefinir las tareas a realizar para cumplir con el *Sprint Goal*. Al contar con un único objetivo, se alienta al Scrum Team a trabajar juntos en lugar de apuntar a lograr objetivos individuales.

Al adoptar el empirismo, la experiencia permite al equipo aprender en entornos complejos y mejorar a medida que avanza (What is a Sprint?, 2024).

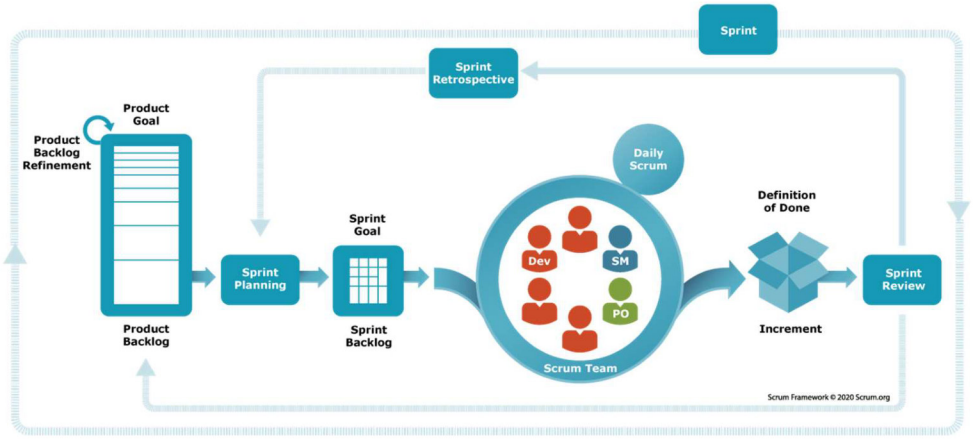


Figura 6. ¿Qué es un Sprint?

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/2023-10/the-sprint-2.png

2. Sprint Planning

El Sprint Planning (Planificación del sprint) inicia el Sprint, establece el trabajo a realizar y se crea mediante el trabajo colaborativo de todo el equipo. En este evento, el PO se asegura de que todos los participantes estén al tanto de los elementos principales del Product Backlog y su relación con el objetivo del producto. Durante el Sprint Planning se define el Objetivo del Sprint que representa el valor de este, además, junto con el Product Owner, se seleccionan y perfeccionan los elementos del Product Backlog para incluirlos en el Sprint actual, finalmente, los desarrolladores descomponen los elementos seleccionados. El Sprint Backlog es el conjunto del objetivo del Sprint, los elementos del Product Backlog y el plan de entregarlos (What is Sprint Planning?, 2024).

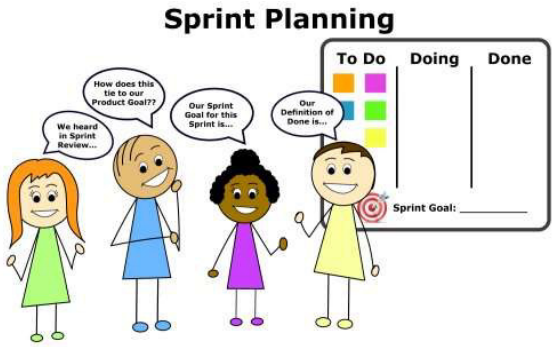


Figura 7. Sprint Planning

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/s3fs-public/styles/cke_media_resize_medium/public/2023-11/Sprint-Planning-v2.jpg?itok=sIFIEfjr

3. Daily Scrum

El propósito de la Daily Scrum es inspeccionar el progreso hacia el objetivo del Sprint y adaptar el Sprint Backlog, ajustando el próximo trabajo planificado. Es un evento de 15 minutos, para el equipo de desarrolladores, el Producto Owner y Scrum Master pueden participar o no. La Daily Scrum debe centrarse en el progreso del objetivo del Sprint y sobre el plan para el siguiente día de trabajo, lo que mejora la concentración y autogestión. La Daily Scrum identifica impedimentos, mejora las comunicaciones, elimina la necesidad de otras reuniones y promueve la rápida toma de decisiones (What is a Daily Scrum?, 2024).

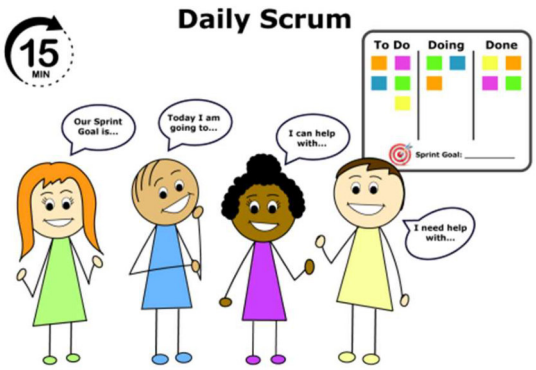


Figura 8. Daily Scrum

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/s3fs-public/styles/cke_media_resize_medium/public/2023-11/The-Daily-Scrum-2.0.png?itok=-F-lrW-U

4. Sprint Review

El Sprint Review tiene como objetivo inspeccionar el resultado del Sprint y determinar adaptaciones. El equipo presenta los resultados de su trabajo a los interesados y analiza el progreso al Objetivo del Negocio. Por una parte, los interesados pueden colaborar sobre qué hacer a continuación, pudiéndose ajustar el Backlog. Por otra parte, los miembros del equipo explican los elementos que se han terminado y cuáles no, además, responden preguntas acerca del trabajo terminado y el Product Owner proyecta fechas de entrega a partir del trabajo terminado. El resultado del Sprint Review es un Product Backlog revisado que permite determinar los elementos para el próximo Sprint (What is a Sprint Review?, 2024).

Figura 9. Sprint Review

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/s3fs-public/styles/cke_media_resize_medium/public/2023-11/Sprint-Review-v2.jpg?itok=Eet6b9gX

5. Sprint Retrospective

La Sprint Retrospective tiene como objetivo aumentar la calidad y la eficacia. El equipo Scrum analiza cómo fue el último Sprint acerca de las personas, interacciones, procesos, herramientas y su “Definition of Done”. Se analiza qué salió bien, qué problemas se encontraron y cómo se resolvieron, en el caso de que se hayan resuelto. Se identifican los cambios más útiles para mejorar la efectividad, se pueden agregar esos cambios al Product Backlog para el próximo Sprint. La Sprint Retrospective concluye el Sprint, y se toma compromiso para mejorar en el próximo. El Scrum Master anima a mejorar el proceso para hacerlo más agradable y efectivo para el próximo Sprint (What is a Sprint Retrospective?, 2024).

Figura 10. Sprint Retrospective

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/s3fs-public/styles/cke_media_resize_medium/public/2023-11/Sprint-Retrospective.png?itok=RNJKC2NK

iv. Scrum Artifacts

Cada artefacto de Scrum representa trabajo o valor y están diseñados para maximizar la transparencia de la información que proveen, junto con un enfoque que permite medir el progreso, con el objetivo de cumplir su compromiso. Los compromisos establecen las responsabilidades del Product Owner, Scrum Master y Developers.

1. Product Backlog y Product Goal

La única fuente de trabajo para el equipo es el Product Backlog. Este remite a una lista ordenada de lo que el equipo necesita realizar para mejorar el producto, por lo que el Product Backlog va evolucionando a medida que suceden los Sprints. En el Sprint Planning se seleccionan los elementos del Product Backlog que el Scrum Team puede realizar durante ese Sprint y durante el refinamiento del Sprint se desglosan los elementos del Product

Backlog en otros más pequeños y precisos. El Product Owner ayuda a los Developers a comprender los elementos y los desarrolladores son los responsables del dimensionamiento de cada elemento del Product Backlog (What is a Product Backlog?, 2024).

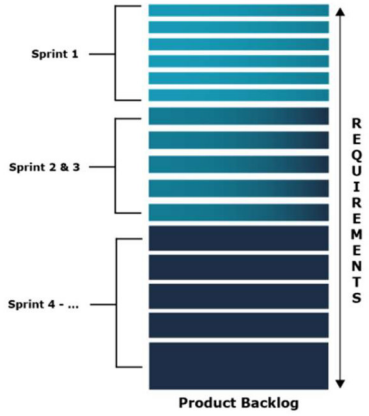


Figura 11. Product Backlog

Fuente: https://scrumorg-website-prod.s3.amazonaws.com/drupal/s3fs-public/styles/cke_media_resize_medium/public/2023-11/Product%20Backlog.png?itok=JYzkqEbq

El compromiso del Product Backlog es el Product Goal. De cierta forma, el Product Goal describe un estado futuro del producto, es el objetivo a largo plazo, debe ser claro y conciso, dado que es una declaración que proporciona contexto, dirección y propósito para el Scrum Team y los stakeholders. El Product Goal debe ser visible para todos y el responsable de crearlo y comunicarlo es el Product Owner. En un momento dado, el Product Goal es único en todo el Product Backlog. En el Sprint Review, los participantes inspeccionan el incremento y el progreso realizado hacia el Product Goal (What is a Product Goal?, 2024).

2. Sprint Backlog y Sprint Goal

El Sprint Backlog es un plan para los Developers, que demuestra cuál es el trabajo que ellos planean realizar durante el Sprint para lograr el Sprint Goal. El Sprint Backlog se actualiza a medida que se avanza en el mismo y debe estar suficientemente detallado para poder ser inspeccionado durante el Daily Scrum (What is a Sprint Backlog?, 2024).

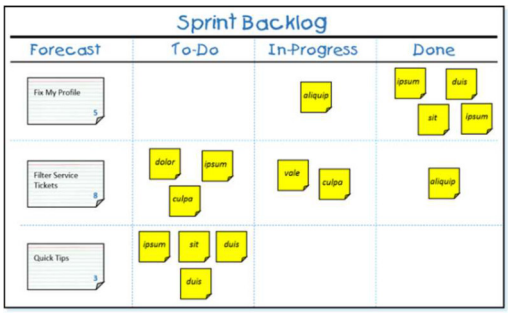


Figura 12. Sprint Backlog

Fuente: https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-sprint-backlog

El Sprint tiene un único objetivo que es el Sprint Goal, este proporciona flexibilidad e indica cuál es trabajo real para lograrlo. El Sprint Goal enfoca a todo el Scrum Team y evita que el equipo trabaje en iniciativas separadas. Este, se crea durante la Scrum Planning y se agrega al Sprint Backlog. El Sprint Goal es una guía para el trabajo de los Developers durante el Sprint y, junto con Product Owner, pueden negociar el alcance del Sprint Backlog, sin modificarlo. Es importante que el Sprint Goal es lo que el equipo pretende lograr durante el Sprint y este debe ser alcanzable para evitar un impacto negativo (What is a Sprint Goal?, 2024).

3. Increment y Definition of Done

Un Incremento es un paso hacia el Product Goal y se suma a todos los incrementos anteriores, además, para proporcionar valor, un incremento debe ser utilizable. Se pueden crear varios incrementos dentro de un Sprint, todos ellos se suman y se muestran en la Sprint Review.

La Definition of Done es el compromiso para el Increment y contiene todas las características y estándares para que un Increment pueda ser liberado. También, brinda a todos la comprensión compartida de qué es lo que debe cumplirse para afirmar que el trabajo se completó y qué estándares se cumplieron.

v. Jira

Jira es una herramienta para la gestión de proyectos ágiles y es la única fuente de información de todo el ciclo de vida del desarrollo, además, de permitir a los equipos trabajar de forma comunicativa, colaborativa y coordinada. Jira es compatible con la metodología ágil

de gestión de proyectos de desarrollo de software, provee interfaces para la planificación ágil, tableros scrum personalizables, junto con herramientas para estimar, crear informes y medir la velocidad de trabajo. Con Jira se pueden crear tableros de Scrum para gestionar proyectos complejos bajo objetivos únicos y promoviendo entregas iterativas e incrementales (Jira Software, 2024).

7. Ejecución del cronograma

a. Contexto

El desarrollo de software que se describirá es realizado en un equipo ágil conformado por desarrolladores, Quality Assurance (QA), una SM y una PO. El Scrum Team desarrolla soluciones para varios proyectos en paralelo y este es solo uno de ellos. Además, del Scrum Team, se irá interactuando con otros sectores del área de tecnología de la empresa que se describirán oportunamente. En las distintas ceremonias de scrum y otras reuniones con los usuarios se relevó la necesidad de desarrollo para lograr obtener un listado inicial de historias. A continuación, se detallarán estas historias, ya que servirán como un punto de inicio para la ejecución de las tareas. A medida que se avance en los distintos sprints estas historias se irán refinando en historias más detalladas conforme se vaya adquiriendo mayor conocimiento del producto.

b. Tarea 1 - Historias

Como se ha descrito recientemente, las historias a listar serán refinadas conforme se avance con el proyecto y se logre un mejor entendimiento de las particularidades del producto. Jira es la herramienta que se utiliza para gestión del proyecto.

El listado inicial de historias es:

H1	Como equipo de desarrollo, quiero diseñar el modelo de datos y crear las estructuras de datos para satisfacer las necesidades descritas en el relevamiento.
H2	Como equipo de desarrollo, quiero diseñar los procesos y sus relaciones para satisfacer las funcionalidades relevadas.
H3	Como equipo de desarrollo, quiero desarrollar los procedimientos almacenados, triggers y jobs para cumplir con el diseño elaborado.
H4	Como equipo de desarrollo quiero realizar el diseño macro de los microservicios a desarrollar para lograr comprender el alcance de las tareas.
H5	Como equipo de desarrollo quiero diseñar y desarrollar los microservicios en ambientes bajos para interactuar con el desarrollo de base de datos.

H6	Como equipo de desarrollo quiero cumplir con la ejecución de pruebas del equipo QA de la mesa y cumplir con los requisitos de seguridad para implementar en ambientes productivos.
H7	Como equipo de desarrollo quiero realizar la implementación de la solución, apagado y migración de datos para finalizar con las tareas planificadas.

Tabla 1. Historias, primera iteración

c. Tarea 2 – Modelo de datos y Tablas

En la ceremonia de planificación del primer Sprint se ha determinado incluir la H1, esta tiene el suficiente detalle como para no requerir refinamiento. La definición lograda está basada en el relevamiento del comportamiento del sistema actualmente productivo y en las mejoras que pretenden introducir los usuarios. A continuación, se detalla la escritura de la H1.

H1: Como equipo de desarrollo, quiero diseñar el modelo de datos y crear las estructuras de datos para satisfacer las necesidades descritas en el relevamiento.
• Como usuario quiero guardar información acerca de las cuentas que poseen el beneficio para toma de decisiones. • Como usuario quiero guardar información acerca del estado de la gestión del alta y baja del beneficio para controles operativos. • Como usuario quiero poder otorgar el beneficio a una cuenta sin tener que migrarla de cliente para fidelizar el cliente para mejorar la experiencia del cliente. • Como usuario quiero registrar información de la gestión para resguardar la trazabilidad. • Como usuario quiero registrar información de comunicaciones para resguardar interacciones con el usuario. • Como usuario quiero escribir las sentencias SQL para registrar la información descripta anteriormente.

La implementación actual permite realizar una solicitud de alta o baja de beneficio. Por un lado, permite conocer las cuentas relacionadas de un cliente y los clientes relacionados a un tipo y número de documento determinado (o persona física). Por otro lado, se accede a conocer el estado de asignación del beneficio para una cuenta determinada, y la posibilidad de acceder al mismo. También, permite relevar el estado de una solicitud pendiente de alta o baja de beneficio. El nuevo modelo de datos a diseñar deberá reflejar las

capacidades de la implementación actual y eliminar la restricción del sistema productivo, de tener que migrar de cliente a una cuenta determinada para poder acceder al beneficio.

Se deberá registrar en un servidor SqlServer la gestión de los beneficios, la relación entre los grupos de cuentas de los clientes, las comunicaciones y la trazabilidad de la gestión. Para poder gestionar el alta y baja de beneficios se deberá reutilizar un circuito de asignación de beneficios, de la misma manera, para registrar las comunicaciones se reutilizará un circuito de envío de comunicaciones.

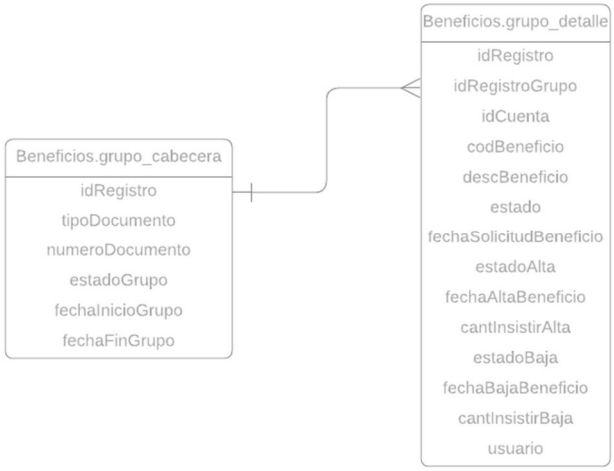


Figura 13. Modelo de datos base

Fuente: Elaboración propia, basada en la práctica.

El modelo de datos diagramado no incluye las tablas de procesos de solicitud de alta y baja de beneficios ni de comunicaciones. Según el análisis realizado hasta esta instancia, se ha determinado que las tablas del modelo actual solo registrarán datos de personas que ya hayan solicitado el beneficio. No se registran datos de personas que aún no cuenten con el beneficio. Esto se debe a que los datos de los clientes se registran en otro sistema, externo a la base de datos, que se consumirá en la etapa de desarrollo de microservicios.

Basándonos en el modelo descrito recientemente, se escriben las sentencias SQL para la creación de las tablas con sus respectivos índices.

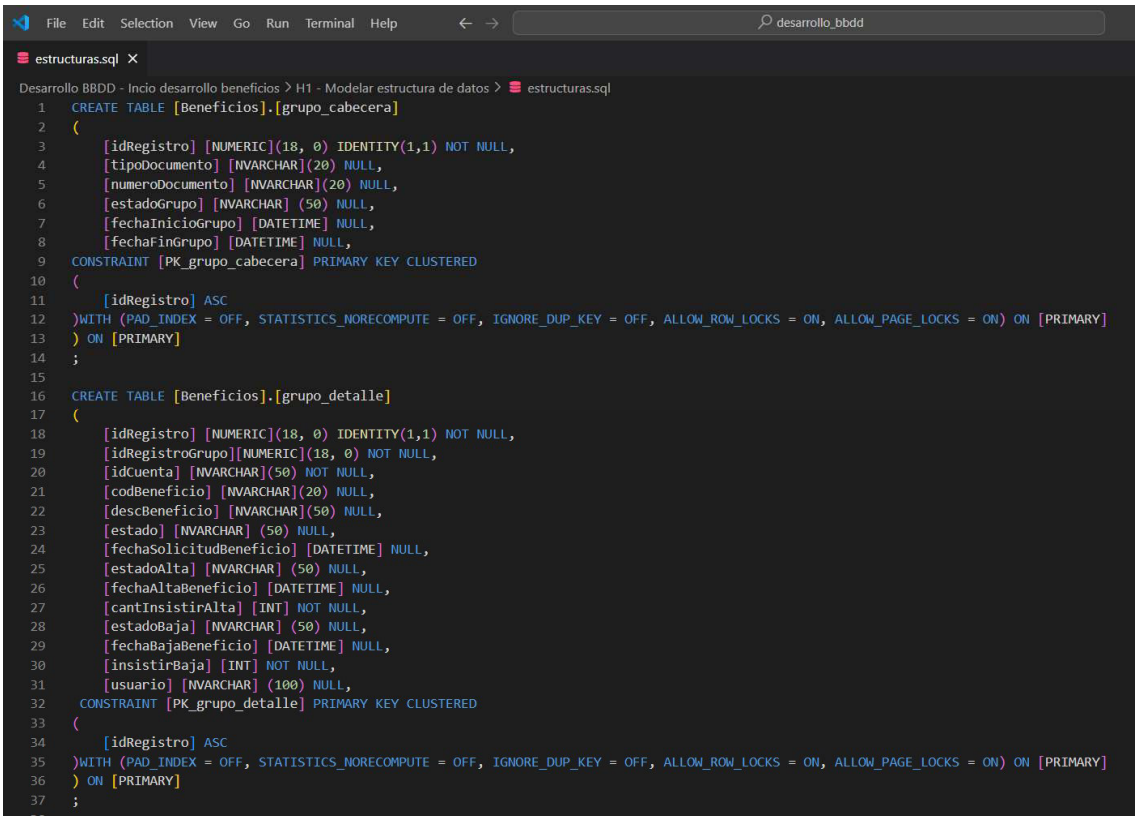


Figura 14. Tablas del modelo en Visual Studio Code

Fuente Elaboración propia, basada en la práctica.

d. Tarea 3 y tarea 4 – Diseño y Desarrollo de base de datos.

