PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

Escuela de Ingeniería.
Departamento de Ciencia de la Computación

DESARROLLO DE HERRAMIENTAS WHITE-LABEL PARA LA VENTA INMOBILIARIA EN MOME

BENJAMÍN VICENTE GOECKE

Trabajo de Título para optar al Título
de Ingeniero Civil de Computación.

Profesor guía:
JAIME NAVON

20 de enero del 2025

Indice General

Índice de Figuras

Resumen

En este informe, se muestra el trabajo realizado por Benjamín Vicente Goecke en el trabajo de título, entre septiembre y diciembre del 2024, desempeñándose como CTO en MOME. Los proyectos establecidos fueron la creación de un sistema para páginas web *white-label* para la venta inmobiliaria, y la integración de inteligencia artificial en los canales de fuerza de ventas.

Dentro los proyectos, se resolvieron problemas de arquitectura, de la modelación de la base de datos del inventario, del diseño del sistema web requerido para las páginas *white-label*, y del desarrollo de un sistema para integrar una inteligencia artificial para ayudar a la fuerza de ventas.

En cada uno, se logró demostrar las competencias al aplicar conocimientos de computación para entender problemas complejos, diseñar y planificar soluciones innovadoras dado las características de los problemas, y llevar el desarrollo de estas soluciones, desde el realizado en el corto plazo dentro del trabajo de título, hasta el largo plazo con las estrategias y planificación realizada.

Abstract

This report outlines the word done by Benjamín Vicente Goecke during his thesis project, between September and December 2024, working as CTO at MOME. The established projects included the creation of a system for *white-label* web pages for real estate sales, and the integration of artificial intelligence into sales force channels.

Within those projects, challenges were addressed in areas such as architecture, database modeling for the inventory system, the design of the web system required for the *white-label* pages, and the development of a system to integrate artificial intelligence to assist the sales force.

In each case, the competencies were addressed by applying computer science knowledge to understand complex problems, design and plan innovative solutions based on the characteristics of the problems, and carry out the development of these solutions, from the work done within the time of the thesis project, to the long-term, with the strategies and planning carried out.

1. Introducción

1.1. Contexto de la empresa y las soluciones a desarrollar

MOME es una Start-Up tecnológica del mundo inmobiliario (*proptech*). Su objetivo es proveer tecnología a inmobiliarias, para que puedan vender por canales propios, sin depender de muchos terceros. Fue fundada en mayo del 2024, y, al momento de la escritura de este informe, cuenta con 4 empleados full-time y contó con un desarrollador practicante part-time. Yo soy CTO y el único empleado técnico full time. Además, a finales del trabajo de título, contó con 4 clientes.

El software que busca desarrollar es similar a Shopify en mundo de *e-commerce*, o Justo en los restaurantes. Es decir, una plataforma *white-label* para proyectos inmobiliarios, permitiendo la creación de su página web, gestión de su inventario, herramientas para la comunicación con clientes, integraciones con diferentes servicios y reportaría.

El mercado inmobiliario es considerablemente diferente con otros mercados. Primero, el valor por unidad es el más alto de los bienes esenciales, y por tanto, la compra es infrecuente. Segundo, dado que suele ser la inversión más grande de una persona, la decisión de compra es lenta, se toma con mucha información, y el proceso de venta es largo.

Considerando el contexto internacional actual, es un mercado que recientemente se ha encontrado en una recesión, y no había tenido la presión hacia la digitalización que otros mercados han tenido. Por último, el funcionamiento en la venta inmobiliaria es considerablemente distinto en países como Estados Unidos, donde los procesos están estructurados para requieran corretajes.

En este contexto, la solución que se busca desarrollar inicialmente cuenta de 2 pilares. Uno es el desarrollo de los medios propios, como es la página web. Esta deberá ser optimizada para motores de búsqueda (SEO), y para la experiencia y conversión (*user experience*, UX). Además, deberá ser alimentada por un gestor de contenido e inventario (*content management system*, CMS).

El segundo pilar es la gestión de clientes. Se busca desarrollar la integración a las bandejas de entrada (*inbox*) de mensajes de los clientes, para poder asistir a la respuesta inicial de estos con inteligencia artificial, y proveer herramientas para la gestión y seguimiento de los clientes. Lo anterior, para entregar visibilidad del proceso a los vendedores y las inmobiliarias.

1.2. Objetivos del Trabajo de Título

Dentro del marco del trabajo de título, las competencias que se escogieron a demostrar son:

- Aplicar conocimientos avanzados de Ciencia de la Computación, Ingeniería de Software y Sistemas de Información para entender problemas complejos y abiertos.
- Diseñar aplicaciones de software complejas, siguiendo los estándares de la Ingeniería de Software, y tomando aspectos de seguridad y complejidad computacional.
- Desarrollar soluciones innovadoras basadas en conocimientos avanzados de Ingeniería de Computación.

Estas competencias se pueden entender como etapas en el desarrollo de soluciones tecnológicas, en donde hay que entender los problemas a resolver, diseñar soluciones, llevar desarrollos y estrategias de trabajo, y finalmente, implementar estas con la programación. Defino en más detalle cada paso en el Anexo 2..

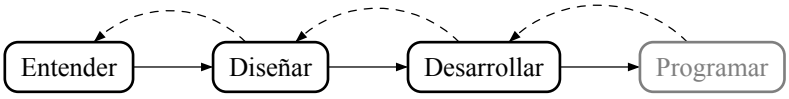

Figura 1: Etapas del desarrollo de software

En la inscripción del trabajo de título, se plantearon los siguientes 2 proyectos a desarrollar:

1. El desarrollo de sistema *white-label* para la creación de sitios web de proyectos inmobiliarios. Estos es, la creación de componentes que puedan ser personalizados como deseen las inmobiliarias, y la integración a un *content management system* (CMS) que permita administrar las características e inventario de las unidades.

2. La implementación de un *shared inbox* impulsada por inteligencia artificial. Plataforma que permita el control y la gestión centralizada de la fuerza de ventas vía canales como WhatsApp, impulsada por inteligencia artificial, para otorgar respuestas 24/7 y reducir el trabajo realizado por la fuerza de ventas.

Estos proyectos, a pesar de resolver problemas distintos, comparten la necesidad de un sistema centralizado que permita su administración. Por lo tanto, también se buscó desarrollar un sistema de datos centralizado administrable en un *dashboard*.

1.3. Contexto adicional

Es importante considerar que se inició el desarrollo de estos proyectos sin ningún conocimiento previo de la industria inmobiliaria. También se comenzó sin ningún desarrollo tecnológico previo. Por lo tanto, se investigó e iteró desde cero que soluciones serían más adecuadas para todos los componentes de la solución que será desarrollada inicialmente en la empresa.

