UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL SANTA FE

Informe de Proyecto Final de Carrera

Desarrollo de sistema de soporte a la producción para una empresa dedicada a la elaboración de quesos

Alumnos:

Milton Bernhardt, LU: 24003, mail: miltonbernhardt@gmail.com

Elias Brizi, LU: 24079, mail: eliasbrizi@gmail.com

Director de proyecto:

Ramos, Juan Carlos

Índice

1 - Introducción

1.1 - Temática

El proyecto tratado en el presente informe, nace a partir de la inquietud de un interesado de implementar un sistema que le permita llevar un seguimiento de la producción de su empresa dedicada a la elaboración de quesos. Éste sistema, reemplazaría la forma actual en la que se realiza el seguimiento de la producción.

1.2 - Problema

La empresa lleva un control de la producción mediante el uso de una planilla de cálculo, lo que, además de resultar en un proceso engorroso y poco amigable con el usuario, presenta varios problemas de consistencia, así como también ineficiencias a la hora de cargar datos o consultar información.

Por un lado, no es posible el acceso concurrente a la planilla, lo que genera problemas si más de un usuario al mismo tiempo la requiere para trabajar. Dentro de la misma planilla se lleva información desde la producción, hasta los clientes y los pedidos que se debían confeccionar, por lo que sí, por ejemplo, se está cargando en la planilla la producción del día, otro usuario debe esperar a que la misma esté libre para poder cargar un pedido que está listo para ser despachado.

Por otra parte, se requiere agilizar el proceso de carga de datos, haciendo uso de formularios bien diseñados, que minimicen la ocurrencia de errores a la hora de la carga de datos.

Otro punto importante, es el hecho de que al manejarse tantos procesos de la empresa mediante un único archivo, y que todo el personal que tenía que cargar información debe tener acceso al mismo, no existe confidencialidad de la información. Por ejemplo, los operarios de producción pueden fácilmente visualizar información que no les corresponde, como los clientes de la empresa.

1.3 - Objetivo general

El objetivo principal es proveer una solución en un sistema de información, que dé soporte a la actividad de producción de la empresa local, dedicada a la elaboración de quesos.

1.4 - Objetivo específicos

Para lograr el objetivo principal del proyecto, primeramente se deberán concretar los siguientes objetivos específicos:

- Alcanzar el conocimiento general de los procesos de la empresa, principalmente aquellos relacionados con el área de producción.
- Lograr un diseño de software que dé soporte a los aspectos principales de la producción dentro de la empresa, contemplando las actividades que van desde la elaboración del producto hasta la emisión de remitos a los clientes.
- Construir e implementar el sistema en producción.

1.5 - Ubicación en el tiempo y espacio

Este proyecto se lleva a cabo a lo largo del primer semestre del año 2022, tomando inicio en el mes de febrero, con finalización estimada en el mes de julio. El proyecto se desarrolla en las ciudades de Esperanza y Santa Fe.

1.6 - Alcance del proyecto

El alcance de este proyecto abarca las etapas de *análisis global del problema, aprendizaje y familiarización con las herramientas de desarrollo, diseño, desarrollo, pruebas, despliegue de la solución y migración de datos.* Dentro de estas seis etapas, se abarcó todo el proceso necesario para lograr el despliegue de un sistema con las funcionalidades mínimas que cubra las necesidades de los interesados. A lo largo de este documento se dará a conocer más en detalle en qué consistió cada una de estas etapas.

1.7 - Alcance del producto

La solución propuesta para el problema presentado anteriormente, pretende, tal como lo indica el título del proyecto, dar soporte a la producción que se realiza en la empresa, permitiendo llevar el registro de la misma, y agilizando los procesos de carga y consulta de datos, así como también aportar a los procesos de toma de decisión presentando información de manera visual, en distintos formatos de gráficas que ayuden a la compresión del estado actual de la producción en la empresa.

Cabe aclarar que si bien lo llamamos “soporte a la producción”, este abarca desde las áreas de insumos, hasta el armado de pedidos y registro de clientes.

Los requerimientos completos del sistema requerido superan los que se pueden cubrir con el desarrollo del proyecto. Por lo que se acuerda el desarrollo de una versión inicial del sistema, con características suficientes para que sea útil.

Dentro de esta primera versión del sistema se incorporarán como funcionalidades principales, llevar un registro de la producción, de los pedidos armados para los clientes, del stock de productos, y la generación de un remito que se requiere para que el transportista pueda declarar la carga que transporta. En el capítulo 10 de este informe, se presenta información detallada acerca de las funcionalidades del sistema.

1.8 - Estructura del informe

El presente informe se organiza en diez capítulos y dos anexos. En un principio se presenta la problemática actual sobre la cual lleva a cabo el proyecto, luego de que manera se desarrolla la solución, finalizando por las conclusiones y perspectivas futuras del proyecto.

- Capítulo 1 - Introducción
- Capítulo 2 - Contexto
- Capítulo 3 - Gestión del proyecto
- Capítulo 4 - Análisis del proyecto
- Capítulo 5 - Diseño de la solución
- Capítulo 6 - Desarrollo del sistema
- Capítulo 7 - Despliegue
- Capítulo 8 - Funcionamiento del sistema

- Capítulo 9 - Conclusiones
- Capítulo 10 - Perspectivas futuras
- Bibliografía y referencias
- Anexo A - Cronograma
- Anexo B - Historias de usuario

2 - Contexto

2.1 - La Empresa

"La Chacra S.A" es la empresa a partir de la cual surge el proyecto descrito a lo largo de este informe. Se encuentra ubicada en la localidad de Colonia Cavour, provincia de Santa Fe, y se dedica a la elaboración de productos lácteos, principalmente quesos, además de contar también con producción kosher.

A continuación, se resume en cuatro pasos el proceso de producción de quesos que la empresa realiza, y en el próximo apartado se cuenta como el sistema interviene en cada uno de ellos.

1. Inicio del proceso de producción: se comienza vertiendo la leche en una tina donde posteriormente se realizan distintos procesos y agregados de ingredientes según el queso que se vaya a elaborar.
2. Pesaje y embalaje de las hormas producidas: una vez que las hormas de queso estén elaboradas y con el tiempo adecuado de estacionamiento, pasan por un proceso de pesaje y embalaje. Cada una, ya dentro de una bolsa, pasa por una balanza para ser pesada y luego se empaquetan en cajas. Para cada una de estas cajas, se imprime una etiqueta con un código de barras que contiene información del lote de producción de las hormas que contiene la caja, y del peso total del contenido de la caja.
3. Armado de expediciones: las expediciones representan el armado de un determinado pedido para un cliente. Cuando llega un pedido, se arma el mismo con distintos lotes de producción, según el tipo de producto, o la cantidad que quede disponible de ese lote.
4. Generar remito como comprobante para el cliente: luego de que un pedido se haya armado para el cliente, se genera un remito que se entrega al cliente, o al transportista, como comprobante.

2.2 - El sistema

Dentro de la empresa, el rol que desempeña el sistema desarrollado a lo largo de este proyecto, es dar soporte a los procesos de producción de quesos.

Resumiendo en algunas palabras el contenido que se detalla en las secciones correspondientes, el alcance que tiene el sistema dentro de la empresa corresponde al proceso completo de elaboración de los quesos, desde que se vierte la leche en las tinas hasta que se entrega el producto al cliente.

A lo largo de este proceso hay cuatro etapas principales donde se debe interactuar con el sistema:

1. Inicio del proceso de producción: en este punto del proceso, se genera el lote de producción en el sistema y se cargan los datos iniciales que se tienen del mismo, como ser la cantidad de litros usados, la tina en la que se realiza el proceso, la fecha, entre otros.
2. Pesaje y embalaje de las hormas producidas: el segundo punto en el que

interviene el sistema a lo largo de este proceso, se carga el peso obtenido por cada uno de estos lotes, y mediante este dato se calcula un rendimiento asociado a cada lote de producción.

3. Armado de expediciones: durante el armado de los pedidos para los clientes, se cargan en el sistema las distintas expediciones que se van generando. Las expediciones son cargadas mediante la lectura de los códigos de barra presentes en las cajas de los productos. También existe la posibilidad de que los usuarios generen expediciones en forma manual.
4. Generar remito como comprobante para el cliente: a partir de todas las expediciones generadas para el cliente, se genera un remito en el sistema con un costo estimativo según los precios cargados en el sistema.

3 - Gestión del proyecto

3.1 - Metodología

Al momento de la elección de un enfoque a utilizar para llevar a cabo el proyecto, nos basamos en la incertidumbre de las necesidades formuladas por los interesados en el sistema. A partir de esto, nos inclinamos por utilizar una metodología de desarrollo de software que nos permita la capacidad de poder trabajar en forma reactiva ante los cambios de requerimientos expresados por los stakeholders.

Más específicamente, utilizamos una metodología ad hoc a lo largo del proyecto, la cual se basó en prácticas propuestas por dos metodologías existentes, que son Kanban y Extreme Programming (de ahora en más, XP). Para ello, adoptamos sobre todo las prácticas de programación de XP, como ser el enfoque de los test unitarios y la refactorización de código, entre otros, y basamos en Kanban las prácticas de gestión en lo que respecta al avance del proyecto.

A continuación, se presentan las adaptaciones realizadas a la metodología XP que consideramos más importantes:

- **Comunicación con el cliente**: nos encontramos con una situación en la que los interesados no pueden formar parte del equipo de desarrollo, tal como dicta la metodología XP, por lo que para mantener una comunicación activa con los interesados y un buen flujo de retroalimentación respecto a los desarrollos, al final de cada etapa del proyecto se realizaron reuniones con los mismos con el fin de validar los avances y los requerimientos del sistema. Cabe destacar que si bien estas reuniones ocurrían en plazos de un mes, siempre fue posible mantener una comunicación constante con los interesados vía WhatsApp en las ocasiones que fuese necesario.
- **Programación en parejas**: esta práctica no nos resultaba factible por tres motivos:
    - El equipo de trabajo lo conformabamos tan sólo dos personas,
    - Los horarios en los que cada integrante podía aportar al desarrollo del sistema eran excluyentes,
    - En último lugar por la distancia geográfica dada por el hecho de residir en distintas ciudades.

En lugar de eliminar por completo esta práctica, nos decidimos por un flujo de trabajo que permitió retener ciertas de las ventajas que la misma presenta. Se trabajó realizando revisiones de código, las mismas ocurrían mediante la sincronización del sistema de versionado que utilizamos (Git), o al momento de aprobar los pull request, generados al momento de terminar de trabajar en una funcionalidad.

3.2 - Planificación y estimación de esfuerzo

Durante la planificación del proyecto se definen las distintas etapas que conforman el proyecto, además de definir el alcance que tuvo el mismo. Las etapas definidas durante la planificación del proyecto, se encuentran detalladas en el inciso 3.3.

Luego de la definición de las etapas del proyecto, se procede a estimar el esfuerzo que conlleva cada una de las mismas. Para esta estimación, se hace hincapié principalmente en el desarrollo, ya que se presume que es la etapa del proyecto que mayor tiempo demanda.

Se define entonces una estimación de 320 horas de trabajo para la etapa de desarrollo. Para realizar esta estimación, se toma como base las historias de usuario generadas en la primera etapa de análisis del problema, a partir de ahí, se buscan las similitudes entre las funcionalidades requeridas y funcionalidades similares implementadas por el equipo de trabajo, en proyectos similares en cuanto a tecnologías y arquitectura.

La estimación inicial se realiza de este modo y no basada en *story points* como propone la metodología XP, dado que la planificación se lleva a cabo antes de realizar un análisis más exhaustivo de las historias. Sin embargo, posteriormente se utilizan los puntos de historia para la planificación de las iteraciones. Tomamos la duración de un punto de historia como equivalente a un día ideal de trabajo, que en nuestro caso equivale a cuatro horas.

Las 320 horas de trabajo definidas para el desarrollo se distribuyen en el cronograma a lo largo de dos meses, tomando como referencia, semanas de 40 horas de trabajo en total. Dentro de estos dos meses, el trabajo se divide en tres etapas: *zero feature release, primera iteración, segunda iteración,* que son explicadas a detalle en el capítulo 6.

Luego de definir la etapa de desarrollo, agregamos al cronograma la etapa de despliegue, a la cual asignamos la duración normal de una iteración.

La planificación inicial finaliza con la confección de este informe, a la cual se le asignó inicialmente un tiempo de 4 semanas.

3.3 - Etapas del proyecto

1. **Análisis global del problema y relevamiento de requerimientos**: como primera etapa del proyecto, se analiza el problema desde la perspectiva de los interesados y se procede con un primer relevamiento de requerimientos. Esta etapa se da a conocer más a detalle en el siguiente capítulo.
2. **Aprendizaje y familiarización con las herramientas**: esta etapa la tomamos a partir la idea de una iteración cero, presente en el ciclo de vida de la metodología ágil *Feature-Driven Development* (FDD), y tiene como objetivo el aprendizaje y la familiarización, por parte del equipo de trabajo, con las herramientas utilizadas en el proyecto.
3. **Diseño de la solución**: en tercera instancia, se realiza un diseño de la solución, el cual consiste en definir un diagrama de clases de dominio inicial, para guiar a la solución del problema, así como también en definir la arquitectura del sistema.

4. **Desarrollo de la solución**: en base a las tres primeras etapas, en esta etapa se lleva a cabo el desarrollo de la solución, siguiendo los lineamientos ya definidos por el diseño, y las *task cards* provenientes de las historias de usuario generadas a partir de la primera etapa del proyecto.
5. **Pruebas**: para llevar a cabo las pruebas del sistema, se realizan tests unitarios y de integración, los cuales se corren de manera automática. Además, utilizamos un *API client* para probar el correcto funcionamiento de los servicios REST de la aplicación durante el desarrollo.
6. **Despliegue de la solución y migración de datos**: para el despliegue de la solución se procede a configurar el equipo de la empresa que oficia de servidor, donde posteriormente se instaló una instancia de Docker, donde se despliega la infraestructura previamente definida. Posteriormente se procede con la migración de datos mediante la ejecución de un script, con los datos previamente trabajados y listos para cargar.
7. **Redacción del informe final**: se lleva a cabo la redacción del presente informe.

4 - Análisis del sistema

En éste capítulo, se describen los métodos utilizados para la captura de requerimientos del sistema, seguido de las *historias de usuario* definidas para el mismo.

4.1 - Método de captura de requerimientos

Para la captura de requerimientos del sistema, nos basamos en el concepto de *historias de usuario* presente en la metodología XP, dado que es un método sencillo de aplicar gracias a su lenguaje coloquial.

El proceso de captura de requerimientos es de naturaleza iterativa, donde en primer lugar se llevan a cabo reuniones con los interesados, seguido de la elaboración de historias a partir de lo definido en las reuniones. Posteriormente se realizan *spikes*[1] en formato de interfaz de usuario, los cuales se validan por los interesados, para los requerimientos que no hayan quedado claramente definidos al momento de la elaboración de las historias de usuario.

4.2 - Historias de usuario

A continuación, se presentan los requerimientos definidos para el sistema, los cuales no se mantienen estables a lo largo del proyecto.

En principio, se presentaron los siguientes requerimientos:

1. Cargar producción
2. Cargar tipos de productos
3. Consultar lotes producidos
4. Consultar trazabilidad de lote
5. Cargar insumos
6. Cargar recetas de queso
7. Ver litros de leche usados para producción
8. Cargar expediciones de producto
9. Gestión de clientes
10. Cargar precios
11. Emitir Remito
12. Cargar devolución de producto
13. Visualizar ventas
14. Gestión de permisos de usuario

Luego de la primera iteración, se realiza una reunión con los interesados del proyecto, donde se presentan y validan los avances en el desarrollo del proyecto. En este encuentro, los interesados manifiestan inquietud por funcionalidades adicionales a las previamente

[1] *Spikes*: son prototipos que ayudan a comprender la funcionalidad que se quiere implementar.

definidas, cambiando incluso la prioridad de las mismas, y la definición de lo que sería la versión mínima aceptable del sistema. Los requerimientos que se agregan en este punto son los siguientes:

15. Stock de embalaje
16. Ver rendimiento
17. Ver stock de productos
18. Cargar expediciones con lector de código de barras

Dentro del alcance de este proyecto, no se desarrollan las historias 5, 6 y 12, ya que luego de negociar con los interesados el alcance de la versión mínima del sistema, se definen como requerimientos de baja prioridad respecto al resto de funcionalidades.

En el Anexo B se presentan más en detalle las historias correspondidas dentro del alcance de este proyecto.

5 - Diseño de la solución

En éste capítulo se presenta el diseño del sistema desarrollado a lo largo del proyecto, dando a conocer primeramente la arquitectura del mismo, luego el dominio del problema, y finalmente las tecnologías que se utilizaron para llevarlo a cabo.

5.1 - Arquitectura de la solución

Para definir la arquitectura del sistema, nos basamos principalmente en los requerimientos de los interesados, además de tener en cuenta los equipos que ellos tienen para trabajar, como así también las prestaciones de los mismos.

Dentro de este proceso de toma de decisión, tomamos en consideración el hecho de que el sistema debería ser accedido desde diferentes equipos, por lo que un sistema tipo cliente-servidor sería una muy buena opción. Consideramos también que, por las limitaciones de hardware de algunos de los equipos, una aplicación que se pueda ejecutar en el navegador sería apropiada. Finalmente, descartamos la posibilidad de que la aplicación funcione en la nube, ya que, en primer lugar, nuestro sistema dependería de la conexión a internet de la empresa, la cual es inestable, y en segundo lugar, pero más importante, los interesados quieren que los datos sean almacenados dentro de la empresa.

Luego de tomar en consideración estos puntos, llegamos a la siguiente arquitectura para la solución, que se basa en un sistema distribuido tipo cliente-servidor, y se encuentra desplegada en un servidor dentro de la empresa. El sistema se ejecuta sobre contenedores Docker, lo que minimiza el cambio de ambiente de ejecución para la aplicación y por tanto, facilita el despliegue del sistema. En la siguiente figura se muestra la arquitectura del despliegue del sistema, donde la aplicación y la base de datos funcionan en contenedores separados.

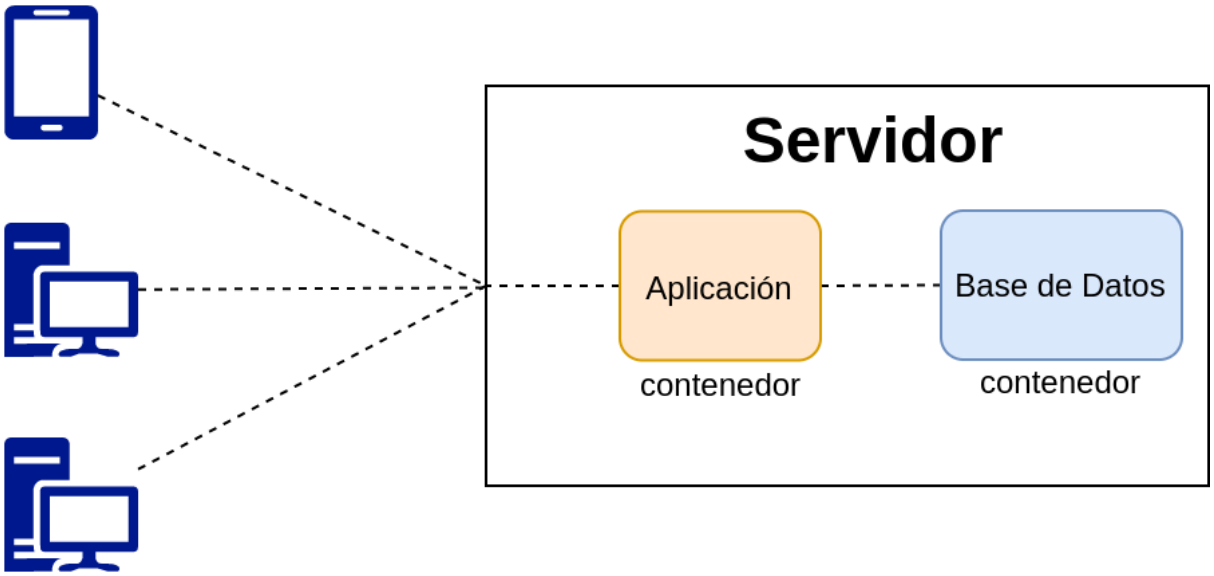


Diagrama de arquitectura a nivel instalación

Dentro de la arquitectura distribuida tipo cliente-servidor, el servidor web expone sus servicios a partir de una API REST, los cuales son consumidos mediante una aplicación cliente que se ejecuta en el navegador web del usuario utilizando el protocolo HTTP. En los apartados 5.1.1 y 5.1.2 se explica más a detalle la arquitectura interna del sistema, tomando como referencia el siguiente diagrama:

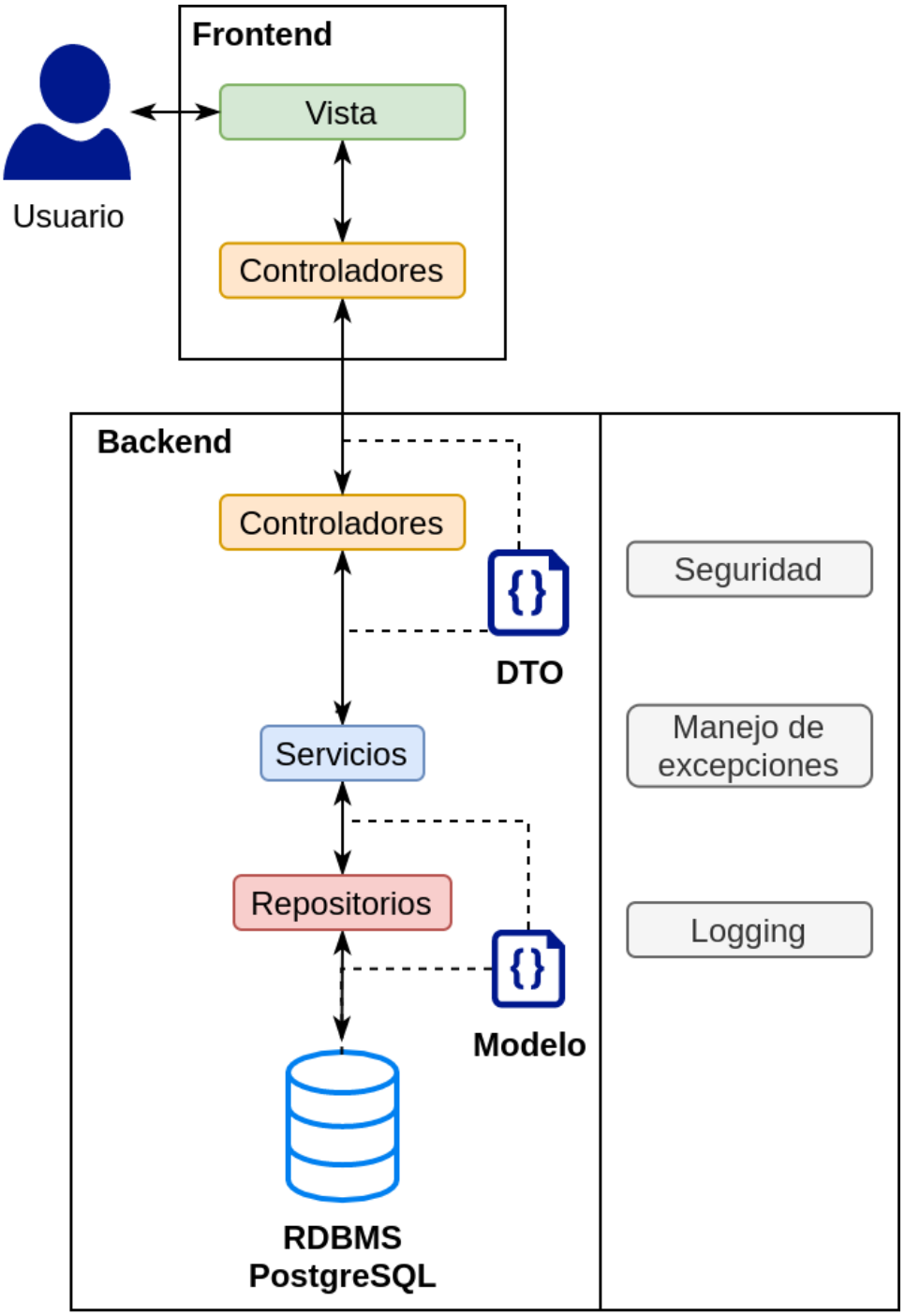


Diagrama de la arquitectura del sistema

5.1.1 - Backend

Internamente, el *backend* está compuesto por las siguientes capas:

- **Capa de Controladores:** esta capa está compuesta por los puntos de acceso a la aplicación. Aquí se procesan las solicitudes de los usuarios, se invoca a los servicios correspondientes, y se retorna una respuesta al cliente mediante *Objetos de transferencia de datos (DTOs)*.
- **Capa de Servicios:** en esta capa se realiza el procesamiento que se requiera, es donde se ejecuta la lógica de negocios. Esta capa a su vez se comunica con el repositorio, o capa de acceso a datos, para interactuar con la persistencia de los datos.
- **Repositorio o capa de Acceso a Datos:** capa de interacción con la base de datos. Aquí se define como se leen, crean, editan y eliminan los datos almacenados en la base de datos.

