Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Integrando la documentación técnica de las APIs en el ciclo de vida del desarrollo de software

Integrating technical documentation of APIs into the software development lifecycle

Juan José Barroso Giménez

La Laguna, 7 de julio de 2023

D. **Casiano Rodríguez León**, con N.I.F. 42020072S profesor Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

I N F O R M A (N)

Que la presente memoria titulada:

"Integrando la documentación técnica de las APIs en el ciclo de vida del desarrollo de software"

Ha sido realizada bajo su dirección por D. **Juan José Barroso Giménez**, con N.I.F. 54947335J.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de julio de 2023

Agradecimientos

Y después de todo este tiempo, ¡por fin llegó la hora de agradecer!

A Casiano Rodríguez, mi tutor universitario, le agradezco profundamente por mostrarme que el entorno universitario puede ser divertido e interesante cuando se está apasionado por lo que se hace. Casiano, gracias por la paciencia y por todas las enseñanzas que me brindaste durante esta etapa de mi vida.

A Juan José Coello mi tutor en Salsa, porque siempre me dio consejos que funcionaban y porque fue un gran tutor en este proyecto. Juanjo, gracias por comprometerte también en esta meta y ayudarme a sacarla adelante.

A Juan Manuel Barroso, porque sin toda su ayuda quizás no habría podido alcanzar esta meta. Juanma, gracias por todos los consejos y todas las veces que me animaste a seguir adelante con este proyecto. ¡Al final sí se pudo!

A Daniela Villegas porque me ayudó a encontrar motivación en los momentos más complicados y porque me enseñó el valor del trabajo duro. Dani gracias por nunca dejar de creer que si podía lograrlo.

A Salsa, por ser esa startup increíble que me motivó a aprender y trabajar con alegría. Salsa, thank you so much! Son personas increíbles y con una cultura admirable.

A mis padres y amigos que nunca dejaron de animarme, aunque todos los años les decía lo mismo “Este año sí que termino el máster“ siempre me respondían “dale, termínalo ya“.

A todos los profesores que me impartieron clases, quiero agradecerles por su paciencia y por siempre buscar la mejor manera de ayudarnos a aprender.

¡Gracias Totales!

Licencia

Resumen

Las APIs son ventajosas al facilitar la transición de aplicaciones monolíticas a microservicios en la nube, en respuesta a la necesidad de agilidad y generación de valor. En la "economía de APIs", las empresas son consumidores y productores de APIs, que actúan como componentes esenciales simplificando el desarrollo de productos y servicios. Este enfoque brinda flexibilidad, eficiencia y una ventaja competitiva en el mercado actual.

En ese proceso la documentación técnica desempeña un papel fundamental en el éxito de las empresas, ya que constituye el primer punto de contacto con los desarrolladores. La retroalimentación constante de los consumidores y la mejora continua de la documentación aumentan significativamente las probabilidades de éxito de las empresas, por lo que debe ser dinámica y adaptarse a las exigencias de un mercado demandante. Para lograrlo, es esencial integrarla en el ciclo de vida del desarrollo de software.

En este estudio, se investigan y establecen pautas necesarias para que las empresas puedan publicar una documentación sólida, basada en una estructura moderna e integrada en el proceso de desarrollo. Se utilizan técnicas como la evaluación comparativa para recomendar herramientas tecnológicas apropiadas según el escenario y los requisitos de la empresa.

Se consideran factores como el tiempo, los recursos disponibles y la calidad esperada para establecer un conjunto de criterios que posteriormente se evalúan en una muestra. Los resultados obtenidos destacan la importancia de identificar la tipología y responsabilidad de cada documento, ya que establecer estas responsabilidades garantiza la solidez de los resultados. Se establece una forma y narrativa efectiva para presentar los beneficios del producto tecnológico.

Además, se concluye que la generación automática de la especificación del API es un componente clave para alcanzar los objetivos planteados en este trabajo. Por último, se establece que la calidad del producto en sí mismo no es suficiente para garantizar el éxito; factores como una mala documentación pueden perjudicar directamente el desempeño de la empresa.

Keywords: API economy, agilidad, documentación, producto, desarrolladores de software, OpenAPI, guías de usuario, tutoriales, REST API, infraestructura digital, proveedores de infraestructura, ciclo de vida del desarrollo de software, objetivos.

Abstract

APIs have multiple advantages, and one of the most prominent is their ability to facilitate the transition from monolithic applications to cloud-based microservices. This transition has emerged as a response to the growing need for agility and value generation. As a result, the concept of 'API economy' has emerged, where companies play two fundamental roles: consumers and producers of APIs. In this new approach, APIs act as essential building blocks or components that simplify the process of developing and building products and services. This paradigm enables greater flexibility and efficiency in the development of technological solutions, providing companies with a competitive advantage in the current market.

Crucially, within this landscape, meticulous technical documentation assumes a pivotal role in determining the success of companies, serving as the initial touchpoint for developers. The continuous feedback loop between consumers and the relentless enhancement of documentation substantially augment the probability of success for enterprises. Thus, it becomes imperative for documentation to possess a dynamic and adaptable nature, ensuring alignment with the demands of an exacting market. Integration into the software development lifecycle stands as an indispensable prerequisite.

The present study delves into the comprehensive investigation and establishment of vital guidelines that enable companies to publish robust documentation, firmly rooted in a contemporary and seamlessly integrated structure within the development process. Leveraging techniques such as comparative evaluation, appropriate technological tools are recommended, tailored to specific company scenarios and requirements.

Multiple factors, encompassing time, resource availability, and expected quality, are conscientiously considered establishing a set of criteria that is subsequently evaluated across a representative sample. The findings distinctly underscore the paramount importance of identifying the typology and responsibility of each document, as meticulous delineation of these responsibilities guarantees the solidity of outcomes. Moreover, a compelling form and narrative are thoughtfully constructed to effectively communicate the benefits of the technological product.

Furthermore, a key conclusion derived from this research asserts that the automated generation of API specifications, exemplified by OpenAPI, constitutes a pivotal element in successfully attaining the objectives. It is firmly established that product quality alone does not suffice to ensure triumph; rather, factors such as subpar documentation directly impede a company's performance.

Keywords: API economy, agility, documentation, product, software developers, OpenAPI, user guides, tutorials, REST API, digital infrastructure, infrastructure providers, software development lifecycle, objectives.

Índice general

Índice de Figuras

Índice de Tablas

Abreviaturas y Acrónimos

TFM Trabajo de fin de máster

HTTP Hypertext Transfer Protocol

API Application Programming Interface

APIs Application Programming Interfaces

OAS OpenAPI Specification

SaaS Software as a Service

AWS Amazon Web Services

SAT Servicio de Atención técnica

SLA Service Level Agreement

SDCL Ciclo de vida del desarrollo de software

HTML Hypertext Markup Language

SEO Search Engine Optimization

CTO Chief Technology Officer

CEO Chief Executive Officer

SOAP Simple Object Access Protocol

REST Representational State Transfer

POC Proof of Concept

RAML RESTful API Modeling Language

Capítulo 1

Introducción

1.1. Contexto y justificación

Los beneficios de las APIs son numerosos, pero podemos resumir que las APIs nos permiten evolucionar desde aplicaciones monolíticas (on-premises) a micro-servicios en la nube. Esta evolución, combinada con la necesidad de crear valor para los usuarios con rapidez y agilidad, ha dado lugar al concepto de "*API economy*", en el que las empresas asumen dos roles: consumidores y productores. Empresas que buscan innovar creando productos y servicios que minimicen recursos y tiempo, usan las API como bloques de construcción o componentes para simplificar el proceso de construcción.

El deseo de acelerar la construcción de nuevos productos con base tecnológica genera una oportunidad de monetizar las APIs, lo que da lugar a un nuevo tipo de cliente: los desarrolladores de software. Ejemplos de esta nueva generación de productos que se están convirtiendo en infraestructura digital son Stripe[18], Plaid[10] y Marqeta[7].

Solo las empresas que priorizan la experiencia del usuario destacan en un mercado competitivo, y para las empresas de infraestructura digital mencionadas la documentación forma parte de esa experiencia de usuario. La documentación no es un artefacto secundario, es un artefacto crucial que permite a un desarrollador evaluar el producto antes de integrarse.

La calidad de esta documentación es de suma importancia en la elección de proveedores de infraestructura porque los desarrolladores valoran la claridad y facilidad de uso. Una documentación bien elaborada facilita la integración con el producto, genera confianza y fortalece la posición de la empresa en el mercado.

La documentación técnica se ha convertido en una parte esencial en el ámbito de la ingeniería, ya que juega un papel fundamental tanto en el desarrollo como en la implementación de productos tecnológicos. Y por eso es importante integrar la generación y publicación de la documentación del producto en el ciclo de vida del desarrollo de software, porque no es un artefacto estático que no cambia en el tiempo; sino todo lo contrario la documentación técnica debe considerar las técnicas modernas del desarrollo de software, como la revisión de código, el seguimiento de cambios, el versionado y la entrega continua. Estos aspectos garantizan una experiencia satisfactoria para proveedores y consumidores de productos tecnológicos.

A partir de esta idea, podemos identificar la necesidad de mejorar las deficiencias y fortalecer las buenas prácticas en la documentación técnica de las empresas que ofrecen APIs como su producto principal. Ya que en algunos casos ni se cuenta con un portal

para desarrolladores fácil de utilizar, ni un área de pruebas para que un desarrollador pueda probar los recursos expuestos por la API directamente desde un navegador. Todos estos aspectos convierten a la documentación en un factor negativo a la hora de probar o integrar el producto.

1.2. Objetivos

1.2.1. Objetivo principal

El objetivo principal de este trabajo es investigar y establecer las pautas necesarias para integrar la documentación técnica en el ciclo de vida del desarrollo de software (SDCL) en startups tecnológicas que tienen como su producto principal APIs y sus clientes objetivos son desarrolladores (*API Economy startups*). Con estas pautas podrán elegir de manera eficiente las herramientas y/o tecnologías más adecuadas para publicar la documentación técnica de sus APIs, así como del mismo modo podrán establecer una estructura innovadora en la información publicada, mejorando la comunicación y la comprensión de sus APIs. Fomentando una mayor adopción y satisfacción entre los desarrolladores. El marco de la investigación está diseñado para ayudar a estas empresas a prevenir retrasos y simplificar los procesos de desarrollo del software, aumentando así la eficiencia, la productividad y la capacidad de mantenimiento a largo plazo. Además, también se considerarán otros aspectos relevantes, como la seguridad y la escalabilidad, para brindar una solución integral adaptada a las necesidades específicas.

1.2.2. Objetivos específicos

A nivel funcional, pretende alcanzar los siguientes objetivos:

1. Proporcionar una guía para evaluar las necesidades y requisitos de las startups para publicar su documentación efectivamente.
2. Identificar y analizar las diferentes herramientas y tecnologías disponibles en el mercado para la publicación de documentación.
3. Desarrollar criterios de evaluación que permitan comparar y seleccionar las herramientas o tecnologías más adecuadas según las necesidades de la empresa.
4. Establecer las diferencias y tipologías de la documentación técnica.
5. Establecer pautas y recomendaciones para las directrices de la API REST.
6. Evaluar la viabilidad y eficacia de las propuestas y casos de estudio, demostrando su utilidad y beneficios en de forma práctica.

La consecución de estos objetivos se considerará positiva e innovadora, ya que contribuirá a mejorar la eficiencia y la toma de decisiones de la empresa, aportando un enfoque más metódico basado en la selección de herramientas y técnicas adecuadas.

1.3. Hipótesis, tema y preguntas de investigación

La hipótesis plantea que al implementar los hallazgos derivados de este trabajo de fin de máster en el ciclo de vida del software de la documentación técnica, se lograrán significativamente mayores posibilidades de generar una documentación de alta calidad, mejorar la experiencia de integración de nuevos clientes, evolucionar las API de acuerdo con una estructura moderna y adoptar las herramientas y tecnologías apropiadas en función de los objetivos del proyecto.

Adicionalmente, se plantea que las pautas establecidas en este trabajo desempeñan un papel fundamental en la implementación de las soluciones que se presentarán como resultados posteriormente. Estas pautas serán evaluadas para comprobar su idoneidad en la mejora del rendimiento y la comprensión del producto tanto para los desarrolladores involucrados en su creación como para los consumidores de la documentación.

La investigación se enfocará en proponer un marco para categorizar los distintos tipos de documentación, así como directrices para mejorar el estilo de escritura y la estructura del contenido. Además, se evaluarán y propondrán herramientas y tecnologías que aporten un mayor valor al proceso de desarrollo de la documentación técnica. Se tomarán en consideración los desafíos a los que se enfrentan las startups tecnológicas en la actualidad, siempre teniendo en cuenta las mejores prácticas en el desarrollo de software, como la revisión de código, el seguimiento de cambios y versiones, la entrega continua y los patrones de diseño, entre otros.

Dentro de los temas de investigación, surge el interés por abordar las siguientes interrogantes:

- ¿Cuál es la clave para generar una documentación de calidad?
- ¿Existen soluciones en el mercado que aborden las deficiencias identificadas en la documentación técnica?
- ¿Cuáles son los factores más importantes que pueden agilizar el proceso de creación de la documentación técnica?
- ¿Cómo podemos lograr los objetivos de mejora en un plazo de 3 meses en lugar de 6?
- ¿Cuáles son las mejores prácticas para mantener actualizada y relevante la documentación técnica a lo largo del ciclo de vida del producto?
- ¿Qué estrategias se pueden implementar para asegurar la colaboración efectiva entre los equipos de desarrollo y los escritores técnicos en el proceso de creación de la documentación técnica?

1.4. Estructura de esta memoria

El resto de este trabajo de fin de máster está organizado de la siguiente forma: En el capítulo dos se tratan los antecedentes, el estado del arte y los conceptos teóricos. En el capítulo tres se habla de metodologías, diseño de investigación, población, muestra e instrumentos y técnicas de investigación. Los capítulos cuatro y cinco contienen los

resultados, discusiones y recomendaciones. Finalmente, los capítulos seis y siete establecen las conclusiones. Se han añadido en los apéndices la encuesta de requerimientos, las Salsa REST API Guidelines, capturas de las pruebas de concepto, la especificación OpenAPI y capturas del Sistema.

Capítulo 2

Marco teórico

2.1. Antecedentes y estado actual del arte

En el contexto actual, las API (Interfaces de Programación de Aplicaciones) han adquirido una importancia creciente en el desarrollo de software, sin embargo, a medida que el uso de API se ha vuelto más extendido, surge la necesidad de proporcionar una documentación efectiva que respalde a los desarrolladores en su trabajo. Existen varios libros y estudios publicados a lo largo de estos últimos años, pero en esta sección tomaremos en cuenta un par de estudios por su íntima afinidad con esta investigación.

