Noelia Fernanda Segovia

Reingeniería de un Sistema de Administración de Tarjetas

2024

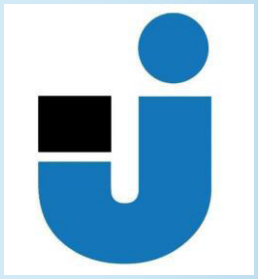
Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA

Informe final

Reingeniería de un Sistema de Administración de Tarjetas

Noelia Fernanda Segovia

Florencio Varela, noviembre 2024

ESTUDIANTE

Nombre y apellido: Noelia Fernanda Segovia

Correo electrónico: noe0754@gmail.com

ORGANIZACIÓN DONDE SE REALIZA LA PRACTICA PROFESIONAL SUPERVISADA

Nombre de la empresa: Red Link S.A

Dirección: AV. BOUCHARD 557 PISO:5

Telefono: 4119-1200

Sector: UIP- Equipo de administración de tarjetas

TUTOR ORGANIZACIONAL

Nombre y apellido: Silvia Brambilla

Correo electrónico: brambilla_silvia@redlink.com.ar

DOCENTE SUPERVISOR

Nombre y apellido: Dr. Ing. Martín Morales

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

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS

Nombre y Apellido: Lic. y Prof. Carolina Kelly

Correo electrónico: kellygcarolina@gmail.com

COORDINADOR DE LA CARRERA DE INGENIERIA EN INFORMÁTICA

Nombre y apellido: Dr. Ing. Martín Morales

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

1. Resumen

El presente informe describe el desarrollo y despliegue del Módulo 1 (Consultivo) del sistema SoatWeb, realizado como parte de la Práctica Profesional Supervisada (PPS). El motivo de esta práctica fue la necesidad de eliminar la dependencia de aplicaciones de escritorio, implementando un sistema accesible desde la web que facilitara la consulta de información clave para los usuarios. El propósito fue lograr una plataforma moderna y accesible que mejorara la eficiencia y disponibilidad del servicio.

La metodología incluyó la configuración de OpenShift para el entorno de producción, la automatización mediante *pipelines* de GitLab CI/CD, y la realización de pruebas exhaustivas para garantizar la calidad y estabilidad del sistema. Los resultados obtenidos permitieron alcanzar un sistema funcional y confiable, que proporciona una mejora significativa en la accesibilidad y gestión de la información consultiva.

Palabras clave: SoatWeb, OpenShift, CI/CD, Módulo Consultivo, POC, Desarrollo Web.

2. Abstract

This report outlines the development and deployment of Module 1 (Consultative) of the SoatWeb system, conducted as part of the Supervised Professional Practice (PPS). The motivation for this practice was the need to eliminate the dependency on desktop applications by implementing a web-accessible system that facilitates key information consultation for users. The purpose was to achieve a modern and accessible platform that improves service efficiency and availability.

The methodology included configuring OpenShift for the production environment, automating deployments through GitLab CI/CD pipelines, and conducting extensive testing to ensure system quality and stability. The results yielded a functional and reliable system that significantly enhances the accessibility and management of consultative information.

Keywords: SoatWeb, OpenShift, CI/CD, Consultative Module, POC, Web Development.

3. Dedicatorias y agradecimientos

Este proyecto académico representa años de dedicación y perseverancia, y ha sido una fuente de enorme satisfacción personal. Agradezco profundamente a mi familia, amigos, y compañeros de trabajo, y a cada persona que estuvo en este camino conmigo, quienes siempre me brindaron su apoyo incondicional durante este recorrido. También deseo destacar el papel fundamental de la universidad, de esta carrera, y de cada uno de los profesores que me guiaron en las diversas materias que cursé, proporcionándome las herramientas y conocimientos necesarios para avanzar. Este proyecto no solo representa un logro académico, sino también la realización de un sueño personal y la satisfacción de haber alcanzado una meta significativa.

4. Índice

5. Índice de imágenes

6. Introducción

El presente proyecto de reingeniería surgió como una necesidad de mejorar la experiencia del cliente, la eficiencia y la funcionalidad del sistema actual, denominado Sistema Online de Administración de Tarjeta (SOAT), encargado de gestionar el ciclo de vida de las tarjetas de débito.

Esta aplicación gestiona desde la creación de una tarjeta hasta su baja, administrando todo el ciclo de vida completo. SOAT tiene aproximadamente 30 años de antigüedad, por lo que se identificó la necesidad de modernizarlo.

El trabajo fue llevado a cabo en la empresa Red Link S.A, la cual es una empresa argentina que gestiona una red de servicios electrónicos y soluciones tecnológicas para el sector financiero y otros sectores bancarios. Su principal actividad es ofrecer una plataforma interconectada para transacciones financieras, como operaciones en cajeros automáticos, pagos electrónicos, *home banking*, y servicios de banca móvil.

Para este proyecto se realizó un análisis exhaustivo del sistema actual y, utilizando nuevas tecnologías, se buscó optimizar los procesos, aumentar la velocidad de procesamiento, reducir errores y mejorar la experiencia del usuario. El proyecto fue diseñado para aplicar la metodología *scrum*[1]. Se realizó una investigación de funcionalidades para direccionar y priorizar las distintas etapas del mismo.

La reingeniería tiene como objetivo transformar SOAT en una plataforma web, eliminando la necesidad de que cada usuario realice actualizaciones e instalaciones adicionales en cada nueva versión del programa.

[1] Scrum es una metodología ágil de gestión de proyectos que se utiliza principalmente en el desarrollo de software, pero también en otros tipos de proyectos. Se enfoca en dividir el trabajo en pequeños ciclos iterativos llamados *sprints*, que suelen durar entre 1 y 4 semanas. Durante cada *sprint*, el equipo trabaja en un conjunto específico de tareas priorizadas (*product backlog*) para entregar incrementos funcionales del producto.

Para lograrlo, se emplean tecnologías como Angular para el desarrollo del *frontend*[2] y Java para el desarrollo del *backend*[3], enfocándose en la implementación de un *Backend For Frontend* (BFF).

En el marco de la presente Práctica Profesional Supervisada (PPS), el proyecto se centrará en el desarrollo e implementación de una primera versión modernizada del sistema, abarcando principalmente las funcionalidades consultivas del sistema actual.

7. Objetivos

En este proyecto, las metas por alcanzar, a largo plazo, son:

A. Construir una nueva herramienta que sea más eficaz, simple e intuitiva.

B. Desarrollar un sistema que mejore la experiencia de sus usuarios a partir de una mejor comprensión de sus necesidades.

C. Desarrollar una herramienta que funcione desde cualquier plataforma web para obtener acceso desde cualquier ubicación y actualización permanente.

D. Asegurar la escalabilidad del sistema, diseñando la nueva plataforma de manera que sea capaz de manejar un aumento sostenido en la cantidad de usuarios y transacciones, garantizando un rendimiento óptimo en todo momento.

E. Garantizar la seguridad de los datos: implementar medidas de seguridad robustas para proteger la información sensible de los usuarios y cumplir con las normativas de protección de datos.

---

[2] El *frontend* es la parte de un sitio web que interactúa directamente con los usuarios, encargándose de la interfaz visual y la experiencia de usuario. Se desarrolla con tecnologías como HTML, CSS y JavaScript.

[3] El *backend* es la parte que gestiona el procesamiento de datos, la lógica del negocio y la comunicación con la base de datos, operando detrás de escena para que el sistema funcione correctamente. Utiliza lenguajes del lado del servidor como Python, Java o Node.js.

Las metas concretas y medibles por alcanzar son:

A. Investigar y evaluar el sistema actual para determinar todas las funcionalidades con las que cuenta.

B. Investigar y elegir las tecnologías y herramientas más apropiadas para el desarrollo del nuevo sistema, considerando factores como escalabilidad, seguridad, y facilidad de mantenimiento.

C. Crear un diseño de arquitectura que permita la fácil adición de nuevas funcionalidades y la escalabilidad del sistema en función de la demanda.

D. Desarrollar un prototipo y maqueta del nuevo sistema para validar el diseño y la experiencia del usuario antes de la implementación completa.

E. Desarrollar un plan de continuidad del negocio.

8. Tareas por ejecutar

En esta sección se explican las tareas a desarrollar a lo largo del proyecto.

Tarea 1: Identificación y Modularización del Sistema Actual

- Descripción: Analizar el sistema existente para identificar todas sus funcionalidades y dividirlas en módulos.
- Acción: Documentar y clasificar las funcionalidades en Módulo uno (Consultivo), que abarca funciones relacionadas con consultas y visualización de información.
- Objetivo: Facilitar la reingeniería y el desarrollo modular del nuevo sistema.

Tarea 2: Prueba de Concepto (POC)

- Descripción: Desarrollar una Prueba de Concepto para validar la interacción entre el *frontend* y el *backend*.

- Acción: Crear una pantalla simple que realice una consulta al *backend* y muestre los resultados.
- Objetivo: Verificar la viabilidad técnica y establecer una base para el desarrollo.

Tarea 3: Diseño de la Base de Datos

- Descripción: Crear la estructura de la base de datos que soportará el nuevo sistema.
- Acción: Analizar los requisitos de datos y diseñar el esquema de la base de datos relacional.
- Objetivo: Asegurar una base de datos eficiente y escalable.

Tarea 4: Análisis de Seguridad

- Descripción: Determinar los mecanismos de autorización y autenticación que se implementarán en el sistema.
- Acción: Investigar y definir los métodos de autenticación (por ejemplo, OAuth2, JWT) y autorización (roles, permisos).
- Objetivo: Asegurar que el sistema cuente con medidas de seguridad robustas.

Tarea 5: Refinamiento del Módulo uno (Consultivo)

- Descripción: Mejorar y detallar las especificaciones del Módulo uno antes de iniciar el desarrollo.
- Acción: Revisar y ajustar los requisitos funcionales y no funcionales del Módulo Consultivo.
- Objetivo: Garantizar que el Módulo Consultivo esté completamente definido y listo para el desarrollo.

Tarea 6: Desarrollo del Módulo Consultivo

- Descripción: Implementar el Módulo uno que incluye consultas y detalles de tarjetas, productos, prefijos[4], usuarios y entidades.
- Acción: Desarrollar funcionalidades de consulta y visualización de datos en el *frontend* y *backend*.
- Objetivo: Proveer una herramienta consultiva robusta y eficiente.

Tarea 7: Desarrollo del sistema de *Login* y Seguridad

- Descripción: Implementar el sistema de *login*, autorización y gestión de contraseñas.
- Acción: Desarrollar funcionalidades de autenticación, autorización y cambio de contraseñas.
- Objetivo: Asegurar un acceso seguro y controlado al sistema.

Tarea 8: Gestión de Usuarios

- Descripción: Implementar funcionalidades de creación, modificación y eliminación de usuarios.
- Acción: Desarrollar funcionalidades CRUD para el manejo de usuarios.
- Objetivo: Facilitar la administración de usuarios del sistema.

Tarea 9: Configuración para Despliegue

[4] Prefijos: son los primeros dígitos de la tarjeta, que identifican al emisor y el tipo de tarjeta. Estos prefijos permiten identificar rápidamente la red y el banco emisor de la tarjeta.

- Descripción: Preparar la configuración del proyecto para su despliegue en entornos de desarrollo y producción en Openshift.
- Acción: Configurar flujos de trabajo de CI/CD y ajustar configuraciones específicas de Openshift.
- Objetivo: Asegurar un proceso de despliegue ágil y eficiente.

Tarea 10: Generación de *Release* y carga inicial de datos

- Descripción: Preparar y desplegar la primera versión del sistema, asegurando que todos los componentes estén listos para su uso en el entorno de producción.
- Acción: Generar una versión, realizar pruebas finales y cargar datos iniciales en el entorno de producción.
- Objetivo: Asegurar un lanzamiento suave y funcional de la primera versión del sistema.

Tarea 11: Implementación del Módulo Consultivo en Producción

- Descripción: Desplegar el Módulo uno (Consultivo) en el entorno de producción, proporcionando a los usuarios las funcionalidades de consulta y visualización de información.
- Acción: Realizar pruebas finales y el despliegue del módulo operativo en el entorno de producción.
- Objetivo: Iniciar la transición al nuevo sistema mediante el despliegue de las funcionalidades consultivas, permitiendo a los usuarios comenzar a interactuar con la nueva plataforma.

9. Cronograma de trabajo

Nº tarea / Sprint	SPRINT 1	SPRINT 2	SPRINT 3	SPRINT 4	SPRINT 5	SPRINT 6	SPRINT 7	SPRINT 8
Tarea 1	X							
Tarea 2	X							
Tarea 3		X						
Tarea 4		X						
Tarea 5			X					
Tarea 6			X	X				
Tarea 7				X	X	X		
Tarea 8						X		
Tarea 9							X	
Tarea 10								X
Tarea 11								X

10. Desarrollo

En esta sección se proporcionará una descripción detallada de los avances realizados y resultados obtenidos en el desarrollo del proyecto.

10.1 Identificación y Modularización del Sistema Actual

El objetivo principal fue comprender la estructura y los componentes del sistema existente para facilitar su análisis y posterior desarrollo.

Para realizar esta tarea, fue necesario acceder al código fuente y a la documentación relevante del aplicativo. También se ejecutaron pruebas para comprender su comportamiento. Paralelamente, se enfocó en la experiencia de los usuarios, identificando los principales problemas a través de jornadas de observación y entrevistas con diferentes áreas involucradas que utilizaban la aplicación por diversos motivos.

Inicialmente, se decidió realizar una prueba utilizando el aplicativo. Se empleó un usuario en un ambiente de desarrollo y se accedió al sistema, observando que, al iniciar sesión, se presentaba un “menú” poco intuitivo, al igual que los buscadores. A partir de esta experiencia, se pudieron identificar las siguientes secciones:

- *Login*
- Menú
- Consulta de tarjetas
- Configuraciones de la entidad
- Modificaciones
- Operaciones
- Confirmaciones

Posteriormente, se levantó un entorno de desarrollo con el código fuente y se realizaron las configuraciones necesarias para efectuar una prueba de depuración.

Esto permitió comenzar a comprender el flujo del aplicativo, tanto a nivel funcional como a nivel de código. Se configuró un entorno local utilizando el IDE Visual Studio 2013[5]. Durante estas pruebas, se observó que cada sección estaba representada por una clase específica en el código. Para obtener una mayor comprensión, fue necesario explorar el lenguaje C++, cuyo conocimiento hasta ese momento era limitado. No obstante, con la ayuda de los manuales del aplicativo, se logró realizar un primer relevamiento general.

Asimismo, en el marco del rediseño del sistema de administración de tarjetas, se llevó a cabo un relevamiento enfocado en el usuario. Se realizaron entrevistas y observaciones tanto internas como con entidades clientes, con el objetivo de detectar comportamientos de usuario y oportunidades de mejora. Se entrevistó a más de 15 personas, incluyendo clientes de entidades bancarias. La participación de estos clientes fue crucial, ya que brindó una perspectiva directa sobre sus necesidades y expectativas respecto al nuevo sistema. De esta manera, se pudo asegurar que el desarrollo del nuevo aplicativo estuviera alineado con las demandas del mercado y las preferencias de los usuarios finales.

Dado que el aplicativo actual abarcaba varias secciones obtenidas a partir de los distintos relevamientos, pruebas y documentación realizados, se optó por dividir sus funcionalidades de la siguiente manera.

El Módulo uno (Consultivo) abarcó exclusivamente operaciones relacionadas con consultas y la visualización de información sobre el ciclo de vida de las tarjetas y todas las configuraciones relacionadas con las mismas. También incluyó la sección de gestión de usuarios y configuraciones propias de cada entidad.

Este módulo fue el enfoque principal adoptado para el presente informe y se analizará en profundidad en los siguientes apartados.

[5] Visual Studio 2013 es una versión del entorno de desarrollo integrado (IDE) de Microsoft que está diseñada para ayudar a los desarrolladores a escribir, compilar, depurar y mantener aplicaciones.

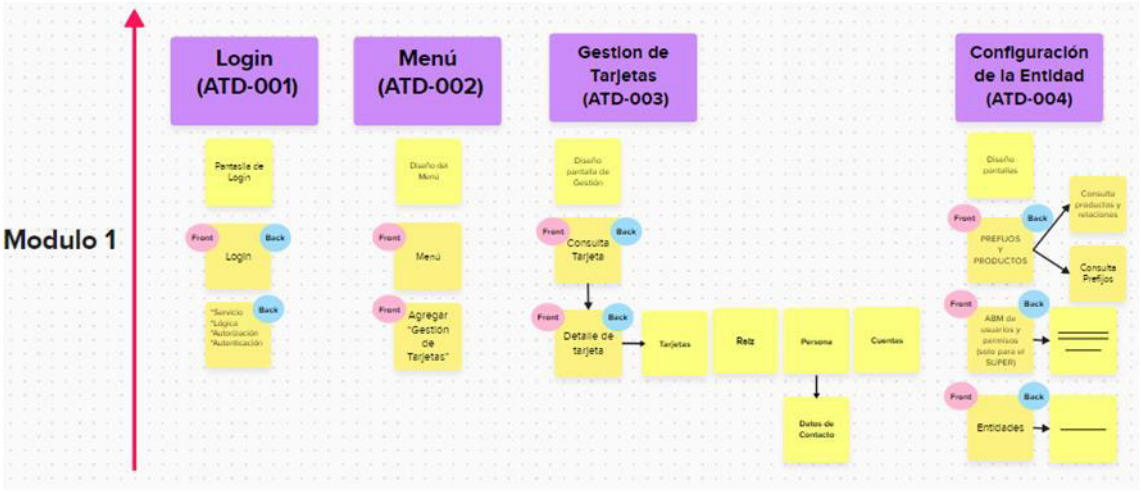


Imagen 1. Representación del Módulo uno.

Fuente: Elaboración propia, basada en la práctica.

La Imagen 1 representa las principales funciones que debía contener el nuevo sistema para este primer módulo. Cuenta con un "*login*", un "menú", una sección de "gestión de tarjetas" y una sección para las "configuraciones de las entidades". Dentro de cada función, se identificaron los requerimientos necesarios para que estuvieran completas, al igual que en el sistema de escritorio.

Para el "*login*", fue necesario diseñar una pantalla y desarrollar tanto el *frontend* como el *backend*, contemplando la creación de un servicio que abarcara la lógica de autorización y autenticación.

En cuanto al "menú", se requirió el diseño de una interfaz que, a nivel *frontend*, se integrara en el proyecto, incluyendo la sección de "gestión de tarjetas" para esta primera etapa.

La "gestión de tarjetas" siguió los mismos pasos que los anteriores: diseño de una pantalla y desarrollo *frontend* y *backend* para la consulta de tarjetas, además de proporcionar un apartado de detalle que incluyera información más específica de cada tarjeta.