Los datos dentro del backend se representan principalmente por dos tipos de clases:

- **Modelo:** compuesto por las clases que representan el dominio y se corresponden con las tablas en la base de datos.
- **Objetos de Transferencia de Datos (DTOs):** son objetos utilizados para la comunicación con los clientes. Son versiones simplificadas de las entidades del modelo.

Transversalmente a los ítems anteriores, se ejecutan los siguientes módulos:

- **Módulo de Seguridad:** es el encargado de asegurar los recursos del sistema, de validar las credenciales de los usuarios y de otorgar y renovar tokens de acceso. Se utiliza en el sistema para restringir el acceso de los diferentes usuarios. Se explica de mejor manera su implementación en la sección 5.4.
- **Módulo de Manejo de Excepciones:** encargado de procesar los errores que surjan durante la ejecución, devolviendo los mensajes correspondientes al cliente.
- **Módulo de Logging:** es el encargado de registrar los eventos ocurridos en el backend.

5.1.2 - Frontend

En el *frontend,* seguimos un patrón de diseño de aplicación de una única página (SPA por sus siglas en inglés). Este patrón, es altamente utilizado en la actualidad debido a que resulta en un diseño que responde de manera rápida a la interacción del usuario, ya que no requiere de recargar la página durante el uso de la aplicación, resultando en una experiencia de usuario similar al de una aplicación nativa. Ejemplos de aplicaciones que utilizan este patrón son: Gmail, Google Docs, GitHub, Twitter y Facebook.

En la arquitectura del sistema, el *frontend* está simplemente dividido en dos componentes:

- **Vistas:** definen cómo se organiza la información, es el punto de interacción con el usuario.
- **Controladores:** encargados de la comunicación con el *backend*. Realizan el procesamiento de la información ingresada por el usuario a *DTO* para enviarla al *backend*, y a su vez convierten la información recibida del servidor para ser mostrada en las vistas.

5.2 - Diagramas de dominio de la solución

A continuación, se muestra en las siguientes figuras las clases que fueron definidas para modelar el dominio del problema, como así también el diagrama de tablas de la base de datos relacional en la que se persisten los datos del sistema.

5.2.1 - Diagrama de clases

El modelo de dominio del sistema sufrió cambios a lo largo del transcurso del proyecto, impulsados, principalmente, por el cambio de requerimientos descripto en la sección 4.2. En las figuras se muestra el diagrama de clases de dominio inicial que se generó en el análisis, seguido del diagrama de clases dominio del sistema que se llevó a producción.

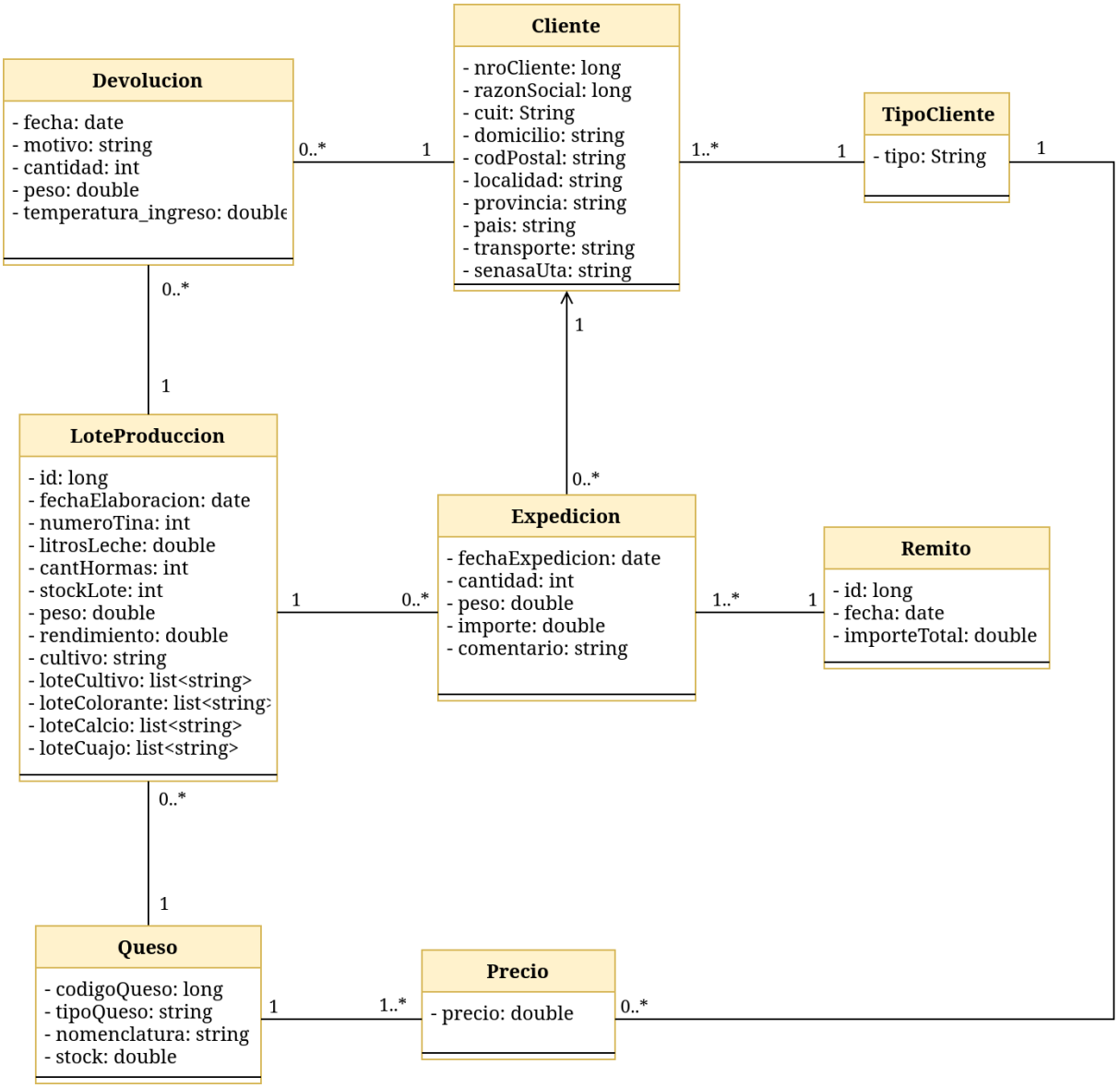


Diagrama de clases inicial

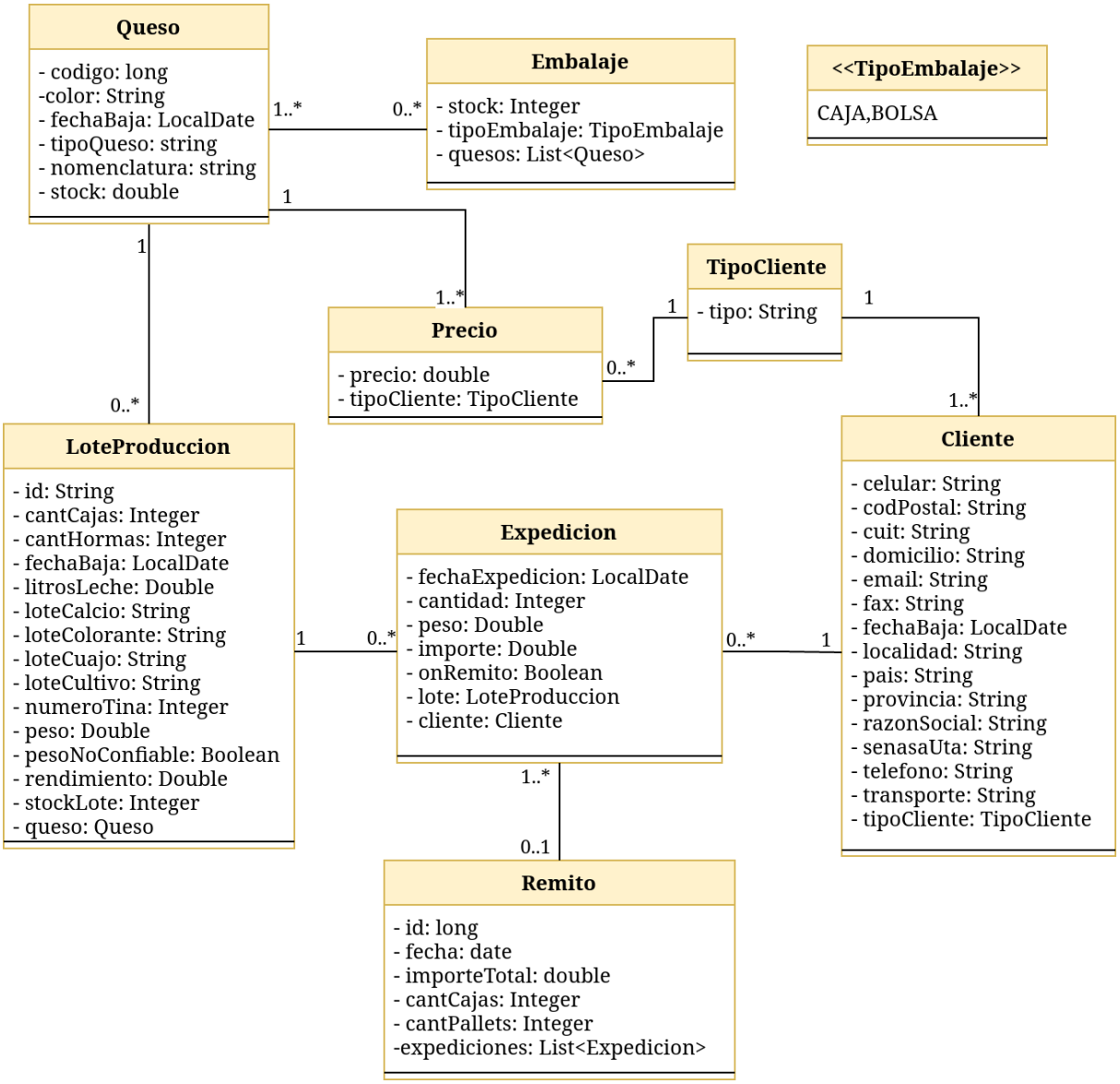


Diagrama de clases final

5.2.2 - Diagrama de tablas

En la siguiente figura, se presenta el diagrama de tablas que actualmente se encuentran presentes en la base de datos. Adicionalmente a las tablas correspondientes al modelo del dominio del sistema, se almacenan los usuarios del mismo, en conjunto con sus permisos.

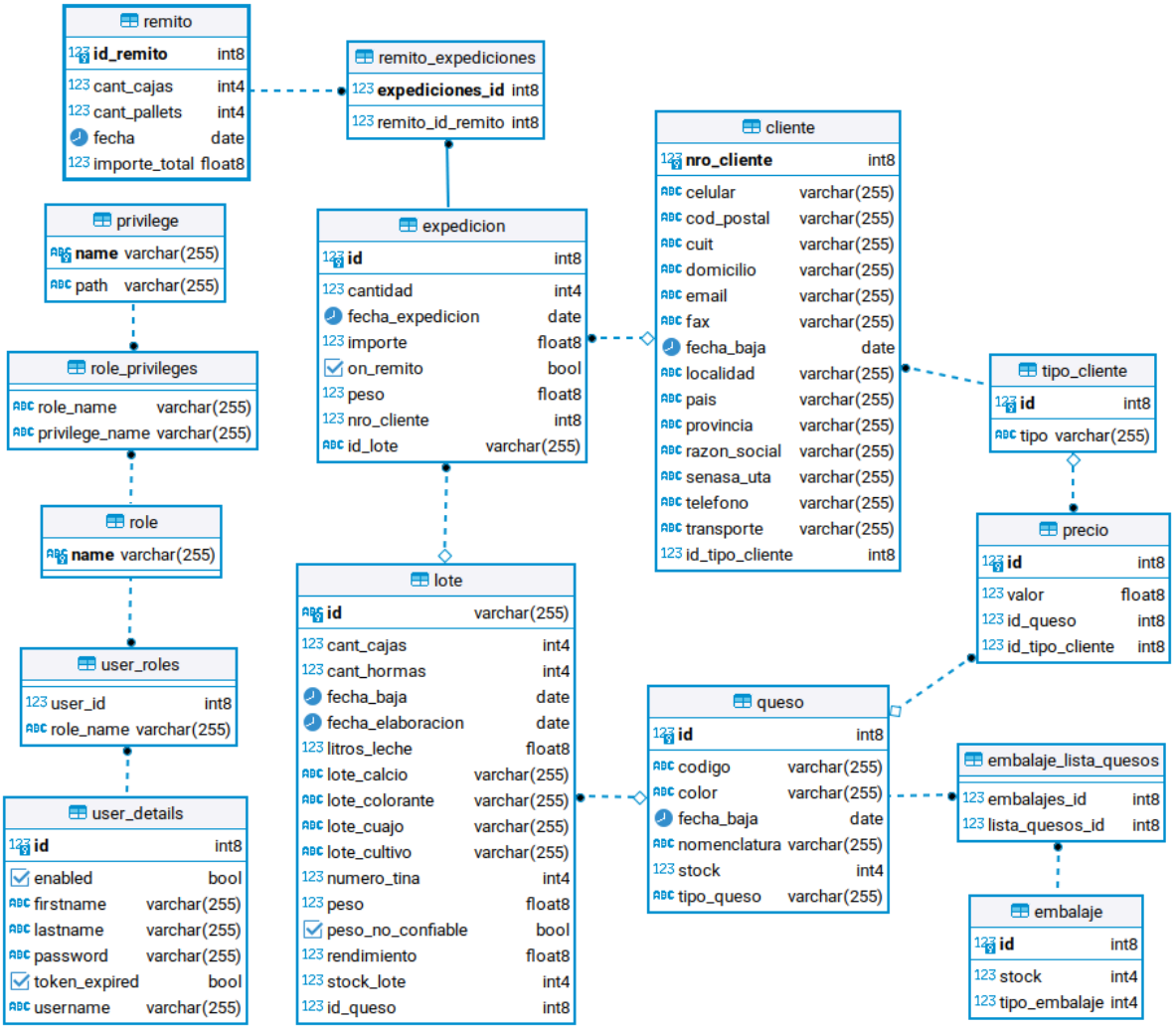


Diagrama de tablas del sistema

5.3 - Tecnologías utilizadas y entorno de desarrollo

A continuación se dan a conocer las tecnologías utilizadas para el desarrollo del proyecto, detallando las mismas a lo largo de cuatro secciones. En primer lugar se presentan las tecnologías utilizadas para el desarrollo del backend del sistema, seguido de las utilizadas para el frontend, luego las que se utilizaron para el testing del sistema, finalizando con las herramientas de apoyo al desarrollo.

5.3.1 - Tecnologías de backend

- **Java 11:** Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y multiplataforma. La elección se debió a ser el lenguaje de programación con el que el equipo contaba con mayor experiencia.
- **Maven:** Es una herramienta de software para la gestión de dependencias y construcción de proyectos.
- **PostgreSQL:** Es un sistema de gestión de bases de datos relacional orientado a objetos, seleccionado principalmente por su eficiencia, por ser un proyecto *open source* y por contar con una gran comunidad detrás.
- **Hibernate:** Es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación.
- **Spring Framework:** Es una solución liviana con potencial para la construcción de aplicaciones para empresas. Spring es modular, lo que le permite usar solo las partes del framework que necesita, sin tener que cargar con el resto. En particular, las principales características que utilizamos en este proyecto fueron:
    - **Spring Boot versión 2.5.6:** Es un proyecto que entrega interfaces ya implementadas en base a las que se consideran que son las mejores prácticas dentro de la industria.
    - **Spring Data:** Ofrece una abstracción de la capa de acceso a la persistencia. Al utilizarse con Spring Boot, las interfaces de configuración en este proyecto se implementan con Hibernate como mapeador objeto-relacional.
    - **Spring Security:** Se enfoca en proveer los mecanismos necesarios para la autenticación y autorización de una aplicación Java.
- **JWT:** Es un estándar abierto basado en JSON para la creación de tokens de acceso que permiten la propagación de identidades.
- **Docker:** Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

5.3.2 - Tecnologías de frontend

- **Javascript:** Es un lenguaje de programación que es una de las tecnologías principales de la Web, en conjunto con HTML y CSS.
- **React:** Es una librería de *frontend* en Javascript que permite la construcción de interfaces de usuario.
- **Axios:** Es una librería de Javascript que proporciona un cliente HTTP, que se utilizó para la comunicación con el *backend* del sistema.
- **Material UI:** Es una librería de componentes React que ofrece un conjunto de

componentes UI, estilizados bajo los estándares de *Material Design*[2]. Se utilizó para acelerar el desarrollo en cuestiones de diseño de interfaces de usuario.

5.3.3 - Tecnologías de testing

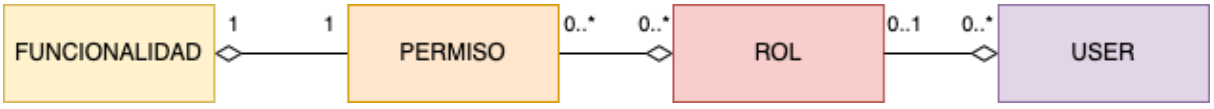
- **JUnit:** Es una de las herramientas más populares para realizar *testing unitario* en Java. Consiste en un conjunto de librerías que posibilitan verificar el comportamiento y salidas de un componente específico.
- **Insomnia / Postman:** Son herramientas que permiten realizar testing de las APIs mediante el envío de diferentes peticiones HTTP. Estas herramientas nos permitieron hacer pruebas de integración del sistema a medida que íbamos desarrollando el mismo.

5.3.4 - Tecnologías de apoyo al desarrollo

- **IntelliJ IDEA:** IDE utilizado para el desarrollo del backend.
- **Visual Studio Code:** Es un editor de código fuente multiplataforma creado por Microsoft. Se utilizó principalmente para el desarrollo del frontend del sistema.
- **GitHub / Git:** GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.
- **Trello:** Es un software de administración de proyectos, con interfaz web y clientes para iOS y Android.
- **DBeaver:** Es una herramienta multiplataforma de acceso universal a bases de datos. Se utilizó en desarrollo y producción para correr distintos *scripts* en la base de datos, así como también para la migración de datos.

5.4 - Seguridad

El sistema implementa un esquema de seguridad en donde las distintas funcionalidades están asociadas a un permiso único, tal que para un usuario poder usar una funcionalidad, debe tener dicho permiso. Los permisos a su vez, son agrupados en roles, los cuales son asignados a los usuarios, y es la manera en la que los usuarios obtienen sus permisos. Los usuarios pueden estar asociados a un único rol.



Cada funcionalidad está identificada por un *URI path*[3] que se corresponde con una interfaz de usuario del lado del frontend. Todas las acciones posibles dentro de la página están marcadas por el URI path que identifica la funcionalidad, por lo que cuando se hace la solicitud al backend se conoce que tipo de funcionalidad está utilizando el usuario. A partir

[2] Material Design es un sistema desarrollado por Google y consiste en una serie de pautas, componentes y herramientas que respaldan las mejores prácticas de diseño de interfaz de usuario.

[3] Ruta Identificador de recursos uniforme o URI path nos permite identificar los diferentes recursos que nuestro sistema ofrece mediante el navegador. Ejemplo: “/api/v1/clientes”

de eso, y conociendo los permisos del usuario, se permite o deniega la petición.