Como rol de CTO, me desempeño también en gran parte de las reuniones comerciales para entender los problemas que enfrentan los clientes. También, participé en la formación de la estrategia de la construcción del producto que ofrecemos, y coordiné y asistí en el trabajo más técnico de mis compañeros, como por ejemplo, en instalar Microsoft Office.

1.4. Orden del cuerpo de este informe

Este informe, se destacarán los diferentes desafíos que involucraron el desarrollo de los proyectos mencionados, y como se demostró el análisis del problema, el diseño de la solución y la implementación de esta, en sus secciones correspondientes.

Previo a aquello, explicaré las estrategias de trabajo que se siguieron en todos los proyectos, cubriendo tanto metodología de trabajo como el principio de composición, que permitió la rápida iteración en el desarrollo de software.

2. Estrategias de trabajo

2.1. Metodología de trabajo

Se decidió seguir fuertemente los principios ágiles en el desarrollo de software (*Manifesto for Agile Software Development*, 2001). Especialmente, se buscó iteración continua y rápida, simplicidad de soluciones, y participación activa con mis compañeros no técnicos.

Yo implementé y organicé un seguimiento de tareas, usando Linear. Fui reconociendo los proyectos y tareas claves, y las organicé en tareas accionables concretas. Con el equipo, iterativamente fuimos priorizando los proyectos, y cada día revisábamos el avance.

Si bien llevamos ciertos rituales asociados a la metodología Scrum como reuniones de revisión diarias (*The Scrum Guide*, 2020), decidimos no seguirla estrictamente dado el contexto de la empresa, donde no existe un producto completamente definido como tal, y se espera validar y experimentar con nuestros primeros clientes.

2.2. Composición en el desarrollo de software

Uno de los principios fuertemente usados en el desarrollo fue la composición. Se entiende como composición el patrón que permite construir sistemas complejos a partir de sistemas más simples e independientes. Una buena composición sigue los principios SOLID (Martin, 2000), aplicados a cualquier nivel de abstracción que involucre la solución.

Por ejemplo, un desarrollo que se realizó fue la generación de PDFs de cotización de proyectos inmobiliarios. La cotización usa la información de una unidad, del cliente, descuentos aplicados, precio UF referencial, y fichas de las unidades y proyectos.

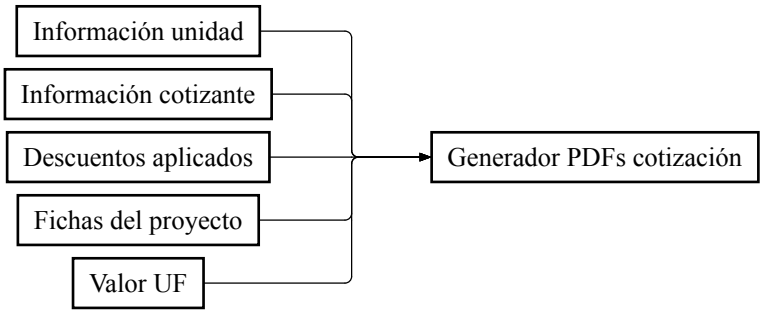

Figura 2: Entradas requeridas para la generación de cotizaciones

La generación misma requiere la implementación de un servidor con una rutina que utilice un sistema que pueda unir la información y generar el PDF. Información como el precio de la UF podría ser obtenido en esta misma rutina, pero esto haría que la generación de PDFs sea más compleja, dado que también se estaría encargando de como se obtiene información.

Uno de todas formas podría programar un software desacoplado, que permita a través de un *dashboard*, cargar la información necesaria, obtener el último valor de la UF, y generar el PDF. Si bien esta es la meta, iniciar así caería en un desarrollo de tipo cascada, contrario al principio ágil, dado que requiere el desarrollo de múltiples funcionalidades antes de tener una versión funcional.

Así que se trató la composición no solo como un principio a nivel de software en un repositorio de código particular, sino como un principio a nivel de sistema y la creación de bloques que puedan ser usados ágil e independientemente, desde pequeños servicios para solucionar rápidamente los problemas, como pequeñas librerías que permitan ser reutilizadas para otros proyectos.

En el caso del ejemplo, lo que se realizó fue montar un servidor que habilite públicamente una rutina que genere PDFs, que tenga como entrada toda la información necesaria, incluyendo las fichas como PDF que deben ser enviadas como binarios. De esta forma, se pudo usar macros (Google Apps Script) de Google Sheet para la carga de datos, y validar la solución.

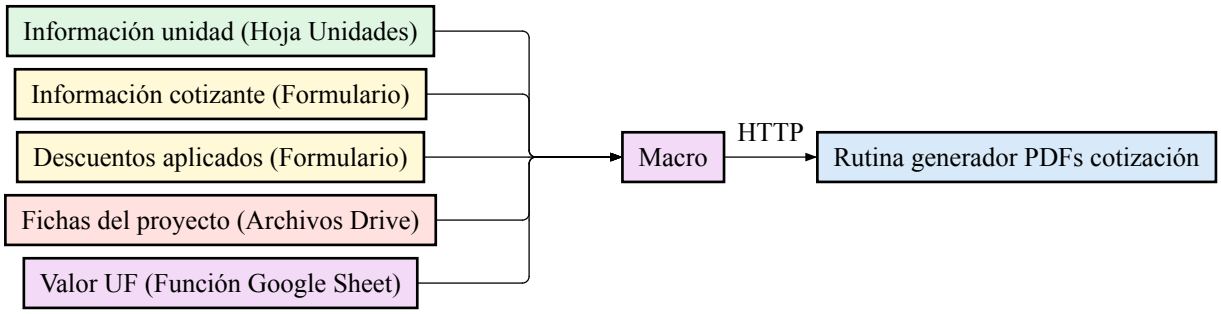

Figura 3: Implementación inicial de la generación de PDFs

Este desarrollo no solo permitió obtener una solución funcional rápidamente, sino que también permitió añadir funcionalidades adicionales para agilizar la operación, como el envió de la cotización al cliente automáticamente, el guardado de los clientes en una tabla, y el guardado de las cotizaciones generadas en una carpeta de Google Drive.

3. Elección de la arquitectura de la aplicación

3.1. Problema a resolver

El sistema que se buscó construir dentro de MOME, tiene 3 componentes principales: las páginas *white-label* para los proyectos inmobiliarios, la base de datos y procesos, y el panel de administrador.

El sistema *white-label* provee las páginas de los proyectos a los cotizantes. Su información cambia infrecuentemente, la del proyecto mismo podría cambiar cada mes, y la información de precios y disponibilidad podría cambiar cada semana. Este sistema requiere tener una velocidad de carga muy alta, SEO y buen UX, y permitir la personalización de la página sin grandes costos.