Posteriormente, se detalla la escritura de la H2.

H2: Como equipo de desarrollo, quiero diseñar los procesos necesarios y sus relaciones para satisfacer las funcionalidades relevadas.

A continuación, se enumeran los procesos necesarios para poder gestionar el alta y baja de beneficios para los clientes.

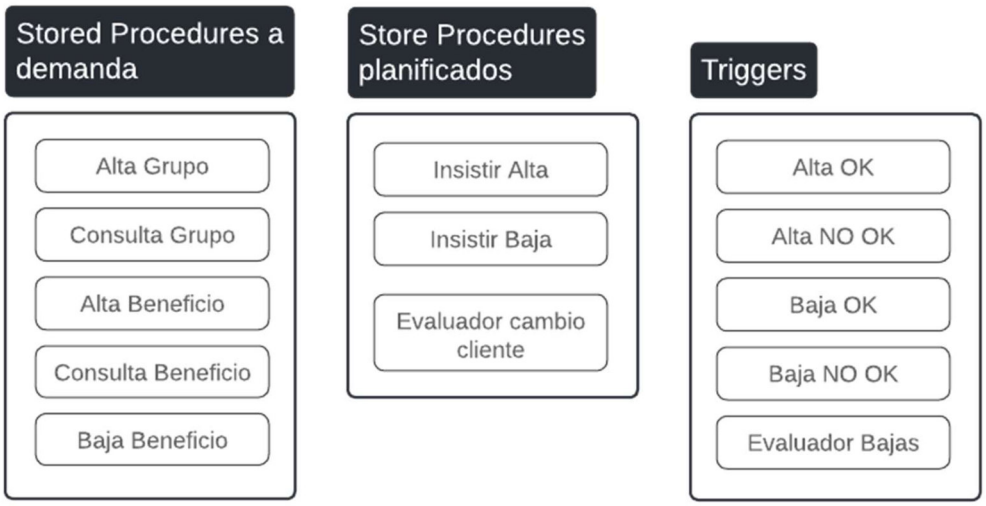


Figura 15. Stored Procedures

Fuente Elaboración propia, basada en la práctica.

A continuación, se describe el diseño de la interacción entre los procesos, triggers, tablas previamente detalladas.

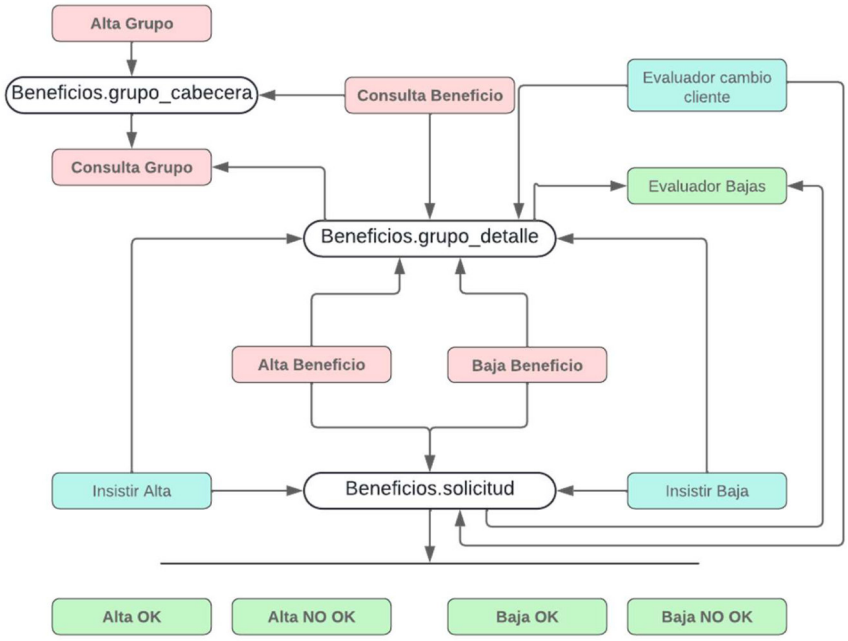


Figura 16. Relaciones de entidades de base de datos

Fuente Elaboración propia, basada en la práctica.

Se detalla la escritura de la H3.

H3: Como equipo de desarrollo, quiero desarrollar los procedimientos almacenados, triggers y Jobs para cumplir con el diseño elaborado.

Stored Procedures a demanda:

1. Como usuario quiero dar de alta un grupo de cuentas de un cliente para registrar aquellas que tienen el beneficio.
2. Como usuario quiero conocer todas las cuentas de un cliente integrantes de un grupo para poder solicitar altas y bajas.
3. Como usuario quiero registrar la solicitud de alta del beneficio de una cuenta determinada para otorgar el beneficio al cliente.
4. Como usuario quiero conocer el estado de asignación del beneficio de una cuenta determinada para tomar decisiones operativas.
5. Como usuario quiero registrar la solicitud de baja del beneficio de una cuenta determinada para quitar el beneficio al cliente.

Stored Procedures planificados:

1. Como usuario quiero que las solicitudes de alta rechazadas sean reintentadas por 30 días para otorgar el beneficio a la cuenta.
2. Como usuario quiero que las solicitudes de baja rechazadas sean reintentadas por 30 días para quitar el beneficio a la cuenta.
3. Como usuario quiero que se dé la baja el beneficio a una cuenta que cambio de titular para cumplir con las condiciones comerciales.

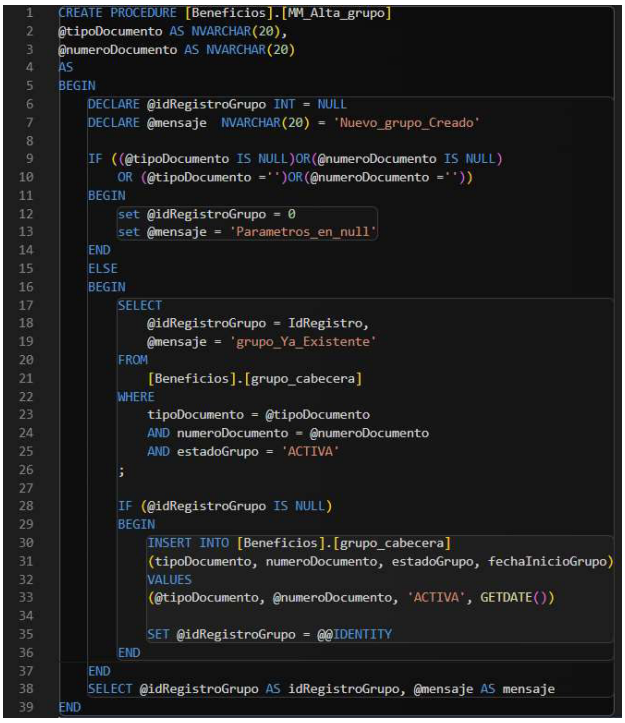
Trigger

1. Como usuario quiero que se registre la confirmación de la solicitud de alta del beneficio para finalizar la gestión de alta.
2. Como usuario quiero que se registre el rechazo de la solicitud de alta del beneficio para continuar con los reintentos.
3. Como usuario quiero que se registre la confirmación de la solicitud de baja del beneficio para finalizar la gestión de baja.
4. Como usuario quiero que se registre el rechazo de la solicitud de baja del beneficio para continuar con los reintentos.
5. Como usuario quiero que se detecten las confirmaciones de baja para evaluar si se alcanzó el mínimo de integrantes en el grupo.

Los Stored Procedures a demanda serán llamados desde los microservicios desarrollados en etapas posteriores, ante la necesidad de un usuario.

1. Alta Grupo: un grupo está integrado por una cantidad limitada de cuentas de un cliente, todos los integrantes del grupo tienen el beneficio o se está gestionando el alta del beneficio. La cantidad mínima de integrantes de un grupo es dos y la cantidad

máxima es determinada por los usuarios de negocio (se registra en una tabla de configuración). Se podrá solicitar el alta de un grupo si el cliente aún no tiene activo su grupo. Ante la solicitud de creación de un grupo, se crea un nuevo registro en la tabla beneficios.grupo_cabecera.



```
CREATE PROCEDURE [Beneficios].[MM_Alta_grupo]
@tipoDocumento AS NVARCHAR(20),
@numeroDocumento AS NVARCHAR(20)
AS
BEGIN
    DECLARE @idRegistroGrupo INT = NULL
    DECLARE @mensaje  NVARCHAR(20) = 'Nuevo_grupo_Creado'

    IF ((@tipoDocumento IS NULL)OR(@numeroDocumento IS NULL)
        OR (@tipoDocumento ='')OR(@numeroDocumento =''))
    BEGIN
        set @idRegistroGrupo = 0
        set @mensaje = 'Parametros_en_null'
    END
    ELSE
    BEGIN
        SELECT
            @idRegistroGrupo = IdRegistro,
            @mensaje = 'grupo_Ya_Existente'
        FROM
            [Beneficios].[grupo_cabecera]
        WHERE
            tipoDocumento = @tipoDocumento
            AND numeroDocumento = @numeroDocumento
            AND estadoGrupo = 'ACTIVA'
        ;

        IF (@idRegistroGrupo IS NULL)
        BEGIN
            INSERT INTO [Beneficios].[grupo_cabecera]
            (tipoDocumento, numeroDocumento, estadoGrupo, fechaInicioGrupo)
            VALUES
            (@tipoDocumento, @numeroDocumento, 'ACTIVA', GETDATE())

            SET @idRegistroGrupo = @@IDENTITY
        END
    END
    SELECT @idRegistroGrupo AS idRegistroGrupo, @mensaje AS mensaje
END
```

Figura 17. Procedure alta grupo

Fuente Elaboración propia, basada en la práctica.

2. Consulta Grupo: la consulta del grupo permitirá conocer a todas las cuentas integrantes de un grupo, el identificador del grupo, el estado de asignación del beneficio, el estado del alta y de la baja.

```sql
ALTER PROCEDURE [Beneficios].[MM_Consulta_Grupo]
@tipoDocumento NVARCHAR (20),
@numeroDocumento NVARCHAR(20)
AS
BEGIN
    SELECT
        DISTINCT
        c.tipoDocumento,
        c.numeroDocumento,
        d.idCuenta,
        c.IdRegistro idRegistroGrupo,
        d.estadoGrupo,
        d.estado_alta,
        d.estado_baja
    FROM

        [Beneficios].[grupo_cabecera] c
        INNER JOIN [Beneficios].[grupo_detalle] d
        ON c.IdRegistro = d.idRegistroGrupo
        AND d.ESTADO != 'BAJA'
    WHERE
        c.estadoGrupo = 'ACTIVO'
        AND c.tipoDocumento = @tipoDocumento
        AND c.numeroDocumento = @numeroDocumento
END
```

Figura 18. Procedure consulta grupo

Fuente Elaboración propia, basada en la práctica.

3. Alta Beneficio: para una cuenta determinada, el proceso de alta de beneficio permite registrar la solicitud de alta del beneficio, la fecha de solicitud, el usuario que solicitó el alta. Ante la solicitud de alta del beneficio, se crea un nuevo registro en *beneficios.grupo_detalle*. Adicionalmente, se crea un registro en la tabla *beneficios.solicitud*, esta tabla corresponde a un desarrollo previo y sirve de interfaz con un microservicio que gestiona solicitudes de alta y baja de todo tipo de beneficios que se pueden otorgar a una cuenta. Véase Figura 28. API Solicitud de Beneficios.

```sql
CREATE PROCEDURE [Beneficios].[MM_Alta_Beneficio]
@idRegistroGrupo INT,
@idCuenta NVARCHAR(50),
@usuario NVARCHAR (20)
AS
BEGIN
    --Declaraciones
    --Validaciones Varias
    BEGIN TRAN
        INSERT [Beneficios].[grupo_detalle]
        (idRegistroGrupo, idCuenta, codBeneficio, descBeneficio, fechaSolicitudBeneficio,
        estado, estadoAlta, cantInsistirAlta, cantInsistirBaja, usuario)
        VALUES
        (@idRegistroGrupo, @idCuenta, @codBeneficio, @descBeneficio, GETDATE(),
        'SOLICITADO','SOLICITADO', 0, 0, @usuario)

        SET @id_referencia = @@IDENTITY

        INSERT [Beneficios].[solicitud]
        (idCuenta, descBeneficio, codBeneficio, tip_solicitud, fec_solicitud, estado,
        id_referencia, tip_beneficio, usuario)
        VALUES
        (@idCuenta, @descBeneficio, @codBeneficio, 'AD', GETDATE(), 'SOLICITADO',
        @id_referencia, 'MM', @usuario)

        SET @mensaje = 'Beneficio solicitado'
    COMMIT
    END
    SELECT  @mensaje AS mensaje
END
```

Figura 19. Procedure alta beneficio

Fuente Elaboración propia, basada en la práctica.

4. Consulta Beneficio: para una cuenta determinada, el proceso de consulta de beneficio permite consultar el estado de asignación del beneficio, el indicador del grupo, la fecha de solicitud, de alta y de baja del beneficio, junto con los estados de alta y baja del beneficio.

```sql
ALTER PROCEDURE [Beneficios].[MM_Consulta_Beneficio]
@idCuenta NVARCHAR(50)
AS
BEGIN
    SELECT
        idRegistroGrupo,
        idCuenta,
        estado,
        fechaSolicitudBeneficio,
        fechaAltaBeneficio,
        estadoAlta,
        fechaBajaBeneficio,
        estadoBaja
    FROM
        [Beneficios].[grupo_cabecera] c
        INNER JOIN
        [Beneficios].[grupo_detalle] d
        ON c.idregistro = d.idRegistroGrupo
    WHERE
        c.estadoGrupo = 'ACTIVA'
        AND d.idCuenta = @idCuenta
        AND d.estado != 'BAJA'
END
```

Figura 20. Procedure consulta beneficio

Fuente Elaboración propia, basada en la práctica.

5. Baja Beneficio: para una cuenta determinada, el proceso de baja de beneficio permite registrar la solicitud de baja del beneficio, la fecha de solicitud y el usuario que solicita la baja. Ante la solicitud de baja de beneficio, se actualiza el registro existente para esa cuenta de la tabla *beneficios.grupo_detalle*.

```sql
CREATE PROCEDURE [Beneficios].[MM_Baja_Beneficio]
@idRegistroGrupo INT,
@idCuenta NVARCHAR(50),
@usuario NVARCHAR (20)
AS
BEGIN
--Declaraciones
--Validaciones
    BEGIN TRAN
        UPDATE [Beneficios].[grupo_detalle]
        SET estadoBaja = 'SOLICITADO'
        WHERE IdRegistro = @IdRegistroDet

        INSERT [Beneficios].[solicitud]
        (idCuenta, descBeneficio, codBeneficio, tip_solicitud, fec_solicitud, estado,
        idReferencia, tip_beneficio, usuario)
        VALUES
        (@idCuenta, @descBeneficio, @codBeneficio, 'RM', GETDATE(), 'SOLICITADO',
        @idReferencia, 'MM', @usuario)

        SET @mensaje = 'Baja de bono solicitado'
    COMMIT
    SELECT @mensaje AS mensaje
END
```

Figura 21. Procedure baja beneficio

Fuente Elaboración propia, basada en la práctica.

Los Stored Procedures planificados se ejecutarán diariamente dentro de un Job por el SQL Server Agent:

1. Insistir Alta: cabe aclarar que la solicitud de alta del beneficio que se registra en el proceso Alta Beneficio no implica, necesariamente, que el beneficio se otorgue. El resultado de esa solicitud se recibe en *beneficios.solicitud* como un OK ante la confirmación de la operación de alta de beneficio y un NOOK en caso contrario. En el caso de recibir un NOOK, la gestión de alta de beneficio continua por un circuito de reintentos hasta que el beneficio se otorgue o se cumpla el límite de reintentos.

```
CREATE PROCEDURE [Beneficios].[MM_InsistirAlta]
AS
BEGIN
    --Declaraciones
    DECLARE lote_insistir CURSOR FOR
        SELECT DISTINCT d.idregistro, d.cantInsistirAlta, d.idCuenta, d.codBeneficio
        FROM [Beneficios].[grupo_detalle] d
        WHERE
        d.estado = 'SOLICITADO' AND d.estadoAlta = 'ALTA_INSISTIR_API'
    OPEN lote_insistir
    FETCH NEXT FROM lote_insistir
    INTO @idregistro_det, @cantInsistirAlta, @idCuenta, @codBeneficio
    WHILE @@FETCH_STATUS = 0
    BEGIN
        --Validaciones
        UPDATE [Beneficios].[be_solicitud_bonos_por_api]
        SET estado='SOLICITADO',
            fec_update_solicitud_be = NULL
        WHERE id_referencia = @idregistro_det
        AND idCuenta=@idCuenta
        AND tip_solicitud='AD'
        AND tip_beneficio = 'GB_MM'
        FETCH NEXT FROM lote_insistir
        INTO @idregistro_det, @cantInsistirAlta, @idCuenta, @codBeneficio
    END
    CLOSE lote_insistir
    DEALLOCATE lote_insistir
END
```

Figura 22. Procedure Insistir alta

Fuente Elaboración propia, basada en la práctica.

2. Insistir Baja: de la misma manera, en la baja del beneficio, la solicitud no implica que se haya dado de baja el beneficio. El resultado de esa solicitud se recibe en *beneficios.solicitud* como un OK ante la confirmación de la operación de baja del beneficio y un NOOK en caso contrario. En el caso de recibir un NOOK, la gestión de baja de beneficio continua por un circuito de reintentos hasta que el beneficio se dé la baja o se cumpla el límite de reintentos.

```sql
ALTER PROCEDURE [Beneficios].[MM_cantInsistirBaja]
AS
BEGIN
    DECLARE lote_insistir CURSOR FOR
        SELECT DISTINCT d.idregistro, d.cantInsistirBaja, d.idCuenta, d.codBeneficio
        FROM [Beneficios].[grupo_detalle] d
        WHERE d.estado_baja IN ('BAJA_INSISTIR_API','BAJA_INSISTIR_API_CT')
    ;
    OPEN lote_insistir
    FETCH NEXT FROM lote_insistir
    INTO @idregistro_det, @cantInsistirBaja, @idCuenta_movil, @codBeneficio

    WHILE @@FETCH_STATUS = 0
    BEGIN
        --Validaciones
        UPDATE [Beneficios].[solicitud]
        SET estado='SOLICITADO',
            fec_update_solicitud_be = NULL
        WHERE id_referencia = @idregistro_det
        AND idCuenta=@idCuenta
        AND tip_solicitud='RM'
        AND tip_beneficio = 'MM'

        FETCH NEXT FROM lote_insistir
        INTO @idregistro_det, @cantInsistirBaja, @idCuenta, @codBeneficio
    END
    CLOSE lote_insistir
    DEALLOCATE lote_insistir
END
```

Figura 23. Procedure insistir baja

Fuente Elaboración propia, basada en la práctica.

