Navarro, Gabriel

Croupier Stock

2020

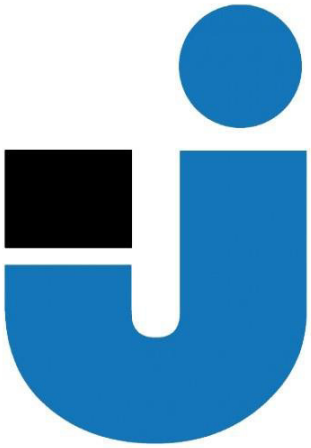
Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática



TRABAJO FINAL DE LA PRÁCTICA PROFESIONAL SUPERVISADA

Croupier Stock

Estudiante:

Gabriel Navarro

Tutores:

Prof. Lía Lavigna

Dr. Ing. Martin Morales

Ing. Jésica Guzman

Ing. Guillermo Fritz

Buenos Aires, 2020

|PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
Croupier Stock
Informe de Avance

DATOS DEL ESTUDIANTE

Apellido y Nombres: Navarro Gabriel

DNI: 38612058

Nº de Legajo: 7238

Correo electrónico: Gabriel.navarro@hotmail.es

Cantidad de materias aprobadas al comienzo de la PPS: 44

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16.

DOCENTES SUPERVISORES

Apellido y Nombres: Ing. Jésica Guzmán

Correo electrónico: jvguzman06@gmail.com

Apellido y Nombres: Dr. Ing. Martín Morales

Correo electrónico: martin.unaj@gmail.com

DOCENTE TUTOR DEL TALLER DE APOYO PARA LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Prof. Lia Lavigna

Correo electrónico: lialavigna@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Santander Tecnología

Dirección: Uspallata 2953

Teléfono: 5030-4100

Sector: Desarrollo de Software Bancario

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Ing. Guillermo Fritz

Correo electrónico: gfritz@santandertecnologia.com.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Índice

1. Introducción
    1.1 Objetivos
    1.2 Tareas a Ejecutar
    1.3 Cronograma de Trabajo
        1.3.1 Diagrama de Grantt
2. Desarrollo
    2.1 Tecnologías Utilizadas
        2.1.1 Lenguaje Java
        2.1.2 Lenguaje Kotlin
        2.1.3 Framework SrpingBoot
        2.1.4 Maven
        2.1.5 Base de Datos Oracle
        2.1.6 Librería Liquibase
        2.1.7 GitLab
        2.1.8 Openshift
        2.1.9 Metodología Scrum
        2.1.10 Swagger
    2.2 Arquitectura
        2.2.1 Estructura API
        2.2.2 Arquitectura API
        2.2.3 Ecosistema de APIs
    2.3 Metodología de Desarrollo
    2.4 Autenticación
    2.5 Patrón de Diseño State
        2.5.1 Estados de un envío
    2.6 Integración *frontend - backend*
        2.6.1 Login
        2.6.2 Pantalla de recepción
        2.6.3 Pantalla de Stock
        2.6.4 Solapa de Destrucción

2.6.5 Reporte
2.6.6 Historial
3. Conclusiones
3.1 Reflexión sobre las prácticas profesionales como espacio de formación

1. Introducción

La presente Práctica Profesional Supervisada (PPS) tiene como finalidad el desarrollo del sistema Croupier Stock para la red de sucursales del Banco Santander.

La iniciativa surgida en el equipo de Delivery, bajo la gerencia de Medios de Pago, llevó a cabo el desarrollo de una aplicación para tener el *tracking* -rastreo- punto a punto de un envío. Es decir, el momento de su 'creación', hasta la remisión a sucursal o domicilio. En el presente, Andreani es la empresa que tiene a cargo el servicio de mensajería – Courier-. Sin embargo, este proyecto no contempló una solución rápida y necesaria para la operatoria actual que existe en las sucursales, donde el control y manejo de stock no es eficiente, dado que los operarios no tienen el total control y conocimiento de los paquetes que existen en su stock. Esta situación se presenta debido al uso de un aplicativo llamado ASGM (Sistema de Autogestión de Stock General), el cual posee múltiples errores y limitantes. Su principal problema es la respuesta errónea a los clientes que intentan retirar sus envíos, y, desde las sucursales, desconocen si son poseedores o no de los paquetes demandados por los clientes.

La aplicación 'Croupier Stock' surge para paliar las limitaciones existentes en este sistema y reemplazarlo totalmente, brindando un estado consistente de los envíos, un sistema de reportes, actualización en tiempo real de eventos por parte de Andreani, escalabilidad, estados dinámicos, mejor UX (experiencia de usuario), entre otros.

1.1 Objetivos

- Diseñar y desarrollar un sistema que permita un eficiente control y manejo de paquetes, tarjetas retenidas y stock en general dentro de las sucursales (backend).
- Reemplazar la aplicación existente con el menor impacto negativo durante la transición.
- Realizar las integraciones necesarias y convenientes a las APIs (interfaz de aplicación disponible para ser utilizada por terceros) de los diversos sistemas (Andreani, APIs internas al banco).
- Publicar una API para que sea utilizada por los diferentes sistemas del banco que así lo requieran.
- Impactar en la satisfacción del cliente, brindándole una mejor experiencia con la información obtenida, agilizando las operaciones presenciales.

1.2 Tareas a ejecutar

El desarrollo del proyecto se llevará a cabo durante seis meses. El mismo será trabajado bajo metodologías ágiles, dividido en iteraciones o *sprints* (intervalo de trabajo que tiene como resultado un incrementable o entregable) de dos semanas de duración cada uno. Cada dos sprints, se cumplirá una etapa.

• Relevamiento: el mismo parte de la necesidad urgente respecto al manejo del stock de las sucursales. Éste ya fue realizado, sin embargo, en cada iteración se

incluirán reuniones con operarios de las sucursales para tener una retroalimentación más eficaz.

• Propuesta de MVP (Producto Viable Mínimo): análisis de lo mínimo e indispensable que tendrá la aplicación para poder ser puesta en producción con sucursales piloto, para luego ser masificado.

• Desarrollo de MVP: se desarrollará lo propuesto en el MVP, con sus funcionalidades básicas y necesarias de backend. El frontend -interfaz con la que interactúa el usuario- será realizado por otro desarrollador.

• Fase de prueba de MVP: una vez puesto en producción el MVP, el mismo tendrá que ser testeado por las mismas sucursales, para validar el correcto funcionamiento y ser liberado para las demás sucursales de la red.

• Liberación de aplicación para toda la red del banco: pasada la etapa de prueba, se liberará una versión estable para todas las sucursales, teniendo una retroalimentación que realizará e integrará nuevos *features* (funcionalidades).

• Seguimiento: durante cada *sprint* se realizará el refinamiento de las tareas, plannings (reuniones para definir las tareas a realizar durante el sprint), retrospectivas, test por parte de los *products owners* (los dueños del producto y quienes lo aprueban) y análisis de los requerimientos a desarrollar durante el mismo.

1.3 Cronograma de trabajo

1.3.1 Diagrama Grantt

Tarea	Etapa uno	Etapa dos	Etapa tres	Etapa cuatro	Etapa cinco	Etapa seis
RELEVAMIENTO						
Análisis de problemática						
Reuniones con usuarios						
Reuniones con el negocio						
Refinamientos y análisis por sprint						
DESARROLLO						
Propuesta de MVP						
Configuración de ambientes						
Elección de tecnologías						
Desarrollo de endpoints de backend						

Integración con Andreani						
Integración APIs del banco						
Exposición de API						
Prueba de Aceptación						
Pruebas de aceptación de MVP (sucursales piloto)						
Pruebas en sucursales masivas						
Pruebas de Product Owners durante sprints						

2. Desarrollo

Como se mencionó en el informe preliminar, el objetivo principal de la Práctica Profesional Supervisada es el desarrollo (*backend*) de un sistema para gestionar y controlar el stock de manera eficiente dentro de cada sucursal del Banco Santander.

Dicho proyecto surgió como un desarrollo “rápido” (táctico como se lo denomina dentro de la entidad bancaria), por lo que debía ser algo de rápida puesta en producción; sin embargo, se manifestaron requerimientos y mejoras solicitadas para todas las sucursales. En un primer momento se requirió la intervención individual y personal del desarrollo (backend) del proyecto, pero debido a la demanda de ampliarlo a otras filiales, el mismo permitió y demandó la participación

de otros integrantes. Por ello, el alcance original del aplicativo se extendió, y pasó a ser un desarrollo entre varias personas.

En primera instancia, las decisiones técnicas fueron tomadas por la figura del arquitecto, el cual proveyó al equipo una arquitectura para la aplicación basada en la experiencia personal, y, realizando las consultas, se adoptaron los lenguajes de programación y frameworks a utilizar. Para dicha decisión, se tomaron en cuenta conocimientos del equipo, experiencias, homologaciones referidas por parte de la Institución - esto es debido a que no todos los lenguajes de programación están homologados- y soporte.

A continuación, se enumeran los puntos relevantes del proyecto y se detalla cada uno de ellos:

- Tecnologías utilizadas.
- Metodología de desarrollo.
- Ejecución del Cronograma de tareas.

2.1 Tecnologías utilizadas

2.1.1
Lenguaje de Programación Java

Java es un tipo de lenguaje de programación y una plataforma informática, creada y comercializada por Sun Microsystems en el año 1995.

Se constituye como un lenguaje orientado a objetos, compilado y su intención es permitir que los desarrolladores de aplicaciones escriban el programa

una sola vez y lo ejecuten en múltiples dispositivos. Para ello, Java incorpora una importante innovación llamada JVM, Java Virtual Machine. La máquina virtual de Java permite ejecutar el código de este lenguaje, de modo que Java es soportado en cualquier sistema que incorpore su propia máquina virtual.

Por el conocimiento y experiencia con dicho lenguaje de programación, la mayor parte del código del aplicativo será desarrollado en Java, cuyo logo se observa en la Ilustración #1.

Ilustración 1 - Lenguaje de Programación Java
Fuente: https://www.genbeta.com/desarrollo/java-el-lenguaje-mas-usado-y-su-evolucion

Cuando se compila un programa realizado en Java se genera un archivo denominado bytecodes, cuya funcionalidad consiste en ser traducido a lenguaje de máquina. Ese archivo puede ser entendido por la correspondiente máquina virtual y procesado tal como el sistema operativo real requiera.

2.1.2
Lenguaje de Programación Kotlin

Kotlin es un lenguaje de programación pragmático pensado para funcionar con Máquina Virtual de Java (JVM) y Android. Además, puede ser compilado a código fuente de Javascript.

Éste se caracteriza por contar con una perfecta combinación de características claramente orientadas a la funcionalidad durante la programación, centrándose en la seguridad, la claridad y la interoperabilidad.

Los programas en Kotlin pueden utilizar los frameworks y librerías de Java existentes. Asimismo, su interoperabilidad no requiere capas de adaptación.

A continuación, en la Ilustración #2 se observa el logo de Kotlin.

Ilustración 2 – Kotlin
Fuente: https://institutotame.com/de-java-a-kotlin-parte-1/

A modo de incorporar experiencia en un lenguaje nuevo, con el incentivo del arquitecto y desarrolladores Sr (con más años de experiencia) en el equipo, se utilizará dicho lenguaje para los clientes HTPP (protocolo de transferencia de hipertextos) que se comuniquen con otras APIs.

2.1.3
Framework Spring Boot

Spring Boot, publicado en 2012, es una solución para el *framework* (entorno de trabajo que facilita el desarrollo de *software*) Spring de Java que sigue el principio de "convención sobre configuración" y reduce la complejidad del desarrollo de nuevos proyectos basados en Spring. Para ello, Spring Boot proporciona la estructura básica configurada del proyecto, que incluye las pautas para usar el marco y todas las bibliotecas de terceros relevantes para la aplicación, lo que allana el camino para comenzar el desarrollo de forma más rápida. De esta manera, se simplifica mucho la creación de aplicaciones independientes y reproducibles.

La Ilustración #3 muestra el logo del framework Spring Boot.

Ilustración 3- SpringBoot
Fuente: https://www.mancomun.gal/es/noticias/meetup-de-corunajug-montando-un-framework-sobre-spring-boot/

Las características de Spring Boot pueden resumirse de la siguiente manera:

- Incorporación directa de aplicaciones de servidores web/contenedores como Apache Tomcat o Jetty, eliminando la necesidad de incluir archivos WAR (Web Application Archive).
- Simplificación de la configuración de Maven gracias a los POM (Project Object Models) "starter".
- Configuración automática de Spring en la medida de lo posible.
- Características no funcionales, como métricas o configuraciones externalizadas.