Por ejemplo, tenemos un permiso llamado CARGAR_LOTES, el cuál relacionamos al URI path "/cargar/lotes", de esta manera el usuario que tenga asignado este permiso puede visualizar y acceder a la página para cargar lotes, usando la ruta determinada por el permiso, y luego sería capaz de cargar lotes al sistema.

Tanto la asignación de permisos a roles como la asignación de roles a usuarios son procesos dinámicos. Todas las relaciones son fáciles de modificar y se encuentran definidas a nivel de base de datos.

Para el desarrollo del proyecto, todavía no se contempla ofrecer funcionalidades que permitan definir roles y permisos a usuarios mediante la aplicación. Los roles a asignar son definidos por los interesados. De nuestro lado, modelamos las asignaciones correspondientes mediante un script en la base de datos.

Dentro de este script definimos contraseñas a las cuales les aplicamos un algoritmo de cifrado[4]. Hacemos esto con el objetivo de que las contraseñas guardadas en la base de datos no se persistan en formato plano.

En el ambiente productivo para determinar si las credenciales de un usuario son válidas, se busca la información del usuario en la base de datos, luego se encripta la clave ingresada por el usuario y se compara con el valor obtenido de la base de datos, si las contraseñas coinciden entonces damos como auténtico el inicio de sesión.

Cuando un usuario inicia sesión se genera un *token*. Este token sirve para identificar al usuario en las peticiones que se realizan sobre HTTP. En el token transmitimos la información del nombre del usuario que inicio sesión y el tiempo de expiración del mismo, que es predefinido al inicio de la app en el servidor y es el resultado de la suma de dicho valor con el valor de la hora en el que el usuario inicio sesión. Al token se le aplica otro proceso de cifrado, en el cual combinamos la información presente en el mismo y un valor **salt**[5], mediante una operación de hash[6] y el resultado es el dato enviado al frontend. El objetivo de esto es que la información presente en el token sólo pueda ser leída en el backend.

Teniendo un inicio de sesión exitoso, desde el frontend se consultan por los permisos que el usuario tiene con el objetivo de solo mostrarle las páginas donde el usuario puede accionar.

Tenemos 2 secuencias básicas respecto al uso de mecanismos de seguridad en todos los flujos de la aplicación: cuando se inicia sesión y cuando se ejecutan funcionalidades del sistema.

---

[4] es un proceso en el que se convierte el valor original de la información, conocida como texto plano, en una forma alternativa conocida como texto cifrado

[5] valor aleatorio que combinamos con la contraseña en el proceso de hashing, para la obtención de una contraseña única y encriptada.

[6] algoritmo matemático que transforma una entrada arbitraria de datos en una nueva serie de caracteres con una longitud fija. Independientemente de la longitud de los datos de entrada, el valor hash de salida tiene siempre la misma longitud.

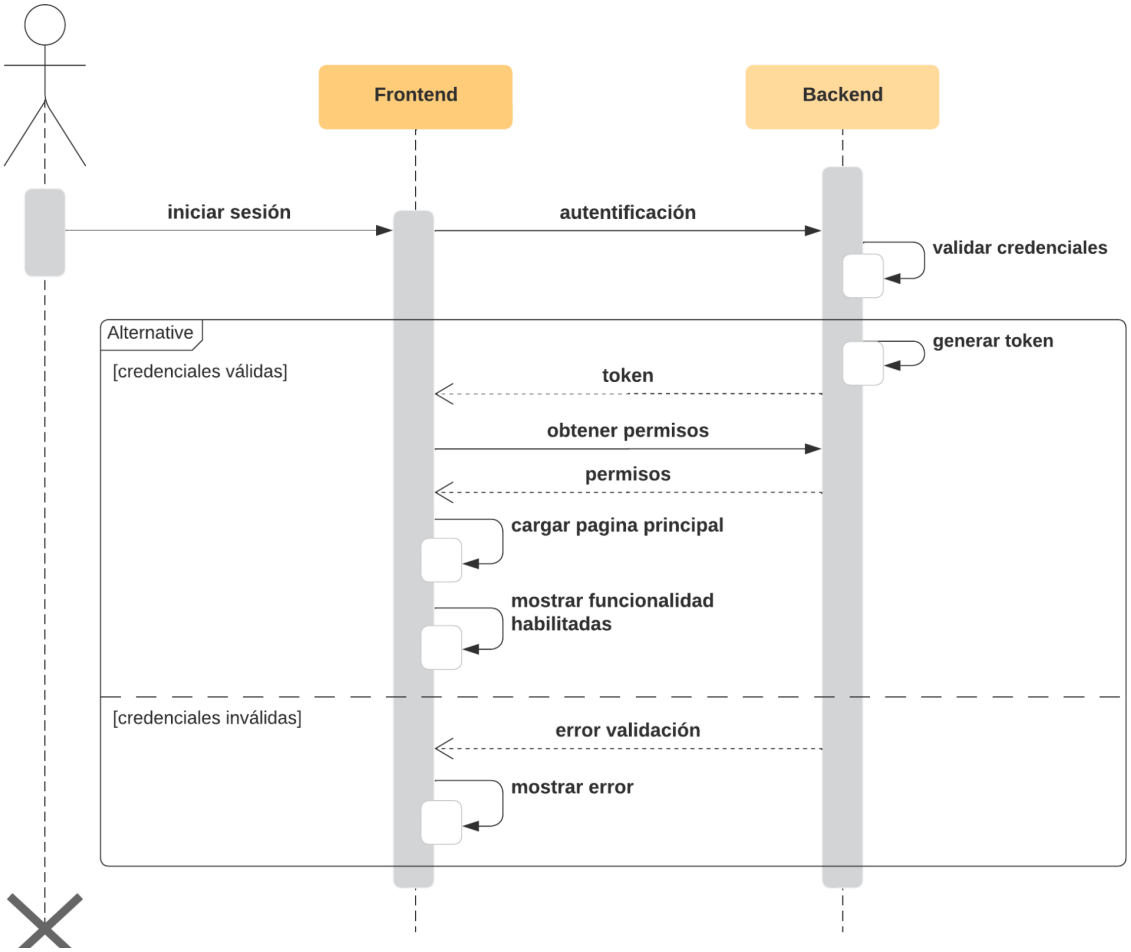


Diagrama de secuencias - Inicio sesión

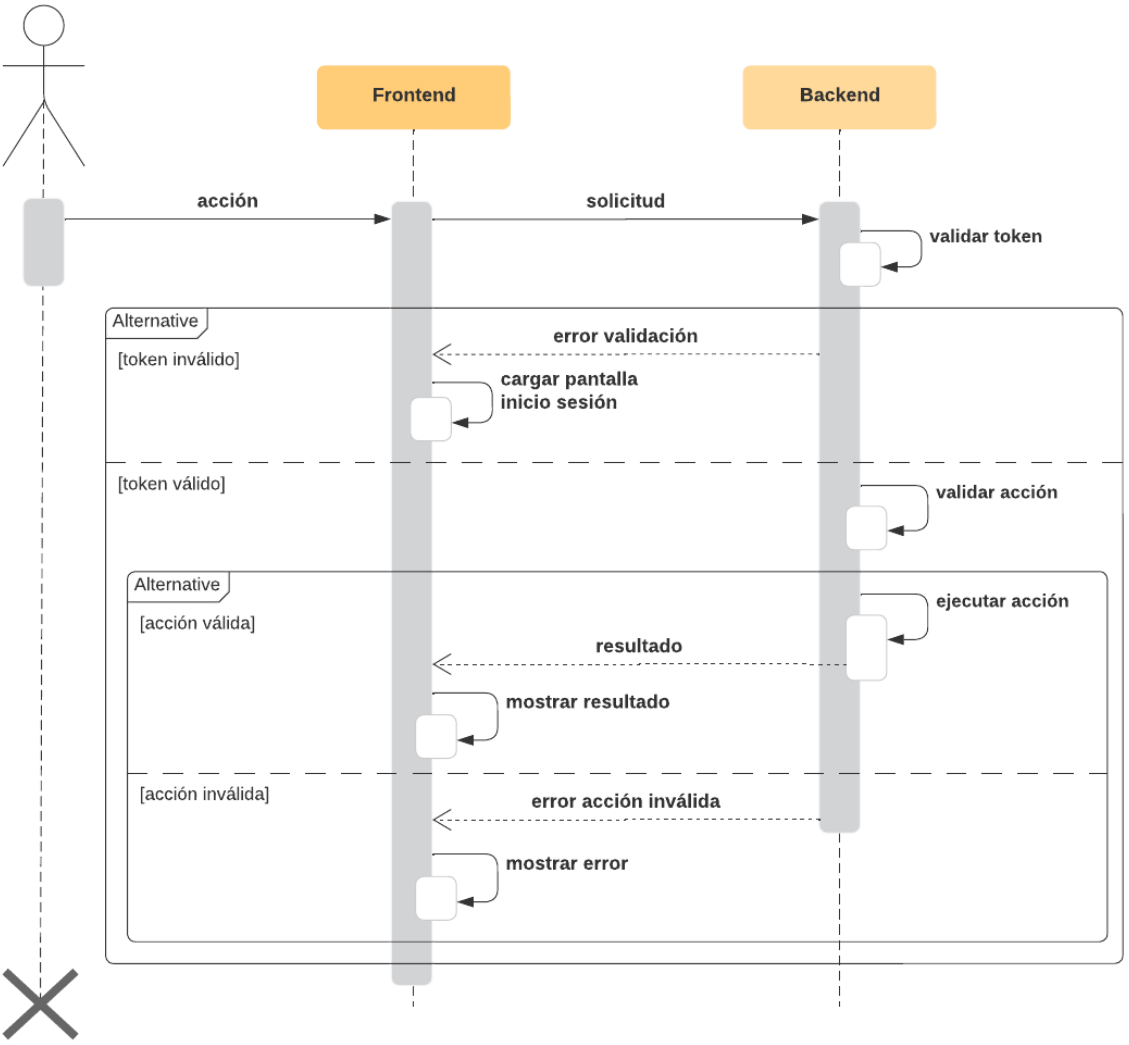


Diagrama de secuencias - acción sobre el sistema

6 - Desarrollo del sistema

La cuarta etapa de este proyecto, abarca el desarrollo del sistema. Este capítulo explica cómo fue llevado a cabo este proceso, explicando la metodología que se siguió junto con sus distintas etapas.

6.1 - Metodología de desarrollo

Como se explica en el capítulo 3, se utiliza una metodología de desarrollo iterativa, influenciada principalmente por *Kanban* y *Extreme Programming*. De este modo, las funcionalidades del sistema o historias de usuario, se encuentran divididas en unidades más pequeñas, siguiendo la metodología XP, llamadas tareas, o *task cards*. Estas tareas se organizan en un tablero kanban, de acuerdo a las historias asignadas para cada iteración, lo que nos permite tener noción del avance del proyecto.

6.2 - Iteraciones

El trabajo de desarrollo se realiza a lo largo de 4 iteraciones. Se define una duración de 3 semanas para las iteraciones, a excepción de la *Zero Feature Release*, y de la tercera iteración, que se agrega durante el transcurso del proyecto como etapa necesaria para poder realizar el despliegue del sistema.

Al inicio de cada iteración se define el alcance que se tendría, en base a las prioridades asignadas por los *stakeholders* a las historias de usuario, y en base al riesgo de desarrollo de las mismas. A su vez, al final de cada iteración se realizan reuniones de validación con los *stakeholders*, donde también se introducen nuevos requerimientos al sistema.

A continuación se detalla en qué consiste cada una de estas etapas y el alcance de las mismas.

6.2.1 - Iteración "Zero Feature Release"

Tomamos la idea de *Zero Feature Release* (ZFR) de la metodología XP. Esta etapa consiste en desarrollar el esqueleto del sistema, sin funcionalidad alguna. Esto incluye la inicialización de los proyectos para *frontend* y *backend*, el desarrollo de las clases del dominio, establecer la conexión con la base de datos y lograr la comunicación entre la aplicación cliente y la aplicación servidor.

El objetivo de esta etapa, es verificar que las diferentes partes del sistema funcionen de manera integrada desde un inicio en el proyecto, para ahorrar complicaciones de integración en etapas posteriores.

En esta etapa también se profundiza en el análisis de los requerimientos mediante la elaboración de *spikes*, que permiten un mayor entendimiento de la lógica que debería realizar el sistema por medio de la definición de una versión inicial de las interfaces de usuario final.

Debido a que la cantidad de trabajo para realizar en esta etapa es acotado, se le asigna solamente una semana de duración, dado que si se le asigna la misma duración que las otras iteraciones, el resto del tiempo en la etapa sería ocioso.

6.2.2 - Primera Iteración

Dentro de la primera iteración del desarrollo, se decide trabajar en las siguientes 5 historias:

- 1 - Cargar Producción
- 2 - Cargar Tipos de Productos
- 8 - Cargar Expediciones de Productos
- 9 - Gestión de Clientes
- 10 - Cargar Precios

La decisión de trabajar específicamente con estas historias, se debe principalmente a que se busca llegar a una versión mínima del sistema en dos iteraciones, por lo que se eligen las historias según la dependencia que existe entre las mismas a la hora de operar el sistema.

Luego de esta primera iteración, se lleva a cabo una reunión con los *stakeholders*, donde se presentan los avances del proyecto y se validan los mismos, tomando nota de los pequeños cambios que se deban realizar. También, como se menciona en el capítulo 4, los interesados manifiestan en esta reunión nuevas necesidades sobre el sistema, cambiando también la prioridad de las mismas.

Conforme al avance de esta primera iteración, notamos como la velocidad de desarrollo crece hacia el final de la etapa. Ésto debido principalmente a, no solo la reutilización de código dado que varias interfaces de usuario y servicios del lado del backend compartían funcionalidades y diseños, sino que también por el acostumbrarse a la estructura del proyecto.

6.2.3 - Segunda Iteración

A partir del incremento de velocidad de desarrollo que se tiene hacia finales de la primera iteración, en esta segunda se logra desarrollar una mayor cantidad de funcionalidades. En principio, el plan para la segunda iteración consiste en realizar los cambios sugeridos en la reunión, ya que los mismos no representaban una gran carga de trabajo, además de desarrollar las siguientes historias:

- 3 - Consultar Lotes Producidos
- 11 - Emitir un Remito
- 14 - Permisos de Usuarios
- 17 - Ver Stock de Productos
- 18 - Cargar Expediciones Mediante Código de Barras

Estas historias suman una carga de trabajo similar a la de la primera iteración, basándonos en los puntos de historia asignados. La selección de las mismas se basa en ponderar: llegar a una versión mínima luego de esta iteración, y las nuevas inquietudes de los interesados; con estas 5 historias planificadas, se llegaría a una versión mínima del sistema, manteniendo aún las principales inquietudes de los *stakeholders*.

Como se menciona anteriormente, en esta etapa se logra desarrollar una mayor cantidad de funcionalidades a la esperada, por lo que el alcance final de esta etapa también incluye las siguientes historias:

- 4 - Consultar Trazabilidad de un Lote
- 7 - Visualización de litros usados para elaboración
- 13 - Visualizar Ventas
- 15 - Stock de Embalaje

Finalizada esta etapa, se realiza la reunión planeada al final de cada iteración, para presentar los avances del proyecto y tener una retroalimentación de los interesados. En esta reunión, como en la de la iteración anterior, se plantean cambios mínimos al sistema, así como también nuevos requerimientos, que no fueron incluidos en el presente informe dado que exceden el alcance.

En vista de seguir con la planificación del proyecto, se negocia en esta misma reunión, cuales son los cambios mínimos al sistema para llevarlo a producción, y que además, en el tiempo de un mes se realice el despliegue del sistema.

6.2.4 - Tercera Iteración

Luego de haber definido que se requieren cambios en el sistema para llevarlo a producción, se planifica una tercera iteración de corta duración, una semana, ya que la cantidad de trabajo es mínima y se quiere avanzar con el despliegue del sistema.

Dentro de los cambios mínimos que se realizan en esta iteración, se encuentran:

- Agregar un apartado que permita ver remitos y anularlos
- Agregar un apartado que permita ver las expediciones
- Agregar opción de expedir un lote completo
- Cambiar el rango de las consultas de meses a semanas

En el tiempo asignado para este trabajo, se logra desarrollar estos cambios en el sistema, de modo que al finalizar esta iteración, se comienza con la etapa de despliegue.

6.3 - Control de versiones

Como se menciona anteriormente en este informe, en el apartado 5.3.4 “Tecnologías de apoyo al desarrollo”, utilizamos el sistema de control de versiones Git, alojando el código dentro de la plataforma GitHub.

Para el control de versiones, tomamos como línea base la iteración “Zero Feature Release” (ZFR), punto en el que empezamos a utilizar el flujo de control de versiones que se explica en los siguientes párrafos. A continuación, se presenta un diagrama a modo de ejemplo del flujo que se siguió, y seguidamente se explica el mismo.

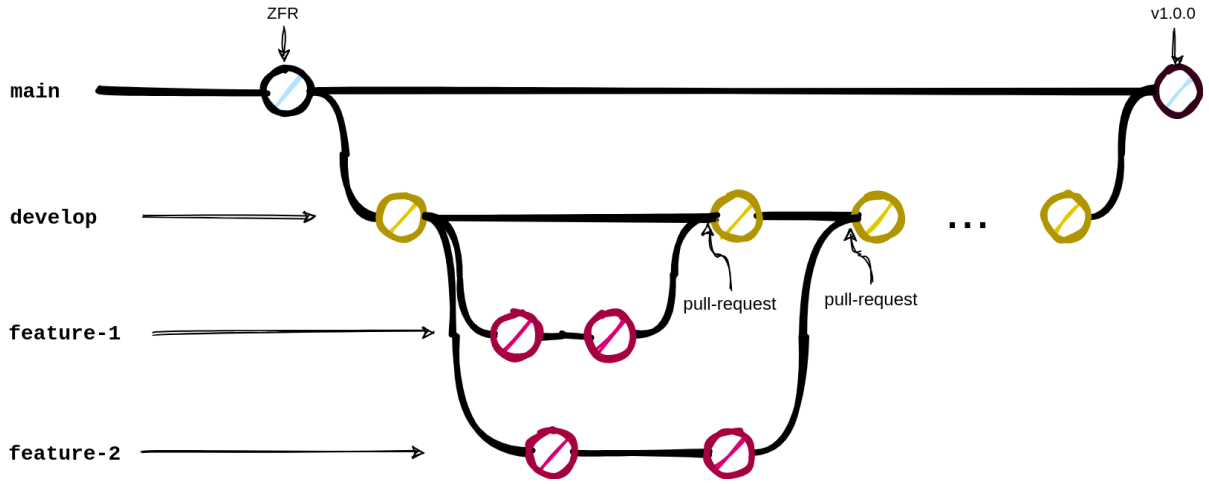


Diagrama de ejemplo de control de versiones

A partir del ZFR, se crea la rama “develop”, que se utiliza como base para el desarrollo de las distintas funcionalidades. En esta rama siempre se mantiene una versión del proyecto en un estado funcional, donde además, todas las funcionalidades pasan sus correspondientes tests. Al momento de desarrollar una nueva *feature* o funcionalidad se crea una rama aparte en la cual se desarrolla la misma, y cuando se termina el trabajo, se crea un pull request a modo de revisión de código y para la aceptación de cambios por parte de todo el equipo. En casos de realizar arreglos de bug que aparecen en el código, se sigue el mismo proceso. Al finalizar las iteraciones, la primera versión del proyecto se lleva a la rama master.

6.4 - Pruebas

La quinta etapa de este proyecto, como se encuentra definido en el capítulo 3 de este informe, no es una etapa que se lleva a cabo de manera posterior al desarrollo, si no que ambas son realizadas en paralelo.

Si bien no se lleva a la práctica el concepto de *programación dirigida por pruebas (TDD, por sus siglas en inglés)* que propone XP, se establece en el equipo de trabajo la convención de que para aprobar un *pull-request*, deben estar escritos los tests asociados a la funcionalidad implementada, de modo que siempre se mantiene un alto *nivel de cobertura* en los test del sistema.

6.4.1 - Pruebas unitarias

Para llevar a cabo las pruebas unitarias aprovechamos las facilidades que brinda la inyección de dependencias de Spring Boot, pudiendo así testear casi todas las funcionalidades del sistema, principalmente las de las capas de servicio. Más específicamente, utilizamos la inyección de dependencias, que nos permite simular el comportamiento de los componentes de los cuales las funcionalidades a testear dependen. De esta forma podemos lograr validar el comportamiento de las funcionalidades de forma muy controlada.

Para las pruebas unitarias utilizamos el framework JUnit, como mencionamos en el

apartado de tecnologías, pudiendo así correr las pruebas de manera automatizada. Además, los tests unitarios se escribieron de forma que se evalúen la mayoría de los posibles comportamientos de los componentes, evaluando así la mayor cantidad de caminos posibles en el flujo del sistema.

Las capas de controlador y repositorio no fueron testeadas de forma unitaria. Aunquea la primera capa posea funcionalidades a testear, como lo son la comunicación del exterior con la capa de servicio, las validaciones de campos que recibe como parte de las solicitudes REST, o los permisos necesarios para ejecutar las acciones, no se escribieron pruebas automatizadas por limitaciones de tiempo, teniendo en cuenta que además estas funcionalidades se prueban de manera integrada al momento de establecer comunicación entre el *frontend* y el *backend* de la aplicación.

Por otro lado, para la capa de repositorio prescindimos de los tests unitarios ya que el framework de Spring Boot nos provee interfaces con las funcionalidades básicas requeridas ya implementadas, con lo cual consideramos innecesarias las pruebas sobre esta capa.

Para ejemplificar los tests unitarios hechos vamos a usar como demostración los tests hechos sobre la función *save()* de la capa de servicio de precio.

La función es la siguiente:

```
public PrecioDTO save(PrecioDTO dto) throws PrecioAlreadyExistsException {
    validateExistencePrecio(dto.getIdQueso(), dto.getIdTipoCliente());
    var precio = precioFromDTO(dto);
    precio = repository.save(precio);
    return new PrecioDTO(precio);
}
```

Se crearon los siguientes *mocks*[7] de los servicios necesarios para ejecutar la acción de save().