Un estudio realizado por *Michael Meng, Stephanie M. Steinhardt, Andreas Schubert,* los tres del *Departamento de Estudios Empresariales en Ciencias de la Información, Universidad de Ciencias Aplicadas de Merseburg, Merseburg, Alemania. (Department of Business Studies and Information Sciences, Merseburg University of Applied Sciences, Merseburg, Germany), en octubre del 2020 se llamó *"Optimizing API Documentation: Some Guidelines and Effects"* [30] El estudio "Optimización de la Documentación de API: Algunas Pautas y Efectos" aborda la importancia de optimizar las especificaciones de las APIs y presenta los resultados de una prueba empírica. Se comparó el desempeño de dos grupos de desarrolladores que trabajaron con una API desconocida. Un grupo tuvo acceso a una documentación optimizada siguiendo pautas de diseño de documentación de API, mientras que el otro grupo utilizó una documentación no optimizada. Los resultados revelaron que los desarrolladores que utilizaron la documentación optimizada cometieron menos errores y fueron más rápidos en la ejecución de las tareas. Se concluye que las pautas utilizadas en el estudio tienen un efecto positivo en las interacciones iniciales con una API.

Otro estudio titulado *API Documentation Generator*[29] realizado el año pasado por *Nafis Kamal* en el *Programa de Maestría en Seguridad y Computación en la Nube de la Escuela de Ciencias de la Universidad Aalto* En el estudio .ªPI Documentation Generatorrealizado por Nafis Kamal en la Universidad Aalto, se presenta una técnica única para cumplir con los requisitos de documentación de API. Se evalúan herramientas como Postman, Redocly, SwaggerHub, JavaDoc y AutoREST, y se compara su proceso de documentación con una implementación prototipo. Se realizó un estudio aleatorio para determinar la importancia de los requisitos de documentación de API y simplificar la documentación, enfocándose en las necesidades de los desarrolladores. El estudio identificó dificultades recurrentes que pueden abordarse con la implementación de la documentación sugerida.

En el capítulo 4 del libro “API Management: An Architect’s Guide to Developing and Managing APIs for Your Organization“ (API Management: Una guía para arquitectos sobre el desarrollo y gestión de APIs para su organización.) por “De, Brajesh“ en el 2017 [23]. Habla sobre la importancia de documentar una API REST para su adopción exitosa. Las APIs exponen datos y servicios que los consumidores desean utilizar. Una API debe ser diseñada con una interfaz que el consumidor pueda entender. La documentación de la API es clave para que los desarrolladores de aplicaciones comprendan la API. La documentación debe ayudar al desarrollador a aprender sobre la funcionalidad de la API y permitirles comenzar a usarla fácilmente. Este capítulo examina los aspectos de documentar una API y algunas de las herramientas y tecnologías disponibles para la documentación de API, incluyendo RAML, Swagger, API Blueprint y otras.

En relación con este trabajo, resulta destacable el valor encontrado en la tabla - EST Modeling Capabilities", donde se realiza una comparativa de diversas tecnologías utilizando una técnica de investigación que consiste en evaluar varios criterios en cada columna y analizar los resultados obtenidos para cada tecnología.

Table 4-4. *REST Modeling Capabilities*

Criteria	Swagger 2.0	RAML	API Blueprint
Resources	Supports resource definition	Supports resource definition	Supports resource definition
Nested resources	Supports nested resource definition	Supports nested resource definition	Supports nested resource definition
Representation metadata	Supports the JSON schema	Supports inline and external definitions in any format	Supports only inline definitions in any format
Composition/ inheritance	Inheritance supported by sub types	Supports inheritance of traits and resource types	Supports resource model inheritance
API version metadata	Supported via `apiVersion` tag	Supported via `version` tag	No explicit tag to specify the API version
Authentication	Has tags defined to support Basic, API Key, and OAuth2	Supports Basic, Digest, and OAuth2	Supported via custom header definitions
Methods/action	Supported	Supported	Supported
Query parameters	Supported	Supported	Supported
Path/URL parameters	Supported	Supported	Supported
Header parameters	Supported	Supported	Supported
Documentation	Supported	Supported	Supported

Figura 2.1: Tabla 4-4. “REST Modeling Capabilities“ de “API Documentation“

De estos trabajos podemos reciclar ciertas técnicas como la evaluación comparativa de diferentes herramientas, la observación como instrumento de investigación y la efectividad de buenas prácticas en el ciclo de desarrollo del software de la documentación técnica.

Además de los estudios previamente mencionados, se llevaron a cabo evaluaciones de otras investigaciones relacionadas en el ámbito, sin embargo, se decidió no incluirlas en los antecedentes de este proyecto debido a que su contenido no se alineaba plenamente con los objetivos y alcance de la presente investigación. Entre esos estudios esta *“A study of the effectiveness of usage examples in REST API documentation“* (Un estudio sobre la efectividad de ejemplos de uso en la documentación de API REST.) por *S M Sohan, Frank Maurer, Craig Anslow y Martin P. Robillard* todos de la **Universidad de Calgary, Canadá.** [32].

2.2. Conceptos y teorías clave

Durante el desarrollo del proyecto, se llevó a cabo una evaluación integral de varias tecnologías relacionadas. Con el fin de construir una base sólida para la comprensión de los conceptos que se destacan a medida que avanza la lectura, es necesario mencionar y explicar brevemente algunos de estos conceptos.

Documentación técnica de las APIs:

Se refiere a la información y recursos que describen el funcionamiento, características y uso de una API (Interfaz de Programación de Aplicaciones). Esta documentación tiene como objetivo proporcionar a los desarrolladores y usuarios de la API los detalles necesarios para integrar y utilizar la funcionalidad ofrecida por dicha API de manera efectiva.

Portal de documentación:

Un portal de documentación es un sitio web o plataforma dedicada a proporcionar información y recursos relacionados con la documentación de un producto, servicio o tecnología. Está diseñado para ser una fuente centralizada de documentación técnica, guías de usuario, ejemplos de código y cualquier otra información relevante para los desarrolladores.

API (interfaz de programación de aplicaciones)

Una API (interfaz de programación de aplicaciones)[34] es un conjunto de reglas y protocolos que permiten la comunicación y la interacción entre diferentes sistemas informáticos. Las APIs son la base del desarrollo de aplicaciones modernas porque permiten la integración de sistemas dispares y facilitan la reutilización de código y la creación de aplicaciones más complejas. Las APIs también permite a los desarrolladores crear extensiones y complementos para plataformas existentes, así como acceder a servicios y recursos externos.

Existen diferentes tipos de APIs, pero la más relevante para este proyecto será la API RESTful, que permiten la comunicación a través de internet utilizando protocolos como HTTP.

API REST:

API REST o RESTful es una interfaz que permite que distintas aplicaciones se comuniquen y compartan información de manera estandarizada y uniforme utilizando el protocolo HTTP. Es utilizada en el desarrollo de aplicaciones web y móviles para acceder y manipular datos de forma sencilla. Los desarrolladores a menudo implementan API RESTful mediante el uso de métodos de HTTP, entre los más comunes: GET, POST, PUT, DELETE.

Una vez que comprendamos qué son las APIs y cómo se comunican, es importante comprender cómo se definen, documentan y describen esas APIs. Para hacer esto, necesitamos comprender los conceptos y las pautas en el contexto de la documentación técnica.

Referencia técnica del API:

La Definición o Referencia técnica de las APIs, mejor conocida como *API reference* en inglés. Proporciona una guía completa sobre cómo consumir los recursos expuestos por la API. Es el recurso principal para explicar el funcionamiento, la sintaxis y las características y proporciona información sobre los diversos endpoints, métodos, parámetros y respuestas que componen la API.

Los desarrolladores recurren a esta documentación para aclarar dudas, obtener información sobre la funcionalidad y resolver posibles errores en la comunicación. Esta definición puede estar basada en diferentes estándares, en el caso de este trabajo es importante saber que el estándar usado es OpenAPI.

Especificación OpenAPI:

OpenAPI (OAS)[25] es un estándar abierto que permite describir APIs de manera precisa y estandarizada. Mediante el uso de un documento codificado en JSON o YAML, esta especificación proporciona un diccionario completo de términos y conceptos ampliamente reconocidos en el mundo de las APIs, incluyendo los fundamentos de HTTP y JSON.

API Playground:

Es un entorno de pruebas web donde los desarrolladores pueden enviar solicitudes HTTP a la API, ver las respuestas y experimentar con diferentes parámetros y datos. El API Playground generalmente proporciona una interfaz intuitiva y amigable que permite ingresar datos de prueba, realizar llamadas a la API en tiempo real y observar los resultados.

Guías:

En el ámbito de la documentación técnica, las guías son recursos que brindan orientación y mejores prácticas sobre el dominio que resuelve la API. Estas pautas están diseñadas para ayudar a los desarrolladores a comprender mejor la referencia de la API, entender casos de uso y guiarlo a descubrir todo lo que la API es capaz de hacer. Pueden contener ejemplos, diagramas, imágenes, etc.

Notion:

Notion es una plataforma versátil que permite a los usuarios organizar, colaborar y gestionar información de manera efectiva. Los usuarios pueden crear páginas y estructurar su contenido utilizando bloques, que pueden ser texto, imágenes, listas, tablas, archivos adjuntos, entre otros. Estos bloques se pueden organizar, enlazar y relacionar entre sí, lo que proporciona una gran flexibilidad para crear y conectar información de manera personalizada.

Notion permite exportar y compartir el contenido generado en diferentes lenguajes. Para este proyecto, hemos utilizado Markdown (.md) y MDX (.mdx) como los estándares más flexibles para escribir guías de usuario.

Markdown:

Es un lenguaje de marcado que facilita la aplicación de formato a un texto empleando una serie de caracteres de una forma especial. En principio, fue pensado para elaborar textos cuyo destino iba a ser la web con más rapidez y sencillez que si estuviésemos empleando directamente HTML.

MDX:

MDX brinda la capacidad de escribir contenido y código en un solo archivo utilizando la sintaxis de Markdown y aprovechando las capacidades de React.

React:

React es una librería de JavaScript de código abierto desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reutilizables.

Benchmark:

En inglés, benchmark significa “punto de referencia”, y benchmarking significa “evaluación comparativa“.

Es decir, que el benchmarking es una técnica de investigación que consiste en evaluar y analizar procesos, productos, servicios o áreas para compararlos entre sí y tomarlos como punto de referencia para futuras estrategias.

SEO:

También conocido como optimización para motores de búsqueda, es el conjunto de estrategias y técnicas que se utilizan para mejorar la visibilidad y el posicionamiento de un sitio web en los resultados de búsqueda orgánicos de los motores de búsqueda, como Google.

Bulletproof:

El “Bulletproof Notion Workspace“ es un enfoque o metodología que busca optimizar y maximizar la eficiencia de un espacio de trabajo en Notion. Se basa en combinar una estructura predefinida para organizar la información con técnicas de productividad y algunas características avanzadas de Notion para crear un entorno de trabajo estructurado, flexible y efectivo.

Linear:

Linear es una herramienta de gestión de proyectos y seguimiento de problemas diseñada específicamente para equipos de desarrollo de software. La aplicación permite a los equipos organizar y rastrear tareas, colaborar en proyectos, asignar responsabilidades, establecer plazos y realizar un seguimiento del progreso general del proyecto.

Framework:

Es un conjunto de herramientas, bibliotecas y pautas de desarrollo que proporcionan una estructura y un conjunto de funcionalidades predefinidas para facilitar la creación y el desarrollo de software. Al utilizar un framework, los desarrolladores pueden centrarse más en la lógica específica de su aplicación en lugar de preocuparse por aspectos técnicos o repetitivos.

Open Source o Código abierto:

Open source (o código abierto) se refiere a un tipo de software cuyo código fuente es accesible y está disponible para el público en general. En otras palabras, el software de código abierto permite que cualquier persona pueda ver, modificar y distribuir el código fuente de forma gratuita.

In-house Built:

El término“In-house Built“ se refiere a la creación o desarrollo de software dentro de la propia organización o empresa.

Startup:

Es una empresa emergente que se caracteriza por su enfoque innovador y su potencial de crecimiento rápido. Estas empresas suelen surgir con ideas disruptivas y buscan desarrollar productos o servicios novedosos que satisfagan las necesidades del mercado.

GitHub Actions:

Es un servicio de integración continua y entrega continua (CI/CD) proporcionado por GitHub. Permite automatizar y personalizar diferentes flujos de trabajo en el repositorio de GitHub, lo que incluye tareas como la construcción, prueba y despliegue de aplicaciones.

Características o features:

Se refiere a una funcionalidad específica o aspecto distintivo de un producto, servicio o sistema.

Capítulo 3

Metodología

3.1. Diseño de investigación

3.1.1. Enfoque general de investigación

Durante la realización de este TFM, se empleó un enfoque cualitativo [33] para obtener información detallada acerca de los requisitos, percepciones y opiniones de los ingenieros implicados (lo particular). Gracias a este enfoque, se pudieron establecer criterios que ayudaron a definir el marco de trabajo en el que se enmarcan los resultados obtenidos (lo general). La investigación se realizó de forma abierta y flexible basada en la experiencia de estudiar e interactuar con diferentes aspectos relacionados con la ingeniería, se desarrolló intentando seguir un rigor metódico, en forma de un estudio de casos o estudio clásico de investigación, que, según Creswell en "Qualitative Inquiry and Research Design"[28], es una de las cinco clasificaciones de la práctica cualitativa.

Es importante señalar que, dado que este TFM en parte también pretende aportar una solución técnica, en el proceso de investigación siempre se estuvo presente la necesidad de conocer más acerca de la documentación técnica de las APIs y definir ciertas pautas para implementar funcionalidades inexistentes en el software y de ese modo poder alcanzar los objetivos propuestos. Estas pautas quedan expuestas en los resultados obtenidos.

En definitiva, se puede afirmar que este enfoque garantizó que los resultados obtenidos estén alineados con los objetivos específicos de este TFM, sin perder de vista la importancia de un lenguaje claro y accesible.

3.1.2. Descripción general de la investigación

En primer lugar, se llevó a cabo una revisión de la literatura existente en internet en el campo de la documentación técnica 2.2, documentación de APIs REST 2.2 y portales de desarrolladores de las empresas que ofrecen APIs como producto. Luego se recopiló información mediante entrevistas y encuestas a desarrolladores, ingenieros, CTO, de la muestra, estas encuestas siguieron una serie de preguntas, pero no fueron rigurosas, siempre permitiendo la subjetividad de los entrevistados, permitiendo así obtener datos sobre las necesidades, desafíos y preferencias de los criterios a tener en cuenta en la documentación técnica.

A partir de los primeros datos recopilados se realizó un análisis que permitió identificar patrones y tendencias. Esto fue útil para lograr establecer objetivos específicos, valorando

la experiencia del usuario, pero sin descuidar el tiempo disponible para implementar la solución y los recursos disponibles. Con base en estos datos, primero se diseñaron algunas pruebas de conceptos tomando en cuenta el entorno de la muestra. Más adelante se desarrollaron varios escenarios considerando diferentes aspectos relevantes de la documentación técnica de la API, como por ejemplo: la facilidad de uso, la escalabilidad de la solución, la estructura, la actualización a nuevas versiones del API y la personalización de los documentos.