Por último, se agregó la sección de configuración de entidad, donde se proporcionó información acerca de las configuraciones disponibles para que nuestros clientes pudieran acceder a las mismas.

10.2 Prueba de Concepto (POC[6])

El objetivo de esta tarea estuvo en analizar la viabilidad de crear un *frontend* desarrollado en Angular y un *backend* en Java (tarea que, cuando fuera aprobada, serviría de inicio del desarrollo del proyecto).

En este ítem se definieron las tecnologías a utilizar que fueron sugeridas en principio por la empresa donde se desarrolló esta PPS.

Para el *frontend*, se decidió utilizar Angular, un *framework* de desarrollo *frontend* basado en TypeScript[7], creado y mantenido por Google. Angular está diseñado para desarrollar aplicaciones web dinámicas de una sola página (SPA), y su arquitectura modular facilita la organización y reutilización del código. Además, ofrece herramientas como Angular CLI para simplificar la creación, compilación y despliegue de aplicaciones. Las principales razones para elegir Angular fueron:

- Arquitectura basada en componentes: Facilita la organización y reutilización del código.
- TypeScript: Mejora la calidad del código al ofrecer tipado estático y facilita la detección de errores.
- Data Binding bidireccional: Simplifica la sincronización entre el modelo y la vista.
- Angular CLI: Facilita la creación, compilación y despliegue de la aplicación.
- Comunidad activa: ofrece amplía disponibilidad de recursos y soporte.

Para el *backend*, se optó por Java, un lenguaje de programación de propósito general, orientado a objetos, y ampliamente utilizado en el desarrollo de aplicaciones *backend*. En este proyecto,

---

[6] POC (Prueba de Concepto): se refiere a la implementación inicial y preliminar de una idea o solución, destinada a demostrar su viabilidad técnica y funcional.

[7] TypeScript es un superconjunto de JavaScript que añade a JavaScript tipado estático opcional y funciones avanzadas. Ha sido desarrollado por Microsoft y se publicó por primera vez en octubre de 2012. Desde su lanzamiento en 2012, se ha extendido rápidamente entre la comunidad de desarrolladores web.

se utilizó en combinación con Spring Boot, un *framework* que facilita el desarrollo de aplicaciones empresariales, proporcionando configuraciones automáticas y un ecosistema robusto. Las principales razones para elegir Java con Spring Boot fueron:

- Configuración automática: Simplifica la configuración en función de las dependencias del proyecto.
- Servidor embebido: Permite ejecutar la aplicación como un archivo JAR independiente.
- Gestión de dependencias: Usa Maven o Gradle, facilitando la integración de bibliotecas.
- Monitorización y administración: Proporciona herramientas como Spring Boot Actuator que permiten monitorear la aplicación.
- Ecosistema extenso: Se integra fácilmente con otros componentes de Spring, como Spring Data y Spring Security.

En resumen, se decidió utilizar Java con Spring Boot, lo que permite enfocarse más en la lógica de negocio y menos en la configuración y la infraestructura.

Una vez definidos los lenguajes, se seleccionaron los entornos de desarrollo integrados (IDEs). Para el *backend*, se continuó utilizando Eclipse, dado el conocimiento previo y su uso extendido en la empresa.

Imagen 2. Logotipo de Eclipse.

Fuente: Recuperado de https://es.m.wikipedia.org/wiki/Archivo:Eclipse-Luna-Logo.svg

Para el *frontend*, se eligió Visual Studio Code por su integración con Git, terminal integrada y extensiones que facilitan el desarrollo.

Imagen 3. Logotipo de *Visual Studio Code*.

Fuente: Recuperado de https://www.stickpng.com/img/icons-logos-emojis/tech-companies/visual-studio-code-full-logo

Otras herramientas utilizadas en esta PoC fueron:

Postman: Se utilizó para crear y probar las llamadas a los servicios del *backend*, organizando las solicitudes en colecciones para una mejor gestión.

Imagen 4. Logotipo de *Postman*.

Fuente: Recuperado de https://logosdownload.com/postman

SQL Developer: Usado para la interacción con la base de datos Oracle 18c, la cual contiene tablas y procedimientos almacenados con información productiva.

Imagen 5. Logo de *SQL Developer*.

Fuente: Recuperado de https://forums.oracle.com/ords/r/apexds/community/q?question=instale-el-sql-developer-pero-al-abrir-se-queda-cargando-6721

Presentadas las herramientas, se definió que se iba a desarrollar para esta prueba de concepto.

En este caso, de acuerdo con el equipo, se decidió tomar como punto de partida el desarrollo de la consulta de información de una tarjeta para llevar a cabo esta prueba de concepto.

En el aplicativo actual, la consulta se ve así:

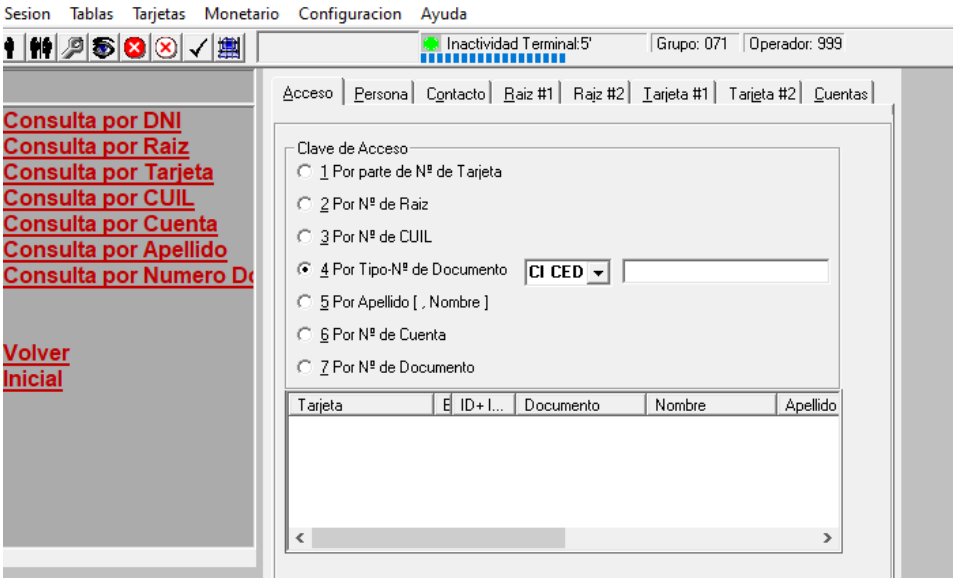


Imagen 6. Consulta de tarjeta de cliente actual.

Fuente: Recuperado del aplicativo administrador del ciclo de vida de las tarjetas.

La consulta de tarjetas se realiza mediante varios parámetros de búsqueda (documento, número de tarjeta, cuenta, entre otros).

El diseño propuesto para el nuevo cliente, acordado con el equipo de CX/UX, fue el siguiente prototipo:

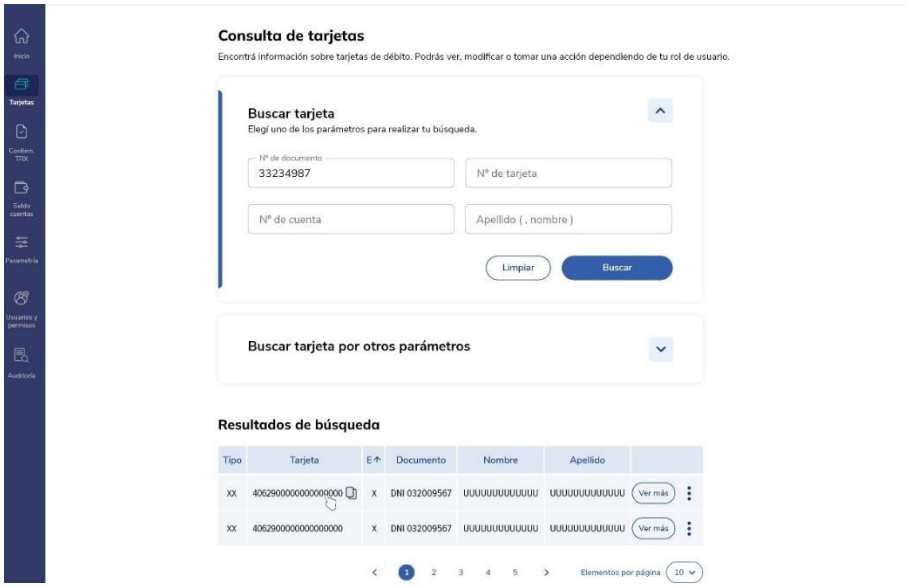


Imagen 7. Prototipo de consulta de tarjetas

Fuente: Recuperado de: https://app.zeplin.io/project/62d177596e05a41096949bc2/screen/63189f1b2fad7949b2362a8f

Definido el prototipo, se procedió a definir qué información recibiría el servicio como entrada y qué datos retornarían. Para esto, en comparación con el aplicativo actual, se estableció la siguiente historia de usuario:

Épica: Gestión de Tarjeta (ATD-003)

Nro. de historia: ATD-005

Como: desarrollador

Quiero: realizar la consulta de tarjetas mediante el número de documento, tarjeta, cuenta, apellido y nombre, CUIL y número de tarjeta raíz (esto representa los primeros 13 dígitos de una tarjeta).

Para: obtener toda la información necesaria sobre las mismas.

Criterios de aceptación:

- Los resultados de las búsquedas serán: tipo de tarjeta (02), número de tarjeta (19), estado de tarjeta (01), tipo de documento (03), número de documento (09), nombre (15), apellido (15), CUIL (12), fecha de alta de la tarjeta (10), cuenta primaria (19).
- Se listarán por defecto 10 resultados por página, dando la posibilidad al usuario de cambiar el valor a 20, 50 o 100.

En el caso del *frontend*, la historia de usuario fue la siguiente:

Épica: Gestión de Tarjeta (ATD-003)

Nro. de historia: ATD-006

Como: usuario del SOAT Web.

Quiero: realizar la consulta de tarjetas mediante el número de documento, tarjeta, cuenta, apellido y nombre, Cuil y número de tarjeta raíz (esto representa los primeros 13 dígitos de una tarjeta).

Para: obtener toda la información necesaria sobre las mismas.

Criterios de aceptación:

- El usuario deberá ingresar al menos un parámetro de búsqueda.
- El campo de número de documento deberá permitir ingresar tanto números como letras y hasta 9 caracteres. No deberá permitir ingresar caracteres especiales.
- El campo de número de tarjeta deberá permitir ingresar únicamente números y hasta 19 caracteres. No deberá permitir ingresar caracteres especiales.
- El campo de número de cuenta deberá permitir ingresar únicamente números y hasta 19 caracteres. No deberá permitir ingresar caracteres especiales.
- El campo de CUIL deberá permitir ingresar únicamente números y hasta 11 caracteres. No deberá permitir ingresar caracteres especiales.
- El campo número de tarjeta raíz deberá permitir ingresar únicamente números y hasta 19 caracteres. No deberá permitir ingresar caracteres especiales.
- Los resultados se mostrarán con paginado.

- Se listarán por defecto 10 resultados por página, dando la posibilidad al usuario de cambiar el valor a 20, 50 o 100.
- En caso de no encontrarse resultados, se deberá presentar un mensaje de error y se le solicitará que se modifiquen los parámetros de búsqueda.

Definido qué es lo que se va a desarrollar, se procedió a continuar con el cómo lo haríamos.

Para el *front*, se generó un proyecto nuevo desde Visual Studio Code, el cual se configuró utilizando la versión 13.0.0 de Angular y npm (Node Package Manager) como administrador de paquetes para Node.js, lo que nos permitió gestionar las dependencias necesarias y los *scripts* de construcción.

La estructura inicial del mismo fue la siguiente:

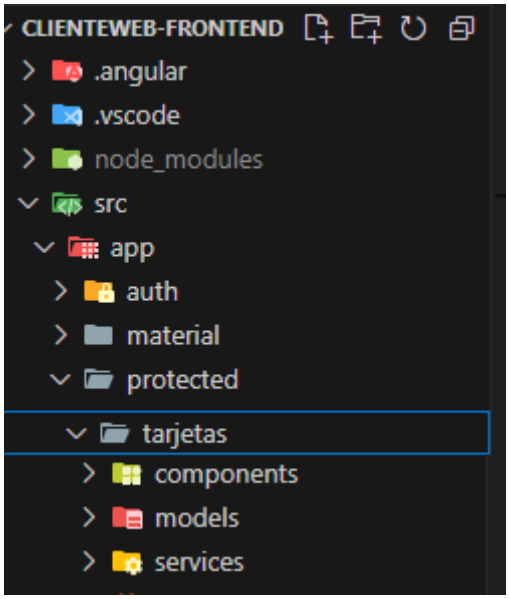


Imagen 8. Estructura del proyecto a nivel *front*.

Fuente: Elaboración propia, basada en la práctica.

Dentro de *components* se encuentra todo el desarrollo de lo referido a la pantalla de consulta de tarjeta, divido en tres componentes principales: tarjeta-consulta, resultado-tarjetas y cliente-*search*. Cada uno de estos componentes representa una parte del prototipo propuesto. Se avanzó

con el desarrollo de la sección *services* y en esta parte se estableció la llamada al servicio que consumirá del *backend*. En *models* se definió el modelo de datos que se utilizará: en este caso, por un lado, está el modelo de *request*, llamémosle *client.interface.ts*, que son los parámetros de búsqueda, y el otro modelo es el *response*, que representa los resultados que se retornarán de la consulta, denominado *tarjeta.interface.ts*.

Como se está trabajando con un proyecto Angular, hay otros directorios, archivos que se generaron por defecto, que no se detallarán.

En el caso del *backend*, la versión seleccionada fue Java 11. Se creó el proyecto utilizando Spring Boot y Spring Data lo cual nos facilitó la implementación de repositorios basados en JPA[8]. Y la estructura del mismo es la siguiente:

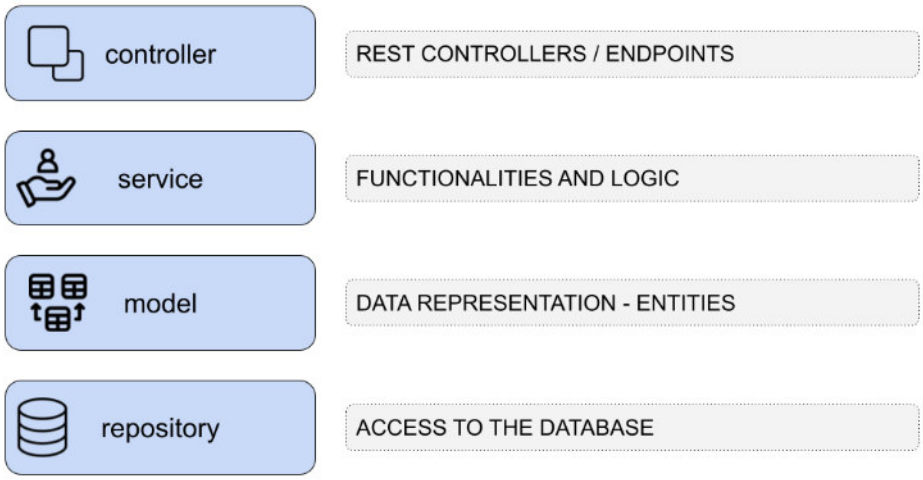


Imagen 9. Componentes del proyecto.

Fuente: Recuperado de https://gustavopeiretti.com/estructura-de-paquetes-spring-boot/

[8] JPA (Java Persistence API) es una especificación de Java que define una forma estándar de mapear objetos Java a bases de datos relacionales.

Una vez creado el proyecto y configurado el archivo de propiedades inicial, que incluye la configuración de la conexión a la base de datos, el *path* de la aplicación y el *path* de la documentación, se procedió con el desarrollo del servicio.

Como resultado de esta PoC, se obtuvo la siguiente solución inicial:

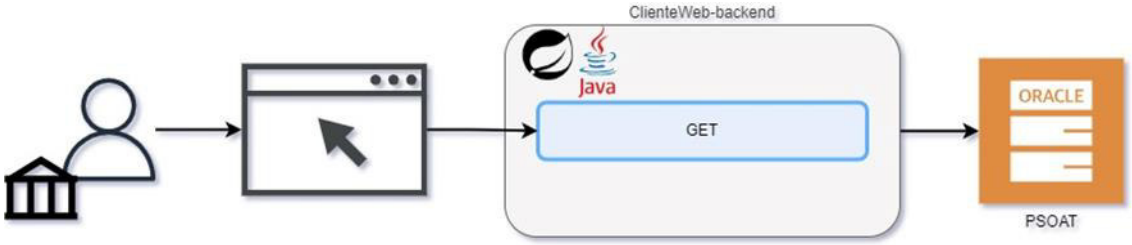


Imagen 10. Diagrama de solución.

Fuente: Elaboración propia, basada en la práctica.

La imagen 10 ilustra la arquitectura inicial de la PoC. A partir de esta prueba, se concluyó que las herramientas y tecnologías seleccionadas son totalmente compatibles entre sí, permitiendo una integración efectiva. Además, en términos de conocimiento y desarrollo técnico, se verificó que es factible llevar a cabo el proyecto.

Asimismo, se obtuvo una pantalla funcional que incluye las opciones de búsqueda, mostrando correctamente los datos consultados desde la base de datos. Esta solución cumple con los requisitos definidos en las historias de usuario tanto del *frontend* como del *backend*.

10.3 Diseño de la Base de Datos

Esta tarea apunta a la creación de la estructura de la base de datos que se requiere para el nuevo sistema, para garantizar una base de datos eficiente y escalable.

Para definir las tablas requeridas, se analizó la estructura existente y se determinó qué nuevas entidades eran necesarias crear. Por políticas internas de la empresa, solo se manejan bases de datos relacionales, ya que permiten organizar y relacionar datos de manera eficiente. Cada tabla

se compone de filas y columnas, y las relaciones entre ellas se establecen mediante claves primarias y foráneas, lo que facilita el uso de SQL (*Structured Query Language*) para realizar consultas complejas.

Nuestro sistema de gestión de bases de datos es Oracle, una plataforma ampliamente utilizada en el ámbito empresarial.

Antes de diseñar un nuevo esquema, fue necesario analizar el esquema actual para preservar los datos sensibles y mantener la trazabilidad del sistema que será reemplazado.

El proyecto se enfoca en la reingeniería del módulo consultivo, identificando las tablas involucradas. Las nuevas tablas se centran en la gestión de usuarios, roles[9] y permisos, con un prefijo "SOATWEB" para su identificación en el nuevo sistema.

Estas tablas permitirán, entre otras cosas, relacionar el alias del *Active Directory* [10](AD) con los operadores y gestionar roles, transacciones y categorías de usuarios.

Como resultado de esta tarea, se obtuvo el siguiente diagrama de entidad-relación (DER):

[9] Roles: se refiere al conjunto de permisos que se asignan a un usuario, determinando las acciones que puede realizar dentro del sistema.