El panel de administrador es consumido por los gestores de los proyectos, por su personal de ventas y administradores de precios y disponibilidad de las unidades. El acceso a este es infrecuente y por pocos usuarios. Este debe ser intuitivo, cómodo de usar, y mantener los datos consistentes y actualizados.

La base de datos y procesos (*backend*), es consumido por los otros dos sistemas, y permite la integración con otros servicios, como el *shared inbox* y la inteligencia artificial. Este sistema debe permitir diferentes niveles de lectura y modificación a la información, la subida de imágenes, poseer un sistema de tareas, y actualizar los datos en tiempo real.

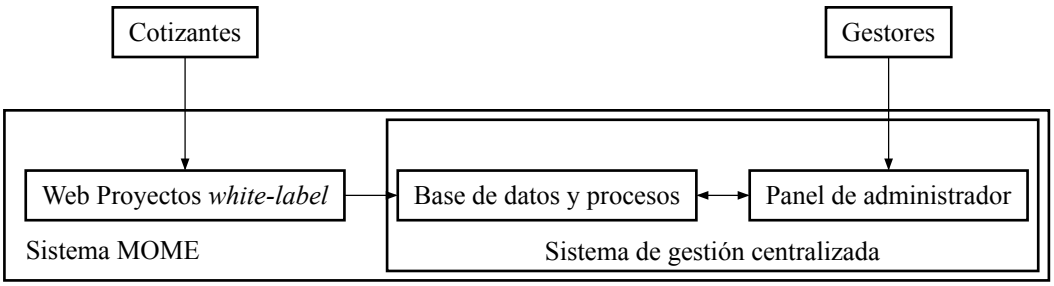

Figura 4: Arquitectura simplificada que se busca desarrollar

3.2. Diseño de la solución

Dado el contexto actual de la empresa, se buscó una arquitectura que permita la rápida iteración sin comprometer sustancialmente la escalabilidad. Para cumplir, la arquitectura deberá permitir fácilmente componer diferentes funcionalidades.

El diseño de el sistema *white-label* se detalla en su sección su sección correspondiente. El panel de administrador, se decidió emplear un sistema simple de *single page application* (SPA) con *React* que se conecta al *backend*, utilizando Vite como *build tool*. Finalmente, el *backend* requirió más análisis, dado que requiere estado persistente tanto en datos como en tareas.

Además de los requerimientos técnicos, es muy importante que el sistema permita la rápida iteración y desarrollo, sin comprometer sustancialmente lo que pueda hacer el sistema. Es por esto, es que se decidió usar un *Backend as a Service* (BaaS) para el desarrollo del backend. De este modo, no se necesitará preocuparse de la infraestructura necesaria para las funcionalidades requeridas.

Utilizar un BaaS sacrifica el control y la flexibilidad en el sistema que se desarrolla, en comparación a soluciones donde uno gestiona la infraestructura, como utilizar un servicio como AWS o Google Cloud. Ese sacrificio es aceptable dado que no hay requerimientos que necesiten arquitectura compleja, y que la etapa temprana de la empresa, en la cual se busca validar soluciones.

Para ver que BaaS utilizar, se realizó un análisis comparativo. Además de revisar que tan rápido sería la iteración de funcionalidades con las diferentes soluciones, se características como que tan fácil sería migrar fuera de la solución cloud. Entre las funcionalidades importantes a considerar, se revisaron las siguientes:

Infraestructura	Funcionalidad
Actualización de datos en tiempo real (*real time sync*)	• Ver el estado actual de la IA sin tener que recargar la página. • Consistencia de datos a través de los diferentes usuarios.
Sistema de tareas	• Ejecutar la IA luego de cierto tiempo de espera. • Permitir actualizar datos con otros sistemas recurrentemente.
Subir imágenes y archivos	• Fotos para las plantas de las unidades y las imágenes de la página del proyecto.
Integración fácil navegador (cliente)	• Agilizar el desarrollo reduciendo la cantidad de código.
Sistema de permisos flexible	• Fácil granularidad de permisos sin comprometer la escalabilidad del sistema. • Evitar que lógica de la aplicación quede en sistemas propietarios o lejanos.

Tabla 1: Infraestructura requerida para funcionalidades del backend

3.3. Desarrollo realizado

Se revisaron varias soluciones, y decidí realizar una comparativa en detalle de 4 de estas: Supabase, Firebase, Cloudflare Durable Objects, y Convex. Aunque Durable Objects no es estrictamente un BaaS, facilita la creación de un backend. El detalle de la comparativa se puede ver en el Anexo 3..

Luego de pruebas de concepto con las diferentes soluciones, y un análisis realizado con la comparativa, se decidió utilizar Convex, destacado del resto de las soluciones por los diferentes atributos:

- Sistema generalmente imperativo y explicito, que evita que el comportamiento de funcionalidades varíe lejos de donde está implementado, como es el caso de las *security rules* y *triggers* en soluciones como Firebase y Supabase.
- Excelente integración con el cliente, que permite agilizar el desarrollo de la aplicación, y reducir la dependencia de lógica de negocios en el frontend.
- Buenas herramientas de desarrollo. Sistema de testing y subida a producción simple.
- Tuvo la implementación, e iteración de funcionalidades, más rápida de todas las soluciones.

Implementa el principio de consultas y mutaciones separadas (Meyer, 1997), permitiendo optimizaciones por defecto, como cache automático en consultas y complicidad ACID en mutaciones (Haerder & Reuter, 1983), importante dado el alto ticket por unidad. Además, permite generar rutinas públicas, que pueden ser llamadas desde la integración que proveen para el navegador.

Se realizó un desarrollo de funcionalidades en Convex, incluyendo las funcionalidades de la gestión de inventario y de la inteligencia artificial que se mencionan más adelante, validando que la solución era adecuada.

Convex no provee un servicio para hostear servicios frontend, por lo que se utilizó Cloudflare temporalmente, dado que posee plan gratuito, y porque ya es parte de la infraestructura de MOME dado que es usado como DNS. Luego del trabajo de título, se evaluará migrar a otro servicio más simple aún, como Vercel.

4. Base de datos del inventario

4.1. Problema a resolver

El sistema que se busca desarrollar debe permitir gestionar diferentes tipos de proyectos. Existen proyectos de casas, parcelas, departamentos, oficinas, y locales comerciales, donde estos pueden tener bienes secundarios como estacionamientos y bodegas, cada uno con precio distinto, y atributos únicos.

Comercialmente, los inmuebles suelen separarse en primarios y secundarios, buscando que los bienes secundarios se vendan junto a los primarios. Legalmente, cada inmueble es independiente. Por ejemplo, en Chile, estacionamientos y oficinas como la misma entidad legal: bienes de dominio exclusivo (*Ley de copropiedad inmobiliaria*, 2022).