Posteriormente, teniendo en cuenta estos escenarios, se celebraron varias reuniones con diferentes perfiles para discutir y planificar los aspectos técnicos que comenzaban a surgir de la investigación.

Finalmente, se validó la propuesta mediante un proyecto piloto con la startup. Se evaluó tanto los puntos desfavorables como la eficiencia y la eficacia de la propuesta.

3.2. Población y muestra

3.2.1. Población

Aunque los resultados de este trabajo provienen de una muestra, la investigación intentó enfocarse en startups tecnológicas con entre 2 y 20 desarrolladores que buscan publicar e iterar rápidamente en la documentación técnica de su producto, integrándola en el ciclo de vida del desarrollo de software. Y que además valoran la calidad, soluciones modernas, las buenas prácticas como la revisión de código, el seguimiento de cambios, el versionado y la entrega continua.

Criterios de inclusión:

1. Startups tecnológicas que desarrollan y publican APIs REST.
2. Startups tecnológicas que desean aprender a integrar en el ciclo de desarrollo de software la generación ágil de la documentación técnica.
3. Desarrolladores que participan en la creación y gestión de la documentación técnica de las APIs en sus empresas.

Criterios de exclusión:

1. Empresas no relacionadas con el ámbito tecnológico.
2. Startups tecnológicas que desarrollan y publican APIs SOAP.
3. Empresas tecnológicas que no están involucradas en el desarrollo de APIs o SaaS.
4. Desarrolladores de software que no están involucrados en la creación y gestión de portales de desarrolladores.
5. Proyectos que no requieren de constantes cambios en la documentación.

3.2.2. Muestra

Siguiendo las recomendaciones de la práctica cualitativa se seleccionó una muestra y se optó por el **muestreo por criterios específicos**. Seleccionando a una startup llamada **Salsa**[15] que es una empresa fundada por John Kramer como CEO y Juan Manuel Barroso como CTO, en San Francisco, California. Salsa es una startup americana con ADN español que ofrece APIs y componentes que finalmente permiten a desarrolladores integrar la gestión de nóminas en sus productos con muy poco esfuerzo.

En otras palabras, Salsa simplifica la gestión de nóminas encargándose de tareas complejas como control de la normativa, los cálculos de impuestos y cotizaciones, los reportes y el movimiento de fondos. De esta manera, las aplicaciones de software pueden centrarse en construir una experiencia integrada para gestión de nóminas con muy pocos recursos.

Criterios específicos:

1. Salsa ofrece un “API“ en que el principal usuario son desarrolladores de software que van a consumir la integración a través de interfaces REST.
2. Salsa no dispone de un portal de documentación centralizado e integrado en proceso de desarrollo de las APIs.
3. Los usuarios demandan constantemente el acceso al portal de la documentación durante el proceso de evaluación del producto y la fase de integración.
4. Salsa aplica un enfoque ágil al desarrollo de software e integración continua en su ciclo de desarrollo como ventaja competitiva.

3.3. Instrumentos y técnicas

3.3.1. Entrevistas y encuestas

Las entrevistas permitieron obtener una perspectiva integral y contextualizada de los puntos de vista de los ingenieros, ya que al compartir sus conocimientos y experiencias de manera abierta promovió la recolección de información cualitativa. Además, las entrevistas capturaron aspectos subjetivos de los participantes. Así como matices que enriquecieron notablemente la comprensión del caso de uso investigado. La realización de las entrevistas se llevó a cabo de manera estructurada usando una encuesta [Apéndice A], pero también se brindó flexibilidad en el proceso, lo que permitió que surgieran nuevos temas y la exploración de aspectos no previstos inicialmente en el tema de investigación. La aplicación de esta técnica proporcionó un sólido fundamento para el análisis y la interpretación de los datos obtenidos en este estudio.

Las preguntas abarcaron diversos aspectos. Intentando limar incertidumbre en factores como el grado de participación de los ingenieros en la redacción del contenido, las limitaciones de seguridad al utilizar plataformas de terceros para alojar la documentación, los recursos de ingeniería disponibles para implementar un flujo de entrega continua (DevOps), el conocimiento de lenguajes de programación, experiencia en desarrollo web y experiencia de usuario, la necesidad de tener un alto nivel de control y flexibilidad para personalizar la herramienta. Estas respuestas proporcionarán información valiosa para

determinar las mejores prácticas y recomendaciones en la implementación del SDCL en la documentación técnica.

Véase Apéndice A.

3.3.2. Gestión del proyecto con la empresa y el tutor

La gestión del proyecto desde el punto de vista de la muestra se dividió en tres etapas estratégicas.

En la primera etapa, el enfoque se centró en establecer expectativas de los objetivos tanto a corto como a largo plazo, a través de una evaluación de las soluciones disponibles en el mercado y la generación correspondiente de la documentación técnica. En esta etapa se implementaron 6 POC (Proof of Concept) 3.3.3 que más adelante se describirán como “Artefactos“. En la segunda etapa, se llevaron a cabo 3 pruebas de concepto, esta vez enfatizando características mucho más específicas. Las POCs fueron más rigurosas sobre las herramientas más prometedoras durante la fase inicial.

Por último, en la tercera etapa, se definió y se clasificó la documentación en distintas tipologías y se definieron los criterios para las directrices o “*Guidelines*“ del API. Cabe destacar que la realización de últimas etapas fue un esfuerzo conjunto de toda la empresa, alcanzando así el último y significativo hito de este proceso.

Juan José Coello, Lead Engineer en Salsa, guio estas etapas en reuniones principalmente semanales. Con respecto a la metodología, se siguió la metodología Bulletproof[31], que se basa en los sistemas de productividad populares para crear un marco de trabajo sólido en Notion, basado en pequeños proyectos que van formando una base de conocimiento para todos los miembros.

Además, se contó con el apoyo de Juan Manuel Barroso, CTO de Salsa y diferentes ingenieros para discutir y explorar las soluciones descritas en este trabajo.

Se llevaron a cabo reuniones con el tutor universitario cada 15 días para corregir y revisar los avances de la investigación y asegurar que el proyecto mantuviera un enfoque académico. Para hacer el seguimiento de los objetivos, se utilizaron las mismas herramientas colaborativas que se usaron con la empresa.

3.3.3. Artefactos o pruebas de conceptos

Los artefactos o pruebas de concepto desempeñaron un papel fundamental en la investigación, permitiendo valorar las herramientas antes de que la empresa dedicara recursos a su realización. Estas pruebas fueron realizadas en entornos controlados, mayormente entornos locales y brindaron la oportunidad de evaluar la factibilidad técnica, operativa y económica. Al minimizar los riesgos y proporcionar un entorno cómodo para evaluar cada herramienta, las pruebas de concepto, fueron la fuente principal de las tablas comparativas que se muestran en los resultados de este TFM 4.4. Finalmente, al enfocar el esfuerzo solo en las herramientas que prometían satisfacer los criterios que se mencionan posteriormente, provocó un aumento considerable de información en la investigación.

Durante la primera etapa del proyecto con Salsa se siguieron pautas generales en cada prueba de concepto, y como parte de la investigación se registraron individualmente en

una base de datos en Notion, formando así una vista general que finalmente serviría para seleccionar las tres herramientas más prometedoras:

Pautas generales:

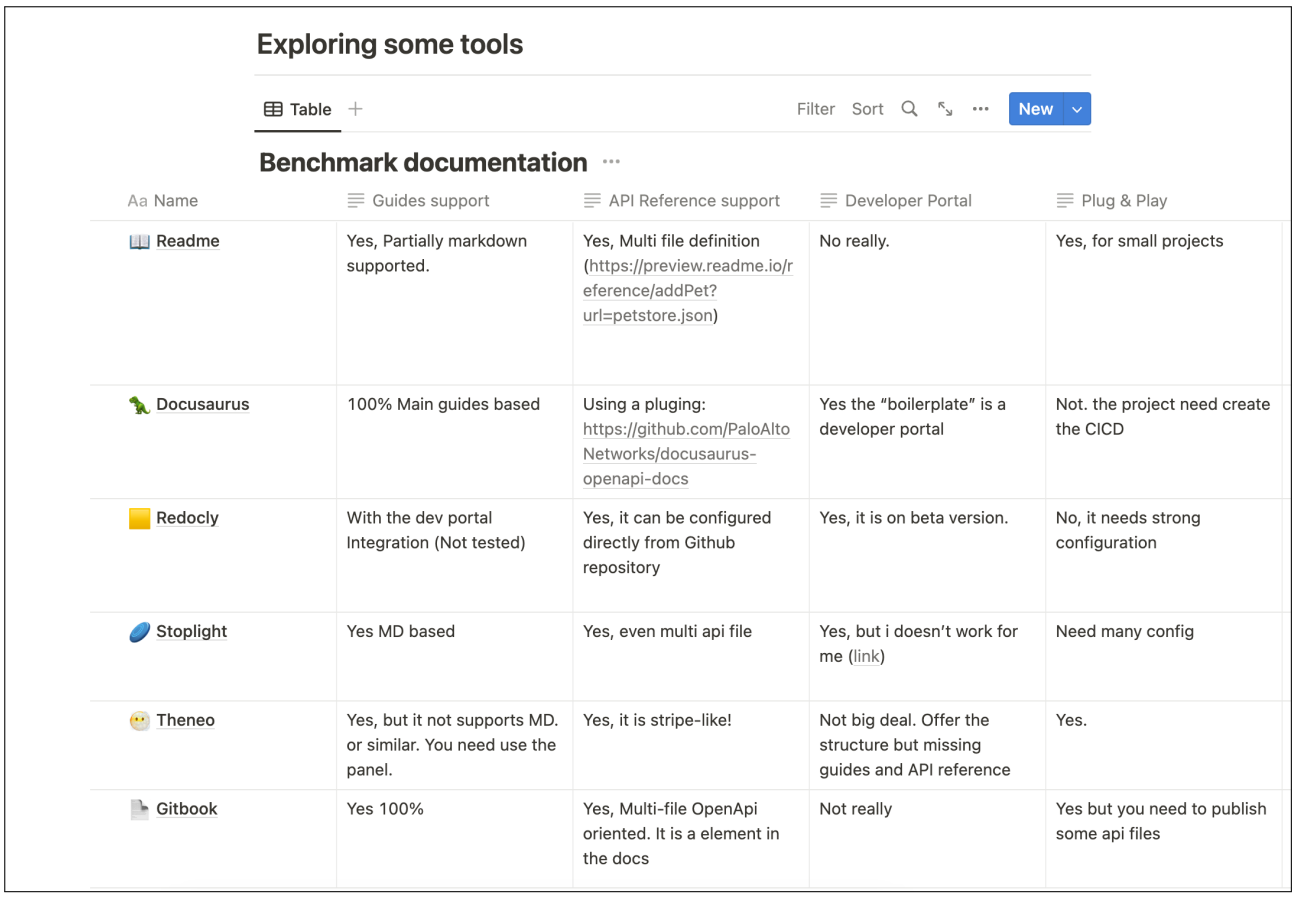
- **Inicia un proyecto y sigue los primeros pasos de la documentación:** Esto consistía en identificar el portal de la documentación y seguir los primeros pasos para acceder o instalar la herramienta, intentando prestar atención en la facilidad de uso y claridad de las instrucciones.
- **Describe el proceso para migrar una guía de usuario:** En cada POC se exportaba desde Notion un documento en formato Markdown (.md) y se importaba en la herramienta para asegurarnos que era viable y comprobar que no existieran problemas de formato.
- **Describe el proceso para implementar el API reference:** En esta etapa el software del API no era capaz de generar la especificación, por lo que se tuvo que usar diferentes OAS que facilitaban las pruebas.
- **Registre todos los problemas encontrados:** Se registraron los problemas que iban surgiendo a medida que se avanzaba en la POC.

Y de forma más específica, una vez realizados las pautas generales recolectaron los siguientes datos:

- **Soporte MD o MDX** para las guías.
- **Soporte de "API reference"** basado en OpenAPI.
- **Generación** del portal del desarrollador.
- **Documentación** descriptiva y bien organizada
- **Plug-and-play** fácil de poner en producción
- **Integración continua** flujos de CICD. 4.3
- **Editor de texto** colaborativo con posibilidad de comentarios.
- **Bloques de código** muestra ejemplos de código.
- **Facilidad de uso** Enfocado a perfiles NO-CODE [27].
- **Nivel de personalización**.
- **Capaz de documentar Widgets**.
- **API Playground:** capaz de hacer llamadas desde el navegador a la API.
- **SEO** amigable con los motores de búsqueda.
- **Soporte técnico**.
- **Precio** Enlace a la lista de precios.

En esta etapa se evaluaron las siguientes herramientas creando pruebas de concepto de las seis primeras:

1. **ReadMe** C.1
2. **Docusaurus** C.2
3. **Redocly** C.3
4. **Stoplight** C.4
5. **Theneo** C.5
6. **GitBook** C.6
7. **Markdoc**
8. **Codehike**
9. **Slate**



Exploring some tools

Table + Filter Sort New

Benchmark documentation

Aa Name	Guides support	API Reference support	Developer Portal	Plug & Play
📖 Readme	Yes, Partially markdown supported.	Yes, Multi file definition (https://preview.readme.io/reference/addPet?url=petstore.json)	No really.	Yes, for small projects
🦖 Docusaurus	100% Main guides based	Using a pluging: https://github.com/PaloAltoNetworks/docusaurus-openapi-docs	Yes the “boilerplate” is a developer portal	Not. the project need create the CICD
🟨 Redocly	With the dev portal Integration (Not tested)	Yes, it can be configured directly from Github repository	Yes, it is on beta version.	No, it needs strong configuration
🪀 Stoplight	Yes MD based	Yes, even multi api file	Yes, but i doesn’t work for me (link)	Need many config
😬 Theneo	Yes, but it not supports MD. or similar. You need use the panel.	Yes, it is stripe-like!	Not big deal. Offer the structure but missing guides and API reference	Yes.
Gitbook	Yes 100%	Yes, Multi-file OpenApi oriented. It is a element in the docs	Not really	Yes but you need to publish some api files

Figura 3.1: Tabla comparativa y registros de las POCs

3.3.4. Evaluación comparativa

Durante la realización de las pruebas de conceptos [3.3.3], se fue recolectando información detallada de las herramientas con el fin de realizar dos iteraciones para evaluar y comparar los resultados. En la primera iteración, se realizó una evaluación inicial de ReadMe, Docusaurus, Redocly, Stoplight, Theneo, GitBook. Utilizando los criterios evaluados durante las POCs.

