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

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

Gratefully to my parents and siblings

ACKNOWLEDGEMENTS

Write in a sober style your acknowledgements to those persons that contributed to the development and preparation of your thesis.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

ABSTRACT

The abstract must contain between 100 and 300 words. The abstract must be written in English and Spanish. In the case of doctoral theses, the layout of the abstract page is different, so please check the template provided by the OGRS.

Keywords: thesis template, document writing, (**Write here the keywords relevant and strictly related to the topic of the thesis**).

RESUMEN

El resumen debe contener entre 100 y 300 palabras. El resumen debe ser escrito en inglés y español. En el caso de tesis de doctorado, el formato de la página del resumen es distinta, por favor verifique la plantilla entregada por la Dirección de Postrgrado.

Palabras Claves: plantilla de tesis, escritura de documentos, **(Colocar aquí las palabras claves relevantes y estríctamente relacionadas al tema de la tesis)**.

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, la empresa cuenta con más de 650 clientes activos y más de 100 empleados (Corporateit, 2021) entre Chile, Argentina, Perú, Colombia y México. Al mismo tiempo, esta se ha posicionado a si misma como uno de los líderes del mercado SaaS en América Latina por el éxito que ha tenido su principal producto, LastMile.

Junto con LastMile, Beetrack 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 para 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 áreas en los que se tiene interacciones cruzadas entre distintos equipos. En particular, el trabajo

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

1.2. Descripción del proyecto

Los objetivos del proyecto 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 estos procesos 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.

Para poder lograr los objetivos propuestos, la empresa decidió crear su propio ERP (*Enterprise Resource Planning*) interno de manera de poder adaptar su *software* a sus proceos 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 a los diferentes clientes de Beetrack.

Cabe mencionar que a pesar de que *Hive* tiene un enfoque interno, este *software* posee conexiones con, principalmente, 3 *softwares* 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 *software* 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, como 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 le 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 todo lo que se tenía desde cero, creando un *software* especializado para esta tarea. Por otro lado, el alumno 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 en el ERP de la empresa. Por el motivo anterior, el alumno 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.

1.4. Competencias evidenciadas

Mediante el trabajo realizado dentro de Beetrack, el alumno busca evidenciar 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 evidenciada 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 evidenciada 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 trabajo 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 Martinez y el alumno, 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 lo que se denomina como puntos de esfuerzo. Estos últimos son una forma de medir cuánto tiempo o dificultad tiene cada tarea, de manera de poder mantener un nivel de tareas 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 y 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.

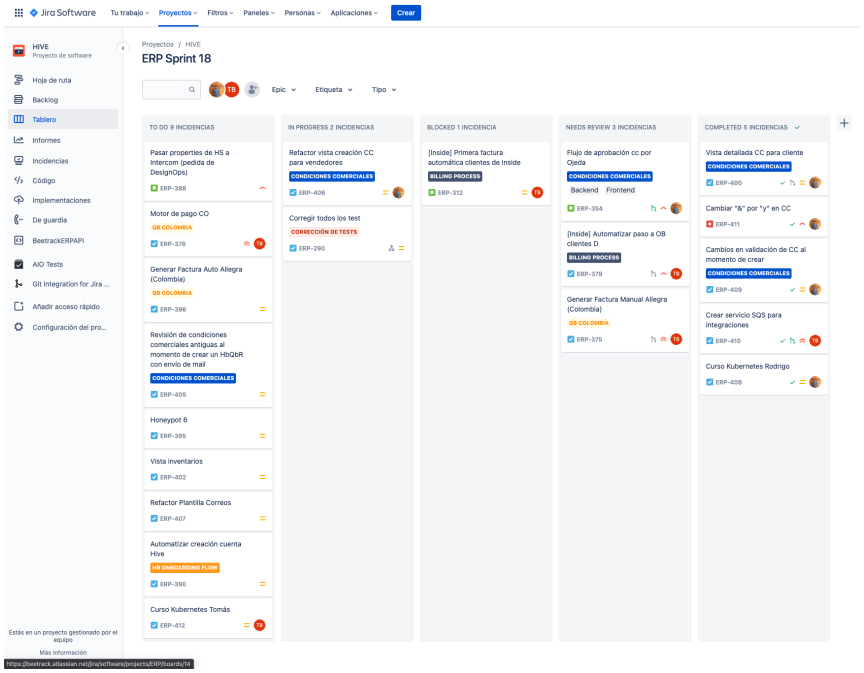


Figura 2.1. Tablero *Kanban* utilizado en el proyecto.

