



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

DESARROLLO DE HIVE MEDIANTE LA MEJORA DE RENDIMIENTO E INTEGRACIÓN DE PROCESOS DE NEGOCIOS

RODRIGO IGNACIO HANUCH GONZÁLEZ

Trabajo de Título para optar al título de
Ingeniero Civil de Industrias, Diploma en Ingeniería de Computación

Profesor Supervisor:
JORGE BAIERA ARANDA

Santiago de Chile, Diciembre 2021

© MMXXI, RODRIGO IGNACIO HANUCH GONZÁLEZ



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

DESARROLLO DE HIVE MEDIANTE LA MEJORA DE RENDIMIENTO E INTEGRACIÓN DE PROCESOS DE NEGOCIOS

RODRIGO IGNACIO HANUCH GONZÁLEZ

Miembros del Comité:

JORGE BAIERA ARANDA

MEMBER A

MEMBER B

MEMBER C

Trabajo de Título para optar al título de

Ingeniero Civil de Industrias, Diploma en Ingeniería de Computación

Santiago de Chile, Diciembre 2021

© MMXXI, RODRIGO IGNACIO HANUCH GONZÁLEZ

*Dedicado a mis padres y hermana,
quienes me apoyaron, y toleraron,
incondicionalmente durante mis
estudios.*

AGRADECIMIENTOS

Agradecimientos especiales a Jorge Baier por guiarme en todo el proceso de titulación y sus repetidos apoyos académicos durante mi carrera. Agradezco también a mi equipo de *TechOps* y a Nicolás Kipreos por recibirme con los brazos abiertos en su equipo y dentro de Beetrack. Finalmente agradezco a mi Centro de Alumnos por su apoyo incondicional en todo momento durante la carrera.

ÍNDICE GENERAL

Agradecimientos	IV
Índice de figuras	VII
Abstract	IX
Resumen	X
Capítulo 1. Introducción	1
1.1. Contextualización de la empresa	1
1.2. Descripción del proyecto con el cual se trabajó: <i>Hive</i>	2
1.3. Descripción de los trabajos realizados	3
1.4. Competencias evidenciadas	3
Capítulo 2. Desarrollo del proyecto	5
2.1. Metodologías de trabajo	5
2.2. Tecnologías utilizadas	8
Capítulo 3. Aumento de cobertura de pruebas e integración continua	12
3.1. Objetivos y contextualización	12
3.2. Desarrollo	13
3.3. Resultados	17
Capítulo 4. Servicio de Condiciones Comerciales	19
4.1. Objetivos y contextualización	19
4.2. Desarrollo	20
4.2.1. Modelamiento	20
4.2.2. <i>Solver</i> de condiciones y arquitectura	23
4.2.3. <i>Refactoring</i> y generación de facturas automáticas	26
4.2.4. Creación de condiciones por parte de vendedores y aprobación de gerentes	27

4.3. Resultados	31
Capítulo 5. Conclusiones	34
5.1. Competencias evidenciadas	34
5.2. Conclusiones generales	37
Referencias	38

ÍNDICE DE FIGURAS

2.1.	Tablero <i>Kanban</i> utilizado en el proyecto.	6
2.2.	<i>Gitflow</i> (Atlassian, 2021).	7
2.3.	Flujo de integración continua (Velotio Technologies, 2019).	8
2.4.	Diagrama de tecnologías y arquitectura.	11
3.1.	Cobertura inicial del ERP (captura tomada el 3 de agosto de 2021).	13
3.2.	Clasificación y etiquetado de los diferentes archivos y sus estados.	14
3.3.	Cambio en flujos de Circle CI y falla de <i>RSpec</i> previa a la puesta en producción del código.	16
3.4.	Cobertura final del ERP (captura tomada el 15 de noviembre de 2021).	17
3.5.	Archivos modificados y generados durante creación de nuevas pruebas.	18
4.1.	Esquema de información de condición comercial.	22
4.2.	Modelos de cobros y condiciones comerciales.	23
4.3.	Relaciones de modelos de condiciones comerciales.	24
4.4.	Esquema de delegación de <i>solver</i> y calculadora.	25
4.5.	Relaciones de modelos de condiciones comerciales.	26
4.6.	Vista general de condiciones comerciales preaprobadas.	28
4.7.	Correo electrónico de solicitud de aprobación de asociación comercial.	29
4.8.	Vista general de condiciones comerciales preaprobadas.	29
4.9.	Vista de aprobación y rechazo de asociaciones comerciales.	30

4.10. Correo electrónico de aprobación (izquierda) y rechazo (derecha) de asociación comercial.	30
4.11. Vista general de condiciones comerciales preaprobadas.	31

ABSTRACT

This scope of the present report is to present the work carried out by Rodrigo Ignacio Hanuch González within the company Beetrack S.A. to obtain the degree of Civil Industrial Engineer with a diploma in Computer Science. For four months the student worked as a full-stack software Engineer in the company's operations team. The pupil developed two projects: the systematic increment of the coverage of the testing suite and their use in the continuous integration of the software and the creation of a specialized service to host and solve the commercial conditions of the clients.

The first project consisted in increment of the coverage of the company's ERP testing suite, along with integration of these into the continuous integration flow in Circle CI. The main objective of this project was to be able to increase the reliability of the software when making changes to the codebase so that in the event of an unexpected change to the code, the tests would fail and potentially erroneous code would be prevented from being put into production.

The second project covered the problem of not having a dedicated service and model to host flexible commercial conditions, together with a cost calculation service for the generation of invoice drafts. The main objectives of the project were the creation of a flexible model that could cover existing commercial conditions, as well as future conditions, together with an increment in the precision of the invoicing calculation for customers in order to reduce the average error during the creation of invoice drafts.

Through the two projects presented, the student was able to demonstrate the three declared competencies to obtain the degree of Engineer. The report is divided into three sections, the first in which the company and work are contextualized, a second that details the projects in which he worked and a last one that evidences how each declared competency is fulfilled, alongside general conclusions.

RESUMEN

En el presente informe se presenta el trabajo de título realizado por Rodrigo Ignacio Hanuch González dentro de la empresa Beetrack S.A. para optar al grado de Ingeniero Civil de Industrias con diploma en Ingeniería de Computación. Durante cuatro meses el alumno trabajó como Ingeniero de *Software full-stack* en el equipo de operaciones de la empresa. El estudiante desarrolló dos proyectos: el aumento sistemático de la cobertura del *testing* y uso de este en la integración continua, junto con la creación de un servicio especializado para mantener y resolver las condiciones comerciales de los clientes.

El primer proyecto consistió en el aumento de la cobertura de las pruebas automatizadas del ERP de la empresa, junto con la integración de estas al flujo de integración continua en Circle CI. El objetivo principal de este proyecto fue el poder aumentar la seguridad del equipo al hacer cambios en el sistema de manera de que ante un cambio fallido las pruebas fallaran y se evitara poner en producción código potencialmente erróneo.

El segundo proyecto abarcó el problema de no tener un servicio y modelo dedicado al almacenamiento de condiciones comerciales flexibles, junto con un servicio de cálculo de costos para la generación de pre-facturas. Los principales objetivos del proyecto fueron la creación de un modelo flexible que pudiese abarcar las condiciones comerciales existentes, como las que se fueran a crear en un futuro, junto con un aumento en la precisión del cálculo de facturación para los clientes de manera de disminuir el error promedio durante la creación de las pre-facturas.

A través de los dos proyectos el alumno logró demostrar el cumplimiento de las tres competencias declaradas para obtener el grado de Ingeniero. El informe se divide en tres secciones, una primera donde se contextualiza la empresa y el trabajo, una segunda que detalla los proyectos en los cuales el estudiante trabajó y una última en donde se exponen las evidencias declaradas junto con conclusiones generales.

CAPÍTULO 1. INTRODUCCIÓN

1.1. Contextualización de la empresa

Beetrack S.A. (de ahora en adelante, “Beetrack” o “la empresa”) es una empresa que brinda un SaaS (*Software as a Service*) dentro del rubro logístico, con un enfoque en la resolución del problema de última milla. Esta fue fundada el año 2013 por Sebastián Ojeda y Nicolás Kipreos como un emprendimiento con el fin de apoyar al creciente comercio electrónico en el proceso de seguimiento y despacho. El día de hoy, 8 años después, Beetrack cuenta con más de 650 clientes activos y más de 100 empleados (Corporateit, 2021) en Chile, Argentina, Perú, Colombia y México. Al mismo tiempo, esta se ha posicionado a si misma como una de los líderes del mercado SaaS en América Latina por el éxito que ha tenido su principal producto, LastMile.

Junto con LastMile, la empresa también ofrece un SaaS de planificación, como lo es PlannerPro. Por un lado, LastMile permite tener un seguimiento en tiempo real de las entregas que se realizan, disminuyendo la incertidumbre de los clientes sobre el estado de su pedido. Por otro lado, PlannerPro es utilizado para planificar, optimizar y diseñar rutas de entregas y así reducir los costos operacionales de reparto de la empresa que lo utilice.

Actualmente la empresa es la más grande del rubro de SaaS de última milla en latinoamérica, siendo su competencia directa SimpliRoute y Drivin. Beetrack busca posicionarse como la mejor alternativa del mercado constantemente mediante la inclusión de nuevos productos y el servicio de calidad que entrega.

Desde el punto de vista organizacional, la empresa se divide en múltiples áreas. Las anteriores son: administración, *customer success*, *payments*, desarrollo, recursos humanos, *design ops*, *marketing*, operaciones, producto, *revenue operations*, ventas y *data science*. Cada una de estas áreas trabaja en forma independiente con algunos cruces entre ellas en los que se tiene interacciones entre distintos equipos. En particular, el trabajo realizado fue

Múltiples partes del presente trabajo se basan fuertemente en (Burotto, 2021), especialmente las secciones 1 y 2.

en el área de operaciones con interacciones con los equipos de ventas y *design ops*, entre otros.

1.2. Descripción del proyecto con el cual se trabajó: *Hive*

La empresa decidió crear su propio ERP (*Enterprise Resource Planning*) interno de manera de poder adaptar su *software* a sus procesos y no sus procesos al *software*. El nombre de este ERP es *Hive*, el cual cuenta con diferentes partes y módulos, cada uno con el acceso restringido a los empleados correspondientes por área del módulo. Actualmente, el módulo más utilizado es el de *Payments*, el cual corresponde al del área contable de la empresa que permite generar facturas y aprobar pre-facturas para los diferentes clientes de Beetrack.

Los objetivos del ERP son dos. En primer lugar, el llevar tecnología a todas las áreas de la empresa de manera de poder mejorar y facilitar los procesos internos. Dentro de esta misma idea, el segundo objetivo es poder lograr la mayor automatización posible en ellos de manera de que se cometan menos errores y que las diferentes áreas de la empresa puedan enfocarse realmente en lo que les compete por sobre la realización de procesos estandarizados.