[7] Definimos como *mock* a aquellas clases que nos sirven para simular el comportamiento de otras clases, lo que nos facilita el testing de los servicios que se componen de las clases a simular.

```
@MockBean
PrecioRepository repository;

@MockBean
QuesoRepository quesoRepository;

@MockBean
TipoClienteService tipoClienteService;
```

```
@Test
void Save__OK() {
    var precioDTOToSave = new PrecioDTO();
    precioDTOToSave.setValor(1000D);
    precioDTOToSave.setIdTipoCliente(1L);
    precioDTOToSave.setIdQueso(1L);

    var tipoClienteMock = new TipoCliente();
    tipoClienteMock.setTipo("tipo");
    tipoClienteMock.setId(1L);

    var quesoMock = new Queso();
    quesoMock.setId(1L);
    quesoMock.setTipoQueso("tipoQueso");
    quesoMock.setNomenclatura("nomenclatura");
    quesoMock.setCodigo("codigo");

    var precioSaved = new Precio();
    precioSaved.setId(1L);
    precioSaved.setValor(1000D);
    precioSaved.setQueso(quesoMock);
    precioSaved.setTipoCliente(tipoClienteMock);

    var expectedPrecio = new PrecioDTO();
    expectedPrecio.setId(1L);
    expectedPrecio.setValor(1000D);
    expectedPrecio.setIdTipoCliente(1L);
    expectedPrecio.setIdQueso(1L);

    when(tipoClienteService.get(1L)).thenReturn(tipoClienteMock);
    when(quesoRepository.findById(1L)).thenReturn(Optional.of(quesoMock));
    when(repository.existsByQuesoAndTipoCliente(1L, 1L)).thenReturn(false);
    when(repository.save(any(Precio.class))).thenReturn(precioSaved);

    var dtoActual = service.save(precioDTOToSave);
    assertEquals(expectedPrecio, dtoActual);
}
```

Test exitoso

```java
@Test
void Save__Tipo_Cliente_Not_Found() {
    PrecioDTO precioDTOToSave = new PrecioDTO();
    precioDTOToSave.setValor(1000D);
    precioDTOToSave.setIdTipoCliente(1L);
    precioDTOToSave.setIdQueso(1L);

    when(tipoClienteService.get(1L)).thenThrow(new TipoClienteNotFoundException());
    TipoClienteNotFoundConflictException thrown = assertThrows(
            TipoClienteNotFoundConflictException.class, () -> service.save(precioDTOToSave)
    );
    assertEquals(ErrorMessages.MSG_TIPO_CLIENTE_NOT_FOUND, thrown.getMessage());
}
```

Test de falla: el cliente no existe.

```java
@Test
void Save__Queso_Not_Found() {
    PrecioDTO precioDTOToSave = new PrecioDTO();
    precioDTOToSave.setValor(1000D);
    precioDTOToSave.setIdTipoCliente(1L);
    precioDTOToSave.setIdQueso(1L);

    TipoCliente tipoClienteMock = new TipoCliente();
    tipoClienteMock.setTipo("tipo");
    tipoClienteMock.setId(1L);

    when(tipoClienteService.get(1L)).thenReturn(tipoClienteMock);
    when(quesoRepository.findById(1L)).thenReturn(Optional.empty());
    QuesoNotFoundConflictException thrown = assertThrows(
            QuesoNotFoundConflictException.class, () -> service.save(precioDTOToSave)
    );
    assertEquals(ErrorMessages.MSG_QUESO_NOT_FOUND, thrown.getMessage());
}
```

Test de falla: el queso no existe

```java
@Test
void Save__Precio_Already_Exists() {
    PrecioDTO precioDTOToSave = new PrecioDTO();
    precioDTOToSave.setValor(1000D);
    precioDTOToSave.setIdTipoCliente(1L);
    precioDTOToSave.setIdQueso(1L);

    TipoCliente tipoClienteMock = new TipoCliente();
    tipoClienteMock.setTipo("tipo");
    tipoClienteMock.setId(1L);

    Queso quesoMock = new Queso();
    quesoMock.setId(1L);
    quesoMock.setTipoQueso("tipoQueso");
    quesoMock.setNomenclatura("nomenclatura");
    quesoMock.setCodigo("codigo");

    Precio precioSaved = new Precio();
    precioSaved.setId(1L);
    precioSaved.setValor(1000D);
    precioSaved.setQueso(quesoMock);
    precioSaved.setTipoCliente(tipoClienteMock);

    var expectedPrecio = new PrecioDTO();
    expectedPrecio.setId(1L);
    expectedPrecio.setValor(1000D);
    expectedPrecio.setIdTipoCliente(1L);
    expectedPrecio.setIdQueso(1L);

    when(tipoClienteService.get(1L)).thenReturn(tipoClienteMock);
    when(quesoRepository.findById(1L)).thenReturn(Optional.of(quesoMock));
    when(repository.existsByQuesoAndTipoCliente(1L, 1L)).thenReturn(true);
    PrecioAlreadyExistsException thrown = assertThrows(
            PrecioAlreadyExistsException.class, () -> service.save(precioDTOToSave)
    );
    assertEquals(ErrorMessages.MSG_PRECIO_ALREADY_EXISTS, thrown.getMessage());
}
```

Test de falla: el precio ya fue guardado anteriormente

6.4.2 - Pruebas de integración

Llevamos a cabo pruebas de integración de forma manual para el frontend y de forma automatizadas en código para el backend.

Frontend

Para el frontend se buscó comprobar el correcto funcionamiento de los componentes del mismo en funcionamiento con el backend, de forma que la comunicación fuera “fluida”, de modo que, la información que se pidiera y enviara mantenga un cierto estándar entre ambos.

Estas pruebas consisten en usar las distintas funcionalidades del sistema de forma que no haya comportamientos erráticos en el mismo. Definimos como comportamientos erráticos del sistema a aquellos comportamientos que no esperábamos del mismo, como pueden ser:

- Mensajes de error al utilizar las funcionalidades del sistema del modo que se diseñaron.
- Mensajes de éxito al utilizar las funcionalidades del sistema de un modo incorrecto.
- Errores en la consola del navegador o en el log de Spring Boot mientras se utiliza el sistema.
- Que al hacer clic sobre botones no suceda lo esperado, por ejemplo, al querer guardar/actualizar/eliminar una entidad parezca no suceder nada el sistema.

Backend

En cambio para el backend, incluimos pruebas automatizadas que buscan verificar el correcto funcionamiento de todos los componentes del backend, desde la capa más baja hasta la capa más alta, todo en conjunto.

Se hicieron pruebas para todos los *flujos* del sistema. Definimos como *flujo* a la secuencia del sistema que comienza con una petición REST al backend, pasando por los componentes del controlador, capa de servicio, capa de repositorio y finalmente por la base de datos. En estas pruebas la base de datos fue simulada usando una base de datos en caliente, a la cual se le cargan los datos cada vez que se corren las pruebas.

De forma general, para probar un flujo del sistema validamos los siguientes comportamientos:

1. Que la petición sea exitosa.
2. Que la petición falle por no tener permisos.
3. Que la petición falle por no cumplir las validaciones sobre los campos enviados.
4. Que la petición falle por errores en reglas de negocio.

Para ejemplificar estas validaciones vamos a usar el flujo del *guardado de precios*.

Para contextualizar para que sea posible guardar un precio deben cumplirse las siguiente pautas:

- El cuerpo del mensaje debe contener los siguientes campos: *valor*, *idQueso* y *idTipoCliente.*
- Todos los campos deben tener un valor mayor o igual 1.
- En caso de guardarse exitosamente el precio se debe retornar:
    - un campo *data* que contenga como contenido los datos del precio guardado (*id*, *valor*, *idQueso* y *idTipoCliente*)
    - un campo message con el mensaje de éxito “*Se creó el precio correctamente*”
    - código de estado igual a 200
- En caso de error se debe retornar:
    - un campo *message* informando del error ocurrido
    - un código de estado que refleje el error ocurrido
        - 400: cuando fallan las validaciones sobre los campos del mensaje

enviado
- 401: cuando se hace un petición sin permisos adecuados
- 404: cuando se trata de guardar un precio de un queso o un tipo de cliente que no existe
- 409: cuando se trata de guardar un precio que ya existe

Entonces tenemos los siguientes tests:

1. Petición exitosa: se cumple que todos los campos son válidos, el queso y el tipo de cliente existe, no se guardó previamente el precio y se tiene los permisos necesarios.
2. Permisos faltantes: no se poseen los permisos suficientes para poder guardar el precio.
3. Validaciones sobre campos enviados:
    a. campos faltantes: no existe algún campo en el cuerpo del mensaje que sea obligatorio para el guardado del precio
    b. campos con valores menores a 1: existe algún campo con valor menor a 1.
4. Validaciones de reglas de negocio:
    a. precio ya existente: no se puede guardar un precio que ya se guardó previamente.
    b. queso no encontrado: no se puede guardar un precio respecto a un queso que no existe.
    c. tipo de cliente no encontrado: no se puede guardar un precio respecto a un tipo de cliente que no existe.

```
    @Test
    void Save__OK() {
        var dtoToSave = new PrecioDTO();
        dtoToSave.setValor(371.00);
        dtoToSave.setIdTipoCliente(3L);
        dtoToSave.setIdQueso(3L);

        var expectedPrecio = new PrecioDTO();
        expectedPrecio.setId(9L);
        expectedPrecio.setValor(371.00);
        expectedPrecio.setIdTipoCliente(3L);
        expectedPrecio.setIdQueso(3L);

        var response = rest.post(dtoToSave);
        assertNotNull(response.getBody());
        assertEquals(HttpStatus.OK, response.getStatusCode());

        var successfulResponse = rest.mapper().convertValue(response.getBody(), new
TypeReference<SuccessfulResponse<PrecioDTO>>() {
        });
        assertEquals(SuccessfulMessages.MSG_PRECIO_CREATED, successfulResponse.getMessage());
        assertEquals(expectedPrecio, successfulResponse.getData());
    }
```

1 - Test petición exitosa

```java
@Test
void Save__Unauthorized() {
    var dtoToSave = new PrecioDTO();
    dtoToSave.setValor(371.00);
    dtoToSave.setIdTipoCliente(3L);
    dtoToSave.setIdQueso(3L);

    var thrown = assertThrows(
            HttpClientErrorException.Unauthorized.class, () -> rest.postUnauthorized(dtoToSave)
    );
    assertEquals(HttpStatus.UNAUTHORIZED, thrown.getStatusCode());
}
```

2 - Test permisos faltantes

```java
@Test
void Save__Invalid_Fields__Fields_Not_Found() throws JsonProcessingException {
    var dtoToSave = new PrecioDTO();
    dtoToSave.setId(1L);

    var thrown = assertThrows(
            HttpClientErrorException.BadRequest.class, () -> rest.post(dtoToSave)
    );

    var response = rest.mapper().readValue(thrown.getResponseBodyAsString(), ErrorResponse.class);
    assertEquals(HttpStatus.BAD_REQUEST, thrown.getStatusCode());
    assertEquals(ErrorMessages.MSG_INVALID_BODY, response.getMessage());
    assertEquals(3, response.getErrors().size());
    assertEquals(ValidationMessages.NOT_FOUND, response.getErrors().get("valor"));
    assertEquals(ValidationMessages.NOT_FOUND, response.getErrors().get("idQueso"));
    assertEquals(ValidationMessages.NOT_FOUND, response.getErrors().get("idTipoCliente"));
}
```

3.a - Test validaciones sobre campos enviados: campos faltantes

```
    @Test
    void Save__Invalid_Fields__Other_Validations() throws JsonProcessingException {
        var dtoToSave = new PrecioDTO();
        dtoToSave.setValor(0D);
        dtoToSave.setIdTipoCliente(0L);
        dtoToSave.setIdQueso(0L);

        var thrown = assertThrows(
                HttpClientErrorException.BadRequest.class, () -> rest.post(dtoToSave)
        );

        var response = rest.mapper().readValue(thrown.getResponseBodyAsString(), ErrorResponse.class);
        assertEquals(HttpStatus.BAD_REQUEST, thrown.getStatusCode());
        assertEquals(ErrorMessages.MSG_INVALID_BODY, response.getMessage());
        assertEquals(3, response.getErrors().size());
        assertEquals(ValidationMessages.CANNOT_BE_LESS_THAN_1, response.getErrors().get("valor"));
        assertEquals(ValidationMessages.CANNOT_BE_LESS_THAN_1, response.getErrors().get("idQueso"));
        assertEquals(ValidationMessages.CANNOT_BE_LESS_THAN_1,
response.getErrors().get("idTipoCliente"));
    }
```

3.b - Test validaciones sobre campos enviados: campos con valores menores a 1

```
    @Test
    void Save__Precio_Already_Exists_Conflict() throws JsonProcessingException {
        var dtoToSave = new PrecioDTO();
        dtoToSave.setValor(371.00);
        dtoToSave.setIdTipoCliente(1L);
        dtoToSave.setIdQueso(1L);

        var thrown = assertThrows(
                HttpClientErrorException.Conflict.class, () -> rest.post(dtoToSave)
        );
        var response = rest.mapper().readValue(thrown.getResponseBodyAsString(), ErrorResponse.class);
        assertEquals(HttpStatus.CONFLICT, thrown.getStatusCode());
        assertEquals(ErrorMessages.MSG_PRECIO_ALREADY_EXISTS, response.getMessage());
    }
```

4.a - Test validación regla de negocio: precio ya existente

```
@Test
void Save__Queso_Not_Found_Conflict() throws JsonProcessingException {
    var dtoToSave = new PrecioDTO();
    dtoToSave.setValor(371.00);
    dtoToSave.setIdTipoCliente(1L);
    dtoToSave.setIdQueso(5L);

    var thrown = assertThrows(
            HttpClientErrorException.Conflict.class, () -> rest.post(dtoToSave)
    );
    var response = rest.mapper().readValue(thrown.getResponseBodyAsString(), ErrorResponse.class);
    assertEquals(HttpStatus.CONFLICT, thrown.getStatusCode());
    assertEquals(ErrorMessages.MSG_QUESO_NOT_FOUND, response.getMessage());
}
```

4.b - Test validación regla de negocio: queso no encontrado

```
@Test
void Save__Tipo_Cliente_Not_Found_Conflict() throws JsonProcessingException {
    var dtoToSave = new PrecioDTO();
    dtoToSave.setValor(371.00);
    dtoToSave.setIdTipoCliente(4L);
    dtoToSave.setIdQueso(1L);

    var thrown = assertThrows(
            HttpClientErrorException.Conflict.class, () -> rest.post(dtoToSave)
    );
    var response = rest.mapper().readValue(thrown.getResponseBodyAsString(), ErrorResponse.class);
    assertEquals(HttpStatus.CONFLICT, thrown.getStatusCode());
    assertEquals(ErrorMessages.MSG_TIPO_CLIENTE_NOT_FOUND, response.getMessage());
}
```

4.c - Test validación regla de negocio: tipo de cliente no encontrado

6.4.3 - Pruebas de validación

Las pruebas de validación del sistema se realizan a lo largo del proceso de desarrollo, por un lado, incluyendo a los interesados en los diseños de las interfaces por medio de la presentación de spikes y bocetos que ayudan a definir las funcionalidades, y por otro lado en las reuniones que se realizan entre iteraciones, donde se presenta lo trabajado hasta el momento y los usuarios validan las funcionalidades, y definen si se aceptan o debe haber cambios.

6.4.4 - Pruebas de despliegue

El proceso de despliegue del sistema se explica en el siguiente capítulo, sin embargo, consideramos necesario incluir este inciso dentro del capítulo de pruebas. A diferencia de

las pruebas que se explican en el siguiente capítulo, las pruebas de despliegue consideradas en este inciso abarcan realizar pruebas sobre el sistema ya desplegado, para verificar su funcionamiento.

Dentro de las pruebas que se realizan en este punto, la más trivial es entrar al sistema desde un navegador web sobre la misma máquina que se utiliza como servidor y autenticarse, que en caso de no dar error, indica que el sistema se encuentra en funcionamiento y hay comunicación entre las principales partes del sistema: *frontend, backend y base de datos.*

La siguiente prueba es navegar por las diferentes opciones del sistema, cargar algunos datos y verificar que los mismos se están persistiendo y mostrando en las consultas que se realizan desde el sitio frontend.

Seguido de esto se debe verificar que el sistema sea capaz de generar remitos, ya que para generarlos utiliza algunos archivos que de no estar cargados correctamente, impedirán que se utilice la funcionalidad.

Otro punto importante a probar es que la aplicación pueda ser accedida correctamente desde cualquier equipo conectado a la red, por lo que para esto se ingresa la dirección de la aplicación en el navegador de otro equipo conectado a la red, y se realiza la prueba de autenticarse en el sistema, que si retorna un error, indicará que hay algún fallo en las configuraciones del mismo.

Finalmente, se debe testear funcionamiento del escáner de código de barras, por lo que en el equipo en el que se utilizará el mismo, se debe acceder al sistema, conectar el escáner, y en la pantalla correspondiente realizar escaneos para verificar que está funcionando correctamente.

6.4.5 - Pruebas de lectura de códigos de barra

Para la prueba del componente encargado de leer e interpretar los códigos de barras (en la sección 8.3.2.1 se detalla esta funcionalidad), se diseñan un conjunto de pruebas teniendo en cuenta el dispositivo que se utiliza para la lectura de los códigos.

Los códigos en las etiquetas están conformados por tres secciones: el número de lote, un relleno de ceros, y el peso de la caja. El número o identificador de lote corresponde a la primera sección del código, y está compuesto por 12 o 13 dígitos, como se detalla en la primera historia de usuario presente en el Anexo B: está formado por la fecha de elaboración, el código del queso (Q) y el número de tina (T) (ddmmaaaaQQQT). El peso de la caja corresponde a la última sección del código y está compuesto de 4 dígitos. El relleno con ceros varía en tamaño hasta completar un código de 23 dígitos.

A continuación se muestra el conjunto de códigos que se utilizó para probar el componente con el escáner de código de barras, en conjunto con una descripción de los resultados esperados de la lectura de los mismos.

Estos tres primeros códigos pertenecen al mismo lote, pero poseen pesos diferentes. Los tres códigos se deben leer de manera correcta y sin causar errores.

El siguiente código, pertenece a un lote diferente al anterior, y en el proceso de carga de una expedición solo puede hacer un lote al cual estará asociada, por lo que si se lee en el proceso de carga de la misma expedición, debe informar un error y no agregar el código a la lista de códigos leídos.

El último código que se diseña para la prueba, consiste en un número de lote que posee un dígito más, ya que hay casos en los que por la conformación del número de lote, se puede dar esta situación. El resultado esperado ante esta lectura, es que sea correctamente identificado el número de lote de 13 dígitos, y el peso indicado en la etiqueta.

6.5 - Refactorización de código

La refactorización de código, o reescritura de código, es otra de las prácticas sugeridas por XP que tiene por objetivo mejorar el código escrito, ya sea para hacerlo más eficiente, o más entendible.

Dentro del proyecto aplicamos esta práctica en todos los momentos de desarrollo, teniendo en cuenta el principio de “propiedad colectiva del código”. Los requisitos para realizar este proceso, fueron que la función o el componente que fuera reescrito, debería seguir pasando los test ya definidos.

A continuación, detallamos a modo de ejemplo una refactorización hecha en la capa de servicio del lote, mostramos como se sigue el proceso para refactorizar los métodos *update()* y *delete()*.

En primer lugar, ambos métodos verifican que el lote exista antes de seguir con su flujo, y en caso contrario lanzan una excepción. Para el método *delete()* además se abstrajo el dar de baja el lote a un método aparte.

Las justificación para la refactorización hecha en este ejemplo cumple con dos simples razones:

1. *En primer lugar, la de reutilización del código*. En este caso ambos métodos verifican que el lote exista previamente antes de continuar su ejecución, y antes de la refactorización, existía código duplicado para realizar esta operación.
2. *Entender de mejor manera lo que está sucediendo en el método principal.* Cuando se lee el método *delete()* previo a la refactorización, hay que adentrarse en cómo lleva a cabo la funcionalidad para poder entender el método. Posteriormente a la refactorización basta con leer el código por arriba para entender el funcionamiento del método.

Antes de refactorizar

```
    public LoteDTO update(LoteUpdateDTO dtpUpdate) throws QuesoNotFoundConflictException,
LoteNotFoundException {
        var dto = new LoteDTO(dtpUpdate);
        if (!repository.existsByIdNotFechaBaja(dto.getId())) {
            throw new LoteNotFoundException();
        }
        repository.deleteById(dto.getId());
        return persist(dto);
    }
```

Método update() inicial

```
    public String delete(String id) throws LoteNotFoundException {
        if (!repository.existsByIdNotFechaBaja(id))
            throw new LoteNotFoundException();

        var lote = repository.getById(id);

        if (expedicionRepository.existsByLote(lote) || devolucionService.existsByLote(lote)) {
            lote.setFechaBaja(dateUtil.now());
            repository.save(lote);
            return id;
        }

        repository.deleteById(id);
        return "";
    }
```

Método delete() inicial

Después de refactorizar

```
    public LoteDTO update(LoteUpdateDTO dtoUpdate) throws QuesoNotFoundConflictException,
LoteNotFoundException {
        var dto = new LoteDTO(dtoUpdate);
        checkExistenceLote(dto.getId());
        return persist(dto);
    }
```

Método update() final

```
public void delete(String id) throws LoteNotFoundException {
    checkExistenceLote(id);

    if (expedicionRepository.existsByIdLote(id) || devolucionRepository.existsByIdLote(id))
        darBajaLote(id);
    else
        repository.deleteById(id);
}
```

Método delete() final

```
private void checkExistenceLote(String id) {
    if (!repository.existsByIdNotFechaBaja(id)) {
        throw new LoteNotFoundException();
    }
}
```

Método extraído checkExistenceLote()

```
private void darBajaLote(String id) {
    var lote = repository.getById(id);
    lote.setFechaBaja(dateUtil.now());
    repository.save(lote);
}
```

Método extraído darBajaLote()

7 - Despliegue

Al finalizar la tercera iteración en la etapa de desarrollo del sistema, que consiste en dar los últimos ajustes para la versión mínima del mismo, se comienza con la etapa de ***despliegue***, la cual se describe a lo largo de este capítulo. Ésta etapa se divide en tres actividades que se detallan a continuación: pruebas de despliegue, migración de datos y puesta en producción.