[10] Active Directory (AD) es un servicio de directorio desarrollado por Microsoft para administrar redes de usuarios y recursos, facilitando la autenticación y el control de acceso en entornos empresariales.

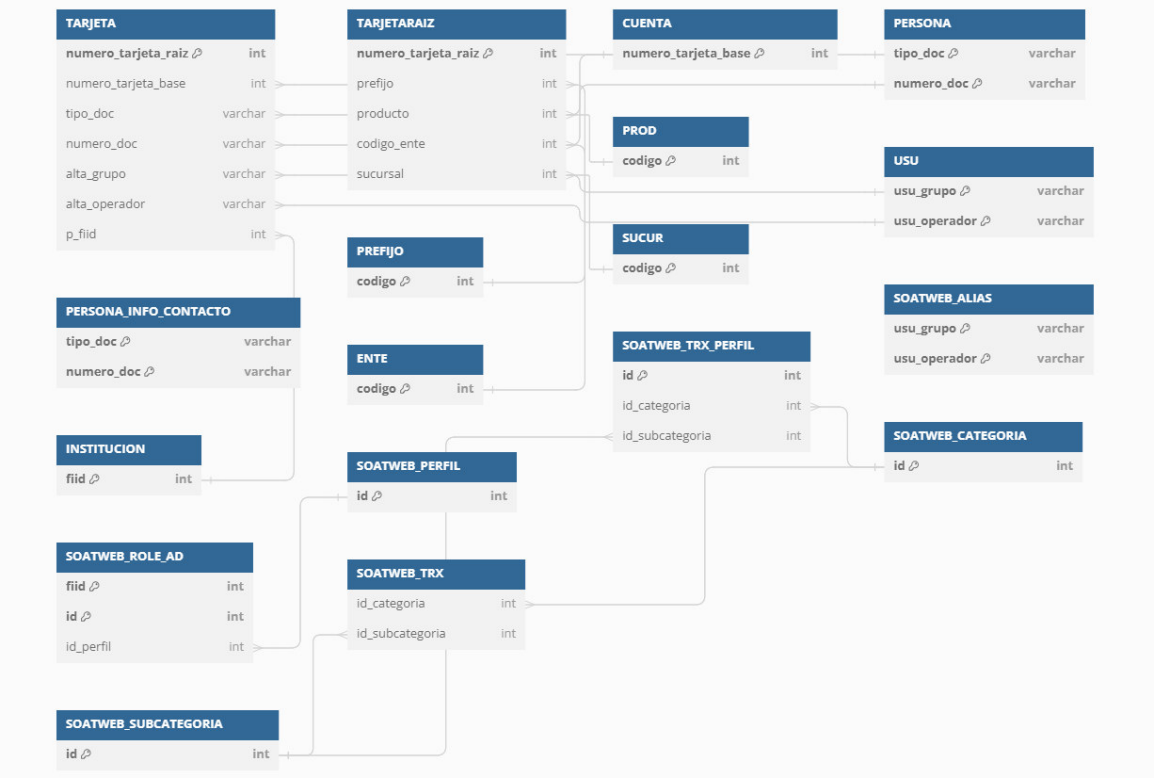


Imagen 11. Diagrama de Entidad-Relación.

Fuente: Elaboración propia, basada en la práctica.

Además, se observó que el diseño planteado quedó normalizado hasta la tercera forma normal (3NF), lo que minimiza las redundancias y asegura la integridad de los datos. Para tablas grandes, como las de tarjeta y persona, se implementó el particionamiento[11] por institución, lo que optimiza el rendimiento de bases de datos de gran volumen.

El plan de mantenimiento se decidió conservar como estaba, incorporando las nuevas tablas del SoatWeb al proceso de *backup* existente.

Esta tarea permitió obtener una visión más clara de la estructura de nuestra base de datos y cómo continuar trabajando para garantizar su eficiencia y escalabilidad.

[11] El particionamiento es una técnica utilizada en bases de datos para dividir grandes tablas en subconjuntos más pequeños, lo que mejora el rendimiento y facilita el manejo de los datos.

10.4 Análisis de Seguridad

La finalidad de esta tarea fue determinar los mecanismos de autorización y autenticación requeridos en el nuevo sistema, no solo por motivos de seguridad y robustez, sino también para cumplir con las normativas del Banco Central de la República Argentina (BCRA), dado que el sistema maneja el ciclo de vida de las tarjetas de débito.

En primera instancia, fue necesario investigar sobre los conceptos de autorización y autenticación, ya que, aunque parecen similares, cada uno abarca un aspecto diferente.

La autenticación verifica la identidad del usuario que intenta acceder al sistema, que ocurre cuando se loguea en nuestro sistema.

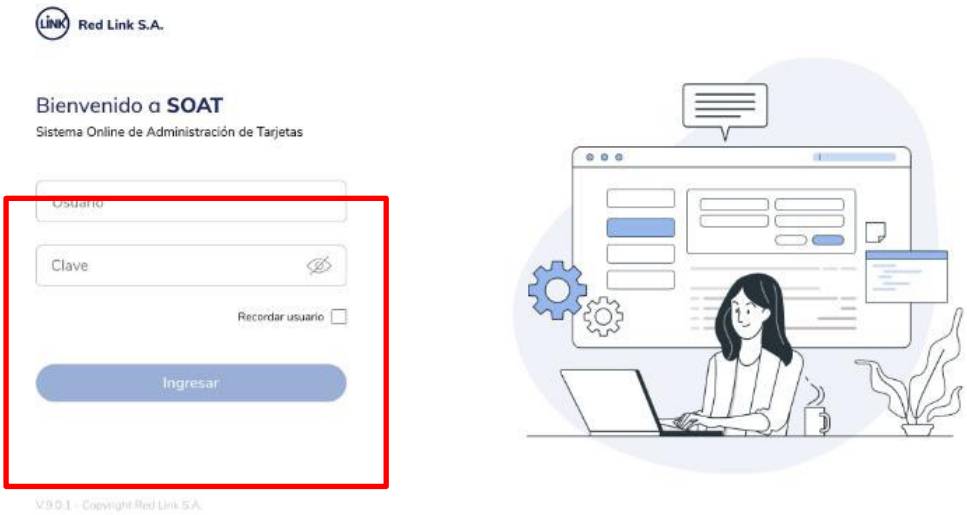


Imagen 12. Prototipo de pantalla de *login*.

Fuente: Recuperado de https://app.zeplin.io/project/62d177596e05a41096949bc2/screen/6321f61e5458f1a83acfc740

Este proceso de autenticación, a su vez, da paso a la autorización.

En el sistema actual, el *login* se realiza ingresando un grupo, un operador y una contraseña definida por el usuario. El sistema consulta la base de datos para verificar si los datos coinciden con los almacenados, y, de ser así, habilita las transacciones disponibles para el usuario. Sin

embargo, este enfoque necesita ser reemplazado por un sistema más seguro, escalable y fácil de usar.

Para el nuevo sistema, se decidió implementar un mecanismo de autenticación basado en tokens (JWT [12]) por varias razones: mejora la seguridad mediante cifrado, reduce la exposición de las credenciales y permite que las sesiones tengan un tiempo de vida limitado. Además, los tokens facilitan una arquitectura sin estado, lo que mejora la escalabilidad y permite integrarse con servicios de terceros mediante estándares como OAuth 2.0

En cuanto a la experiencia del usuario, el uso de tokens permite una autenticación más fluida, evita el ingreso repetitivo de credenciales y facilita la implementación de *Single Sign-On* (SSO). Los tokens también incluyen información personalizada sobre el usuario, lo que permite controlar el acceso sin necesidad de realizar consultas adicionales a la base de datos.

El nuevo sistema eliminará el uso de grupos y operadores, reemplazándolos por usuarios y contraseñas que estarán registrados en el *Active Directory* (AD) de la empresa.

El usuario debe estar registrado tanto en el AD como en nuestra base de datos, ya que muchos de nuestros clientes utilizan una estructura de grupo y operador. El rol asignado al usuario en el AD debe existir en nuestra base de datos para validar su acceso. Si las credenciales ingresadas coinciden con las almacenadas en el AD, se generará un JWT para la sesión del usuario.

Este JWT incluirá parámetros que permitirán validar si un usuario tiene acceso a ciertos recursos del sistema. Este proceso es conocido como autorización, ya que define las operaciones que puede realizar un usuario en función de sus permisos.

El flujo básicamente sería el siguiente:

[12] JSON Web Token (JWT) es un estándar abierto para intercambiar información de manera segura entre un cliente y un servidor, generalmente utilizado en sistemas de autenticación y autorización.

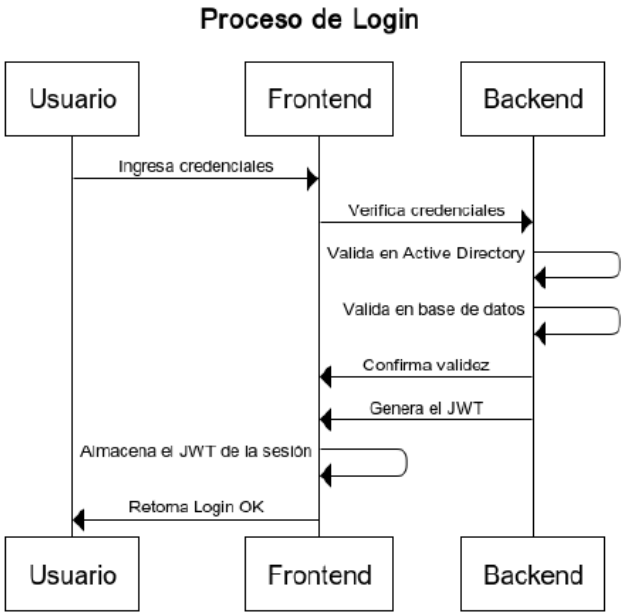


Imagen 13. Diagrama de secuencia de *login*.

Fuente: Elaboración propia, basada en la práctica.

Después de la autenticación, el sistema solicita al *backend* la lista de transacciones permitidas para el usuario según su rol. El *backend* consulta las tablas de la base de datos que relacionan roles y transacciones, y con esta información, adapta la interfaz para el usuario, habilitando o deshabilitando opciones según los permisos otorgados.

En cada operación posterior, el *frontend* incluye el JWT en las solicitudes al *backend*. Este último verifica que la operación solicitada esté permitida, y que los permisos del usuario sean los correctos según lo especificado en el JWT. Este flujo asegura que solo los usuarios autorizados puedan realizar ciertas acciones dentro del sistema.

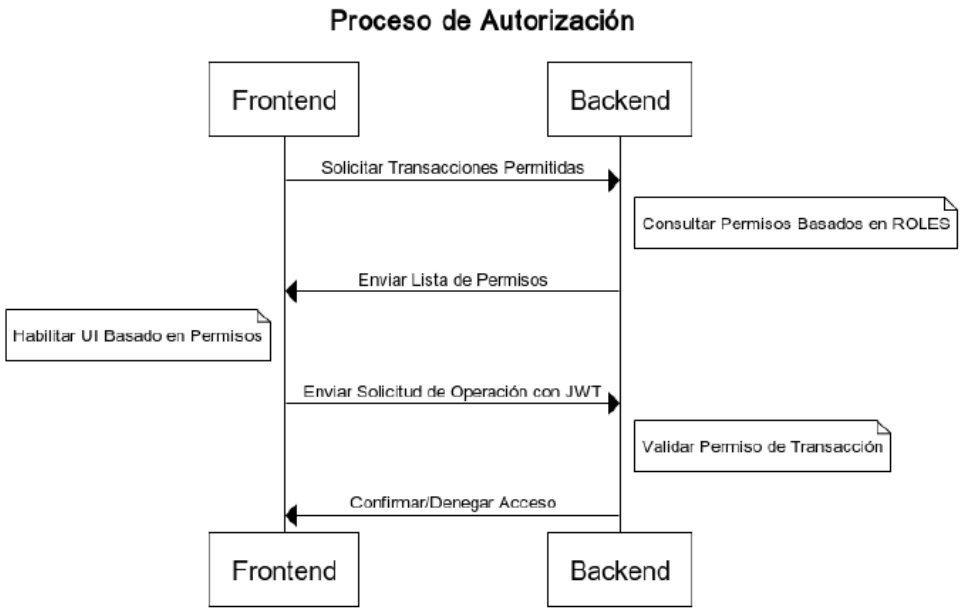


Imagen 14. Diagrama de secuencia del proceso de Autorización.

Fuente: Elaboración propia, basada en la práctica.

Para concluir esta tarea, se puede afirmar que se adquirió un entendimiento más profundo de los aspectos esenciales relacionados con la "seguridad en un sistema". Se logró delimitar claramente los alcances de la Autenticación y la Autorización, además de definir el flujo completo de ambos procesos.

Asimismo, se estableció que, según las normativas del BCRA, las sesiones deben tener un tiempo de inactividad máximo de 15 minutos, y las contraseñas deben renovarse cada 30 días, una política similar a la implementada en el Active Directory (AD). Las contraseñas también deben cumplir ciertos requisitos de seguridad.

Finalmente, fue necesario investigar el funcionamiento del AD para entender a fondo la gestión de usuarios, roles y contraseñas, lo cual resultó clave para la implementación de los nuevos mecanismos de Autenticación y Autorización.

10.5 Refinamiento del Módulo uno (Consultivo)

En este apartado, se explicará cómo se mejoraron y detallaron las especificaciones del Módulo uno antes de iniciar el desarrollo. Esto implicó revisar y clarificar toda la documentación y los requisitos del Módulo. El objetivo fue asegurarse que todas las características, funcionalidades y restricciones estuvieran claramente comprendidas y definidas antes de comenzar con el trabajo de desarrollo.

Para llevar a cabo esta tarea, denominada “refinamiento del primer módulo”, que incluyó funcionalidades generales como el inicio de sesión, “menú”, “gestión de tarjetas” y “configuración de la entidad”, se siguió una metodología ágil y estructurada. Los pasos seguidos fueron:

A. Desarrollo de prototipos

Contamos con un equipo de CX/UX que estuvo encargado de crear los prototipos de pantallas. Para ello, realizamos diversas reuniones en las que analizamos las pantallas del sistema anterior y revisamos cada una de las secciones. En particular, en la pantalla de detalle de tarjeta, fuimos seleccionando los campos correspondientes para cada pestaña.

1. Pestaña tarjeta:

Campo	**Descripción**
Tipo_Tarjeta (2)	Tipo de tarjeta
Fecha_Vto (05)	Fecha de vencimiento
Estado_Tarjeta (01)	Estado de la tarjeta
Alta_Fecha (10)	Fecha de alta de la tarjeta
Limite_Credito (02) - Importe (06)	Código de límite de crédito + Importe
Limite_Debito (02) - Importe (06)	Código de límite de débito + Importe
Numero_Miembro (01)	Indica si es adicional o titular
Numero_Version (01)	Número de versión de la tarjeta
Digito_Verificador (01)	Dígito verificador

Categoria_Comision (02)	Código de categoría de comisión asociada a la tarjeta
Vigencia (03)	Meses de vigencia
Nro_Cliente (12)	Número del cliente interno del banco
Alta_Grupo (03)	Grupo que dio de alta la tarjeta
Alta_Operador (03)	Operador que dio de alta la tarjeta
Alias	Alias que dio de alta la tarjeta
Alta_Fecha (08)	Fecha en la que se dio de alta la tarjeta
Alta_Hora (08)	Hora en la que se dio de alta la tarjeta
Numero_Tarjeta_Base (19)	Número de la tarjeta base
Cod_Ult_Denuncia (16)	Código de denuncia, emitido al bloquearse la tarjeta a través del sistema 0800
Grupo_Afinidad (04)	Código de grupo de afinidad asignado por Visa para tarjetas tipo Electron

2. Pestaña raíz:

Campo	**Descripción**	
Numero_Tarjeta_Raiz (19)	Número de Tarjeta Raíz	
Sucursal (04) - Descripción (30)	Código de Sucursal + Descripción	
Producto (04)	Agrupador de características de comportamiento	
Nro_Cliente (12)	Número del cliente interno del banco	
Estado_Raiz (01)	Estado de la Tarjeta Raíz	1 - Habilitada 9 - Raíz bloqueada ("Temporalmente") X - Dada de Baja
Tipo_Cuenta (02)	Tipo de Cuenta	

Numero_Cuenta (19)	Número de Cuenta
Apod_Tipo_Doc (03)	Apoderado - Tipo de Documento
Apod_Numero_Doc (09)	Apoderado - Número de Documento
Apod_Apellido (15)	Apoderado - Apellido
Apod_Nombre (15)	Apoderado - Nombre
Codigo_Ente (06)	Indica si la tarjeta pertenece a una cuenta que posee acreditamiento de haberes, indica el código con el cual se identifica a la empresa en el sistema
Cant_Miembro (02)	Cantidad de Adicionales
Tipo_Domic_Pin (01)	Domicilio PIN: P - Particular, L - Laboral, S - Sucursal
Tipo_Domic_Plas (01)	Domicilio Plástico: P - Particular, L - Laboral, S - Sucursal
Domicpart_Calle (45)	Calle del Domicilio Particular
Domicpart_Numero (05)	Número del Domicilio Particular
Domicpart_Piso (02)	Piso del Domicilio Particular
Domicpart_Depto (03)	Depto. del Domicilio Particular
Domicpart_Local (20)	Local del Domicilio Particular
Domicpart_Codpos (15)	Código Postal del Domicilio Particular
Domicpart_Codprov (02)	Código Provincia del Domicilio Particular
Domicpart_Telef (15)	Teléfono del Domicilio Particular
Domiclab_Calle (45)	Calle del Domicilio Laboral
Domiclab_Numero (05)	Número del Domicilio Laboral

Domiclab_Piso (02)	Piso del Domicilio Laboral
Domiclab_Depto (03)	Depto. del Domicilio Laboral
Domiclab_Local (20)	Local del Domicilio Laboral
Domiclab_Codpos (15)	Código Postal del Domicilio Laboral
Domiclab_Codprov (02)	Código Provincia del Domicilio Laboral
Domiclab_Telef (15)	Teléfono del Domicilio Laboral
Alta_Grupo (03)	Grupo que dio de alta la Raíz
Alta_Operador (03)	Operador que dio de alta la Raíz
Alias	Alias que dio de alta la Raíz
Alta_Fecha (08)	Fecha en la que se dio de alta la Raíz
Alta_Hora (08)	Hora en la que se dio de alta la Raíz

3. Pestaña PIL: en este punto del proyecto no se relevaron campos o datos a informar.
4. Pestaña cuentas:

Campo	**Descripción**
Tipo_Cuenta (02)	Tipo de cuenta
Numero_Cuenta (19)	Número de la cuenta
Estado_Cuenta (01)	Estado de la cuenta
Alta_Grupo (03)	Grupo que dio de alta la cuenta
Alta_Operador (03)	Operador que dio de alta la cuenta
Alta_Fecha (08)	Fecha en la que se dio de alta la cuenta
Alta_Hora (08)	Hora en la que se dio de alta la cuenta

5. Pestaña persona:

Campo	Descripción
Tipo_Doc (03)	Tipo de documento
Numero_Doc (09)	Número de documento
Apellido (15)	Apellido
Nombre (15)	Nombre
Cuil (12)	CUIL
Sexo (01)	M - Masculino, F - Femenino, X - Auto percibido
Profesion (20)	Valor informado por el banco
Fecha_Nacim (08)	Fecha de nacimiento
Estado_Civil (01)	S - Soltero/a, C - Casado/a, D - Divorciado/a, V - Viudo/a
Nacionalidad (15)	Nacionalidad
Alta_Grupo (03)	Grupo que dio de alta a la persona
Alias	Alias que dio de alta a la persona
Alta_Fecha (08)	Fecha en la que se dio de alta la persona
Alta_Hora (08)	Hora en la que se dio de alta la persona
Observaciones (60)	Campo opcional, algunos bancos lo utilizan
Segmento (25)	Segmento de facturación
Dir_Calle (60)	Calle del domicilio
Dir_Numero (10)	Altura del domicilio
Dir_Piso (02)	Piso del domicilio
Dir_Depto (03)	Departamento del domicilio
Dir_Provincia (30)	Provincia del domicilio
Dir_Localidad (40)	Localidad del domicilio
Tel_Personal_Cod_Area (04)	Código de área - Teléfono personal
Tel_Personal_Numero (10)	Número - Teléfono personal
Tel_Laboral_Cod_Area (04)	Código de área - Teléfono laboral
Tel_Laboral_Numero (10)	Número - Teléfono laboral

Tel_Laboral_Interno (05)	Interno - Teléfono laboral
Tel_Celular_Cod_Area (04)	Código de área - Celular
Tel_Celular_Numero (10)	Número - Celular
Email (100)	Email

En la sección “configuración de la entidad” se trabajó del mismo modo, se fueron viendo las diferentes pantallas y seleccionando que información mostrar.