Cabe mencionar que a pesar de que *Hive* tiene un enfoque interno, este *software* posee conexiones con, principalmente, 3 sistemas externos a este. En primer lugar, posee conexión con los productos de la empresa, esto es LastMile y PlannerPro. En segundo lugar el ERP está conectado con un Hubspot, el CRM (*Customer Relationship Manager*) que Beetrack utiliza. En esta plataforma se encuentra toda la información relacionada a todos los clientes, como el estado de estos dentro de la empresa. Finalmente, *Hive* también se encuentra conectado fuertemente con Quickbooks, un SaaS utilizado para llevar la contabilidad de la empresa, permitiendo generar facturas, como también conciliar pagos de clientes. En este sentido, *Hive*, al estar conectado con estos sistemas externos, facilita

la comunicación y automatización de procesos internos, tanto contables como de administración, de manera centralizada al agregar lógica de negocio a los diferentes procesos involucrados.

1.3. Descripción de los trabajos realizados

El alumno ejerció como Ingeniero de *Software Full Stack*, es decir, que estuvo a cargo tanto de las decisiones de diseño de las soluciones tecnológicas, del desarrollo del *backend*, que corresponde a la lógica de las aplicaciones, y del *frontend*, que corresponde a la interfaz que el usuario final utiliza. Las principales tareas desarrolladas corresponden al análisis y modelamiento de esquemas de negocio y el análisis de espacios que pueden llevar a mejoras en el *software*, junto con el desarrollo de las soluciones a los problemas encontrados.

Por un lado, la empresa propuso al alumno abordar el desafío de la reestructuración de la forma en que se estaba pensando y utilizando las condiciones comerciales de facturación de cada cliente en Hubspot. En particular se sugirió rehacer desde cero todo lo que se tenía, creando un *software* especializado para esta tarea. Por otro lado, el estudiante realizó su práctica profesional en la empresa y se dio cuenta, durante ese período de tiempo, que existía una falta de pruebas unitarias y de integración de estas en el ERP de la empresa. Por el motivo anterior, el universitario le propuso a la empresa aumentar significativamente el nivel de cobertura de código de su ERP, junto con la integración de estas pruebas al flujo de integración continua en el *software* Circle CI de manera de prevenir que un pedazo de código, potencialmente erróneo, se ponga en producción.

1.4. Competencias evidenciadas

Mediante el trabajo realizado dentro de Beetrack, el alumno busca dar a conocer las competencias del perfil de egreso. La primera competencia que se busca evidenciar es el “**Aplicar conocimientos avanzados de Ciencia de la Computación, Ingeniería de**

Software y Sistemas de Información para entender problemas complejos y abiertos”.

Esta se vio ampliamente evidenciada mediante la implementación de las pruebas de integración en *Hive*, junto con el la forma en que se desarrolló el servicio de condiciones comerciales. La segunda competencia corresponde a **“Diseñar y desarrollar modelos y artefactos de software y simular soluciones a problemas de la Ciencia e Ingeniería de Computación, cumpliendo con restricciones técnicas, sociales y éticas”**, la cual se puede ver expresada por la implementación y subsecuentes pruebas realizadas con el servicio de condiciones comerciales para la facturación automática el mes de octubre junto con la simulación de ejecución de código en las pruebas unitarias creadas. En tercer, y último lugar, la competencia **“Analizar en forma sistemática las diferentes problemáticas de los usuarios, y diseñar productos o sistemas de software que queden expresados mediante algún lenguaje de programación, de acuerdo a los estándares de la ingeniería de software”** se pudo ver expuesta mediante el proceso de diseño e implementación de tanto el *frontend* en *Hive* para la creación de las condiciones comerciales con el nuevo servicio creado, como el *backend* de este servicio.

CAPÍTULO 2. DESARROLLO DEL PROYECTO

2.1. Metodologías de trabajo

Durante el trabajo de título, el alumno se desempeñó junto al equipo de *TechOps*, el cual incluye a *Hive* y al *Customer Portal*, proyecto que pretende facilitar la visibilidad de los productos contratados y entregar información de manera transparente a los clientes de Beetrack. Este equipo, conformado por Tomás Burotto, Juan José Martínez y el estudiante, utiliza herramientas y metodologías propias de desarrollos ágiles para sus proyectos, en específico, se utiliza *Scrum* con *Sprints* de dos semanas.

Scrum se define como “...un proceso en que se llevan a cabo un conjunto de tareas de forma regular con el objetivo principal de trabajar de manera colaborativa, es decir, trabajo en equipo.” (APD, 2019), siendo esta metodología la más utilizada hoy en día en el mundo de desarrollo. Un *Sprint* se caracteriza por comenzar con un *Sprint Planning*, en el cual se determinan las tareas que cada colaborador realizará en el período de tiempo establecido, asignándole a cada tarea puntos de esfuerzo. Estos últimos son una forma de medir cuánto tiempo o dificultad tiene cada tarea, de manera de poder mantener su nivel constante a lo largo de los diferentes ciclos e ir afinando el nivel de carga que cada integrante del equipo puede abarcar.

Tanto las tareas como los puntos de esfuerzos asignados a estas se encuentran ordenadas en un tablero *Kanban*, el cual permite organizar las labores. En particular, en la empresa se utiliza Jira para llevar el seguimiento de las tareas, ciclos de desarrollo, documentación y especificación de las labores. Cada integrante del equipo junta los requerimientos que llegan a lo largo del ciclo y los escribe en el tablero para ser posteriormente asignados en el comienzo del próximo. En la figura 2.1 se puede ver un ejemplo de cómo se ve el tablero en Jira.

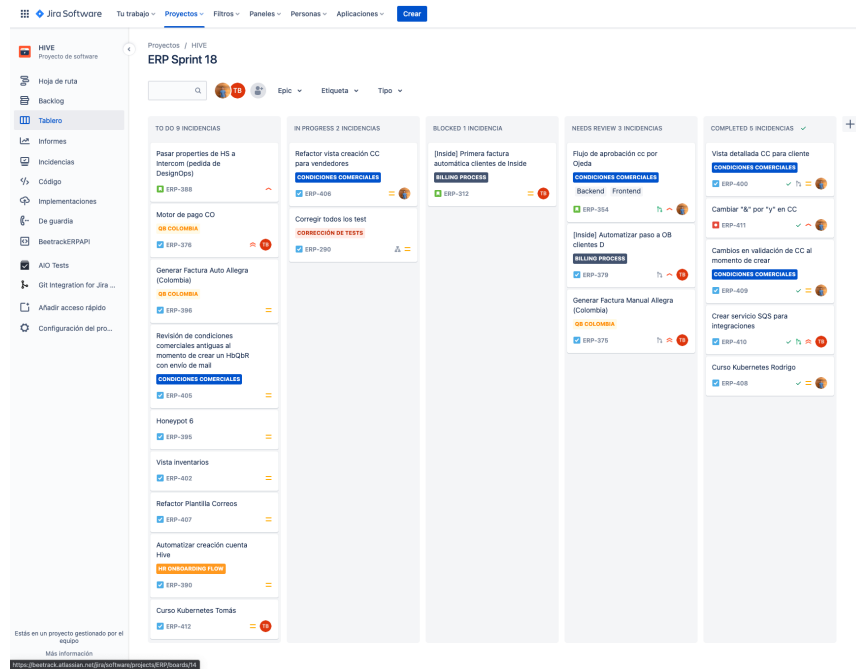


Figura 2.1. Tablero *Kanban* utilizado en el proyecto.

Debido a la cantidad de cambios que se deben realizar constantemente en el código, se requiere un versionamiento de este, por lo que se utiliza GitHub. Esta herramienta permite el uso de Git con *Gitflow*. La principal ventaja de esta herramienta es que permite mantener diferentes versiones, como también utilizar un repositorio en donde se almacena el código en diferentes estados mediante *branches*. Este uso de *branches* o “ramas” permite mantener diferentes ambientes aislados uno del otro. El proyecto de *Hive* cuenta con 3 ramas fijas, que nunca son eliminadas, *master*, *staging* y *develop*. La primera rama es aquella que se encuentra en producción, es decir, es la que es visible para los usuarios finales, la segunda es un ambiente de pruebas (principalmente de estabilidad y de *quality assurance*), y finalmente, la última es la rama de desarrollo, que es aquella en la cual se encuentra el código no probado por personas y que no se encuentra en ningún servidor de manera activa.

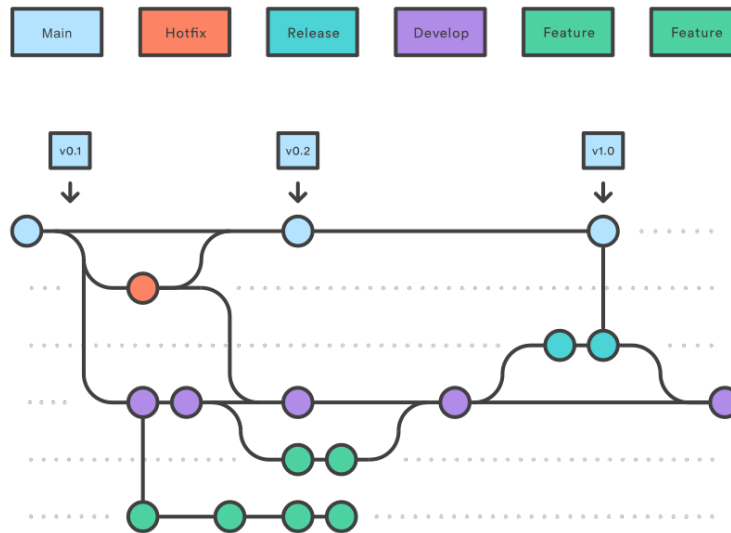


Figura 2.2. *Gitflow* (Atlassian, 2021).

Cuando se quiere desarrollar una nueva funcionalidad, hacer correcciones o realizar cualquier tipo de cambio de código, la práctica es crear una nueva rama, hacer los cambios respectivos en el código, hacer una solicitud de cambio y revisión cruzada entre los miembros que tienen acceso al mismo, para finalmente integrar los cambios a la rama principal. En la figura 2.2 se puede ver cómo se vería un diagrama de un flujo en Git.

Finalmente, dentro de la empresa se utilizan procesos de integración continua, los cuales consisten en escritura de código que permite la puesta en producción del que se subió a GitHub de manera automatizada en servidores remotos. En particular, se utiliza el SaaS Circle CI como intermediario. El flujo de integración continua, normalmente, comienza al subir el código a la rama *master* o *staging*, lo que gatilla una acción en los servidores de Circle CI, que descarga los archivos recién subidos, corre pruebas sobre estos, verifica la integridad, y finalmente los compila para subirlos a los servidores remotos especificados. La importancia de estas automatizaciones dentro de las metodologías de trabajo es que en el minuto en que estas automatizaciones fallan se despliegan alertas que permiten evitar que los servidores en producción se caigan por errores no previstos, sumado al hecho de que se permite ahorrar tiempo con tareas sistemáticas y bien establecidas. En la figura 2.3 se puede ver el proceso descrito anteriormente.