7.1 - Pruebas del proceso de despliegue

En primera instancia en la etapa de despliegue se procede a realizar pruebas del proceso de despliegue con el objetivo de acotar el margen de fallas que se puedan dar al momento de poner el sistema en producción.

Más específicamente, en esta etapa se revisan y actualizan los archivos *Dockerfile* y *docker-compose* que se escribieron en un principio al momento de realizar el ZFR (ver apartado 6.2.1), además de configurar de manera adecuada las variables de entorno.

Luego de lograda la correcta ejecución dentro de los entornos de desarrollo, se prosigue por realizar una instalación de cero en un sistema similar al que utiliza la PC que se utiliza como servidor en la empresa, ya que el sistema se desarrolla en entornos Linux y Mac, y el sistema se debe desplegar sobre un sistema operativo Windows. Dentro del sistema Windows se realiza la instalación de Docker, seguido de la importación de imágenes docker creadas del sistema y la base de datos, y el despliegue mediante la herramienta *docker-compose*.

Se realizan también pruebas de migración de datos, ya que se cuenta con la planilla de cálculo que utilizan para la producción, aunque con los datos desactualizados.

A lo largo de este proceso de pruebas de despliegue, se documentan los pasos que se deben seguir para la puesta en producción del sistema y migración de los datos. Los mismos son detallados en el inciso correspondiente.

7.2 - Migración de datos

Durante la reunión con los interesados luego de la segunda iteración, se define realizar una migración de datos para disminuir el tiempo de carga de los mismos para comenzar con el uso del sistema. También se especifica durante la reunión, los datos que deben ser migrados al sistema:

- Clientes
- Productos
- Lista de precios
- Toda la producción de la cual la empresa cuente con stock al día de la fecha.

Durante las pruebas de despliegue, se define realizar las siguientes actividades para la migración de los datos:

1. Como mencionamos anteriormente en el documento, la información existente se dispone en una planilla de cálculo, organizada de acuerdo a las

necesidades de la empresa. Esta planilla se toma como base para la migración.

2. Los datos se trabajan de modo que se definen valores por defecto en los campos que son requeridos para el funcionamiento del sistema y no existen en la planilla.
3. Luego se exportan los datos en formato CSV de acuerdo a lo requerido en el nuevo sistema.
4. Seguidamente se generan *scripts SQL* de inserción de datos para poblar la base de datos.
5. Una vez generados los scripts se procede a correrlos en la base de datos de desarrollo y se prueba el sistema.
6. Luego de verificado el correcto funcionamiento con los datos migrados en el sistema en desarrollo, se procede a la generación de un *script* para generar la base de datos y posteriormente importar los datos migrados.

7.3 - Puesta en producción

Como se mencionó anteriormente en la sección 7.1, durante las pruebas del despliegue del sistema, se genera documentación con los pasos a seguir para el despliegue. En la puesta en producción del sistema no se hace más que seguir este instructivo, cuyos pasos se detallan a continuación:

1. El día anterior al despliegue del sistema en producción se busca en la empresa la planilla de producción con los datos actualizados y se realizan las actividades descritas anteriormente para la migración de los datos (7.2).
2. Se configura una dirección IP fija en la PC servidor: esta dirección es a la que se hace referencia para ingresar al sistema.
3. Se configura también la dirección IP en las configuraciones del sistema y se construyen las imágenes Docker.
4. Se exportan las imágenes.
5. Se realiza una instalación de Docker en la máquina servidor.
6. Luego se importan las imágenes en el servidor y se corre el *docker-compose* para levantar la infraestructura.
7. En este punto se conecta con la base de datos por medio de DBeaver y se ejecuta el script para la creación de las tablas y la migración de los datos, también en este paso se cargan los usuarios en el sistema con los roles que se habían definido para los mismos.
8. Se verifica que la instalación del sistema sea correcta y se crean accesos directos al mismo en las diferentes máquinas que se debe acceder.

8 - Funcionamiento del sistema

En este capítulo se detallan las diferentes funcionalidades del sistema que fueron desarrolladas a lo largo del proyecto. A continuación, se detallan cada una de ellas, organizadas en secciones según el nombre que poseen en el menú principal del sistema, a excepción de la primera.

8.1 - Inicio de sesión

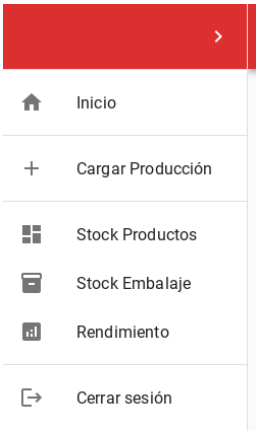
Al entrar a la aplicación por medio de un navegador web, la primera pantalla que se presenta al usuario es la de inicio de sesión. Por medio de esta pantalla, los usuarios accederán al sistema y dispondrán de las distintas funcionalidades que tengan asignadas según su rol.

Pantalla de inicio de sesión

8.2 - Página Principal

Luego de iniciar sesión, se presenta la pantalla principal del sistema, donde se dispone a la izquierda de un menú desplegable con las distintas funcionalidades dentro del mismo. Las opciones de este menú variarán según las credenciales del usuario y los accesos que tenga el mismo. A continuación, se muestra una imagen de la pantalla principal con todas las opciones del menú disponibles, seguido de un ejemplo del menú con funcionalidades más limitadas.

Pantalla principal del sistema



Menú con funcionalidad limitadas

El menú del sistema, se divide en cuatro partes: *carga, administración, consulta,* y finalmente, *visualización*. En las siguientes secciones se describen cada una de estas secciones a detalle con sus funcionalidades.

8.3 - Sección de carga

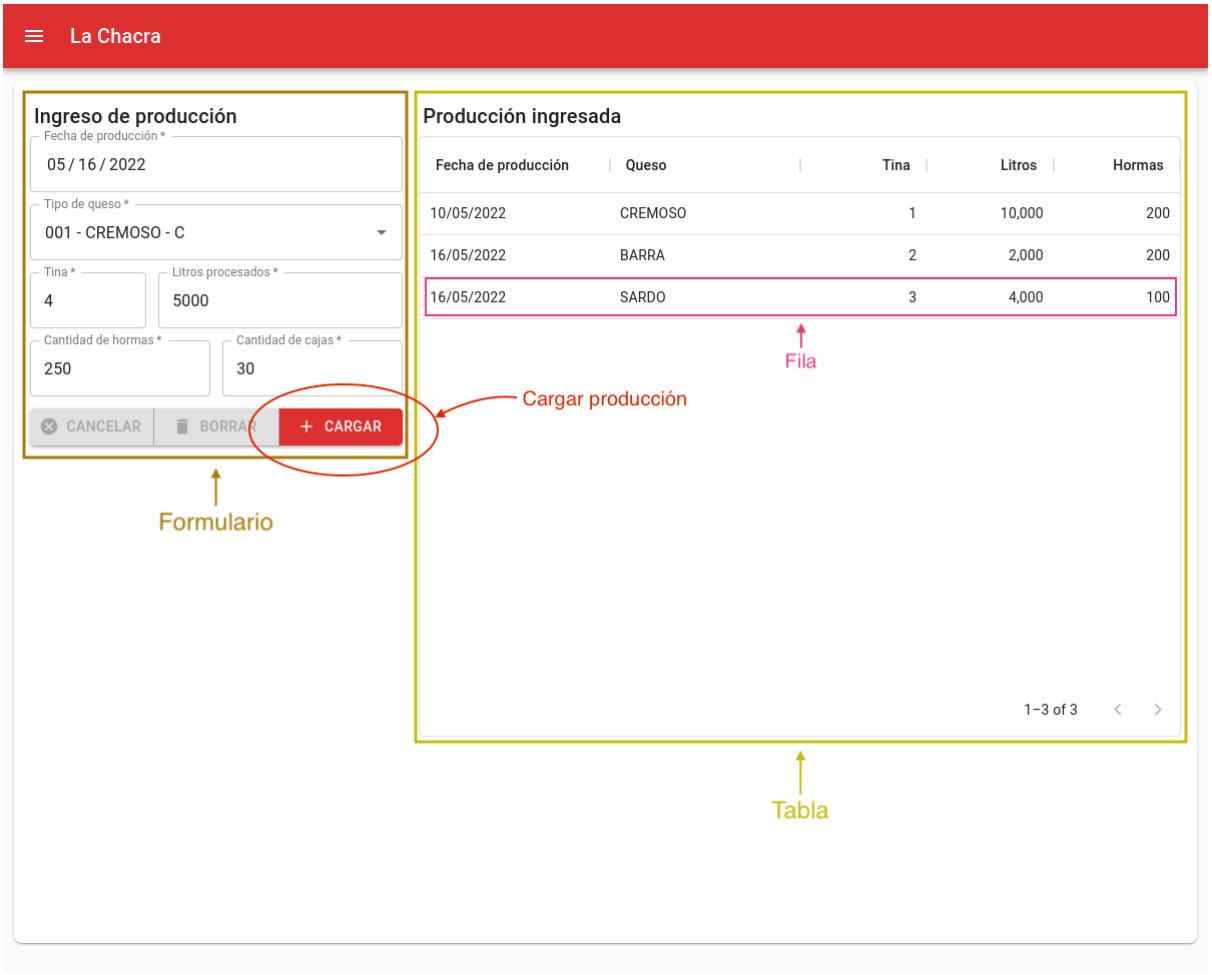
La primera sección del menú corresponde a la carga de producción, carga de expediciones y generación de remitos. Esta sección está destinada a los usuarios no administrativos en la empresa, cuya tarea dentro del sistema es la carga de estos datos. Los permisos de los usuarios no administrativos, se limitan a esta sección.

A continuación se describen cada una de estas opciones.

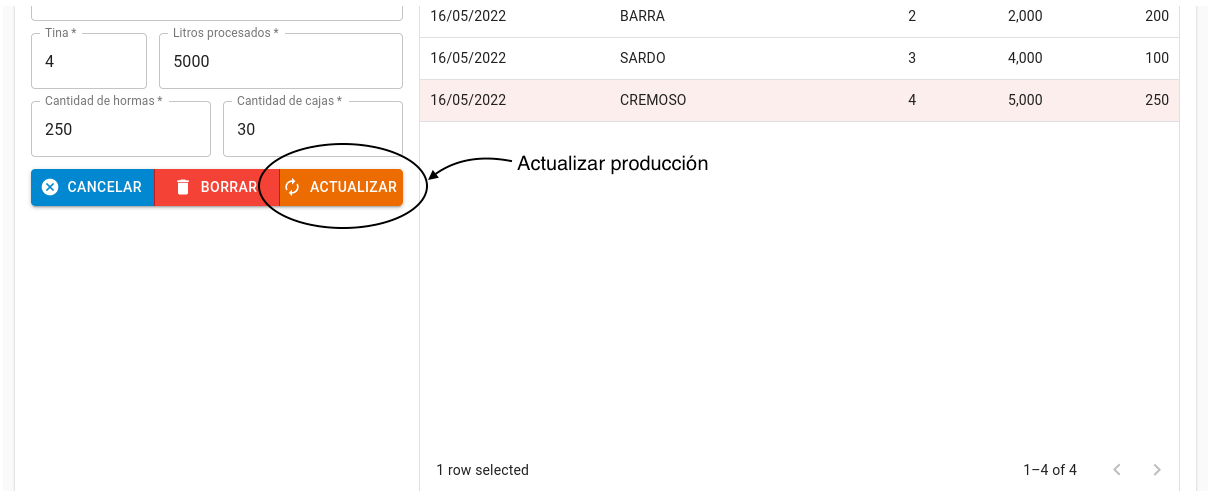
8.3.1 - Cargar producción

La primera opción disponible en el menú es la de cargar producción. Esta pantalla consiste en uno de los diseños de interfaz de usuario que se utilizó en el sistema, donde del lado izquierdo de la pantalla se ubica un formulario de carga de datos, y a la derecha, una tabla. Dentro de “Cargar Producción”, la tabla corresponde a producciones cargadas durante el período que el usuario permanezca en esta pantalla, con el objetivo de permitir editar datos cargados erróneamente. Si el usuario hace doble clic sobre una fila de la tabla, se selecciona esta producción cargada y se permite la edición de sus datos.

En el momento que el usuario selecciona una fila de la tabla para edición, se habilitan los otros dos botones debajo del formulario, permitiendo también borrar el lote de producción cargado o cancelar el proceso.



Pantalla de carga de producción



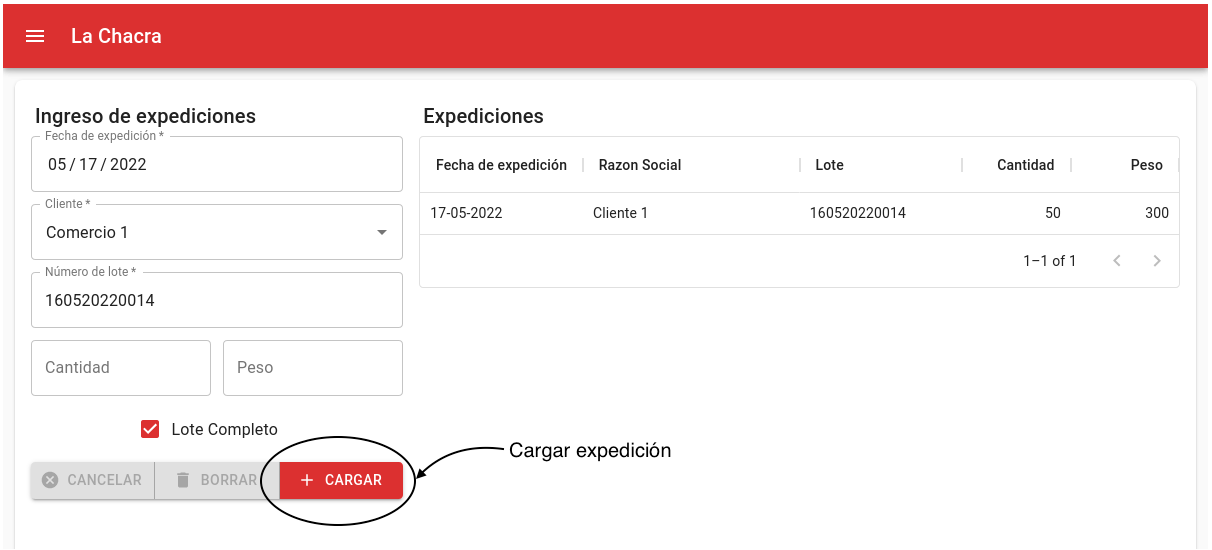
Edición de producción en pantalla de carga

8.3.2 - Cargar Expediciones

La siguiente opción dentro del menú del sistema, es la pantalla de carga de expediciones.

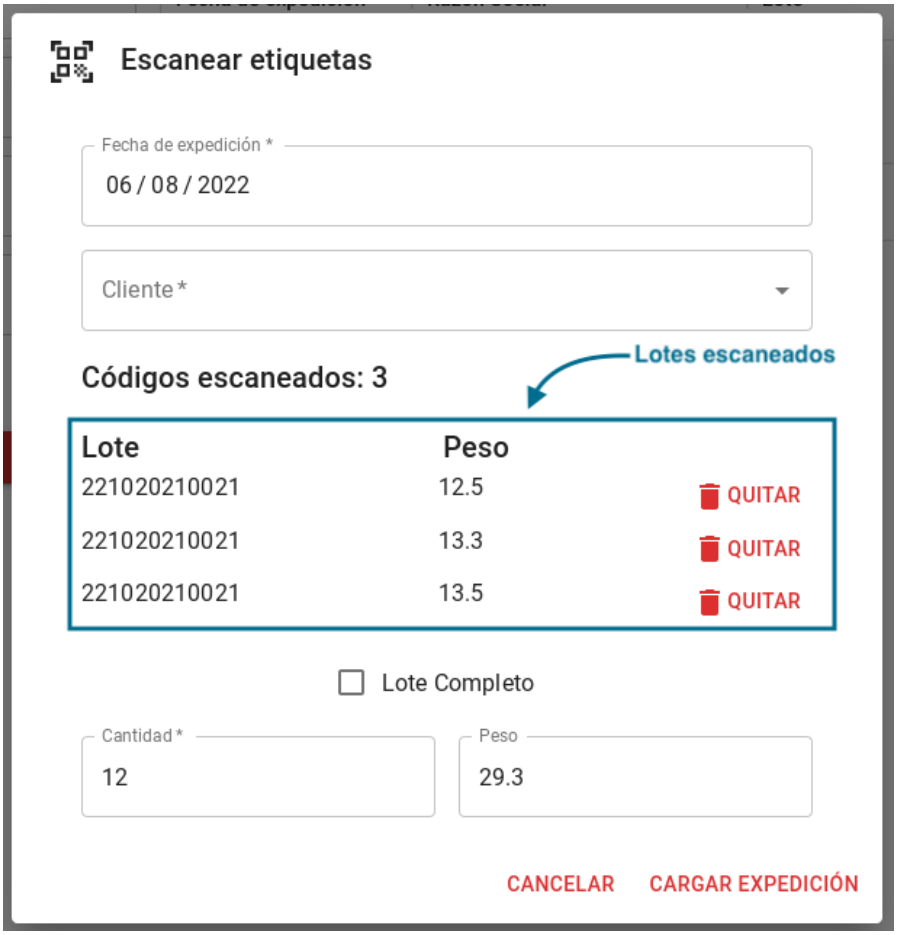
Esta pantalla sigue la misma lógica que la pantalla de carga de producción.

Dentro del sistema, la expedición asocia a un cliente con un lote en particular. En el momento en que se crea una nueva expedición, se decrementa el stock del correspondiente lote de producción, como también el del producto correspondiente.



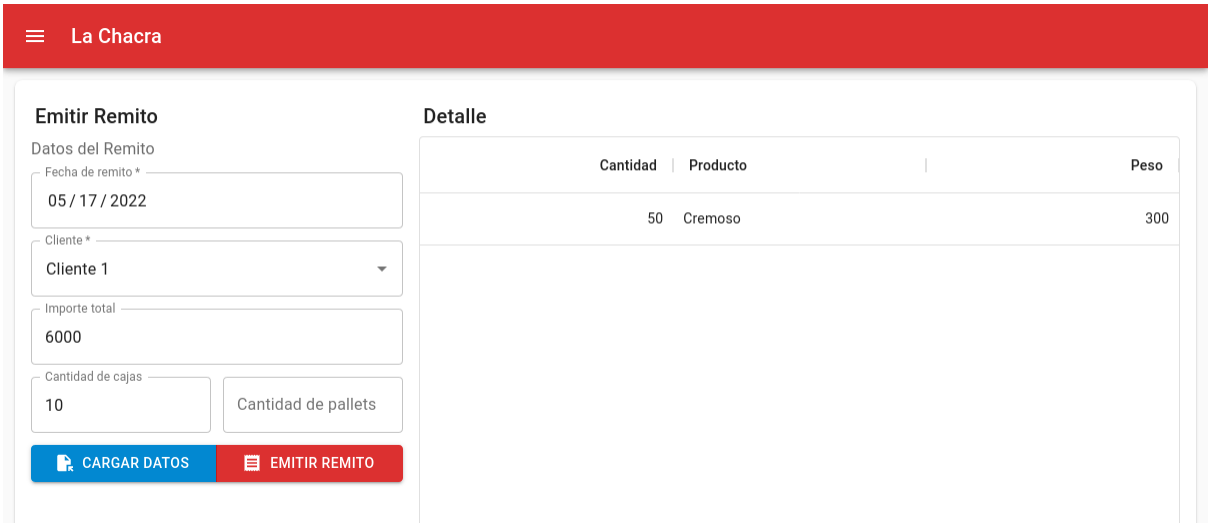
8.3.2.1 - Carga por codigo de barras

Alternativamente a la carga manual de expediciones, existe la posibilidad de que para un tipo de queso el usuario cargue el número de lote y el peso mediante la lectura de códigos de barras. Las expediciones se suelen formar por varias cajas de un mismo lote, por lo que se realizan lecturas de distintas etiquetas, y el peso de cada caja se va agregando en el total que es el peso de la expedición. Para acceder a la opción de cargar expediciones mediante código de barras, el usuario debe tener conectado a la computadora un escáner de código de barras, y simplemente escanear una de las etiquetas estando en la pantalla de expediciones, inmediatamente se abre el diálogo que se muestra a continuación, presentando los códigos escaneados y el resto de los datos de la expedición.

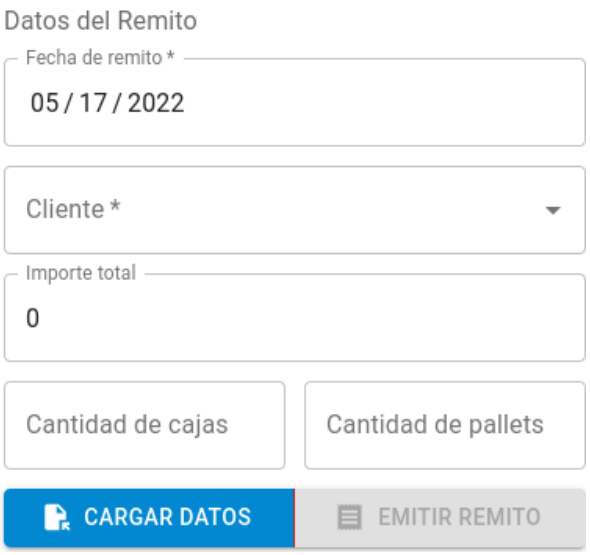


8.3.3 - Emitir remito

Una vez cargadas las expediciones para un cliente particular, se puede emitir el remito correspondiente para el mismo. Dentro de esta pantalla, el usuario debe, en primer lugar, seleccionar el cliente para el que se quiere hacer el remito, luego debe presionar en “Cargar Datos”, que como resultado, muestra en la tabla los datos que saldrán impresos en el remito, a modo de verificación. En caso de ser correctos estos datos, el usuario presiona el botón “Emitir Remito”, y en caso contrario, se pueden proceder a editar las expediciones según el apartado 8.5.2. El usuario puede completar la cantidad de cajas o pallets que conforman el pedido para el que se generó el remito y puede cambiar la fecha del remito que por defecto es la actual.



Pantalla de emisión de remitos



Formulario de emisión de remitos al cargar la página

8.4 - Sección de administración

La siguiente sección disponible dentro del menú, está conformada por las pantallas de administración de productos, precios y clientes de la empresa, con funcionalidades para crear, modificar, consultar, y dar de baja en cada una de las páginas.