3. Evaluador cambio cliente: una cuenta determinada pertenece a un cliente de la empresa, pero la cuenta puede asociarse a otro cliente si así la persona lo desea. El beneficio otorgado a una cuenta está condicionado al tipo y número de documento de la persona física. En el caso de que la cuenta se migre a otro cliente, se deberá gestionar la baja correspondiente del beneficio a la cuenta. Este proceso realiza la gestión descripta.

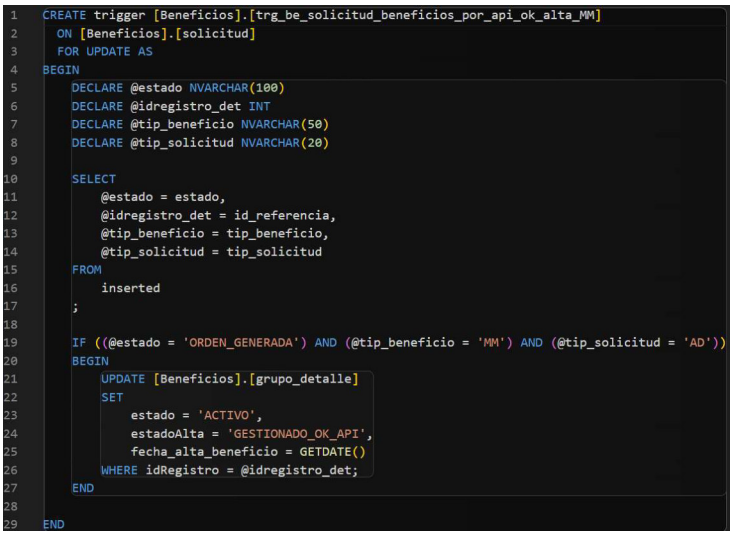
```sql
CREATE PROCEDURE [Beneficios].[MM_Baja_Condicion_Comercial]
AS
BEGIN
    --Declaraciones
    DECLARE CT_Cursor CURSOR FOR
    SELECT  fd.IdRegistro, fd.cuenta, fc.tipoDocumento, fc.numeroDocumento, p.tipoDocumento, p.numeroDocumento, fd.codBeneficio, fd.descBen
    FROM ···
    OPEN CT_Cursor
    FETCH NEXT FROM CT_Cursor
    INTO @IdRegistroDet, @cuenta, @fc_tipoDocumento, @fc_numeroDocumento, @p_tipoDocumento, @p_numeroDocumento, @fd_codBeneficio, @fd_descB
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (@fc_tipoDocumento != @p_tipoDocumento AND @fc_numeroDocumento != @p_numeroDocumento)
        BEGIN
            BEGIN TRAN
                UPDATE [totalizacion].[grupo_detalle]
                SET estado = 'BAJA', estadoBaja = 'SOLICITADO_CT'
                WHERE IdRegistro = @IdRegistroDet
                INSERT [totalizacion].[solicitud]
                (cuenta, descBeneficio, codBeneficio, tip_solicitud, fec_solicitud, estado, id_secuencialidad, tip_bono, usuario)
                VALUES
                (@cuenta, @fd_descBeneficio, @fd_codBeneficio, 'RM', GETDATE(), 'SOLICITADO', @IdRegistroDet, 'MM', @usuario)
            COMMIT
        END
        FETCH NEXT FROM CT_Cursor
        INTO @IdRegistroDet, @cuenta, @fc_tipoDocumento, @fc_numeroDocumento, @p_tipoDocumento, @p_numeroDocumento, @fd_codBeneficio, @fd_d
    END
    CLOSE CT_Cursor
    DEALLOCATE CT_Cursor
END
```

Figura 24. Procedure baja condición comercial

Fuente Elaboración propia, basada en la práctica.

Los triggers serán ejecutados ante eventos en la tabla donde se solicitan las altas y bajas de beneficios y la tabla grupo detalle:

1. Alta OK: el trigger registrará la respuesta positiva del alta del beneficio para una cuenta determinada y establecerá el estado del beneficio, correspondientemente.

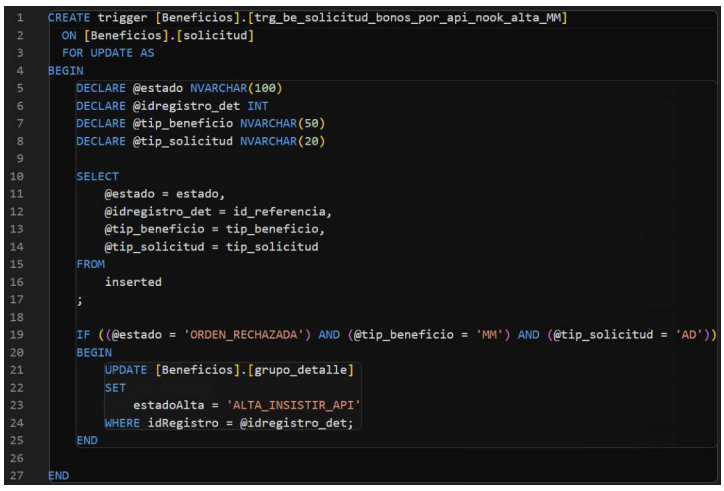


```
CREATE trigger [Beneficios].[trg_be_solicitud_beneficios_por_api_ok_alta_MM]
  ON [Beneficios].[solicitud]
  FOR UPDATE AS
BEGIN
    DECLARE @estado NVARCHAR(100)
    DECLARE @idregistro_det INT
    DECLARE @tip_beneficio NVARCHAR(50)
    DECLARE @tip_solicitud NVARCHAR(20)

    SELECT
        @estado = estado,
        @idregistro_det = id_referencia,
        @tip_beneficio = tip_beneficio,
        @tip_solicitud = tip_solicitud
    FROM
        inserted
    ;

    IF ((@estado = 'ORDEN_GENERADA') AND (@tip_beneficio = 'MM') AND (@tip_solicitud = 'AD'))
    BEGIN
        UPDATE [Beneficios].[grupo_detalle]
        SET
            estado = 'ACTIVO',
            estadoAlta = 'GESTIONADO_OK_API',
            fecha_alta_beneficio = GETDATE()
        WHERE idRegistro = @idregistro_det;
    END

END
```

Figura 25. Trigger alta ok

Fuente Elaboración propia, basada en la práctica.

2. Alta No OK: el trigger registrará la respuesta negativa del alta del beneficio para una cuenta determinada y establecerá el estado del alta en el valor correspondiente para que sea tomado por el proceso Insistir Alta.



```
CREATE trigger [Beneficios].[trg_be_solicitud_bonos_por_api_nook_alta_MM]
  ON [Beneficios].[solicitud]
  FOR UPDATE AS
BEGIN
    DECLARE @estado NVARCHAR(100)
    DECLARE @idregistro_det INT
    DECLARE @tip_beneficio NVARCHAR(50)
    DECLARE @tip_solicitud NVARCHAR(20)

    SELECT
        @estado = estado,
        @idregistro_det = id_referencia,
        @tip_beneficio = tip_beneficio,
        @tip_solicitud = tip_solicitud
    FROM
        inserted
    ;

    IF ((@estado = 'ORDEN_RECHAZADA') AND (@tip_beneficio = 'MM') AND (@tip_solicitud = 'AD'))
    BEGIN
        UPDATE [Beneficios].[grupo_detalle]
        SET
            estadoAlta = 'ALTA_INSISTIR_API'
        WHERE idRegistro = @idregistro_det;
    END

END
```

Figura 26. Trigger alta no ok

Fuente Elaboración propia, basada en la práctica.

3. Baja OK: el trigger registrará la respuesta positiva de la baja del beneficio para una cuenta determinada y establecerá el estado del beneficio correspondientemente.

4. Baja No OK: el trigger registrará la respuesta negativa de la baja del beneficio para una cuenta determinada y establecerá el estado de la baja en el valor correspondiente para que sea tomado por el proceso Insistir Baja.
5. Evaluador Baja: el beneficio está condicionado por la cantidad de integrantes del grupo, en el caso de que la cantidad de integrantes sea menor al límite inferior de dos, se deberá gestionar la baja del beneficio del último integrante y registrar la baja del grupo cuando se confirme la gestión.

```
ALTER TRIGGER [Beneficios].[trg_evaluador_limite_inferior_MM]
  ON [Beneficios].[grupo_detalle] FOR UPDATE AS
BEGIN
    --Declaraciones
    SELECT
        @estado = estado, @idRegistroGrupo = idRegistroGrupo, @codBeneficio = codBeneficio, @descBeneficio = descBeneficio
    FROM inserted ;
    IF (@estado = 'BAJA')
    BEGIN
        SELECT @cant_actual = COUNT(*) FROM [Beneficios].[grupo_detalle]
        WHERE estado != 'BAJA' AND idRegistroGrupo = @idRegistroGrupo
        IF (@cant_actual = 0)
        BEGIN
            UPDATE [Beneficios].[grupo_cabecera] SET estadoGrupo = 'BAJA' fecha_fin_grupo = GETDATE() WHERE IdRegistro = @idRegistroGrupo
        END
        ELSE BEGIN
            IF (@cant_actual = 1)
            BEGIN--solitario
                SELECT @IdRegistroDet = IdRegistro, @idCuenta = idCuenta FROM [Beneficios].[grupo_detalle] WHERE estado != 'BAJA'
                BEGIN TRAN
                    --La baja de la grupo se hace en el trigger OK de baja
                    UPDATE [Beneficios].[grupo_detalle]
                    SET estado_BAJA = 'SOLICITADO'
                    WHERE IdRegistro = @IdRegistroDet
                    INSERT [Beneficios].[be_solicitud_bonos_por_api]
                    (idCuenta, descBeneficio, bono_cd, tip_solicitud, fec_solicitud, estado, id_referencia, tip_beneficio, usuario)
                    VALUES
                    (@idCuenta, @descBeneficio, @codBeneficio, 'RM', GETDATE(), 'SOLICITADO', @IdRegistroDet, 'GB_MM', @usuario)
                COMMIT
            END
        END
    END
END
```

Figura 27. Trigger evaluador baja

Fuente Elaboración propia, basada en la práctica.

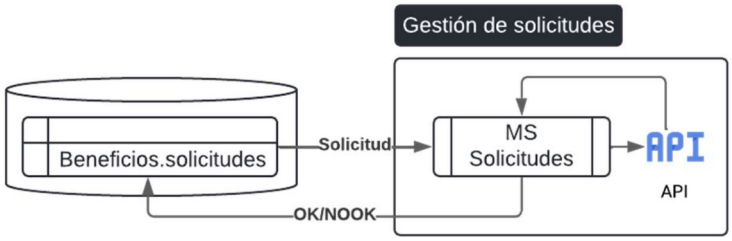


Figura 28. API Solicitud de Beneficios.

Fuente Elaboración propia, basada en la práctica.

e. Tarea 5 – Adquisición de conocimientos y diseño de microservicios.

La tarea 5 consiste, en primera instancia, en adquirir los conocimientos necesarios para realizar el Desarrollo de Backend. A medida que se avance con las siguientes tareas, se irá detallando el uso de las técnicas y conceptos explicados anteriormente. En segunda instancia, se deberá realizar el diseño macro que se describe en la historia 4:

H4: Como equipo de desarrollo quiero realizar el diseño macro de los microservicios a desarrollar para lograr comprender el alcance de las tareas.

A partir del relevamiento realizado con los representantes de los usuarios y del análisis de la documentación del sistema que se pretende reemplazar, surge la necesidad de poner a disposición dos endpoints. El primer endpoint tendrá como objetivo proporcionar información sobre las cuentas de cada cliente de un tipo y número de documento determinado. La información provista deberá permitir determinar si la persona ha constituido su grupo, qué cuentas tienen el beneficio, el estado de asignación del beneficio y si cumple o no con determinadas validaciones. El segundo endpoint permitirá la creación de un grupo, la solicitud de la modificación del grupo y gestionar altas y bajas de beneficio en determinadas cuentas.

En la Figura 29 se describe el diseño macro del objetivo de desarrollo para satisfacer la consulta de parque. En la consulta se permite obtener todas las cuentas de un tipo y número de documento con cierto detalle que se explicará más adelante. La base de datos CRM (*Customer Relationship Management*) contiene la información de los clientes y cuentas relacionadas a un tipo y número de documento. De esta base se obtendrán otros datos adicionales. El DAO-PARQUE tendrá como objetivo acceder a esa base de datos y proporcionar información subyacente. La base da datos de Beneficios, que se describió anteriormente, permite gestionar los beneficios asociados a las cuentas del cliente. El DAO-BENEFICIOS tendrá como objetivo realizar las operaciones de alta, baja y modificación de los beneficios registrados en dicha base de datos. Adicionalmente, se requiere realizar validaciones sobre las cuentas con información registrada en la base de datos CRM. El FCD-Beneficios (Facade) deberá procesar la consulta del usuario de la API para un tipo y número de documento solicitado, y tendrá la lógica necesaria para procesar la información obtenida de los *DAO's* y de las validaciones para permitir generar una respuesta que detalle la totalidad de cuentas y las clasifique para informar si la cuenta pertenece a las siguientes tres categorías: cuentas que ya poseen el beneficio, cuentas que no poseen el beneficio, pero

pueden acceder a este y cuentas que no poseen el beneficio y no pueden acceder a este porque no cumplen con alguna de las validaciones.

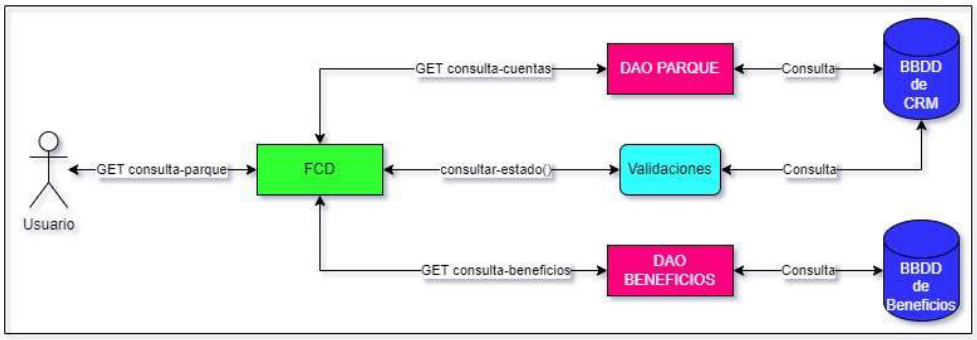


Figura 29. Diseño macro - Consulta parque

Fuente Elaboración propia, basada en la práctica.

En la Figura 30 se describe el diseño macro del objetivo de desarrollo para satisfacer la modificación de parque. En la modificación de parque se reciben las cuentas para las que se requiere dar de alta o dar de baja el beneficio. En este caso, la consulta al DAO-PARQUE y las consultas de validaciones son individuales y se aplican solo a las cuentas que se quieren modificar en la solicitud. En la consulta de parque, el alcance del DAO-PARQUE y las validaciones, es para todas las cuentas del tipo y número de documento. El DAO-BENEFICIOS se utiliza, en primera instancia, para consultar si la cuenta posee o no el beneficio, lo cual es necesario para determinar si se puede solicitar el alta o la baja. En segunda instancia, una vez que se realizaron las validaciones pertinentes, el DAO-BENEFICIOS registrará la solicitud de altas y bajas de beneficios mediante los SP de la base de datos de beneficios, desarrollados previamente. El lote de altas y bajas se ejecutará en una transacción, lo que implica que, si fallase la ejecución de una de las altas o bajas del lote, se revertirá el lote completo.

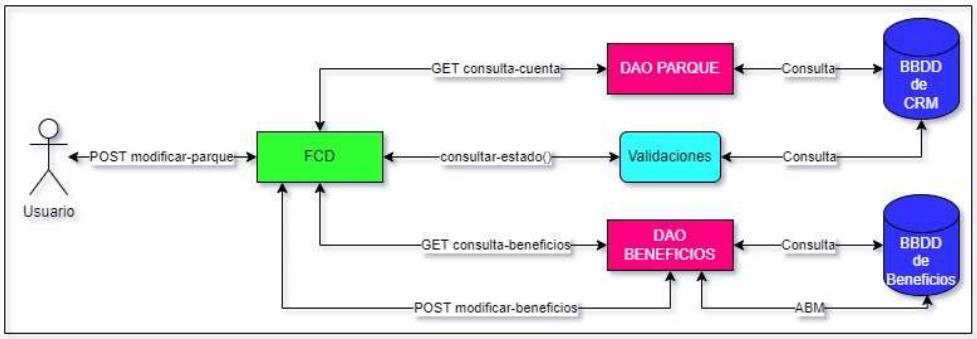


Figura 30. Diseño macro - Modificación parque

Fuente Elaboración propia, basada en la práctica.

f. Tarea 6 – Desarrollo de microservicios.

La tarea 6 consiste en el desarrollo de los microservicios descritos en los diseños macro de la consulta y modificación de parque. Para el desarrollo de los microservicios de la consulta de parque, se toma la Historia 5 del Backlog, la que debe ser refinada en nuevas historias que se detallarán a continuación.

H5: Como equipo de desarrollo quiero diseñar y desarrollar los microservicios en ambientes bajos para interactuar con el desarrollo de base de datos.
H5.1: Como equipo de desarrollo quiero diseñar y desarrollar el dao-parque para obtener información de cuentas de un tipo y número de documento de la base de datos CRM en el endpoint consulta-cuentas. H5.2: Como equipo de desarrollo quiero diseñar y desarrollar el dao-beneficios para obtener información de beneficios de un tipo y número de documento de la base de datos Beneficios en la consulta-beneficios. H5.3: Como equipo de desarrollo quiero diseñar y desarrollar el fcd-beneficios para unificar la información del dao-parque y del dao-beneficios de un tipo y número de documento en la consulta-parque, agregando lógica de validaciones y clasificando las cuentas en las categorías descritas anteriormente. H5.4: Tarea de análisis de validaciones para determinar si un cliente posee deuda. H5.5: Tarea de análisis de validaciones para determinar si una cuenta ya tiene el beneficio activo. H5.6: Tarea de análisis de validaciones para determinar si un cliente tiene órdenes pendientes en el CRM. H5.7: Como equipo de desarrollo quiero diseñar y desarrollar la evolución del fcd-beneficios para integrar las validaciones de las tareas H5.4, H5.6 y H5.7 en la consulta-parque. H5.8: Como equipo de desarrollo quiero diseñar y desarrollar la evolución del fcd-beneficios para agregar el endpoint de modificar-parque.

A continuación, se describirá detalladamente la primera historia de desarrollo de un microservicio del proyecto. Para el resto de las historias de la misma índole, se limitará a explicar la lógica y particularidades de estas.

En el siguiente Sprint se realizará el desarrollo de la siguiente historia:

H5.1: Como equipo de desarrollo quiero diseñar y desarrollar el dao-parque para obtener información de cuentas de un tipo y número de documento de la base de datos CRM en el endpoint parque-cliente.