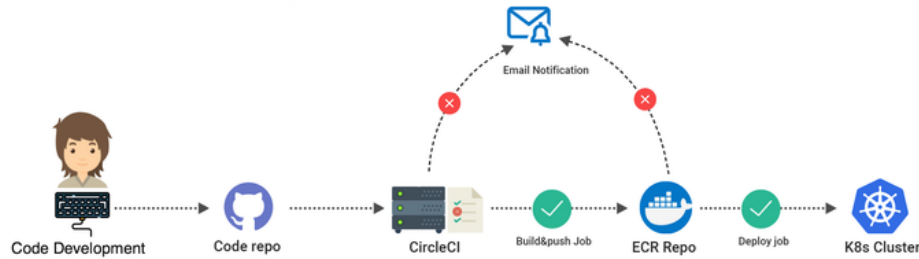


Figura 2.3. Flujo de integración continua (Velotio Technologies, 2019).

2.2. Tecnologías utilizadas

El ERP de la empresa se basa en una arquitectura que separa de manera lógica los componentes. *Hive* mismo tiene un desacoplamiento entre *frontend* y *backend*; el primero corresponde a todo lo que concierne a la forma de visualizar e interactuar con la aplicación, mientras que el segundo se encarga de todas las operaciones y lógica de negocio de la aplicación, junto con sus interacciones. Estas dos partes se comunican mediante varios *endpoints* a través de una API (*Application Programming Interface*) que el *backend* expone.

Por un lado, una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones (Red Hat, 2021). Por otro lado, un *endpoint* es una URL (*Uniform Resource Locator*), o punto de acceso, mediante el cual una aplicación, o API, puede utilizar para obtener recursos y comunicarse con otra aplicación. En el caso del ERP, los *endpoints* cuentan con una capa de seguridad para evitar accesos indeseados de aplicaciones no verificadas, teniendo que identificarse con un JWT (*JSON Web Token*), el cual funciona como llave de acceso para la API y el *backend*. En particular, se utiliza la librería *Devise* para generar los JWT que son utilizados por la aplicación *frontend* y son únicos por sesión de usuario.

El *backend* de la aplicación está desarrollado en el lenguaje Ruby con el *framework* Rails, por lo que se denomina *Ruby on Rails* (o RoR). Este *framework* permite construir

aplicaciones que sigan los patrones de diseño de ingeniería de *software* de manera muy simple y que puedan escalar rápidamente. El patrón más utilizado por este *framework* es el MVC (Modelo Vista Controlador), el cual divide a la aplicación en las tres partes mencionadas.

En primer lugar, el modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar. En segundo lugar, las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Finalmente, el controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él (Bascón Pantoja, 2004).

En particular, para almacenar los datos de largo plazo de los modelos se utiliza una base de datos con un motor PostgreSQL (o PSQL). Cabe mencionar que no todos los datos se almacenan en este tipo de memoria y que no todas las operaciones se pueden realizar con esta. Es por lo anterior que también se usa Sidekiq, una librería que permite la ejecución de tareas de manera asíncrona y/o en una fecha determinada mediante el almacenamiento de los datos necesarios en memoria *caché*, para lo que se utiliza el motor Redis.

Dada la naturaleza del ERP, como se mencionó anteriormente, este interactúa con otros servicios de dos maneras: por un lado, para el servicio de condiciones comerciales, proyecto que el alumno realizó, se utiliza GraphQL, para el resto de los servicios, tales como Hubspot y Quickbooks, entre otros, se utilizan APIs REST. Una API REST es una API que se ajusta a los principios de diseño de REST, un estilo de arquitectura también denominado transferencia de estado representacional (IBM Cloud Education, 2021). Este estilo es uno de los más populares para hacer APIs, pero es menos flexible al compararlo con GraphQL. Este último, es un lenguaje de consultas y servicio para APIs que prioriza el entregar a los consumidores exactamente la información solicitada y no más que eso. El principio de este lenguaje es hacer las APIs más flexibles, rápidas y posicionarse como alternativa a REST (Red Hat, 2019).

El *frontend* de *Hive* está desarrollado en JavaScript con el *framework* React. El código está construido sobre el principio de componentes reusables en las diferentes partes de la aplicación de manera de seguir el principio DRY (*Don't Repeat Yourself*). Dado que el *frontend* es aquel con el cual los usuarios interactúan, este requiere estilo, el cual está dado por el *desing system*, *Honeypot*, de Beetrack. Este sistema de diseño es utilizado en todas las aplicaciones de la empresa de manera de que se pueda mantener una consistencia visual y de usabilidad entre las diferentes partes que componen los *softwares* que se producen dentro de la empresa. En particular, este *design system* abarca elementos como paleta de colores, diseños visuales de botones, formularios y componentes, entre otros, mientras que al mismo tiempo se tratan como paquetes prehechos, lo que permite reutilizarlos.

En último lugar, tanto la aplicación de *Hive* como el servicio de condiciones comerciales están alojadas en GKE (Google Kubernetes Engine). Este servicio permite mantener las aplicaciones mediante imágenes compiladas de Docker en servidores que son fácilmente escalables tanto vertical como horizontalmente mediante *pods*, y al mismo tiempo se encarga de los procedimientos para generar estos servidores y que se coordinen entre ellos. Los *pods* son los objetos más pequeños y básicos que se pueden implementar en Kubernetes y representan una instancia única de un proceso en ejecución (Google Cloud, 2021). Otra ventaja que provee GKE es la facilidad para poder volver a obtener imágenes que ya no se encuentran en uso en el caso de que ocurra algún error, lo que permite aumentar la disponibilidad del servicio entregado. En la figura 2.4 se puede ver la arquitectura con la cual el alumno trabajó.

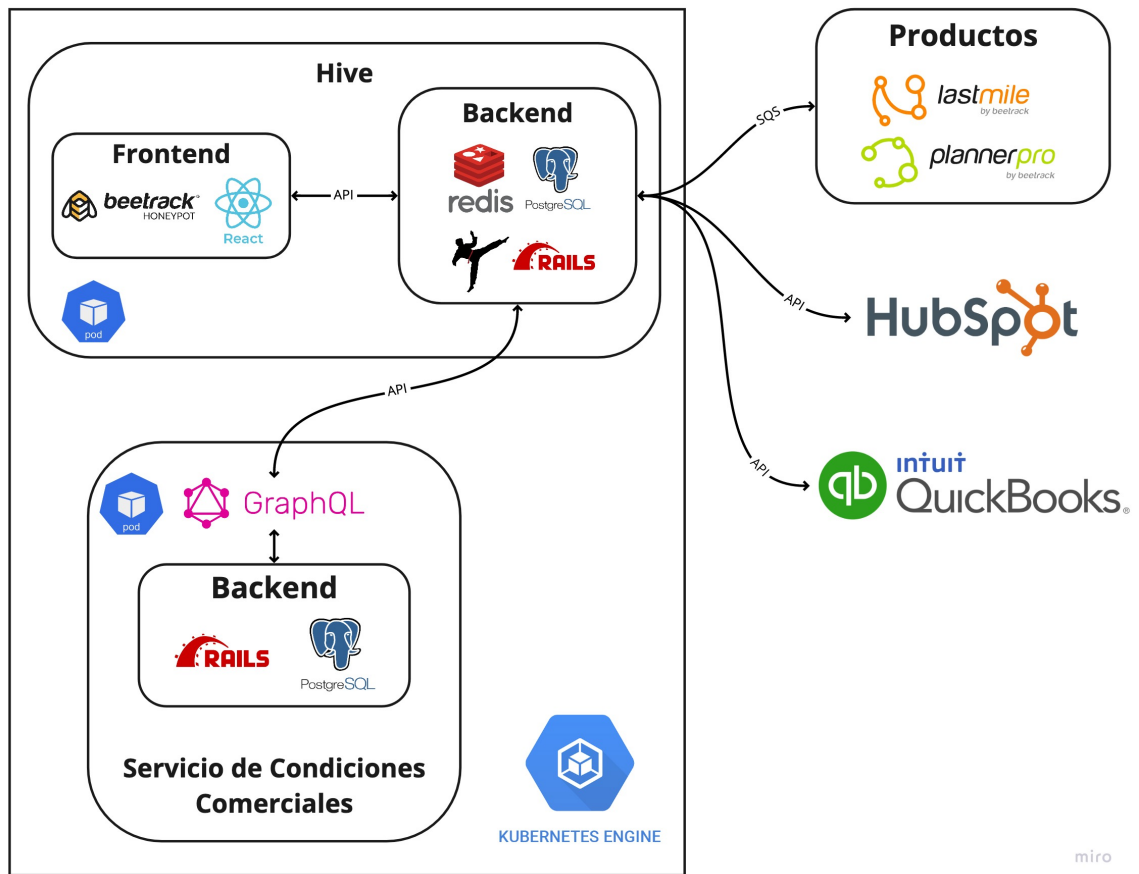


Figura 2.4. Diagrama de tecnologías y arquitectura.

CAPÍTULO 3. AUMENTO DE COBERTURA DE PRUEBAS E INTEGRACIÓN

CONTINUA

3.1. Objetivos y contextualización

Una práctica muy utilizada dentro del desarrollo de proyectos grandes de *software* es el uso de pruebas unitarias y de integración de los diferentes componentes de la aplicación sobre la que se trabaja. Esta práctica tiene dos objetivos principales: primero el poder tener seguridad de que el código escrito efectivamente cumple el propósito para el cuál fue programado, y en segundo lugar, el tener la seguridad de que al cambiar una parte del código, este siga cumpliendo su función original. En la práctica, el segundo punto es sumamente útil al momento de realizar cambios al *codebase* existente, dado que se quiere seguir teniendo la misma funcionalidad, pero cambiando el código interno, en lo que se conoce como *refactor* y mediante las pruebas se puede tener la seguridad de que todo siga en orden después de haber realizado los cambios.

En este sentido, es paradójico que, una empresa que se dedica a producir *software*, no haga pruebas rigurosas de sus desarrollos internos. Es por lo anterior, que el alumno le sugirió a su superior, Nicolás Kipreos, aumentar significativamente la cantidad de pruebas que existen para el ERP, junto con la integración de estas a un flujo de integración continua de manera de que se pudiese tener una mayor seguridad y cofianza al momento de poner en producción el nuevo código.

El principal beneficio que el área de *TechOps* tendría mediante la implementación de una mayor cantidad de pruebas y la integración de estas pruebas a un flujo de integración continua serían tres. En primer lugar, el tener la seguridad de que el código efectivamente hace lo que se pretende que haga, en segundo lugar el saber que el código no se caerá de manera inesperada frente a un cambio y, en tercer lugar, el forzar a que el código sea probado antes de ser puesto en producción (IBM, 2021), evitando que los errores lleguen a los usuarios de la plataforma.

La meta que se estableció por parte del alumno fue de llegar a una cobertura mínima de 60 % sin pruebas fallidas, junto con el forzar a que el flujo de *deployment* de la aplicación fallara en caso que alguna prueba no funcionara en Circle CI.