Se utilizó la base de datos antes mencionada para comparar las características de las seis herramientas, discutiendo sobre el aporte de cada una sobre los objetivos propuestos en el proyecto. Los detalles se muestran en los resultados 4.4

Para la segunda iteración, se seleccionaron las tres opciones más destacadas (**ReadMe, Redocly y Docusaurus**) y se llevó a cabo una evaluación exhaustiva. En esta segunda evaluación se tomaron en cuenta las características o “features“ 2.2 principales como:

- **Gestión de documentos**
- **API reference**
- **Editor de texto**
- **Gestor de categorías**
- **Personalización**
- **Seguridad**
- **Analítica**
- **Monitorización**

- **Soporte técnico**

Las cuales posteriormente se desglosaron en criterios más pequeños identificados a partir de datos cualitativos obtenidos en las entrevistas. Luego fueron valorados en las pruebas de conceptos con detenimiento y registrando en tablas los resultados 4.4 obtenidos por cada característica de la lista.

En conclusión, la evaluación comparativa consistió en dos iteraciones en las que se evaluaron y compararon las soluciones usando las pruebas de concepto como base. Los resultados de esta evaluación se registraron y proporcionaron la información necesaria para elegir la herramienta más adecuada para Salsa.

3.3.5. Metodología Bulletproof

El “Bulletproof Notion Workspace“ es un enfoque o una metodología que busca optimizar y maximizar la eficiencia de un espacio de trabajo en Notion. Se basa en la combinación de técnicas de productividad y las características avanzadas de Notion para crear un entorno de trabajo estructurado, flexible y efectivo.

Bulletproof se basa en una página de nivel superior llamada “Vault“, lo que organiza las bases de datos maestras en tres sub páginas:

Buckets o Áreas y Bolts

La página de **Buckets** es una base de datos en sí misma. Esa base de datos incluye las categorías de alto nivel de la información. La página de **Bolts** contiene bases de datos que demuestran y miden el progreso. Continúan expandiéndose a medida que haces tu trabajo.

Bytes

A fines prácticos de esta investigación es el recurso más importante. Las bases de datos dentro de la página de **Bytes** apoyan el trabajo, incluyendo información específica del tema en cuestión, metas y notas, tareas, referencias.

A continuación se muestra cómo se definen los metas datos de cada Byte.

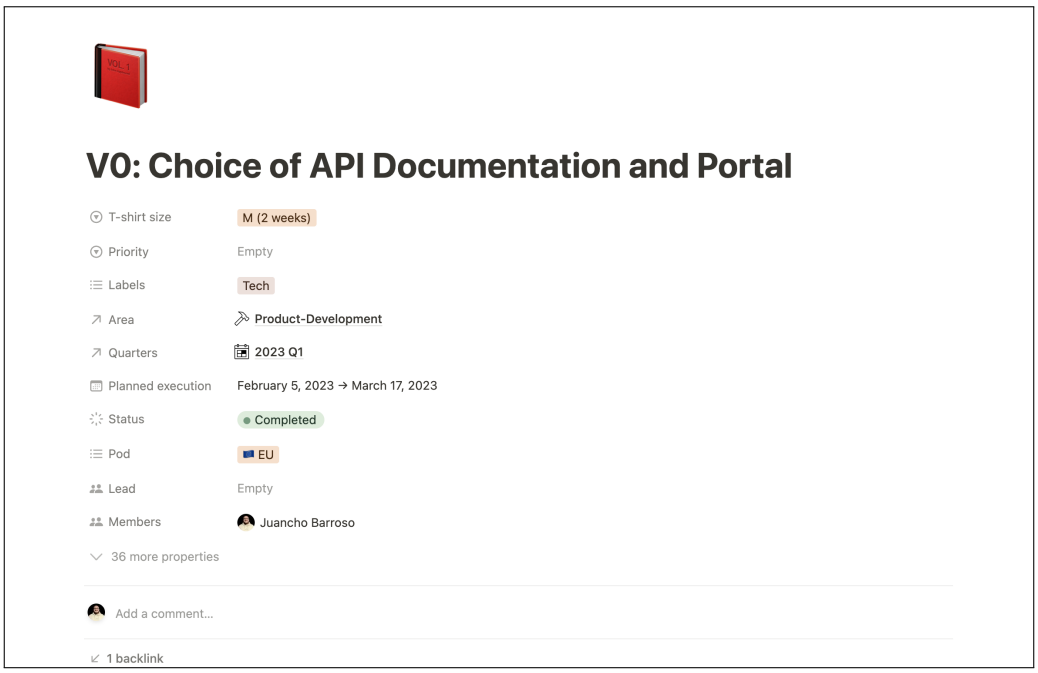


Figura 3.2: Bytes: cabecera y meta-datos

Seguidamente, encontraremos una lista de tareas que suman el total del esfuerzo para completar los objetivos del recurso en Notion.

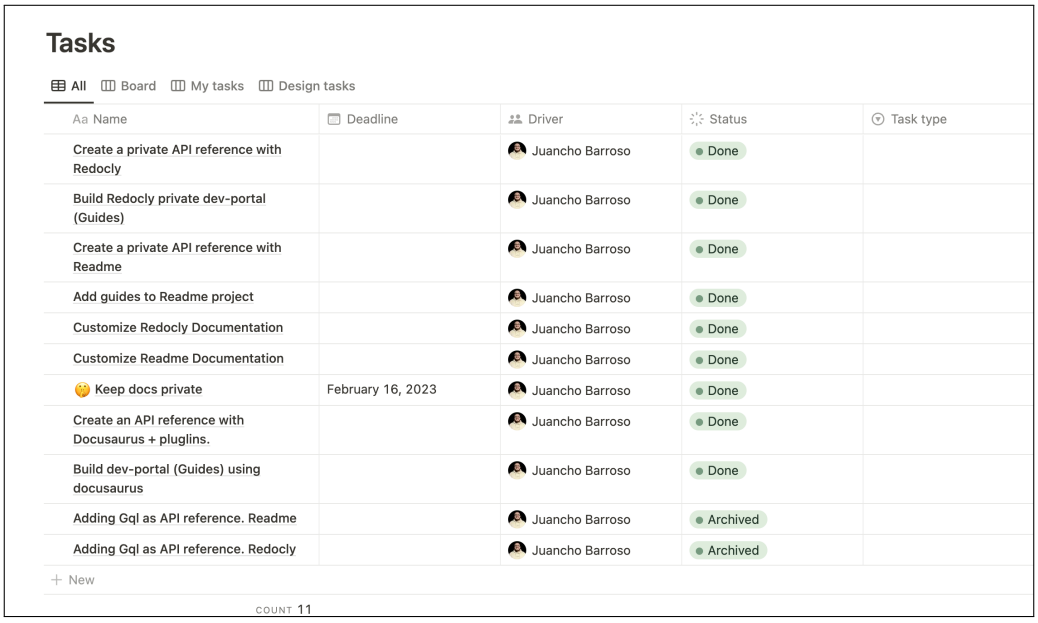


Figura 3.3: Bytes: Tabla de tareas

La base de datos principal dentro de Bytes es la base de datos de **Recursos o Resources**. Tan a menudo como sea posible, se almacena información en **Resources**.

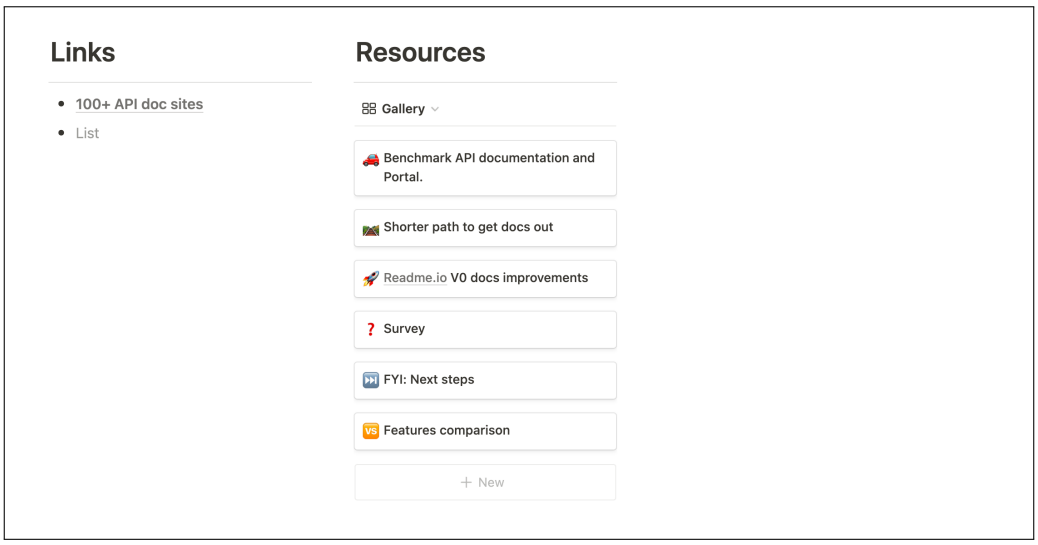


Figura 3.4: Bytes: Enlaces y recursos

En resumen, la metodología Bulletproof se utiliza como un instrumento en la investigación debido a su capacidad de organizar y gestionar información de un proyecto de manera eficiente. Proporciona estructuras claras y flexibles que garantizan la precisión y coherencia de los datos, adaptándose a las necesidades de la investigación. Además, fomenta la colaboración y mejora la productividad del equipo de investigación al facilitar la comunicación y el intercambio de información en tiempo real gracias a la característica colaborativa de Notion.

3.3.6. “Build days“ y proyecto en Linear

Los “build days“ o días de construcción en español, son una estrategia que utiliza Salsa para construir soluciones en un periodo de tiempo muy corto. Los días de construcción se realizan durante una semana cada seis semanas y consisten en que de forma intensiva todos los miembros de la empresa dediquen su tiempo en un mismo proyecto. Estos hitos suelen ser proyectos que no requieren más de una semana. La estrategia de “Build days“ forma parte de los instrumentos de este trabajo porque es una estrategia para alcanzar objetivos rápidamente y a la hora de aplicar las pautas indicadas en los resultados de este trabajo fue clave.

Durante una semana el equipo estuvo trabajando en un proyecto llamado *“Build days - Documentation site + V1 APIs“* donde el objetivo principal era integrar la documentación en los procesos de desarrollo.

Para la gestión de las tareas se usó un proyecto en Linear2.2. Por todos los beneficios que ofrece Linear, se decidió crear un proyecto con la lista de tareas necesarias para generar la versión “V1“ del API.

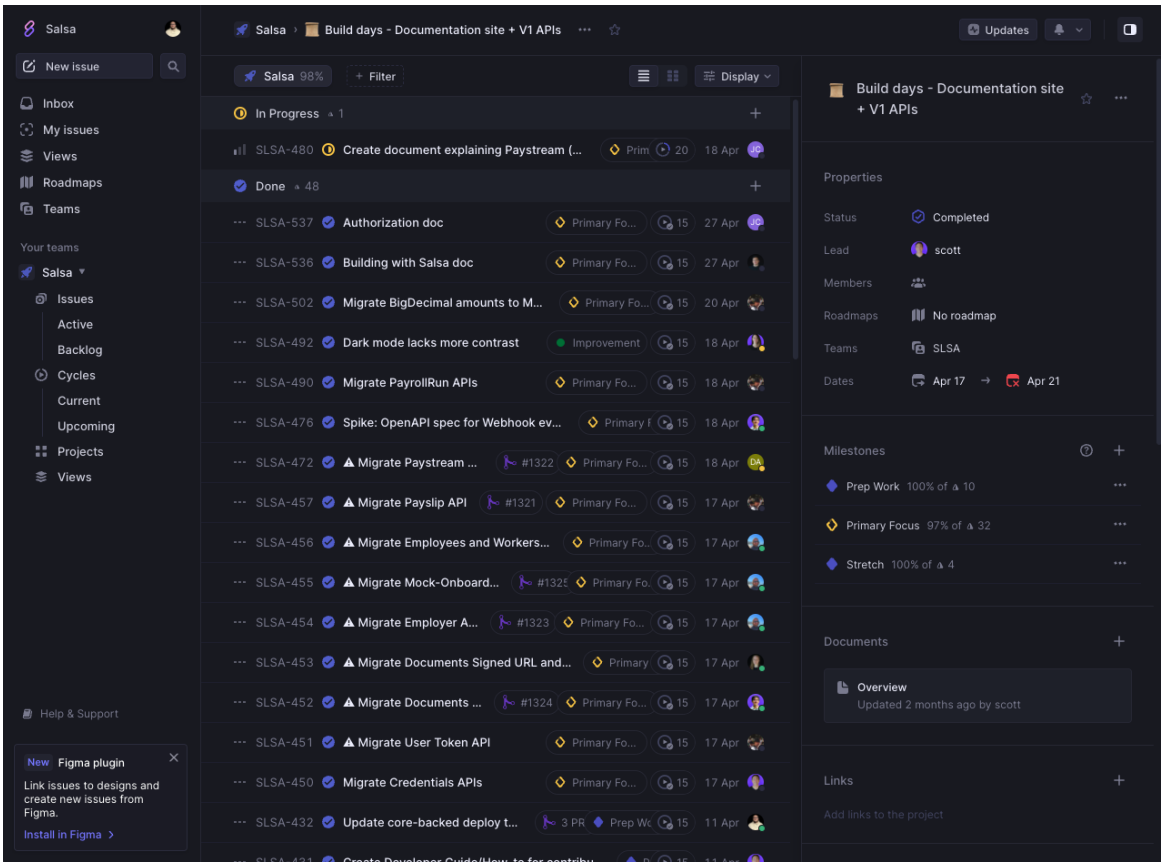


Figura 3.5: Build days: Documentation site + V1 APIs

Un apunte importante, antes de que la empresa se dedicara a realizar este hito, se hicieron pruebas de conceptos 3.3.3 que modelaban el objetivo.

Capítulo 4

Resultados

En este capítulo se describen los resultados en cada etapa de la investigación; comenzando por la búsqueda y evaluación de la mejor solución tecnológica en términos de documentación que posteriormente servirían de la base para crear una documentación técnica de alta calidad basada en 4 tipos de documentos.

4.1. No hay un tipo documentación, ¡hay cuatro!

En el intento de mejorar las guías de usuario se descubrió que existe una clasificación muy efectiva para organizar y escribir los diferentes tipos de documentación que existen. En esta sección se expondrán los resultados obtenidos después de aprender más sobre dicho descubrimiento.

Hasta este punto, Salsa[15] venían escribiendo los diferentes documentos acerca de sus productos en páginas de Notion2.2. Esto por la agilidad que ofrece Notion a la hora de publicar el contenido y la simplicidad de escribir nuevas páginas. Notion es un gran producto, pero no está diseñado para publicar documentación técnica para APIs, ya que no se conecta con las herramientas que usan los desarrolladores para crear el producto y hace muy complicado hacer cambios a documentación existente porque todos sus cambios son “en caliente”.