Este *framework* será sobre el cual se desarrollará la aplicación Croupier-Stock.

2.1.4
Apache Maven

Apache Maven (Ilustración #4) es un software que permite gestionar proyectos.

Ilustración 4- Maven
Fuente: http://javadesde0.com/guia-basica-maven/

Al crear un proyecto de Maven, se generará automáticamente una estructura de carpetas muy concreta que ya se presenta predefinida. Como se observa en la Ilustración #5:

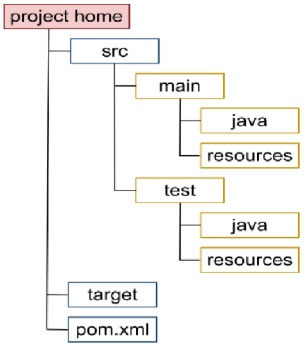


Ilustración 5 - Estructura proyecto Maven
Fuente: http://javadesde0.com/guia-basica-maven/

- **src/main/java**: contiene el código fuente. El contenido de este directorio se conoce bajo el nombre de "módulo".
- **src/main/resources**: ubicación de los recursos estáticos (XML, propiedades, etc.) que necesita del módulo para funcionar correctamente.
- **src/test/java:** ruta con los ficheros de pruebas (*testing*) para verificar el correcto funcionamiento del módulo.
- **src/test/resources**: contiene los recursos estáticos utilizados por los *tests*.
- **pom.xml**: el Project Object Model (POM) es el encargado de gestionar y construir los proyectos, puesto que contiene el listado de dependencias que son necesarias para que el proyecto funcione. Toda la información de la propuesta está basada en este fichero. El mismo tiene una extensión XML (*Extensible Markup Language*) y desde la propia web oficial de Apache Maven, lo definen como el núcleo central, por lo que se puede afirmar que es el "núcleo" del proyecto.

2.1.5
Base de Datos ORACLE

Oracle Database es un sistema de gestión de base de datos de tipo objeto-relacional (ORDBMS, por el acrónimo en inglés de Object-Relational Data Base Management System), desarrollado por Oracle Corporation. En la Ilustración #6 se presenta su logo.

Ilustración 6- Oracle
Fuente: http://www.linware.com.ar/blog/2013/07/02/oracle-y-su-primera-base-de-datos-en-la-nube-database-12c/

En este desarrollo, la API será montada sobre una base de datos Oracle, con el siguiente diagrama de entidad – relación (forma visual de ver las relaciones entre las tablas de una base de datos), como se observa en la Ilustración #7.
:

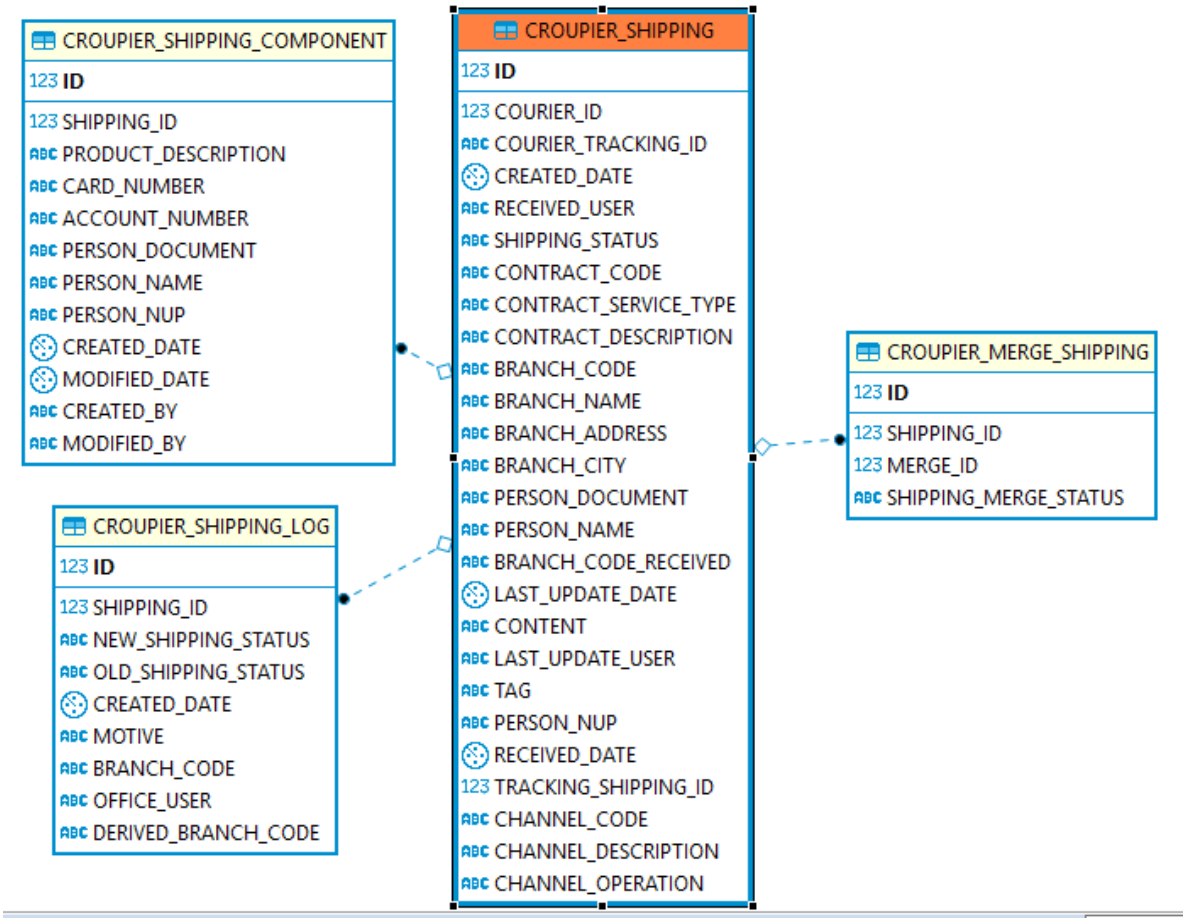


Ilustración 7- DER
Fuente: Elaboración propia basada en la práctica - Base de Datos de desarrollo

La entidad principal es 'Croupier_Shipping', que presenta gran cantidad de información respecto a los envíos. Dicha tabla, entre sus campos más importantes, posee un ID propio (generado por la base de datos), el "tracking id", el cual representa el código manejado por la entidad de correo, sucursal de destino del envío, sucursal donde efectivamente se recibió, fechas de actualización, recepción, un estado, un id para relacionarlo con otro sistema, información del cliente dueño

del envío (DNI, nombres, nup - identificador único dentro de la Entidad Bancaria -), entre otros.

La tabla 'Croupier_Shipping_Log' posee la información respecto a los cambios de estados y transiciones de los envíos. La misma se utiliza para tener un rastreo para auditoría y los diversos registros y reportes que pueden consultar los usuarios de las sucursales. Por ejemplo, consultar los envíos que se encuentran desde *x* cantidad de días en stock, recibidas durante el día, entre otros.

Por otro lado, la entidad 'Croupier_Shipping_Component' posee los componentes de los envíos. En una primera instancia, solamente se tenía información respecto a los paquetes, con mínima información respecto a su interior. Por necesidades de los usuarios, se debió agregar qué componentes poseían en su interior e información de estos, obteniéndolos a partir de otra API.

Por último, la tabla 'Croupier_Merge_Shipping' se creó para las situaciones específicas de fusión de sucursales. Durante una unión de éstas, dicha tabla funciona como auxiliar para realizar la derivación de los envíos de las sucursales que desaparecen o fusionan, a otras.

2.1.6
Librería Liquibase

Liquibase (Ilustración #7) es una librería Open Source (código abierto), totalmente independiente del sistema de Base de Datos, que permite realizar el seguimiento, gestión y aplicación de cambios en el modelo de datos.

Ilustración 8- Liquibase
Fuente: https://www.liquibase.org/

2.1.7
GitLab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en *Git* (control de versiones de código). En la Ilustración #8 se presenta el logo de esta tecnología.

Ilustración 9- GitLab
Fuente: https://es.wikipedia.org/wiki/GitLab

Esta herramienta es ofrecida y homologada por el Banco Santander como repositorio de todos los proyectos existentes (para migrarse) y, de forma obligatoria,

para los nuevos. El sistema de trabajo (*gitflow)* propuesto por el equipo de DevOps (desarrollo y operaciones en conjunto) y basado en la convención será el de tres ramas principales en el repositorio: *master, staging y development*.

Por una parte, Master es el *branch (*rama) que refleja lo que está en producción y siempre tiene que estar igualado con ésta.

Por otra parte, Staging es un paso previo a Master, lo que podría considerarse un escenario de 'pre producción'.

Por último, Development es la rama de trabajo constante. De ella saldrán las ramas donde se desarrollarán todas las funcionalidades, las cuales serán '*mergeadas' (*integradas a development), previo a un *merge request (*una solicitud de integración), y que será aprobada por otro desarrollador para tener una instancia de revisión de código.

Además, se contará con un *pipeline* (una automatización de tareas), en lo que respecta a la subida de código en las ramas *development, staging y master.* Una vez que se ejecuten dichos pipelines, se realizará una compilación del código, una ejecución de todos los test, un análisis de calidad de código con la herramienta "Sonar" y un *deploy (*despliegue de la aplicación) en los ambientes correspondientes: *development* en desarrollo, *staging* en pre producción y *master* en producción.

Los despliegues serán en *Openshift*, una herramienta impulsada por la Institución bancaria para tener las aplicaciones 'en la nube'.

2.1.8
Openshift

OpenShift (Ilustración #9) es la plataforma de desarrollo, con características de Cloud Computing (computación en la nube) de la capa PaaS (Plataforma como Servicio), que ofrece la empresa Red Hat.

La aplicación se *deployará* en esta plataforma, luego de ser configurada correctamente en el proyecto, con ayuda del equipo de DevOps y el arquitecto del equipo.

Ilustración 10- OpenShift
Fuente: https://revistabyte.es/actualidad-it/ibm-red-hat-openshift/

La aplicación debe configurarse en el directorio *okd.* Dichos archivos funcionan para agregar propiedades, secretos, estrategias de *deployment,* entre otros.

2.1.9
Metodología Scrum

Scrum es un proceso en el que se aplica, de manera regular, un conjunto de prácticas acordes para trabajar colaborativamente, en equipo y obtener el mejor resultado posible de un proyecto. En la Ilustración #10 se observa un diagrama clásico de esta metodología.

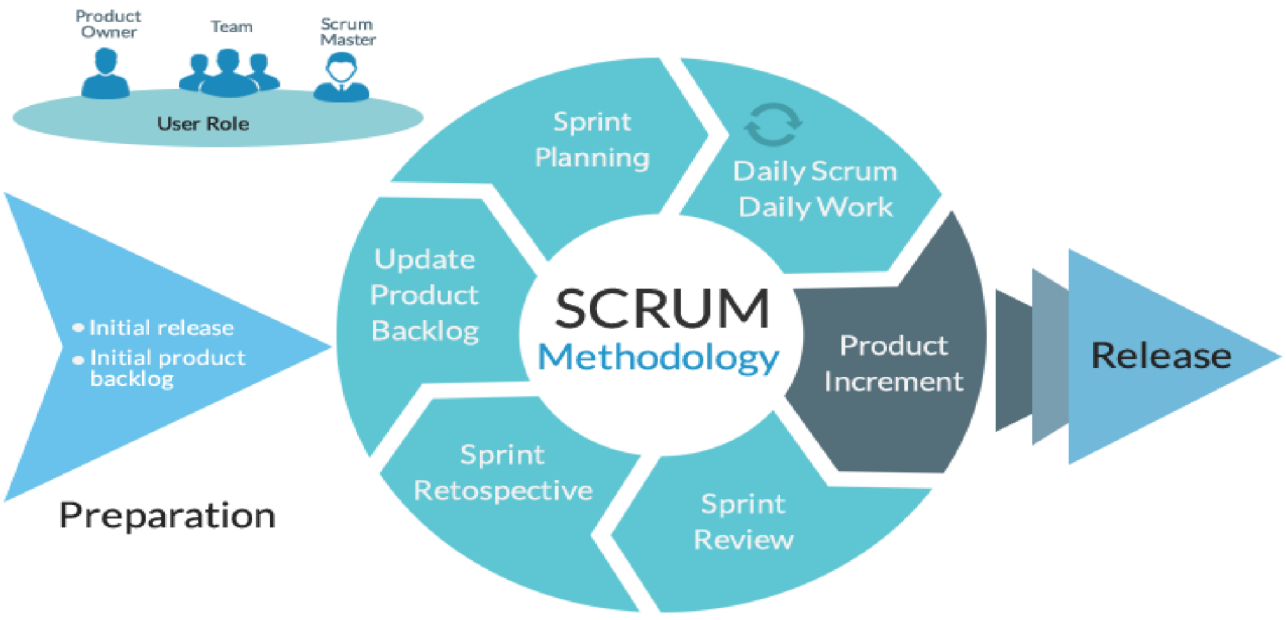


Ilustración 11- SCRUM
Fuente: https://blog.wearedrew.co/ventajas-y-desventajas-de-la-metodologia-scrum

Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio sobre la manera de trabajar de equipos altamente productivos. Aunque, en Santander Tecnología, esta metodología recién se está implementando y se utilizan muchos aspectos que ayudan al desarrollo de calidad del producto en tiempo y forma.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener rápidos resultados. Es debido a esto que los requisitos son cambiantes o poco definidos, cuando la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

El equipo para desarrollar el proyecto cuenta con un *producto owner (*participante del lado del usuario final), un *scrum master*, que es el facilitador del proyecto, el encargado de que el equipo pueda trabajar correctamente y cumplir sus objetivos. Además, de los desarrolladores, por un lado, se cuenta con la participación del arquitecto, quien trabaja de manera *cross* en las demás iniciativas de la gerencia; por otro lado, tres desarrolladores *backend* (desarrollo de la API) y, por último, un desarrollador *frontend (*quien realiza las pantallas visuales al usuario consumiendo la información de la API).