3.2. Desarrollo

En RoR, se hace uso de la herramienta RSpec para hacer el *testing* de manera automatizada y poder correr las pruebas correspondientes. En el caso de *Hive*, cuando el alumno llegó al proyecto, este tenía una cobertura de 35,44 %. Esto quiere decir que solamente una de cada tres líneas de todo el código existente en *Hive* era ejecutada y probada. Cabe mencionar también que aproximadamente un tercio de las pruebas que existían fallaban, es decir, el código no cumplía con las pruebas que se realizaban sobre el, teniendo un comportamiento diferente al esperado. En la figura 3.1 se puede observar el estado inicial de la cobertura de las pruebas del ERP.

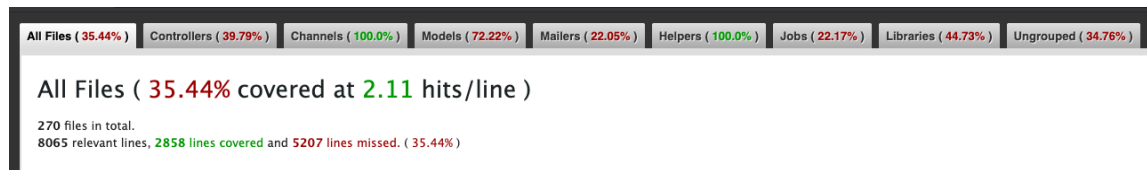


Figura 3.1. Cobertura inicial del ERP (captura tomada el 3 de agosto de 2021).

Dado que *Hive*, en agosto, tenía aproximadamente 8000 líneas de código, el proceso del aumento de la cobertura del *testing* comenzó mediante la revisión y mapeo de todos los archivos existentes y su estado de cobertura de pruebas. En otras palabras, se revisó todo el código fuente del ERP para saber cuáles archivos eran cubiertos por las pruebas automatizadas, cuáles no y cuáles requerían correcciones. De esta manera se podía ver con claridad en qué partes se tenía que enfocar los esfuerzos de trabajo y se podía saber cuáles archivos era más críticos de revisar y probar.

Workers Testing



Creación: Tomás Burotto
Última actualización: ago. 02, 2021 por Rodrigo Hanuch · 2 personas la han visto

Para testear Workers, Sidekiq tiene una guía interesante con información

<https://github.com/mperham/sidekiq/wiki/Testing>

- Automatic Account Creation

Nombre	Descripción	Estado
account_deactivation_worker.rb		COMPLETE / PASSING
base_worker.rb		COMPLETE / PASSING
deal_creation_worker.rb		COMPLETE / PASSING
free_pilot_creation_worker.rb		COMPLETE / PASSING
paid_pilot_creation_worker.rb		COMPLETE / PASSING
properties_loader_worker.rb		COMPLETE / PASSING
upgrade_pilot_worker.rb		COMPLETE / PASSING

- Automatic Invoicing

Nombre	Descripción	Estado
automatic_draft_generator_worker.rb		COMPLETE / PASSING
automatic_invoicing_worker.rb		COMPLETE / PASSING

- AWS

Nombre	Descripción	Estado
upload_file_s3_worker.rb		MISSING

- Beetrack
1. Accounts

Figura 3.2. Clasificación y etiquetado de los diferentes archivos y sus estados.

El alumno realizó el mapeo de los archivos junto a Tomás Burotto y dejaron los registros en una sección especial de Jira. Esta documentación se organizó por secciones y subsecciones de archivos y carpetas, siguiendo la misma estructura de *Hive*. Una vez que se transpararon todos los archivos, se procedió a ponerles etiquetas del estado en que se encontraban, tal como se puede observar en la figura 3.2.

Se decidió tener 3 estados para la cobertura: “*complete*”, en el cual 100 % de las líneas correspondientes del archivo eran cubiertas, “*partial*”, para indicar que no todas las líneas

del archivo eran cubiertas y “*none*”, para indicar que el archivo de pruebas existía, pero ninguna línea era cubierta.

Por otro lado, también existían 3 etiquetas para el estado en que se encontraban las pruebas: “*passing*” que aplicaba en caso de que todos los *test* estuviesen pasando, “*failing*” para indicar que una o más pruebas fallaban y “*skipped*”, que indicaba que una o más de las pruebas eran saltadas. Finalmente, se hizo una etiqueta adicional llamada “*missing*” que se aplicó en el caso de que no existiese el archivo de pruebas correspondiente.

Una vez que el alumno tuvo visibilidad del estado del *testing* de la aplicación, este procedió a programar las pruebas automatizadas del código existente. Esto implicó el crear los archivos faltantes para el código que no era probado y leer el código existente, de manera de poder realizar pruebas adecuadas y no simplemente hacer pruebas que pasaran con cualquier resultado. La principal ventaja de realizar pruebas muy detalladas es que permite tener un desglose muy granular de qué falla en caso de que algo no funcione dentro del sistema, de manera de que se pueda corregir rápidamente y sin mayor conflicto. En particular, el alumno se enfocó en realizar pruebas automatizadas para la sección de *workers*. El principal motivo del enfoque en estos archivos es debido a que los *workers*, son procesos que corren de manera asíncrona del código principal del ERP y que son automatizaciones sensibles, como, por ejemplo, la generación de facturas y pre-facturas de manera automática, notificación a usuarios internos de la empresa, y cambios de estados en los contratos y datos de Hubspot, los cuales son vistos por los vendedores.

Luego de que el estudiante cumpliera la meta establecida, este procedió a la siguiente tarea, la cual consistió en el cambio del *deployment code* de Circle CI. El flujo de puesta en producción de Circle CI funciona mediante la declaración de pasos a seguir al momento de hacer un *code deploy*. Previamente, la implementación de este código solamente contenía la compilación de los archivos y subida de estos a GKE. Para poder realizar la automatización de las pruebas en este entorno, se tuvo que levantar una imagen virtual de PostgreSQL en el contenedor de Circle CI y hacer que no finalizara su conexión luego de que se construyera, esto dado que para correr RSpec (el comando utilizado para correr las

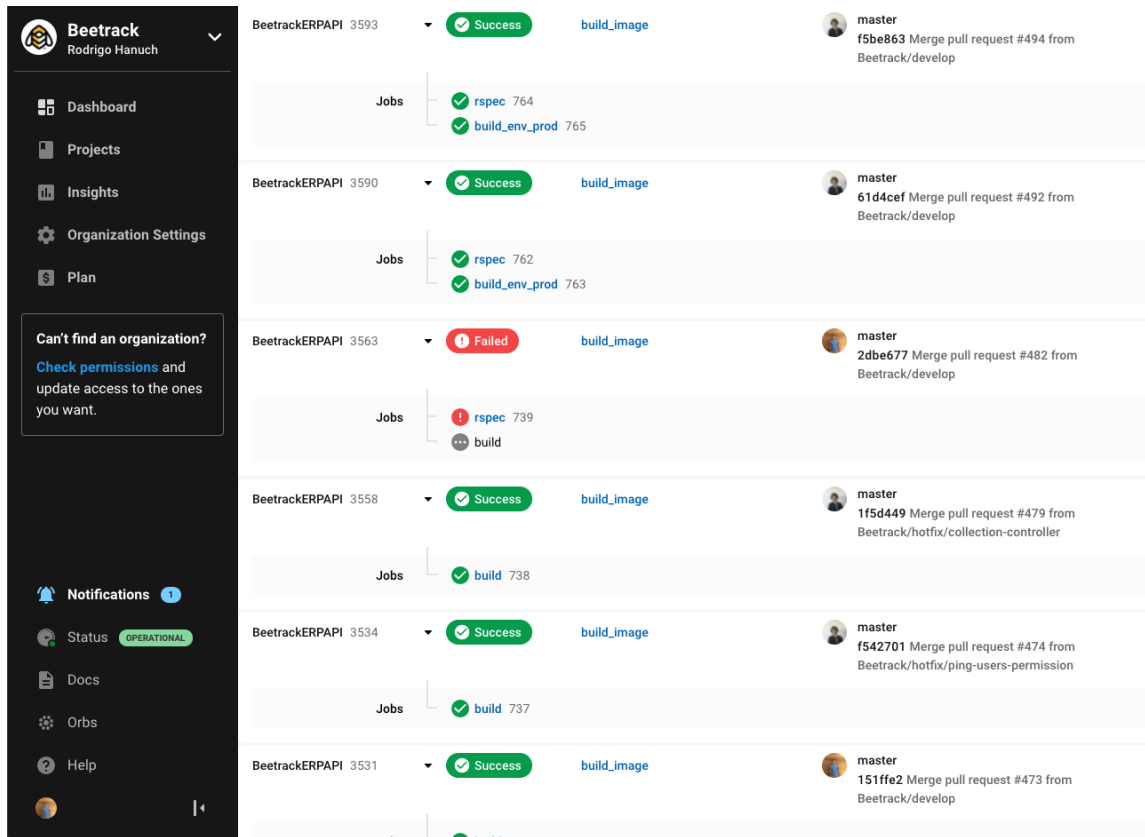


Figura 3.3. Cambio en flujos de Circle CI y falla de *RSpec* previa a la puesta en producción del código.

pruebas), se tiene que tener entradas en las bases de datos y para esto se tiene que tener una conexión abierta.

El código que el alumno escribió para el flujo de Circle CI fue hecho de tal manera de que tuviese 2 pasos: en primer lugar las pruebas automatizadas, y luego, en caso de que la primera etapa fuese exitosa, el segundo paso, el *deploy* y compilación del código para ser puesto en producción. Lo anterior permite que se pueda cortar el flujo en caso de que el primer paso falle, liberando antes los servidores que se utilizan de manera común en Beetrack para hacer *deployment* tanto de los productos principales, como del *software* interno de la empresa. En la figura 3.3 se puede observar tanto el cambio de un flujo único sin pruebas automatizadas a uno con pruebas y el hecho de que cuando falla *RSpec*, el flujo se corta de inmediato, evitando el poner código defectuoso en producción.

3.3. Resultados

Los principales resultados obtenidos de este proyecto fueron superiores a los propuestos originalmente. En particular se logró lo siguiente:

1. Cobertura de final de 64,04 % en *Hive*.

Se logró una cobertura de un 64,04 % del ERP, tal como se puede ver en la figura 3.4. Esto fue posible mediante la **simulación** de la ejecución de código previo a su puesta en producción y la aplicación de **conocimientos avanzados** de *testing* para el *framework* de RoR. Sumado a esto, también se tuvo extremo cuidado al momento de desarrollar las pruebas, evitando el escribir contraseñas o información sensible en estos mediante el uso de las herramientas proveídas por Rails, **cumpliendo con restricciones técnicas y éticas** para evitar que estas sean filtradas de manera pública a internet.

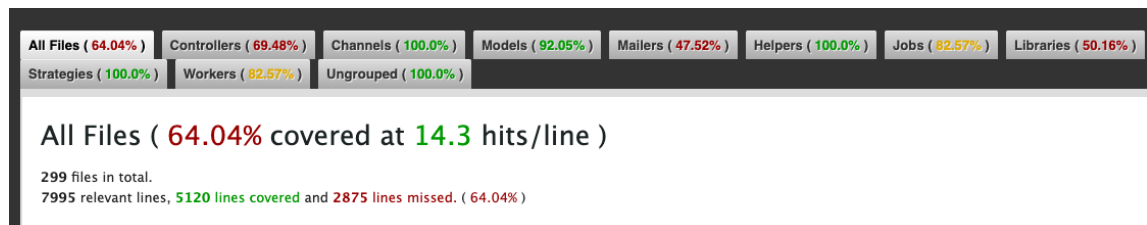


Figura 3.4. Cobertura final del ERP (captura tomada el 15 de noviembre de 2021).