–En la siguiente imagen se puede ver el aspecto de la documentación en Notion antes de hacer la migración:

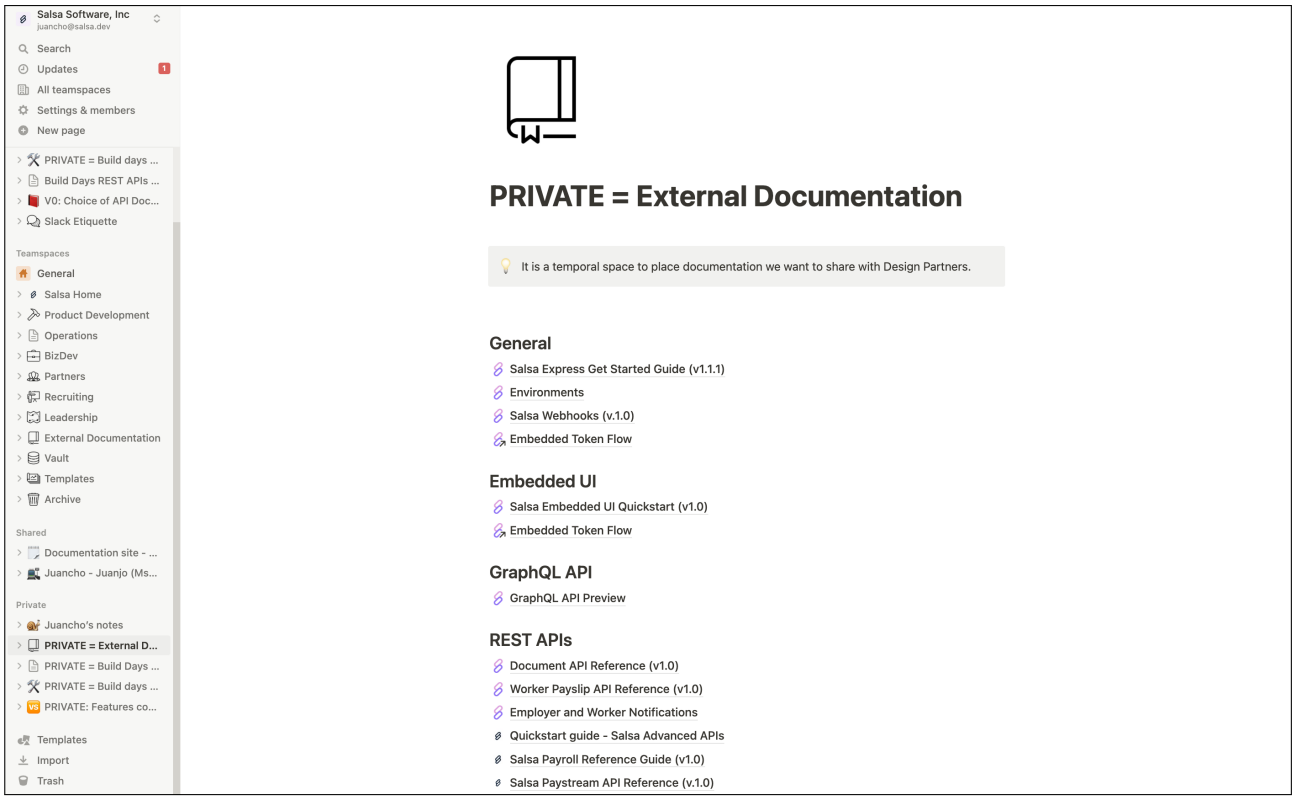


Figura 4.1: External Documentation: Legacy

Para mejorar las guías de Notion se tomó la decisión de migrarlas a una nueva herramienta. Pero antes era necesario desarrollar los criterios y estructura de la nueva documentación.

4.1.1. El secreto: 4 tipos de documentación

A lo largo de la investigación sobre la importancia de una buena documentación surge la interrogante **¿Cuál es el secreto de una buena documentación?** ¿Qué es lo que hacen los casos de éxito como Stripe [18], Plaid[10] y Marqueta [7] para que su documentación sea un referente en internet?. Después de entender y comparar algunas referencias se llegó a la conclusión que el secreto está en **la clasificación, estructura y responsabilidad de cada documento**.

Esto llevó a descubrir que la documentación debe estructurarse en torno a cuatro tipos de documentos con diferentes responsabilidades: **Tutoriales, Guías prácticas, Referencias técnicas y Explicación**. Cada una de ellas requiere un modo de escritura distinto y se descubrió que los desarrolladores de software necesitan estos cuatro tipos de documentos en momentos y circunstancias diferentes; pero a pesar de eso, todos estos deberían estar integrados en la documentación técnica y deben convivir entre ellos, generando un balance coherente para el consumidor.

Es fundamental que la documentación esté estructurada de manera **explícita** en relación con estas funciones, y que cada una se mantenga claramente definida.

La siguiente tabla proporciona mayor claridad para el editor o escritor de la documentación en cuanto al tipo de contenido que corresponde a cada sección dentro de la documentación técnica. Así como también para el desarrollador que consume la documentación.

	Tutorial	**Guías prácticas**	**Referencias técnicas**	**Explicación**
Orientada ha	Aprender	Una meta	Información	Entender
Debe	Ayudar a nuevos clientes a comenzar	Mostrar como resolver un problema específico	Describir el mecanismo	Explicar
Forma verbal	Como si fuera una lección	Una serie de pasos a seguir	Una descripción concisa	Una descripción desarrollada
Analogía	Enseñar a un niño a saltar la cuerda	La receta de una tarta de chocolate	Una referencia de una enciclopedia	Un articulo de como funciona el motor a 4 tiempos

Figura 4.2: Tutoriales, Guías prácticas, Referencias técnicas y Explicación

Se estableció que objetivo principal es guiar al editor/escritor en cuanto al estilo de la redacción, es decir el “qué escribir“ y “dónde ubicarlo“. La idea es evitar que el editor invierta mucho tiempo tratando de moldear la información que desea transmitir, ya que cada tipo de documentación tiene un propósito único.

De hecho, finalmente resultó que sin aplicar de manera explícita los cuatro cuadrantes de este esquema era sumamente desafiante mantener la documentación por buen camino. Los criterios de cada tipo de documentación difieren entre sí, por lo que cualquier intento de documentación que no mantenga esta estructura se vio afectado.

4.1.2. Tutoriales

Los tutoriales son lecciones que acompañan paso a paso al lector de la documentación guiándolos para aprender algo específico, en este caso a los desarrolladores de software. Los tutoriales tienen la responsabilidad de mostrar lo que pueden hacer con el producto.

La forma narrativa de este tipo de documentación se basa en pensar que el escritor es el profesor y es responsable de lo que el estudiante hará. Bajo su instrucción, el estudiante llevará a cabo una serie de acciones para lograr algún objetivo. El objetivo debe ser significativo, pero también alcanzable para un principiante. Por lo que los tutoriales son los que convertirán a los aprendices en usuarios del producto.

Un tutorial mal diseñado o simplemente ausente impedirá que un proyecto consiga convencer a nuevos usuarios a quedarse. Durante la investigación, después de comparar varios productos se descubrió que la mayoría de los proyectos similares tienen tutoriales realmente complicados. Y recordando los antecedentes de este TFM, descubrimos que una documentación optimizada mejora directamente la productividad de los usuarios.

Cómo escribir “Tutoriales“:

Permitir que el usuario aprenda haciendo: es esencial brindar al usuario la oportunidad de aprender a través de la práctica activa. De esta manera, el usuario podrá desarrollar habilidades prácticas, adquirir experiencia significativa y fortalecer su comprensión del tema.

Ayuda al usuario a comenzar: el enfoque debe estar en proporcionar los recursos e instrucciones necesarias para que el principiante adquiera una base sólida y no todo el contexto a la primera.

Asegure de que el tutorial funciona: Lo más importante es que las acciones solicitadas al principiante deben funcionar como se indica porque es fundamental garantizar que las instrucciones sean claras, concisas y rigurosamente probadas para evitar confusiones o resultados no deseados.

Asegure que el usuario vea resultados de inmediato: Cada acción realizada debe tener un impacto, por más pequeño que sea. Si el desarrollador tiene que realizar acciones extrañas e incomprensibles durante un período prolongado antes de ver algún resultado, esto puede generar frustración y desmotivaciones.

Debe ser agnóstico y extensible: el tutorial debe ser confiable y repetible, tomando en cuenta las variaciones en los sistemas operativos, niveles de experiencia y herramientas utilizadas.

Enfóquese en los pasos concretos, no en los conceptos abstractos: Es importante evitar la tentación de introducir abstracciones del negocio demasiado pronto, ya que el aprendizaje efectivo se desarrolla desde lo simple y concreto hacia lo general y abstracto.

Proporcione la explicación mínima necesaria: Es fundamental limitar la cantidad de explicaciones extensas y centrarse en proporcionar solo la información necesaria para completar el tutorial. De otro modo sería una *Explicación*.

Enfoque únicamente los pasos que el usuario necesita seguir: una API pueden tener múltiples opciones o enfoques, en el contexto del tutorial, es importante limitar la información a lo estrictamente necesario para el proceso.

Ejemplo de un tutorial:

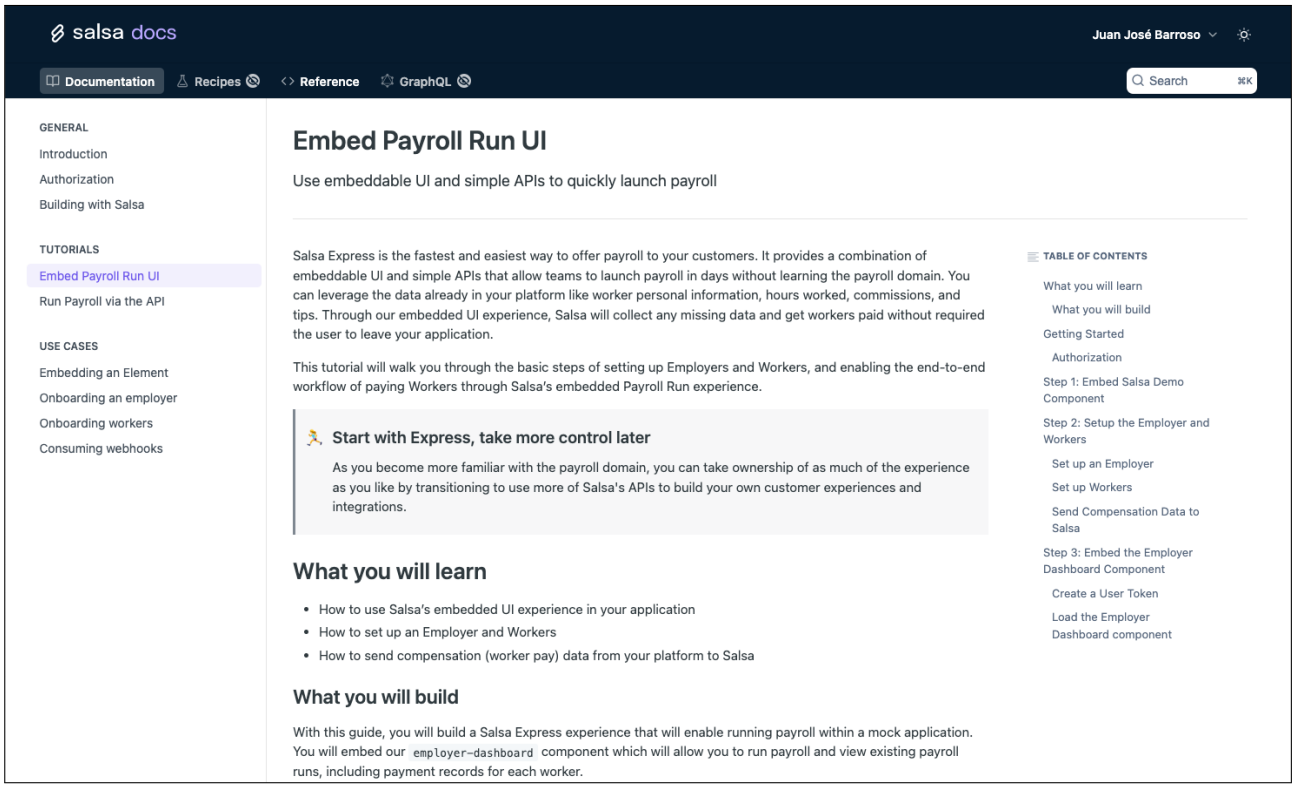


Figura 4.3: Tutorial: Embed Payroll Run UI (Incrustar interfaz de usuario de ejecución de nómina)

4.1.3. Guías prácticas o casos de uso

Guías prácticas o casos de uso, son una forma efectiva de llevar a los lectores a través de los pasos necesarios para resolver problemas del mundo real. A diferencia de los tutoriales, las guías prácticas están dirigidas a usuarios con cierto nivel de experiencia y conocimiento previo. Usuarios que ya conocen el dominio, pero necesitan conocer algo muy específico; es decir que se puede asumir que el desarrollador de software ya sabe cómo hacer cosas básicas y usar herramientas básicas.

Puede proporcionar instrucciones paso a paso, indicaciones claras y ejemplos relevantes para ayudar al usuario a resolver el problema planteado.

Cómo escribir "Guías prácticas o Casos de uso":

Proporcione una ruta a seguir: Deben presentar una serie de pasos que deben seguirse en orden para lograr el objetivo práctico planteado.

Centrarse en los resultados: Enfocarse en lograr un objetivo práctico y evitar cualquier otra información que pueda dar como resultado una distracción. Evite las explicaciones de conceptos en las guías prácticas.

Resolver un caso de uso en particular: Debe abordar un problema o pregunta específica que el desarrollador pueda responder a "¿Cómo puedo. . . ¿' El enfoque debe estar en proporcionar la respuesta y pasos concretos para abordar ese caso de uso en particular.

No explique conceptos: No es el lugar para discusiones de ese tipo. Si las explicaciones son importantes, enlace a ellas.

Permita cierta flexibilidad: Es beneficioso proporcionar explicaciones y opciones que permitan a los desarrolladores adaptar la ruta a su contexto específico. Esto les brinda la oportunidad de comprender cómo aplicar los conceptos en diferentes escenarios y sistemas.

El nombre guía bien: El título debe comunicar de manera concisa y precisa lo que se espera lograr al seguir los pasos proporcionados. Por ejemplo, en lugar de utilizar un título como “Onboarding an employer“, que es vago e indeterminado, un título más claro y específico como “How you can onboard a business as a payroll employer using our hosted UI experience.“ En el caso de que sea muy largo como este ejemplo use subtítulos que aclaren inmediatamente el objetivo.