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.

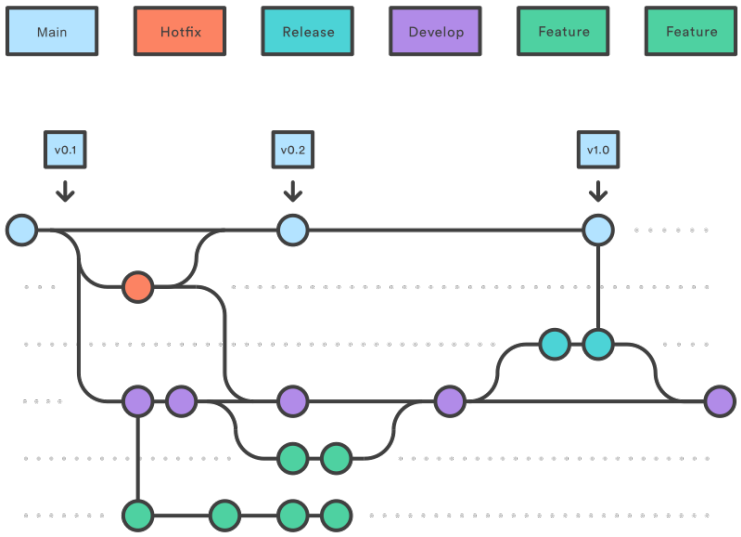


Figura 2.2. *Gitflow* (Atlassian, 2021).

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 código mediante lo que se denomina un *pull request*, para finalmente integrar los cambios a la rama principal una vez que los cambios hayan sido aprobados por los otros miembros del equipo. 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 código 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 descargan el código recién subido, corren pruebas sobre este, verifican su integridad, y finalmente compilan el código para subirlo 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.

2.2. Tecnologías utilizadas

El ERP de la empresa se basa en una arquitectura que separa de manera lógica los componentes. *Hive* mismo tiene una separación entre *frontend* y *backend*, el primero correspondiendo 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 rápidamente 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 del modelo 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 hace uso de 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, mientras que para el resto de los servicios, tales como Hubspot y Quickbooks, entre otros, se utiliza 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

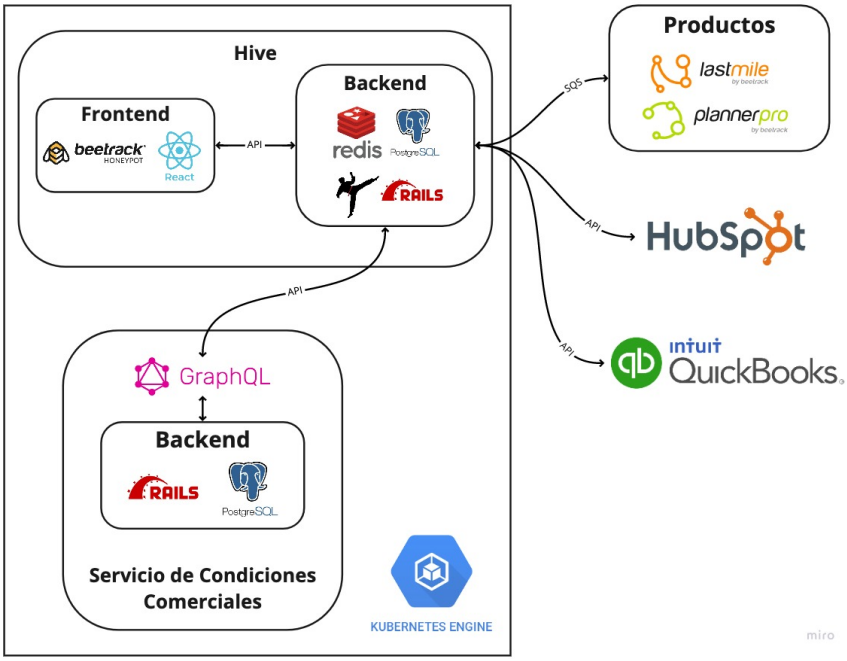


Figura 2.3. Diagrama de tecnologías y arquitectura.