2. Escritura de más de 2800 líneas de pruebas automatizadas.

El incremento de la cobertura se logró mediante la escritura o modificación de 115 archivos con un total de 2816 líneas de código nuevas dedicadas exclusivamente a la automatización del proceso de pruebas, lo cual se puede ver en la figura 3.5.

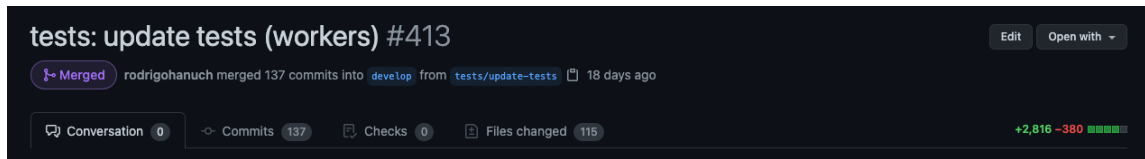


Figura 3.5. Archivos modificados y generados durante creación de nuevas pruebas.

3. Implementación de flujo de integración continua con pruebas automatizadas.

La implementación del flujo de pruebas automatizadas en Circle CI tiene como principal resultado el incremento en confiabilidad del código escrito, y, por lo tanto, la disminución del *downtime* generado por líneas no probadas para el equipo *TechOps* y para la aplicación de *Hive*. Sumado a lo anterior, esta implementación también consideró un flujo con menor cantidad de código al reescribir, en gran parte, los comandos que generaban los *deployments* a GKE, reutilizando código y haciéndolo más legible al aplicar **patrones de diseño de software** y eliminando muchos *code smells* (antipatrones de diseño, que ocurren cuando el código no sigue estándares y que pueden indicar problemas más profundos (Fowler, 2006)).

CAPÍTULO 4. SERVICIO DE CONDICIONES COMERCIALES

4.1. Objetivos y contextualización

Actualmente, la empresa tiene más de 700 clientes activos, cada uno con diferentes contratos, o condiciones comerciales, y se utiliza un archivo de Excel para mantener estas condiciones actualizadas. Al mismo tiempo, este Excel también contiene la forma en que se calculan los costos totales por el uso del servicio para cada cliente. Similar a lo que ocurría con la falta de pruebas automatizadas, también resulta curioso el hecho de que Beetrack, siendo una empresa SaaS y teniendo su propio ERP, utilice esta plantilla para calcular cuánto cobrarle a los clientes. Es por lo anterior que decidió crear un servicio especializado en el almacenamiento y cálculo de estas condiciones comerciales, cuyo único propósito fuese resolver el problema de cómo calcular los costos por cliente y poder almacenar los modelos que representaran los contratos generados con estos.

Uno de los puntos más cruciales sobre este servicio y su contexto es el hecho de que ya se había intentado dos veces algo similar. El primer intento fue mediante un desarrollo interno en el ERP, que resultó no tener el rendimiento esperado en términos de precisión al momento de facturar y se concluyó que no era responsabilidad de *Hive* como tal tener esta información. El segundo intento fue la migración de estos planes a un proyecto que, en su minuto, se encontraba en fase *beta* en Hubspot, en el cual se prometía poder tener la información siempre actualizada. Finalmente terminó ocurriendo lo mismo que en el primer intento, pero con mejor precisión al momento de facturar de manera automática, entregando, en el mejor de los casos, una precisión de 43 %.

Teniendo en consideración los requisitos y los resultados anteriores, los objetivos principales del proyecto fueron dos. El primero fue construir un modelo lo suficientemente flexible como para poder abarcar los diferentes tipos de ventas que los vendedores realizan, tanto para productos como servicios. El segundo objetivo fue el programar un servicio de resolución comercial, o calculadora, que fuese, nuevamente, lo suficientemente flexible

como para poder abarcar el modelo, pero que al mismo tiempo no necesitara información contextual más allá del uso para saber cuánto cobrarle a cada cliente, mientras que al mismo tiempo entregue glosas prehechas por producto.

El equipo de operaciones decidió no acoplar este proyecto a *Hive* de manera de evitar el cruce indebido de información, poder mantener las ideas aisladas y comenzar distribuir los servicios que posee el ERP. Lo anterior sumado al hecho de que se prevee que, eventualmente, otros servicios y productos de la empresa, tales como el *Customer Portal*, consuman los recursos expuestos por este servicio de manera de que los clientes también puedan saber cuánto se les cobrará o el nivel de costo que tienen para un día determinado del mes. Como se puede ver en la figura 2.4, del capítulo 2, la arquitectura que se decidió utilizar es una en la cual el servicio de condiciones está aislada del ERP, mientras que al mismo tiempo no limita a que otro servicio pueda consumir la información del servicio nuevo, lo cual hubiese ocurrido en caso de haber escrito el código en *Hive* dada su naturaleza monolítica.

4.2. Desarrollo

4.2.1. Modelamiento

El proceso de desarrollo del nuevo servicio de condiciones comerciales comenzó con un análisis de los diferentes planes existentes y que se encontraban en uso por parte de la empresa. Luego de realizar este análisis se definieron 4 modelos diferentes a nivel de bases de datos que permitirían almacenar de manera flexible las condiciones con las cuales se les factura a los clientes.

En primer lugar se tiene el modelo del plan, en este se almacena la condición comercial como tal, junto con una descripción del uso que puede tener esta condición y el producto y moneda asociado a la condición. Dentro del campo “condition” se almacena el tipo de tramo y costos asociado que tendrá por ellos. Se decidió que se puede tener tres tipos de tramos diferentes: base, variable y por paso.

A continuación se detalla cada tipo de tramo:

1. Tramo base: va desde cero unidades hasta la cantidad de unidades determinada con un costo fijo total por esas unidades. Solamente se puede tener un tramo de este tipo por plan.
2. Tramo variable: funcionan mediante una definición de inicio a final del tramo en término de cantidad de unidades, con un precio unitario por la cantidad de unidades utilizadas. Se pueden tener entre cero e infinitos tramos variables por plan.
3. Tramo por pasos: se utiliza para poder representar los productos que se venden mediante paquetes; esto es especialmente útil para el servicio de correos en el cual se cobra mediante paquetes de 10.000 correos, siendo los primeros 10.000 gratis y los siguientes teniendo un costo de 1 UF por bolsa adicional. Este tipo de tramo es excluyente con los dos anteriores y funciona de manera recurrente.

Adicionalmente, el campo “`condition`” contiene dos variables utilizadas al momento de calcular el costo total de la factura de un cliente. Estos se detallan a continuación:

1. `general_minimum`: se utiliza para asegurar un monto mínimo de facturación del producto independiente del uso del cliente. Se pone en uso una vez que ya se haya calculado el costo por el uso que el cliente haya tenido.
2. `add_base_to_variable`: se utiliza al momento de calcular el costo del uso del producto y sirve para saber si el plan que se negoció con el cliente se cobra sumando el costo base o utilizando el costo unitario del tramo variable en caso de aplicar.

El esquema de información del campo “`condition`” se puede ver en la figura 4.1, mientras que en la figura 4.2 se puede observar una manera visual de entender el modelo de precios y condiciones comerciales de los clientes. Es importante mencionar que se hizo un uso avanzado de RoR mediante la creación de artefactos de validación que permiten mantener los esquemas de datos establecidos en la figura 4.1. Al mismo tiempo, también

```

{
  "base": {
    "price": "float",
    "qty": "int",
  },
  "variable": [
    {
      "from": "int",
      "to": "int",
      "price": "float",
    },
  ],
  "step": {
    "free_amount": "int",
    "price": "float",
    "step": "int",
  },
  "add_base_to_variable": "bool",
  "general_minimum": "float",
}

```

Figura 4.1. Esquema de información de condición comercial.

se utilizó validación de los datos en si mismos, forzando a que siempre se tengan tramos positivos, sin solapamiento y que sean rangos acotados, de manera de que las condiciones siempre tengan las formas de la figura 4.2.

En segundo lugar, el modelo de relación plan (`plan_relations`) tiene el rol de asociar el contrato del cliente con el plan creado de manera desacoplada y permitiendo la reutilización de planes estándares.

En tercer lugar, “`plan_relation_proposals`” tiene el propósito de permitir que se pueda tener una agrupación de relaciones de planes en un mismo modelo en caso de que un vendedor haya pactado con un cliente condiciones no estándar. Las condiciones estándar se definieron en conjunto con los vendedores a lo largo de múltiples reuniones y una posterior validación de Felipe Porter, jefe del área de ventas de la empresa. En caso que las condiciones que un vendedor otorgó no sean las predefinidas se acordó que estas requerirían aprobación de Felipe Porter o Sebastián Ojeda (CEO). En caso contrario, las condiciones para el cliente determinado se aprueban de manera automática y se pueden utilizar al momento de facturar.

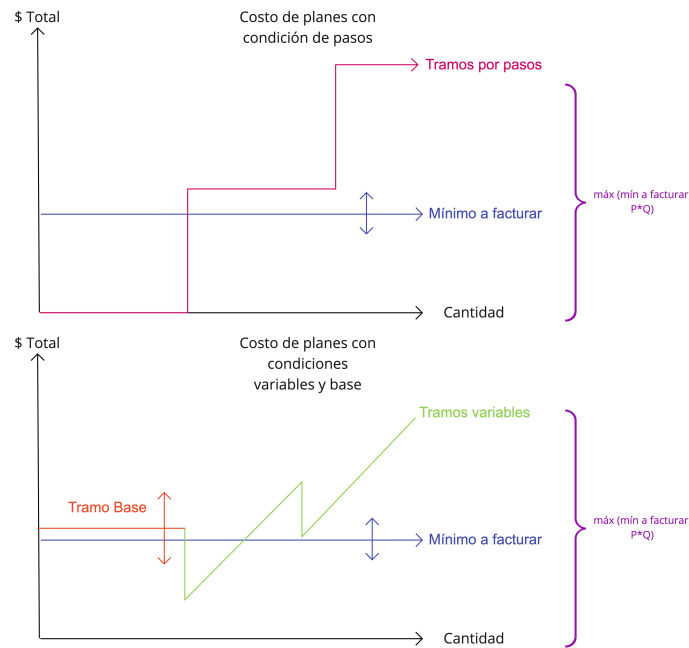


Figura 4.2. Modelos de cobros y condiciones comerciales.

En cuarto, y último lugar, el modelo “comments”, tiene el propósito de contener el cuerpo del comentario por parte del vendedor en caso de que quiera hacer notar algo de los planes que está creando.

En la figura 4.3 se puede observar los diferentes modelos y sus relaciones, junto con sus campos y tipos de datos por campo.