En la ceremonia de refinamiento del Sprint anterior se realizó el refinamiento de la historia 5. En la ceremonia de planificación Sprint en curso, se seleccionó la historia 5.1, la cual fue analizada por el equipo para determinar si se cumplía la DOR (*Definition Of Ready*). DOR es un conjunto de criterios que una historia debe cumplir antes de ser considerada lista para trabajar por el equipo. Durante la planificación se realiza la estimación de esfuerzo utilizando la técnica de Fibonacci a través de una herramienta web llamada *Planning Poker* (PP). Fibonacci es una secuencia numérica donde cada número es la suma de los dos anteriores, comúnmente usada en estimaciones ágiles por su simplicidad y efectividad en representar el esfuerzo relativo. En PP, cada Dev registra en secreto su estimación personal basada en su experiencia, que luego es revelada y contrastada con las estimaciones del resto de los Devs. PP es una técnica de estimación que fomenta la discusión y consenso entre los miembros del equipo para obtener una estimación precisa del esfuerzo requerido. Ante diferencias entre las estimaciones del grupo, se discute hasta llegar a un acuerdo. Como parte del Sprint Goal definido, se detalla la “Migración del sistema de Beneficios”. Otras historias, bugs y tareas son incluidos en el Sprint.

Como se anticipó anteriormente, se describirá el detalle de la historia 5.1. En el DAO-PARQUE se deberá desarrollar un endpoint (parque-cliente) que tendrá dos parámetros de entrada (tipo y número de documento). El DAO deberá acceder a la base de datos CRM, que tiene información de las cuentas, y obtener determinados campos de todo el parque de cuentas de todos los clientes de ese tipo y número de documento.

Una vez que se ha iniciado el Sprint, se asignan todos los elementos del Sprint Backlog a los diferentes miembros del equipo. Luego, el equipo de desarrollo sin la presencia del resto de los integrantes del Scrum Team, se reúne para definir las subtareas y realizar una estimación en jornadas de estas. Para el desarrollo de APIs se suelen crear las siguientes subtareas: Diseño, Desarrollo, Swagger, Code Review, Refactor, DDT, Deploy, Pruebas y Documentación.

Descripción de las subtareas:

1) Diseño: es la primera tarea que se realiza en el contexto de una historia de desarrollo de microservicio. Este se plasma en un diagrama de secuencia UML, que es una

representación gráfica que muestra la interacción de los objetos en un sistema en orden temporal. Se utiliza la herramienta web draw.io, una aplicación en línea para la creación de diagramas de manera sencilla y colaborativa. Una vez realizado, se expone ante el resto de los desarrolladores para consensuar el mismo y recibir sugerencias de cambios.

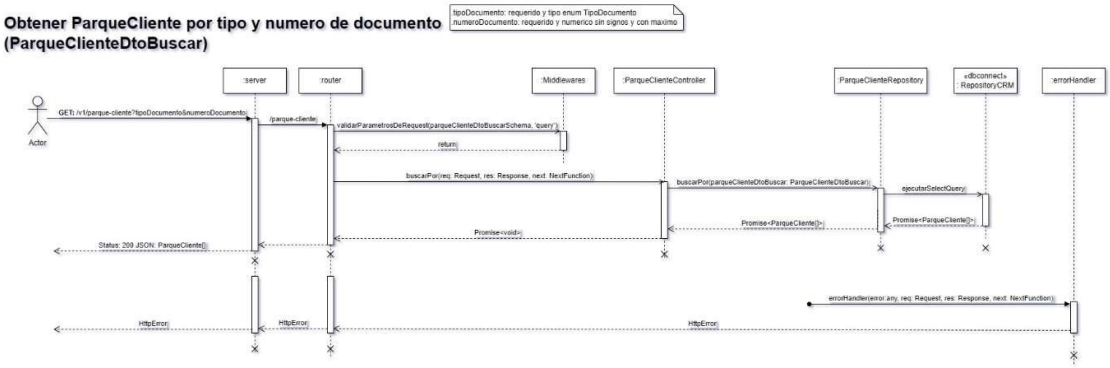


Figura 31. Diseño Dao Parque - Consulta Parque Cliente Secuencia

Fuente Elaboración propia, basada en la práctica.

En la Figura 31 se expone el Diseño del Diagrama de la consulta GET parque-cliente del DAO-PARQUE, que muestra la secuencia de acciones y entidades involucradas. En una aplicación Node.js del framework Express el server tiene como propósito iniciar y gestionar el servidor HTTP, este escuchará por las peticiones a los endpoints expuestos por el router. También, el server ejecuta funciones middleware para realizar tareas de autenticación, manejo de errores, gestión de sesiones, etc. En este caso, el router mediante una función middleware, validará los parámetros de entrada con Joi, luego de la validación, se llama a la función buscarPor de ParqueClienteController. El controlador ejecutará la función buscarPor de ParqueClienteRepository, a su vez, el repositorio ejecuta una función que se conecta a la base de datos y ejecuta un query mediante Sequelize. La respuesta obtenida de la base de datos, el parque de cuentas, es retornado al usuario de la API que realizó el requerimiento.

En la tarea de diseño, también se documentan los *namespaces* con sus funciones y las interfaces que se utilizarán en el desarrollo, como se demuestra en las Figuras 32 y 33. Un namespace se utiliza para agrupar lógicamente el código y evitar colisiones de nombres. Esta documentación permite establecer un acuerdo entre el equipo de Devs acerca de la denominación y tipos de datos de cada campo, antes de iniciar el desarrollo del microservicio.

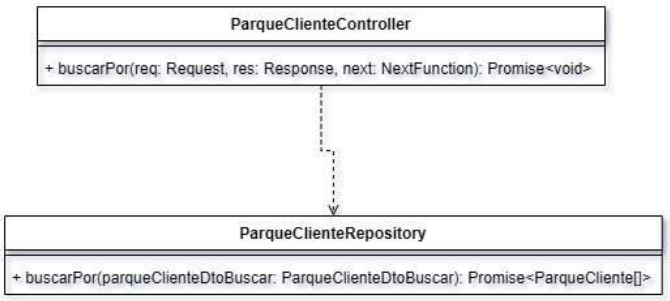


Figura 32. Diseño Dao Parque - Consulta Parque Cliente Namespace

Fuente Elaboración propia, basada en la práctica.

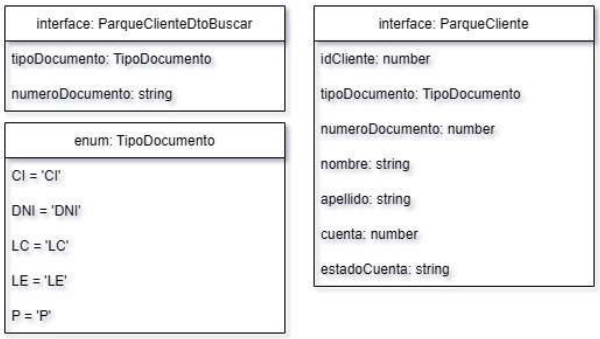


Figura 33. Diseño Dao Parque - Consulta Parque Cliente Interfaces

Fuente Elaboración propia, basada en la práctica.

2) Desarrollo:

a. Git: en esta etapa se va a crear una nueva API, para lo que se debe crear un nuevo proyecto en GitLab. Para la creación del proyecto, se selecciona la opción "*New Proyect*" de la siguiente Figura.

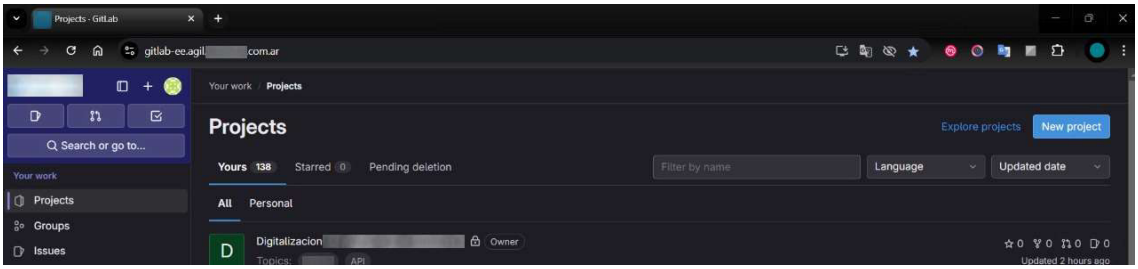


Figura 34. Gitlab Proyects

Fuente Elaboración propia, basada en la práctica.

Luego de seleccionar proyecto vacío, se especifica el nombre del proyecto “DAO-PARQUE” y se selecciona “*Create Proyect*”.

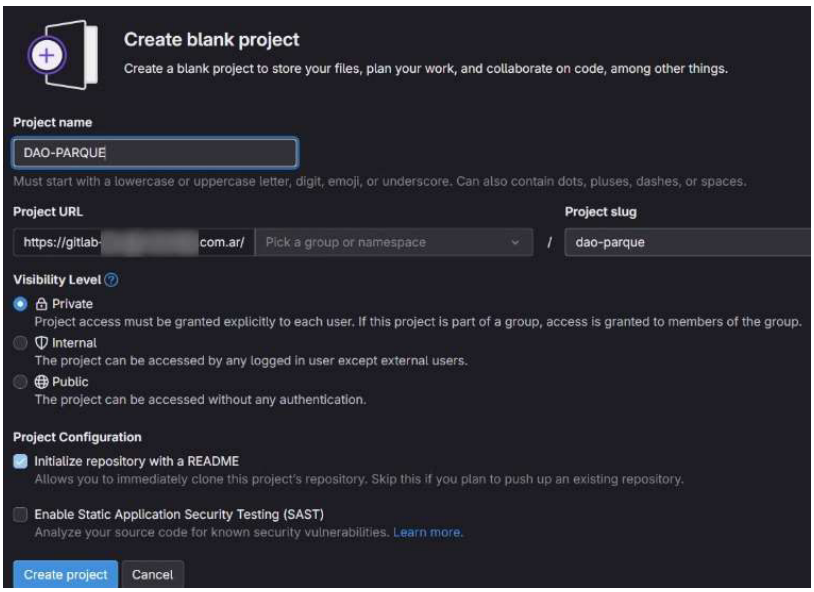


Figura 35. GitLab New Proyect

Fuente Elaboración propia, basada en la práctica.

b. Template: Si bien se parte de un proyecto en blanco en GitLab, a fines prácticos, se toma de referencia un template que contiene la arquitectura de la aplicación y configuración básica para el desarrollo de un DAO. Se crea una rama de desarrollo con el identificador de la Historia de Jira, en la que se irán confirmando los cambios realizados en esta etapa.

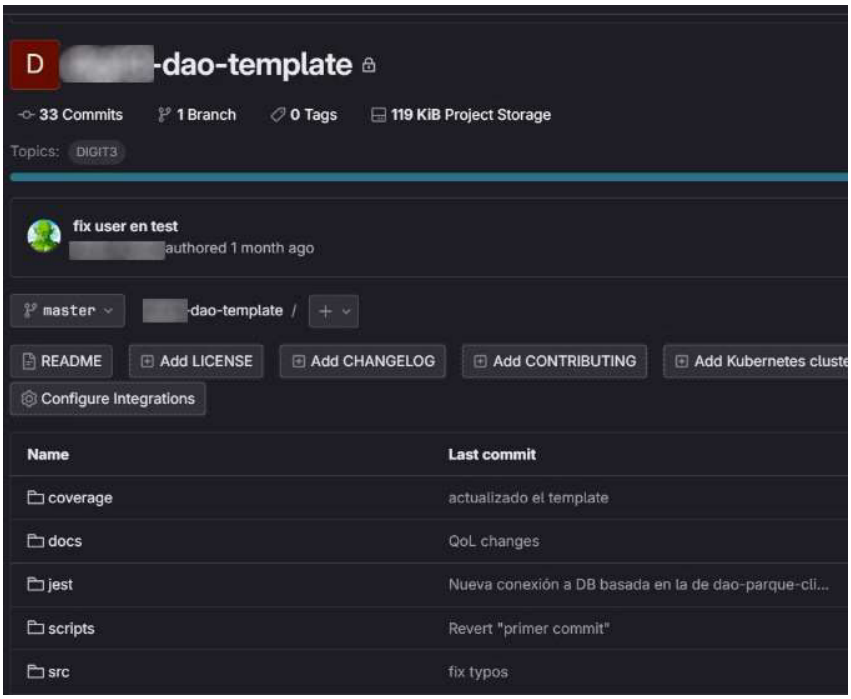


Figura 36. Dao-Template

Fuente Elaboración propia, basada en la práctica.

c. La estructura general de proyecto template es la siguiente:

- coverage: contendrá los informes de cobertura de las pruebas automatizadas de Jest.
- docs: se guarda el diseño explicado anteriormente.
- jest: almacena archivos de configuración para las pruebas.
- src: contiene las siguientes subcarpetas:
    - configuration: contiene scripts y archivos de configuración de Sequelize.
    - middlewares: contiene una serie de middlewares predefinidos.
    - router: contiene el archivo routers.ts.
    - schemas: contendrá los archivos de Joi.
    - server: contiene el archivo de configuración de Express.
    - types: contendrá la definición de las interfaces, enums, etc.
    - utils: contendrá archivos de script y configuración en común.
    - app.ts: inicia el servidor Express.
    - swagger: contendrá el archivo de definición de swagger y scripts para la generación automática del swagger.

- test: la carpeta tendrá una subcarpeta por cada carpeta de src para los test de Jest.
- .env: contiene configuración de variables de entorno necesarias para la configuración de la aplicación Node.js. Suele contener pares clave-valor.
- .package.json: es un archivo de configuración de Node.js donde se especifican las dependencias, los scripts, configuraciones, etc.

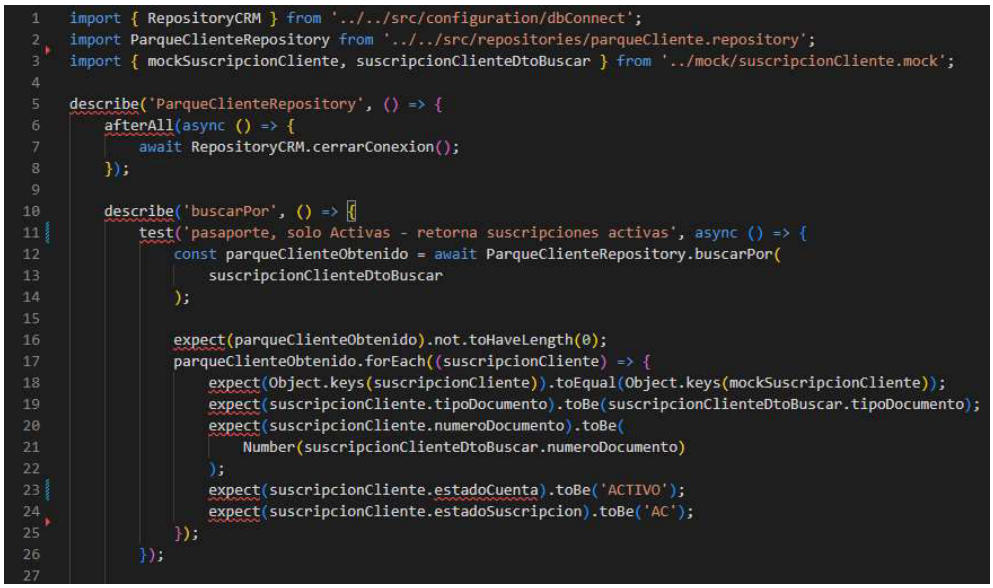
d. TDD - repository: en la ejecución de esta sección del cronograma, siguiendo los principios de TDD, se explicará cómo se realizó la creación y validación de un conjunto de pruebas que guiará el desarrollo del código de manera incremental. Esto se hará, utilizando las capacidades de Jest, asegurándose de que se respetan los más altos estándares de calidad y eficiencia.

En primer lugar, se explica el archivo en la ruta test/repositories llamado parqueCliente.repository.spec.ts, o sea el archivo del test del repositorio. El sufijo *.spect.ts* sirve para distinguir la implementación de las pruebas, permite a Jest configurar los *runners* de pruebas. Es decir, el test parqueCliente.repository.spec.ts corresponde a la implementación en parqueCliente.repository.ts. El repositorio se conectará a la base de datos CRM mediante una conexión Sequelize, enviando parámetros de entrada y recibiendo una salida. De cierta forma, el objetivo es *mockear* el parámetro de entrada del repositorio y poder definir una serie de salidas esperadas del repositorio. De esta manera, podemos predecir cuál será la salida y condicionar la ejecución del test. Las funciones comúnmente utilizadas en jest son las siguientes:

- Describe: se utiliza para agrupar pruebas relacionadas de una misma función, clase o módulo.
- Test: se utiliza para definir un caso de prueba individual, su nombre describe qué se está probando y qué resultado se espera.
- Expect: se utiliza para hacer afirmaciones sobre el valor de una variable o resultado de una función.
    - toHaveLength: verifica la longitud de arrays o strings.
    - toEqual: compara valores, ideal para objetos y arrays.
    - toBe: compara valores de manera estricta (primitivos y referencias).

En la Figura 37, las sentencias *import* de TypeScript importan módulos, funciones, objetos o clases desde otros archivos o módulos. Al importar RepositoryCRM se obtiene las funciones y configuración del acceso a la base de datos, importando ParqueClienteRepository

se importa la definición del repositorio y al importar desde la ruta mock, se obtienen los mocks necesarios para la batería de test a ejecutar. Mediante el mock suscripcionClienteDtoBuscar se especifica el cliente que se buscará en la base de datos, simulando la entrada desde el controller. En la línea 13 se ejecuta la función del repository con la palabra clave *await*, la cual se guarda en la constante parqueClienteObtenido, *await* se utiliza para esperar la resolución de una promesa, la ejecución se pausa hasta que la promesa devuelva un resultado o sea rechazada. A partir de la línea 17 se especifican los valores esperados para la respuesta. De esta forma, siguiendo los principios TDD, antes de desarrollar el repository se definen en el test las distintas pruebas que deberá cumplir para poder superar la prueba unitaria.



```
1  import { RepositoryCRM } from '../../src/configuration/dbConnect';
2  import ParqueClienteRepository from '../../src/repositories/parqueCliente.repository';
3  import { mockSuscripcionCliente, suscripcionClienteDtoBuscar } from '../mock/suscripcionCliente.mock';
4
5  describe('ParqueClienteRepository', () => {
6      afterAll(async () => {
7          await RepositoryCRM.cerrarConexion();
8      });
9
10     describe('buscarPor', () => {
11         test('pasaporte, solo Activas - retorna suscripciones activas', async () => {
12             const parqueClienteObtenido = await ParqueClienteRepository.buscarPor(
13                 suscripcionClienteDtoBuscar
14             );
15
16             expect(parqueClienteObtenido).not.toHaveLength(0);
17             parqueClienteObtenido.forEach((suscripcionCliente) => {
18                 expect(Object.keys(suscripcionCliente)).toEqual(Object.keys(mockSuscripcionCliente));
19                 expect(suscripcionCliente.tipoDocumento).toBe(suscripcionClienteDtoBuscar.tipoDocumento);
20                 expect(suscripcionCliente.numeroDocumento).toBe(
21                     Number(suscripcionClienteDtoBuscar.numeroDocumento)
22                 );
23                 expect(suscripcionCliente.estadoCuenta).toBe('ACTIVO');
24                 expect(suscripcionCliente.estadoSuscripcion).toBe('AC');
25             });
26         });
27
```

Figura 37. Test Repository

Fuente Elaboración propia, basada en la práctica.