Con la metodología Scrum, se cuentan con *sprints (*intervalos de trabajo), los cuales son de dos semanas y los mismos contarán con diferentes etapas, en el que el resultado será un incremento o mejora al desarrollo hecho.

Bajo esta metodología, se contarán con *dailys (*reuniones diarias de no más

de quince minutos), donde se explicará lo realizado el día anterior, lo que se hará durante el día y si existiera algún problema que complicara el desarrollo de una labor.

Por otro lado, al comienzo de cada sprint, se realizará la *planning*, que corresponde al planeamiento de las tareas a ejecutar durante el sprint. Con ello, se completará el *backlog (*pizarra con tareas), con las funcionalidades y errores a desarrollar o corregir durante el siguiente intervalo de trabajo. Para la selección de las tareas, se tendrá fuertemente en cuenta la decisión y opinión del sector de negocios, comerciales y el punto de vista fuera de tecnología, la que es representada por el *producto owner.*

Al final de cada sprint, se realizarán la retrospectiva y *review*, en las que se analizan los resultados obtenidos, puntos, tareas o actitudes a mejorar o llevar a cabo. De este modo, se tendrán en cuenta las acciones que se deberán mantener debido a los resultados positivos e indagar qué se debe incorporar o adoptar para la siguiente iteración.

El resultado de todo este trabajo es una *release,* es decir, un producto incrementable que puede ser mostrado al cliente y usuario final con sus respectivas mejoras y funcionalidades nuevas.

2.1.10
Swagger

Swagger es una herramienta que facilita la documentación y visualización de la API. Para ser activada se configura mediante una simple *Annotation* (anotación de Spring). En la Ilustración #12 se observa el logo.

Ilustración 12- Swagger
Fuente: https://www.logolynx.com/topic/swagger

Esta librería, una vez activada, provee de una página a la cual se accede mediante la URL del proyecto /swagger-ui para poder observar todas las funcionalidades que brinda la API. Esto es muy útil como documentación para presentar a otros equipos que quieran utilizar dichos recursos. En la Ilustración #13 se muestra una vista generada por Swagger para algunas de las funcionalidades relacionadas a los envíos.

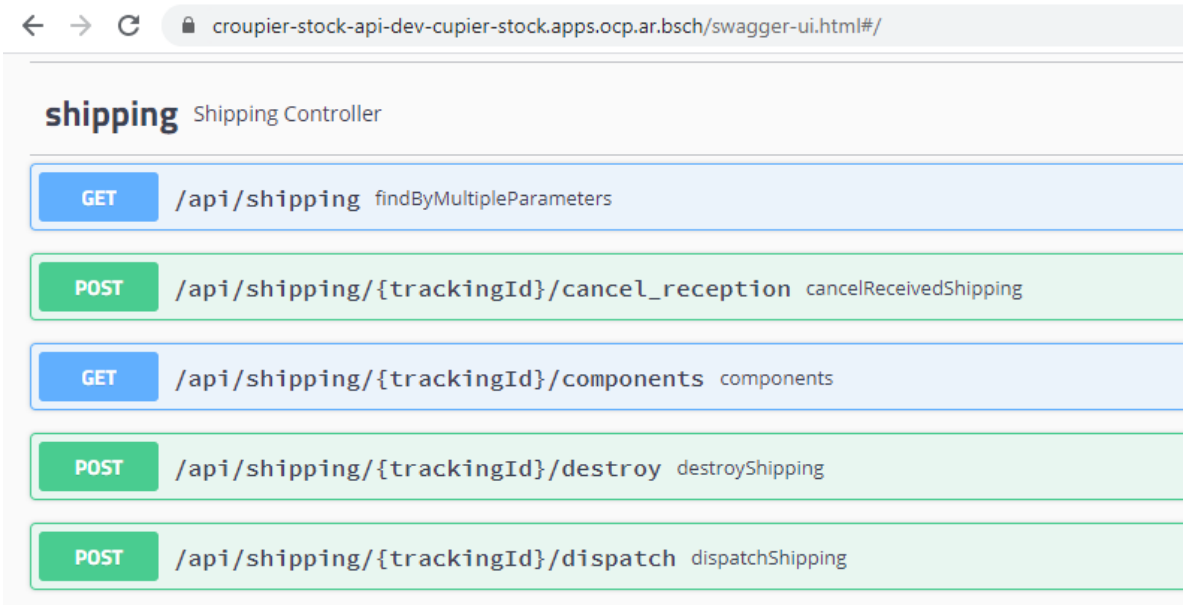


Ilustración 13- Swagger
Fuente: Elaboración propia basada en la práctica - proyecto en OCP ambiente de desarrollo

Posteriormente, se explicará el funcionamiento de cada API expuesta en la anterior imagen.

2.2 Arquitectura

La API, para poder contar con todas sus funcionalidades y servicios, necesitará comunicarse con otros sistemas externos: propios del banco y fuera del mismo. En los siguientes apartados se hará hincapié en la estructura y arquitectura en sí de la solución.

2.2.1 Estructura de la API

Al usar el gestor de dependencias Maven, se obtiene cierta estructura por defecto para los directorios del proyecto, sin embargo, en la Ilustración #14 se observa la configuración final de carpetas y módulos.

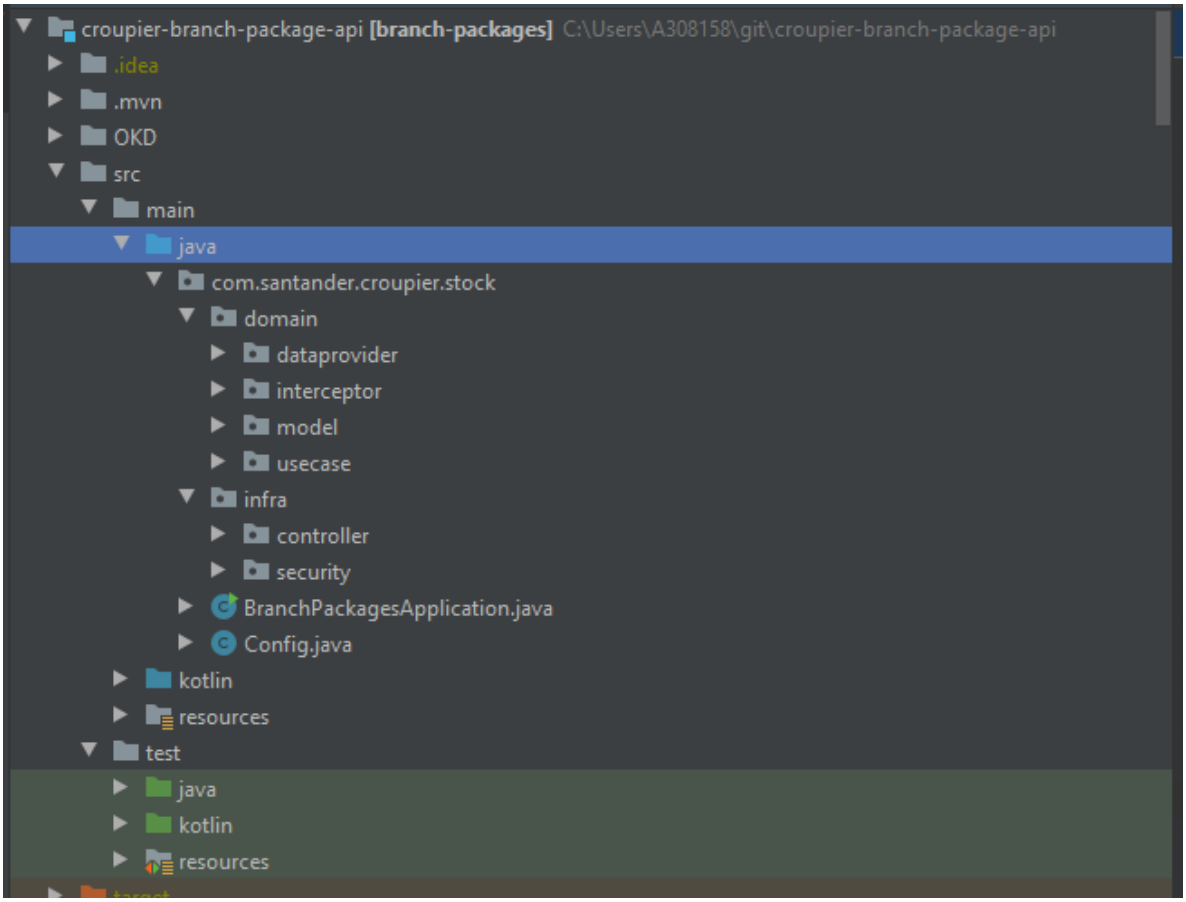


Ilustración 14- Arquitectura API

Fuente: Elaboración propia basada en la práctica – proyecto levantado localmente

La carpeta OKD presenta los archivos de configuración para poder deployar las aplicaciones en la nube (Openshift). Entre otros, presenta *templates* (plantillas), que son provistas por los DevOps para el correcto funcionamiento, otros archivos con parámetros, secretos, configuraciones de los servicios, etc.

Por otro lado, el código fuente se divide en dos grandes estructuras: domain e infra. Domain es el directorio donde se encuentran las clases los módulos con las clases de objetos de dominio, modelos e interfaces. Además, tiene las interfaces, llamadas providers, que se comunican con el módulo de infra, las cuales son usadas por las clases llamadas 'UseCases' (casos de usos), donde se encuentra la lógica de negocio. En la Ilustración #15 se observan los submódulos.

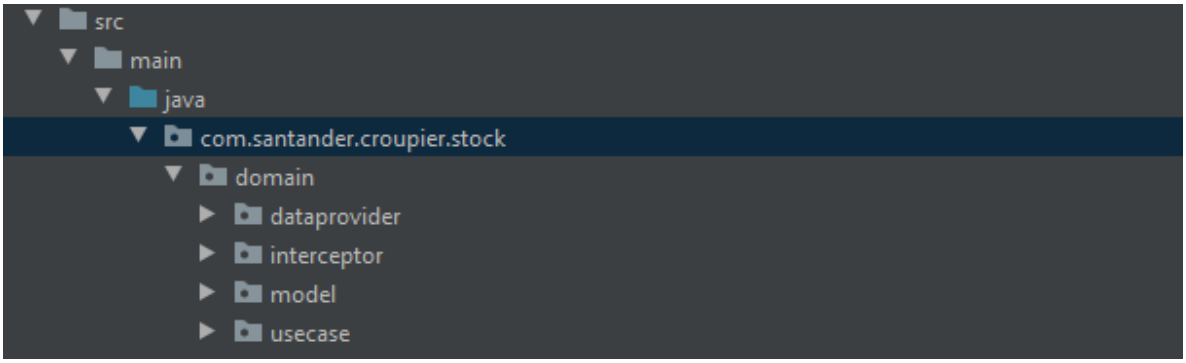


Ilustración 15- Arquitectura API

Fuente: Elaboración propia basada en la práctica – proyecto levantado localmente

Por su lado, infra posee la estructura de controladores, rutas y apis que se "exponen" y son consumidas por el frontend (la parte visual de los usuarios) y demás aplicaciones interesadas en usar los servicios desarrollados. Esto se observa en la Ilustración #16.