4.2.2. Solver de condiciones y arquitectura

El paso siguiente a la creación de los modelos a utilizar, fue el de la generación de un módulo dentro del servicio de condiciones comerciales que permitiese calcular la facturación adecuada para un cliente en específico dado el uso que este tuvo durante el mes. El módulo de cálculo se basa en un servicio específico para cada producto existente dentro de la empresa, es decir, por cada producto se tiene un servicio específico para calcular y resolver el cobro a realizar para ese producto dado el uso entregado.

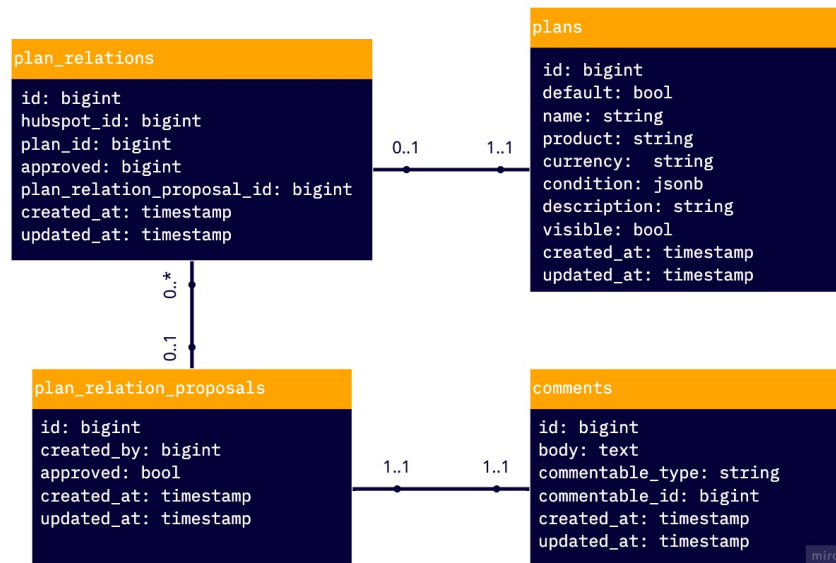


Figura 4.3. Relaciones de modelos de condiciones comerciales.

La información externa que el servicio de cálculo requiere es el identificador del cliente, junto con el uso de los productos. El servicio de cálculo, con la información entregada, se programó de manera de que pase por una calculadora general y delegador; de este modo se subdelega el cálculo del monto a facturar a los subservicios específicos por productos. Este último diseño se basó en la implementación del patrón de diseño “mediador”, en el cual se restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador (Refactoring Guru, 2021b) y “fachada”, en el cual se proporciona una interfaz simplificada a una biblioteca (Refactoring Guru, 2021a). Por otro lado, la información del tipo de cobro se obtiene directamente desde la base de datos del servicio mediante el producto que se quiere calcular y el identificador del cliente para el cual se está calculando.

Una vez que se realiza el cálculo por producto, el servicio retorna la información a un método que agrupa los diferentes resultados, suma el total y entrega un desglose por producto y por moneda, especificando el servicio de resolución utilizado, junto con la glosa a utilizar al momento de generar la factura en el ERP. En la figura 4.4 se puede ver de manera general el flujo utilizado en el *solver* y calculadora de costos.

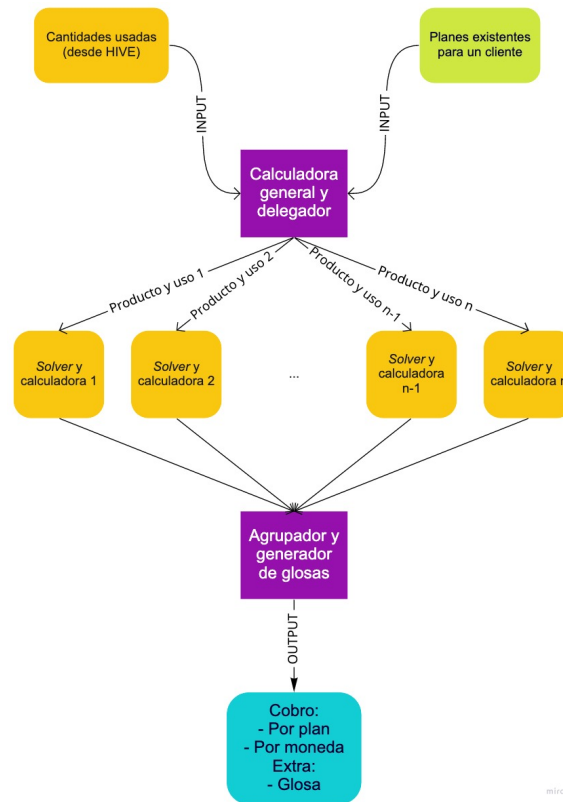


Figura 4.4. Esquema de delegación de *solver* y calculadora.

Posteriormente, una vez que el alumno terminó de programar el servicio de cálculo, este procedió a conectar el ERP con el servicio de condiciones comerciales. Para realizar la conexión se optó por utilizar una API con GraphQL expuesta por parte del servicio de condiciones comerciales. Por parte de *Hive*, se utilizó Artemis, una librería de RoR que permite conectar el cliente, en este caso el ERP, con la API de manera casi invisible, exponiendo los métodos como si fuesen nativos de la aplicación (Nishijima, 2021). Esta implementación permite, al mismo tiempo, que otras aplicaciones dentro de Beetrack consuman el servicio mediante el ERP, en caso de ser requerido. En la figura 4.5 se puede observar la arquitectura con los dos *pods* diferentes de GKE, uno para el ERP y otro para el servicio de condiciones comerciales y su comunicación mediante una API con GraphQL.

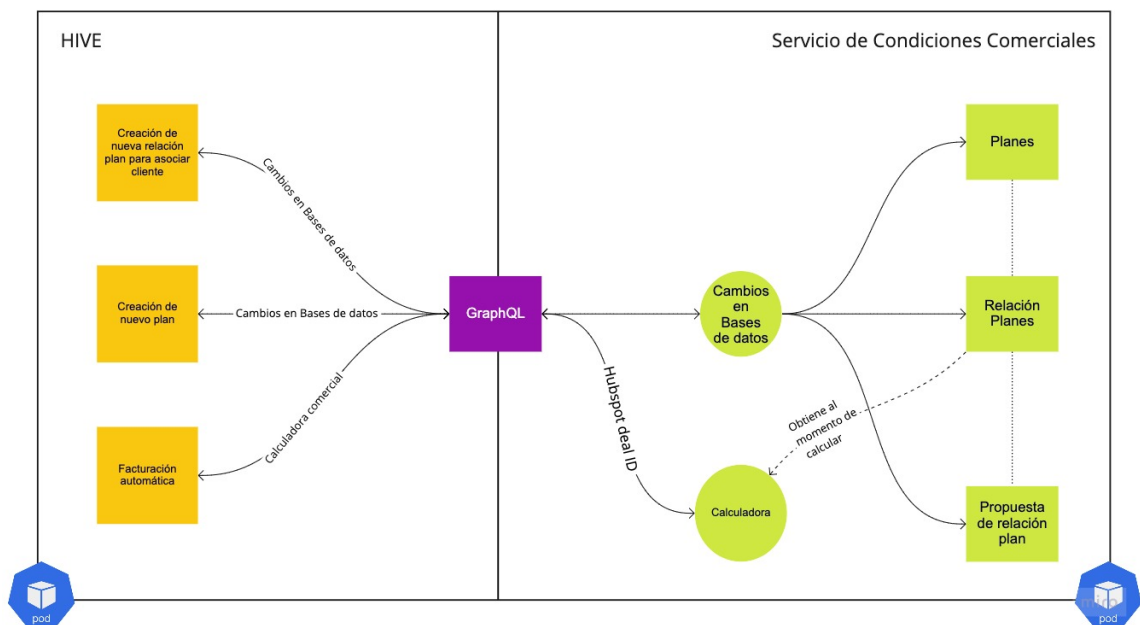


Figura 4.5. Relaciones de modelos de condiciones comerciales.

4.2.3. Refactoring y generación de facturas automáticas

Una vez que el alumno conectó el servicio de condiciones comerciales al ERP, este procedió a desacoplar el sistema antiguo que generaba las facturas de manera automática los 28 de cada mes. Esto se hizo en favor del servicio nuevo externo a *Hive*, adaptando el código antiguo al sistema nuevo. Esto permitió un desacoplamiento de la lógica por parte del ERP, delegándosela al sistema creado. Por otro lado, el cambio implicó la eliminación de más de mil líneas de código en el ERP, con la generación de glosas y cálculo de montos a facturar siendo los principales elementos removidos.

Posteriormente, el alumno le pidió al CEO de la empresa, Sebastián Ojeda, el archivo de Excel en el cual mantenía las condiciones comerciales de manera de poder poblar el servicio. Para esto, el estudiante se comprometió a no entregar los datos de facturación a nadie sin autorización previa, manteniendo un código ético intachable dada la sensibilidad de la información. Adicionalmente, dado el elevado número de contratos de la empresa

(mayor a 700), se decidió escribir código que transcribiese el archivo excel al formato del servicio, generando más de 1500 planes diferentes válidos.

Luego de que se transcribieran los planes generados al servicio de condiciones comerciales, el universitario se coordinó con el equipo de *Data Science* de manera de poder hacer simulaciones de facturación antes de poner en funcionamiento el servicio de generación de pre-facturas reales. De esta manera, se simuló una ejecución del problema real y se puso a prueba el código desarrollado para tener conocimiento de cuán preciso era. El alumno trabajó con el equipo de *Data Science*, dado que este es el encargado de la carga de datos de uso de los clientes, por lo que se pudo generar simulaciones con diferentes niveles de uso y para diferentes clientes. Los resultados obtenidos fueron altamente prometedores, resultando en una precisión de facturación de 87,9 %. Finalmente, el 28 de octubre corrió el servicio de generación automática de pre-facturas exitosamente y recibió comentarios muy positivos por el equipo contable de la empresa dado que tuvieron que modificar muy pocas pre-facturas antes de emitirlas.

4.2.4. Creación de condiciones por parte de vendedores y aprobación de gerentes

En paralelo, junto con la generación automática de pre-facturas, el alumno trabajó en 4 vistas dentro del ERP, las cuales corresponden a dos formularios y dos vistas generales. Adicionalmente se crearon 3 plantillas de correos mediante HTML, CSS y Ruby embebido; estas se detallarán más adelante. Estas vistas fueron validadas y diseñadas junto a Grace Lillo, del equipo de *design ops*, de manera de que fueran realmente útiles para los usuarios finales. Cabe mencionar que estas vistas fueron programadas utilizando los componentes de *Honeypot* en React.

En primer lugar, el estudiante generó una vista general en donde los vendedores pueden ver las condiciones comerciales preaprobadas por los gerentes, de manera de poder tener una noción de cuáles son las condiciones de cada plan antes de utilizarlas para un cliente determinado. A continuación, en la figura 4.6, se puede ver la vista mencionada anteriormente.