Ejemplo de un Caso de uso:

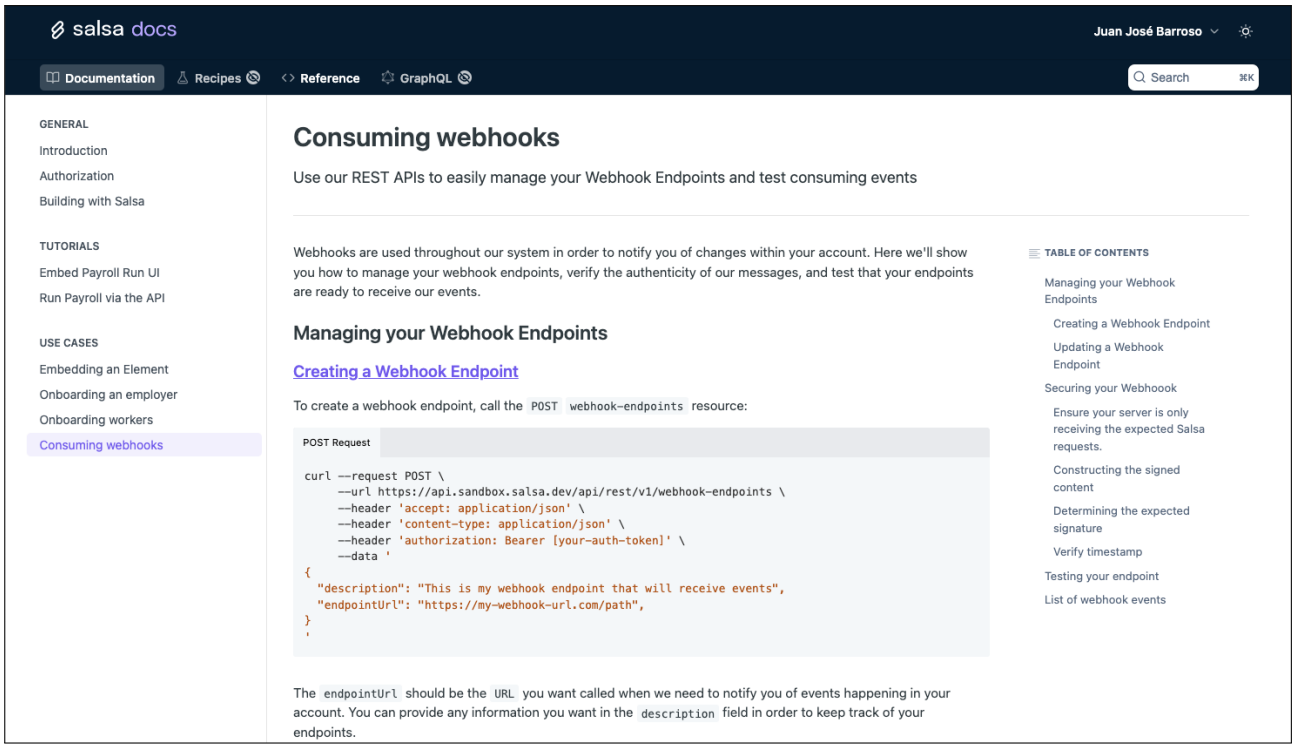


Figura 4.4: Casos de uso: Consuming webhooks

4.1.4. Referencias técnicas

Se centran en proporcionar información técnica detallada y en cómo utilizar los endpoints, métodos, respuestas, errores, paginación, etc.

En esta sección no se definirán las “Referencias técnicas“ porque más adelante se dedica una sección exclusivamente para mostrar los descubrimientos y resultados de la referencia del API o especificación OpenAPI. Véase [4.2]

Ejemplo del API reference:

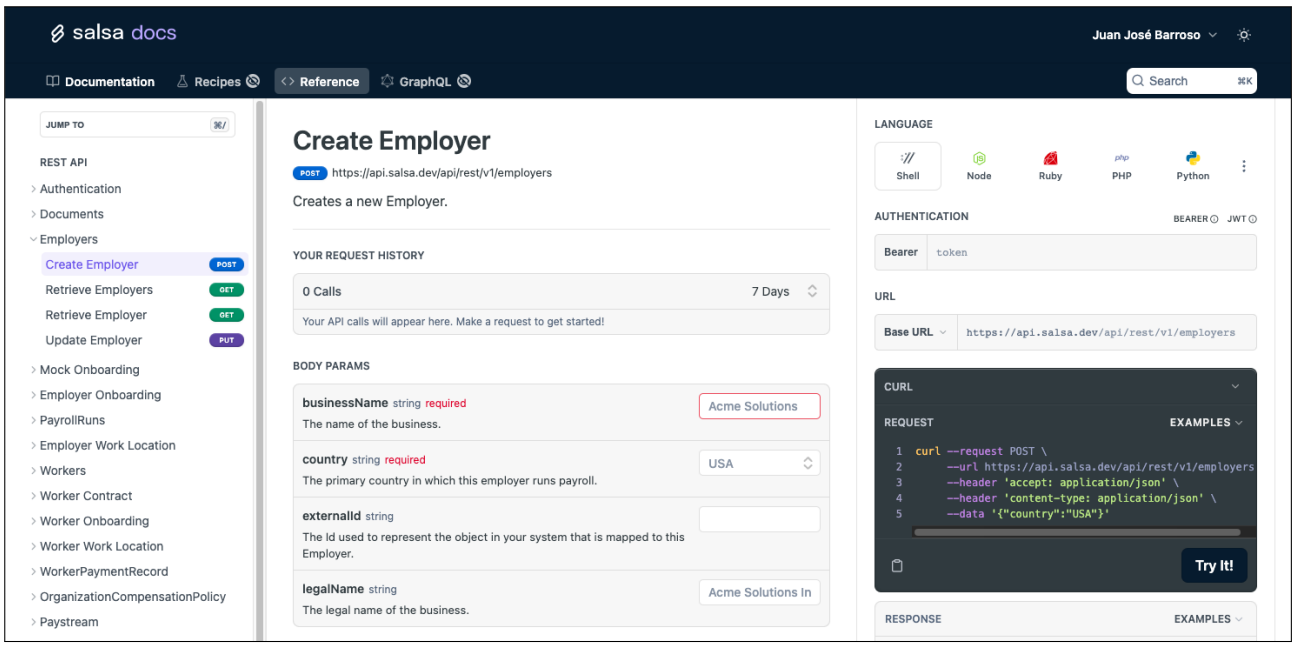


Figura 4.5: API reference

4.1.5. Explicación

Crear explicaciones puede ser un desafío, ya que requiere abordar un tema desde cero y proporcionar una explicación clara y coherente. Deben responder a la pregunta –¿Qué es. . . ?. A diferencia de los Tutoriales, Guías prácticas o Referencias, las explicaciones no están orientadas a una tarea específica. si no a una comprensión más amplia y profunda sobre concepto.

Ejemplo de una Explicación:

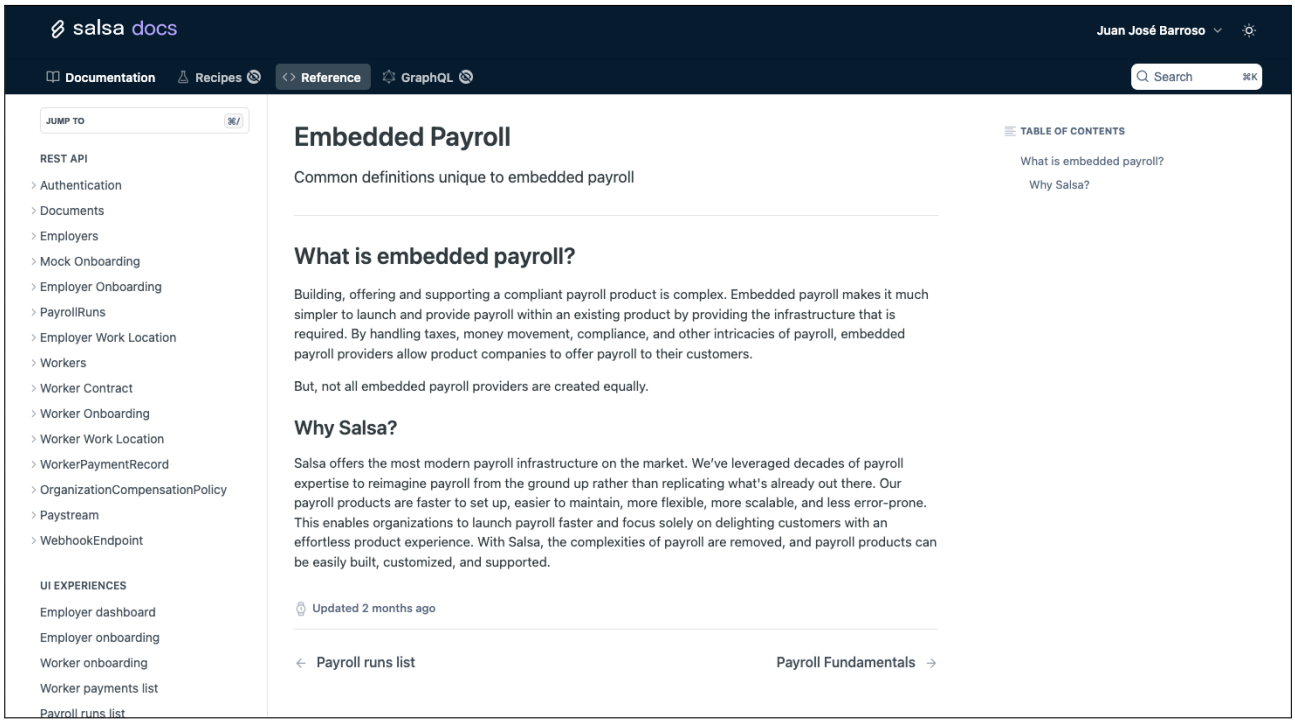


Figura 4.6: Explicación: What is embedded payroll?

4.2. La especificación OpenAPI como la clave

Durante la fase de pruebas de concepto [3.3.3], surgieron interrogantes sobre la capacidad del software del API para generar automáticamente su especificación. Se determinó que, con el fin de asegurar la mejor experiencia de desarrollo posible, era crucial adaptar el software del API para que pudiera generar la especificación de forma automática con un solo comando. Esto se detalla en el apéndice D, lo que permitiría su generación desde un flujo de CI/CD. Aunque existen enfoques como API First, donde se define el contrato del API antes de su implementación, en este caso particular se optó por generar la especificación del API a partir de una versión del mismo.

Esto planteó un desafío para los ingenieros, ya que hasta ese momento, el API se estaba adhiriendo al estándar "JsonAPI"para describir su estructura. Después de evaluar todas las herramientas disponibles [4.4] para abordar esta cuestión y discutir las alternativas de OpenAPI [5.3.2], se llegó a la conclusión de que era necesario realizar ciertos cambios en el proyecto del API para alcanzar este objetivo.

Decisión a tomar:

- Adoptar OpenAPI como el estándar para definir las APIs.
- Pivotar y dejar de usar JsonAPI para describir la estructura de las APIs.

Algunos criterios relacionados:

- Con respecto a JsonAPI: Las herramientas del mercado no han adaptado JsonAPI como el estándar, incluso las herramientas basadas en JsonAPI, RAML y API Blueprint, se consideraron inmaduras para usarlas en producción.
- OpenAPI parece ser el estándar mejor soportado por las opciones disponibles en el mercado y es necesario para la documentación del API.
- Parece que estos estándares, aunque son conceptos iguales, no juegan bien juntos en la documentación técnica.

¿Por qué OpenAPI?:

- Porque facilita el diseño de la API gracias a las herramientas que existen para ello.
- Familiaridad y preferencias del equipo de desarrollo.
- Generación automática de documentación: Al tener una especificación de OpenAPI, es posible utilizar herramientas y bibliotecas para generar automáticamente la documentación de la API. Esto ahorra tiempo y esfuerzo
- Validación y pruebas: Las especificaciones de OpenAPI permiten realizar validaciones y pruebas automáticas en la API para verificar su cumplimiento con la especificación.
- Facilita la creación de SDK y clientes: es más sencillo generar automáticamente SDK, existen múltiples herramientas que permiten generar el código fuente de servidores de APIs a partir de su diseño, la documentación de una API ya implementada.

De esta manera se debe resaltar la importancia de implementar este cambio. Sin la especificación **OpenAPI** era casi imposible lograr los objetivos de este proyecto. A continuación se presentan las pautas y definiciones que aseguran que las referencias

técnicas de la documentación estén alineadas con la calidad, lenguaje y estructura del resto de documentos.

4.2.1. Pautas para la especificación del API en Salsa

En esta parte, se buscó mantener la simplicidad y evitar cambios de diseño o reestructuraciones importantes en la capa REST del código. Se siguieron directrices generales y amplias [5.3.3], las cuales se utilizaron como base para establecer pautas específicas. Estas pautas aseguraron la coherencia de las descripciones, títulos y conceptos en las referencias técnicas con el resto de la documentación. Cabe mencionar que algunas de estas pautas hacen referencia a directivas específicas para entornos basados en Kotlin[5].

API REST Guidelines de Salsa

Las directrices para la especificación del API en Salsa se basaron en las *“Microsoft REST API Guidelines“* como punto de partida. A partir de esta referencia, se definieron todos los aspectos técnicos importantes para dar forma a la API. En el Apéndice B de este trabajo, se incluye una copia de la versión más reciente de estas directrices hasta la fecha.

Prerrequisitos:

Familiarízate con:

- La “API REST Guidelines“ de Salsa. (público)
- La documentación del “core-backend“ de Salsa (privado)
- Algunos ejemplos de las POC.
- Asegúrate de revisar la “Lista de chequeo de REST APIs“ antes de trabajar en REST APIs.
- Usa las anotaciones de Salsa dependiendo del tipo de API, mira las opciones disponibles en SalsaApiAnnotations.

Crea un `<Domain>RestController` bajo el paquete `inbound.rest`.

Controladores

Por ejemplo, para una API de employer se debe usar:

```
@SalsaEmployerController  (1)
@SalsaErrorHandlerV1      (2)
@Tag(name = "Workers")    (3)
internal class WorkerRestController() {
    ....
}
```

1. API con contexto de empleador.

2. Usando el manejo de errores de Salsa, V1
3. Tag del API con el nombre del recurso

Añadir operaciones al controlador REST usando las anotaciones

```
  @PostMapping("/{employerId}/workers")    (1)
  suspend fun createWorker(
      @Parameter(description = "Employer ID")
      @PathVariable employerId: String, (2)
      @RequestBody input: CreateWorkerRestInput
  ): RestApiEnvelope<WorkerRestDomain> (3) {
  return RestApiEnvelope(
      data = workerRestService.createWorker
      (employerId, input) (4)
  )
}
```

1. Adicionalmente, se agrega un path adicional que comienza desde la url base del controlador, incluye employerId para poder ejecutar las mismas operaciones para diferentes empleadores.
2. Usa `@Parameter` para enriquecer la documentación de OpenAPI
3. Todas las operaciones REST devolverán el `RestApiEnvelope` genérico anotado con el tipo de respuesta real.
4. No añadas lógica en los controladores, delega en un `RestService`

Nota: La razón por la que no queremos agregar ninguna lógica a los controladores REST es porque estamos trabajando para generarlos automáticamente en función de la especificación OpenAPI, por lo que cualquier lógica tendrá que residir en cualquier otro lugar.