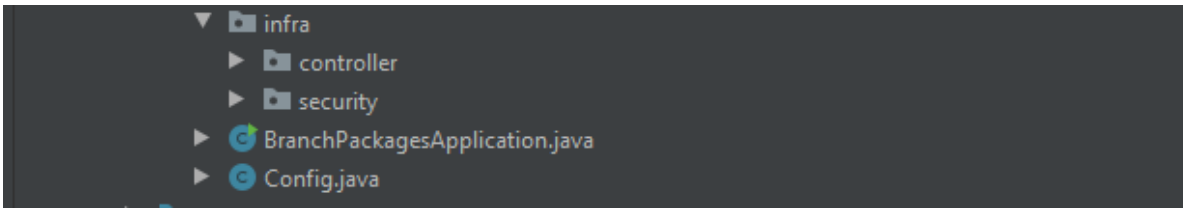


Ilustración 16- Arquitectura API
Fuente: Elaboración propia basada en la práctica – proyecto levantado localmente

Estos controladores, poseen, gracias al framework mencionado anteriormente SpringBoot, inyecciones de dependencias que son los 'UseCases' del domain, invocando a los mismos y ejecutando su lógica. Por otro lado, en infra también se maneja la seguridad de la aplicación, con un submódulo dedicado a su configuración e interacción con una librería provista por el Banco Santander, para realizar autenticaciones llamada Security Provider. Dicho módulo se utiliza para autenticar los usuarios de las sucursales, asignarle roles y permisos para determinar qué acciones pueden ejecutar y llevar a cabo y cuáles no.

Otra de las estructuras que se observa, es la de test, donde existen todas las clases y recursos para ejecutar las pruebas unitarias y de integración que posee la aplicación (desde probar un caso de uso concreto hasta una prueba integral o desde un controlador hasta un acceso a base de datos).

Corresponde mencionar también el directorio kotlin, el cual agrupa el código desarrollado en dicho lenguaje y, como se explicó en el apartado sobre este lenguaje de programación, puede integrarse sin inconvenientes con Java. En esta ruta se encuentran las clases que sirven como clientes para comunicarse con otras aplicaciones (más adelante se explicará el ecosistema de aplicaciones que existe).

2.2.2 Arquitectura de la API

La aplicación integral cuenta con el diagrama de la Ilustración #17.

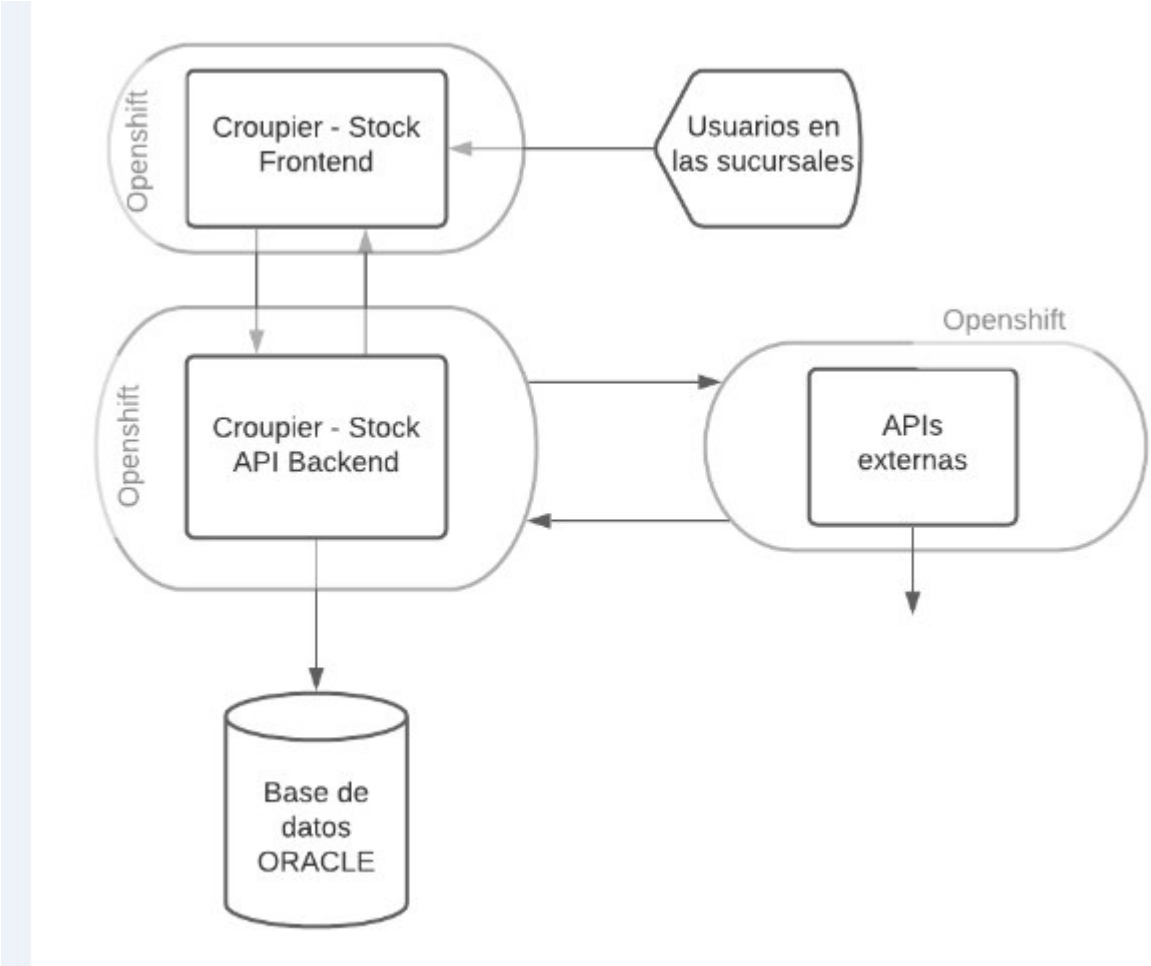


Ilustración 17- Arquitectura API

Fuente: Elaboración propia basada en la práctica

Los usuarios finales, operarios del Banco Santander en las sucursales, utilizarán el frontend de la aplicación, que también está deployada en la nube. Este

frontend consume e invoca el backend, la API que es tema de desarrollo de la Práctica Profesional Supervisada. El backend maneja la lógica de negocio y servicios, interactuando con la base de datos Oracle y con aplicaciones externas, que existen dentro de la gerencia Medios de Pago y fuera de ella.

2.2.3 Ecosistema de aplicaciones

La aplicación de Croupier – Stock surge dentro de la Iniciativa Delivery (una gerencia cuenta con varias iniciativas), por lo que existe cierta retroalimentación entre diferentes APIs o sistemas dentro de la misma gerencia. En la Ilustración #18 se observa el diagrama de comunicaciones dentro de la gerencia de Medios de Pago enfocado en la Iniciativa de Delivery.

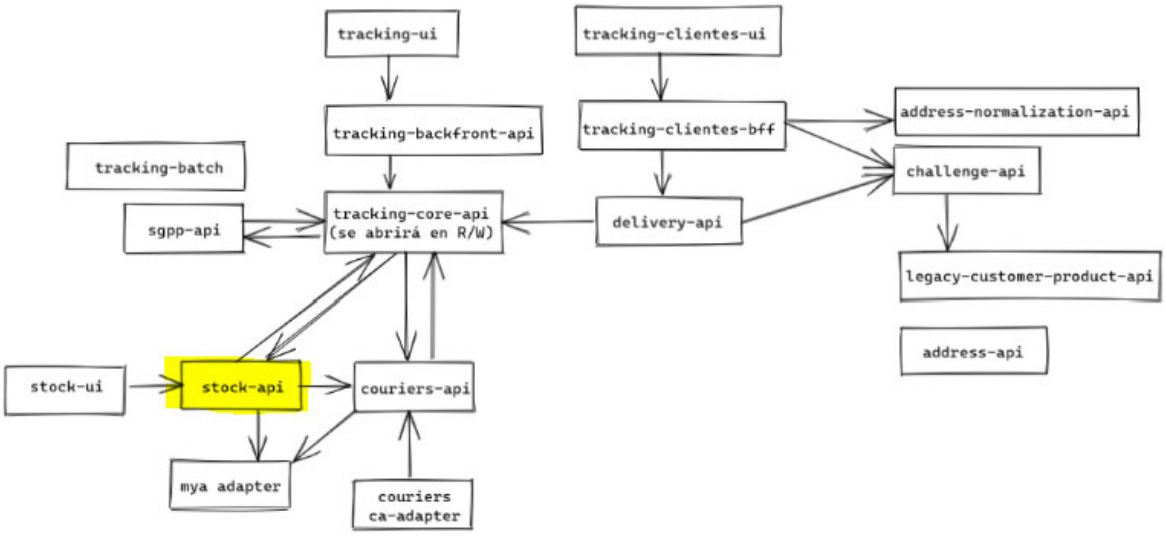


Ilustración 18- Arquitectura API
Fuente: Elaboración propia basada en la práctica.

Como se indica en la Ilustración #18, la aplicación de Stock es tan solo una parte de todas las interacciones que existen entre las diversas APIs. La más directa es la siguiente: el backend de la aplicación interactúa con Couriers-Api, que es la encargada de la interacción con los diferentes couriers (hoy en día Andreani y próximamente Correo Argentino). Con dicho sistema se realiza el intercambio de información respecto a los envíos, siendo éste el que notifica 'pre arribos' a las diferentes sucursales del Banco Santander a través de la aplicación. Del mismo modo, desde Stock se obtiene información sobre los diversos envíos a través de ellos. Por otro lado, Stock – Api interactúa con tracking-core-api y con dicho aplicativo se realiza un ida y vuelta intercambiando información respecto a los envíos (componentes y estados), siendo el sistema el que notifica los cambios de estado que sufre un envío. Por su parte, la aplicación Tracking cuenta con el recién mencionado core, y su frontend. Es la aplicación central de la inicativa delivery, que agrupa una enorme cantidad de información para tener el registro punta a punta de un envío: desde su creación en archivos batch (archivos por lotes), hasta consultas online a los diferentes sistemas de la Iniciativa Delivery. Por último, Croupier – Stock también interactúa con el mya-adapter, que es un módulo que se comunica con un aplicativo para enviar notificaciones a los clientes (emails), avisando del estado de sus productos: *se encuentra en sucursal* y es *próximo a ser destruido*, entre otros.

Además, el backend también se relaciona con APIs y sistemas fuera de la Iniciativa Delivery, como son los sistemas de tarjetas, que se utilizan para recolectar cierta información respecto a los clientes. Por último, los consumidores externos de la Gerencia de Medios de Pago son aplicaciones utilizadas por oficiales comerciales. En sus sistemas, consumen los servicios para notificar y saber si un cliente a quien están atendiendo y llega a una sucursal, posee productos en ésta o

en otra, siendo un punto de impacto altamente positivo, ya que el objetivo inicial siempre fue mejorar la atención de los consumidores finales.

2.3 Metodología de Desarrollo

Como se mencionó con anterioridad, se trabajó bajo la metodología ágil Scrum. Por ende, al comienzo de cada sprint, se definirán las tareas a desarrollar durante el mismo, dándole un valor basado en esfuerzo a cada una. En ese momento se podrán asignar las mismas o ir tomándolas durante el transcurso. Existe también, bajo esta metodología, la posibilidad de refinar los requerimientos para tener una mejor definición de éstos y facilitar su desarrollo. Por ejemplo, una reunión de refinamiento contará, mínimamente, con un representante comercial y un desarrollador, donde se intentará definir con el mayor detalle posible un requerimiento o funcionalidad.

Una vez comenzada una tarea, se creará una rama a partir de *development* con la nomenclatura *feature/{nombre de funcionalidad}*. En la misma se comenzará con la programación, despejando dudas de negocio con el producto owner o el scrum master, y, las técnicas, con el arquitecto o los demás desarrolladores del equipo.