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, mientras que al mimso tiempo se encarga de los procedimientos para generar estos servidores y que se coordinen entre ellos. Otra ventaja que provee GKE es la facilidad para poder volver 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 2.3 se puede ver la arquitectura con la cual el alumno trabajó.

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 testing unitario de los difernte componentes de la aplicación que 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 por 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 reazlizar cambios al *code base* 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 el o los cambios.

En este sentido, es paradójico que, una empresa que se dedica a producir *software*, no haga pruebas 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 pricipal 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 usuario 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 fallase en Circle CI.

3.2. Desarrollo

En RoR, se hace uso de 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 relizaban sobre el, teniendo un comportamiento diferente al esperado. En la figura 3.1 se puede observa el estado inicial de la cobertura de las pruebas del ERP.

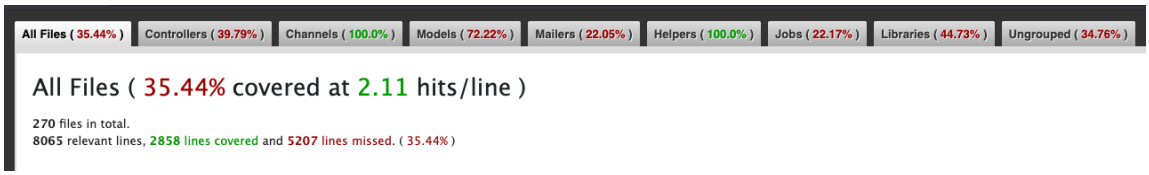
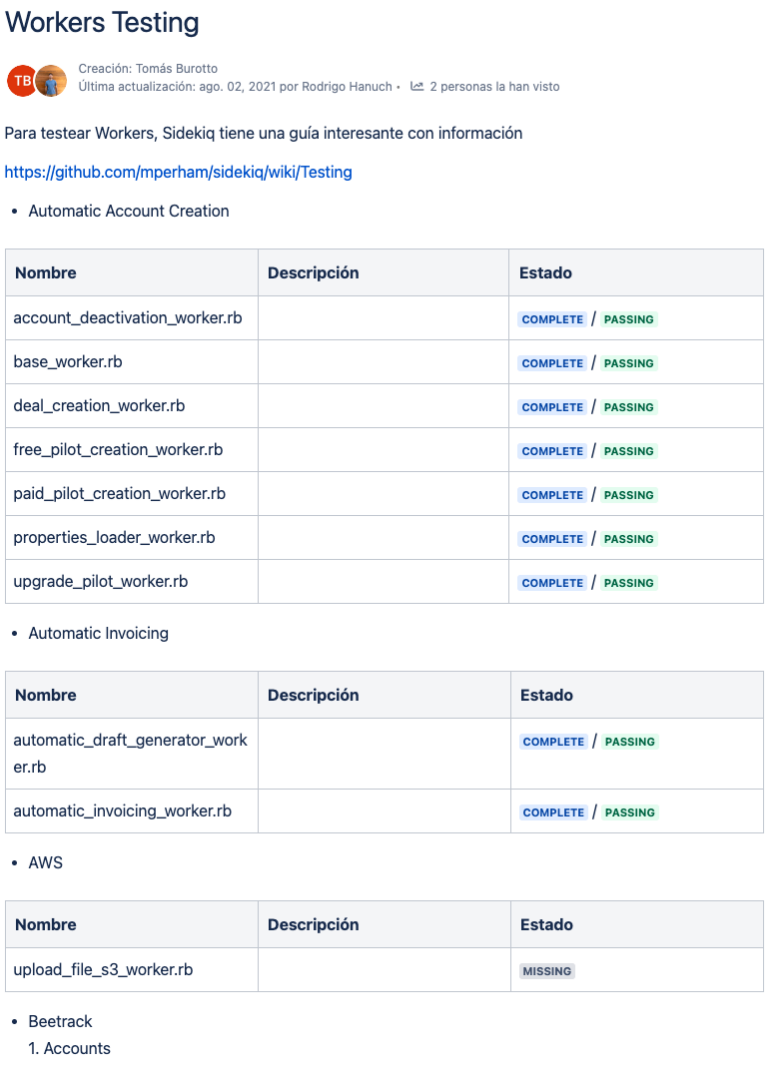