Resultado en las referencias del API:

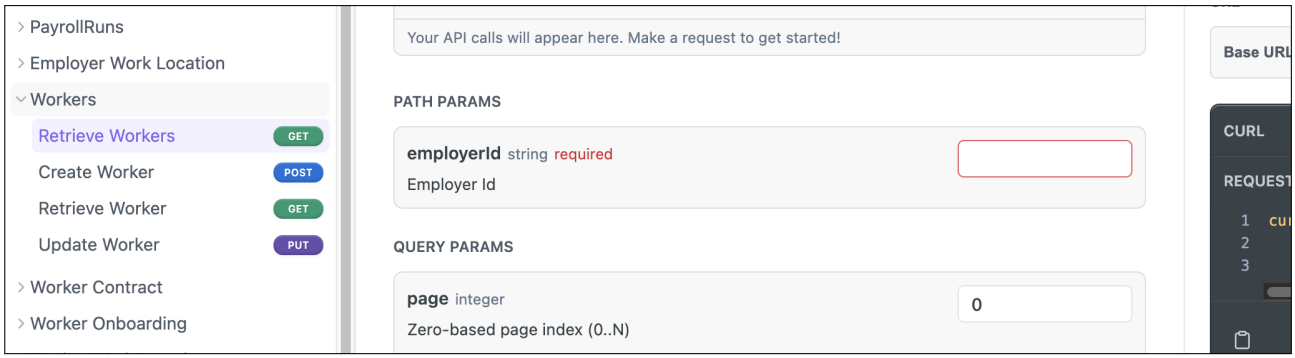


Figura 4.7: @Tag(name = "Workers") añade el Workers en la lista de recursos

Servicios

Crea un servicio REST que contendrá la lógica para llamar a los servicios de aplicación correspondientes.

El servicio REST correspondiente, que contendrá las mismas operaciones REST, devolverá `RestDomain`:

```
@Component
class WorkerRestService(
    private val findWorkerUseCase: FindWorkerUseCase,
) {
    @PreAuthorize(
      "hasAuthority('${SalsaAuthorizationScopes.WORKER_READ}')"
    )suspend fun getAllWorkers(
          employerId: String,
          pageable: SalsaPageable? = null
    ): List<WorkerRestDomain> {
        val workers =
        findWorkerUseCase.findAllWorkersByEmployerId(
        DomainEntityId.fromString(employerId),
        pageable?.toRepositoryPageable())
        return workers.map { WorkerRestDomain.fromDomain(it) }
    }
```

Resultado en las referencias del API:

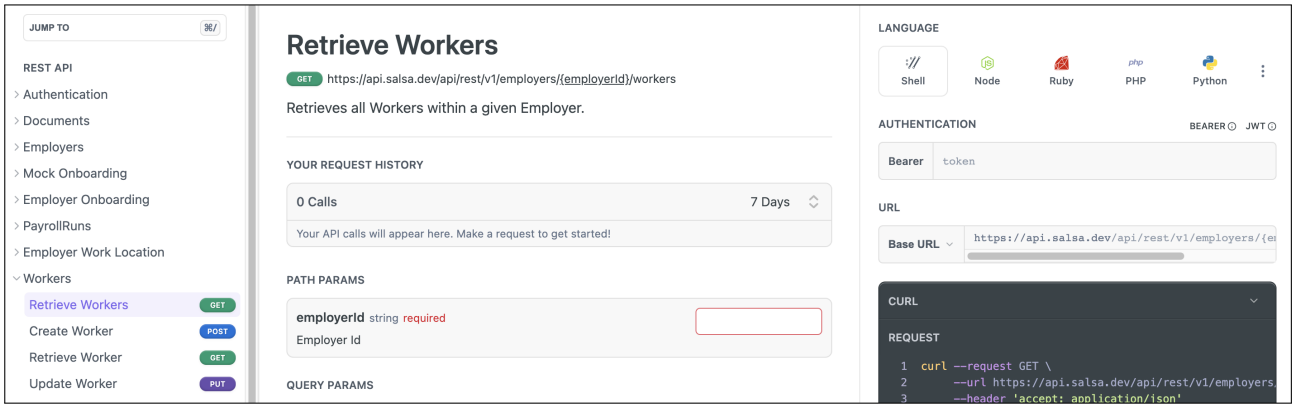


Figura 4.8: Expone el recurso Workers en la documentación

Entradas y Salidas

- Las respuestas REST serán clases de datos con el nombre `<EntityName>RestDomain` mientras que los parámetros REST serán `<Name>RestInput`
- Use las anotaciones `@Schema` en cada `RestDomain` y `RestInput` para enriquecer la documentación de OpenAPI.
- Cada campo de cada objeto debe tener al menos una descripción y datos de ejemplo.
- Cuando sea necesario, los objetos REST tendrán métodos `fromDomain` y `toDomain` que transformarán en objetos de dominio:

```
@Schema(name = "MyEntity", descriptio="description...")
data class MyEntityRestDomain(
    @Schema(descriptio="description...", example="1")
    val id: String,
```

```
    @Schema(descriptio="description...", example="1")
    val name: String
) {
    companion object {
        fun fromDomain(myEntity: MyEntity):
        MyEntityRestDomain {
            ...
        }
    }
    fun toDomain(): MyEntity {
        ...
    }
}
```

Cada objeto individual debe tener sus propias funciones

Resultado de las Entradas en las referencias del API:

Figura 4.9: Añade los parámetros de entrada de la URL.

Resultado de las Respuestas en las referencias del API:

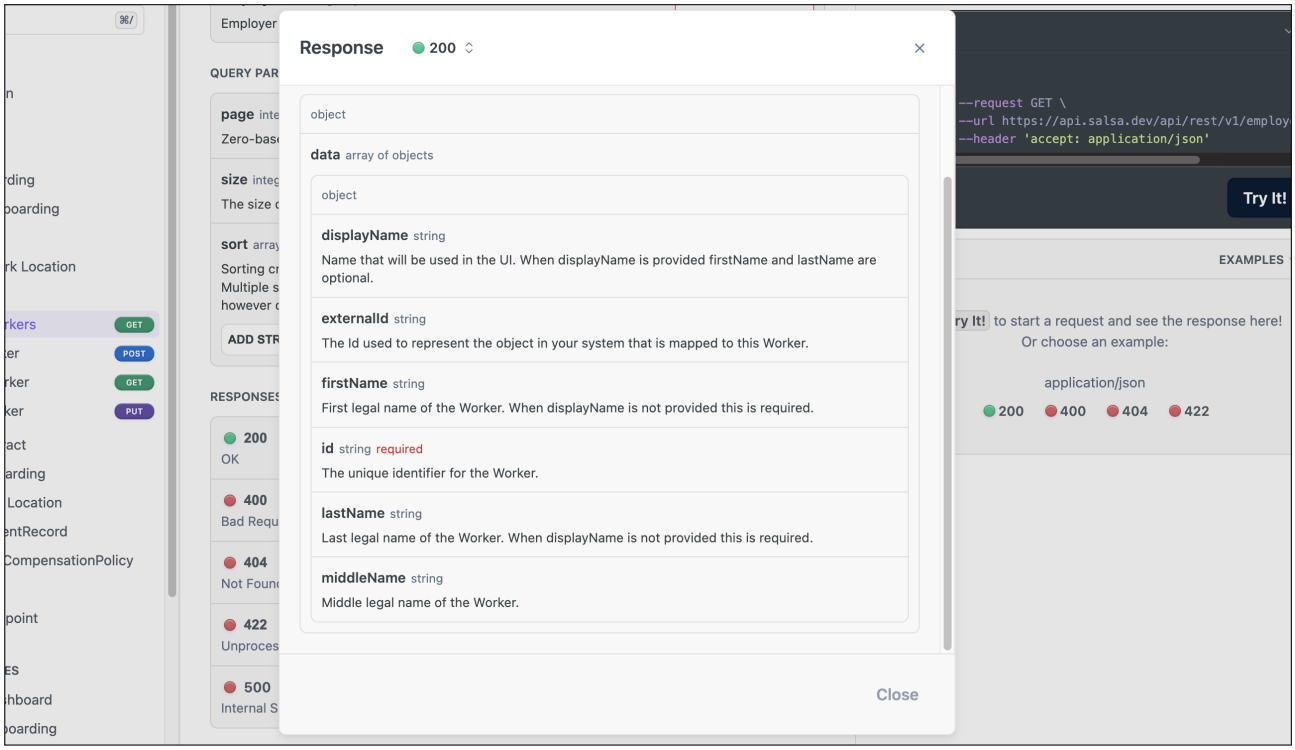


Figura 4.10: Añade las respuestas correctas

Pagination

Cada operación de lista debe soportar el parámetro `SalsaPageable` que soporta paginación por página/tamaño.

Todo: Definir cómo implementar la paginación basada en cursores.

El método `findAll` en el repositorio de servicios debe aceptar el parámetro `org.springframework` que filtrará automáticamente con base en la paginación.

Resultado en las referencias del API:

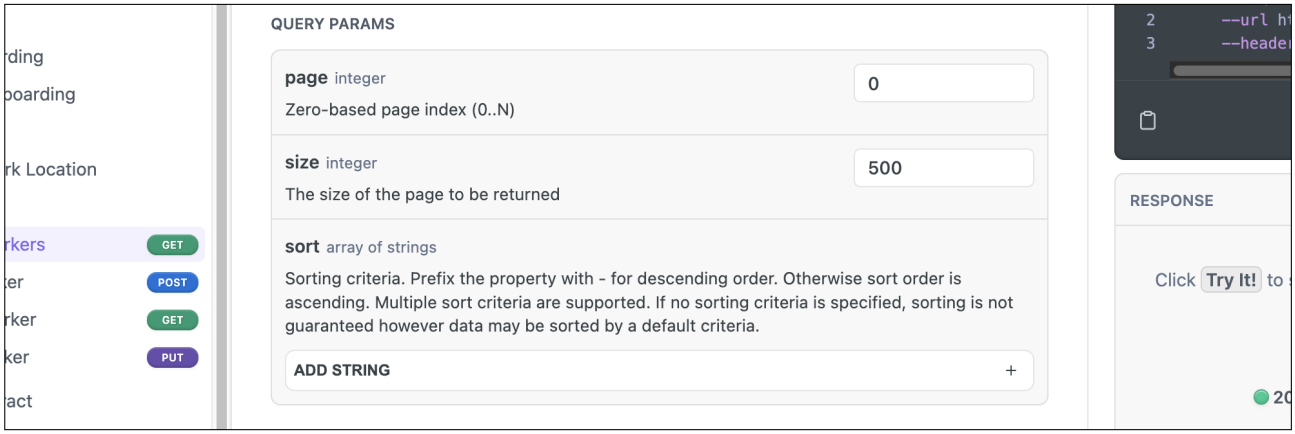


Figura 4.11: Añade los parámetros para la paginación.

Errores

La anotación `@SalsaErrorHandlerV1` manejará todas las excepciones estándar de Salsa. Para errores específicos hay dos alternativas:

1. `SalsaInternalException`: Para errores del lado del servidor donde desea fallar, pero no tiene una respuesta adecuada para el usuario. El sistema producirá un mensaje genérico agradable y la excepción se registrará.
2. `ResponseStatusException` Captura cualquier excepción que le interese en el `RestService` y arroje un `ResponseStatusException` con información relevante para el usuario. Los mensajes de error en estas excepciones deben ser amigables para el usuario y nunca revelar detalles de implementación.

Resultado en las referencias del API:

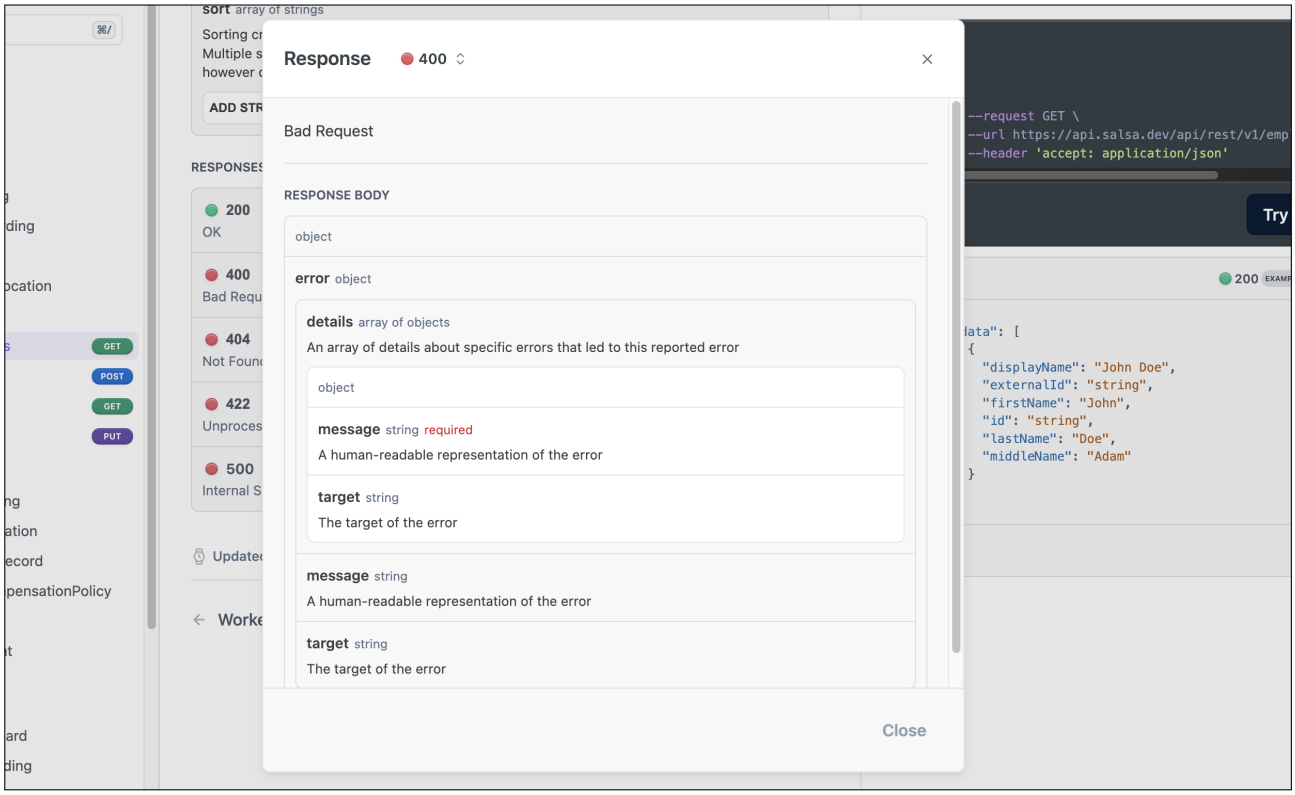


Figura 4.12: Añade el detalle de los errores del API.