En segundo lugar, se explica el archivo parqueCliente.repository.ts de la ruta src/repositories que contiene la implementación del repository. En la Figura 38, se repite el import de RepositoryCRM para la conexión a la base de datos; un segundo import a las interfaces definidas en el archivo src/types/suscripcionCliente.interface.ts (Figura 39) donde se definen las interfaces SuscripcionClienteDtoBuscar y SuscripcionCliente. SuscripcionClienteDtoBuscar contiene la definición de los parámetros de búsqueda del repositorio y SuscripcionCliente define la salida de este. En el último import, se realiza para obtener el query que se ejecutará en base de datos y que se resguarda en el archivo /utils/queries/queries.ts.

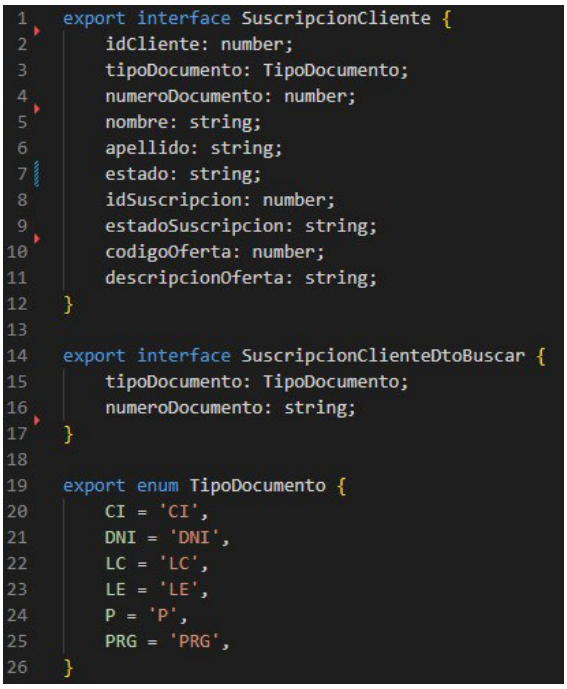
En la definición de la función asincrónica buscarPor del namespace ParqueClienteRepository, se especifica que recibe como parámetro una constante del tipo

SuscripcionClienteDtoBuscar y retornará una promesa de tipo *array* de SuscripcionCliente. Una función asíncrona (*async*) permite escribir código asíncrono de forma legible y estructurada, usada en combinación con *await*. Una promesa (*promise*) es un objeto que representa la eventual finalización de una operación asíncrona y su resultado. La función retorna la constante parqueCliente que es el resultado de la ejecución asíncrona de la función ejecutarSelectQuery que recibe como parámetro un objeto que contiene el *query, los replacements y fieldMap*. El query obtenerParque es la sentencia SQL que se va a ejecutar en la base de datos CRM para obtener los datos detallados anteriormente. Los *replacements* permiten parametrizar la consulta SQL mientras que evita ataques de inyecciones de código. Los *fieldMaps* transforma los resultados de la consulta en un objeto JavaScript.

```
import { RepositoryCRM } from '../configuration/dbConnect';
import {
    SuscripcionCliente,
    SuscripcionClienteDtoBuscar,
} from '../types/suscripcionCliente.interface';
import { queries } from '../utils/queries/queries';

namespace ParqueClienteRepository {
    export async function buscarPor(
        suscripcionClienteDtoBuscar: SuscripcionClienteDtoBuscar
    ): Promise<SuscripcionCliente[]> {
        const parqueCliente: SuscripcionCliente[] = await RepositoryCRM.ejecutarSelectQuery({
            query: queries.obtenerParque,
            replacements: {
                TIPODOCUMENTO: suscripcionClienteDtoBuscar.tipoDocumento,
                NUMERODOCUMENTO: String(suscripcionClienteDtoBuscar.numeroDocumento)
            },
            fieldMap: {
                ID_CLIENTE: 'idCliente',
                TIPO_DOCUMENTO: 'tipoDocumento',
                NUMERO_DOCUMENTO: 'numeroDocumento',
                NOMBRE: 'nombre',
                APELLIDO: 'apellido',
                ESTADO: 'estado',
                ID_SUSCRIPCION: 'idSuscripcion',
                ESTADO_SUSCRIPCION: 'estadoSuscripcion',
                CODIGO_OFERTA: 'codigoOferta',
                DESCRIPCION_OFERTA: 'descripcionOferta',
            },
        });
        return parqueCliente;
    }
```

Figura 38. Repository

Fuente Elaboración propia, basada en la práctica.



```
export interface SuscripcionCliente {
    idCliente: number;
    tipoDocumento: TipoDocumento;
    numeroDocumento: number;
    nombre: string;
    apellido: string;
    estado: string;
    idSuscripcion: number;
    estadoSuscripcion: string;
    codigoOferta: number;
    descripcionOferta: string;
}

export interface SuscripcionClienteDtoBuscar {
    tipoDocumento: TipoDocumento;
    numeroDocumento: string;
}

export enum TipoDocumento {
    CI = 'CI',
    DNI = 'DNI',
    LC = 'LC',
    LE = 'LE',
    P = 'P',
    PRG = 'PRG',
}
```

Figura 39. Interfaces

Fuente Elaboración propia, basada en la práctica.

e. TDD – controller: en primer lugar, continuando con la metodología TDD, en la Figura 40 correspondiente a parqueCliente.controller.test.ts se define parte de los test desarrollados para la implementación parqueCliente.controller.ts. En el contexto de pruebas unitarias del controlador, mediante la función jest.mock(), se reemplaza todo el módulo parqueCliente.repository por una versión simulada. Luego, mediante "*as jest.Mock*" se establece que la función buscarPor debe ser tratada como un mock de Jest, lo que permite acceder a los métodos específicos como, por ejemplo, *mockResolvedValue* para asignar el mock mockSuscripcionCliente al valor de respuesta de la función mockeada buscarPor. A partir de la línea 17, se definen las siguientes constantes:

- req: de tipo Request de Express, es una interfaz que contiene información sobre la solicitud HTTP, mediante la función getMockReq se genera un objeto mockeado especificando los query params a partir del mock suscripcionClienteDtoBuscar.

- res: de tipo Response de Express que es una interfaz que representa el objeto de respuesta al cliente, mediante la función getMockRes se genera un objeto simulado de respuesta.

- next: representa la función next() que usan los middlewares de Express para pasar el control al siguiente middleware en la cadena, mediante la función getMockRes.

A continuación, es llamada la *función asíncrona buscar* (del *controller)*. Los parámetros recibidos por esta función son *req, res y next*. Por último, se evalúan los resultados de la ejecución con las funciones *expect* de Jest, particularmente que la función haya sido llamada con el mock con parámetros de entrada, que el status de la respuesta haya sido llamado con un status 200 de HTTP y que el contenido de la respuesta haya sido llamado con el mock de respuesta asociado. Otro test realizado sobre el controller consiste en que la función buscarPor mockeada se le asigne un error y validar que next sea llamado con un Error.

```
import ParqueClienteController from '../../src/controllers/parqueCliente.controller';
import ParqueClienteRepository from '../../src/repositories/parqueCliente.repository';
import { NextFunction, Request, Response } from 'express';
import { getMockReq, getMockRes } from '@jest-mock/express';
import { StatusCodes } from 'http-status-codes';
import { mockSuscripcionCliente, suscripcionClienteDtoBuscar } from '../mock/suscripcionCliente.mock';

jest.mock('../../src/repositories/parqueCliente.repository', () => ({
    buscarPor: jest.fn(),
}));

describe('ParqueClienteController', () => {
    describe('buscarPor', () => {
        const mockBuscarPorRepository = ParqueClienteRepository.buscarPor as jest.Mock;

        test('con repo retornando Parque - responde status 200 y json con ParqueCliente', async () => {
            const req: Request = getMockReq({
                query: {
                    tipoDocumento: suscripcionClienteDtoBuscar.tipoDocumento,
                    numeroDocumento: String(suscripcionClienteDtoBuscar.numeroDocumento),
                },
            });
            const res: Response = getMockRes().res;
            const next: NextFunction = getMockRes().next;
            mockBuscarPorRepository.mockResolvedValue([mockSuscripcionCliente]);

            await ParqueClienteController.buscarPor(req, res, next);

            expect(mockBuscarPorRepository).toBeCalledWith(suscripcionClienteDtoBuscar);
            expect(res.status).toBeCalledWith(StatusCodes.OK);
            expect(res.json).toBeCalledWith([mockSuscripcionCliente]);
        });
```

Figura 40. Test Controller

Fuente Elaboración propia, basada en la práctica.

En segundo lugar, se explica la implementación de controller en parqueCliente.controller.ts. de la Figura 41. Se especifica la función asincrónica buscarPor que define un bloque *try-catch* para prevenir fallos inesperados y manejar los bloques errores

de manera controlada. En el bloque *try* se toman los parámetros del *Request* tipoDocumento y numeroDocumento para conformar el suscripcionClienteDtoBuscar que se utiliza para convocar a la función buscarPor del *Repository*. Luego, el *Response* se le asigna el código HTTP 200 y el resultado de la llamada al repositorio. En caso de error en el bloque *try*, se ejecutará el bloque *catch* invocando a la función *Next* enviando el error que causó la excepción.

```
import { NextFunction, Request, Response } from 'express';
import {
    SuscripcionCliente,
    SuscripcionClienteDtoBuscar,
    TipoDocumento,
} from '../types/suscripcionCliente.interface';
import ParqueClienteRepository from '../repositories/parqueCliente.repository';
import { StatusCodes } from 'http-status-codes';

namespace ParqueClienteController {
    export async function buscarPor(req: Request, res: Response, next: NextFunction): Promise<void> {
        try {
            const suscripcionClienteDtoBuscar: SuscripcionClienteDtoBuscar = {
                tipoDocumento: req.query.tipoDocumento as TipoDocumento,
                numeroDocumento: String(req.query.numeroDocumento),
            };
            const parqueCliente: SuscripcionCliente[] = await ParqueClienteRepository.buscarPor(
                suscripcionClienteDtoBuscar
            );
            res.status(StatusCodes.OK).json(parqueCliente);
        } catch (error: any) {
            next(error);
        }
    }
```

Figura 41. Controller

Fuente Elaboración propia, basada en la práctica.

f. TDD – schema: Antes de avanzar con la explicación del schema, se hace un breve comentario sobre la función middleware *validarParámetrosDeRequest*, expuesta en la Figura 42, perteneciente al *template* y que utiliza Joi para la validación de los parámetros de la petición en el *Request*. Básicamente, la función toma los datos del *Request* que pueden ser de tipo *body, query o params* y los valida mediante el schema que se desarrollará a continuación. *Body* refiere a los parámetros enviados en el cuerpo de la solicitud, los de tipo *query* forman parte de la consulta y son opcionales, y los *params* forma parte de la ruta y no son opcionales. Al especificar en false el valor de *abortEarly* se define que la comprobación deberá contemplar todas las validaciones del schema antes de finalizar. Siguiendo los principios TDD, primero se desarrolla el test del schema, en la Figura 43, del archivo suscripcionClienteDtoBuscar.schema.test.ts, el lineamiento a seguir es generar un test, a partir de los mocks, para cada casuística que se quiere testear. Empezando por aquella que provee parámetros válidos y no retorna

error, especificando el *Expect* correspondiente para cuando el Error no está definido y continuando por las que se espera que retornen un error: parámetros faltantes, parámetros inválidos, tanto para el número de documento como para el tipo de documento. En los casos de error, se especifica el mensaje de error esperado y el Expect correspondiente.

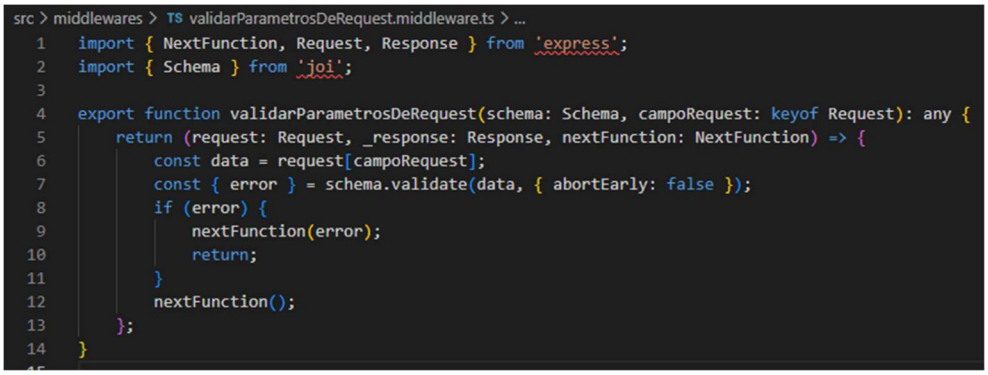


Figura 42. ValidarParámetros

Fuente Elaboración propia, basada en la práctica.

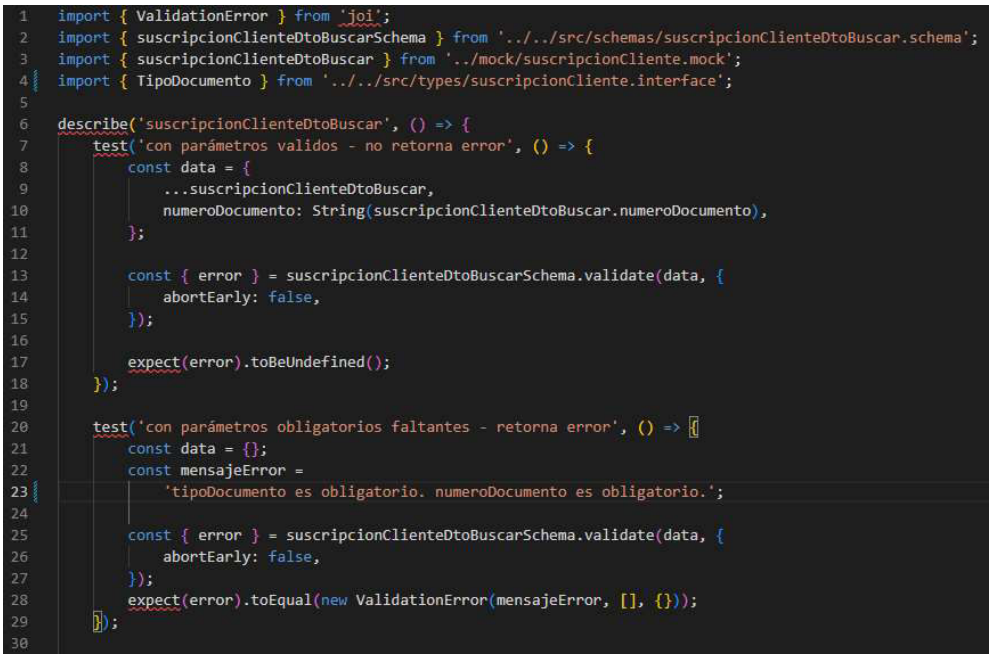


Figura 43. Test Schema

Fuente Elaboración propia, basada en la práctica

Luego, se desarrolla el Schema Joi, en la Figura 44, del archivo suscripcionClienteDtoBuscar.schema.ts. En este caso, el schema especifica tres expresiones regulares (regex), que deberá cumplir el campo número de documento, dependiendo del

campo tipo de documento. Una regex es un patrón de cadenas de texto para validar un formato especifico. En este schema, se establece que tanto el “tipo de documento” como el “número de documento” son campos obligatorios. Además, se requiere que el “tipo de documento” coincida con uno de los valores declarados en el enum definidos en el archivo de interfaces.

```
import Joi from 'joi';
import { TipoDocumento } from '../types/suscripcionCliente.interface';
import { crearObjectSchema } from '../utils/funciones';

const regex_Dni_Ci = /^\d{7,9}$/;
const regex_P_Le_Lc = /^(?:[A-Z]{2}\d{6}|\d{6,12})$/;
const regex_Prg = /^\d{3,24}$/;

const suscripcionSchema = {
    tipoDocumento: Joi.string()
        .valid(...Object.values(TipoDocumento))
        .required(),
    numeroDocumento: Joi.alternatives().conditional('tipoDocumento', {
        switch: [
            {
                is: Joi.valid(TipoDocumento.DNI, TipoDocumento.CI),
                then: Joi.string().pattern(regex_Dni_Ci).required(),
            },
            {
                is: Joi.valid(TipoDocumento.P, TipoDocumento.LE, TipoDocumento.LC),
                then: Joi.string().pattern(regex_P_Le_Lc).required(),
            },
            {
                is: TipoDocumento.PRG,
                then: Joi.string().pattern(regex_Prg).required(),
            },
        ],
        otherwise: Joi.string().required(),
    }),
};
export const suscripcionClienteDtoBuscarSchema = crearObjectSchema(suscripcionSchema);
```

Figura 44. Schema

Fuente Elaboración propia, basada en la práctica

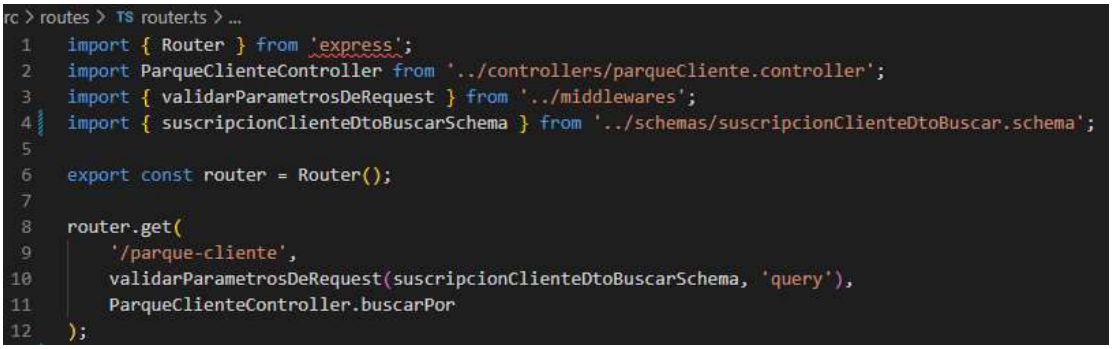
g. TDD – router: Continuando con los principios TDD, el archivo router.test.ts de la Figura 45. Test-Router, contiene la definición del test para el router. El Router en Express permite organizar las rutas de la aplicación, en este caso, la ruta desarrollada es parque-cliente. En el test se mockean la función buscarPor del controllador y la función validarParametrosDelRequest del middleware. Mediante la función request de la librería Supertest es posible enviar solicitudes HTTP a APIs, con el objetivo de poder probar la funcionalidad de las rutas de ellas. Se valida que el *middleware* haya sido llamado con el *schema*, que se haya llamado la función del controlador y que la response sea la esperada, según el mock definido.

```
import { NextFunction, Request, Response } from 'express';
import request from 'supertest';
import { server } from '../../src/server/index';
import ParqueClienteController from '../../src/controllers/parqueCliente.controller';
import { validarParametrosDeRequest } from '../../src/middlewares';
import { suscripcionClienteDtoBuscarSchema } from '../../src/schemas/suscripcionClienteDtoBuscar.schema';

jest.mock('../../src/controllers/parqueCliente.controller', () => ({
    buscarPor: jest.fn((_req: Request, res: Response) => {
        res.send('Controller mockeado.');
    }),
}));

jest.mock('../../src/middlewares/validarParametrosDeRequest.middleware.ts', () => ({
    validarParametrosDeRequest: jest
        .fn()
        .mockReturnValue((_req: Request, _res: Response, next: NextFunction) => {
            next();
        }),
}));

describe('router', () => {
    test('GET - /v1/parque-cliente - llama a ParqueClienteController.buscarPor', async () => {
        const response = await request(server).get('/v1/parque-cliente');

        expect(validarParametrosDeRequest).toBeCalledWith(suscripcionClienteDtoBuscarSchema, 'query');
        expect(ParqueClienteController.buscarPor).toBeCalled();
        expect(response.text).toBe('Controller mockeado.');
    });
});
```

Figura 45. Test-Router

Fuente Elaboración propia, basada en la práctica

El archivo router.ts de la Figura 46. Router crea y exporta la instancia de Router que será usada en el server, configura la ruta parque-cliente que responderá a solicitudes GET intercalando el middleware de validación de Joi e invocando a la función buscarPor del controller.



```
rc > routes > TS router.ts > ...
import { Router } from 'express';
import ParqueClienteController from '../controllers/parqueCliente.controller';
import { validarParametrosDeRequest } from '../middlewares';
import { suscripcionClienteDtoBuscarSchema } from '../schemas/suscripcionClienteDtoBuscar.schema';

export const router = Router();

router.get(
    '/parque-cliente',
    validarParametrosDeRequest(suscripcionClienteDtoBuscarSchema, 'query'),
    ParqueClienteController.buscarPor
);
```

Figura 46. Router

Fuente Elaboración propia, basada en la práctica

h. Server y App: El código del server/index.ts configura una instancia de Express con soporte para JSON, lo que permite convertir las solicitudes entrantes en formato JSON. También incluye CORS para que el servidor pueda responder a solicitudes desde otros orígenes, y se integra la documentación de la API,

utilizando Swagger. Además, se importa el router definido anteriormente para exponer las rutas bajo el path '/v1'.