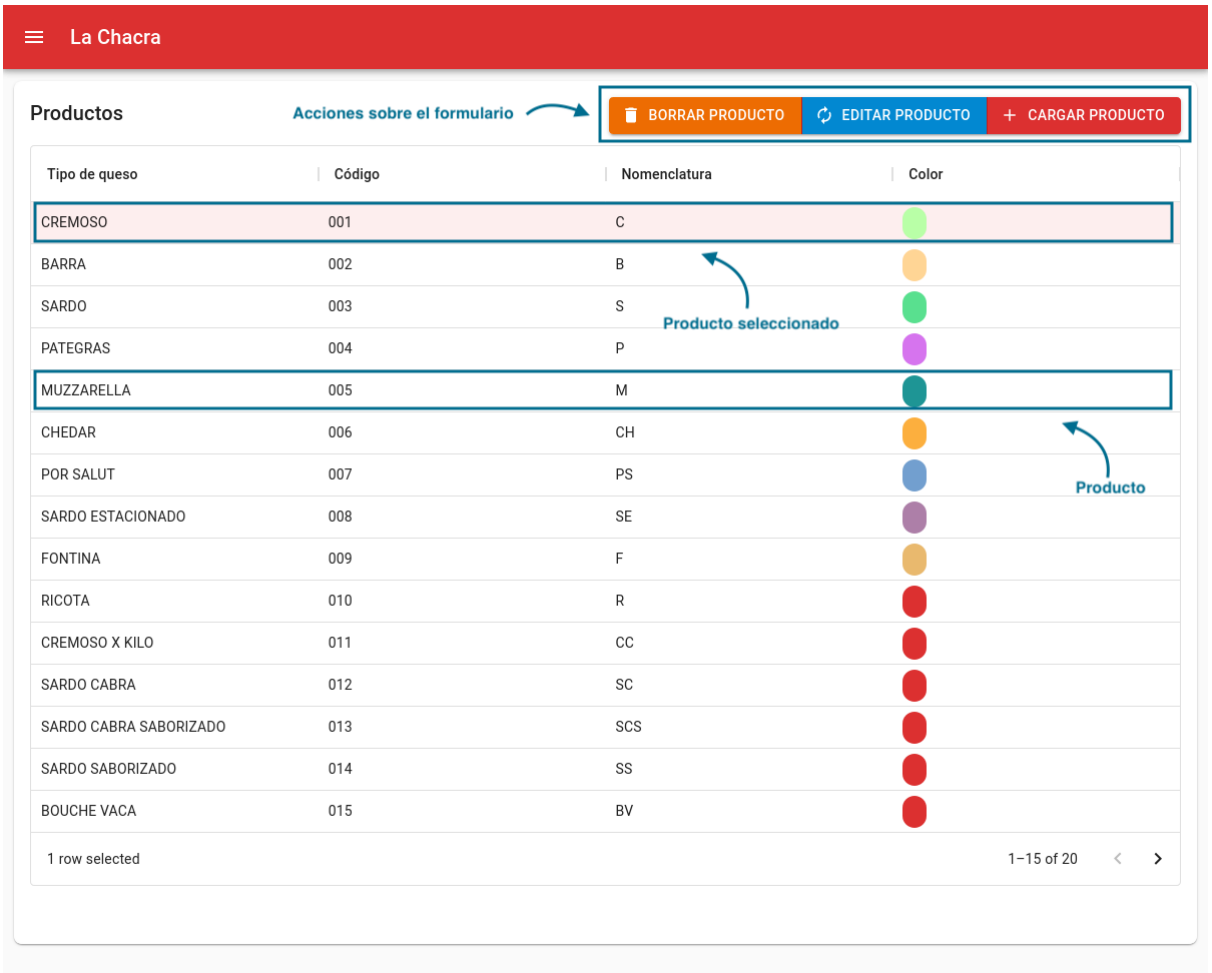
8.4.1 - Productos

La pantalla de productos sigue otro de los diseños de interfaces de usuario utilizados dentro del sistema. Se presenta una tabla que ocupa toda la pantalla mostrando, en este caso, los productos cargados en el sistema junto a la información de cada uno de ellos. Sobre la parte superior derecha de la tabla se encuentran 3 botones que permiten distintas acciones sobre los elementos de la tabla, permitiendo agregar nuevos ítems, actualizarlos o eliminarlos. Al agregar un nuevo ítem, o cargar un producto en este caso, se abre un diálogo

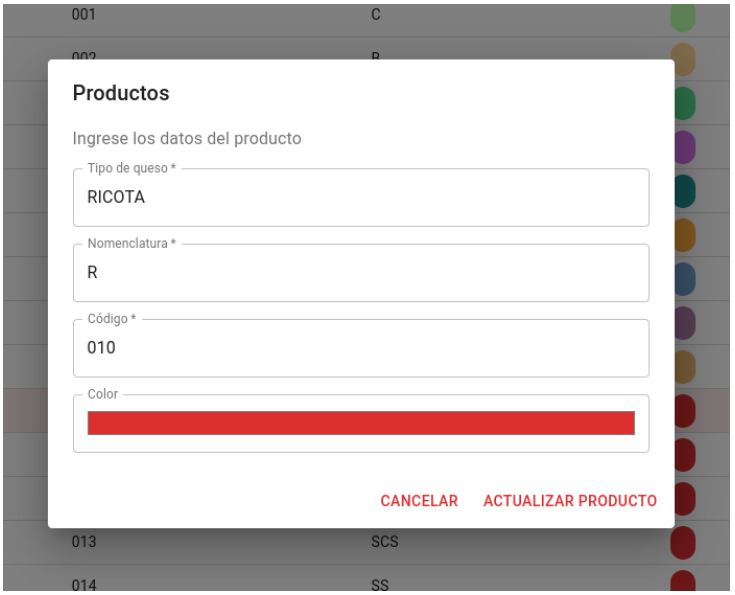
donde el usuario ingresa los diferentes datos del producto.

El usuario puede también editar un producto ya existente, seleccionando el mismo en la tabla y presionando el botón “Editar Producto”. Al realizar esta acción, se abre un diálogo similar al que se abre para crear un nuevo producto, donde el usuario puede editar todos los datos del mismo y actualizarlo.

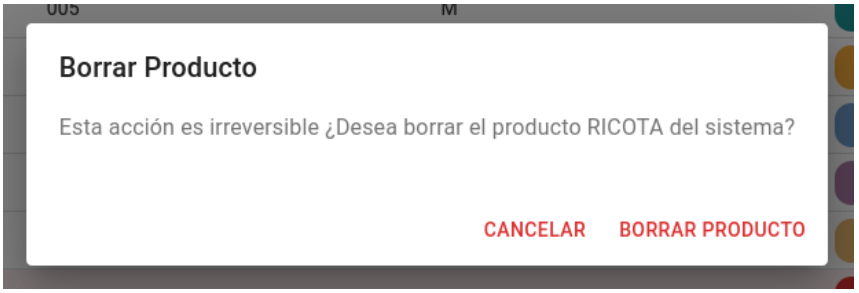
La tercera opción que tiene el usuario en esta pantalla es la de borrar un producto. Para borrar un producto, el usuario debe seleccionarlo en la tabla y posteriormente hacer clic en el botón “Borrar Producto”. El sistema muestra un diálogo de confirmación de la acción donde el usuario puede continuar con el borrado o bien, cancelar el proceso.



Pantalla de carga de productos



Diálogo de edición de producto



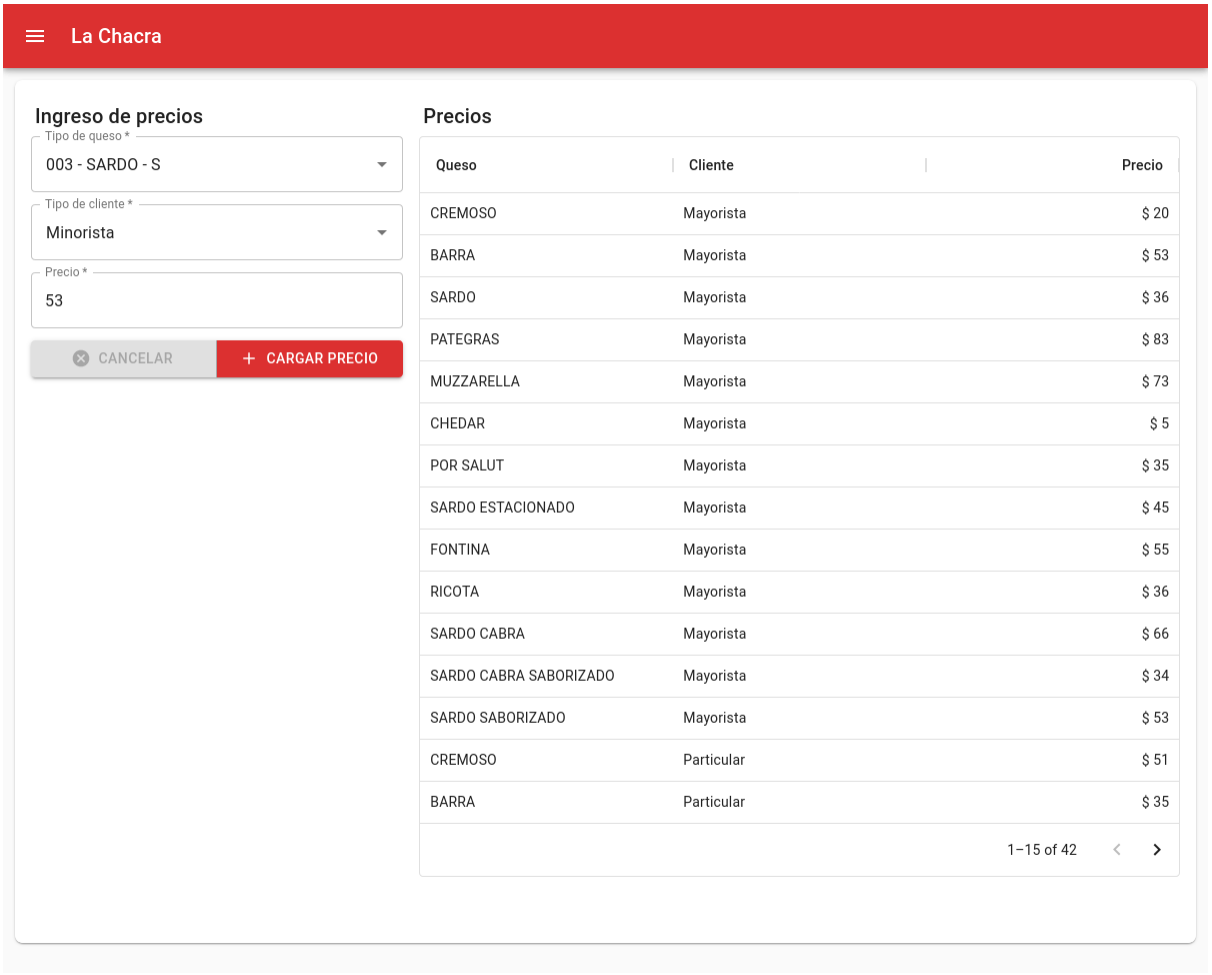
Diálogo de confirmación de borrado

8.4.2 - Precios

Continuando con las opciones en el menú, se encuentra la funcionalidad para cargar precios. Dentro de esta pantalla, se muestra un formulario para cargar precios, en conjunto con una tabla que muestra los precios ya cargados en el sistema.

Si el usuario desea cargar un nuevo precio, debe seleccionar el tipo de producto, y tipo de cliente al que estará asociado el precio, deberá completar el campo de precio, y hacer clic sobre el botón “Cargar Precio”.

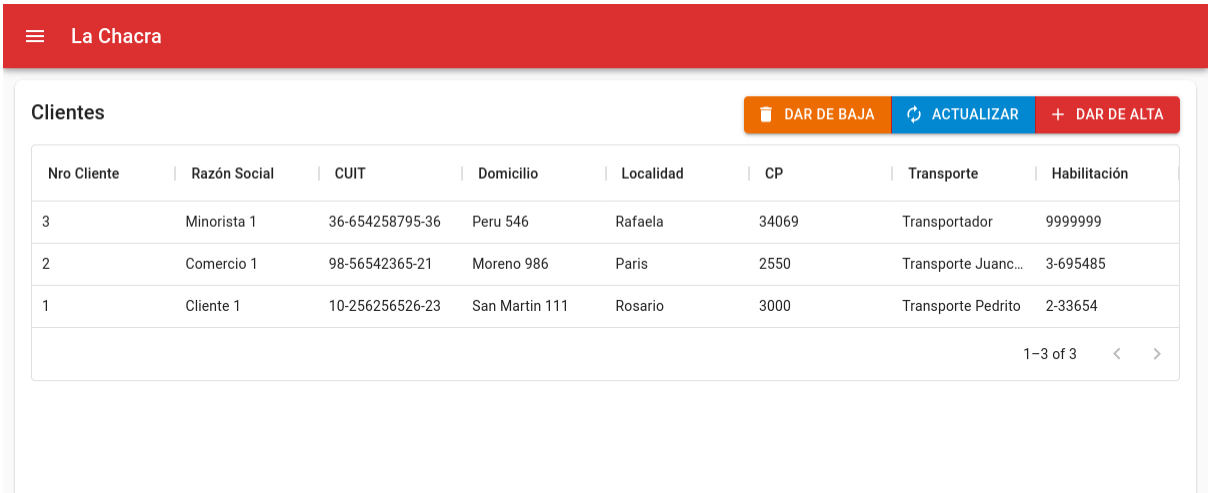
Para actualizar un precio, el flujo del proceso es similar al que se sigue en las pantallas “Cargar Producción” y “Cargar Expedición”, presentadas en los incisos 8.3.1 y 8.3.2 respectivamente.



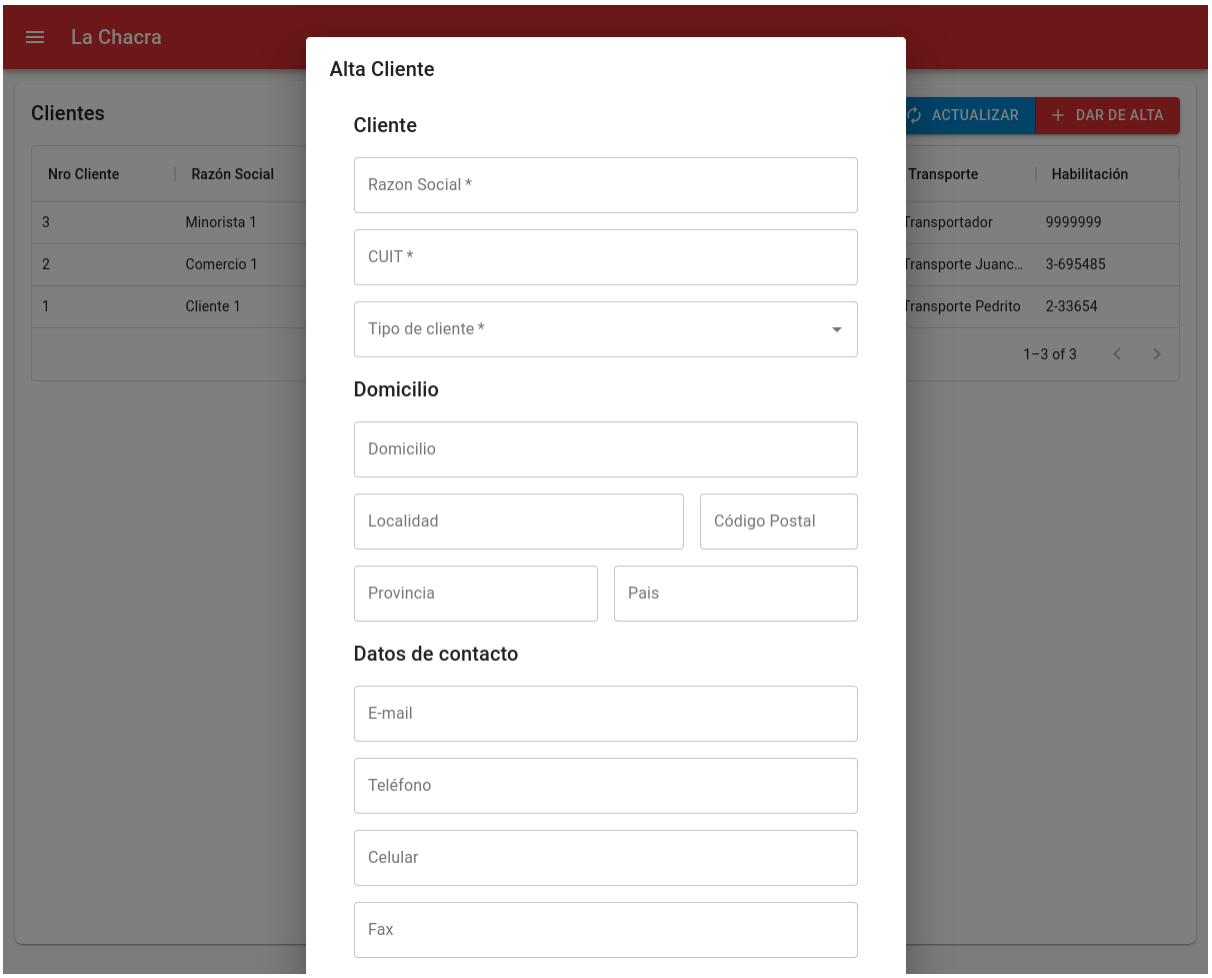
Pantalla de carga de precios

8.4.3 - Clientes

La pantalla de clientes sigue la misma lógica que la interfaz de usuario de productos detallada en el apartado 8.6, para cargar, actualizar y dar de baja de clientes.



Pantalla de gestión de clientes



Formulario de carga/edición de clientes

8.5 - Consulta

La tercera sección en el menú principal, está destinada a la consulta de los datos cargados en el sistema. Además de poder realizar consultas, el usuario tiene aquí la posibilidad de editar los datos cargados, como también borrarlos.

8.5.1 - Producción

El siguiente apartado en el menú del sistema es el de producción. Ésta es una pantalla de consulta de datos, donde del lado izquierdo se presentan unos campos que permiten búsquedas por diferentes intervalos de tiempo, y del lado derecho, una tabla que presenta los resultados.

Una particularidad de esta tabla, es que se muestra en distintos colores la celda correspondiente al saldo de la producción, según el lote esté empezado o no, sin stock,o con stock negativo.

El usuario puede editar cualquier lote presente en la tabla. Al hacer doble clic sobre una fila de la misma, se abrirá un diálogo con la información del lote, permitiendo actualizar la información del mismo. Además, dentro del diálogo existe la opción de borrar el lote, la cual primero se debe habilitar desde la esquina superior derecha para evitar borrar un lote por un descuido.

Pantalla de consulta de visualización

Diálogo de actualización de producción

8.5.2 - Expediciones

La pantalla de consulta de expediciones, posee el mismo diseño y las mismas funcionalidades que la pantalla anterior de consulta de producción, donde se realiza una consulta por un rango de tiempo, y se pueden editar las expediciones mostradas en la tabla como resultado de la consulta.

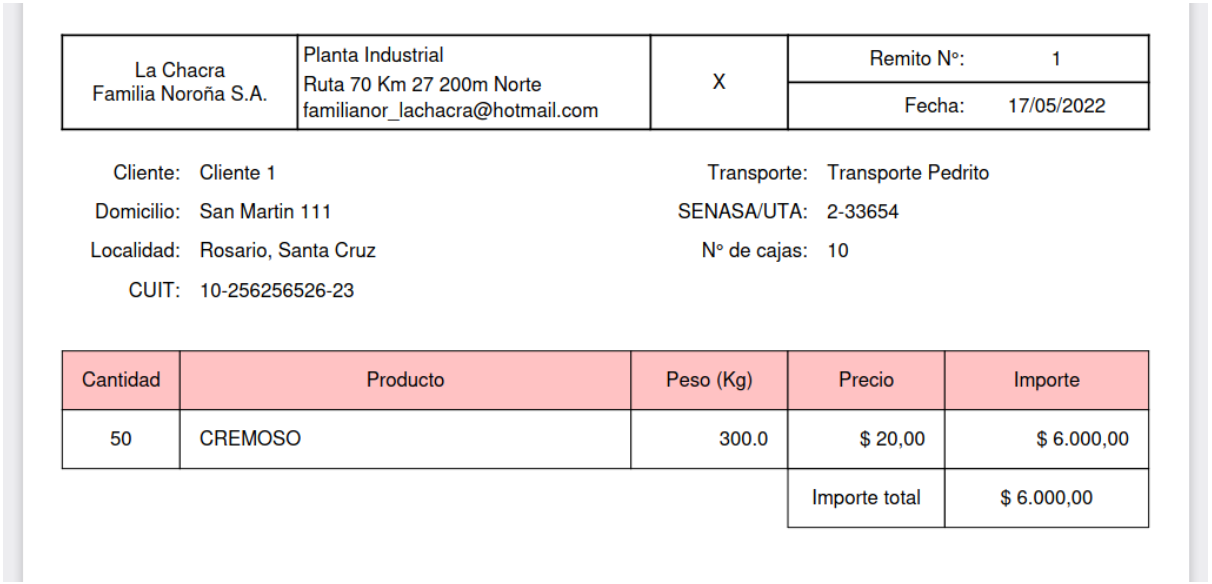
8.5.3 - Remitos

Dentro del menú del sistema, el usuario tiene la opción de ver los remitos generados en el sistema. Esta pantalla sigue el mismo diseño y funcionamiento que las dos anteriores.

El usuario tiene en cada una de las filas de la tabla las opciones de ver o anular el remito correspondiente. Si el usuario presiona en “ver” en una fila, el sistema generará un archivo en formato PDF del remito correspondiente y lo abrirá en el navegador. En cambio, si el usuario presiona “Anular”, se le mostrará un diálogo de confirmación permitiendo anular el

remito. Si el usuario anula el remito, también se borrarán las expediciones asociadas al mismo, devolviendo el stock a los los lotes de producción correspondientes.