La gestión de inventario es distinto al resto de los comercios online. Es el único bien esencial donde cada unidad es significativamente distinta. Otros bienes, como autos, pueden tener unidades que son prácticamente iguales. En cambio, en bienes raíces, cada una tiene una posición única, y atributos que deben ser considerados como propios de la unidad, y no de un modelo compartido.

Es importante que este sistema permita cargar los datos fácilmente y que pueda alimentar a diferentes funcionalidades. Por ejemplo, el cotizador web, con sus respectivos programas, modelos, y bienes secundarios, la generación y registro de cotizaciones en PDF, y canales externos, como otros portales.

Además, si bien este mercado prioriza soluciones enfocadas únicamente a proyectos de viviendas, el objetivo de MOME es abarcar todo tipo de proyectos inmobiliarios, permitiendo la gestión de estos, y conectando estos a las funcionalidades desarrolladas en el sistema.

4.2. Diseño de la solución

Para la modelación de estos datos en el sistema de gestión de inventario consideré 3 características principales. Primero, que datos se normalizaran para evitar repetición. Segundo, como se distinguirán los diferentes tipos de unidades. Y por último, como se relacionan los bienes secundarios con las unidades, permitiendo las restricciones empleadas en los proyectos inmobiliarios.

4.2.1. Normalización y agrupación de datos

Sobre la normalización de datos, los proyectos inmobiliarios de viviendas suelen agrupan las unidades de proyectos en torno a programas, y modelos. Ambos deben ser guardados como entidades separadas y con ciertos atributos:

- Los programas suelen ser usados para guardar cuantos dormitorios y baños tienen las unidades, pero esto lleva a 2 problemas. Primero, hay modelos, como "Studio" y 1D+1B, que son diferentes a pesar de que tienen la misma cantidad de dormitorios y baños. Segundo, esto forzaría a que cada proyecto se agrupe por programas, lo que no ocurre en algunos casos, como los proyectos de oficinas, que tienen baños. Soportar ambos casos añade complejidad innecesaria al sistema.

- En el caso de los modelos, se suele asociar un metraje correspondiente, dado que el mismo modelo está normalmente asociado a columnas de edificios, pero aún que estén en la misma columna, el metraje puede cambiar levemente, y debe ser reflejado al momento de la venta. En los modelos se suele asociar un plano de venta, que puede ser levemente distinto a sus unidades.

Además, hay proyectos que agrupan por torre. Uno de nuestros clientes, poseía una limitación en el flujo de cotización que proveían: este se separaba en torre, a pesar de que lo que se vendía era el proyecto completo. Esto es debido a que se asume que las torres son independientes en el software de gestión que emplean, que es uno de los más usados en Chile.

Es por esto, es que decidí no normalizar datos de las unidades que se suelen asociar a los programas y modelos. Los datos que irán asociados a los programas y modelos estarán más orientados a la visualización y marketing de las unidades, y no la información misma de la unidad.

4.2.2. Diferentes tipos de unidades

Sobre como diferenciar los diferentes tipos de unidades, se debe considerar que cada tipo puede requerir atributos diferentes. Por ejemplo, los departamentos poseen piso y dormitorios, mientras que una casa no posee piso.

Una opción, que se hace en soluciones importantes en el mercado, es separar los diferentes tipos en diferentes tablas, incluyendo bienes secundarios. Esto permite más fácilmente añadir restricciones en los diferentes tipos. Sin embargo, separarlos añade complejidad al sistema, dado que cualquier funcionalidad asociada a las unidades debe considerar toda tabla de unidades.

La solución que se escogió es tener todo tipo de inmueble en una misma tabla, incluyendo secundarios, que serán diferenciados por un atributo. Esto permite mayor flexibilidad en el sistema, ya que cualquier tipo de funcionalidad puede ser implementado directamente en la tabla de unidades, sin requerir unir datos con cada tipo. Y si se requiere restringir, se puede hacer proactívamente.

Para cada tipo de datos, se añadió un atributo discriminador, que permite diferenciar los diferentes tipos de unidades. Dependiendo del tipo, se valida los atributos que son requeridos. Esto además permite mayor flexibilidad al modelar relaciones entre unidades, por ejemplo, en caso que una unidad requiera ser cotizada en conjunto con otra unidad.

4.2.3. Relaciones entre unidades

Finalmente, las relaciones de como están asociadas las unidades en la cotización, fueron modeladas con una tabla distinta, siguiendo la misma estrategia del atributo discriminador presente en las unidades. Cada unidad podría tener un número arbitrario de relaciones, y se definieron 2 tipos de relaciones: directas a otra unidad, o con un filtro.

Esto aborda todos los casos observados en los proyectos inmobiliarios durante este trabajo. Por ejemplo, un departamento podría tener 2 relaciones directas, uno a un estacionamiento y otro a una bodega. O también, podría existir 2 relaciones que permitan elegir un número máximo de estacionamientos, y una bodega cualquiera opcional.

Si bien estas relaciones ayudan a enforzar las restricciones del plan de venta de los proyectos, se consideró que estas son referencias iniciales para los clientes, y guías y defectos para los vendedores, y no restricciones requeridas en los procesos del sistema.

Por lo tanto, se decidió inicialmente solo verificar las restricciones de la unidad escogida, y no de las relacionadas, para evitar complejidad. Además, se considera que el inventario es estático y solo cambia de estado y precio, por lo que ninguna relación podría quedar en un estado indefinido.

4.3. Desarrollo realizado

Previo a la implementación misma de la base de datos, se estudió que información poseían las inmobiliarias de su inventarios. Los 4 clientes utilizaban distintas estrategias para organizarlo, así que se vió que información era común y que información es clave para los procesos de la venta.

A pesar de que algunos clientes usaban sistemas especializados de gestión de inventario inmobiliario, todos usaban realmente una hoja de cálculo para llevar el inventario. Así que para iterar sobre la solución y proveer mejor experiencia de *onboarding*, se implementó un sistema que valide y cargue los datos desde una hoja tabular.

Para eso, se buscó una librería de validación que permita cargar datos tabulares y evite fallar por filas inválidas. No se encontró, así que se decidió extender una. Entre las librerías populares, está Valibot, que fue creado con una arquitectura modular (Hiller, 2023) que no poseen otras librerías. Con esta librería, se creó 2 esquemas, con el mismo formato que las librerías *build-in*:

`arrayDropInvalid` Obtiene solo los valores validos de una lista.

`namedTuple` Permite cargar una fila de una tabla con los nombres de columnas dados.