Se tendrán en cuenta las buenas prácticas de programación, como TDD (*test driven development)*, que refieren al desarrollo guiado por pruebas, con lo que se intentará realizar primero los *tests* o pruebas unitarias y luego el desarrollo. Tener una buena cobertura de tests del código desarrollado es aconsejable para su mantenimiento. Además, con el incentivo del arquitecto y los desarrolladores más experimentados, se busca aplicar *clean code*, que remite a una serie de prácticas

para tener el código de la mejor manera posible, más mantenible y escalable. Por ejemplo, eliminando código repetido, mejorando siempre que sea posible una clase con la que se trabaje, desacoplando funcionalidades, entre otros.

Una vez desarrollada la funcionalidad correspondiente (código + test unitarios + test de integración + pruebas con el *frontend*), la misma se subirá al repositorio remoto y se solicitará un *merge request* para llevar ese código a la rama de *development*. El *merge request* será analizado por uno o varios desarrolladores, en lo que se llama una revisión de código. Esta funcionalidad sirve para analizar posibles mejoras o sugerencias en el código desarrollado de manera individual y así tener una doble validación de lo que se efectúa.

Durante cada solicitud de *merge request,* se ejecutará un *pipeline* que compilará, ejecutará los test y analizará la calidad del código. Por otro lado, una vez *mergeado* algún requerimiento a las tres ramas principales anteriormente mencionadas, correrá otra secuencia de tareas adicionales a las tres primeras, generará una imagen y *deployará* la misma en los ambientes correspondientes, siendo obligatorio que todos los pasos ocurran correctamente (una falla en la ejecución de los *test* finalizará el pipeline y no se hará el *deploy*).

Cuando el código (funcionalidad o *fix)* se encuentre en el ambiente de desarrollo, el *producto owner* procederá a probar los casos de uso que crea correspondiente, pudiendo realizar pruebas punta a punta o los casos particulares (al no contar con un equipo de testing serán fundamentales las pruebas hechas por los desarrolladores).

Otra instancia a tener en cuenta, es el rol que tendrán los *producto owners*, ya que los *merge request* de la rama *staging* a *master* (lo que significa un pasaje a

producción), solamente podrá ser aprobado y desplegado por ellos, una vez que consideren que esté todo en condiciones.

2.4 Autenticación

Los empleados de las sucursales del Banco Santander para poder utilizar la aplicación Croupier – Stock deberán estar autenticados en todo momento. Para ello, se utilizó una librería provista por el sector de seguridad informática que realizará esta acción: el Security Provider. En la Ilustración #19 se observa la pantalla de *login*.

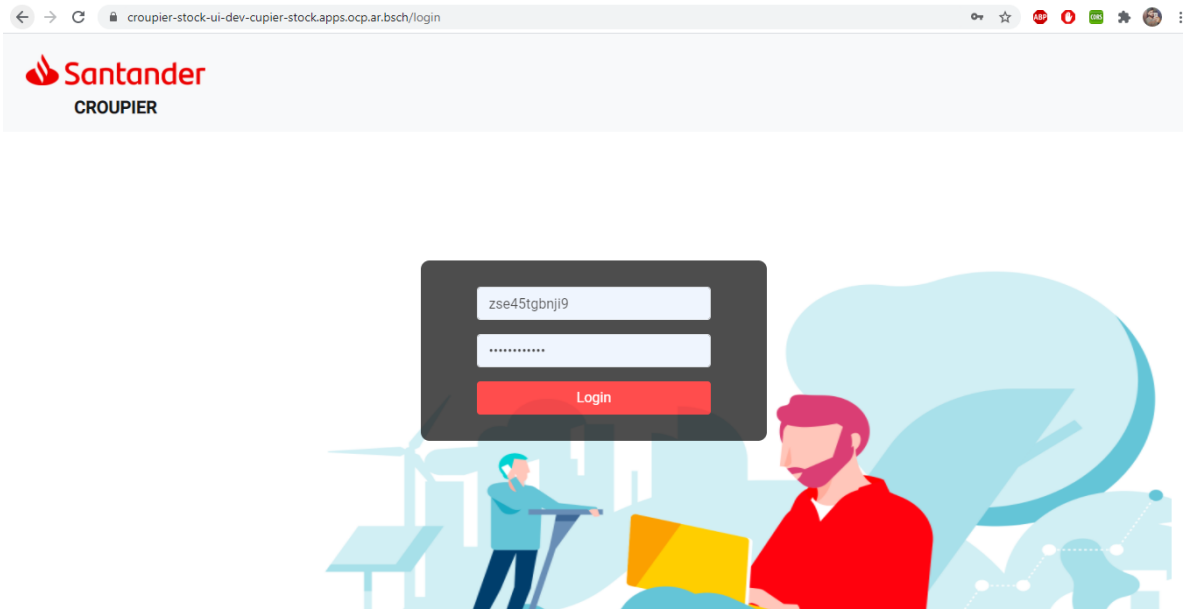


Ilustración 19- Pantalla de Login
Fuente: aplicación en ambiente de Desarrollo

Una vez que el usuario se autentique con credenciales válidas, se le devolverá al mismo una serie de roles con los permisos y acciones que podrá realizar en torno a la aplicación (todo esto es transparente para los usuarios).

2.5 Patrón de diseño State

Como se ha mencionado, el objetivo de la API gira en torno al manejo y gestión de los envíos dentro de las sucursales del Banco Santander.

Una de las bases sobre la que se implementa la lógica de la aplicación, es el cambio de estados que puede sufrir un *shipping* dentro de cada sucursal, permitiendo determinadas permutaciones de estados, las cuales pueden verse en la Ilustración #20.

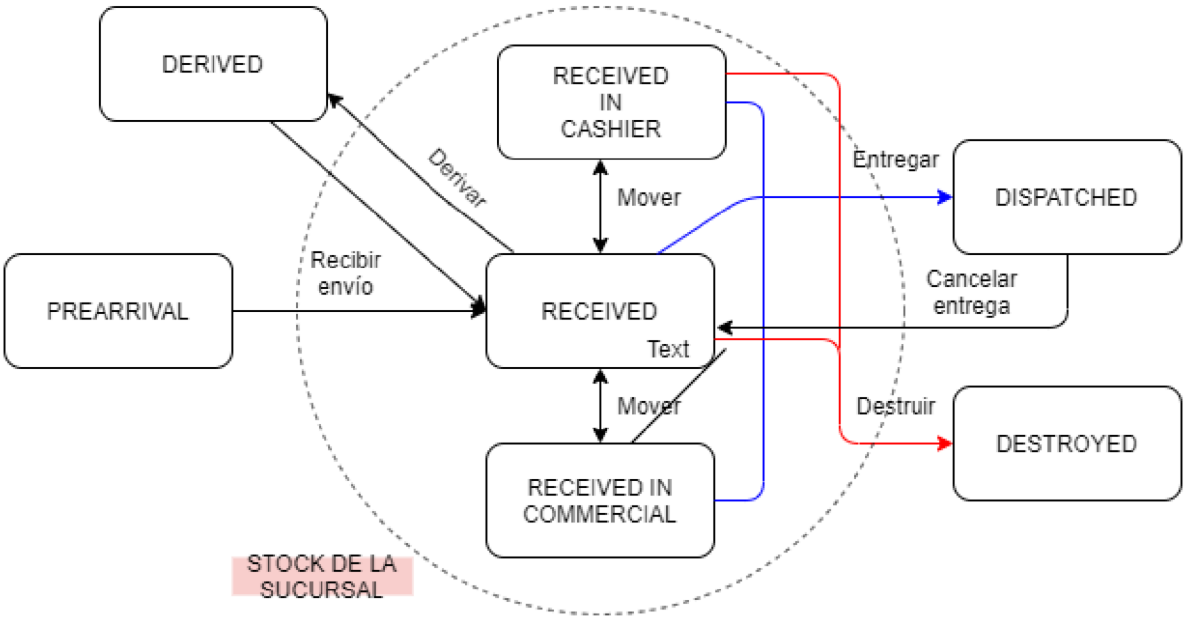


Ilustración 20- Estados de un envío
Fuente: Elaboración propia basada en la práctica

Como se observa en el gráfico anterior, un envío puede atravesar siete estados en nuestro dominio (existe también el estado EMPTY, pero es un estado temporal interno al momento de crear un envío).

El *backend* de la aplicación (la API) tiene una base sobre el Patrón de Diseño State. El mismo consta del siguiente diagrama UML (Diagrama de Clases) que se observa en la Ilustración #21.

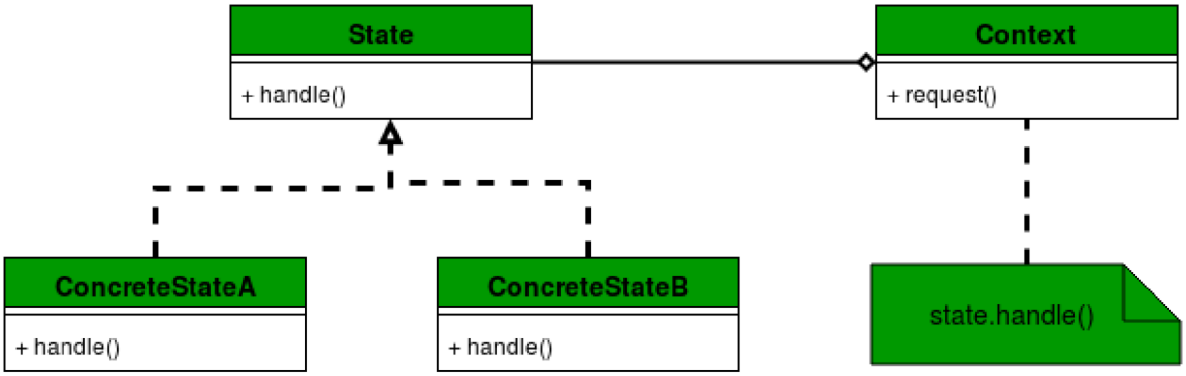


Ilustración 21- Patrón State
Fuente: https://www.geeksforgeeks.org/state-design-pattern/

La base de dicho patrón consiste en el comportamiento en base del estado interno de un objeto, en el presente caso, los envíos. Se tiene una interfaz con funcionalidades que serán implementadas por los estados concretos (Pre arribado, recibido, destruido, etc.). La motivación para implementar esta solución fue la escalabilidad que permite, ya que, en un primer momento, la cantidad de estados

de un envío era simplemente de tres: *recibido, destruido o entregado* y dicha definición varió en el tiempo por necesidades de las sucursales, añadiendo más estados hasta llegar a tener siete. Gracias a este patrón, agregar más es menos costoso, teniendo simplemente que agregar un estado nuevo que implemente la interfaz común y sus métodos: permutar a otro estado en específico y obtener un reporte propio. La desventaja que presenta esta solución es que se incrementa el número de subclases, ya que se debe tener una para cada estado en concreto.

2.5.1 Estados de un envío

El estado PREARRIVAL es el estado de “Pre-arribo”. El mismo refiere a un envío que se encuentra llegando a la sucursal de destino o que fue escaneado (los empleados utilizan lectoras de códigos de barras para facilitar los ingresos) para su posterior ingreso, estando ya físicamente en una sucursal. La API de *Couriers* (existente en el ecosistema de APIs de la Ilustración #16), notifica eventos, entre otras cosas, de las diferentes empresas de correo con los cuales trabaja la entidad bancaria. Alguno de estos eventos es notificado a un *endpoint* (ruta que expone un servicio) que pre arriba los envíos en las sucursales correspondientes, ahorrando mucho tiempo para los operarios de las sucursales, ya que este proceso es automático y transparente a ellos. Durante dicha operación, el envío recupera información en los diferentes sistemas y persiste el registro en la base de datos de la aplicación. Un envío en PREARRIVAL puede permutar solamente al estado RECEIVED (recibido) en cualquier sucursal, sin importar si este tenía otro destino original. Esto se debe a que existen casos en que un paquete llega a otra sucursal por error y el sistema debe permitir ingresar la pieza de todas formas. Una vez que

un cajero realiza la recepción de un paquete pre arribado, el *frontend* le avisará a la API para que cambie el estado de dicho envío, guardando en la base de datos la permutación que se realizó (en la tabla de logs), actualizando el estado a RECEIVED, la sucursal donde se recibió y el legajo del usuario que efectuó dicha acción.

El estado RECEIVED es sinónimo de “recibido en el tesoro”. Un envío en esta instancia ya se encuentra dentro de la sucursal y con la información necesaria y lista para ser trabajado con él. En esta instancia, se agrega quién fue el usuario que recibió la pieza y en qué sucursal.