Figura 3.1. Cobertura inicial del ERP (captura tomada el 3 de agosto de 2021).

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 archivo 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 archvios 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 transpasaron 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 econtraban las pruebas: "*passing*" que aplicaba en caso de que todos los *test* estuviesen pasando, "*failing*" para indicar que una o más pruebas fallaba y "*skipped*", que indicaba que una o más de las pruebas era saltada. 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. El punto de esto, nuevamente, es que al realizar pruebas detalladas, se puede tener un desgolse muy granular de qué falla en caso de que falle algo 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 archvios 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 *drafts* 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 se cumplió la meta establecida, el alumno procedió a la siguiente tarea, la cual consistió en el cambió 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 solo contenía la compilación de los archivos y subida 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,

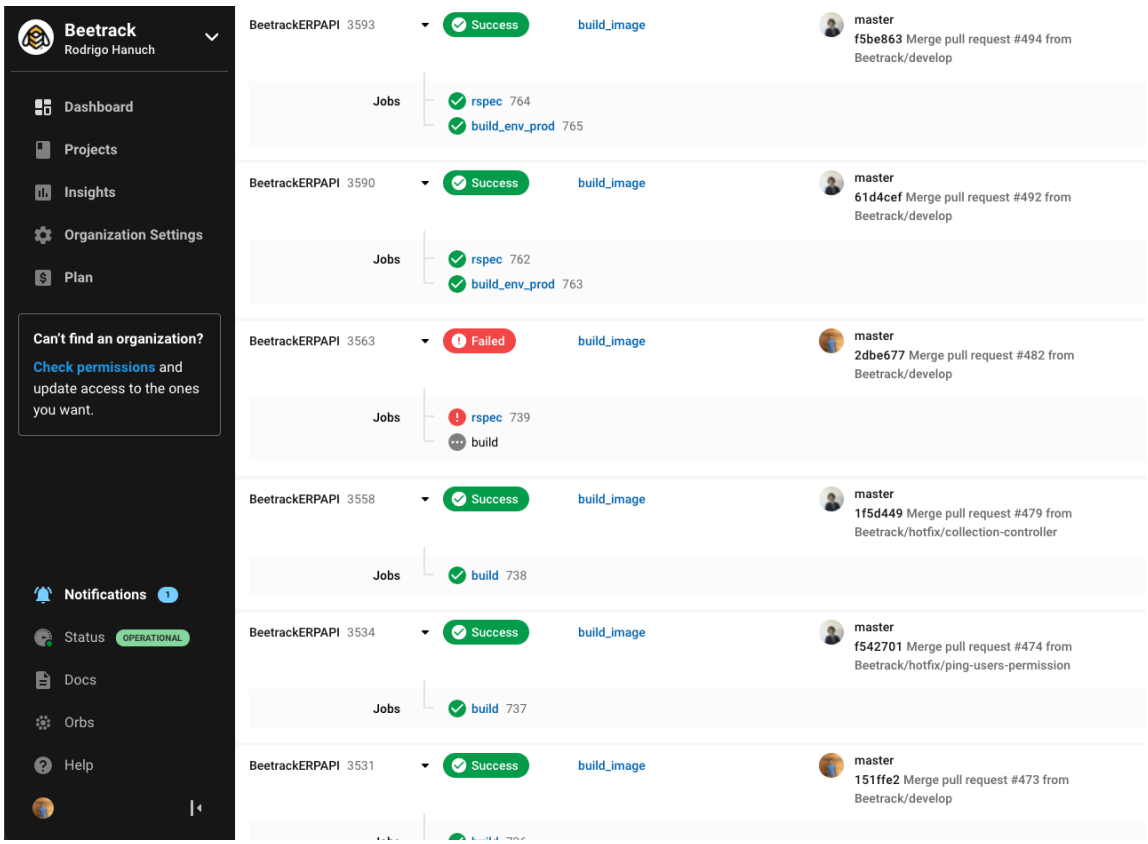


Figura 3.3. Cambio en flujos de Circle CI y falla de *RSpec* previa a la puesta en producción del código.

esto dado que para correr *RSpec* (el comando utilizado para correr las 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 el primer paso fuese existoso, el segundo paso fuese 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 emprsa. 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 defectuso en producción.

3.3. Resultados

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

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

   Se logró una cobertura de casi un 65 % 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.