Estos en conjunto pueden ser compuestos para validar datos de la siguiente forma:

```ts
import * as v from "extended-valibot";
const schema = v.arrayDropInvalid(
  v.namedTuple([
    ["state", v.string()],
    ["price", v.number()],
  ]),
);
const units = v.validate(schema, [["Vendido", 6000], ["Disponible", 7000], [null, 13000]]);
console.log(units); // [["Vendido", 6000], ["Disponible", 7000]]
```

Si bien esto fue realizado para la carga a la base de datos, la validación es un bloque independiente que puede ser usado en diferentes contextos. Esto fue utilizado en la integración de datos directo desde Google Sheets, como se comenta más adelante.

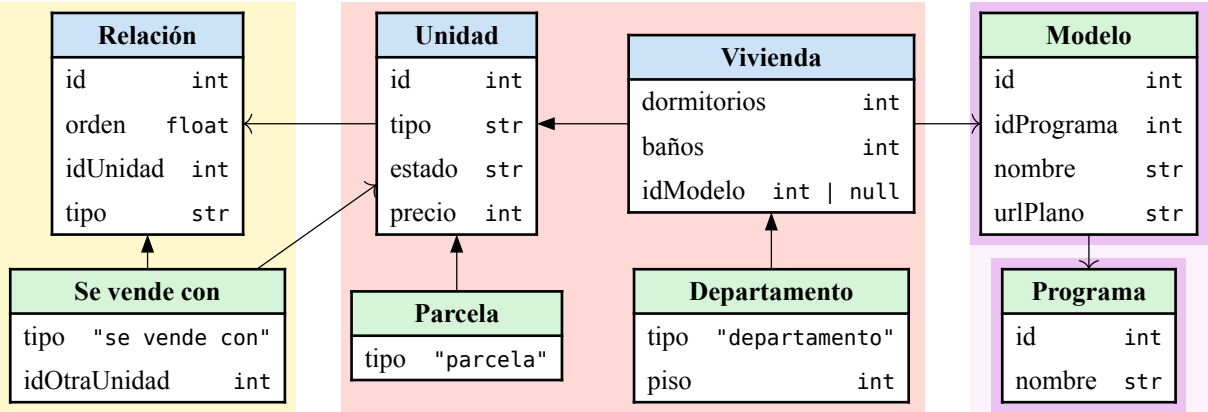

Figura 5: Simplificación del esquema elaborado para la gestión de inventario

Para las tablas de relaciones y las unidades, se implementaron esquemas abstractos que son heredados por otros o por esquemas concretos, que poseen un nombre único dentro de su tabla, y posee atributos específicos. Los modelos y programas son tablas asociadas solamente a viviendas.

```ts
const _unit = v.object({
  type: v.string(),
  state: v.string(),
  price: v.number(),
})
const _residence = _unit.extend({
  bedrooms: v.number(),
  bathrooms: v.number(),
  modelId:
  v.optional(v.id("model")),
})
const apartment =
_residence.extend({
  type: v.literal("apartment"),
  floor: v.number(),
})
const plot = unit.extend({
  type: v.literal("plot"),
})
```

```ts
const model = v.object({
  programId: v.id("program"),
  name: v.string(),
  planUrl: v.string(),
})
const program = v.object({
  name: v.string(),
})
const _relation = v.object({
  order: v.number(),
  unitId: v.id("unit"),
  type: v.string(),
})
const soldWith =
_relation.extend({
  type: v.literal("soldWith"),
  otherUnitId: v.id("unit"),
})
```

```ts
export default defineSchema({
  units: defineTable(v.union(apartment, plot)),
  relations: defineTable(v.union(soldWith)),
  models: defineTable(model), programs: defineTable(program),
})
```

Código 2: Simplificación de la implementación de los esquemas en Convex

5. Sistema de páginas white-label

5.1. Problema a resolver

Uno de los productos que ofrece MOME es la creación de páginas web para los proyectos inmobiliarios. Las páginas, además de proveer información de los proyectos, buscan ser un canal de venta, entregando precios actualizados y disponibilidad de las unidades, en flujos de cotización optimizados para el tipo de proyecto.

Si bien el objetivo es entregar herramientas *white-label* que puedan ser auto-gestionadas por los mismos clientes para la creación de estas, esto no es posible dentro de un desarrollo ágil, especialmente considerando que inicialmente se contaban con 4 clientes con 1 proyecto cada uno, donde el cambio de precios y disponibilidad no muy frecuente para justificar el desarrollo.

5.2. Diseño de la solución

Se armó un plan a largo plazo de desarrollo de estas páginas, considerando un desarrollo ágil y reutilización de los componentes. Inicialmente, cada página será desarrollada de cero, luego, se armará un kit de flujos reutilizable, y finalmente, se buscará armar un sistema autogestionable.

Para que se pueda avanzar con ese plan con la mayor reutilización de código, es esencial poseer tecnologías que permitan tanto la rápida iteración como el uso en diferentes contextos, desde páginas fijas hasta creadas dinámicamente. Se determinó 3 tipos de tecnologías claves: el sistema de componentes, el sistema de diseño, y la herramienta de revisión continua.

5.2.1. Sistema de componentes

Se decidió usar React. Esta es la librería de componentes más usada del mercado, con un gran ecosistema (Devographics, 2024). Además, sus últimos avances, React Server Components (RSC), permitirá en el futuro incluir código condicionalmente en el navegador, evitando la carga de componentes que no se usan en la página cuando este sea autogestionable (React Team, 2020).

Una ventaja de React es que posee múltiples librerías de UI sin estilos mantenidas por empresas establecidas, como React Aria (Adobe) y Base UI (MUI), que permiten fácilmente crear componentes con buena accesibilidad y UX, agilizando la creación de componentes *white-label*.

RSC provee una arquitectura distinta a las páginas web tradicionales, que permite seguir el principio de composición tanto en la obtención de datos como en las vistas de las páginas.

- Las arquitecturas *client-first* suelen ser *single-page applications* (SPA) renderizadas y optimizadas en el servidor. Estas tienen 2 limitaciones:
  - La carga de datos es lejano a donde se usan. Por ejemplo, SvelteKit y NextJS (Pages) emplean una función que carga los datos de toda la página. Cuando solo una sección requiere cierto tipo de datos, la página completa se acopla a esta. Esto pasa en la sección de cotización.
  - Todos los componentes que pueden ser usados por la página son cargados en el navegador. En la etapa de autogestión, una página solo ocuparía un subconjunto de las secciones disponibles, y con esta limitación, tendría que cargar todas, aumentando el tiempo de carga.
- RSC es una arquitectura *backend-first*, pero se diferencia de servidores tradicionales dado que:
  - Permite cargar componentes interactivos de forma declarativa y con composición. Otras alternativa es Astro, que también solo envía el código necesario al navegador.
  - Frameworks con RSC permiten el envío de fragmentos de forma más declarativa y con composición. Frameworks tradicionales recurren a herramientas más imperativos, como HTMX para enviar partes de la página, en vez de re-cargar la página completa en la navegación.