Pantalla de consulta de remitos



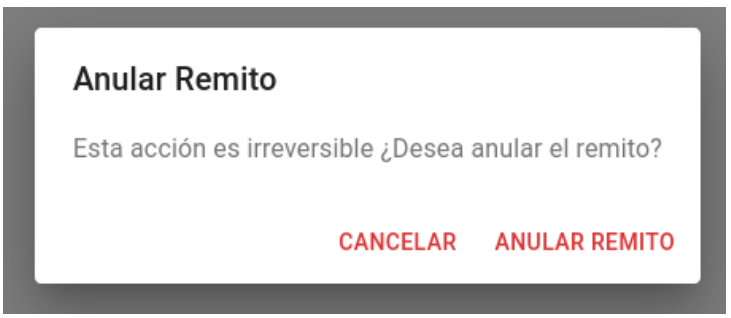
La Chacra Familia Noroña S.A.	Planta Industrial Ruta 70 Km 27 200m Norte familianor_lachacra@hotmail.com	X	Remito N°: 1 Fecha: 17/05/2022

Cliente: Cliente 1
Domicilio: San Martin 111
Localidad: Rosario, Santa Cruz
CUIT: 10-256256526-23

Transporte: Transporte Pedrito
SENASA/UTA: 2-33654
N° de cajas: 10

Cantidad	Producto	Peso (Kg)	Precio	Importe
50	CREMOSO	300.0	$ 20,00	$ 6.000,00
			Importe total	$ 6.000,00

Remito generado por el sistema

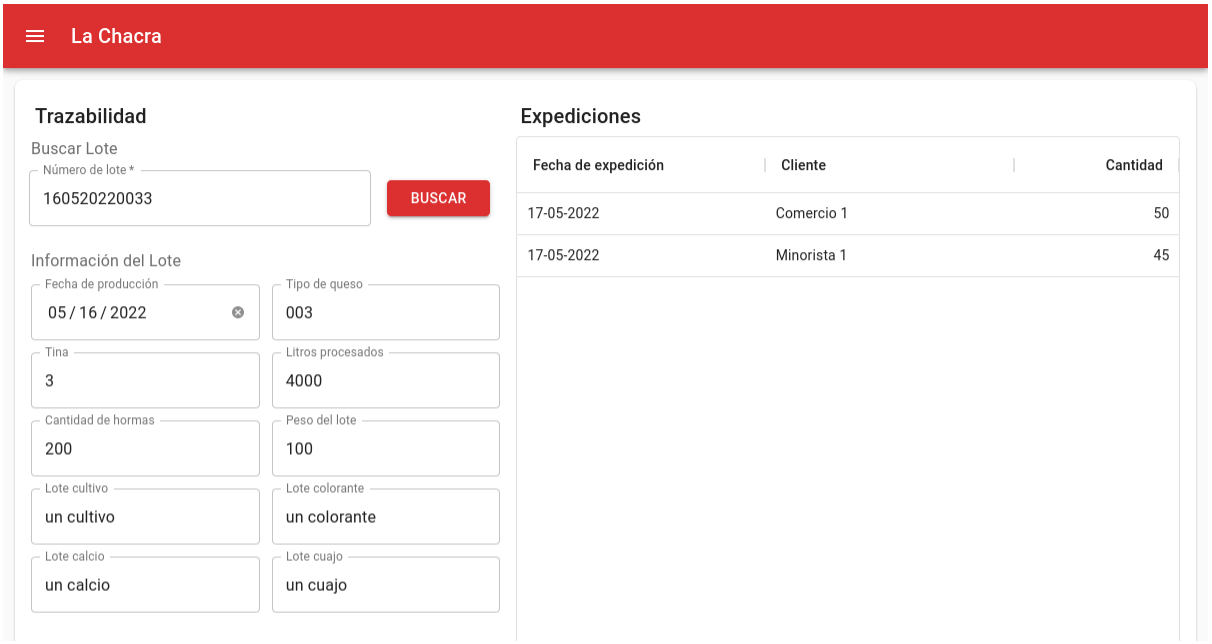


Diálogo de confirmación para anular remito

8.5.4 - Trazabilidad

El usuario también tiene la posibilidad de consultar la trazabilidad de un lote.

La trazabilidad de un lote, se conforma a partir de todos los datos que se hayan ingresado de producción, en conjunto con todas las expediciones que se hayan realizado del mismo. De este modo se tiene información de los clientes a quienes se les entregó el lote.



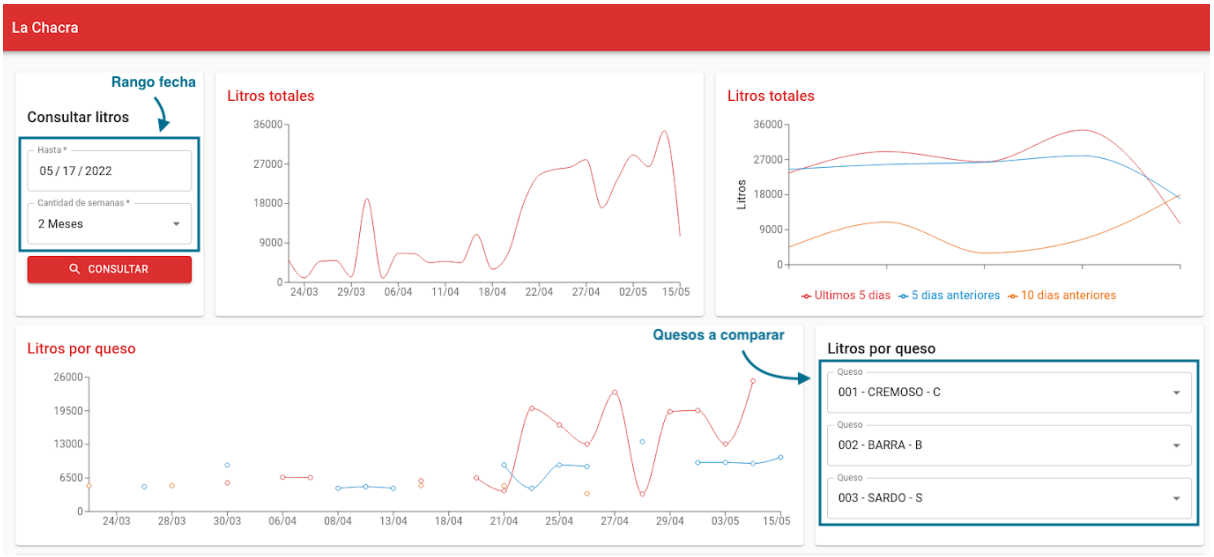
Pantalla de consulta de trazabilidad

8.6 - Visualización

La última sección disponible en el menú, corresponde a la visualización de información. Las distintas secciones en este apartado, se basan en dos diseños: por un lado se tiene un diseño con gráficas, y por el otro un diseño basado en “tarjetas” para las pantallas de visualización de stocks.

8.6.1 - Litros Elaborados

Esta primera pantalla permite visualizar los litros elaborados a lo largo de un determinado tiempo de consulta. Permite además, una comparación entre la elaboración de las tres últimas semanas, y una comparación entre los distintos tipos de productos.

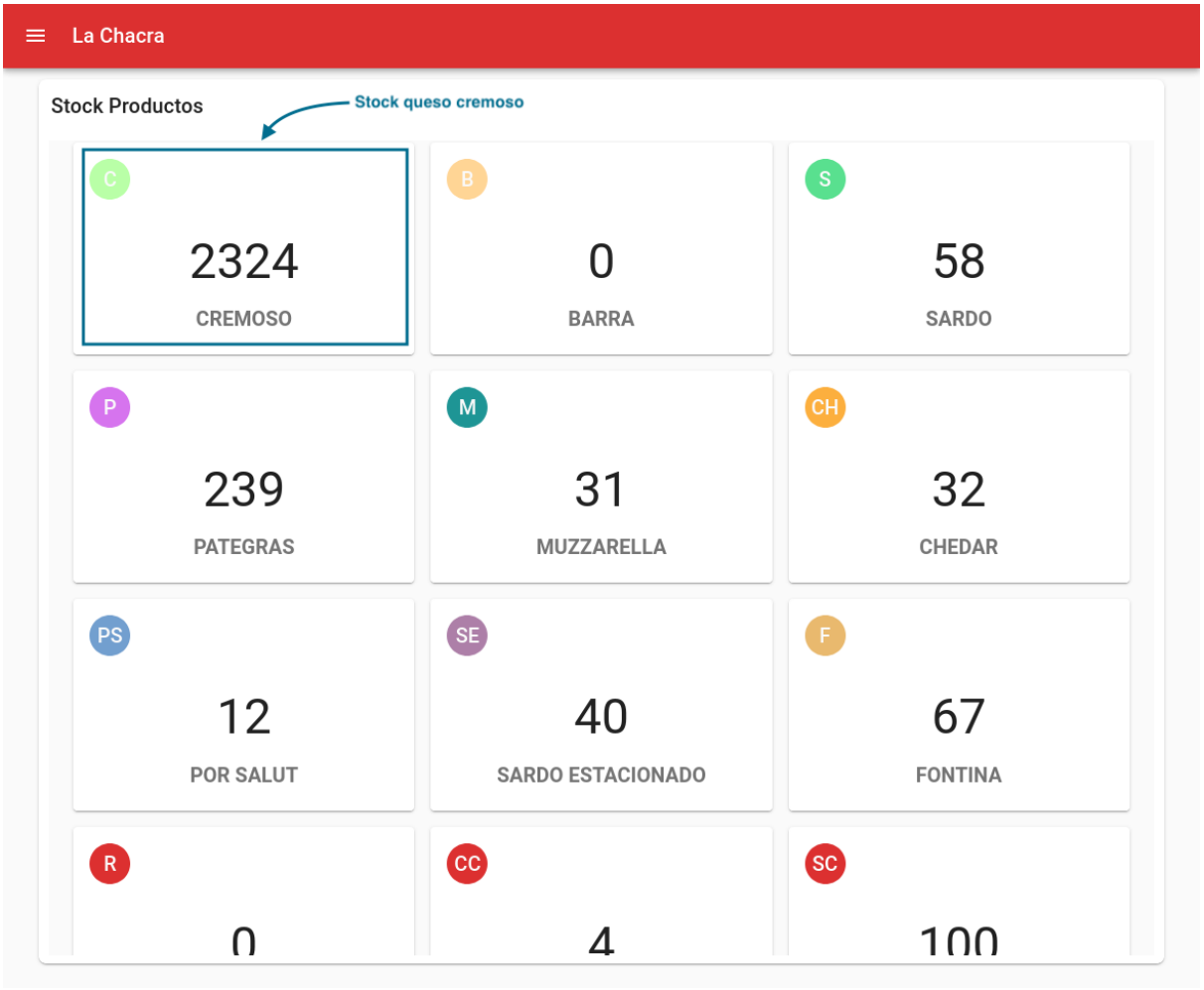


Pantalla de visualización de litros elaborados

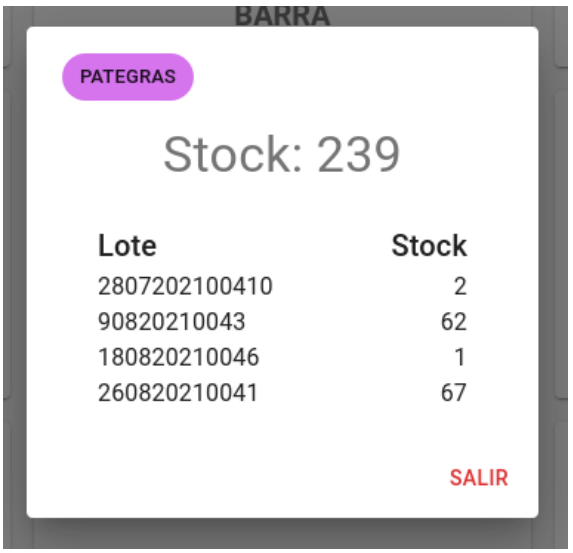
8.6.2 - Stock de Productos

Esta pantalla corresponde al segundo tipo de diseño anteriormente mencionado. Se presenta una lista con distintas tarjetas, con los stocks de los diferentes productos. El objetivo de este diseño es tener una rápida visualización del stock de los productos.

Dentro de esta pantalla, además de presentarse las distintas tarjetas, el usuario puede hacer clic sobre alguna de ellas y se le mostrará en un diálogo los lotes de producción pertenecientes a ese tipo de producto que actualmente cuentan con stock. A continuación, además de la pantalla, se muestra el diálogo de stock.



Pantalla de visualización de stock



Diálogo de visualización de stock

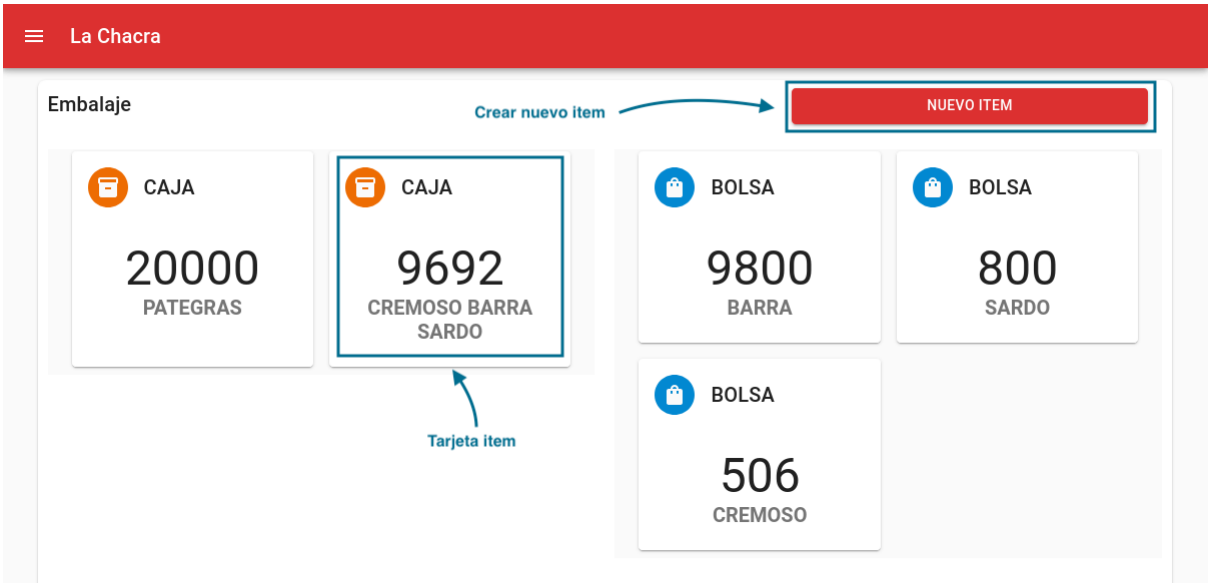
8.6.3 - Stock de Embalaje

La pantalla de stock de embalaje, sigue la misma idea que la de stock de productos, donde mediante tarjetas, se muestra el stock de los diferentes ítems. Dentro de esta pantalla, el usuario tiene dos opciones, puede hacer clic sobre alguna de estas tarjetas, o crear un nuevo ítem.

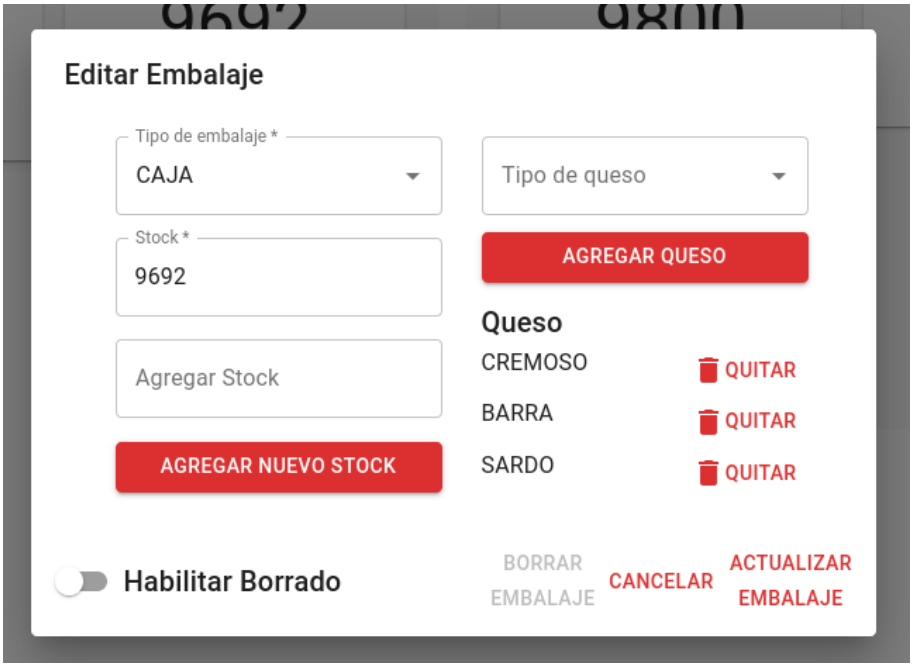
Si el usuario hace clic sobre alguna de estas tarjetas, se abre un diálogo que le permite actualizar el ítem, cambiando los productos a los que corresponde, editando directamente el stock, o agregando stock al ya existente. Adicionalmente, existe la opción de borrar un ítem idéntico a la explicada en 8.5.1.

Si el usuario desea crear un nuevo ítem de embalaje, debe presionar el botón “Nuevo Ítem”, mediante el cual se abre un diálogo similar al de edición, solo que sin el apartado para agregar nuevo stock. Dentro de éste diálogo, el usuario define el tipo de embalaje, que puede corresponder a caja o bolsa, también define el stock actual y los productos al que el mismo corresponde.

El stock de los distintos ítems de embalaje se decrementa a medida que se carga producción en el sistema, de modo que se decrementan tantas bolsas asociadas al producto como hormas se hayan cargado, y se decrementan las cajas asociadas al producto según la cantidad de cajas que se carguen con la producción.



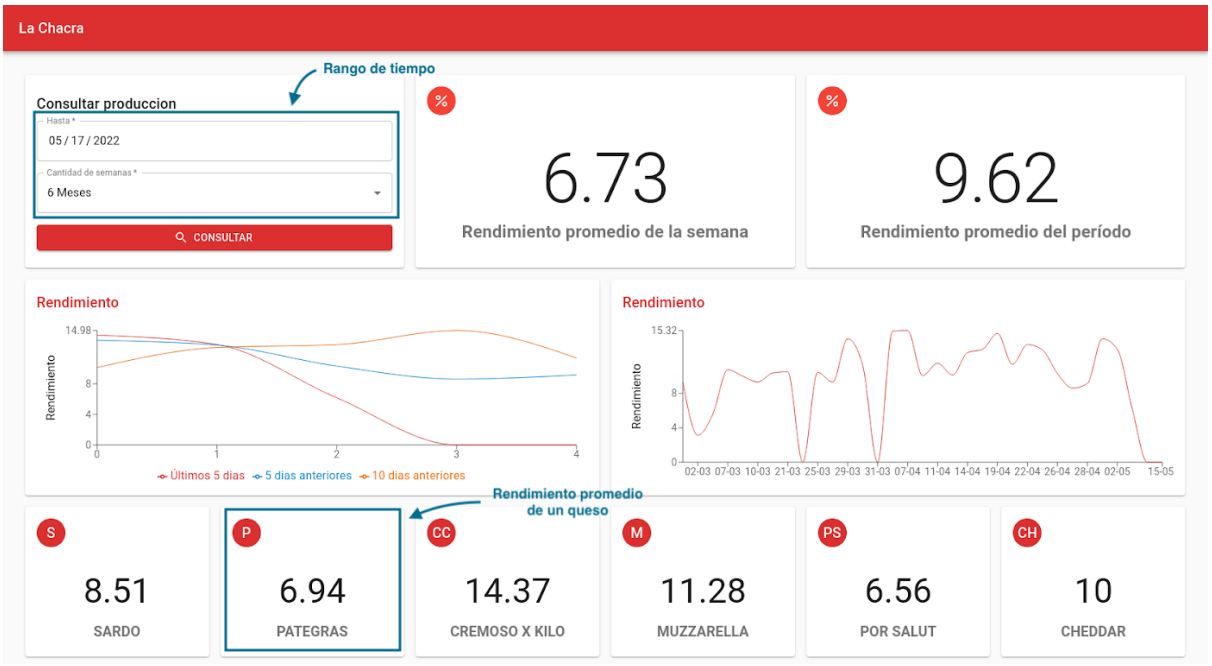
Pantalla de stock de embalaje



Diálogo de actualización de embalaje

8.6.4- Rendimiento

La siguiente pantalla corresponde a la visualización del rendimiento de la producción. Dentro de esta pantalla, no se encuentran más funcionalidades que realizar una consulta para un cierto rango de tiempo. A partir de la consulta, se visualiza el rendimiento promedio de la última semana, como también del período, y además, el rendimiento promedio por queso del período consultado.



Pantalla de visualización de rendimientos

8.6.5 - Ventas

Como última opción en el menú se encuentra la opción para ver las ventas que se realizaron. A pesar de su nombre, esta pantalla no muestra “ventas” en forma de flujo de dinero, sino que se conforma a partir de todos los pedidos que salen de la empresa, el nombre es por costumbre de los interesados.

La pantalla presenta el mismo diseño que la de “Litros Elaborados” presentada en la sección 8.6.1.