Como se observa en la Ilustración #18, en el estado RECEIVED es donde puede existir la mayor cantidad de permutaciones o pasajes de un envío, ya que el mismo puede pasar a DERIVED (derivado), RECEIVED IN CASHIER (pasar a manos de un cajero), RECEIVED IN COMMERCIAL (tenerlo un ejecutivo comercial), DESTROYED (destruido) o DISPATCHED (entregado).

Los estados RECEIVED IN CASHIER y RECEIVED IN COMMERCIAL, juntos con el RECEIVED forman el stock real que posee una sucursal. La recepción se realiza al comenzar cada día (pero puede hacerse en cualquier momento sin restricciones), donde se retiran los paquetes del tesoro y se llevan a línea de caja. Desde allí, los usuarios entregan sus productos a los clientes que llegan en busca de ellos. En caso de que un envío necesite una firma o exista cierto inconveniente con el cliente que requiera verificación por parte de un oficial comercial, el ***shipping*** se envía al sector comercial, donde será trabajado por un oficial y el cliente, pudiendo ser entregado ahí mismo o devuelto a la línea de caja o el tesoro. En la Ilustración #22 se observa el flujo de entrega de un envío.

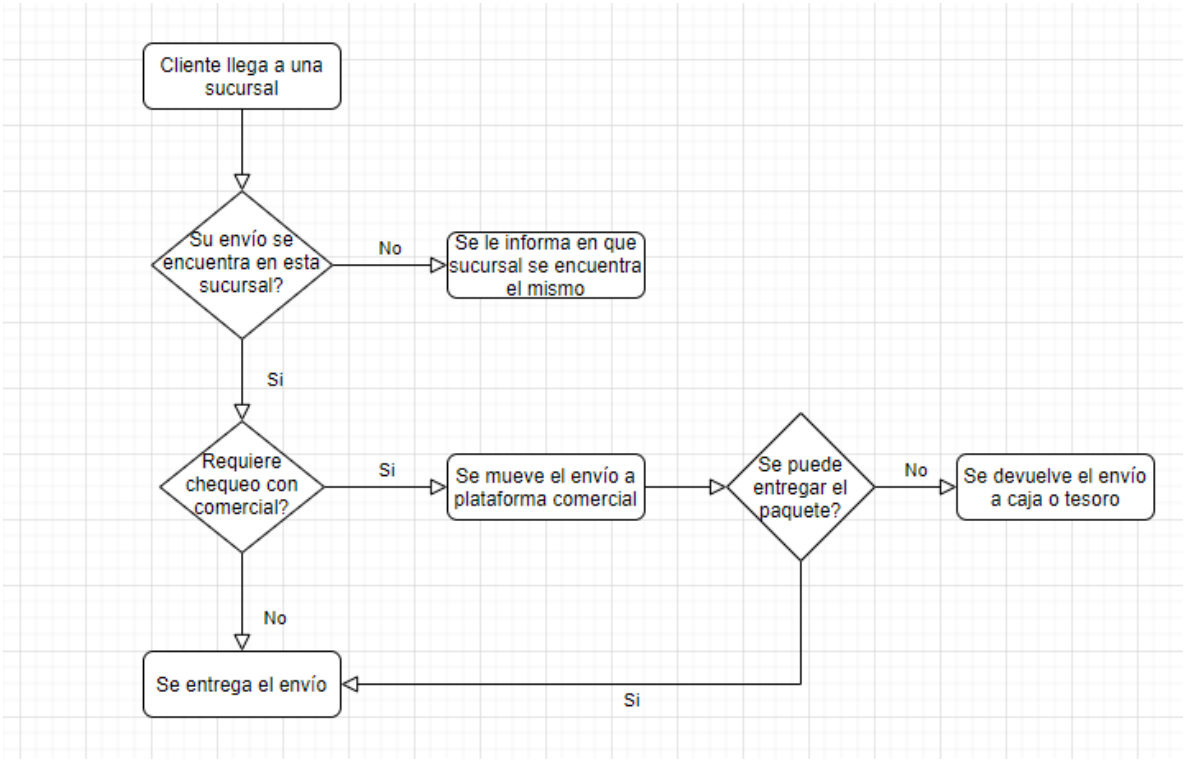


Ilustración 22- Flujo de entrega de envío
Fuente: Elaboración propia basada en la práctica

Por otro lado, el estado DERIVED significa que un envío fue derivado. El mismo se utiliza para expedir, de forma lógica, un paquete de una sucursal a otra, para que luego se lleve físicamente. Esta situación puede darse en caso de que el *Courier* de correo deje por equivocación un paquete en una sucursal errada (el sistema permite ingresar la pieza de todas formas). En caso de necesitarse que el mismo sea enviado a otra sucursal, se deriva para luego ser recibido por la nueva u otra, siendo ésta la única permutación de estado permitida.

También, existe el estado DISPATCHED, el que representa uno de los estados “finales” (porque puede ser reversado). El mismo alude a la entrega de un envío al cliente que se presenta una sucursal del Banco Santander. En la Ilustración

#20 se muestra el flujo para una entrega, donde se debe tener en cuenta si el cliente necesita firmar o no algún documento para poder recibir su paquete. El estado DISPATCHED puede ser reversado en caso de error, siendo el encargado de la sucursal el único con el rol de poder realizar esta acción, anotando de forma obligatoria el porqué de esto. De esta forma, en las tablas de *logs* se mantiene la trazabilidad completa por si surgen temas relacionados con auditoría, ya que, en primera instancia antes de tener esta funcionalidad de reversa implementada, las mismas debían realizarse con un *update* (actualización) directa a la base de datos.

Por último, se cuenta con el estado DESTROYED, el cual hace referencia al estado destruido de un envío. Éste representa también un estado “final, ya que puede ser reversado dentro de las 24 hs (lo mismo con un envío DISPATCHED), además, que el usuario debe poseer el rol y permiso para poder realizar esta acción. Cabe mencionar que, dentro de las sucursales de la institución bancaria, los envíos que son recibidos y no son entregados durante los 30 días siguientes de corrido, deben ser retirados del tesoro y destruidos por cuestiones normativas y legales.

2.6 Integración *frontend – backend*

En el inciso anterior se comentó por los estados que puede pasar un envío dentro de una sucursal del Banco Santander, siendo esto el *core* (núcleo) de la API.

En relación con el *frontend* (parte visual e interactiva de los usuarios), el mismo fue realizado, principalmente, por un desarrollador especializado en estas tecnologías, por lo que las integraciones se realizaron con pruebas e interacciones con él y con la aprobación de los *product owners*.

A continuación, se mostrará un *flow* completo de la aplicación, con las pantallas que visualizan los operadores del banco y una explicación de la lógica que posee por parte de la API en el *backend.*

2.6.1 Login

Como se mencionó con anterioridad, la autenticación se realiza mediante el uso de la librería *security provider* provista por el sector de Seguridad Informática del Banco Santander. Los usuarios de Croupier- Stock deberán estar "logueados" en todo momento. En la Ilustración #23 se observa la pantalla de *login.*

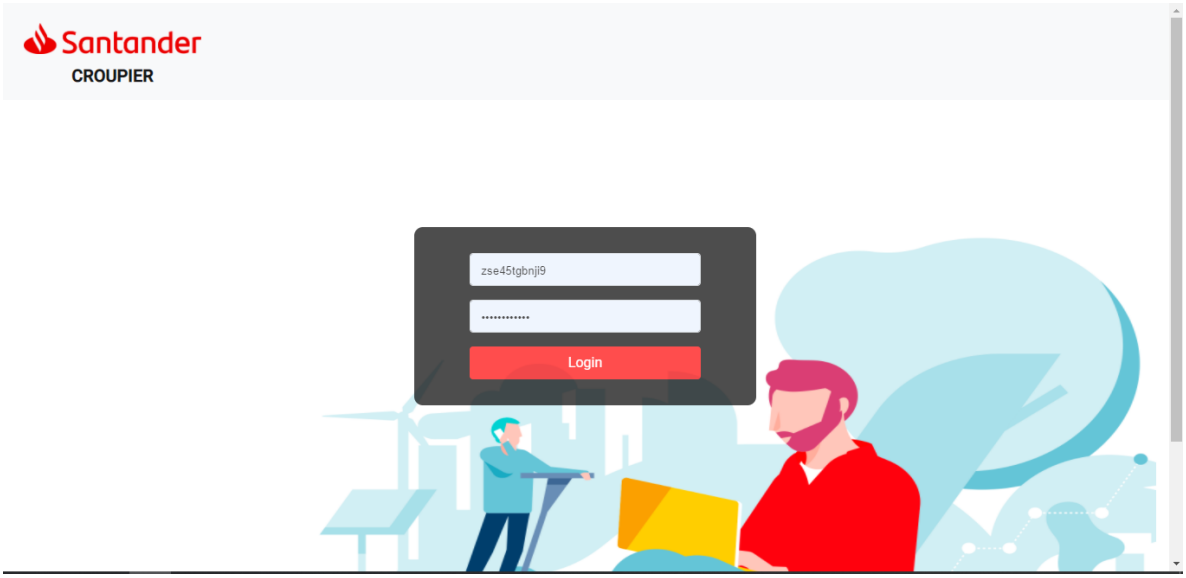


Ilustración 23- Pantalla de Login

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

El operador usará su legajo y clave personal (con la que inician sesión en Windows) para autenticarse. Adicionalmente, se utilizará el módulo de *Spring Security* para dicha funcionalidad.

2.6.2 Pantalla de recepción

Una vez *logueado*, el operador (según su rol), podrá navegar por las diferentes solapas de la aplicación, además, de observar en todo momento información sobre su usuario, sucursal y rol en la parte superior. Una de las más importantes es la pantalla de recepción (Ilustración #24). En ella, el usuario podrá recibir las piezas que llegan físicamente a la sucursal, que pueden ser escaneadas mediante una pistola (escáner de código de barras) y recibir dichos paquetes. La acción de escanear pre arriba el envío obteniendo la información de los diversos sistemas (ya puede haber envíos en dicho estado que son informados por el sistema *Courier* como se mencionó en el apartado de Ecosistema de APIs). Adicionalmente a la pistola, también puede hacerse un ingreso manual.

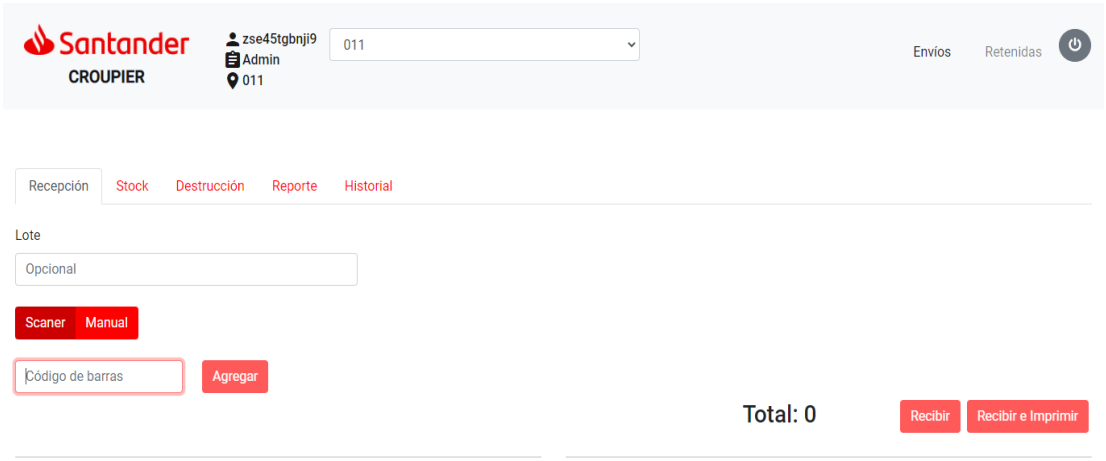


Ilustración 24- Pantalla de recepción

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Una vez que el cajero presione *Recibir o Recibir e Imprimir*, se guardará la información correspondiente a quien recibió la pieza, horarios y sucursal, entre otros, sumado a toda la *data* del envío. Ante esto, el envío pasará del estado pre arribado a recibido.

2.6.3 Pantalla de Stock

La pantalla de stock permite la gestión de todos los envíos que existen en cada sucursal del Banco Santander. Desde allí, los operadores podrán hacer los movimientos internos (a línea de caja – tesoro – plataforma comercial) como se explicó en la solapa de estados de un envío. Dicha pantalla se observa en la Ilustración #25.