En el periodo del trabajo de título, NextJS es el único framework con soporte completo de RSC, y además, la compañía detrás de NextJS, Vercel, tiene empleados trabajando en el desarrollo de React. Por lo tanto, se decidió utilizar NextJS para el desarrollo de las páginas *white-label*.

5.2.2. Sistema de diseño

En el sistema de estilos, se buscó un sistema que permita iterar rápidamente y crear un design system fácilmente, sin sacrificar la mantención de los estilos. Estos requisitos se cumplen en un sistema que permite la colocación, es decir, que los estilos estén junto a donde están aplicados. Es importante notar que este sistema además debe ser compatible con el resto y fácil de migrar.

Hay 2 tipo de soluciones de colocación: estilos en linea y clases atómicas. Sobre estilos en línea, hay librerías como StyleX que han permitido empresas como Instagram escalar sus estilos, pero con el costo de una API restringida. En cambio, clases atómicas como Tailwind permiten una mayor agilidad y flexibilidad, con el costo de tener que enforzar buenas practicas manualmente.

Dado que la prioridad es la agilidad y flexibilidad, y que se busca un equipo de desarrollo pequeño en los primeros años, se decidió utilizar Tailwind. Se estableció tokens de diseño con variables de CSS para colores primarios y base, anchos de secciones, tamaño y familia de fuente, entre otros, que pueden ser consumidos rápidamente a traves de las clases auto-generadas.

5.2.3. Revisión continua

Finalmente, para la revisión continua de los componentes, uno de los requisitos que establecí fue que el equipo de diseño pueda revisar el sistema de diseño tal como el equipo de desarrollo revisa cambios en el código. Esto es, para lograr un desarrollo ágil con la menor cantidad de errores posibles, reduciendo la brecha entre el diseño y la implementación.

La herramienta que se decidió utilizar fue Storybook, que permite documentar los componentes de forma atómica, y además, la revisión constante a través del servicio de Chromatic. Dado que Storybook es open source, y Chromatic ofrece un plan gratuito, nos permite experimentar con la herramienta sin costos. Además, incluye herramientas de tests visuales y de iteración.

5.3. Desarrollo realizado

En el periodo de el trabajo de título, se realizaron completamente 2 páginas de proyectos inmobiliarios, uno de departamentos y otro de parcelas. Ambos fueron trabajados en conjunto con la encargada de UX, se ordenaron las tareas concretas en el sistema de seguimiento, y se estableció un plazo de 2 semanas para la entrega de cada una. Partieron en momentos distintos sin solaparse.

Se configuró en cada uno el hosting tanto para la página misma como para la documentación de componentes con Storybook, utilizando Cloudflare. Se configuró NextJS para que funcione en Cloudflare. Se realizó la configuración inicial de los estilos y librerías de componentes sin estilos.

Para los estilos, se utilizó específicamente Tailwind v4, que se integra directamente con variables de CSS para crear un sistema de estilos a partir de lo especificado en los tokens de diseño.

```css
@theme {
  --color-base-500: #71717a;
  --breakpoint-lg: 1080px;
}
```

```html
<div class="text-base-500 lg:text-lg">
  Texto en color base 500 que se
  agranda en pantallas grandes.
</div>
```

Código 3: Ejemplo de uso de tokens de diseño en Tailwind v4

En cada página, se realizaron entregas constantes, para la revisión del resto del equipo. Una vez terminado el desarrolló de las vista, se añadió las funcionalidades para mejorar SEO e integrar con analíticas. En el periodo del trabajo de título, no se alcanzó a realizar estudios de usabilidad.

Para los datos de disponibilidad y precios de las unidades, se decidió empezar con datos estáticos a partir de los datos reales. En la última semana del trabajo de título, se añadió la carga de datos a partir de Google Sheets y el sistema realizado en la carga de inventario.

6. Inteligencia artificial en inbox

6.1. Problema a resolver

Un gran dolor de la industria inmobiliaria es el seguimiento de leads. El éxito de la venta de proyectos están a la merced de los vendedores, lo que lleva a problemas cuando los clientes contactan fuera del horario laboral, o los vendedores no dan correcto seguimiento.

En diferentes industrias, se han desarrollado diferentes automatizaciones que han permitido reducir la dependencia de vendedores humanos con procesos auto-gestionables, como son los sitios y aplicaciones e-commerce. Si bien estos aportan en el contexto inmobiliario, se quedan cortos dado que el proceso de compra inmobiliaria es más complejo y requiere un seguimiento más cercano.

Es por esto que se buscó desarrollar un sistema de inteligencia artificial generativa que permita reducir la dependencia de vendedores, y a la vez entregue un mayor seguimiento y reportabilidad a los gerentes de ventas.

Una característica importante de este sistema es usar medios de comunicación cercanos a los clientes. WhatsApp es el medio más usado en Latinoamérica, y en primeros análisis, fue el medio más preferible por los clientes. Por lo tanto, se buscó una integración a WhatsApp, que permita potenciar el canal existente de WhatsApp, en el mismo número que utilizan los vendedores.

Para la respuestas automáticas con inteligencia artificial, hay 2 requisitos: primero, debe responder naturalmente, por lo que la IA, además de responder con un lenguaje natural adecuado, debe esperar un pequeño tiempo antes de responder, poseer el contexto de los mensajes anteriores. Segundo, deberá poder ser interrumpido por un vendedor, que tome el control de la conversación.

6.2. Diseño de la solución

WhatsApp se separa en 2, las aplicaciones y API. Conectarse via API requiere un proceso de aprobación opaco, y se pierde el acceso a la aplicación. Por lo tanto, se decidió utilizar un proveedor que pueda enviar mensajes a través de la aplicación, y que permita la integración con la API.

Considerando que el sistema deberá permitir la integración con la API a futuro, se utilizó el patrón plantilla, donde se creó una interfaz que permite la integración via proveedores externos o por API, permitiendo el intercambio a la API en el futuro sin cambiar el sistema.

Los requisitos del sistema de IA hacen este deba estar integrado a una cola de tareas, que permita esperar un *delay*, y cancelar la ejecución si es que el cliente manda algo antes de que la IA responda, o si es que el ejecutivo de ventas toma el control de la conversación.

A esto se le suma la futura inclusión de conexión a sistemas, como por ejemplo, el estado y precios del inventario, y la actualización de información en el CRM, que se obtienen a través de *function calling* a los servicios correspondientes. Esto hace que la tarea de ejecución sea con múltiples pasos y llamadas externas encadenadas.