6. Listado general de productos:

Campo	**Descripción**
Descripción	Descripción del producto
Embozado	Formato de embozado
Max_Cuentas	Máximo de cuentas
Max_Adicionales	Máximo de adicionales
Auto_Emision_Robo	Auto-emisión por robo
Auto_Renovacion	Auto-renovación

7. Detalle de productos:

Campo	**Descripción**
Cuentas	Lista de cuentas asociadas a dicho producto.
Tipo_Cuenta	Tipo de cuenta
Prefijos_Asociados	Lista de prefijos asociados a dicho producto.
Prefijo	Código de prefijo
Destino_Embozadora	Indica información de embozado
Grp_Prod	Grupo de productos
Embozadora	Embozadora
Cod_Emb_Tipo	Código del tipo de embozadora

8. Listado de prefijos:

Campo	Descripción
Codigo	Código de prefijo.
Tipo_Prefijo	Tipo de prefijo
Tipo_Numeracion	Tipo de numeración
Longitud_Pan	Longitud de pan
Vto_Default	Vencimiento default
Vigencia	Vigencia

9. Detalle de prefijos:

Campo	Descripción
Prefijo	Código de prefijo
Tipo_Prefijo	Tipo de prefijo
Tipo_Numeracion	Tipo de numeración
Longitud_Pan	Longitud de pan
Vto_Default	Vencimiento default
Vigencia	Vigencia
Producto_Asociados	Lista de productos asociados a dicho prefijo
Producto	Producto
Prod_Descripcion	Descripción de producto
Tipo_Tarjeta_Alta	Tipo de tarjeta asociada al alta
Vigencia	Vigencia
Tipo_Tarjeta_Emv	Tipo de tarjeta EMV
Check_Cvv	Chequeo de CVV
Cambio_Pin_Oblig	Cambio de PIN obligatorio
Primer_Uso_Atm	Primer uso ATM
Parametros_Pil	Lista de parámetros PIL asociados a los productos del prefijo consultado
Producto	Producto
Pil_Alta	Pil alta

Alta_Default	Valor default alta
Pil_Nv	Pil nueva versión
Nv_Default	Valor default nueva versión

ABM de usuarios y permisos

10. Lista general de usuarios:

Campo	**Descripción**
Nombre (22)	Nombre del usuario
Apellido (22)	Apellido del usuario
Usu_Grupo (03)	Grupo identificatorio del usuario
Usu_Operador (03)	Nro. de operador del usuario
Estado (01)	Estado del usuario
Ult_*Login* (15)	Fecha de último *login*

11. Detalle usuario:

Campo	**Descripción**
Nombre (22)	Nombre del usuario
Apellido (22)	Apellido del usuario
Alias	Alias del usuario
Usu_Grupo (03)	Grupo identificatorio del usuario
Usu_Operador (03)	Nro. de operador del usuario
Estado (01)	Estado del usuario
Rol_Usuario	Rol asignado al usuario

Entidades

12. Detalle entidad

Campo	**Descripción**
Reimp_Pines (01)	Reimprime pines
Reimp_X_Robo (01)	Reimprime por robo
Opera_Pil (01)	Opera con PIL

Tipo_Cuentas_De_La_Entidad	Tipos de cuenta de la entidad

Durante esta tarea, descubrimos que era necesario incluir también esta información, por lo que se agregó a la sección de configuración.

Entes

13. Listado general de entes:

Campo	**Descripción**
Tipo (01)	Tipo de ente
Codigo (06)	Código identificador del ente
Descripción (15)	Descripción del ente
Estado (01)	Estado del ente

14. Detalle ente:

Campo	**Descripción**
Tipo (01)	Tipo de ente
Codigo (06)	Código identificador del ente
Descripción (15)	Descripción del ente
Estado (01)	Estado del ente

Sucursales

15. Listado general de sucursales:

Campo	**Descripción**
Codigo	Código de la sucursal
Descripcion	Descripción de la sucursal
Centro_Pr	Centro de procesamiento
Casa_Visa	Código de Sucursal que Visa asignó
Casa_Anses	Código de Sucursal que Anses asignó
Estado	Estado de la sucursal

B. Refinamiento de historias de usuarios (UH)

Como el sistema fue dividido en cuatro funcionalidades, se procedió a crear las historias de usuarios correspondientes a cada una de las secciones: "gestión de tarjetas", "configuración de la entidad", "*login*" y "menú".

A modo de ejemplo, se detalla a continuación la historia de usuario ATD-007:

Título: Detalle de tarjetas - back

Épica: Gestión de tarjeta (ATD-003)

Nro. de historia: ATD-007

Como: Desarrollador

Quiero: Realizar la consulta de los datos de la Tarjeta, Cuenta, Persona, Contacto y Raíz.

Para: Poder presentarla por pantalla.

Criterios de aceptación:

- Los campos de cada pestaña serán los definidos en los puntos:
  1. Pestaña tarjeta
  2. Pestaña raíz
  3. Pestaña PIL
  4. Pestaña cuentas
  5. Pestaña persona

Otra historia relevante es la ATD-008, relacionada con el *frontend*. En esta historia, el usuario del SOAT Web puede seleccionar una tarjeta para ver más información. La pantalla cuenta con 6 pestañas con información sobre la Tarjeta, Cuenta, PIL, Datos Personales, Contacto y Raíz, pero la pestaña PIL solo figura como título en esta historia.

Ejemplo de la historia ATD-008.

Título: Detalle de tarjetas - *frontend*

Épica: Gestión de tarjeta (ATD-003)

Nro. de historia: ATD-008

Como: Usuario del SOAT Web.

Quiero: Seleccionar una tarjeta.

Para: Ver más información sobre la misma.

Criterios de aceptación:

- La pantalla deberá contar con 6 pestañas con distinta información sobre la Tarjeta, Cuenta, PIL, Datos Personales, Contacto y Raíz.
- La pestaña de PIL no estará contemplada en esta HU, solo figurará como título.
- Los campos de cada pestaña serán los mismo que en la ATD-007.

Diseño de la pantalla: ejemplo prototipo de cuentas:

Imagen 15. Prototipo de pantalla de cuentas.

Fuente: Recuperado de https://app.zeplin.io/project/62d177596e05a41096949bc2/screen/6321f61e5458f1a83acfc7340

C. Documentación

En este paso, una vez refinadas las historias, se acordó desarrollar un documento técnico-funcional para gestionar toda la información del proyecto. Este documento incluyó la estructura del proyecto, los flujos de determinadas funciones y, a medida que se fueron desarrollando los servicios y las pantallas del *frontend*, se fue añadiendo toda esa información. El objetivo era centralizar todos los datos en un solo documento de fácil acceso para cualquier integrante del equipo.

D. Aprobación y priorización

Para este punto, ya se contaba con las historias que componían el Módulo uno.
Se acordó cómo se manejaría la documentación y se determinó que era necesario definir un orden para el desarrollo de las historias. Para esto, se realizaron reuniones de planificación en las que participaron el equipo de desarrollo, los *testers*, el *Technical Owner* (TO), el *Product Owner* (PO) y el *Scrum Master* (SM), con el objetivo de priorizar las historias.
El refinamiento dio como resultado una lista de historias de usuario ordenadas según su prioridad, acordada por el equipo y los interesados en el proyecto.

Menú Front(ATD-021)
Consulta detalles Back (ATD-007)
Consulta Detalles Front (ATD-008)
Consulta de productos Back (ATD-009)
Consulta de productos Front (ATD-010)
Consulta de prefijos Back (ATD-011)
Consulta de Prefijos Front (ATD-012)
Consulta Usuarios Back (ATD-013)
Consulta Usuarios Front (ATD-014)
Consulta de Entidades y Cuentas Back (ATD-015)
Consulta de Entidades y Cuentas Front (ATD-016)
Consulta de Entes Back (ATD-017)
Consulta de Entes Front (ATD-018)
Consulta de Sucursales Back (ATD-019)
Consulta de Sucursales Front (ATD-020)
Login Back(ATD-022)
Login Front(ATD-023)

Imagen 16. Lista de historias priorizadas.
Fuente: Elaboración propia, basada en la práctica.

10.6 Desarrollo del Módulo Consultivo

El desarrollo del Módulo Consultivo representó un hito crucial en el proyecto de reingeniería del sistema de administración de tarjetas. Este módulo consolidó todas las actividades previas

de relevamiento, análisis y planificación, comenzando a generar un valor tangible para los clientes. Se estableció un plazo de dos *sprints* (un mes en total) para completar esta tarea, considerando que ya existía un proyecto base, la prueba de concepto (POC) desarrollada previamente como prueba en el punto 9.2 de este informe. A partir de las historias de usuario definidas en la imagen 17, se generó un mapa con una lista de tareas priorizadas que guio el desarrollo del sistema. Se decidió enfocar primero en el Módulo Consultivo, postergando el desarrollo de *login* y Autorización para una fase posterior debido a la complejidad que estos componentes representaban en este momento del proyecto.

El flujo de desarrollo se dividió en dos partes: *backend* y *frontend*. Cada sección del sistema fue desarrollada siguiendo su respectiva historia de usuario, garantizando que todas las funcionalidades se alinearan con las necesidades identificadas. Para fines prácticos de este informe, se detalló únicamente una tarea de *backend* y otra de *frontend* a modo de ejemplo, ya que el flujo de desarrollo fue el mismo para cada funcionalidad. Esto permitió ilustrar el proceso sin redundancias, facilitando la comprensión de las tareas clave de desarrollo tanto en el *backend* como en el *frontend*.

A continuación, se presentan ejemplos representativos del desarrollo de funcionalidades tanto en el *backend* como en el *frontend* del Módulo Consultivo, usando el detalle de tarjetas como referencia. Esta funcionalidad permitió acceder a información completa y en tiempo real sobre cada tarjeta, asegurando que el usuario pudiera visualizar la información de la misma.

Ejemplo de *Backend*: Consulta de Detalle de Tarjetas (ATD-007)

Antes de comenzar con el desarrollo del servicio, se definieron los nombres de los servicios y los parámetros de entrada que cada uno recibiría para retornar la información necesaria. Dado

que esta funcionalidad debía retornar detalles específicos para cada pestaña de la consulta, incluyendo información sobre la tarjeta, cuenta, persona, contacto y raíz, se definieron los siguientes *endpoints* [13]:

- Para tarjeta: tarjetas/detalleTarjeta
- Para cuenta: cuentas/detalleCuentas
- Para persona: personas/detallePersona
- Para contacto: contacto/detalleContacto
- Para raíz de tarjeta: tarjetasRaiz/detalleRaiz

Cada uno de estos *endpoints* devolvía como salida los parámetros definidos en la historia de usuario correspondiente. Para el desarrollo a nivel código, se siguió el flujo general adecuado para servicios REST. Sin embargo, dado que el flujo fue el mismo en cada *endpoint*, solo se detalló aquí el desarrollo correspondiente al detalle de tarjetas.

1. Controlador (*Controller*)

El controlador expuso el *endpoint* tarjetas/detalleTarjeta, que recibía solicitudes GET. Este método recibía los parámetros fiid y numeroTarjeta para devolver información específica de la tarjeta. Al recibir la solicitud del cliente, el controlador delegaba la lógica de negocio a la capa de servicio y, finalmente, retornaba la respuesta al cliente.

```
@GetMapping("/detalleTarjeta")
@Operation(summary = "Obtiene información detallada de una tarjeta especifica.")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Respuesta exitosa")})
public ResponseEntity<DetalleTarjetaResponse> getDetalleTarjeta(
        @RequestParam @NotBlank @Size(min = 4, max = 4, message = "El campo fiid debe tener una longitud de 4") String fiid,
        @RequestParam @NotBlank @Size(min = 16, max = 19)String numeroTarjeta, HttpServletRequest header) throws ResourceNotFoundException, TokenFailException
{
    String authorizationHeader = header.getHeader(HEADER);

    UsuarioTokenAuxDTO user = jwtTokenUtil.obtenerTokenAuth(authorizationHeader);
    return new ResponseEntity<DetalleTarjetaResponse>(tarjetaService.getTarjetaByNumeroTarjeta(fiid, numeroTarjeta, user) , HttpStatus.OK);
}
```

Imagen 17. Método de detalle de tarjeta.

Fuente: Elaboración propia, basada en la práctica.

[13] Un *endpoint* es una dirección específica (como una URL) en un servidor que permite acceder a un recurso o servicio ofrecido por una API o aplicación web.

2. Servicio (*Service*)

La capa de servicio contenía la lógica de negocio para la consulta de detalles de tarjeta. Esta se encargaba de orquestar las operaciones necesarias para cumplir con la solicitud, incluyendo validaciones y transformación de datos. En este caso, el servicio llamaba al repositorio para obtener el objeto tarjeta desde la base de datos y luego lo convertía en un DTO (Data Transfer Object) para enviarlo de vuelta al controlador.

```
@Override
public DetalleTarjetaResponse getTarjetaByNumeroTarjeta(String fiid, String numeroTarjeta, UsuarioTokenAuxDTO user) throws ResourceNotFoundException{

    String limiteDebito = "";
    String limiteCredito = "";

    Tarjeta tarjeta= tarjetaRepository.findByFiidAndNumeroTarjeta(fiid, HelperAutocompleteFields.formatNroTarjeta(numeroTarjeta))
            .orElseThrow(() -> new ResourceNotFoundException());

    String institucion = institucionRepository.findByFiid(fiid).toString();
    Map<String, String> limitDebCred = new HashMap<String, String>();

    String[] limites = institucion.split(",");
    for (int i=19;i<limites.length;i++) {
        String pair = limites[i];
        String[] keyValue = pair.split("=");
        limitDebCred.put(keyValue[0], keyValue[1]);

        String regex = "^0+(?! $)";
        limiteDebito= limitDebCred.getOrDefault(" atmclassDeb" + tarjeta.getLimiteDebito().replaceAll(regex, ""), "No disponible límite débito");
        limiteCredito= limitDebCred.getOrDefault(" atmclassCre" + tarjeta.getLimiteCredito().replaceAll(regex, ""), "No disponible límite crédito");

    }

    Alias alias = aliasRepository.findByFiidAndUsuGrupoAndUsuOperador(fiid, tarjeta.getAltaGrupo(), tarjeta.getAltaOperador()).orElse(null);

    loggerService.grabarLoggerConsutaTarjeta(CodigosExcepciones.CONS_TD_OK.codigo, user, numeroTarjeta);
    return new DetalleTarjetaResponse(tarjeta, limiteDebito, limiteCredito, alias);

}
```

Imagen 18. Servicio de detalle de tarjeta.

Fuente: Elaboración propia, basada en la práctica.

3. Repositorio (*Repository*)

El repositorio gestionaba la interacción directa con la base de datos, utilizando JPA (Java Persistence API) para realizar operaciones CRUD. *TarjetaRepository* extendía *JpaRepository*, lo que le permitía utilizar métodos predefinidos para interactuar con la base de datos, como *findById, save, delete*, etc. Para esta consulta, se implementó el método *findByFiidAndNumeroTarjeta*, que permitía obtener la tarjeta correspondiente basándose en el "fiid" y el "numeroTarjeta", generando automáticamente la consulta SQL.

```java
@Repository
public interface TarjetaRepository extends JpaRepository<Tarjeta, TarjetaPK>{

    Optional<Tarjeta> findByFiidAndNumeroTarjeta( String fiid, String numeroTarjeta);
```

Imagen 19. Método de consulta a la base de datos.

Fuente: Elaboración propia, basada en la práctica.

4. Base de Datos (*Database*)

Finalmente, la base de datos almacenaba la información persistente. Una vez realizada la consulta, la base de datos devolvía los datos solicitados al repositorio, que a su vez retornaba el objeto al servicio. El servicio procesaba los datos y los empaquetaba en un *response* que el controlador enviaba como respuesta al cliente.

```sql
SELECT * FROM tarjeta WHERE fiid = 'FI123' AND numero_tarjeta = '1234567890';
```

Imagen 20. Consulta que realiza el método a la base de datos.

Fuente: Elaboración propia, basada en la práctica.