Nombre CC	Producto	Detalle de rangos	Precios por rango	Moneda	Comentarios
Geocoding Google Maps plan Chile	Google Maps	0 - ∞	0.1	UF	Este plan aplica para Chile
Geocoding Google Maps plan Mexico	Google Maps	0 - ∞	79	MXN	Este plan aplica para Mexico
Geocoding Google Maps plan Otros	Google Maps	0 - ∞	3	USD	Este plan aplica para otros países
Optimización por ventana horaria plan Chile	Optimización LastMile	0 - ∞	0.1	UF	Este plan aplica para Chile
Optimización por ventana horaria plan Mexico	Optimización LastMile	0 - ∞	79	MXN	Este plan aplica para Mexico
Optimización por ventana horaria plan Otros	Optimización LastMile	0 - ∞	3	USD	Este plan aplica para otros países
LastMile Base Chile	LastMile	0 - 9 10 - ∞	1.2 1	UF	Este plan aplica para Chile
PlannerPro Complemento Chile	PlannerPro	0 - ∞	0.5	UF	Este plan aplica para Chile
LastMile Base Mexico	LastMile	0 - 29 30 - 99	790 690	MXN	Este plan aplica para Mexico

Figura 4.6. Vista general de condiciones comerciales preaprobadas.

En segundo lugar, el alumno realizó una formulario que le permite a los vendedores crear asociaciones comerciales, es decir, asociar una condición comercial existente y preaprobada a un cliente determinado. Parte crucial del formulario es el hecho de que se puedan definir precios personalizados y montos mínimos a facturar solamente para los productos y no para los servicios, ya que esto va en línea con el plan de la empresa de tener planes estandarizados y asimilarse más a un SaaS tradicional.

Un punto importante de este flujo es el hecho de que si el vendedor cambia el precio predefinido del tramo de un producto, se notificará mediante un correo como el de la figura 4.7 a los gerentes de esta propuesta, de manera de que ésta pueda ser aprobada o rechazada. En caso de que el plan preaprobado no tenga cambios en los precios, este se aprobará de manera automática. Lo anterior fue un pedido específico de parte de los gerentes con el propósito de poder tener mayor control sobre los tipos de planes que crean los vendedores para así evitar la generación de esquemas poco comunes. El formulario mencionado se puede ver en la figura 4.8.

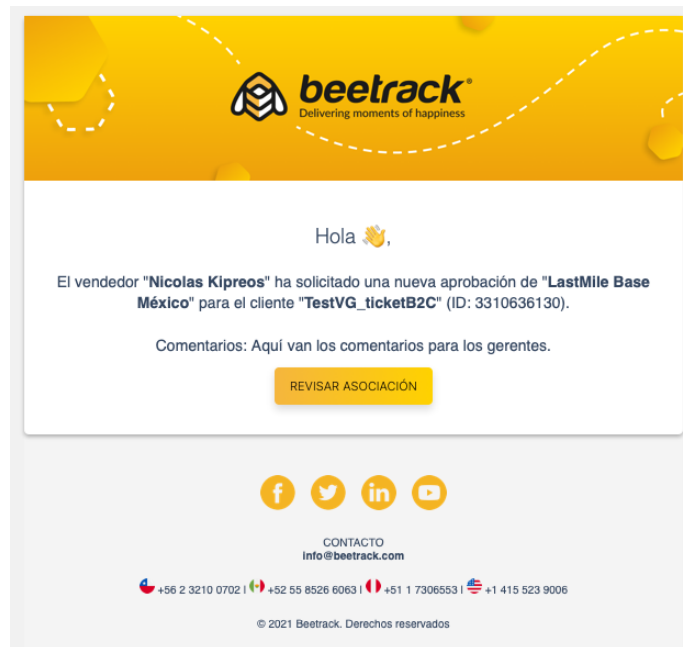


Figura 4.7. Correo electrónico de solicitud de aprobación de asociación comercial.

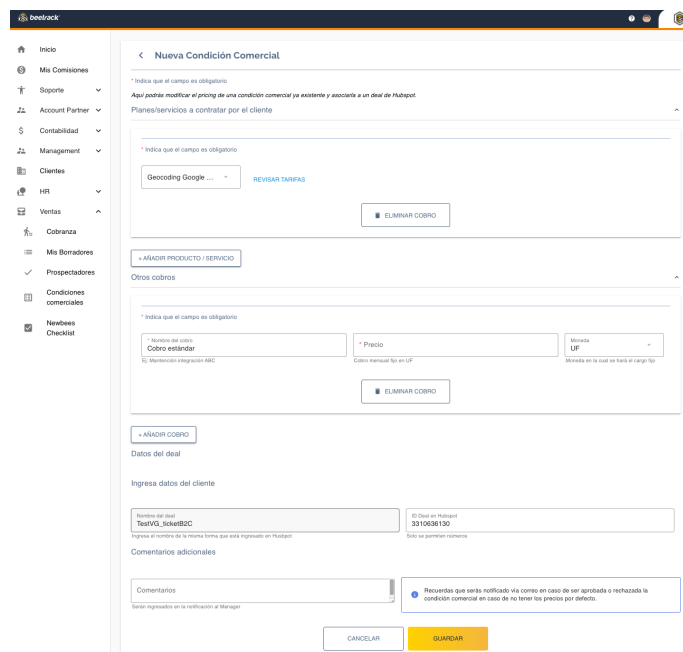


Figura 4.8. Vista general de condiciones comerciales preaprobadas.

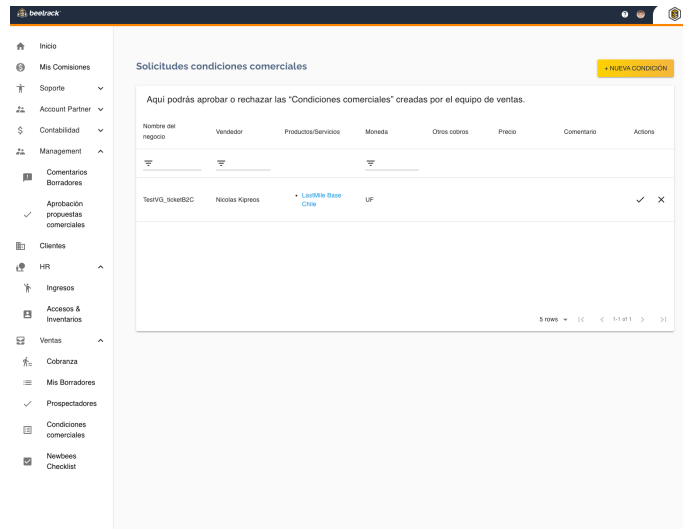


Figura 4.9. Vista de aprobación y rechazo de asociaciones comerciales.

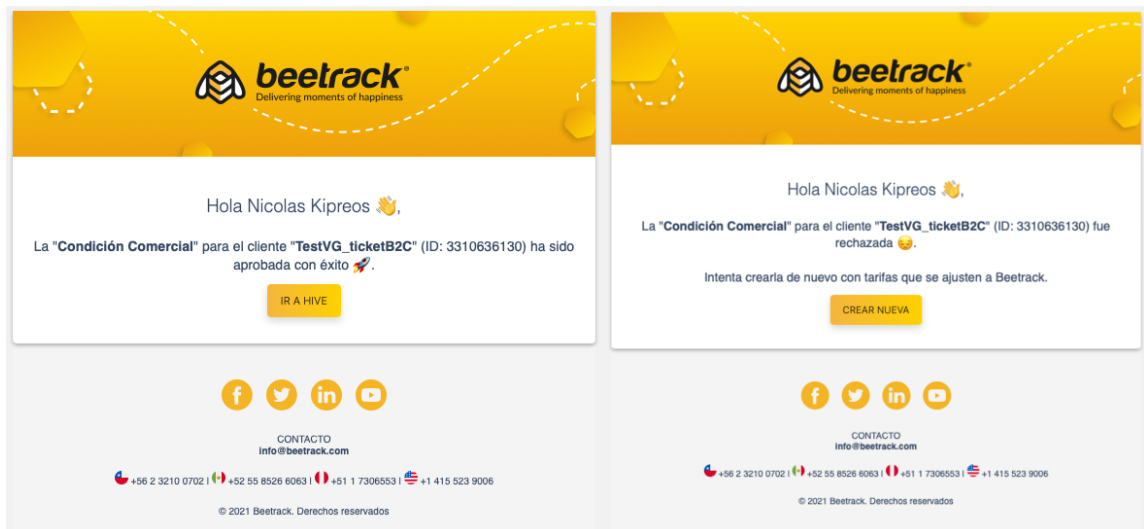


Figura 4.10. Correo electrónico de aprobación (izquierda) y rechazo (derecha) de asociación comercial.

En tercer lugar, el alumno programó una vista para los gerentes en la cual estos pueden aprobar o rechazar las propuestas de asociaciones comerciales creadas con precios no estándares. Esta vista se puede ver en la figura 4.9. Una vez que los gerentes aprueben o rechacen la propuesta, esto se notificará mediante el correo de aprobación o rechazo de la figura 4.10 al vendedor que creó la propuesta.

En cuarto, y último lugar, se realizó un formulario totalmente personalizado para la creación de planes no estandarizados, tanto para productos como servicios. A este formulario no tienen acceso los vendedores, sino que solamente los gerentes. El propósito de esto es poder crear un plan totalmente a medida para un cliente en caso de que un plan estándar no permita abarcar el tipo de contrato que se cerró. Esta decisión de restricción del formulario a los gerentes se generó principalmente para evitar el fomentar que los vendedores vendan el producto de manera no estandarizada y que Beetrack se pueda acercar más a un SaaS tradicional. En la figura 4.11 se puede observar el formulario descrito anteriormente.

Figura 4.11. Vista general de condiciones comerciales preaprobadas.

4.3. Resultados

Los resultados obtenidos para este proyecto estuvieron por sobre lo esperado en un comienzo. En particular se logró lo siguiente:

1. Servicio independiente de *Hive*.

Se creó un servicio independiente del ERP en RoR, que permitirá que se tenga la información descentralizada y se pueda enfocar en su única función de cálculo de costos y almacenamiento de planes de los clientes con tecnologías que no habían sido utilizadas dentro de la empresa anteriormente (GraphQL).

2. Creación de modelos de planes flexibles.

Se logró diseñar modelos de planes y condiciones lo suficientemente flexibles de manera que incluyeran todos los casos existentes y por existir. Junto a esto, la forma en que se abarcó el campo “*condition*” del modelo permitió tener un *solver* flexible que se adaptase a la forma de resolver el costo de los planes de manera correcta. Esta flexibilidad se logró mediante el trabajo entre múltiples áreas de la empresa, de manera de que se pudiese entender de mejor manera el problema existente. Al mismo tiempo, el esquema del modelo se logró definir gracias a un **análisis sistemático de los problemas** de los usuarios y trabajo en conjunto.

3. Servicio con 87,9 % de precisión de facturación.

Se creó un servicio de resolución de condiciones comerciales independiente del estado de los clientes, que permite calcular el costo asociado al uso de cada cliente, por producto, por servicio y por moneda. Esto permitió que se tenga una facturación mucho más exacta que otros intentos para la resolución de este problema, subiendo en más de un 40 % la precisión de la facturación de más de 700 facturas desde el último intento que fue en Hubspot con una certeza de 43 %. Esto se pudo lograr mediante una batería de **simulaciones de soluciones** y aplicación de **conocimientos avanzados de ingeniería de software** de manera de crear un *software* escalable mediante diferentes patrones de diseño.