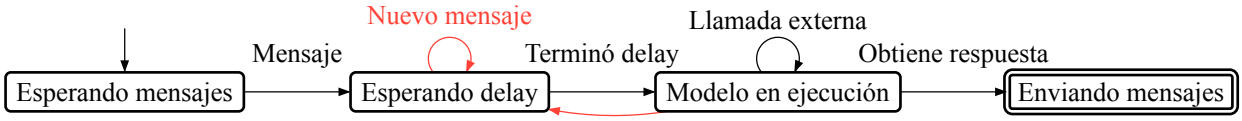

Figura 6: Procesamiento de mensajes en el sistema de IA

Cuando llega un mensaje, se espera un tiempo definido antes de procesar. Si llega un nuevo mensaje, se cancela el procesamiento y se espera el nuevo mensaje, marcado en rojo. Además, si el ejecutivo de ventas toma el control de la conversación, se cancela el procesamiento.

Dado la complejidad del sistema, se buscó realizar un suite de tests, donde se establecieron casos de prueba que establecen el funcionamiento esperado del sistema previo a su implementación completa.

Realizar tests de este tipo involucra simular el avance en el tiempo en la cola de tareas usadas en el entorno de pruebas, que es poblado por rutinas internas. Convex posee una libraría para testear con Vitest, permitiendo simular el avance en el tiempo. Esta característica fue parte de lo que se revisó al elegir la arquitectura.

6.3. Desarrollo realizado

Para la generación de texto en lenguaje natural, se utilizó OpenAI. Esto es dado que provee una API para la creación de conversaciones sin preocuparse del guardado del estado, es muy rápido de configurar en comparación a otras soluciones, y provee *structured-output*, sistema que permite garantizar el formato estructurado deseado en modelos generativos (Liu et al., 2024).

Inicialmente, y previo al estudio de la arquitectura, se desarrolló una versión inicial en Firebase, que fue usada para el primer cliente. Cada mensajes iniciaba el procesamiento en una rutina, y se usaba la base de datos Cloud Firestore, para coordinar la ejecución. Si bien, funciono en gran parte para el primer cliente, el flujo requería código complejo.

Esto es dado al requerimiento permitir la cancelación de la rutina en cualquier momento. Sistemas de tareas, como Cloud Tasks, no proveen los mecanismos requeridos para la cancelación de tareas, se debe implementar manualmente. Como se mencionó anteriormente, Convex se eligió por tener un primitivo de tareas fácil de usar, que a futuro podría ser empleado con Workflows.

Se configuró los tests, añadiendo los mocks para las llamadas externas a OpenAI, WhatsApp, y a las notificaciones, y la simulación del tiempo. Se crearon tests para revisar que:

- Mandar varios mensajes en un corto periodo de tiempo solo hace una ejecución de la IA.
- Cuando llegan mensajes, y la IA responde con una alerta, se notifica al ejecutivo de ventas.
- No se llama la IA cuando llegan mensajes cuando el chat tiene la IA desactivada.
- La IA indica que debe parar y se refleja el cambio en la base de datos.
- El sistema no manda mensajes si el modelo partió la ejecución y se apaga la IA.

Esto en el código, fue implementado en la siguiente forma:

```ts
test("Multiple messages only triggers one AI execution", async () => {
  const t = createConvextTestClient();
  await t.mutation(internal.whatsapp.saveMessage, createSampleMessage());
  await t.mutation(internal.whatsapp.saveMessage, createSampleMessage());
  await t.finishAllScheduledFunctions(vi.runAllTimers);  // Se espera que terminen las tareas
  expect(runAI).toHaveBeenCalledTimes(1);  // Función externa con mock
});
```

Código 4: Ejemplo de test de Convex con Vitest para el sistema de IA

7. Otros desarrollos realizados

Otro desarrollo destacable en el periodo del trabajo de título fue el refractor parcial del el dashboard donde se realizaron las pruebas para la gestión de inventario, por 2 problemas. Primero, una abstracción creada para evitar código, resultó en código más complejo. Segundo, la manera que se definía las rutas, causaba que no se reflejaran cambios en el modo de desarrollo (*hot-reloading*).

Cambiar esto de forma manual tomaría múltiples horas, usar métodos como expresiones regulares no es suficiente, y usar modelos de IA generativa corre el riesgo de cambiar código que no debería ser cambiado. Yo no conocía en ese momento herramientas que pudieran realizar este trabajo, pero si sabia que existían y como se debería transformar el código.

Es por esto que se usó modelos generativos para ayudar a generar código complejo para la refactorización. Se fue iterando con un modelo generativo, iterando el código complejo de transformaciones de *abstract syntax trees* con `jscodeshift`, hasta lograr lo deseado, en un menos de 1 hora, cambiando sobre de 200 lineas de código. Se puede ver el código final en el Anexo 4..

8. Conclusiones

Si bien no se realizó una gran cantidad de desarrollo, en este trabajo de título se logró explorar, validar y diseñar diversas soluciones para dolores de la industria inmobiliaria. Se exploró problemas de arquitectura, modelo de datos, diseño de sistemas web y de inteligencia artificial.

En las secciones de cada problema, se demostró como se entendieron, empleando conocimientos de computación y del contexto inmobiliario, para identificar sus requerimientos claves. Siguiendo diferentes estándares de la ingeniería de software, como SOLID y Agile, se diseñó sistemas complejos que abordaban los requerimientos identificados. Y finalmente, se realizó una planificación del desarrollo e iteración inicial de estos sistemas, validando soluciones y tecnologías.

No se pudo observar resultados concretos de todas las soluciones llevadas a cabo, dado la temprana etapa de la empresa y que se no se inició el trabajo de ventas de la mayoría de los clientes en el periodo del trabajo de título. Sin embargo, este trabajo realizado establece una base sólida para el desarrollo futuro de la empresa, dado las estrategias y tecnologías establecidas.

Bibliografía

Devographics. (2024, diciembre 16). *State of JavaScript 2024*. https://share.stateofjs.com/share/prerendered?localeId=en-US&surveyId=state_of_js&editionId=js2024&blockId=tools_arrows&params=&sectionId=libraries

Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, *15*(4), 287-317. https://doi.org/10.1145/289.291

Hiller, F. (2023). *Implementation of a modular schema library in TypeScript with focus on bundle size and performance*. https://valibot.dev/thesis.pdf

Ley de copropiedad inmobiliaria (2022). https://www.bcn.cl/leychile/navegar?idNorma=1174663&idParte=10326609

Liu, M. X., Liu, F., Fiannaca, A. J., Koo, T., Dixon, L., Terry, M., & Cai, C. J. (2024). *"We need Structured Output": Towards User-centered Constraints on Large Language Model Output* (pp. 1-9). ACM. https://arxiv.org/pdf/2404.07362

Manifesto for Agile Software Development. (2001). http://agilemanifesto.org/

Martin, R. C. (2000). *Design Principles and Design Patterns (SOLID)* (pp. 4-16). https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start

Meyer, B. (1997). *Object Oriented Software Construction* (2nd ed., pp. 751-757).