Ejemplo de respuesta del servicio (código de salida 200 OK):


```
{
    "numeroTarjeta": "4998590000000105   ",
    "tipoTarjeta": {
        "id": "P",
        "nombre": "P - DEBITO PROPIETARIO"
    },
    "estadoTarjeta": {
        "id": "3",
        "nombre": "3 - BAJA POR ROBO"
    },
    "numUltDenuncia": "20100623-1504-30",
```

```
    "nroCliente": "000000000000",
    "altaFecha": "14/12/2009",
    "plasFechaEmision": "14/12/2009",
    "fechavto": "2019-12",
    "vigencia": "120",
    "limiteDebito": {
        "id": "03",
        "nombre": "03 - 001000"
    },
    "limiteCredito": {
        "id": "00",
        "nombre": "00 - No disponible límite crédito"
    },
    "categoriaComision": "02",
    "grupoAfinidad": "0000",
    "numeroTarjetaRaiz": "4998590000000105   ",
    "numeroMiembro": "0",
    "numeroVersion": "0",
    "digitoVerificador": "0",
    "altaAlias": " ",
    "altaGrupo": "999",
    "altaOperador": "003",
    "altaFechaHora": "14-12-2009 - 15.54 hs"
}
```


En caso de no encontrar resultados, el servicio retornaba un código 204 y, si algún parámetro no era válido, retornaba un 404. Este flujo se replicó en todos los servicios de consulta del sistema, manteniendo consistencia en el manejo de respuestas y códigos de estado.

Ejemplo de *Frontend*: Consulta de Detalle de Tarjetas (ATD-008)

En este apartado, se explicó cómo se realizó la integración del *frontend* con el *backend* en una aplicación Angular para consultar los detalles de tarjetas. El flujo seguido comprendió los siguientes pasos:

1. Creación del servicio en Angular

En Angular, se crearon servicios que proporcionan métodos para interactuar con el *backend*. Estos servicios se inyectaron en los componentes que los necesitaban para manejar la lógica de negocio o interactuar con la API REST[14].

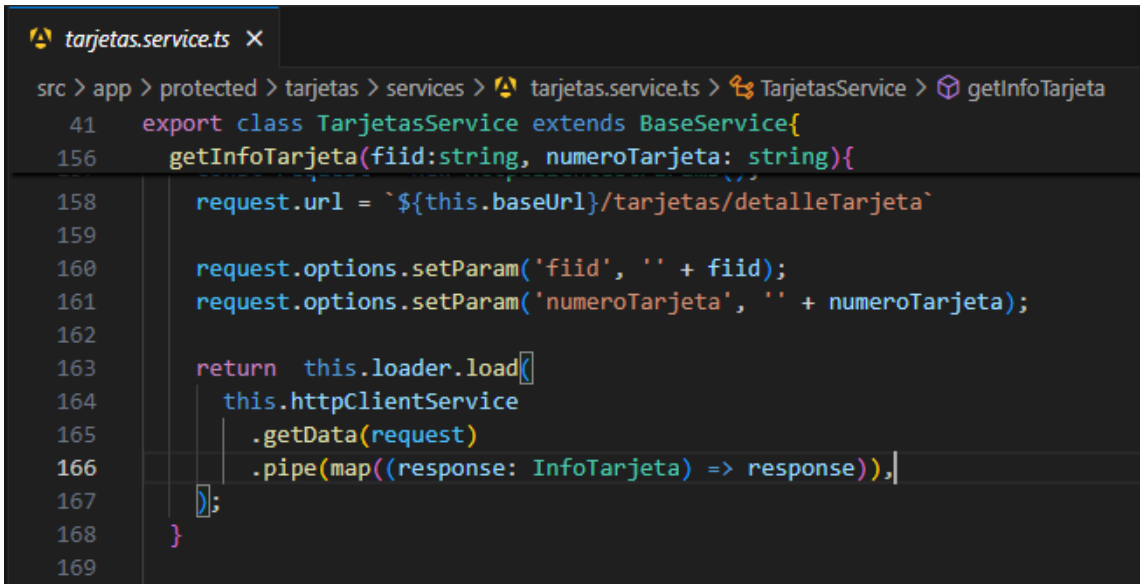


Imagen 21. Servicio de tarjeta.

Fuente: Elaboración propia, basada en la práctica.

El método *getInfoTarjeta* realizaba una solicitud GET al servicio REST.

La respuesta del servicio se transformaba en un objeto “InfoTarjeta”, cuya estructura era:

[14] Una API REST (por sus siglas en inglés, Representational State Transfer) es un estilo de arquitectura para diseñar servicios web que permite la comunicación entre diferentes sistemas a través de protocolos estándar como HTTP. Las APIs RESTful facilitan la interacción entre clientes y servidores al utilizar operaciones comunes y recursos identificados por URLs.

tarjetas.service.ts | info-tarjeta.interface.ts

src > app > protected > tarjetas > models > info-tarjeta.interface.ts > InfoTarjeta

```typescript
import { EstadoTarjeta } from "./estado-tarjeta.model";
import { InfoAlta } from "./info-alta.interface";
import { Limite } from "./limite.model";
import { TipoTarjeta } from "./tipo-tarjeta.model";

export interface InfoTarjeta extends InfoAlta{
    numeroTarjeta:     string;
    tipoTarjeta:       TipoTarjeta;
    estadoTarjeta:     EstadoTarjeta;
    numUltDenuncia:    string;
    nroCliente:        string;
    altaFecha:         string;
    plasFechaEmision:  string;
    fechaVto:          string;
    vigencia:          string;
    limiteDebito:      Limite;
    limiteCredito:     Limite;
    categoriaComision: string;
    grupoAfinidad:     string;
    numeroTarjetaRaiz: string;
    numeroMiembro:     string;
    numeroVersion:     string;
    digitoVerificador: string;
}

```

Imagen 22.Estructura del objeto InfoTarjeta.

Fuente: Elaboración propia, basada en la práctica.

2. Inyección del servicio en un componente

El servicio creado se inyectó en el componente que necesitaba consumir los datos. Esto se realizó a través del constructor del componente. En este caso, el componente *TarjetaComponent* utilizaba el servicio *TarjetaService* para obtener los datos al inicializarse.

Imagen 23. Constructor de TarjetaComponent.

Fuente: Elaboración propia, basada en la práctica.

En el método *ngOnInit*, se llamó a *obtenerInfoTarjeta* para obtener los datos cuando el componente se inicializaba.

3. Renderizado de la interfaz de usuario

Con los datos disponibles en el componente, Angular se encargó de renderizarlos en la vista utilizando el enlace de datos (*data binding*) en el archivo de plantilla tarjeta.component.html.

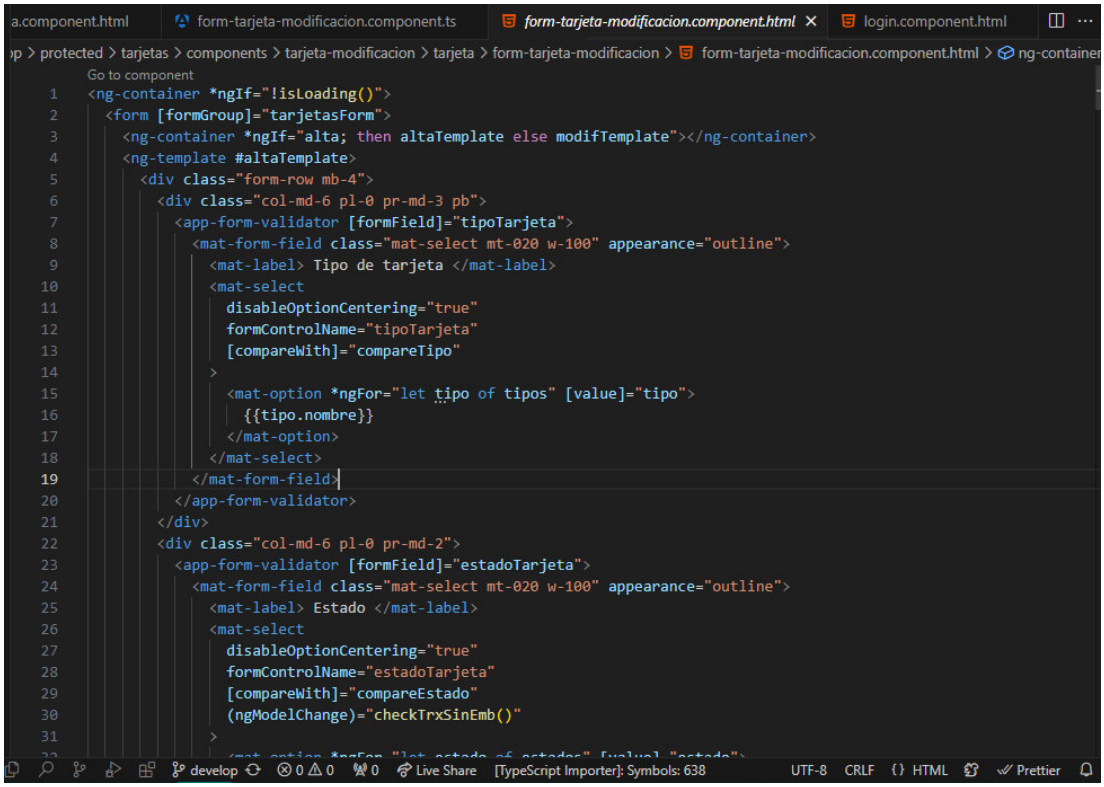


Imagen 24. Carga de datos en el componente de tarjeta.

Fuente: Elaboración propia, basada en la práctica.

Como resultado, se obtuvo la siguiente vista:

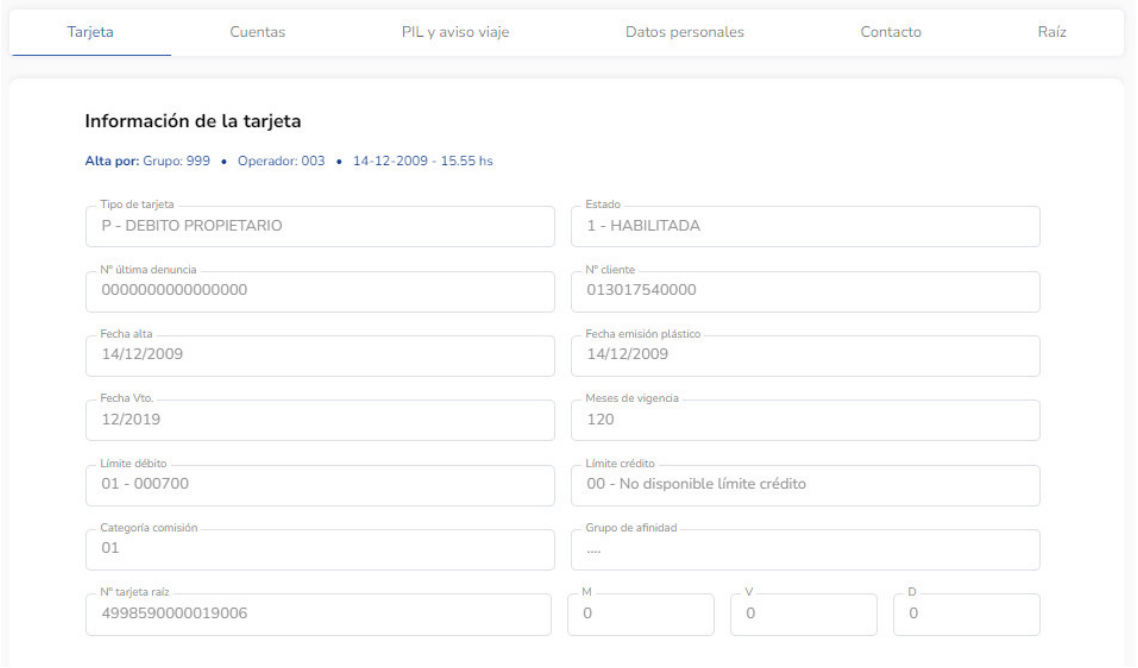


Imagen 25. Pestaña del detalle de tarjeta.

Fuente: Elaboración propia, basada en la práctica.

Como este mismo flujo se siguió para la integración de lo que es el detalle de cuenta, persona, contacto y raíz, simplemente se limitará a mostrar los resultados obtenidos a partir de la integración realizada.

La realización de esta tarea fue fundamental para el proyecto, ya que no solo marcó el inicio del desarrollo y la entrega de valor al cliente, sino que también demostró la viabilidad de las tareas previas. Este hito fue crucial, pues, aunque no se desarrollaron todas las tareas en este informe, el flujo de trabajo fue bastante claro y sencillo en relación con cómo se fue avanzando. Al finalizar este ítem, se concluyó con toda la parte consultiva relacionada con el aplicativo anterior.

Durante este proceso, se logró validar conceptos clave y superar desafíos técnicos que, en etapas iniciales, podrían haber complicado el desarrollo. Además, esta tarea sentó las bases para las siguientes fases, garantizando que el proyecto avanzara de manera sólida y alineada con los objetivos de los clientes.

10.7 Desarrollo del Sistema de *Login* y Seguridad

El desarrollo del sistema de *Login* y Seguridad implementó una solución robusta para garantizar un acceso seguro y controlado al sistema. Esta tarea implicó la creación de funcionalidades clave como la autenticación de usuarios, la autorización basada en roles y la gestión de contraseñas, incluyendo la recuperación y el cambio de las mismas. El objetivo fue establecer una barrera de seguridad eficaz para proteger los datos sensibles y mantener la integridad del sistema, asegurando que solo usuarios autorizados accedieran a las funcionalidades del mismo.

Para llevar a cabo las tareas de autenticación y autorización, se definió un flujo de funcionamiento que comprendía los siguientes pasos: en primer lugar, el usuario accedía a una pantalla de *login*, donde ingresaba su alias de *Active Directory* (AD) y su contraseña. Una vez que el *frontend* enviaba las credenciales al *backend*, estas eran validadas en el AD, obteniendo también el rol del usuario. El alias era verificado en la tabla SOATWEB_ALIAS, determinando su grupo y operador en el sistema. Además, el intento de *login* se registraba en la tabla LOGGER, y, en caso de éxito, se actualizaban la fecha y hora de último *login*, junto con el registro del rol asignado al usuario en ese momento. Finalmente, el *backend* generaba un token *JWT*, que el *frontend* almacenaba en la memoria del navegador para su uso en futuras solicitudes, permitiendo que las transacciones fueran permitidas y mostradas al usuario según su rol.

Para iniciar el desarrollo en el *backend*, se implementó un flujo completo siguiendo los principios de un servicio *REST*. El cliente enviaba una solicitud *HTTP* al controlador, el cual expuso el *endpoint* auth/*login*.

```
@PostMapping("/login")
public ResponseEntity<UserAuthenticationResponse> login(@RequestBody @Valid UserAuthenticationRequest userAuthenticationRequest) throws Exception {

    UserAuthenticationResponse res = new UserAuthenticationResponse();

    res = authenticationService.authenticate(userAuthenticationRequest.getUser(), userAuthenticationRequest.getPassword());
    return new ResponseEntity<UserAuthenticationResponse>(res,HttpStatus.OK);
}
```

Imagen 26. Controlador de la clase autenticación.

Fuente: Elaboración propia, basada en la práctica.

El método POST *login* recibía como parámetros de entrada *user* y *password*. Este controlador delegaba la solicitud al servicio de autenticación, llamado *authenticacionService*, el cual ejecutaba la lógica de negocio mediante el método *authenticate.*

```
public interface AuthenticationService {

    public UserAuthenticationResponse authenticate( String username, final String password) throws Exception;

    public UserLoginResponse searchInfoUser( String username) throws PermisoFailException;

    public void logout( String username) throws Exception;

    public void modificarPassword(UserModifPass requestUsuario) throws InvalidAttributeValueException, OperationNotSupportedExcep
}
```

Imagen 27. Interface de servicio de autenticación.

Fuente: Elaboración propia, basada en la práctica.

En este paso, el servicio validaba en primer lugar la existencia del usuario tanto en el *Active Directory*(AD) como en la base de datos a través del método *preValidateUser.*

```
//Valido que el usuario exista en AD y en BD
private Alias preValidateUser(String username) throws PermisoFailException {

    Alias alias = aliasRepository.findByAliasIgnoreCase(username).orElse(null);

    if(alias == null) {
        throw new PermisoFailException(CodigosExcepciones.ALIAS_NO_BD.codigo, MessageFormat
    }

    return alias;
```

Imagen 28. Método de validación de existencia de usuario a nivel base de datos.

Fuente: Elaboración propia, basada en la práctica.

Si el usuario existía, el servicio autenticaba las credenciales username y password utilizando LdapTemplate para interactuar con el directorio LDAP. Esto incluía la búsqueda del usuario en el directorio LDAP y, en caso de autenticación exitosa, continuaba con la verificación de roles. El sistema comprobaba que el usuario tuviera un único rol asignado, el cual debía estar registrado en la tabla PSOAT y en estado habilitado.

```
//Valido roles,habilitado
private UserLoginResponse postValidateUser(String username, Alias alias) throws PermisoFailException {

    UserLoginResponse user = searchInfoUser(username);

    if(user.getRolesAsignados().size() == 0) {
        throw new PermisoFailException(CodigosExcepciones.USER_SIN_ROL.codigo, MessageFormat.format(CodigosExcepciones.US
    }
    else if(user.getRolesAsignados().size() > 1) {
        throw new PermisoFailException(CodigosExcepciones.USER_MAS_DE_UN_ROL.codigo, MessageFormat.format(CodigosExcepcic
    }
    //Valido que el rol exista en BD
    else if(!roleRepository.existsRoleByRole(user.getRolesAsignados().get(0)))
    {
        throw new PermisoFailException(CodigosExcepciones.ROLE_INEXISTENTE_SOAT.codigo, MessageFormat.format(CodigosExcep
    }
    //Valido que el grupo y operador este habilitado
    else if(!alias.getUsuario().getEstado().equals("1"))
    {
        throw new PermisoFailException(CodigosExcepciones.USER_NO_HABILITADO.codigo, MessageFormat.format(CodigosExcepcic
    }

    return user;
}
```

Imagen 29. Validaciones de usuario.

Fuente: Elaboración propia, basada en la práctica.

Tras las validaciones, el servicio generaba un token JWT mediante el método generateToken-*Login*.

```
public String generateTokenLogin(UsuarioTokenAuxDTO token) {
    return Jwts.builder()
            .setHeaderParam(JwsHeader.ALGORITHM, "HS256")
            .setHeaderParam(JwsHeader.TYPE, "JWT")
            .setHeaderParam(JwsHeader.KEY_ID,jwtRefreshSecret)
            .setSubject(objectToJson(token)).setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))
            .signWith(SignatureAlgorithm.HS256, jwtSecret.getBytes(StandardCharsets.UTF_8)).compact();
}
```

Imagen 30. Método de generación de token.

Fuente: Elaboración propia, basada en la práctica.