Figura 47. Server

Fuente Elaboración propia, basada en la práctica

En cuanto a app.ts, se ejecuta inicialmente *require('dotenv'),config()* para cargar las variables de entorno desde el archivo .env al objeto process.env. Además, se importa un módulo de logger y el objeto de configuración *environment*, y el router previamente explicado. El método *listen* de server inicia el servidor que escucha peticiones en el puerto especificado en *environment.PORT*. El método *on* se utiliza para escuchar cualquier error que ocurra una vez iniciado el servidor. Ambos métodos ejecutan su respectivo *callback*, que son funciones que se pasan como argumentos, para ser invocadas cuando se complete una acción específica. En este caso, los *callbacks* se encargan de registrar el evento correspondiente.

Figura 48. App

Fuente Elaboración propia, basada en la práctica

Al ejecutar el comando *ts-node-dev src/app.ts* se inicia el servidor localmente y se pueden enviar solicitudes desde un navegador web o desde *swagger* como se verá más adelante.

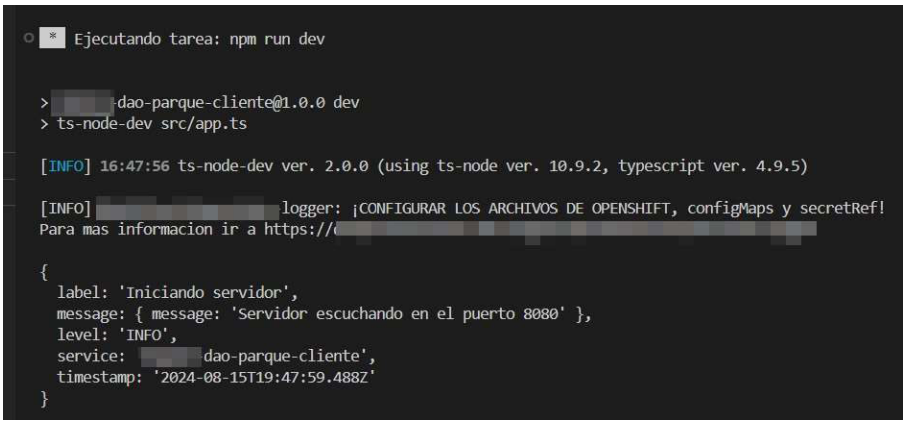


Figura 49. Run App

Fuente Elaboración propia, basada en la práctica

Antes de avanzar con la siguiente tarea dentro de la historia se deben correr todos los test desarrollados y validar la cobertura de código. Para esto, es necesario ejecutar el comando *jest –coverage* que genera un reporte de la cobertura acerca de las líneas, funciones, ramas y sentencias. Se muestra un porcentaje de la cobertura en todo el proyecto, ayudando a identificar partes del código no testeadas. Una vez que se hayan superado todos los test y el porcentaje de cobertura alcanzado sea del 100%, se realiza un *commit* sobre la rama de desarrollo de Git sobre la cual se estuvo trabajando hasta esta instancia.

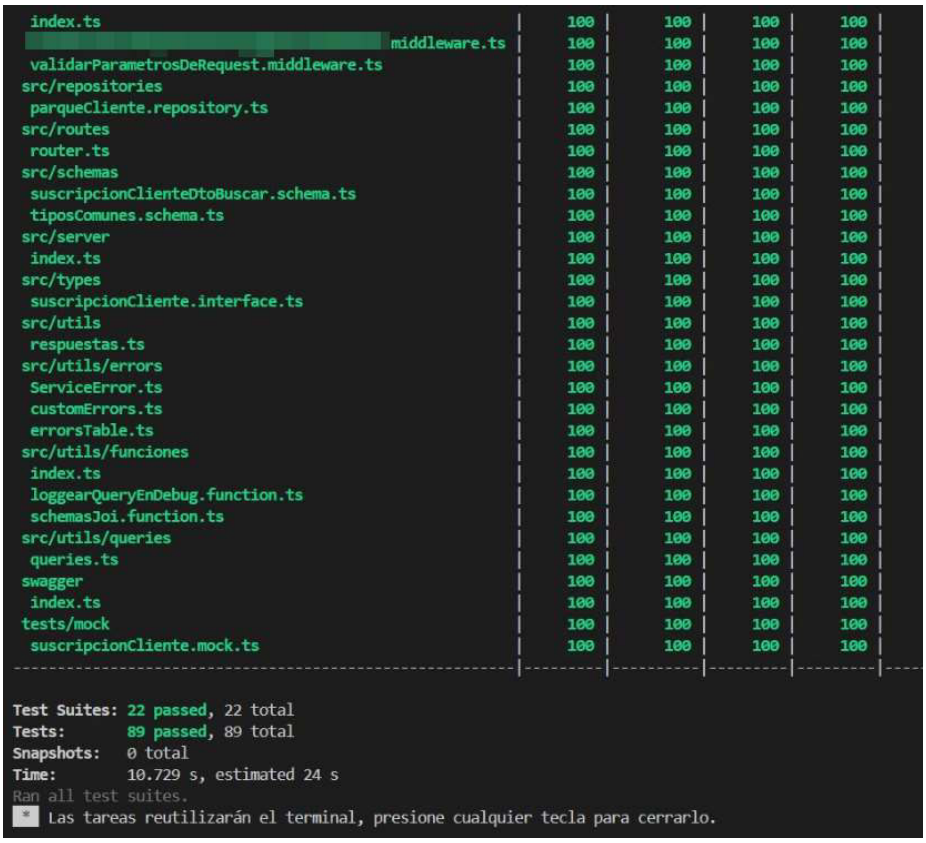


Figura 50. Coverage

Fuente Elaboración propia, basada en la práctica

3) Swagger: el objetivo de esta tarea es documentar la API generando el archivo "specification-swagger.json". Para lograrlo, se deben definir y especificar diferentes campos en este archivo: la versión de especificación OpenAPI swagger, el título, la versión, la descripción, los servidores disponibles, los paths y los componentes. Cada path define las rutas expuestas de la API, los métodos asociados para esa ruta, los parámetros de esa ruta (*query o params*), *requestBody* con el formato de la solicitud en un POST, las posibles respuestas que puede devolver la API (200, 400). Los *components* definen componentes reutilizables como esquemas, parámetros y respuestas. Cada campo dentro del JSON tiene un formato específico que debe ser respetado. La herramienta web https://editor.swagger.io/ permite validar el formato y realizar pruebas de visualización, de forma online se podrá ir modificando la definición del archivo json y el editor mostrará los cambios instantáneamente.

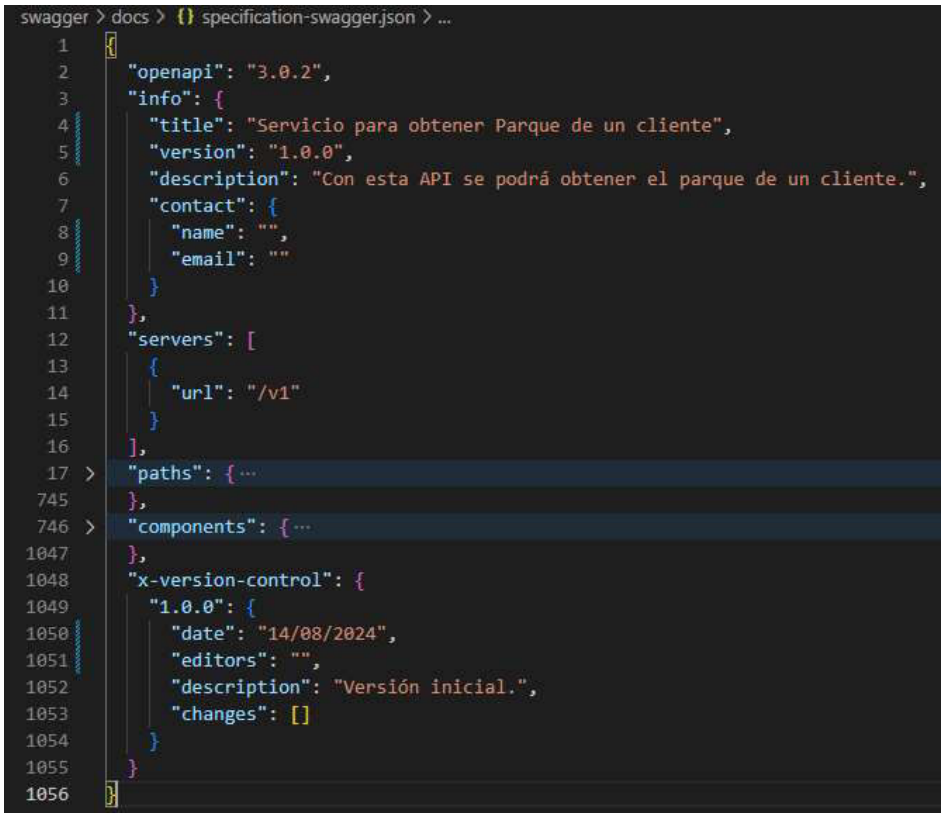


Figura 51. Swagger formato

Fuente Elaboración propia, basada en la práctica

Una vez ejecutada la API, swagger permite probarla. En este caso, el GET que obtiene el parque del cliente para un tipo y número de documento. En la ruta *swagger* del servicio se visualizan las rutas como por ejemplo GET /parque-cliente, se documenta que recibe los parámetros obligatorios y cierta descripción del servicio y de la ruta.

Figura 52. Swagger dao

Fuente Elaboración propia, basada en la práctica

También, se pueden documentar los parámetros informando el tipo de dato y los valores esperados.

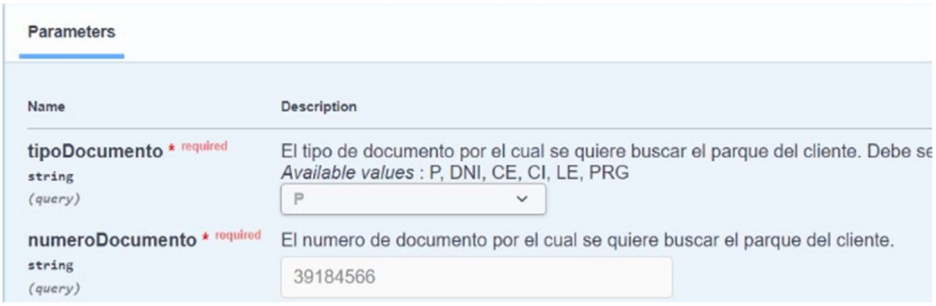


Figura 53. Swagger Parámetros

Fuente Elaboración propia, basada en la práctica

Se documentan las posibles respuestas de la API con el código de la respuesta y ejemplos de cada una de ellas.

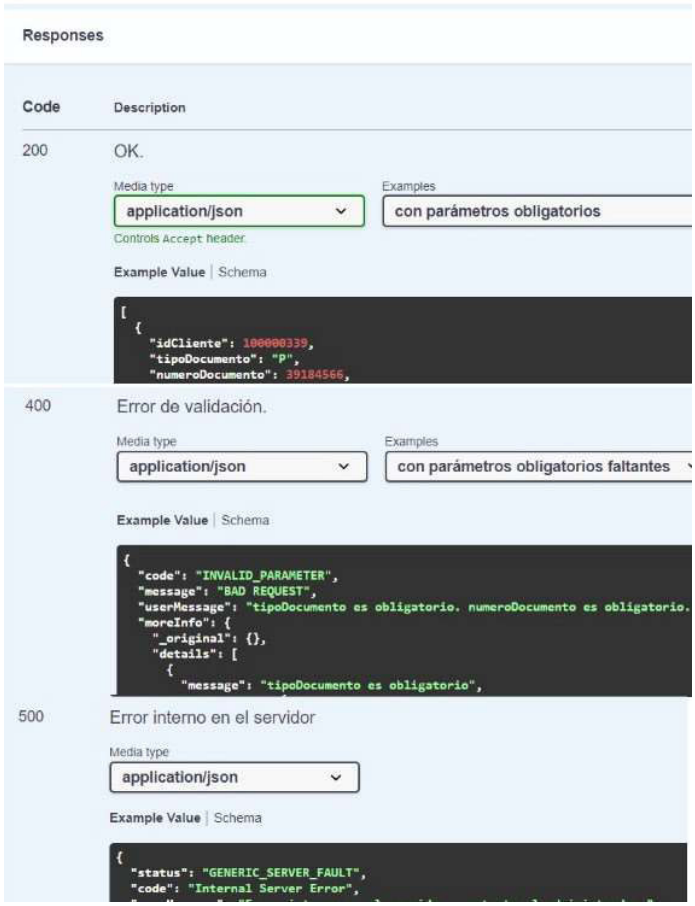


Figura 54. Swagger Responses

Fuente Elaboración propia, basada en la práctica

Por último, es posible probar la ruta con parámetros por defecto o especificando otros distintos, el resultado es la ruta completa del requerimiento y la respuesta en el body y los headers.

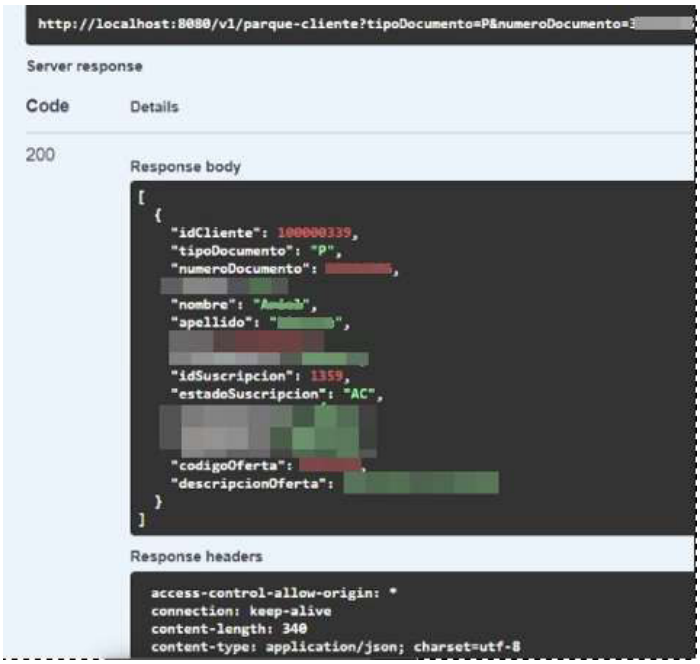


Figura 55. Swagger Try it out

Fuente Elaboración propia, basada en la práctica

4) Code Review y Refactor: como se ha comentado anteriormente, en la etapa de Desarrollo se realizó un commit en Git de la rama de desarrollo del repositorio del DAO. La revisión de código comienza con un *merge-request* desde esta rama a la rama de integración, un *merge-request* es una solicitud para fusionar los cambios de una rama a la rama principal. El objetivo es que uno o varios integrantes del equipo de desarrollo tomen el rol de revisores del código desarrollado. En esta etapa se evalúan aspectos como, por ejemplo: funcionalidad, legibilidad, mantenibilidad, eficiencia, seguridad.

   Si en la etapa de revisión surgen cambios por problemas detectados o mejoras se documentan en comentarios dentro del *merge-request*. El desarrollador responsable debe resolver lo detectado y actualizar la rama de desarrollo en el refactor, que puede incluir simplificación de funciones, eliminación de redundancias y mejoras de modularidad. Luego, se repiten las pruebas para confirmar que el código nuevo no introdujo errores de código.

5) *Deploy* en integración: para realizar el despliegue en el entorno de integración se debe seguir los siguientes pasos:

   a. Configuración: en este caso, la API es nueva y, por lo tanto, se debe realizar la configuración de la aplicación en GitLab. Este paso consiste en configurar el *Webhook* especificando una URL determinada para que pueda escuchar solicitudes de merge permitiendo automatizar la integración continua (CI) de la rama de integración a la rama master. Esto se realiza mediante la validación y ejecución de pruebas automatizadas gestionadas por el *pipeline*. El *merge request* es configurado como "*fast foward merge*", lo que permite mover directamente el puntero de la rama integración a la rama master manteniendo así el historial lineal de commits. Se debe configurar el usuario *automatpipeline* con el rol de *Mainteiner*, lo que le otorga permisos para configurar *pipelines*, administrar el repositorio y gestionar las tareas automatizadas. Un *pipeline* es una serie de procesos automatizados que construyen, prueban y despliegan código. Estas configuraciones son necesarias para habilitar y mantener el CI.

   b. Creación de la imagen: se realiza un commit de la rama de integración a la rama master utilizando *Conventional Commits* para realizar el despliegue y generar el versionado automático de los cambios del código. El versionado consiste en tres números separados por punto, '*a.b.c'*, donde '*a'* corresponde a la versión mayor, '*b'* corresponde a la versión menor y ''*c* corresponde a la versión parche. Entonces, dependiendo del tipo de *commit* (*fix, feat o breaking change*), se modificará una de las tres versiones de la siguiente manera: *fix* incrementa ''*c*, *feat* incrementa '*b'* y *breaking change* incrementa '*a'*. Luego de realizarse el *marge request* se puede visualizar el progreso del despliegue en la herramienta *OpenShift*. *OpenShift* es una plataforma en la nube para la gestión de contenedores, basada en *Kubernetes* que ofreciendo características adicionales (seguridad, herramientas de desarrollo, integración CI/CD, etc.). Una vez finalizado el *pipeline* se puede comprobar la creación de la imagen en la herramienta web *Harbor*. *Harbor* es una solución para el registro de imágenes de aplicaciones que permite almacenarlas, distribuirlas y gestionarlas.

   c. Creación de la aplicación: siguiendo los principios de *GitOps*, desde el repositorio de Git para el despliegue continuo CD se realiza la configuración de la aplicación mediante la creación de una *issue*. Se dispara un *pipeline* y, al finalizar, se crea una aplicación en *ArgoCD* desincronizada. *ArgoCD* es una

herramienta de despliegue continuo CD para *Kubernetes*, se especializa en mantener el estado de un clúster de *Kubernetes* sincronizado con la configuración declarativa almacenada en Git.

d. Sincronización de la aplicación: a través del repositorio de despliegue en Git se debe realizar la configuración y despliegue de la aplicación. Se debe crear una nueva rama y editar el archivo de configuración que se encuentra en formato *yaml*, para cambiar la versión, agregar secrets, variables de entorno, *configMaps*, etc. Una vez modificado el archivo de configuración, se debe realizar un *'merge request'* a la rama principal. Se puede visualizar el progreso del despliegue CD en OpenShift, una vez finalizado, será posible consumir la aplicación en el ambiente de integración. Si se desea desplegar la aplicación en el ambiente de certificación, se debe repetir este paso modificando el archivo de configuración correspondiente.