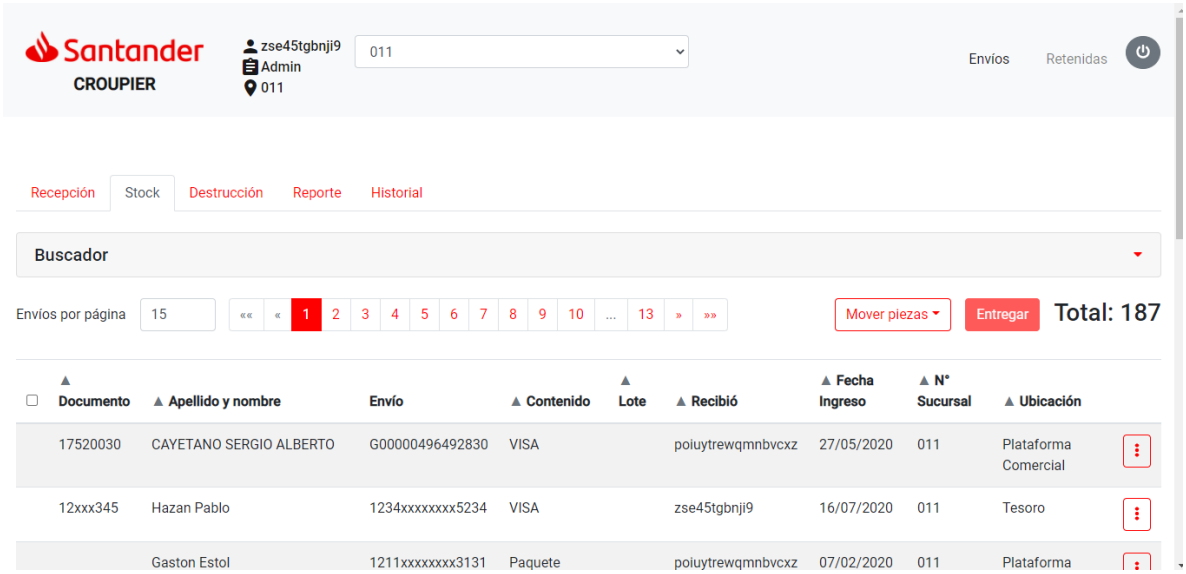


Ilustración 25- Pantalla de Stock

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Como se aprecia en la imagen anterior, se presentan todos los envíos que existen asociados a dicha sucursal. Se facilita un buscador para poder refinar resultados con los campos, con los filtros de documento del titular, apellido y/o nombre del titular, código de barra, número de cuenta, contenido (marca de tarjeta que posee), lote (identificador que se puede poner al recibir múltiples envíos), documento adicional (adicional de una tarjeta), apellido y/o nombre del adicional y la opción de buscar en todas las sucursales. La última opción existe ante el imprevisto de un error de entrega de un envío, si un cliente se acerca a una sucursal y su paquete no se encuentra en ella, buscándola con dicho filtro se puede saber en dónde está realmente. En la Ilustración #25 se aprecia el buscado mencionado.

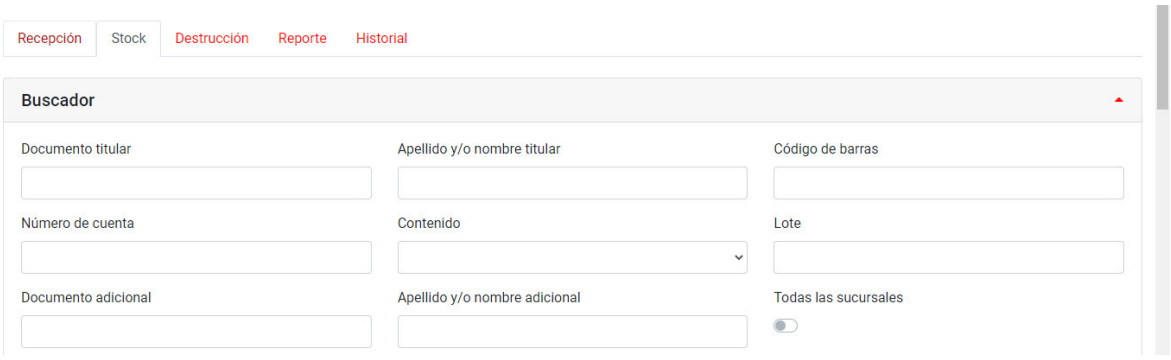


Ilustración 25- Buscador de envíos en stock

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Siguiendo lo descrito respecto a la Ilustración #25, se puede operar con los envíos en stock, moviéndolos internamente a diferentes áreas con la opción de "Mover piezas" o elegir alguna opción de las que presentan "los tres puntitos". Estas permiten realizar las acciones mencionadas anteriormente: entregar el envío (pasa a estado entregado), derivarlo a otra sucursal, cancelar la recepción (para que vuelva a estado pre arribado y pueda ser recibido por otra sucursal o la misma nuevamente). Adicionalmente, permite ver los componentes del envío (datos de las tarjetas en su interior).

2.6.4 Solapa de Destrucción

Esta solapa presenta la gestión relacionada a la destrucción de los envíos. La Ilustración #26 muestra el buscador de dicha pantalla con los filtros de días en stock, lote y contenido. Como ya se mencionó, los envíos que tengan 30 días o más en stock, deben ser destruidos y, en este caso, el envío pasará del estado *recibido*

a *destruido* y se permitirá la opción de *imprimir* los registros involucrados para ser guardados como evidencia.

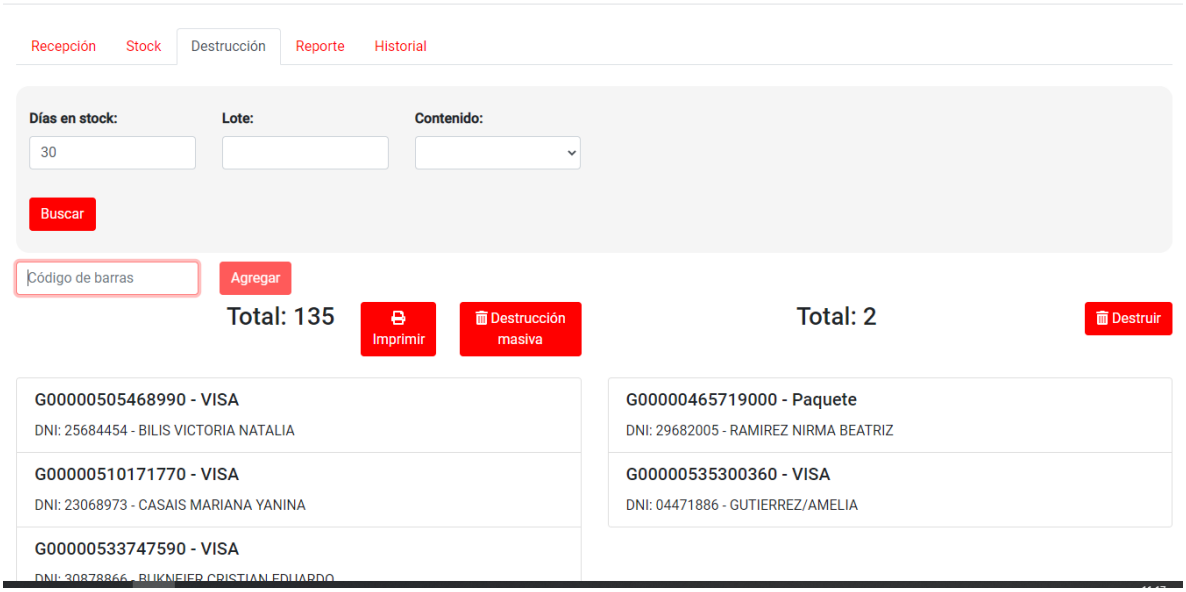


Ilustración 26- Pantalla de Destrucción

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

2.6.5 Reporte

Al final de la jornada, cada sucursal del Banco Santander necesita imprimir un reporte de la operatoria del día. La misma debe informar la cantidad de envíos que fueron *recibidos, entregados y destruidos*. En la Ilustración #27 se observa dicha pantalla, con la opción de buscar envíos por estado, lote o contenido. Además, se incluye el botón de imprimir para guardar la evidencia, que es muy necesaria por cuestiones de auditoría.

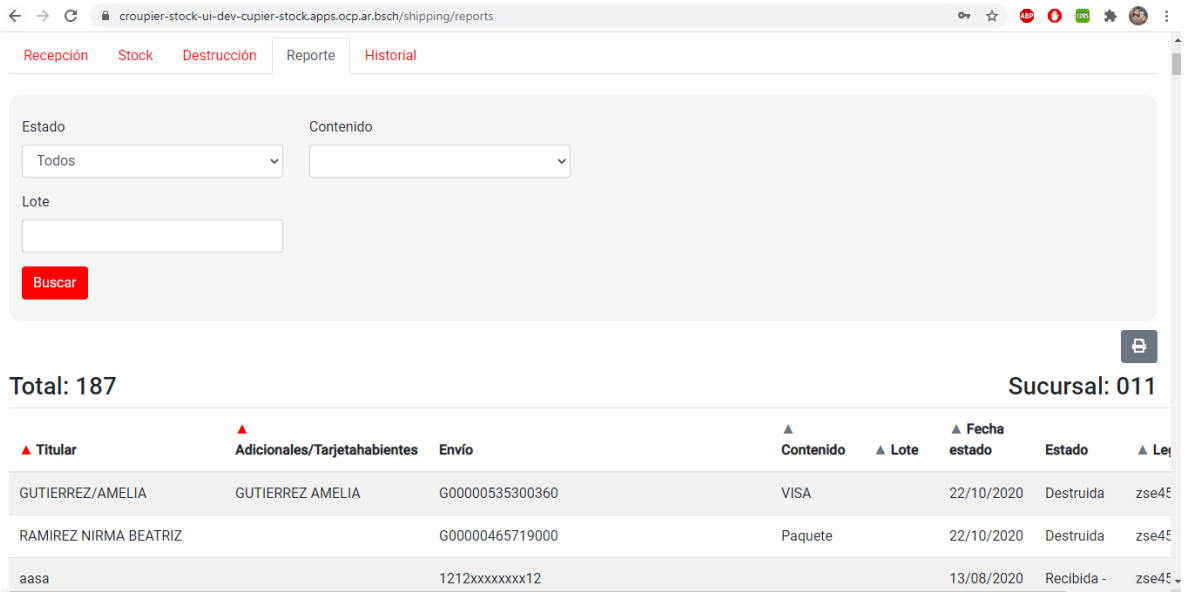


Ilustración 27- Pantalla de Reporte

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

2.6.6 Historial

Por último, la pantalla de historial permite ver todo el circuito que recorrió un envío desde el momento que ingresó a una sucursal del Banco Santander. Dicha solapa presenta, también, un buscador con todos los campos filtrables posibles: documento de titular, nombre y/o apellido de titular, código de barras, número de cuenta, contenido, lote, documento de adicional, nombre y/o apellido de adicional y estado del envío. Esto se aprecia en la Ilustración #28.

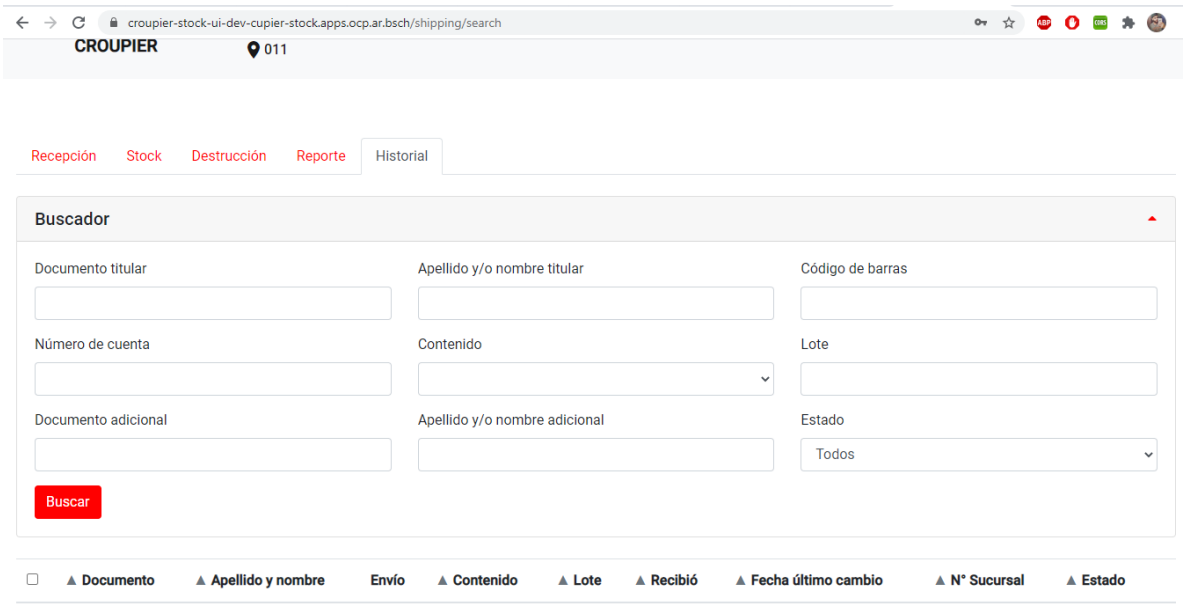


Ilustración 28- Pantalla de Historial

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Una vez realizada una búsqueda válida, se verán los resultados de ésta y una opción para observar el historial del envío seleccionado (Ilustración #29).