Este token contenía tres partes codificadas: encabezado, carga útil y firma, utilizando el algoritmo HS256 [15] y estableciendo tiempos de emisión y expiración (quince minutos después de

[15] HS256 (HMAC-SHA256) es un algoritmo de firma simétrica utilizado comúnmente para generar tokens seguros, como los JWT (JSON Web Tokens). Utiliza la función de hash SHA-256 combinada con una clave secreta compartida para crear un HMAC (código de autenticación de mensaje basado en hash), que garantiza la integridad y autenticidad del token. Al ser un método de clave simétrica, tanto la generación como la verificación de la firma requieren la misma clave secreta.

la creación) para asegurar la integridad del mismo. El servicio también realizaba una actualización en la base de datos, guardando la fecha y hora del último *login*. En caso de errores durante el proceso, se definieron capturas de excepciones para cada posible escenario, y el sistema retornaba el código de estado adecuado según el tipo de error detectado.

```
}catch(AuthenticationException e){
    logger.error("Error autenticación", e.getMessage());
    throw new AuthenticationFailException(null,e.getMessage(), username);

}catch(PermisoFailException e){
    if(!(("107").equals(e.getCodigo()) || ("109").equals(e.getCodigo())) ) {
        loggerService.grabarLoggerLogin(e.getCodigo(), username);
    }
    logger.error("Error permisos", e.getMessage());
    throw new PermisoFailException(e.getCodigo(), e.getMessage(), username);

}
return response;
```

Imagen 31. Mapeo de excepciones.

Fuente: Elaboración propia, basada en la práctica.

Finalizado el proceso de *login* en el *backend*, se agregó un filtro de Spring Security que exigía el uso del token JWT en todas las futuras solicitudes, exceptuando algunos servicios específicos como *login*, *logout* y los correspondientes a *Swagger*[16]. Este filtro aseguró que el token actuara como factor de autorización para las interacciones entre *frontend* y *backend*, proporcionando una capa adicional de seguridad.

El desarrollo del *frontend* comenzó con la creación del servicio *auth.service.ts* en Angular, cuyo método *login* realizaba una solicitud POST al servicio REST de *backend*.

[16] Swagger es un conjunto de herramientas de software que ayudan a diseñar, construir, documentar y consumir servicios web RESTful. Es especialmente útil para crear documentación interactiva y visual para APIs, lo que facilita la comprensión y el uso de las mismas.

```
login(request: LoginRequest): Observable<LoginResponse>{
  const httpClientPostParams: HttpClientPostParams =
    new HttpClientPostParams();

  httpClientPostParams.url = environment.baseUrl + '/auth/login';
  httpClientPostParams.body = request;

  return this.loader
    .load(this.httpClientService.postData(httpClientPostParams))
    .pipe(
      map((response: LoginResponse) => {
        return response;
      }),
    );
```

Imagen 32. Servicio de autenticación, método de *login*.

Fuente: Elaboración propia, basada en la práctica.

La respuesta del servicio se convertía en un objeto *LoginResponse*, que contenía los datos devueltos. Posteriormente, se integró este servicio en el componente *LoginComponent*, el cual utilizaba el método *ngOnInit* para redirigir al usuario según su estado de autenticación y, en caso de que el usuario no estuviera autenticado, se iniciaba el formulario de inicio de sesión definido en *login*.component.html.

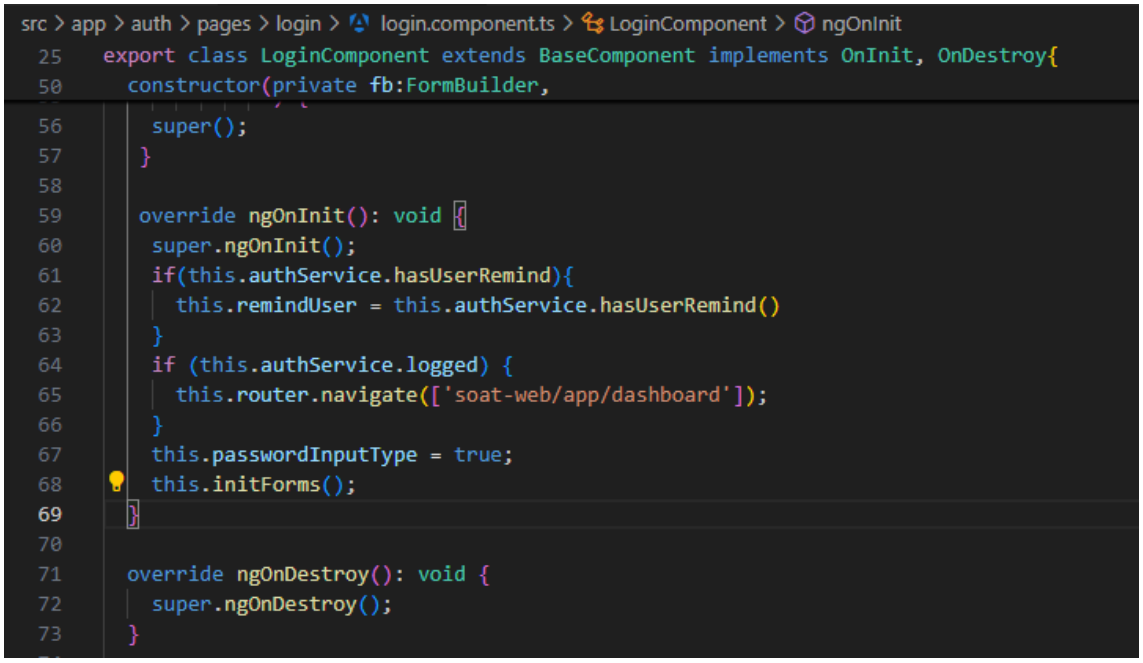


Imagen 33. Constructor de LoginComponent.

Fuente: Elaboración propia, basada en la práctica.

Al completar el formulario, el evento *doLogin ()* validaba los campos e invocaba al servicio de *login*. En caso de éxito, el inicio de sesión se almacenaba en el *local storage* del navegador, incluyendo el menú autorizado y el panel de control para el usuario.

```
    <p class="subtitle mb-0">
      Sistema Online de Administración de Tarjetas
    </p>
  </div>
  <br>
  <form
    [formGroup]="userCredentialsForm"
    class="login mx-auto mx-lg-0"
    (ngSubmit)="doLogin()"
    (keydown.enter)="doLogin()"
  >

  <app-form-validator [formField]="user">
    <div class="form-group position-relative mb-1">
      <input
        required
        type="text"
        placeholder=""
        class="form-control bg-transparent selector-for-some-widget px-3 py-2"
        formControlName="user"
        autofocus
        [ngClass]="{
          focused: userCredentialsForm.controls['user'].value !== ''
        }"
      />
      <label class="col-form-label">Usuario</label>
```

Imagen 34. Parte del componente html del *login*.

Fuente: Elaboración propia, basada en la práctica.

Luego de iniciar sesión, el *frontend* desplegaba un "menú" personalizado según los permisos del usuario, y el token almacenado en el *local storage* se utilizaba en futuras consultas como parámetro de autorización, lo que aseguraba que las interacciones estuvieran restringidas según el rol del usuario.

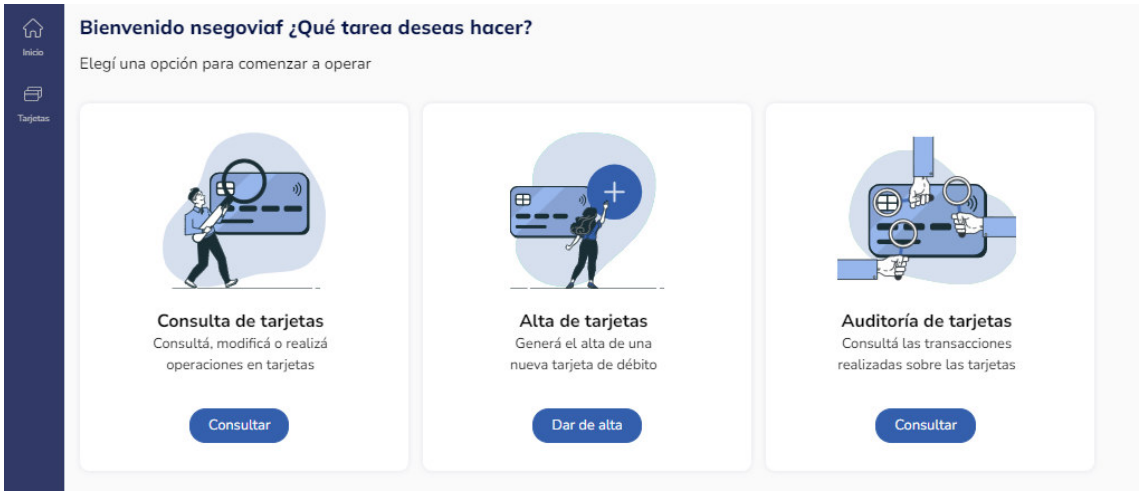


Imagen 35. Panel de inicio.

Fuente: Elaboración propia, basada en la práctica.

Para dimensionar los resultados obtenidos de la ejecución de las tareas desarrolladas en los puntos anteriores, se identificaron las siguientes mejoras incorporadas al proyecto:

Autenticación segura

- Se implementó un sistema de autenticación robusto basado en tokens JWT, que aseguró las sesiones de usuario y garantizó que las solicitudes a servicios protegidos fueran realizadas únicamente por usuarios autenticados y autorizados.

Validación y autorización

- Se integró la validación de credenciales contra el sistema *Active Directory* (AD) para usuarios corporativos, lo que facilitó la gestión centralizada de identidades y permisos.
- Se desarrolló un sistema de autorización basado en roles, asegurando que los usuarios accedieran solo a los recursos y funcionalidades para los cuales estaban autorizados.

Gestión de Sesiones sin Estado

- El *backend* fue diseñado para ser sin estado (*stateless*) mediante el uso de tokens JWT, permitiendo una mayor escalabilidad y reduciendo la dependencia de la persistencia de sesiones en el servidor.

Registro de Actividades y Seguridad

- Se añadieron funcionalidades de *login* para registrar intentos de acceso, así como éxitos o fallos en la autenticación, lo que facilitó el monitoreo y la auditoría de seguridad.

Interfaz de Usuario Intuitiva

- Se desarrolló una interfaz de inicio de sesión amigable y fácil de usar, que guiaba al usuario a través del proceso de autenticación de manera clara y eficiente.
- Se implementaron mecanismos de retroalimentación visual para informar al usuario sobre el estado de su autenticación, como errores de credenciales o éxito en el inicio de sesión.

Redirección Inteligente

- Se incluyó una lógica de redirección automática que llevaba a los usuarios autenticados a sus respectivas áreas, optimizando así su experiencia de navegación.

Gestión de Estados y Formularios

- Los formularios de inicio de sesión se inicializaron de manera efectiva, con validaciones en tiempo real que aseguraban que los datos ingresados por el usuario fueran correctos antes de enviarse al *backend*.
- Se agregó soporte para recordatorios de usuario, permitiendo que el sistema recordara las credenciales si el usuario lo solicitaba.

Seguridad en la Interfaz

- Se implementaron medidas de seguridad en el *frontend*, como la protección contra XSS (*Cross-Site Scripting*) y la configuración de CSP (*Content Security Policy*), asegurando un manejo seguro de las credenciales de usuario.

El desarrollo de este ítem resultó en un proceso de autenticación y autorización seguro, eficiente y escalable, mejorando tanto la seguridad como la experiencia de usuario.

10.8 Gestión de usuarios

Se creó esta tarea debido a que, durante la configuración del flujo de *login*, se identificó la necesidad de desarrollar un ABM de usuarios. Esto resultó esencial tanto para el equipo de seguridad como para el nuestro, ya que permitió otorgar acceso a nuevos usuarios y vincular a aquellos previamente existentes que ya contaban con un alias en el *Active Directory* (AD) y operaban con el sistema anterior. Esta funcionalidad también delegó el control de usuarios administradores a cada entidad, permitiendo que gestionaran sus propios usuarios y evitando así la necesidad de realizar una carga inicial mediante *scripts* en la base de datos.

Se desarrollaron dos servicios y la pantalla correspondiente.

Antes de avanzar con el desarrollo, se generaron las historias de usuario:

Título: Alta y modificación de usuarios – *Backend*

Épica: Configuración de la entidad (ATD-004)

Nro. de historia: ATD-024:

Como: Usuario del SOAT Web.
Quiero: Agregar y modificar usuarios.
Para: Poder gestionar el manejo de usuarios.

Criterios de aceptación:

- Para altas:
    1. Validar la existencia del alias en el AD; en caso de no encontrarse, devolver error.
    2. Verificar que el alias tenga un rol SOAT asignado en el AD; si carece de uno o tiene más de uno, devolver error.
    3. Si el alias existe y tiene un rol asignado, devolver: nombre, apellido y rol.
    4. En caso de confirmarse el alta y no poder asignar grupo/operador, devolver error.
- Para modificaciones:

1. Permitir únicamente la modificación del estado del usuario (habilitado o deshabilitado).

Título: Alta y modificación de usuarios – *Frontend*

Épica: Configuración de la entidad (ATD-004)

Nro. de historia: ATD-025

Como: Usuario del SOAT Web.

Quiero: Agregar y modificar usuarios.

Para: Poder gestionar el manejo de usuarios.

Criterios de aceptación:

- Habilitar esta opción solo para usuarios "Administradores" de cada entidad.
- Validar que el alias sea válido y tenga roles SOAT asignados en el AD.
- Mostrar mensajes de error específicos si el alias no existe en el AD, si carece de un rol SOAT, o si tiene más de un rol asignado.
- Al superar todas las validaciones, mostrar en pantalla el nombre, apellido y rol.
- Implementar mensajes de confirmación y de error según los casos de confirmación o cancelación de altas.

Definidas las historias de usuario, se procedió con el desarrollo de los servicios en el *backend*.

Para la creación de los servicios de alta y modificación, se definieron los siguientes *endpoints*:

- /usuarios/agregarUsuario
- /usuarios/modificarUsuario

Para el servicio de alta de usuarios, el controlador creado manejaba una solicitud POST en el endpoint agregarUsuario, que recibía los campos definidos en el *request*.

```
@PostMapping("/agregarUsuario")
public ResponseEntity<DetalleUsuarioResponse> agregarUsuario(@RequestHeader HeaderFiid fiid, @RequestBody UsuarioRequest requ
    String authorizationHeader = header.getHeader(HEADER);

    UsuarioTokenAuxDTO usuario = jwtTokenUtil.obtenerTokenAuth(authorizationHeader);

    return new ResponseEntity<DetalleUsuarioResponse>(usuarioService.agregarUsuario(fiid,requestUsuario,usuario),HttpStatus.
}
```

Imagen 36. Método para agregar usuario.

Fuente: Elaboración propia, basada en la práctica.

El servicio de alta utilizaba la interfaz *usuarioService*, cuyo método *agregarUsuario* generaba el alta de un usuario. Si se trataba de un usuario nuevo sin grupo y operador asignado, se le asignaban valores nuevos y el servicio realizaba la inserción en la tabla de usuarios.

```
//Caso en que se agrega en la tbl alias y usu -> Usuario nuevo
if(request.getOperador()== null) {

    if(request.getGrupo() != null) {
        seq =  usuarioRepository.getMaxUsuarioGrupoOperadorPorGrupo(fiidFormat, request.getGrupo()).orElse("500001");
    }
    else {
        seq =  usuarioRepository.getMaxUsuarioGrupoOperador(fiidFormat).orElse("500001");
    }
    usuario.setUsuarioPK(new UsuarioPK(fiid.getFiid(),seq.substring(0, 3), seq.substring(3,6)));
    usuario.setNombre(request.getNombre());
    usuario.setApellido(request.getApellido());
    usuario.setEstado("1");
    usuario.setAltaFecha(fecha);
    usuario.setAltaHora(hora);
    usuario.setAltaUsuGrupo(request.getAltaGrupo());
    usuario.setAltaUsuOperador(request.getAltaOperador());
    usuario.setAudFecha(fecha);
    usuario.setAudHora(hora);
    usuario.setAudUsuGrupo(request.getAltaGrupo());
    usuario.setAudUsuOperador(request.getAltaOperador());

    usuario = usuarioRepository.save(usuario);

    if(usuario != null) {
        Alias aliasN = new Alias();
        aliasN.setAlias(request.getAlias());
        aliasN.setFiid(fiid.getFiid());
        aliasN.setUsuGrupo(usuario.getUsuarioPK().getUsuGrupo());
        aliasN.setUsuOperador(usuario.getUsuarioPK().getUsuOperador());
        aliasN.setLogin("N");
        alias = aliasRepository.save(aliasN);
    }
```

Imagen 37. Servicio para agregar usuario, caso de usuario nuevo.

Fuente: Elaboración propia, basada en la práctica.

En caso de que el usuario ya contara con un grupo y operador del sistema anterior, se añadía en la tabla de alias y se actualizaban sus datos en la tabla de usuarios, sincronizando la información de la base de datos con la del *Active Directory(*AD).

```
    else {
        Alias aliasN = new Alias();
        aliasN.setAlias(request.getAlias());
        aliasN.setFiid(fiid.getFiid());
        aliasN.setUsuGrupo(request.getGrupo());
        aliasN.setUsuOperador(request.getOperador());
        aliasN.setLogin("N");
        alias = aliasRepository.save(aliasN);

        Usuario usuarioModif = usuarioRepository
                .findByUsuarioPK_pFiidAndUsuarioPK_usuGrupoAndUsuarioPK_usuOperador(fiidFormat, request.getGrupo(), request.g
        usuarioModif.setNombre(request.getNombre());
        usuarioModif.setApellido(request.getApellido());
        usuarioModif.setAudUsuGrupo(request.getAltaGrupo());
        usuarioModif.setAudUsuOperador(request.getAltaOperador());
        usuarioModif.setAudFecha(fecha);
        usuarioModif.setAudHora(hora);
        usuario = usuarioRepository.save(usuarioModif);


    }
    loggerService.grabarLoggerAltaUsuario(CodigosExcepciones.ALTA_USU_OK.codigo, token, usuario, alias);
    return new DetalleUsuarioResponse(alias, usuario);
}
```

Imagen 38. Servicio para agregar usuario, caso de usuario existente.

Fuente: Elaboración propia, basada en la práctica.

El repositorio UsuarioRepository extendía *JpaRepository*, permitiendo el uso de métodos como *save* y *findBy* para las búsquedas y operaciones de inserción en la base de datos. Para los usuarios nuevos, se definieron métodos específicos que asignaban el grupo y operador.

```
if(request.getGrupo() != null) {
    seq =  usuarioRepository.getMaxUsuarioGrupoOperadorPorGrupo(fiidFormat, request.getGrupo()).orElse("500001");
}
else {
    seq =  usuarioRepository.getMaxUsuarioGrupoOperador(fiidFormat).orElse("500001");
}
```

Imagen 39. Métodos de consulta de grupo y operador.

Fuente: Elaboración propia, basada en la práctica.

El repositorio traducía el método *save* en una operación de inserción (*insert*) en la base de datos, retornando el objeto insertado.