6) DDT: La Definición De Terminado (*DoD Definition of Done*) la establece el equipo y es un conjunto de criterios, que deben cumplirse para considerar que la historia de desarrollo se completó correctamente. La DDT definida por este equipo se muestra en la Figura 56. DDT.

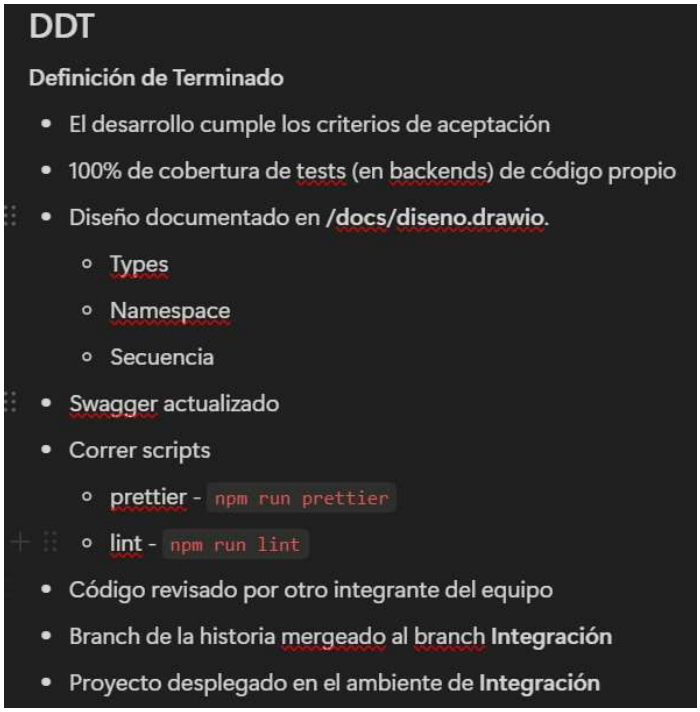


Figura 56. DDT

Fuente Elaboración propia, basada en la práctica

7) Pruebas de Stress: El objetivo de este paso es evaluar el rendimiento de la API bajo condiciones extremas de carga para identificar los límites de la aplicación, el punto donde comienza a fallar. Se busca probar la aplicación en escenarios que sobrepasan las condiciones normales o picos de demanda: número elevado de usuarios, transacciones, solicitudes, etc. Se busca identificar los picos de demanda, garantizar que la aplicación responda ante sobrecargas e identificar posibles acciones de mejora.

Las pruebas de Stress se realizan en *Apache JMeter*, una prueba básica de stress consiste en crear un plan de pruebas y configurar la cantidad de hilos, que indica el número de usuarios que van a consumir la aplicación en forma simultánea; el periodo de subida es la cantidad de tiempo que tarda *Jmeter* en incrementar desde cero a la cantidad de hilos especificada. Luego, se configura una solicitud HTTP indicando el servidor, la url y el tipo de método HTTP. Al ejecutar la prueba, en el árbol de resultados se visualizan los éxitos y los errores de esas solicitudes. Además, en el gráfico agregado se puede ver los resultados totales en tablas y gráficos. Esta información es suficiente para determinar si la aplicación responde correctamente o es necesario analizar la performance en búsqueda de mejoras.

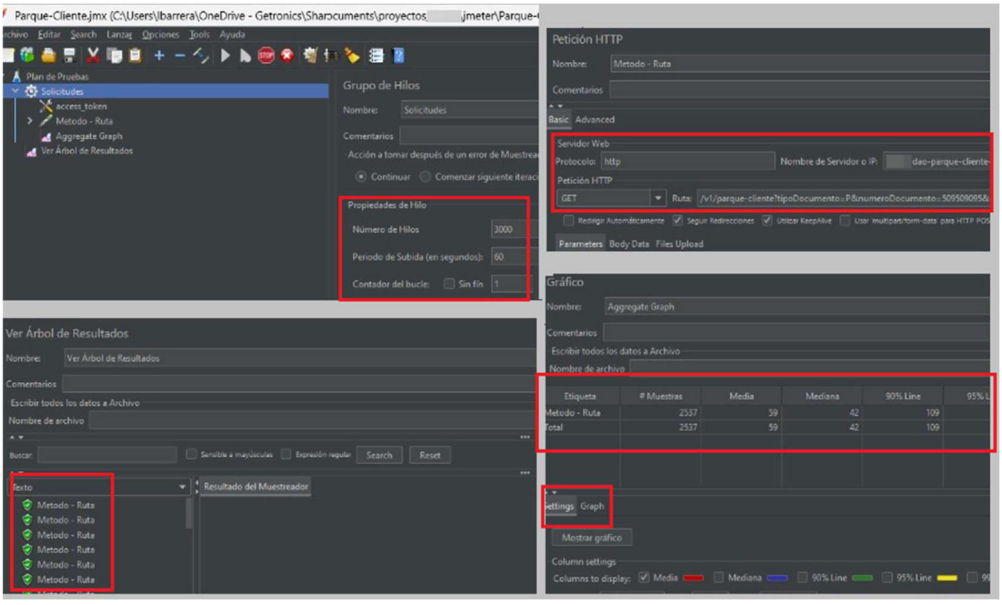


Figura 57. Pruebas Jmeter

Fuente Elaboración propia, basada en la práctica

8) Documentación: En la última tarea de una historia de desarrollo se debe generar la documentación de la aplicación. La documentación está conformada por una serie de archivos que tienen como objetivo brindar información y facilitar el uso de la API.

Se genera un documento con una descripción de la aplicación, los parámetros y las rutas desarrolladas, se generan los cURLs, también un archivo de texto con casos de ejemplo y el swagger explicado anteriormente.

Al finalizar el Sprint se realiza la ceremonia de Review donde se expone ante los usuarios el trabajo realizado en el Sprint en curso. La demostración es dirigida por la PO de la mesa, donde se repasan los compromisos tomados y se presenta un resumen de todas las historias trabajadas y el avance realizado. Luego, cada integrante del equipo expone las nuevas APIs desarrolladas y evolucionadas. Consiste en hacer una demostración de las funcionalidades logradas mediante el swagger de la API, se da un espacio para consultas. La Review es finalizada por la PO enumerando los compromisos asumidos para el próximo Sprint.

La siguiente ceremonia es la Retrospective, esta reunión interna del equipo consiste en poder identificar los logros del equipo, los problemas que surgieron y cómo se solucionaron. El objetivo es poder mejorar el proceso tomando compromisos que tengan la finalidad de lograrlos.

Continuando con la Tarea 6, se irán iterando el resto de las historias refinadas en diferentes sprints. Se explicará, sin entrar en detalle, el objetivo de cada una de ellas para permitir comprender su función en el contexto general en la Migración del Sistema de Beneficios.

H5.2: Como equipo de desarrollo quiero diseñar y desarrollar el dao-beneficios para obtener información de beneficios de un tipo y número de documento de la base de datos Beneficios en la consulta-beneficios.

La historia 5.2 consiste en desarrollar una nueva API que disponga de un endpoint de tipo GET llamado consulta-beneficios, que acceda a la base de datos de beneficios y permita obtener el detalle de todas las cuentas que poseen el beneficio activo, en proceso de alta o en proceso de baja de todos los clientes activos para un mismo tipo y número de documento.

H5.3: Como equipo de desarrollo quiero diseñar y desarrollar el fcd-beneficios para unificar la información del dao-parque y del dao-beneficios de un tipo y número de documento en la consulta-parque, agregando lógica de validaciones y clasificando las cuentas en las categorías descritas anteriormente.

La historia 5.3 consiste en desarrollar una API que consuma y orqueste las APIS desarrolladas en las historias descritas anteriormente, el dao de parque y el dao de beneficios. Un facade simplifica y unifica la interacción de varias APIs incluye agregación de datos, transformación de respuestas y aplicación de reglas de negocio. En esta primera etapa, se

debe desarrollar el endpoint consulta-parque que reciba un tipo y número de documento, el facade deberá obtener la información de la base de datos CRM y la información de la base de datos de beneficios a través de los DAOs, unificar ambas respuestas y generar una salida que permita establecer el estado de las cuentas y de la asignación de los beneficios.

H5.4: Tarea de análisis de validaciones para determinar si un cliente posee deuda.

Se planifican en paralelo las tres tareas de análisis de validaciones. En esta, se debe analizar de qué manera se puede obtener de la base de datos CRM el estado de la deuda de un identificador de cliente. La respuesta de esta consulta será un tipo de datos booleano indicando “verdadero” si tiene deuda y “falso” en caso contrario. Si el cliente tiene deuda, no podrá realizar gestiones de altas o bajas de beneficios. El entregable de esta tarea es una sentencia SQL que cumpla con esta especificación.

H5.5: Tarea de análisis de validaciones para determinar si una cuenta ya tiene el beneficio activo.

La tarea 5.5 consiste en verificar si una cuenta determinada tiene el beneficio activo en la base de datos CRM, de la misma manera que la tarea anterior, la respuesta será un tipo de datos booleano indicando verdadero si tiene el beneficio y falso en caso contrario. En este caso, si la cuenta posee el beneficio activo no podrá solicitar el alta y si no lo tiene, no podrá solicitar la baja de este. El entregable de esta tarea es una sentencia SQL que cumpla con esta especificación.

H5.6: Tarea de análisis de validaciones para determinar si un cliente tiene operaciones pendientes en el CRM.

Esta tarea consiste en verificar si el cliente tiene operaciones pendientes, que el cliente tenga operaciones pendientes impedirá que pueda realizar gestiones de cualquier tipo. La respuesta obtenida también será un booleano indicando “verdadero” si el cliente tiene órdenes pendientes y “falso” en caso contrario. El entregable será una sentencia SQL.

H5.7: Como equipo de desarrollo quiero diseñar y desarrollar la evolución del fcd-beneficios para integrar las validaciones de las tareas H5.4, H5.5 y H5.6 en la consulta-parque.

En la Historia 5.3 se desarrolló el GET consulta-parque del *facade* de beneficios y en las historias 5.4, 5.5 y 5.6 se desarrollaron consultas contra el CRM, que permiten obtener información específica del estado de deuda, asignación del beneficio y operaciones en proceso. La información obtenida en las tareas de desarrollo de consultas permite complementar la información obtenida en los DAOs de parque y de beneficios. Estos datos

complementarios permiten proveer la información necesaria para que el cliente de la API pueda determinar qué operaciones de altas y bajas de beneficios puede realizar y el estado de asignación del grupo de beneficio de un cliente.

H5.8: Como equipo de desarrollo quiero diseñar y desarrollar la evolución del fcd-beneficios para agregar el endpoint de modificar-parque.

Finalmente, en la historia 5.8 se desarrollará un nuevo endpoint en el facade de beneficios. El endpoint modificar-parque recibirá tres parámetros: tipo de documento, número de documento y el listado de cuentas a las que se les deberá dar de alta el beneficio. Si el tipo y número de documento no tiene un grupo creado, se deberá crear el mismo en la base de datos. Si ya posee un grupo, además, se podrán eliminar cuentas del grupo para dar lugar a otras cuentas dentro de un límite pre-acordado. Estas operaciones deberán cumplir una serie de validaciones a partir de las consultas de validación creadas anteriormente.

Con la finalización de esta historia se cumple con la Tarea 6 de desarrollo de APIS en ambientes bajos.

Retomando el objetivo general de la PPS “Diseñar, desarrollar, testear e implementar los microservicios y procedimientos de base de datos necesarios para realizar la migración de un sistema de beneficios” y los diseños macro expuestos en la “Figura 29. Diseño macro - Consulta parque” y la “Figura 30. Diseño macro - Modificación parque” se finaliza la descripción de la tarea actual exponiendo el resultado del endpoint consulta-parque y modificaciones del FCD-Beneficios.

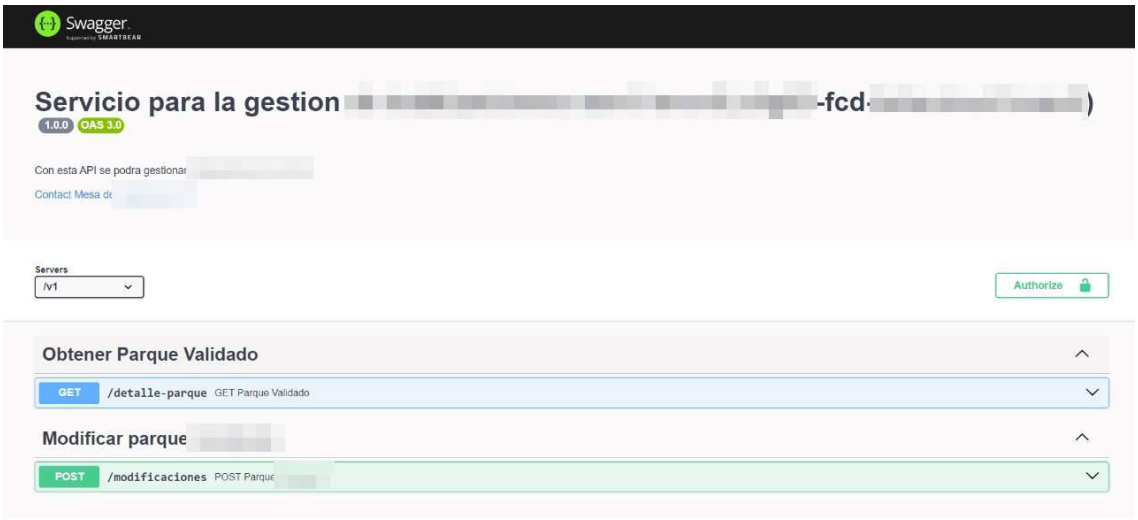


Figura 58. FCD Swagger

Fuente Elaboración propia, basada en la práctica

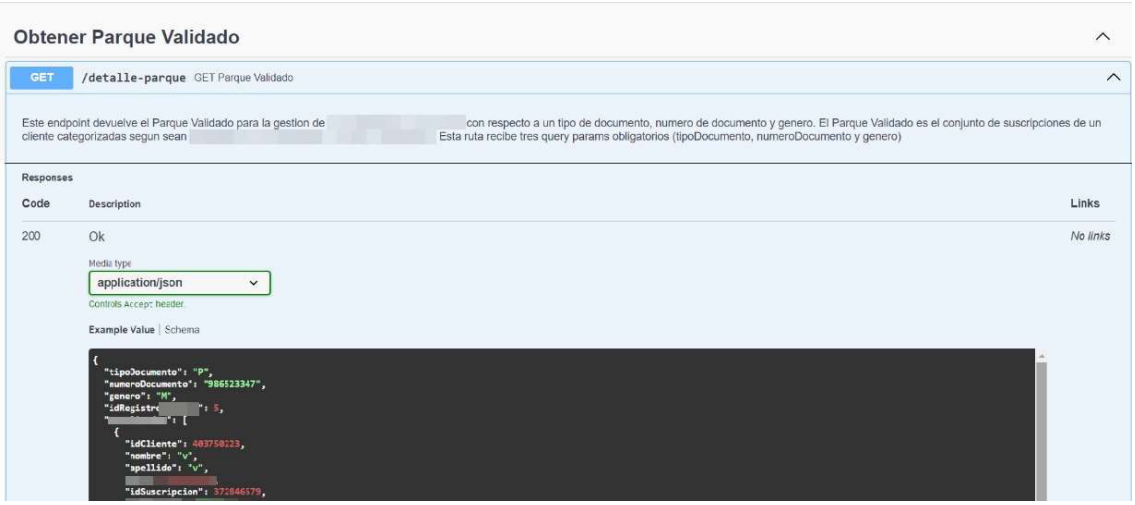


Figura 59. FCD GET

Fuente Elaboración propia, basada en la práctica

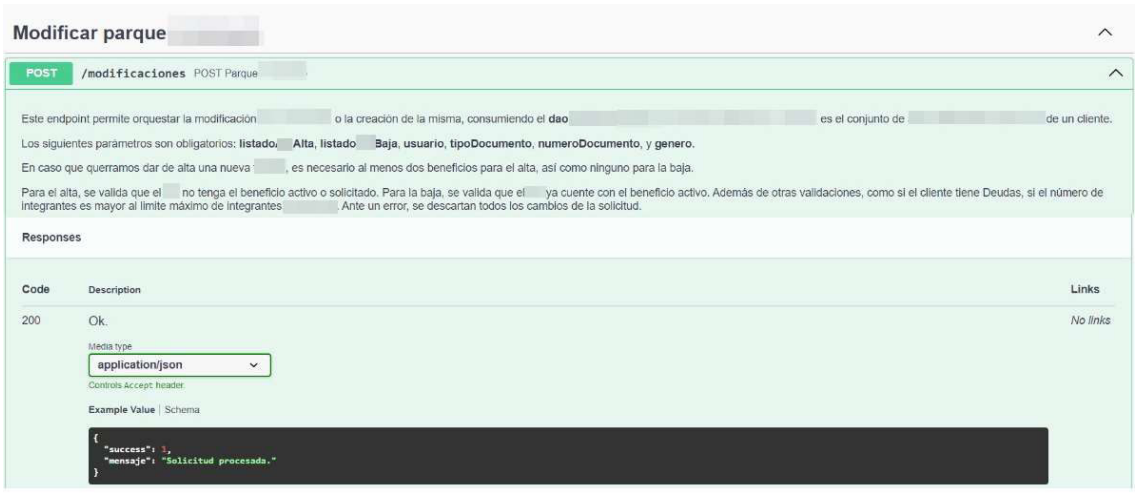


Figura 60. FCD POST

Fuente Elaboración propia, basada en la práctica

g. Tarea 7 – Evaluación y Verificación.

Luego de completar las tareas de una historia de desarrollo, esta está lista para ser tomada por el equipo de QA. El equipo de QA es parte del Scrum Team y, por lo tanto, participa de las mismas ceremonias. En cada Daily, se revisan las historias en las que están trabajando. Si es necesario, se coordinan reuniones entre los miembros del equipo de QA y el desarrollador de la API que se esté testeando. Se mantienen varios canales de comunicación entre desarrolladores y QA, como email, Teams, Jira y Discord. El equipo de QA prueba la funcionalidad de los endpoints, los métodos HTTP, verifica la validación de

parámetros de entrada, y realiza pruebas de estrés, latencia, interoperabilidad, documentación, entre otras. Si durante las pruebas, el equipo de QA detecta un defecto, reporta un bug en Jira, el cual debe ser abordado por uno de los desarrolladores del equipo para su resolución, repitiendo ese proceso por cada bug reportado.

Con el objetivo de aseguramiento de la calidad y seguridad del software, desde el equipo de DevSecOps de la compañía se establecen estándares para la utilización de determinadas herramientas que deben ser aplicadas por todos los equipos ágiles. Dos de estas herramientas son SonarQube y AppScan.