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

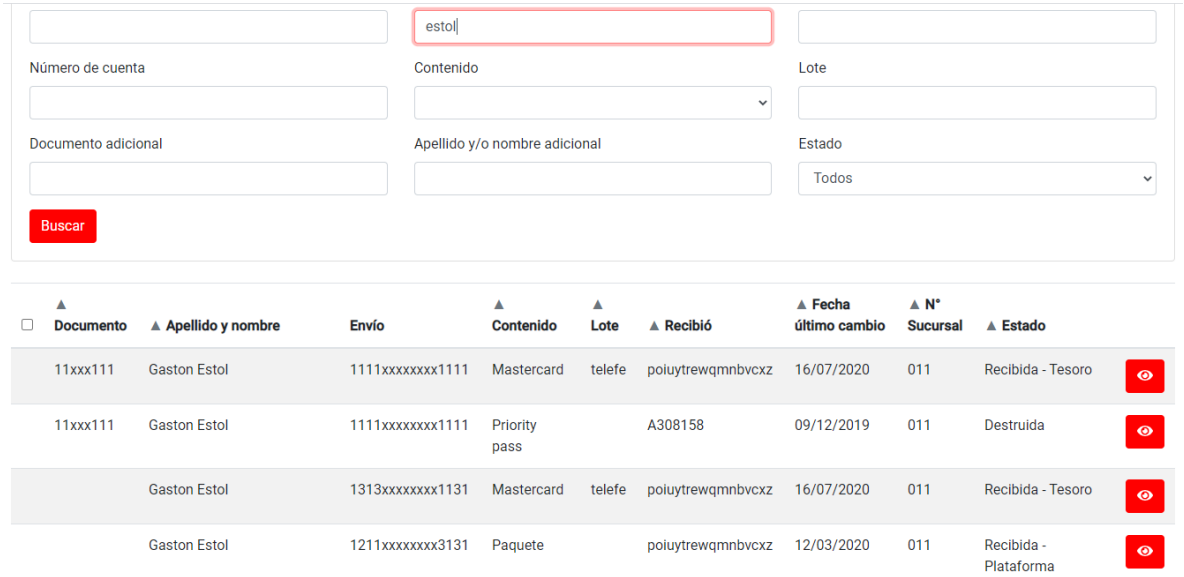


Ilustración 29- Pantalla de Login

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Tocando en la opción para ver el histórico de un envío, se accederá a otra pantalla que mostrará información respecto al cliente (nombre – apellido – documento) y al envío (número de envío – sucursal – contenido – estado – lote). Además, se verán los cambios de estado por los que pasó dicho paquete, indicando desde que estado se pasó a otro, día y horario de operación, usuario que realizó dicha acción y número de sucursal. Esto se observa en la Ilustración #30.

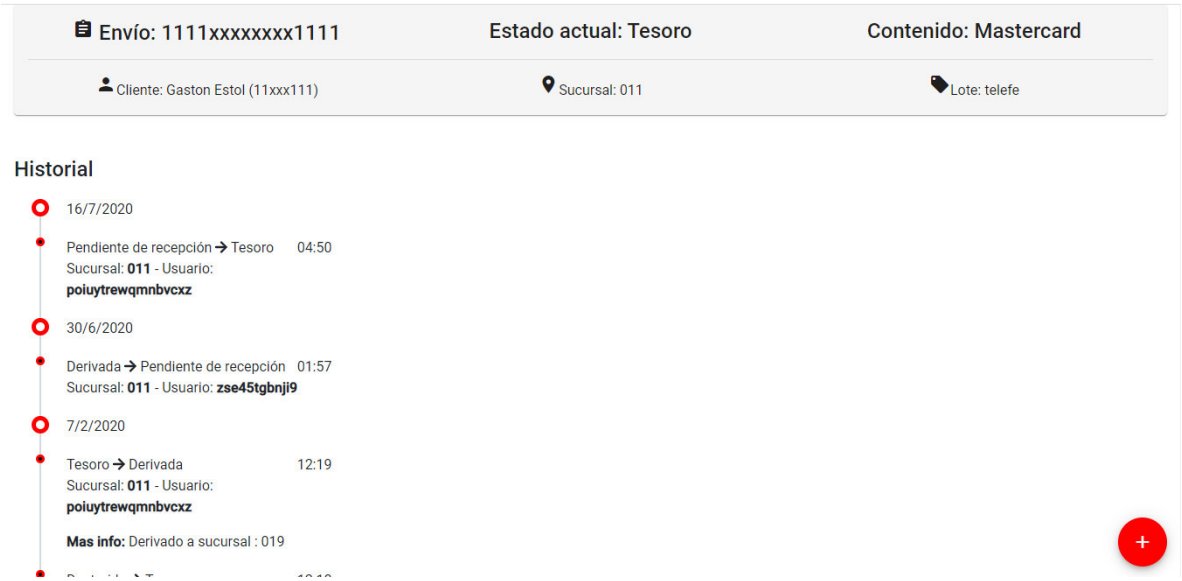


Ilustración 30- Historial de un envío

Fuente: Elaboración propia basada en la práctica – Aplicación en entorno de Desarrollo

Además de lo mencionado, se presenta un botón que facilita, según los roles permitidos, realizar diferentes acciones con el envío seleccionado. Las mismas son las señaladas en la solapa de stock: mover, derivar, entregar, destruir, cancelar recepción. Pero, se agregan dos acciones más que solo pueden ser realizadas por el supervisor de la sucursal: cancelar entrega y cancelar destrucción. Dichas opciones fueron un requerimiento pedido directamente por los usuarios en las sucursales, ya que, ante un error en una entrega o destrucción, esto debía ser corregido directamente mediante un *update* (actualización) a la base de datos. Por otra parte, debe considerarse que estas reversas, solamente pueden ser realizadas dentro del día que se entregaron o destruyeron.

3. Conclusiones

Para iniciar con las conclusiones se toma como punto de partida los objetivos propuestos en la presente Práctica Profesional Supervisada y que dan cuenta que durante el avance de la práctica los mismos fueron alcanzados. La aplicación resultante pasó por las diversas etapas de prueba, entre ellos, piloto con una sucursal, luego con unas pocas más y finalmente a toda la red del Banco Santander, reemplazando un sistema *legacy,* sistema heredado y desarrollado desde hace tiempo. Este cambio representó un hito muy importante para todo Santander Tecnología, sub-empresa encargada de los desarrollos, ya que el aplicativo y puesta en producción se realizó rápidamente, mejorando los indicadores que reflejaron la satisfacción de los clientes. Se fueron corrigiendo errores y pedidos de los usuarios para mejorar la experiencia en todo momento, siempre y cuando fueran realizables y prioritarios (como pedidos de la sucursal de Casa Central).

Por otro lado, se efectuaron múltiples integraciones entre sistemas internos de la entidad bancaria, entre ellos nuevos y ya existentes, lo que le dio un mayor valor agregado a la API. Se “marcó el camino” que siguieron otros aplicativos para el desarrollo y puesta en producción, con trabajo en conjunto con el equipo de DEVOPS como ser el desarrollo del pipeline de integración continúa, el reemplazo del despliegue de las aplicaciones en servers tradicionales por tecnologías en la nube (Openshift), entre los más importantes, siendo pioneros en el uso de estos.

Además, la metodología de trabajo fue eficiente, pudiendo implementar Scrum de una forma positiva. Sin embargo, al ser un proyecto que fue desarrollándose “sobre la marcha”, existieron muchos contra tiempos o cambios de definiciones que implicaron algunos defasajes u horas extras de trabajo. Por ejemplo, el universo de estados varió desde la primera definición “*recibido”* –

"entregado" – *"destruido"*, siendo muchos más los que finalmente quedaron. También se debió realizar *refactors* sobre la aplicación para aceptar nuevas funcionalidades que fueron surgiendo; además de los nuevos estados, reportes, submódulos, perfilamiento, entre otros.

Resumiendo, quedó un sistema aprobado y felicitado por los usuarios, pues mejoró la experiencia de los clientes de manera muy positiva. Es robusto, cuenta con seguridad y escalable, pero con varios y posibles puntos de mejora para incrementar. Algunas propuestas a futuro tendrán en cuenta el desarrollo de gestión de stock de tarjetas retenidas (las que retienen los cajeros automáticos), que es un desarrollo en proceso al momento de redactar este informe. El mismo será un nuevo módulo o solapa dentro de la aplicación original. Además, se tiene en vista una nueva funcionalidad, que será la gestión de chequeras, las cuales siguen un flujo diferente a los paquetes. Este desarrollo está previsto incluirlo en la aplicación a futuro o ser un aplicativo aparte. Por otro lado, se espera subdividir la aplicación con un "back del front", que implicaría separar lógica del *backend* en un *core* y un *backend* exclusivo para el *frontend.* El *core* tendrá la lógica de base de datos y la comunicación desde y hacia otros sistemas. Tener esto más el *backend* exclusivo para el *frontend* del aplicativo, permitirá separar lógica y tener un sistema mucho más mantenible.

3.1 Reflexión sobre las prácticas profesionales como espacio de formación

En el principio de la carrera, si uno no está bien informado sobre una de las labores de un Ingeniero en Informática, por ejemplo, ser un programador, cuesta mucho imaginarse con qué se puede encontrar en el ámbito profesional. Sin

embargo, cada materia a lo largo de estos años nutrió con conocimientos y prácticas que contribuyeron a formarnos y prepararnos para dicho momento.

Rescato algunos puntos clave como ser la primera aproximación a programar en “Fundamentos de Informática” y luego en “Algoritmos y Programación”, pero fue “Metodologías de Programación I” la materia que me hizo un *click* para comprender mejor sobre que podía llegar a tratar la labor de un desarrollador de software. No obstante, con tan solo el segundo año de la carrera en curso, pude conseguir, gracias a la facultad, mi primera pasantía en lo que sería el inicio de mi experiencia laboral, el primer puntapié para insertarme definitivamente en el mercado. A partir de ese momento la curva de aprendizaje fue exponencial, aprendiendo tanto laboral como académicamente.

Tener ejercicio en lo profesional profesional, también me facilitó cursar algunas materias como “Aplicaciones Móviles”, ya que estaba fortalecido de los conocimientos que serían necesarios para cursar dicha materia. Por otro lado, la dificultad de cursar y trabajar al mismo tiempo convierte todo en mucho más satisfactorio por el gran esfuerzo que eso conlleva, el que jamás hubiese sido posible sin el apoyo y el primer gran empujón que nos brinda la facultad, en absolutamente todos los años de cursada: promoviendo pasantías, brindando herramientas para defendernos profesionalmente (conozco muchísimos colegas de la UNAJ en prestigiosas empresas) y siendo un lugar para conocer gente excelente, tanto en conocimientos como humanas.

Finalmente, deseo agradecer a los tutores que me han acompañado durante la PPS y toda la carrera: los ingenieros Martín Morales y Jésica Guzmán, la tutora TAPTA Lía Lavigna. Y en especial la formación profesional y humana que me brindó la Universidad.

Bibliografía

- Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (1994) - Design Patterns.
- Juan Darer (2016) - Desarrollo de aplicaciones web - https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html
- Maria Benito (2013) - ¿Qué es la usabilidad web? - https://www.baeldung.com/java-clean-code
- Philipp Hauer (2020) - Clean Code with Kotlin - https://phauer.com/2017/clean-code-kotlin/
- ¿Qué es SCRUM? (2020) - https://proyectosagiles.org/que-es-scrum/
- ¿Qué son las API y para qué sirven? - https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces
- Spring Tutorial - https://www.baeldung.com/spring-tutorial
- State Pattern - https://www.baeldung.com/java-state-design-pattern