En el caso de modificaciones de usuario, el controlador recibía una solicitud PUT a través del *endpoint* modificarUsuario, con los campos Fiid, Alias, y Estado.

```
@PutMapping("/modificarUsuario")
@ResponseStatus(value = HttpStatus.NO_CONTENT)
public void modificarUsuario(@RequestHeader HeaderFiid fiid, @RequestBody UsuarioModifRequest requestUsuario, HttpServletRe
    String authorizationHeader = header.getHeader(HEADER);

    UsuarioTokenAuxDTO usuario = jwtTokenUtil.obtenerTokenAuth(authorizationHeader);

    usuarioService.modificarUsuario(fiid,requestUsuario, usuario);
}
```

Imagen 40. Método para modificar usuario.

Fuente: Elaboración propia, basada en la práctica.

La interfaz *usuarioService* contenía el método modificarUsuario, que permitía modificar solo el estado del usuario, validando que el alias existiera en el sistema y que el cambio fuera de habilitado a deshabilitado, o viceversa. La fecha y hora de modificación se registraban en la tabla *logger* para fines de auditoría. El repositorio realizaba esta modificación mediante el método *save*, el cual se traducía en una operación de actualización (*update*) en la base de datos. La respuesta para una modificación exitosa era un código 204.

Para integrar el *frontend* con los servicios de agregar y modificar usuario, se crearon métodos en el servicio usuarios.service.ts en Angular, agregarUsuario y modificarUsuario. Estos métodos realizaron solicitudes POST y PUT al *backend*, obteniendo una respuesta transformada en un objeto AgregarUsuarioResponse o en un HttpBaseResponse.

usuarios.service.ts

src > app > protected > usuarios > services > usuarios.service.ts > UsuariosService > modificarUsuario

```
26   export class UsuariosService extends BaseService{
155    agregarUsuario(fiid: string, request: AgregarUsuarioRequest): Observable<AgregarUsuarioResponse>{
157      new HttpClientPostParams();
158
159      httpClientPostParams.url = environment.baseUrl + '/usuarios/agregarUsuario';
160      httpClientPostParams.body = request;
161
162      httpClientPostParams.options.setHeader('fiid', '' + fiid)
163
164      return this.loader
165        .load(this.httpClientService.postData(httpClientPostParams))
166        .pipe(
167          map((response: AgregarUsuarioResponse) => {
168            return response;
169          }),
170        );
171    }
172
173    modificarUsuario(fiid: string, request: ModificarUsuarioRequest){
174      const httpClientPutParams: HttpClientPutParams =
175      new HttpClientPutParams();
176
177      httpClientPutParams.url = environment.baseUrl + '/usuarios/modificarUsuario';
178      httpClientPutParams.body = request;
179
180      httpClientPutParams.options.setHeader('fiid', '' + fiid)
181
182      return this.loader.load(
183        this.httpClientService
184          .putData(httpClientPutParams)
185          .pipe(map((response: HttpBaseResponse) => response)),
186      );
```

Imagen 41. Servicio de autenticación, método para agregar y modificar usuario.

Fuente: Elaboración propia, basada en la práctica.

El componente usuarios-alta.component.ts suscribía el formulario de agregar usuario al servicio agregarUsuario, llamando a este método cuando se enviaba el formulario. En caso de éxito, se mostraba una notificación de éxito y el formulario se restablecía; en caso de error, se desplegaba un mensaje de error. La lógica fue similar para la modificación de usuarios, utilizando el método modificarUsuario().

El formulario para alta de usuario mostraba mensajes en función del resultado de la solicitud, y su interfaz respondía a los diferentes estados: mientras se procesaba la solicitud, se mostraba un indicador de carga, y al completarse la operación se notificaba el éxito o el error al usuario.

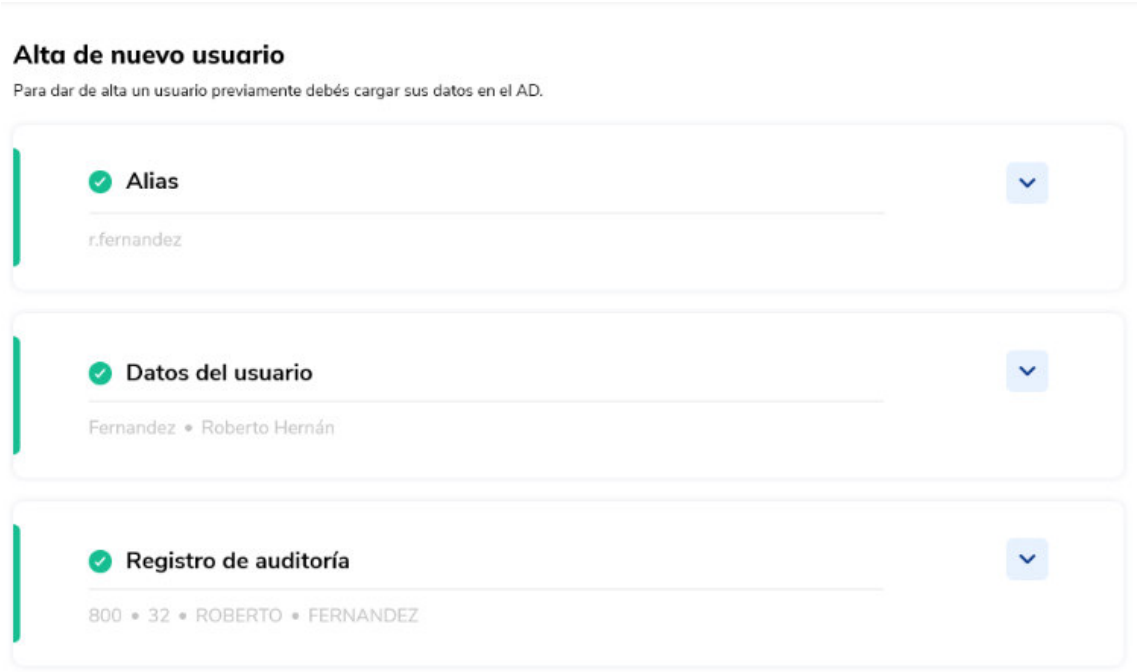


Imagen 42. Formulario de alta de usuario.

Fuente: Elaboración propia, basada en la práctica.

La interfaz de modificación de usuario siguió el mismo flujo de actualización, garantizando consistencia en la experiencia de usuario.

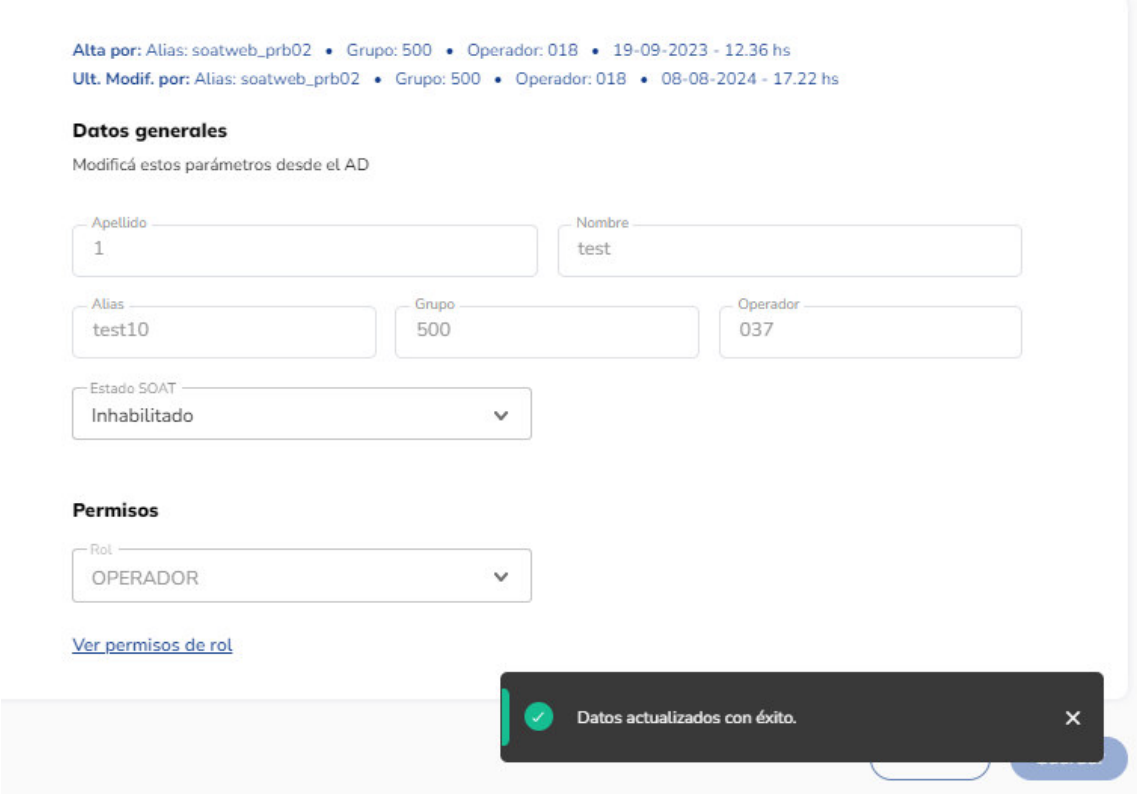


Imagen 43. Ejemplo de modificación de usuario.

Fuente: Elaboración propia, basada en la práctica.

Con estos desarrollos en el *backend* y el *frontend*, se implementaron servicios y pantallas que facilitaron la gestión de usuarios, asegurando un manejo adecuado de validaciones y errores. Esta funcionalidad no solo escaló a nivel de administración, sino que también permitió a los clientes gestionar de forma autónoma la administración de sus usuarios, mejorando tanto la seguridad como la eficiencia del sistema.

10.9 Configuración para despliegue

En esta parte del proyecto, el desarrollo ya se encontraba concluido, por lo cual se procedió a preparar y optimizar la configuración del proyecto para garantizar un despliegue eficiente y automatizado en entornos de desarrollo y producción dentro de OpenShift. Esto incluyó la

creación y configuración de *pipelines* de CI/CD, ajustes específicos en OpenShift, y la implementación de estrategias de despliegue que aseguraron la disponibilidad y estabilidad del sistema en todas las etapas.

Para comenzar con esta tarea, primero se investigó acerca de qué es OpenShift. Se determinó que es una plataforma de contenedores empresarial desarrollada por Red Hat, que permite a las organizaciones construir, desplegar y gestionar aplicaciones en contenedores de manera más eficiente. Está basada en Kubernetes[17], lo cual le proporciona una sólida base para la orquestación de contenedores, pero también añade herramientas adicionales para mejorar la experiencia del usuario, como la integración continua y la entrega continua (CI/CD), la gestión de versiones, la seguridad mejorada y un entorno de desarrollo más controlado.

Luego, se procedió con el diseño e implementación de *pipelines* de CI/CD utilizando GitLab CI para el despliegue tanto del *backend* como del *frontend* de SoatWeb en contenedores. Para ello, se agregaron al proyecto los siguientes archivos:

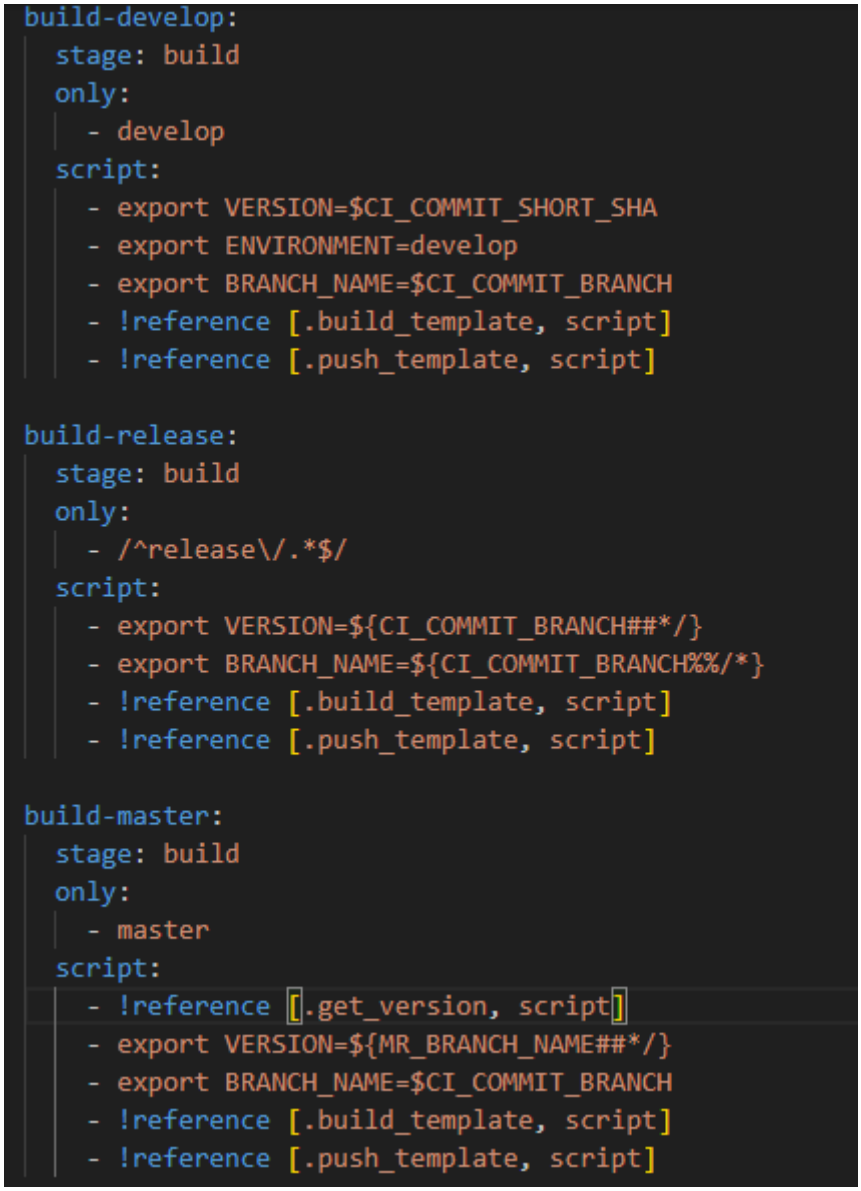
Definición de *Stages* en gitlab-ci.yml

Se definió una etapa de construcción (*build*) que utilizó el Dockerfile para crear una imagen de Docker. Este proceso incluyó:

- Construcción de la Imagen: El trabajo de construcción ejecutó *docker build* para crear la imagen de la aplicación.

-Publicación de la Imagen: La imagen construida se empujó a un registro de contenedores para su uso en despliegues posteriores.

[17] Kubernetes es la herramienta líder para la orquestación de contenedores, proporcionando la infraestructura necesaria para ejecutar aplicaciones en la nube de manera eficiente, escalable y resiliente.



```
build-develop:
  stage: build
  only:
    - develop
  script:
    - export VERSION=$CI_COMMIT_SHORT_SHA
    - export ENVIRONMENT=develop
    - export BRANCH_NAME=$CI_COMMIT_BRANCH
    - !reference [.build_template, script]
    - !reference [.push_template, script]

build-release:
  stage: build
  only:
    - /^release\/.*$/
  script:
    - export VERSION=${CI_COMMIT_BRANCH##*/}
    - export BRANCH_NAME=${CI_COMMIT_BRANCH%%/*}
    - !reference [.build_template, script]
    - !reference [.push_template, script]

build-master:
  stage: build
  only:
    - master
  script:
    - !reference [.get_version, script]
    - export VERSION=${MR_BRANCH_NAME##*/}
    - export BRANCH_NAME=$CI_COMMIT_BRANCH
    - !reference [.build_template, script]
    - !reference [.push_template, script]
```

Imagen 44. Organización de los pasos para la construcción del proyecto.

Fuente: Elaboración propia, basada en la práctica.

En este caso, se organizó el proceso de construcción de imágenes Docker en diferentes trabajos basados en la rama del *commit*. Se utilizaron plantillas para estandarizar los pasos comunes y ajustar la configuración según el tipo de rama (*develop, release/*, master*), asegurando que cada tipo de construcción tuviera las variables y configuraciones adecuadas.

Definición y Configuración de los Contenedores

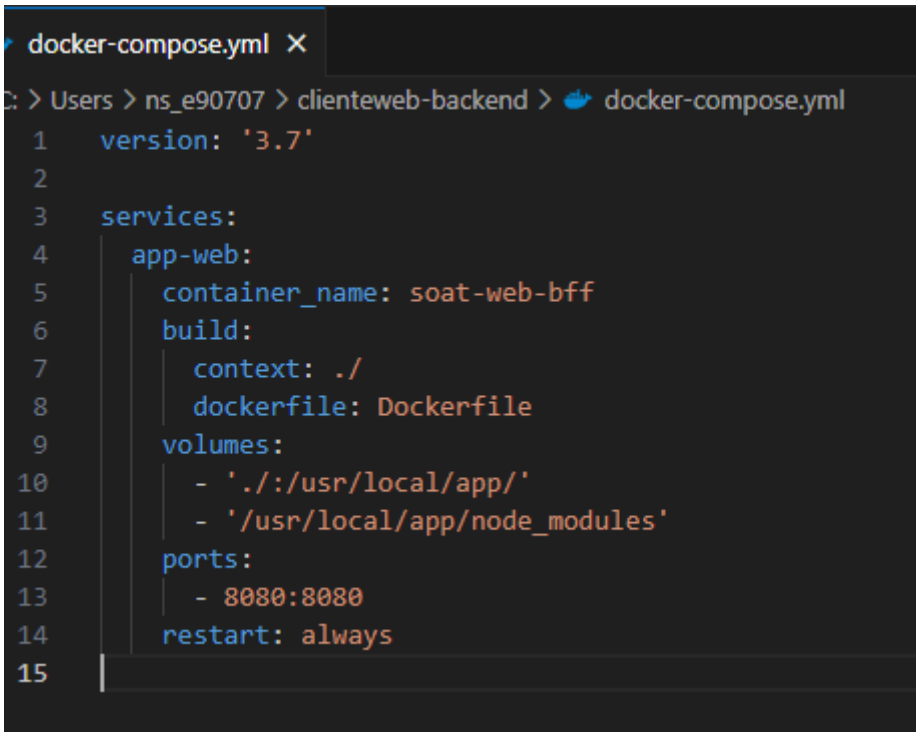


Imagen 45. Definición de *docker-compose*.

Fuente: Elaboración propia, basada en la práctica.

Se definió la configuración del servicio del *backend* del proyecto de la siguiente manera:

- Definición del servicio: Se especificó un servicio llamado *app-web* que sería gestionado por Docker Compose. Este servicio era esencialmente una instancia de un contenedor que ejecutaba la aplicación web.

- Construcción del contenedor:

  - *context*: Indicó que el contexto de construcción era el directorio actual.

  - *dockerfile*: Especificó el uso del Dockerfile en el directorio actual para construir la imagen.

- Configuración de volúmenes: Se montó el directorio local en el contenedor, permitiendo que los cambios en el código fuente se reflejaran en el contenedor sin necesidad de reconstruir la imagen.