8.7 - Páginas alternativas

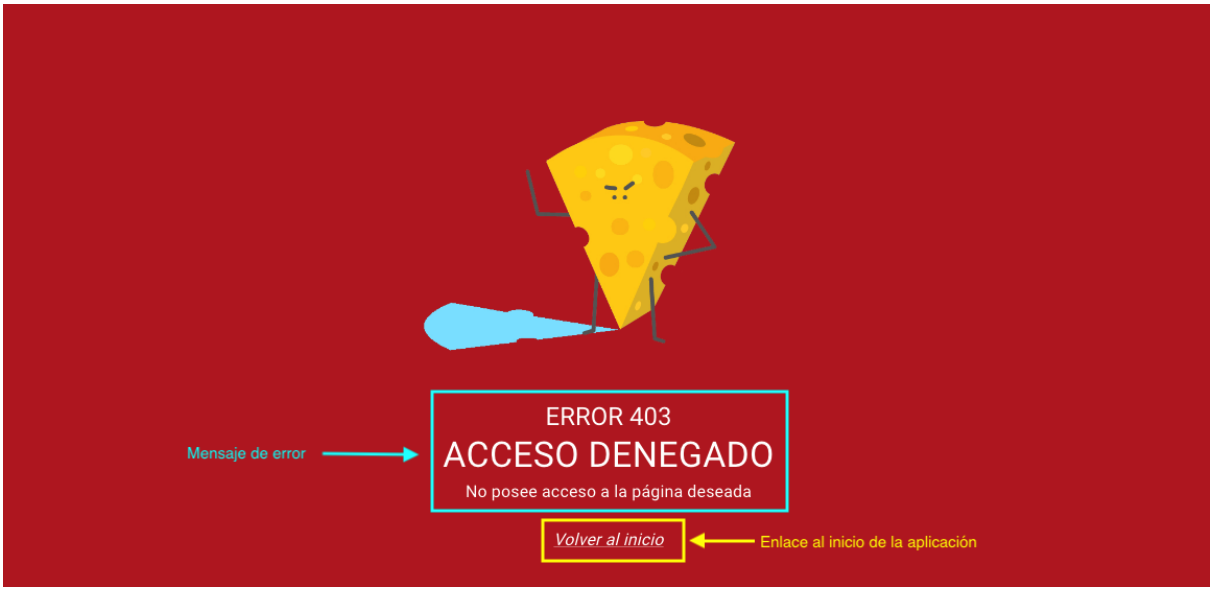
Diseñamos ciertas páginas que informan al usuario de sucesos inesperados. Estas páginas poseen el mensaje del suceso y un enlace al inicio de la aplicación.

8.7.1 - Página no encontrada

Esta página es lanzada cuando el usuario intenta acceder a una página que no existe en la aplicación.

8.7.2 - Página de acceso prohibido

Esta página es accedida cuando el usuario tiene iniciada la sesión e intenta acceder a un página a la cuál no tiene permiso de acceso.

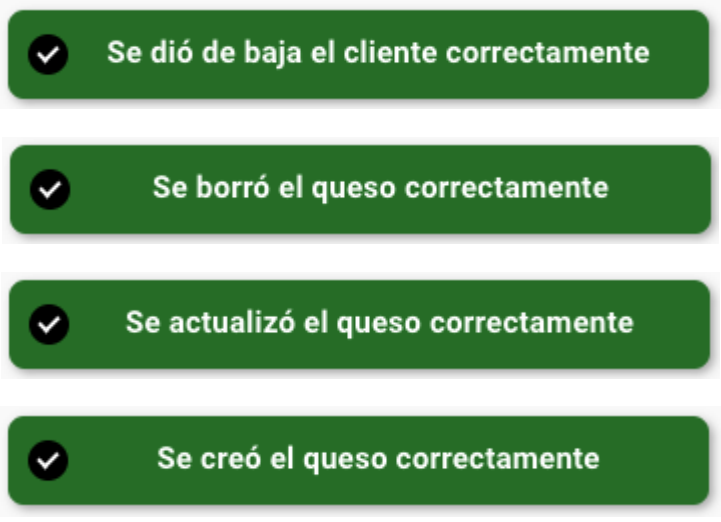


8.8 - Notificaciones

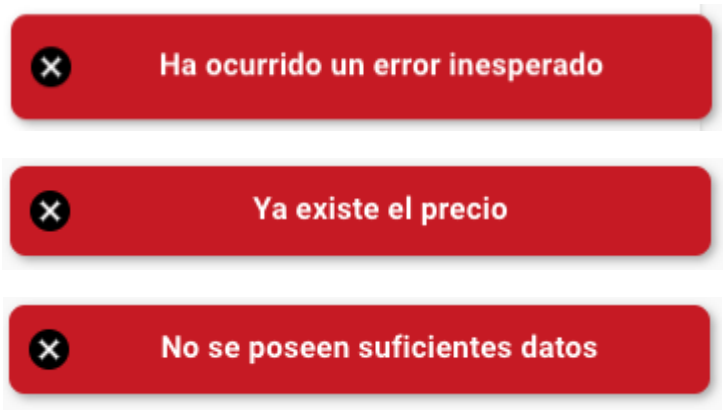
Ante acciones del usuario el sistema lanza diálogos que notifican del resultado de dichas acciones al usuario.

Algunos ejemplos de estos mensajes son mostrados a continuación.

8.8.1 - Mensajes de éxito



8.8.2 - Mensaje de error


Ha ocurrido un error inesperado
Ya existe el precio
No se poseen suficientes datos

9 - Conclusiones

Si nos retrotraemos a los inicios del proyecto, nos vimos ante un desafío de gran dimensión: deberíamos concebir un sistema desde el análisis y desarrollo hasta la puesta en producción del mismo, y nos aterraba la idea de que sería realmente utilizado a diario y no solamente un proyecto más de la carrera que sería cajoneado luego de aprobar la materia, por lo que debíamos lograr algo realmente usable y “sin fallos”. Sumado a esto, el proyecto no sería un simple desarrollo para un cliente, sino que marca la finalización de una etapa y el tan esperado paso de dejar de ser estudiantes para convertirnos en ingenieros.

A lo largo del proyecto, pusimos en práctica conocimientos aprendidos en más de una de las materias dictadas en la carrera. Más allá de los cursos que nos proveyeron de las herramientas de programación que utilizamos en el proyecto, nos encontramos ante la necesidad de comprender y analizar procesos de negocio para dar soluciones a problemas, requerimos de ingeniería del software para llevar la gestión del desarrollo, y sobre todo, necesitamos conocimientos sobre gestión de proyectos para que el mismo no resulte desastroso.

Desde el punto de vista del desarrollo personal, este proyecto, al ser autogestionado, nos representó un valioso aprendizaje en cuanto a nuestras *habilidades blandas*, principalmente a partir de la comunicación con los interesados, y los procesos de negociación, que se produjeron en cuanto al alcance del producto y de las diferentes etapas del mismo.

En cuanto a nuestro desarrollo profesional, el trabajo realizado representa el mayor desafío que debimos enfrentar hasta el momento. Por primera vez debimos concebir un sistema desde el análisis hasta la puesta en producción del mismo. Para lograr esto debimos hacer uso de tecnologías que desconocíamos hasta el momento, o no habíamos utilizado a tal grado. Además de eso, tuvimos que tener en cuenta tareas que anteriormente no nos eran relevantes, como ser el diseño de experiencia de usuario o la migración de datos.

En resumidas palabras, podemos concluir que en este proyecto se vio aplicado todo el aprendizaje adquirido a lo largo de nuestro camino por la facultad y fue una experiencia sumamente gratificante y enriquecedora el poder poner en práctica todos estos conocimientos. Creemos que en este proyecto se vio reflejado nuestro paso por la facultad y ha sido una excelente forma de darle cierre a esta etapa.

10 - Perspectivas futuras

Como se plantea al inicio del informe, el alcance de este proyecto solo abarca la versión mínima del sistema, que es tan simple como puede ser pero a la vez, resuelve la problemática inicial de los interesados.

Dentro de este proyecto, quedaron funcionalidades sin implementar, correspondientes a las historias 5, 6 y 12, y además, al día de la escritura de este informe siguen surgiendo nuevos requerimientos del sistema por parte de los interesados.

Se espera también que a partir del uso del sistema vayan surgiendo nuevas inquietudes de funcionalidades para agregar al mismo, ya que se pretende brindar una solución integral al soporte de la producción de la empresa.

Una de las inquietudes fundamentales hoy día es, no solo la carga de insumos de producción y permitir llevar el stock de esos insumos, sino que también, a partir de los rendimientos y las recetas de cada uno de los productos, permitir un cálculo de los costos de los mismos.

Otra de las inquietudes es la de asociar valores promedio a los productos, para mejorar la experiencia de usuario, mediante sugerencia de valores en los campos y la generación de alertas. Por ejemplo, al cargar producción, se sugiere la cantidad de cajas que se requiere para X cantidad de hormas, o al cargar una expedición, se alerta si el peso promedio por horma de la expedición tiene una desviación de X% con respecto al peso promedio por horma del lote.

Bibliografía y referencias

PROJECT MANAGEMENT INSTITUTE. *Guía de los Fundamentos para la Dirección de Proyectos (Guía del PMBOK V ®)*. Quinta edición.

WAKE, William C. *Extreme Programming Explored.*

Java: https://docs.oracle.com/en/java/.

JUnit: https://junit.org/junit5/docs/current/user-guide/.

Maven: https://maven.apache.org/guides/.

PostgreSQL: https://www.postgresql.org/docs/.

Hibernate: https://hibernate.org/orm/documentation/6.1/.

Spring Framework: https://docs.spring.io/spring-framework/docs/current/reference/html/.

Spring Boot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

Spring Data: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

Spring Security: https://docs.spring.io/spring-security/reference/index.html.

JWT: https://jwt.io/introduction.

Docker: https://docs.docker.com/.

Javascript: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

React: https://reactjs.org/docs/getting-started.html.

Axios: https://axios-http.com/docs/intro.

Material UI: https://mui.com/material-ui/getting-started/installation/.

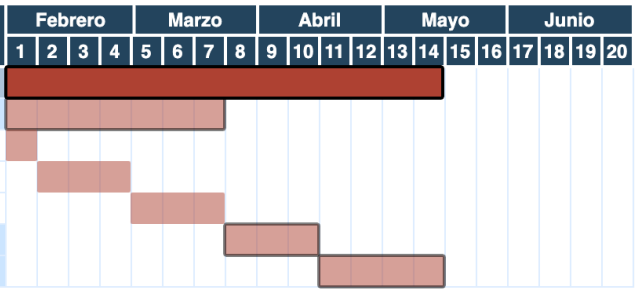
Insomnia: https://docs.insomnia.rest/insomnia/get-started.

Postman: https://learning.postman.com/docs/getting-started/introduction/.

Anexo A - Cronograma

Cronograma planificado

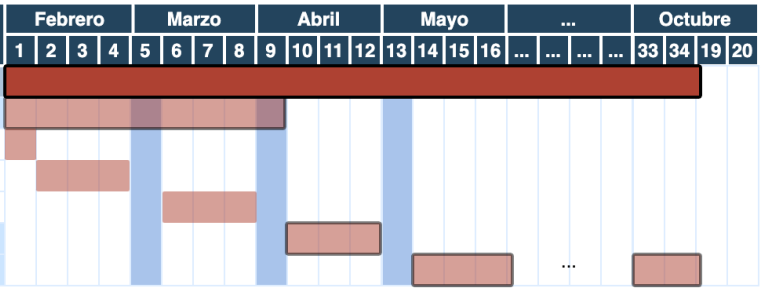
	Tarea	Duración
1	Proyecto	70 dias
2	Desarrollo	7 semanas
3	Zero Feature Release	1 semana
4	Iteración 1	3 semanas
5	Iteración 2	3 semanas
6	Despliegue	3 semanas
7	Redacción de informe	4 semanas



Cronograma real

La redacción del informe se extendió desde mediados de mayo hasta mediados de octubre debido a correcciones y revisiones.

	Tarea	Duración
1	Proyecto	170 dias
2	Desarrollo	7 semanas
3	Zero Feature Release	1 semana
4	Iteración 1	3 semanas
5	Iteración 2	3 semanas
6	Despliegue	3 semanas
9	Redacción de informe	21 semanas



Anexo B - Historias de usuario

<table>
<tr><td>Número: 1</td><td colspan="2">Nombre: Cargar Producción</td></tr>
<tr><td colspan="2">Prioridad de negocio: Alta</td><td rowspan="2">Puntos estimados: 3</td></tr>
<tr><td colspan="2">Riesgo en desarrollo: Baja</td></tr>
<tr><td colspan="3">Descripción:
Se deberá llevar un registro de la producción de la empresa, mediante la carga de lotes de producción. El lote de producción tendrá asociados los siguientes datos:
● Identificador de lote
● Fecha de elaboración
● Tina
● Litros procesados
● Cantidad de hormas producidas
● Tipo de queso
● Saldo del lote
● Peso
● Rendimiento
Los datos que se deben ingresar son:
● Litros procesados
● Fecha de elaboración
● Tina
● Cantidad de hormas producidas
● Tipo de queso
● Peso
Además se deben ingresar los datos de los insumos que se utilizaron para la producción:
● Cultivo
● Lote de cultivo
● Lote de colorante
● Lote de calcio
● Lote de cuajo</td></tr>
<tr><td colspan="3">Observaciones:
● El identificador de lote está formado por la fecha de elaboración, el código de queso (Q) y el número de tina (T) (ddmmaaaaQQQT)
● Saldo del lote: se obtiene a partir de la cantidad de hormas producidas, menos las expedidas de este lote.
● Rendimiento: se calcula con el peso obtenido sobre los litros procesados * 100
● Saldo del lote: debemos dejar que el lote pueda ir a negativo por posibles errores de tipeo, para que luego puedan corregir la situación.
● Los datos de los insumos utilizados pueden ser ingresados posteriormente</td></tr>
</table>

Número: 2	**Nombre**: Cargar tipos de productos
Prioridad de negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Bajo	

Descripción:

Se deberá proporcionar en el sistema la habilidad de cargar, consultar, modificar y eliminar los distintos tipos de quesos que se producen.

Además, por cada queso se deberá conocer su stock actual, que se irá modificando según se generen lotes de producción y expediciones del mismo.

Observaciones:

- Con la misma consideración de la historia 1, debemos permitir que el stock de los tipos de queso tome valores negativos.

Número: 3	**Nombre**: Consultar lotes producidos
Prioridad de negocio: Media	**Puntos estimados: 1**
Riesgo en desarrollo: Baja	

Descripción:

Se deberá mostrar toda la información correspondiente a los lotes de producción que se encuentren en el sistema. Se deberá poder también editar la misma para actualizar la información de los lotes de producción.

Observaciones:

- Se deberá poder diferenciar visualmente los lotes de producción que ya han comenzado a expedirse, los que no, y los que se han agotado. También se deberán poder identificar aquellos lotes que se encuentren con saldo negativo.

Número: 4	**Nombre**: Consultar trazabilidad de un lote
Prioridad de negocio: Media	**Puntos estimados: 1**
Riesgo en desarrollo: Media	

Descripción:

Se se realiza una consulta por lote de producción y se deberán mostrar los siguientes datos:

- Insumos utilizados y sus números de lote.
- Datos cargados del lote de producción.

● Clientes a quienes se expidió el lote.
Observaciones: ● Para un mismo lote de producción se pueden utilizar más de un lote de un mismo insumo

Número: 5	**Nombre**: Cargar Insumos	
Prioridad de negocio: Baja		**Puntos estimados: 3**
Riesgo en desarrollo: Media		
Descripción: Se deberán poder realizar cargas del stock actual de insumos que se utilizan para la producción de quesos, y el mismo se decrementará por cada lote de producción según la receta utilizada en el mismo. Se debe emitir una alerta en el momento que el stock del insumo lleve a un valor bajo determinado por el usuario.		
Observaciones:		

Número: 6	**Nombre**: Cargar recetas de queso	
Prioridad de negocio: Baja		**Puntos estimados: 3**
Riesgo en desarrollo: Media		
Descripción: Para cada tipo de queso cargado en el sistema, se deberá cargar la receta del mismo, indicando ingredientes y la cantidad que se requiere por cada mil litros de leche.		
Observaciones:		

Número: 7	**Nombre**: Visualización de litros usados para elaboración	
Prioridad de negocio: Baja		**Puntos estimados: 1**
Riesgo en desarrollo: Baja		
Descripción: Se mostrarán tablas y gráficos que sinteticen la información disponible de los litros de leche utilizados para la elaboración de los quesos. Esto permitirá tener una noción actualizada de cuanto litros de leche se destinan por cada queso.		

Observaciones:

Número: 8	**Nombre**: Cargar expediciones de producto
Prioridad de negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Baja	
Descripción: El sistema debe permitir cargar las expediciones que se realizan del producto. Las expediciones se realizan para un cliente particular sobre un lote específico. Si fuera del sistema una expedición incluye varios lotes, se cargará una expedición por cada lote. Cada vez que se genere una expedición, se deberá decrementar del stock del lote de producción correspondiente, la cantidad expedida. El importe asociado se calculará a la expedición, a partir del precio del tipo de queso, el tipo de cliente y el peso del queso.	
Observaciones:	

Número: 9	**Nombre**: Gestión de clientes
Prioridad de negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Baja	
Descripción: Se deberá llevar un registro de los clientes de la empresa, con los siguientes datos: ● número de cliente ● razón social ● CUIT ● domicilio ● código postal ● localidad ● provincia ● país ● e-mail ● teléfono	

- celular
- fax
- transporte
- número de habilitación SENASA/UTA

Además, a los clientes se les asignará un tipo de cliente, que se utilizará para poder establecer distintos precios en un mismo tipo de queso. Los tipos de clientes definidos son: Mayorista, Minorista, Particular, Tercerizado.

Se debe poder agregar, actualizar, consultar y eliminar clientes.

Observaciones:

Número: 10	**Nombre**: Cargar precios
Prioridad de negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Baja	

Descripción:

Se deberá llevar un registro de precios de los quesos de acuerdo con el tipo de cliente. Además de contar un histórico de los mismos.

Estará la opción de actualizar o eliminar precios para tipos de clientes o queso.

Observaciones:

Número: 11	**Nombre**: Emitir un remito
Prioridad de negocio: Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Media	

Descripción:

Se deberá poder emitir un remito, que se forma a partir de un conjunto de expediciones a un cliente en particular. El remito deberá seguir el modelo provisto, y mostrar la siguiente información:

- Fecha
- Información del cliente:
- Número de cliente y razón social
- Transporte
- SENASA/UTA -CUIT
- Información de los productos:
    - Producto (tipo de queso)

- Cantidad (total expedida a cliente)
- Peso
- Precio unitario
- Precio total

- Importe total

Los únicos datos que se deberán ingresar aquí son la fecha y el cliente.

Las expediciones que incluye el remito son aquellas no incluidas a partir de la última emisión de remito para el cliente particular.

Observaciones:

Número: 12	**Nombre**: Cargar devolución de producto
Prioridad de negocio: Baja	**Puntos estimados: 1**
Riesgo en desarrollo: Baja	
Descripción: Se podrá cargar la devolución de los productos vendidos a los clientes. Para ello, se deberán registrar los siguientes datos respecto a la devolución de los mismos: ● Fecha de devolución ● Cliente que realizó la devolución ● Lote del queso a devolver ● Cantidad del producto a devolver ● Peso del producto ● La temperatura a la que ingresó el producto ● Motivo de la devolución	
Observaciones:	

Número: 13	**Nombre**: Visualizar ventas
Prioridad de negocio: Media	**Puntos estimados: 2**
Riesgo en desarrollo: Baja	
Descripción: Se debe poder consultar las ventas de un periodo de tiempo en particular, pudiendo visualizar el total y un desagregado según el tipo de queso. Habrá filtros que permitan una mejor desagregación.	
Observaciones:	

Número: 14	**Nombre**: Gestión de permisos de usuario
Prioridad de negocio: Baja	**Puntos estimados: 3**
Riesgo en desarrollo: Alta	
Descripción: Estos permisos restringirán o permitirán ciertas acciones sobre el sistema. Los usuarios	

deberán ingresar al sistema con un nombre de usuario y una contraseña.
Observaciones:

Estas historias se agregan luego de haber cumplido una primera iteración y haber tenido una devolución del stakeholder:

Número: 15	**Nombre**: Stock de embalaje	
Prioridad de negocio: Baja		**Puntos estimados: 3**
Riesgo en desarrollo: Alta		
Descripción: Llevar un conteo del stock de cajas y bolsas para embalaje. El stock de las bolsas de cada producto se debe decrementar según la cantidad de hormas que se ingrese al momento de cargar producción. La cantidad de cajas que se utilicen en para cada lote de producción, se ingresará al momento que se ingresa el lote. Existe un tipo de bolsas por cada queso, y distintos tipos de quesos pueden compartir un mismo tipo de caja.		
Observaciones:		

Número: 16	**Nombre**: Ver Rendimiento	
Prioridad de negocio: Baja		**Puntos estimados: 2**
Riesgo en desarrollo: Alta		
Descripción: Se debe poder visualizar en una pantalla los rendimientos de los lotes de producción. Se debe mostrar una comparación con el rendimiento entre las 3 últimas semanas.		
Observaciones:		

Número: 17	**Nombre**: Ver stock de productos	
Prioridad de negocio: Baja		**Puntos estimados: 1**
Riesgo en desarrollo: Alta		
Descripción: Se debe contar con una pantalla para poder visualizar el stock actual de los diferentes		

productos elaborados. El stock de los productos se debe incrementar cada vez que se carga un lote de producción, y se debe decrementar cada vez que surja una expedición del producto.
Observaciones:

Número: 18	**Nombre**: Cargar expediciones con lector de código de barras
Prioridad de negocio: Baja	**Puntos estimados: 2**
Riesgo en desarrollo: Alta	
Descripción: Las expediciones de producto deben poder cargarse a través de la lectura de los códigos de barra de las cajas de los productos. El código está conformado por el número de lote y el peso de la caja. Si se realiza la lectura de varias cajas de un mismo lote, se debe ir incrementando automáticamente el peso de la expedición.	
Observaciones:	

Número: 19	**Nombre**: Ver remitos generados
Prioridad de negocio: Alta	**Puntos estimados: 1**
Riesgo en desarrollo: Baja	
Descripción: Se debe agregar un apartado que permita ver los remitos que fueron generados en el sistema. La pantalla debe tener el formato de “Ver producción” y debe permitir imprimir los remitos.	
Observaciones:	

Número: 20	**Nombre**: Ver expediciones
Prioridad de negocio: Alta	**Puntos estimados: 1**
Riesgo en desarrollo: Baja	
Descripción: Se debe agregar un apartado que permita ver las expediciones cargadas en el sistema. Debe seguir el formato de “Ver Producción”, y permitir editar y borrar las mismas.	
Observaciones:	

Número: 21	**Nombre**: Expedir un lote completo	
Prioridad de negocio: Alta		**Puntos estimados: 1**
Riesgo en desarrollo: Baja		
Descripción: Se debe agregar una opción dentro de la pantalla de expediciones que permita generar una expedición con un lote completo. Esta opción puede solamente ser utilizada si el peso del lote es confiable, en caso contrario se debe mostrar un error. Se debe agregar un atributo de "peso no confiable" en el lote de producción. Este atributo se cargará dentro de la pantalla de "Ver Producción", en conjunto con el peso del lote.		
Observaciones:		