4. Formularios de creación de condiciones.

El alumno programó las vistas de *frontend* dentro de *Hive* mediante el diseño en conjunto con *design ops*, resultando en los formularios especializados para los vendedores y el totalmente personalizable para los gerentes. Estos formularios quedaron **expresados en JavaScript** en el *framework* de React.

5. Vista de aprobación.

Se desarrolló una vista especializada que permite que los gerentes de la empresa aprueben o rechacen las propuestas de asociaciones comerciales creadas por los vendedores en caso de que no sean tengan valores preaprobados. El objetivo de esta vista es poder tener una visión rápida de lo que están haciendo estos funcionarios de la empresa y también poder tener un control fino de qué tipos de condiciones comerciales se tiene para los clientes. Esta vista quedó plasmada en el lenguaje JavaScript en el *framework* React.

6. Correos de estado propuestas comerciales.

Se crearon 3 correos automáticos que se envían en diferentes momentos acorde a los cambios de estado que ocurran para los planes y para las propuestas de asociaciones comerciales.

CAPÍTULO 5. CONCLUSIONES

5.1. Competencias evidenciadas

El trabajo realizado por el alumno como ingeniero de *software*, en Beetrack durante los cuatro meses evidencian de manera clara cada una de las tres competencias declaradas del perfil. En específico se pudo evidenciar lo siguiente:

1. Aplicar conocimientos avanzados de Ciencia de la Computación, Ingeniería de Software y Sistemas de Información para entender problemas complejos y abiertos.

El estudiante desarrolló un sistema especializado con una arquitectura de software diseñada para ser escalable y replicable rápidamente mediante el uso de tecnologías como GKE y contenedores de Docker, como también hizo uso avanzado de del *framework* RoR utilizando validadores personalizados en los modelos. Adicionalmente, utilizó de manera amplia el *framework* de pruebas RSpec, llevando buenas prácticas de *software* al ERP de manera de poder eliminar el problema de la inseguridad que se tiene por parte del equipo al hacer cambios en el código.

Por otro lado, para poder diseñar de manera adecuada los aparatos de *software* creados, el alumno tuvo que aplicar conocimiento teórico de patrones avanzados de diseño, de manera de que se pudiese tener un código altamente cohesivo y con bajo acoplamiento, algo que es sumamente deseado. Para poder abordar los dos puntos mencionados anteriormente, el estudiante debió comprender las diferentes problemáticas del problema y utilizar sus conocimientos en las diferentes áreas.

Finalmente, el estudiante utilizó conocimientos de sistemas de información, tales como la lógica de negocio de un ERP, que fue en lo que él trabajó. Esto se traduce directamente en la combinación e integración que se realiza con Hubspot y Quickbooks, junto con el uso que *Hive* le da a nivel de lógica de negocio.

2. Diseñar y desarrollar modelos y artefactos de software y simular soluciones a problemas de la Ciencia e Ingeniería de Computación, cumpliendo con restricciones técnicas, sociales y éticas.

Por un lado esta competencia se puede evidenciar mediante el aumento sistemático del nivel de cobertura de código del ERP. Esto se debe a que las pruebas automatizadas son una forma de simulación de ejecución del código en un ambiente aislado, solucionando el problema de confiabilidad y resiliencia ante fallas. En este sentido, el alumno logró diseñar artefactos de *software* que simularan la ejecución del mismo código escrito, manteniendo resguardada información sensible de otros entornos.

Por otro lado, esta competencia también se pudo ver evidenciada en las simulaciones de la emisión de pre-facturas y cálculo de uso por parte de los clientes. Esto se debe a que para poder tener seguridad que el servicio programado tenga una mayor precisión que los sistemas anteriores se debió realizar simulaciones de uso de manera de no intervenir los datos reales. A su vez, esto implicó restricciones sociales y éticas al no poder compartir la información de facturación real con nadie más, y al mismo tiempo se logró un incremento de casi un 50 % en términos de precisión de facturación.

Adicionalmente, se desarrolló modelos y artefactos de *software* mediante la creación de los modelos que el servicio de condiciones comerciales aloja (`plans`, `plan_relations`, `plan_relation_proposals` y `comments`), como también de los artefactos que permiten el cálculo correcto de los montos acorde a los modelos generados. Cabe mencionar que los artefactos generados también cumplen con restricciones técnicas en cuanto al hecho de no mantener registros del resultado final, delegando esa tarea al ERP, dado que *Hive* es el sistema que genera la factura real y mediante el cual se realizan los cobros a los clientes.

3. Analizar en forma sistemática las diferentes problemáticas de los usuarios, y diseñar productos o sistemas de software que queden expresados mediante algún

lenguaje de programación, de acuerdo a los estándares de la ingeniería de software.

Esta competencia se pudo ver evidenciada por las diferentes etapas del levantamiento del servicio de condiciones comerciales: modelamiento, creación de un *solver* y generación de vistas.

Para la etapa de modelamiento se analizó la necesidad de migrar las condiciones comerciales a un servicio especializado que tuviese un esquema y desarrollo flexible de manera de que abarcara todos los planes existentes y los futuros. Esta problemática fue identificada dado el tamaño de la empresa y el hecho de que las condiciones comerciales no habían logrado ser abarcadas de manera correcta en dos intentos anteriores y se resolvió con los esquemas planteados en la sección 4.2.1. Este paso quedó plasmado en el lenguaje Ruby montado sobre el *framework* Rails.

Para la segunda etapa, se identificó la problemática del traspaso del cálculo de facturación mediante un archivo Excel no escalable como un problema por parte de la empresa y del área contable de esta. La solución a este problema se ve evidenciada mediante la creación de un *solver* generalizado que se adapta automáticamente dependiendo de las condiciones comerciales de cada cliente y cada producto, quedando expresado en RoR con un esquema de delegación con patrones de diseño como se puede observar en la figuras de la sección 4.2.2.

Finalmente, para la tercera etapa, se hizo notar por parte de los gerentes la necesidad de acotar los campos editables por los vendedores, por lo que el alumno, junto con Grace Lillo, diseñaron los formularios de creación de condiciones comerciales acorde a lo planteado. Adicionalmente, se realizaron validaciones con todas las partes involucradas, de manera de que se asegurara el resolver el problema de todos. Esta solución de formulario de creación de condiciones comerciales quedó expresada en el *framework* React en el lenguaje JavaScript.

5.2. Conclusiones generales

Acorde a lo trabajado durante cuatro meses por el alumno en Beetrack, se puede afirmar que se cumplieron las tres competencias planteadas inicialmente en la sección 1.4 y que estas fueron ampliamente validadas. Para lograr las metas y competencias propuestas, el estudiante se desempeñó como Ingeniero de *Software full-stack* dentro del área de operaciones en Beetrack.

Los proyectos que el universitario llevó a cabo dentro del equipo de operaciones fueron dos: la creación de un servicio de condiciones comerciales y el aumento sistemático de la cobertura de pruebas automatizadas. El servicio creado le permitirá a la empresa tener una facturación mucho más precisa que antes, logrando un 87,9 % de precisión, mientras que al mismo tiempo le facilitará el tener servicios especializados para el cálculo del costo por uso de cada cliente y producto y que este cálculo se integre de manera simple con el ERP. Por otro lado, el incremento de cobertura por parte de las pruebas automatizadas le entregará una mayor seguridad a al equipo y a *Hive*, al mismo tiempo se podrá detener la puesta en producción del código fallido de manera preventiva gracias a la integración continua que se realizó con Circle CI.

Finalmente, el alumno tuvo la oportunidad de desempeñarse de manera profesional y pudo utilizar de manera amplia los conocimientos adquiridos durante sus años universitarios. Adicionalmente, también logró desempeñarse de buena manera con diferentes áreas mediante un trabajo en equipo, lo que le permitió también hacer uso de las habilidades blandas adquiridas en la carrera. La combinación del uso de capacidades blandas como conocimientos teóricos y prácticos permitieron que el alumno lograra un buen trabajo en equipo y buen desempeño en cuanto a la resolución de problemas y soluciones innovadoras a problemas complejos.

REFERENCIAS

- APD. (2019). *¿Cómo aplicar la metodología Scrum en tus proyectos empresariales?* Descargado 2021-11-08, de <https://www.apd.es/metodologia-scrum-que-es/>
- Atlassian. (2021). *Gitflow Workflow*. Descargado 2021-11-08, de <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- Bascón Pantoja, E. (2004, 12). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. *Acta Nova*, 2, 493 - 507. Descargado de http://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S1683-07892004000100005&nrm=iso
- Burotto, T. (2021). *Mejoras de procesos de negocio mediante el uso de tecnologías de la información: Hive (ERP interno)*. Descargado 2021-10-21, de https://drive.google.com/file/d/1A_3ISZfxddopIL8OPwtdGeAmBQbz9j50/view?usp=sharing
- Corporateit. (2021). *Beetrack duplica su cartera de clientes y su facturación creció en más de un 80 %*. Descargado 2021-11-03, de <https://corporateit.cl/index.php/2021/03/04/beetrack-duplica-su-cartera-de-clientes-y-su-facturacion-crecio-en-mas-de-un-80/>
- Fowler, M. (2006). *Codesmell*. Descargado 2021-11-15, de <https://martinfowler.com/bliki/CodeSmell.html>
- Google Cloud. (2021). *Pod*. Descargado 2021-11-24, de <https://cloud.google.com/kubernetes-engine/docs/concepts/pod#:~:text=Pods%20are%20the%20smallest%2C%20most,and%20share%20the%20Pod's%20resources.>
- IBM. (2021). *How does software testing work?* Descargado 2021-11-14, de <https://www.ibm.com/topics/software-testing>

- IBM Cloud Education. (2021). *¿Qué es una API REST?* Descargado 2021-11-10, de <https://www.ibm.com/cl-es/cloud/learn/rest-apis>
- Nishijima, Y. (2021). *Artemis: Ruby GraphQL client on Rails that actually makes you more productive.* Descargado 2021-11-21, de <https://github.com/yuki24/artemis>
- Red Hat. (2019). *What is GraphQL?* Descargado 2021-11-10, de <https://www.redhat.com/en/topics/api/what-is-graphql>
- Red Hat. (2021). *Qué son las API y para qué sirven.* Descargado 2021-11-10, de <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>
- Refactoring Guru. (2021a). *Facade.* Descargado 2021-11-22, de <https://refactoring.guru/es/design-patterns/facade>
- Refactoring Guru. (2021b). *Mediator.* Descargado 2021-11-22, de <https://refactoring.guru/es/design-patterns/mediator>
- Velotio Technologies. (2019). *Continuous Integration & Delivery (CI/CD) for Kubernetes Using CircleCI & Helm.* Descargado 2021-11-24, de <https://medium.com/velotio-perspectives/continuous-integration-delivery-ci-cd-for-kubernetes-using-circleci-helm-b8b0a91ef1a3>