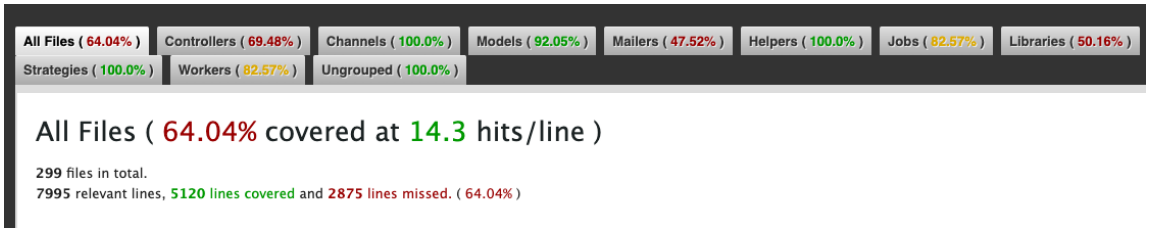


Figura 3.4. Cobertura final del ERP (captura tomada el 15 de noviembre de 2021).

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.

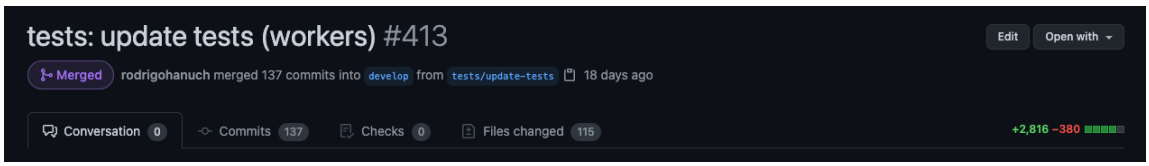


Figura 3.5. Archivos modificados y generados durante creación de nuevas pruebas.

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

4.2. Desarrollo

4.3. Resultados

CAPÍTULO 5. CONCLUSIONES

5.1. Competencias evidenciadas

5.2. Conclusiones generales

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 implementaciónn 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

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

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

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

APÉNDICE

A. FIRST APPENDIX

We can write equations here too:

$$\int_0^\infty e^{-x^2} dx \tag{A.1}$$

And more...

B. AN INTERESTING SHORT STORY

Let us enjoy reading this story of Hunting With The Lion.

It was a dry summer. The animals in the forest were beginning to find it difficult to get food.

A bear, a wolf and a jackal thought it would be better to join hands with a lion and do the hunting. They approached lion and he too agreed. The four of them went off hunting.

The hunting party came across a buffalo. The fox and wolf chased the buffalo. The bear intercepted the buffalo. The lion killed him.

The fox made shares out of the buffalo. When they were about to take their shares the lion roared and said, ”Well friends, the first share is mine for my leadership. The second share is mine for, it is I who killed. The third share is also mine for I need it for my cubs. Anyone who needs a share can take the fourth. But before that you will have to win me.”

All the three left the place without a single word.

MORAL : **If you are might, you are right**.