- Asignación de puertos: Se expuso el puerto 8080 del contenedor al puerto 8080 del *host*, permitiendo que la aplicación fuera accesible desde el navegador.

- Política de reinicio: Se configuró el contenedor para reiniciarse automáticamente si se detenía o si el *host* se reiniciaba.

Despliegue en Kubernetes/OpenShift

El despliegue en Kubernetes/OpenShift fue gestionado mediante un *bootstrapper* [18]en el contexto de CI/CD. Argo CD funcionó como un *bootstrapper* en el contexto de despliegues continuos en Kubernetes. Como una herramienta de GitOps, Argo CD gestionó y automatizó el despliegue de aplicaciones en Kubernetes desde los repositorios de Git, asegurando que el estado deseado en Git estuviera sincronizado con el estado actual del clúster.

[18] Un *bootstrapper* es un componente o herramienta que se encarga de configurar y automatizar la instalación y despliegue de aplicaciones en un clúster, como Kubernetes u OpenShift.

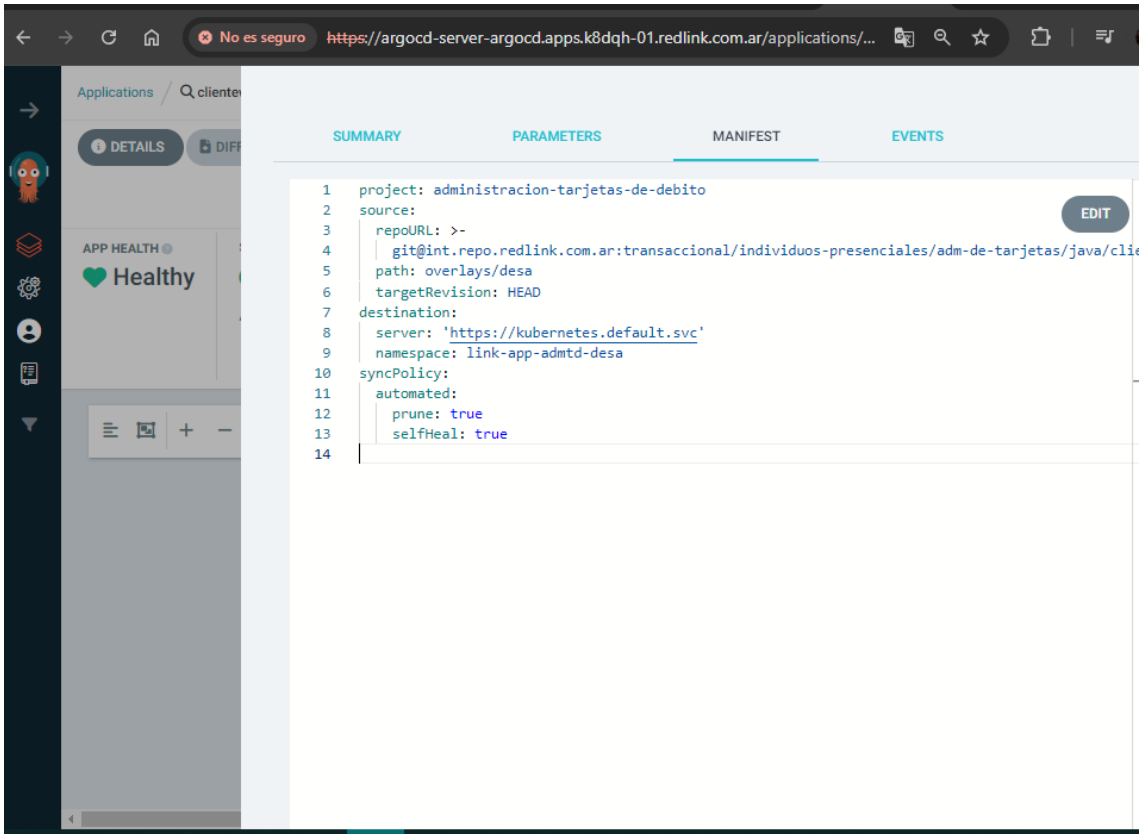


Imagen 46.Configuración de repositorio de infra con Argo CD.

Fuente: Recuperado de https://argocd-server-argocd.apps.k8dqh-01.redlink.com.ar/applications/clienteweb-backend-desa?resource=&conditions=false

Además, se crearon Secrets y ConfigMaps, que son componentes clave en OpenShift para gestionar la configuración y la seguridad de las aplicaciones. Los ConfigMaps se utilizaron para almacenar datos de configuración en pares clave-valor, configurando la conexión a la base de datos de los diferentes ambientes. Los Secrets se crearon para gestionar las credenciales de conexión a la base de datos y la interacción con el Active Directory(AD). Ambos permitieron mantener la configuración y la información sensible separada del código de la aplicación, facilitando la gestión, seguridad y flexibilidad en el despliegue y operación de aplicaciones en Kubernetes y OpenShift.

También se avanzó con la creación de manifiestos, que son archivos de configuración en formato YAML que definieron los recursos a crear y gestionar en un clúster de Kubernetes u OpenShift, incluyendo recursos como Pods, Deployments, Services, ConfigMaps, Secrets, entre otros. Todo esto fue gestionado mediante el *bootstrapper*. Como resultado, se obtuvo un esquema donde la aplicación se desplegó en un *pod*, que es una unidad básica de despliegue y ejecución en Kubernetes. El primer despliegue se realizó en ambientes bajos, en este caso desarrollo.

Finalmente, se configuraron los proyectos de *backend* y *frontend* para el despliegue en ambientes de desarrollo y producción en OpenShift, facilitando la integración entre los componentes de la aplicación y la infraestructura del clúster. Se automatizó el proceso de construcción, prueba y despliegue mediante *pipelines* de CI/CD en GitLab CI, logrando:

- ✓ Automatización del Despliegue: Despliegue continuo con integración de nuevas versiones.
- ✓ Gestión Segura de Configuración: Uso de ConfigMaps y Secrets para manejar configuración y datos sensibles.
- ✓ Optimización de Construcción: Definición de imágenes Docker para entornos consistentes.
- ✓ Herramientas de Gestión: Implementación de Helm y Kustomize para facilitar la administración de despliegues.

10.10 Generación de *release* y carga inicial de datos

El proceso de generación de una versión estable y la carga inicial de datos fue crucial para garantizar que la primera versión del sistema estuviera completamente preparada para su uso en producción. Este proceso implicó la creación y el despliegue de la versión final del Módulo Consultivo del sistema, la ejecución de pruebas exhaustivas para asegurar su funcionalidad y estabilidad, y la carga de los datos iniciales necesarios para su operación.

A partir de la tarea 10.10, se procedió con la generación del entregable creando un *branch* denominado *release/X.X.X* que contenía determinadas funcionalidades. En nuestro caso, se definió que tanto para el *backend* como para el *frontend* se generaría el *release/1.0.0* en ambos repositorios. Esta versión se compiló y se generó una imagen para que el contenedor de OpenShift la utilizara y creara el contenedor de despliegue.

El *release* generado se desplegó en el ambiente de homologación para realizar pruebas, con el objetivo de avanzar hacia producción una vez concluidas las pruebas. Además, se llevaron a cabo las siguientes acciones:

- Se avanzó con la implementación de archivos, en este caso para las nuevas tablas específicas del nuevo cliente.

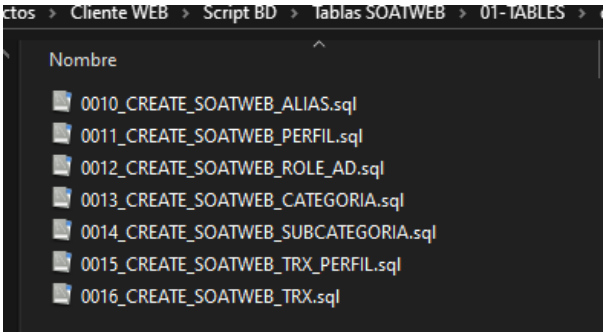


Imagen 47. *Scripts* de base de datos.

Fuente: Elaboración propia, basada en la práctica

- Se creó dentro del *Active Directory* (AD) un directorio para los usuarios específicos de la nueva aplicación.

- En el sistema, se ingresó por única vez para crear un usuario administrador a través de un archivo de configuración. Este usuario luego creó, a través de SoatWeb, todos los usuarios necesarios con los permisos correspondientes, así como también se encargó de dar de alta al usuario administrador principal de cada entidad. El usuario administrador de cada entidad solo pudo gestionar usuarios de su propia entidad.

Se finalizó el desarrollo del Módulo Consultivo, salvo que durante las pruebas se reportaran errores. Se implementaron las nuevas tablas en producción y se habilitó en *Active Directory(*AD*),* un directorio específico para SoatWeb, permitiendo la creación de usuarios con roles determinados. Además, se contó con la versión candidata para producción. Solo quedó pendiente la compra de un certificado de dominio para poder publicar el sistema una vez disponible en producción.

10.11 Implementación del Módulo Consultivo en producción

En este último punto se buscó desplegar el Módulo uno (Consultivo) en el entorno de producción, habilitando a los usuarios para realizar consultas y visualizar información. Este módulo fue una parte crucial del sistema, permitiendo a los usuarios interactuar con los datos de manera efectiva y obtener la información necesaria para sus procesos.

Para llevar a cabo este despliegue, se realizaron varias tareas:

- Pruebas Finales: Se realizaron pruebas exhaustivas del Módulo uno para asegurar que todas las funcionalidades de consulta y visualización estuvieran operativas y funcionaran según lo previsto en el entorno de producción.

- Despliegue del Módulo: Se validó que el entorno de producción estuviera configurado correctamente tanto a nivel de base de datos como a nivel de configuraciones en OpenShift. Se generó una solicitud de fusión a la rama *master* en GitLab con el *release* que se quería implementar en producción. Luego, se realizó una verificación post-despliegue para confirmar que el módulo se instalara correctamente y que funcionara como se esperaba.

-Verificación de Actividad: Una vez desplegada la aplicación, se revisaron los *logs* para confirmar que tanto el *backend* como el *frontend* se hubieran iniciado correctamente.

- Asistencia a Clientes: A medida que cada entidad se integraba, se realizaron reuniones previas para identificar las condiciones específicas de cada una y brindar un acompañamiento integral durante la transición al nuevo sistema.

Con la finalización de esta tarea, se completó con éxito el despliegue de la aplicación en producción. Los usuarios ahora tenían acceso al nuevo sistema, lo cual permitió recopilar opiniones valiosas para evaluar posibles mejoras basadas en su uso. Este despliegue marcó un hito importante para el equipo, ya que representó nuestra primera aplicación web desplegada en OpenShift. Este logro no solo demostró la capacidad de llevar aplicaciones a producción en un entorno de nube moderno, sino que también sentó las bases para futuras implementaciones y optimizaciones.

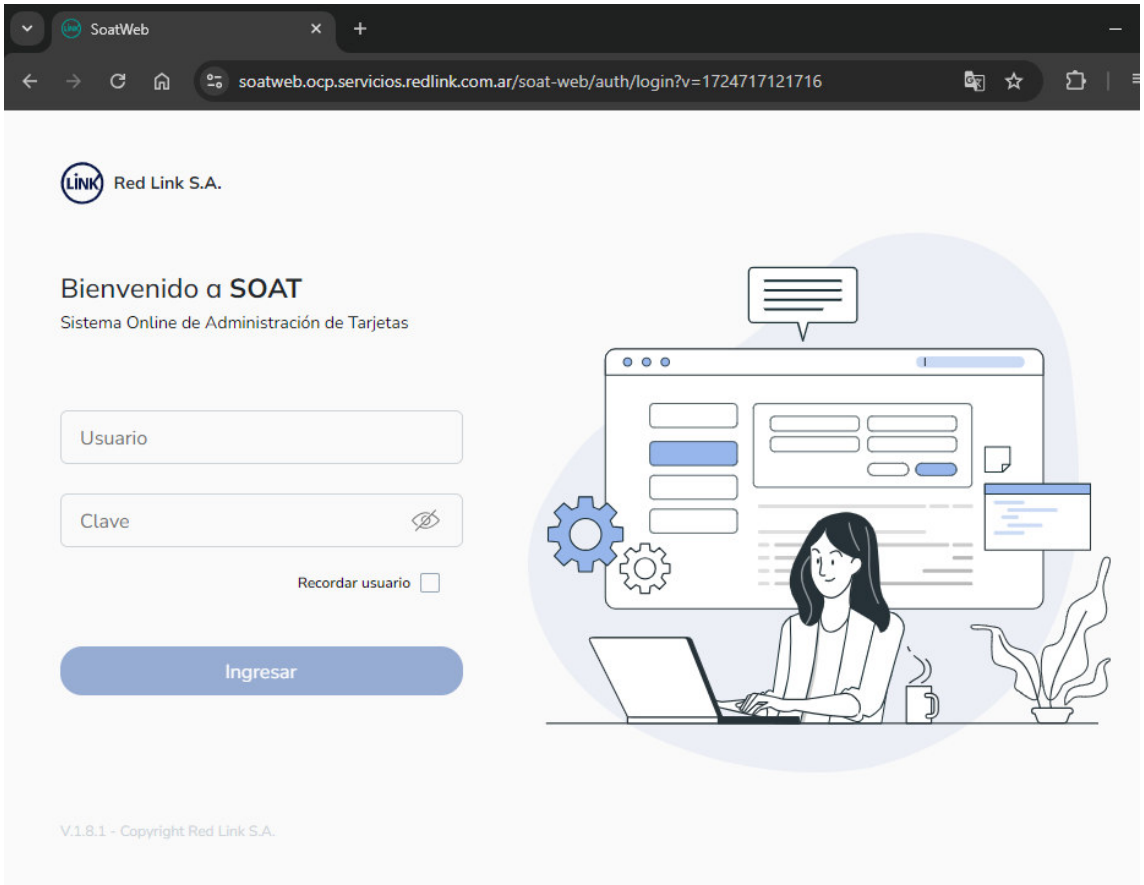


Imagen 48. Aplicación publicada y disponible en producción.

Fuente: Recuperado de https://soatweb.ocp.servicios.redlink.com.ar/soat-web/auth/login

10.12 Conclusiones

El desarrollo y despliegue del Módulo uno (Consultivo) representó un avance significativo, tanto a nivel técnico como profesional, en la reingeniería del sistema SoatWeb. Retomando la motivación inicial del proyecto, se buscó crear una herramienta que mejorara la accesibilidad y eficiencia para los usuarios, eliminando la dependencia de sistemas de escritorio y facilitando la transición hacia un entorno moderno basado en la web. Con este despliegue, los usuarios ahora tienen acceso a la información clave de manera sencilla, con un enfoque que potencia el uso y disponibilidad.

Desde un punto de vista técnico, el proyecto presentó una serie de desafíos que fueron abordados mediante una planificación cuidadosa y una ejecución rigurosa. La implementación en OpenShift y la configuración adecuada del entorno de producción aseguraron que el sistema alcanzara la calidad y estabilidad requeridas. Además, el proceso permitió consolidar conocimientos sobre tecnologías modernas y prácticas recomendadas de despliegue. Como resultado, el sistema no solo es una herramienta efectiva para el uso cotidiano, sino que también proporciona una base robusta para futuras expansiones.

Este módulo consultivo tiene el potencial de ser aplicado a otras áreas de la organización que requieran una plataforma accesible para la consulta de información y el análisis de datos. Asimismo, podrían desarrollarse módulos adicionales que complementen las funcionalidades ya existentes, creando un ecosistema integral de herramientas accesibles desde la web. Otra línea posible de trabajo sería explorar la integración con sistemas de análisis de datos para ofrecer estadísticas en tiempo real y apoyar la toma de decisiones informada.

En cuanto a futuras investigaciones, surge la pregunta de cómo podría evolucionar este sistema para incluir capacidades más avanzadas, como la inteligencia artificial para el análisis predictivo o la automatización de ciertos procesos de decisión. También se podría explorar cómo integrar más entidades y usuarios sin comprometer el rendimiento, o cómo ampliar la funcionalidad para soportar más casos de uso, como la gestión integral de transacciones.

En lo personal, este proyecto representó un verdadero salto en mi carrera profesional, brindándome la seguridad necesaria para avanzar en el ámbito programático y enfrentar nuevos desafíos. Participar activamente en cada etapa del proceso me permitió fortalecer mis habilidades, tanto técnicas como de gestión, y valorar el impacto de mi trabajo en el producto final y en el crecimiento del equipo. Esta experiencia no solo consolidó mis capacidades técnicas, sino que también reafirmó la importancia del trabajo colaborativo y del aprendizaje continuo.

10.13 Bibliografía

Angular Documentation. (n.d.). Architecture Overview. Recuperado de https://docs.angular.lat/guide/architecture

DER (Diagrama Entidad-Relación) dbdiagram.io. (n.d.). Herramienta para generar diagramas de entidad-relación (DER). Recuperado de https://dbdiagram.io

Diagramas de Secuencia WebSequenceDiagrams. (n.d.). Recuperado de https://www.websequencediagrams.com/app

GitLab CI/CD GitLab. (n.d.). Getting started with GitLab CI/CD. Recuperado de https://docs.gitlab.com/ee/ci/

Herramientas para pruebas y calidad del software Selenium. (n.d.). Selenium Documentation. Recuperado de https://www.selenium.dev/documentation/en/

Herramientas para pruebas y calidad del software JUnit. (n.d.). JUnit 5 User Guide. Recuperado de https://junit.org/junit5/docs/current/user-guide/

JavaScript Mozilla Developer Network. (n.d.). JavaScript First Steps: What is JavaScript?. Recuperado de https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript

JPA Baeldung. (n.d.). The Persistence Layer with Spring Data JPA. Recuperado de https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

Kubernetes/OpenShift Red Hat. (n.d.). OpenShift Documentation. Recuperado de https://docs.openshift.com/

Kubernetes/OpenShift The Linux Foundation. (n.d.). Kubernetes Documentation. Recuperado de https://kubernetes.io/docs/home/

Login, Token JWT Bezkoder. (n.d.). Spring Boot JWT Authentication. Recuperado de https://www.bezkoder.com/spring-boot-jwt-authentication/

Login, Token JWT Softtek Blog. (n.d.). Autenticando APIs con Spring y JWT. Recuperado de https://blog.softtek.com/es/autenticando-apis-con-spring-y-jwt

OpenAPI y Swagger Baeldung. (n.d.). Spring REST OpenAPI Documentation. Recuperado de https://www.baeldung.com/spring-rest-openapi-documentation

Spring Boot Spring. (n.d.). Spring Boot Reference Documentation. Recuperado de https://docs.spring.io/spring-boot/docs/current/reference/html

TypeScript Kinsta. (n.d.). ¿Qué es TypeScript? Recuperado de https://kinsta.com/es/base-de-conocimiento/que-es-typescript/#qu-es-typescript