React Team. (2020, diciembre 21). *React Server Components RFC*. https://github.com/reactjs/rfcs/blob/main/text/0188-server-components.md

The Scrum Guide. (2020). https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Anexos

1. Terminología inmobiliaria

Programa Normalmente, se refiere a la combinación de dormitorios y baños que posee una vivienda. Por ejemplo, un programa de 3D+2B se refiere a una vivienda con tres dormitorios y dos baños. Hay casos especiales que no siguen esta regla, por ejemplo, el programa STUDIO, que se consideran distinto a 1D+1B dado que no tiene un espacio dedicado a dormitorio.

Modelo Una variante de un programa, es decir, una agrupación de unidades que se venden bajo el mismo plano de planta. Por ejemplo, un modelo de 3D+2B puede tener modelo A y B, donde el A tiene los 3 dormitorios juntos, mientras que le B tiene dormitorio de visita separado. Se suele llamar también como tipología.

Bienes primarios y secundarios Separación realizada en proyectos inmobiliarios donde se busca que los bienes secundarios se vendan asociados a los primarios. Estacionamiento y bodegas suelen ser bienes secundarios, mientras que viviendas y oficinas son bienes primarios.

2. Proceso de desarrollo

Entender el problema Identificar los requerimientos y sus prioridades, entendiendo el dolor a resolver y cual es el estado del arte, desde lo tecnológico a lo existente en el mercado.

Diseñar la solución Definir como se modelará la solución, determinar que tecnologías son las más adecuadas, y cuales son los componentes claves de la solución.

Llevar el desarrollo Planificar como sera llevado a cabo el desarrollo de la solución en un tiempo establecido, entendiendo un desarrollo progresivo y ágil.

Implementar la solución Programar la solución, armando un producto funcional.

3. Comparativa de Backend as a Service

Funcionalidad		Backend as a Service (BaaS)			
Categoría	Nombre	Supabase	Firebase	Cloudflare DO[1]	Convex
Descipción	Creación servicio	Firestore (2017)	Supabase (2020)	Durable Objects (2021)	Convex (2021)
	Lenguaje	SQL	JS	JS/TS, estándares web	JS/TS, estándares web
	BDD	PostgreSQL	Firestore, no ralacional	SQLite	Documentos, relacional
Funcionalidades	Real Time	✓	✓	WebSockets diracto	✓
	Tareas	Basado en SQL	Google Tasks	Impl. propia	✓
	Subir imágenes	✓	✓	R2	✓
	Integración Autentificación	✓	✓	x	✓
	Permisos[2]	RLP (polizas en la SQL)	Security Rules	Impl. Propia	Impl. Propia
Agilizantes de desarrollo	Servidor local	Complejo	Distinto produción	Levemente diferente	✓
	Dashboard	✓	Complejo de usar	x	✓
	Buena documentación	✓	~	~	✓
	Integración cliente	✓	Deprecada	Implementación propia	✓
	Testing	Basado en SQL	Deprecada	Complejo	Bien integrado
Nice to have	Sin cold start	✓	x	✓	✓
	Deploy automatico	✓	✓	✓	✓
	Previews	✓	✓	✓	✓
	Self hosteable	✓	x	✓	✓
Escalabilidad	Fuerza buenas prácticas	Accesso directo a BDD	Accesso directo a BDD	Libre	✓
	Conexiones Cloud	Por SQL	Google Cloud	Servicios de Cloudflare	Cloud aparte
Costos	Plan gratis	Sí	Sí, por uso	No	Sí (2 personas)
	Siguiente plan	$25/mes	Solo basado en uso	$5/mes	$25/mes/persona

Tabla 2: Compartiva de Backend as a Service.

[1] DO es un primitivo para contruir un backend, y no entrega todas las heramientas comunes de un BaaS

[2] Que sistema de permisos se asume en las funcionalidades que otorga el Backend as a Service

4. Refractor de código con jscodeshift

```ts
export default function (fileInfo, api) {
  const j = api.jscodeshift, root = j(fileInfo.source);
  // Step 1: Identify all helper functions
  const helperFunctions = {};
  root.find(j.FunctionDeclaration).forEach((path) => {
    const functionName = path.node.id.name;
    const returnStatement = j(path).find(j.ReturnStatement);
    if (returnStatement.size() !== 1) return;
    const argument = returnStatement.get(0)?.value?.argument;
    if (argument && argument.type === "CallExpression" && argument.callee.name === "convexQuery") {
      helperFunctions[functionName] = returnStatement.value.argument;
    }
  });
  // Step 2: Replace all usages of the helper functions
  Object.keys(helperFunctions).forEach((helperName) => {
    const inlineEquivalent = helperFunctions[helperName];
    root
      .find(j.CallExpression, { callee: { name: helperName } })
      .replaceWith((path) => {
        const args = path.value.arguments;
        const newCall = structuredClone(inlineEquivalent);
        newCall.arguments[1] = args[0]; // Replace "args" in convexQuery with actual arguments
        return newCall;
      });
  });
  // Step 3: Remove unused helper functions
  Object.keys(helperFunctions).forEach((helperName) => {
    const isHelperUsed = root.find(j.Identifier, { name: helperName }).size() > 1;
    if (!isHelperUsed) root.find(j.FunctionDeclaration, { id: { name: helperName } }).remove();
  });
```

```
  return root.toSource();
}
```

Código de entrada:

```ts
export function getMessagesQuery(args: FunctionArgs<typeof api.messages.list>) {
  return convexQuery(api.messages.list, args);
}

export const Route = createFileRoute("...")({
  loader: async ({ params: { inboxId } }) => {
    await client.ensureQueryData(getMessagesQuery({ inboxId }));
  },
  component: RouteComponent,
});

function RouteComponent() {
  const { inboxId } = Route.useParams();
  const { data } = useSuspenseQuery(getMessagesQuery({ inboxId }));
  // ...
}
```

Código de salida:

```ts
export const Route = createFileRoute("...")({
  loader: async ({ params: { inboxId } }) => {
    await client.ensureQueryData(convexQuery(api.messages.list, { inboxId }));
  },
  component: RouteComponent,
});
```

```
function RouteComponent() {
  const { inboxId } = Route.useParams();
  const { data } = useSuspenseQuery(convexQuery(api.messages.list, { inboxId }));
  // ...
}
```