Siguiendo el propósito de garantizar la calidad del código de las aplicaciones desarrolladas se realiza un análisis estático de código mediante la herramienta SonarQube. SonarQube permite detectar bugs, vulnerabilidades, código duplicado, cobertura de los test y uso de estándares de codificación, entre otros. Este análisis se puede realizar manualmente durante la etapa de desarrollo para anticiparse y aplicar las recomendaciones de la herramienta en esta instancia. También, se puede realizar el análisis automáticamente durante la integración continua en GitLab. En ambos casos, SonarQube proporciona un informe donde indica las vulnerabilidades detectadas y recomendaciones asociadas para poder solucionarlas. En la Figura 61 y en la Figura 62 se muestra el estado general del FCD y el estado particular ante la detección de:

- New bugs: errores en el código que podría causar un comportamiento incorrecto o fallos que afecten el rendimiento, la confiabilidad y estabilidad. Algunos fallos detectables son los siguientes: “null pointer exceptions”, archivos o conexiones que se abren y no se cierran, condiciones siempre verdaderas o falsas y variables no inicializadas.
- New vulnerability: un fallo de seguridad en el código, que puede ser explotado por atacantes, poniendo en riesgo la confidencialidad, integridad o disponibilidad. Algunas vulnerabilidades detectables son las que se indican a continuación: “SQL Injection”, inyección de comandos, autenticación débil y criptografía insegura.
- New security hotspots: son áreas de código que podrían estar relacionadas con vulnerabilidades en las que el desarrollador debería realizar una inspección manual en el manejo de credenciales, criptografía insegura, autorización, SQL Injection.
- Added debt: la deuda nueva, se refiere a la deuda técnica añadida en la última actualización de código, la métrica se refiere al tiempo necesario para corregir problemas de calidad de código.

- New code smells: refiere a problemas de diseño o implementación de código que no causan errores directos, pero pueden dificultar la mantenibilidad del código. Ejemplos pueden ser: funciones largas, duplicación de código, variables no utilizadas, nombres de variables poco descriptivas.
- Coverage: Sonarqube permite integrarse con Jest para medir la cobertura de código, lo que facilita visualizar segmentos no probados.
- Duplications: son fragmentos de código que están repetidos en diferentes partes del proyecto, lo que influye en la mantenibilidad del código y aumentar el riesgo al hacer cambios o refactorizaciones.

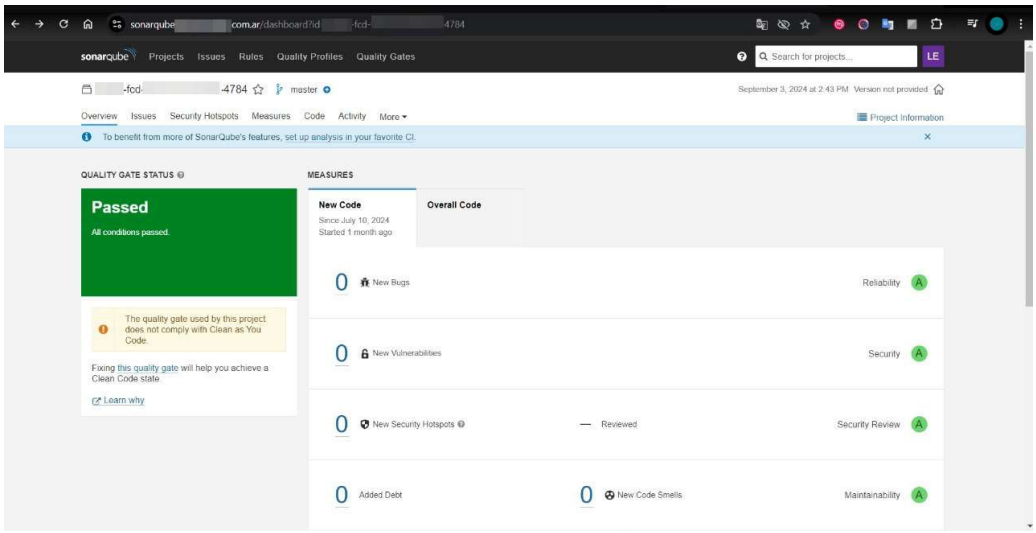


Figura 61. SonarQube detalle

Fuente Elaboración propia, basada en la práctica

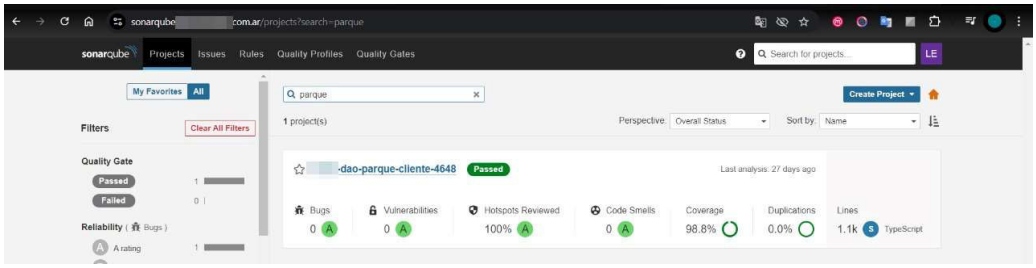


Figura 62. SonarQube resumen

Fuente Elaboración propia, basada en la práctica

AppScan, por su parte, está orientada al análisis de seguridad de las aplicaciones. AppScan analiza el código de manera estática (pruebas SAST) y, dinámicamente, de la aplicación en ejecución (pruebas DAST) para detectar vulnerabilidades de seguridad antes

del despliegue. La aplicación genera un informe detallando las vulnerabilidades detectadas y clasificándolas según sean críticas, altas, medias o bajas. Algunas de las vulnerabilidades detectadas por AppScan son las siguientes:

- Authentication.Credentials.Unprotected
- Inyección
- Desbordamiento de enteros
- Componente de código abierto
- Conjuntos de cifrado débiles - ataque ROBOT: el servidor admite conjuntos de cifrado vulnerables
- Scripting entre sitios reflejado
- Encontrado el patrón de error en la base de datos

En la Figura 63 se muestra la interface principal, que presenta un resumen de la cantidad de aplicaciones analizadas, el estado de resolución. También se clasifican las vulnerabilidades detectadas según sean críticas, altas, medias o bajas.

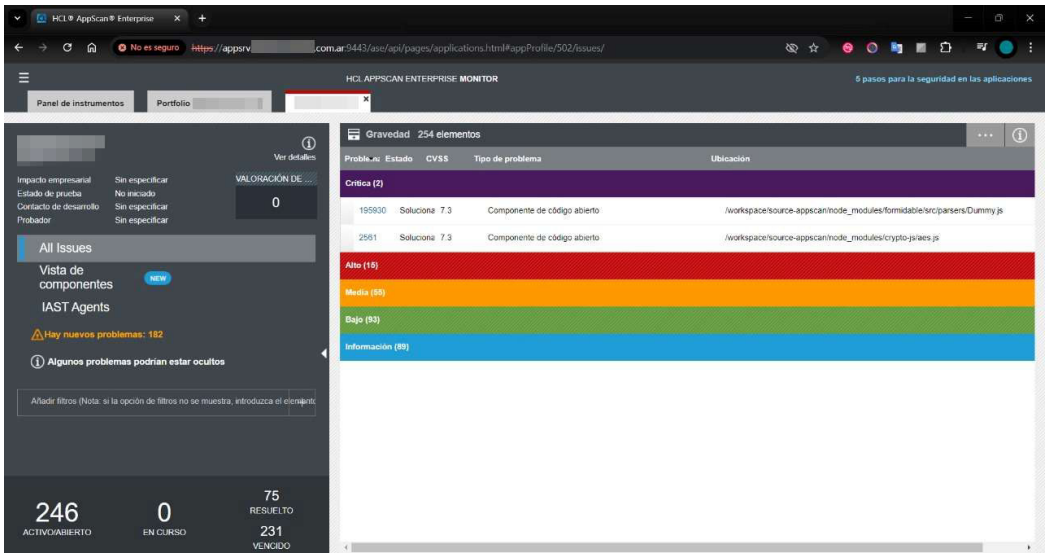


Figura 63. AppScan tablero resumen

Fuente Elaboración propia, basada en la práctica

En la Figura 64 se detalla una vulnerabilidad detectada, se informa la API analizada, la clasificación de la severidad, el archivo donde se detectó la vulnerabilidad, entre otros datos. Una característica para destacar es que la aplicación ofrece información suficiente para poder solucionar el problema detectado. Al hacer clic en el link "*Como solucionar el problema*"

(Figura 65) se informa la causa y la recomendación para su solución dependiendo del tipo de vulnerabilidad detectada.

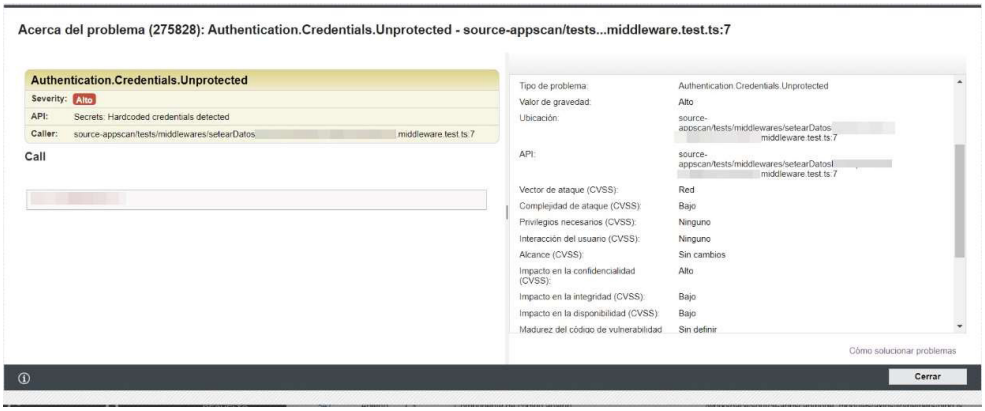


Figura 64. AppScan incidencia

Fuente Elaboración propia, basada en la práctica

Figura 65. AppScan solución

Fuente Elaboración propia, basada en la práctica

h. Tarea 8 – Implementación y migración.

Esta tarea consistió en la implementación de los procesos para la base de datos y microservicios desarrollados, la migración de datos del sistema actual que gestiona los beneficios y, por último, la desactivación de dicho sistema.

Como parte de la planificación de esta tarea, se lograron diferentes acuerdos. Por ejemplo, para poder avanzar con la implementación de los procesos, fue necesario coordinar con otros equipos de desarrollo quienes, a través de aplicaciones web y móviles, comenzaron a consumir las APIs desarrolladas en esta PPS. El impacto que tuvieron estos equipos estuvo dado por el hecho de que debían dejar de consumir el servicio desactivado y comenzar a consumir las nuevas APIs. Una vez acordada la fecha, se procedió con la etapa de implementación del proyecto. Otro acuerdo, realizado con los usuarios de negocio, consistió en que debían administrar un listado, depurado y corregido, con los grupos de cuentas asociadas que poseían el beneficio. Este listado fue necesario para la migración de datos del sistema actual a la nueva base de datos. Finalmente, en la fecha acordada se desactivó el sistema que actualmente gestiona los beneficios. La implementación coordinada de todos los equipos se llevó a cabo en una ventana de varias horas, durante las cuales las funcionalidades reemplazadas no estuvieron disponibles en las aplicaciones web y móviles.

En primera instancia, se implementaron los procesos de la base de datos productiva bajo el rol de administrador de base de datos, donde se crearon las estructuras de datos y los procedimientos almacenados previamente descritos. A continuación, se procedió con la migración de datos desde el listado provisto por los usuarios de negocio, con la información de cuentas con beneficio activo. Esto consistió en insertar los registros en las tablas del modelo para reflejar la relación de las cuentas que poseían el beneficio activo. Se tuvo en cuenta que también fue necesario sincronizar el apagado del sistema y la extracción del listado para que este fuera lo más completo posible. Una vez obtenido el listado, se llevó a cabo el apagado del sistema anterior. Finalmente, se implementaron las APIs desarrolladas, cuyo despliegue fue similar al detallado en esta PPS para los ambientes bajos, dirigiendo las tareas hacia el ambiente productivo. Para esta etapa, la base de datos ya era completamente funcional y contenía los datos actualizados. Una vez finalizada esta tarea, se comunicó a los equipos con dependencia funcional, habilitándolos para avanzar con sus procesos de implementación. Después de que todos los equipos de desarrollo completaran sus implementaciones, un grupo de usuarios fue responsable de probar todas las funcionalidades desarrolladas. Al cumplirse los objetivos acordados, la implementación fue aprobaba y el proceso concluyó satisfactoriamente.

Reflexión sobre la Práctica Profesional Supervisada

La PPS se llevó a cabo en un entorno laboral, representando la culminación de años de aprendizaje académico y profesional. El proyecto fue abordado durante una fase de transición en la que pasé de desempeñar el rol de desarrollador T-SQL, administrador de bases de datos y referente de negocio para el módulo de beneficios en un equipo, a integrarme en un nuevo equipo donde debía mantener esas responsabilidades y, además, adquirir nuevas habilidades en el desarrollo Backend. A medida que avancé en el cronograma, pude realizar diversas tareas, adquirir nuevos conocimientos y aplicar otros ya adquiridos. Las habilidades técnicas, las experiencias y enseñanzas transmitidas por los profesores de la Universidad, resultaron fundamentales en este proceso. Sin embargo, enfrenté varios desafíos. Entre ellos, la superposición de responsabilidades, como la transición al nuevo equipo, la necesidad de desarrollar nuevas competencias, la continuidad en mi rol dentro del equipo anterior, la capacitación del nuevo equipo en las tareas heredadas, y la escritura y desarrollo de la PPS. Otro impedimento significativo fue un cambio de alcance en el proyecto, derivado de la finalización del soporte para una aplicación subyacente, lo que obligó a una replanificación de las tareas e historias comprometidas por el equipo. A pesar de estos desafíos, pude aportar valor desde mi rol, brindando conocimiento del negocio y habilidades previas que contribuyeron al desarrollo de la nueva aplicación. La experiencia adquirida durante esta etapa es fundamental, ya que ha permitido integrarme plenamente en el equipo de desarrollo y adquirir las competencias necesarias para abordar cualquier historia o tarea del backlog. Finalmente, la PPS me brindó la oportunidad de aplicar y ampliar mis conocimientos técnicos, desarrollar habilidades blandas, superar desafíos de gestión y consolidarme como un miembro activo y capaz en un equipo de desarrollo.

Bibliografía

Api Restful. (14 de 07 de 2024). Obtenido de https://www.redhat.com/es/topics/api/what-is-a-rest-api

Arquitectura de tres niveles. (14 de 07 de 2024). Obtenido de https://www.ibm.com/mx-es/topics/three-tier-architecture

Backend. (7 de 7 de 2024). Obtenido de https://nestrategia.com/desarrollo-web-back-end-front-end/#:~:text=En%20otras%20palabras%2C%20el%20Back,la%20comunicaci%C3%B3n%20con%20el%20servidor.

Base de datos Relacional. (21 de 05 de 2024). Obtenido de https://aws.amazon.com/es/relational-database/

Capa de Servicio. (16 de 07 de 2024). Obtenido de https://fineproxy.org/es/wiki/service-layer/

Create Trigger (Transact-SQL). (23 de 05 de 2024). Obtenido de https://learn.microsoft.com/es-es/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16

DAO. (16 de 07 de 2024). Obtenido de https://www.digitalocean.com/community/tutorials/dao-design-pattern

Facade. (16 de 07 de 2024). Obtenido de https://thepowerups-learning.com/patrones-de-diseno-facade/

Git. (03 de 07 de 2024). Obtenido de https://learn.microsoft.com/es-es/devops/develop/git/what-is-git

GitLab. (04 de 07 de 2024). Obtenido de https://formadoresit.es/que-es-gitlab-y-para-que-sirve/

HCL AppScan. (14 de 9 de 2024). Obtenido de https://www.hcl-software.com/appscan

HTTP y API REST. (14 de 07 de 2024). Obtenido de https://code.tutsplus.com/es/a-beginners-guide-to-http-and-rest--net-16340t

Introducción a la programación con Transact-SQL. (23 de 5 de 2024). Obtenido de https://learn.microsoft.com/es-es/training/modules/get-started-transact-sql-programming/

Javascript. (06 de 07 de 2024). Obtenido de https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Jest. (17 de 07 de 2024). Obtenido de https://jestjs.io/es-ES/

Jira Software. (28 de 05 de 2024). Obtenido de https://www.atlassian.com/es/software/jira

Jmeter. (18 de 07 de 2024). Obtenido de https://jmeter.apache.org/

Joi. (17 de 07 de 2024). Obtenido de https://medium.com/@diego.coder/validaci%C3%B3n-de-datos-y-estructuras-en-node-js-con-joi-a157cfc6c4bf

Metodología Ágil. (04 de 07 de 2024). Obtenido de https://www.redhat.com/es/topics/devops/what-is-agile-methodology

Microservicios. (10 de 07 de 2024). Obtenido de https://www.atlassian.com/es/microservices/microservices-architecture

Node. (07 de 07 de 2024). Obtenido de https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Node.js documentation. (09 de 07 de 2024). Obtenido de https://nodejs.org/docs/latest/api/

Patrones de diseño. (16 de 07 de 2024). Obtenido de https://thepowerups-learning.com/patrones-de-diseno/#patrones-estructurales

Procedimientos almacenados. (23 de 05 de 2024). Obtenido de https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16

Repository Pattern. (16 de 07 de 2024). Obtenido de https://www.linkedin.com/pulse/what-repository-pattern-alper-sara%C3%A7

Sequelize. (18 de 07 de 2024). Obtenido de https://sequelize.org/
SonarQube. (14 de 09 de 2024). Obtenido de https://sentrio.io/blog/que-es-sonarqube/
SQL. (21 de 05 de 2024). Obtenido de https://aws.amazon.com/es/what-is/sql/
SQL Server. (23 de 05 de 2024). Obtenido de ¿Qué es SQL Server?: https://learn.microsoft.com/es-es/sql/sql-server/what-is-sql-server?view=sql-server-ver16
SQL Server Agent. (23 de 05 de 2024). Obtenido de https://learn.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver16
SQL Server Management Studio. (23 de 05 de 2024). Obtenido de https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16
Swagger. (18 de 07 de 2024). Obtenido de https://keepcoding.io/blog/que-es-swagger/
TDD. (14 de 07 de 2024). Obtenido de https://intelequia.com/es/blog/post/qu%C3%A9-es-y-para-qu%C3%A9-sirve-un-tdd-o-test-driven-development#:~:text=%C2%BFQu%C3%A9%20es%20Test%20Driven%20Development,antes%20de%20escribir%20el%20c%C3%B3digo.
The Scrum Events. (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events
The Scrum Team. (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team
Typescript vs Javascript. (11 de 07 de 2024). Obtenido de https://profile.es/blog/que-es-typescript-vs-javascript/
Visual Studio Code. (23 de 05 de 2024). Obtenido de https://code.visualstudio.com/
What is a Daily Scrum? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-daily-scrum
What is a Developer? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-developer
What is a Product Backlog? (28 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-product-backlog
What is a Product Goal? (28 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-product-goal
What is a Product Owner? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-product-owner
What is a Scrum Master? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-scrum-master
What is a Sprint Backlog? (28 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-sprint-backlog
What is a Sprint Goal? (28 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-sprint-goal
What is a Sprint Retrospective? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint-retrospective
What is a Sprint Review? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint-review
What is a Sprint? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint
What is Scrum? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/
What is Sprint Planning? (26 de 05 de 2024). Obtenido de https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-sprint-planning