Validaciones

Aquí hay dos opciones para validar los dominios de entrada (JSR 380 o Konform):

Anotaciones de JSR 380

Alguna de estas anotaciones producirán mensajes de error y generarán la documentación OpenAPI correcta. Verifique la salida del documento OpenAPI para asegurarse de que no necesita anotaciones OpenAPI adicionales.

```
data class TestRestInput(
    @get:Size(max = 10)       (1)
```

```
    val strVal: String?
)
```

1. Añade cualquier anotación JSR-380 a sus clases de datos usando `@get:` para hacerlo compatible con el código Kotlin.

También puedes agregar la anotación `@Valid` en tu operación REST:

```
@PostMapping("/test")
suspend fun getTest(
@Valid @RequestBody request: TestRestInput):  (1)
                        RestApiEnvelope<String> {
    return RestApiEnvelope(data = request.strVal)
}
```

JSR-380 muestra un error a la vez; al corregir un problema, se mostrará el error subsiguiente, si está presente.

Konform

A pesar de que Konform implica una configuración más compleja que involucra un objeto de validaciones y llamadas de validación en el RestService, proporciona una expresividad mejorada e informa todos los errores de entrada simultáneamente.

Se debe tener en cuenta que puede ser necesario una capa adicional de validación REST si los nombres de campo en las validaciones del negocio no se alinean con los de la entrada REST.

Resultado en las referencias del API:

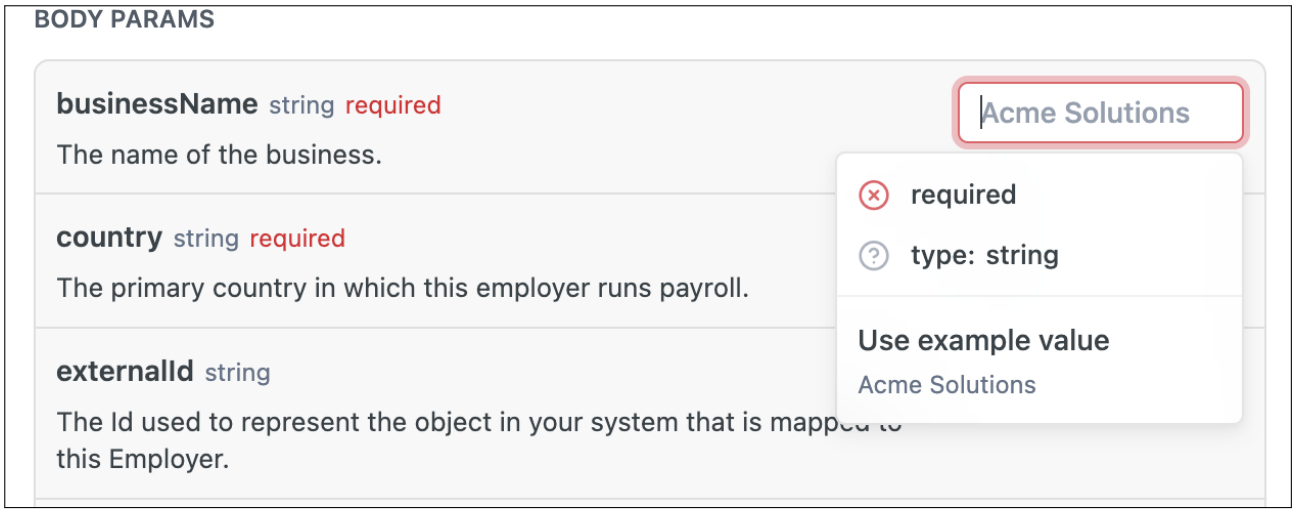


Figura 4.13: Muestra las validaciones que hace el API sobre los parámetros

Polimorfismo

Use el `@JsonTypeInfo` para aceptar entradas de diferentes tipos:

```
@JsonTypeInfo(                              (1)
    use = JsonTypeInfo.Id.NAME,
    include = JsonTypeInfo.As.EXISTING_PROPERTY,
    property = "type")
@JsonSubTypes(
    value = [
        JsonSubTypes.Type(value
        = Option1RestInput::class, name = "Option1RestInput"), (2)
        JsonSubTypes.Type(value
        = Option2RestInput::class, name = "Option2RestInput")
    ]
)
sealed interface MyBaseRestInput {
    val type: String
}

data class Option1RestInput(
    // Tipo de campo para la documentación de la API
    // abierta para que coincida con el tipo de
    // propiedad JsonTypeInfo
    @Schema(description = "Input type",                (3)
            allowableValues = ["Option1RestInput"],
            required = true)
    override val type: String = Option1RestInput::class.simpleName
) : MyBaseRestInput

data class Option2RestInput(
    // Tipo de campo para la documentación de la API
    // abierta para que coincida con el tipo de
    // propiedad JsonTypeInfo
    @Schema(description = "Input type",
            allowableValues = ["Option2RestInput"],
            required = true)
    override val type: String = Option2RestInput::class.simpleName
) : MyBaseRestInput
```

1. Define el campo discriminador usando `type` como nombre.
2. Use las clases como valores del discriminador.
3. Crea un nuevo campo opcional `type` en tus implementaciones para que puedas agregar la anotación `@Schema` y documentar el campo.

Resultado en las referencias del API:

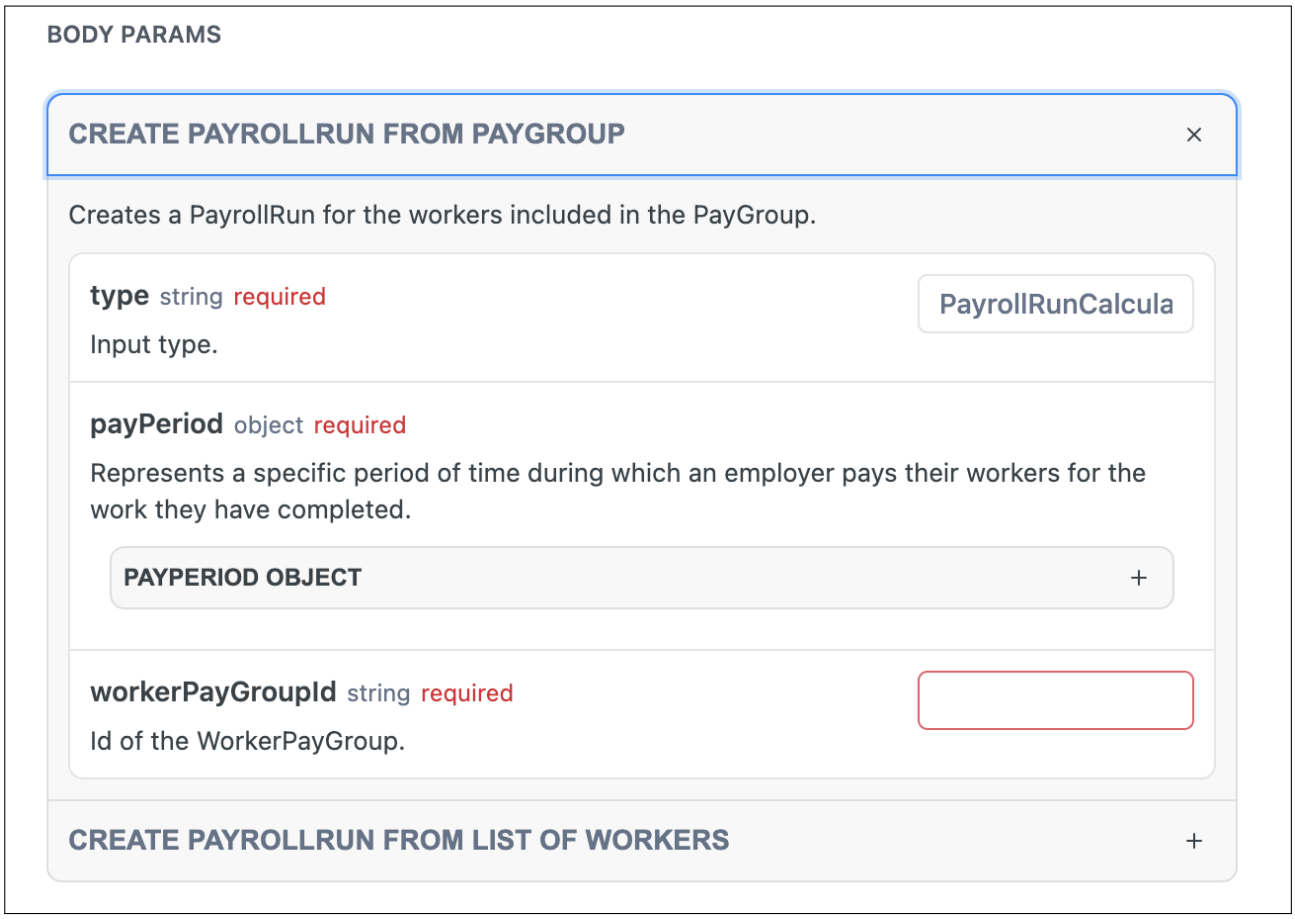


Figura 4.14: Habilita la posibilidad de solo escoger una opción en el cuerpo de la petición

Pruebas automáticas

El código de la definición de la API REST debe ser en su mayoría declarativo, y en general solo debe pasar la información correcta al puerto correcto. Por esta razón, solo queremos probar las happy paths en el nivel de la API REST.

Para crear casos de prueba de API REST, use el ayudante `SalsaWebTestClient` para invocar operaciones REST:

```
@SpringBootTest(webEnvironment =
                SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient(timeout = "PT30S")
class DomainRestControllerApiSpec(
    private val salsaWebTestClient: SalsaWebTestClient
) : ReactiveNeo4JRepositoryFunSpec({
    // Probar solo los casos de uso.
})
```

– Usar las operaciones `get` y `post` para obtener el cuerpo de la respuesta y luego realizar las aserciones de kotest contra él:

```
val responseBody = salsaWebTestClient.get(token,uri,
```

```
                    WorkerRestApiResponse::class.java)
responseBody.error shouldBe null
with(responseBody.data!!) {
    id shouldNotBe null
    firstName shouldBe "Firstname"
}
```

Para la lógica de transformación compleja, considere la posibilidad de introducir pruebas unitarias para cubrir los escenarios que no están cubiertos por las pruebas de API del happy paths.

- Evita probar el framework.
- Evita probar que las anotaciones funcionen.
- Evita probar la lógica de negocio del dominio.
- Enfócate principalmente en asegurarte de que la API esté conectada correctamente.

Documentación OpenAPI

Todos los endpoints, campos y entidades deben ser documentados correctamente.

Utilice la documentación para explicar cómo funciona la plataforma.

Debe documentar los diferentes elementos

Respondiendo a las siguientes preguntas:

En operaciones GET:

¿Qué devuelve esta operación?

```
@Operation(summary=‘‘Devuelve/Recupera...‘‘)
```

En operaciones POST/PATCH:

¿Qué hace esta operación?

```
@Operation(summary=‘‘Crear/Actualizar/Calcular...‘‘)
```

Resultado en las referencias del API:

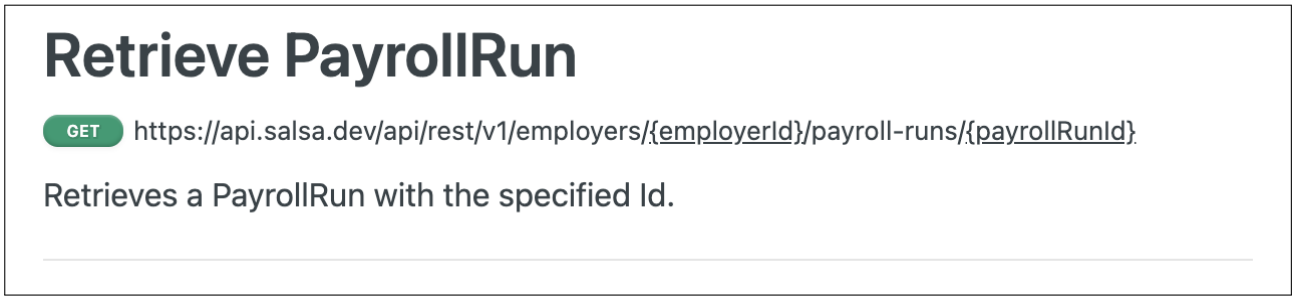


Figura 4.15: Muestra título en "@Operation(summary=“Retrive PayllRun“)"

Descripción de entidad:

¿Qué representa, para qué se utiliza, qué proporciona?

```
@Operation(description=‘‘Representa.../ Se utiliza cuando.../ Proporciona...‘‘)
```

Resultado en las referencias del API:

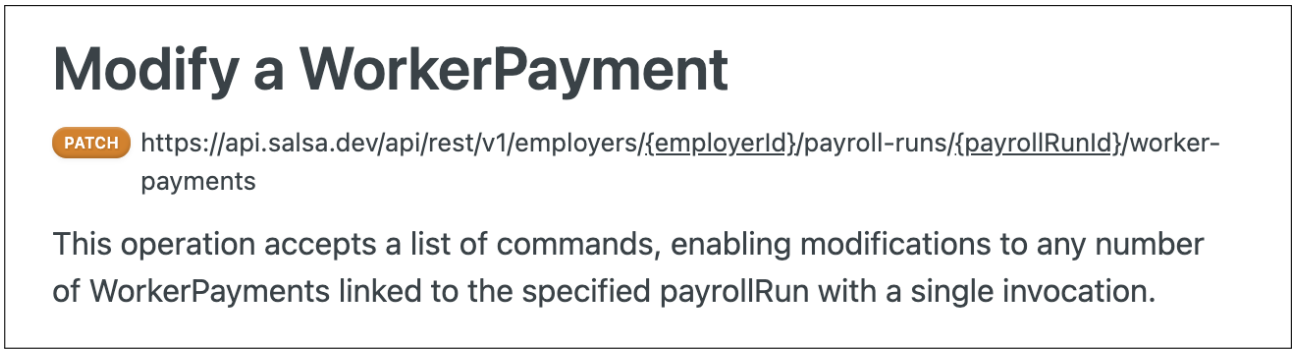


Figura 4.16: Muestra el texto en "@Operation(description=“This operation accepts...“)"

Tiempo verbal

Para cada operación, debemos ser consistentes con la forma en que redactamos el título y descripción de la operación en términos de tiempo verbal.

— Para el **resumen** usar el **futuro**. La manera de pensar en esto es responder a la pregunta: “¿Qué **hará** esta operación? “

- Ejemplo: ‘Crear un nuevo Trabajador‘

— Para la **descripción**, usar el **presente**. La manera de pensar en esto es responder a la pregunta: “¿Qué **está haciendo** esta operación? “

- Ejemplo: ‘Crea un nuevo Trabajador dentro del Empleador dado.‘

Tono explícito

Evite asumir conocimientos previos si no están cubiertos en la documentación; en su lugar, aclare la justificación detrás de la solicitud de datos particulares, como al describir una categoría:

- “Identifica si un trabajador es un empleado o un contratista“ (Mejorable)

Vs

- “Indica el tipo de trabajador. Diferentes tipos, como contratistas y empleados, están sujetos a regulaciones fiscales distintas.“ (Ideal)

Resultado en las referencias del API:

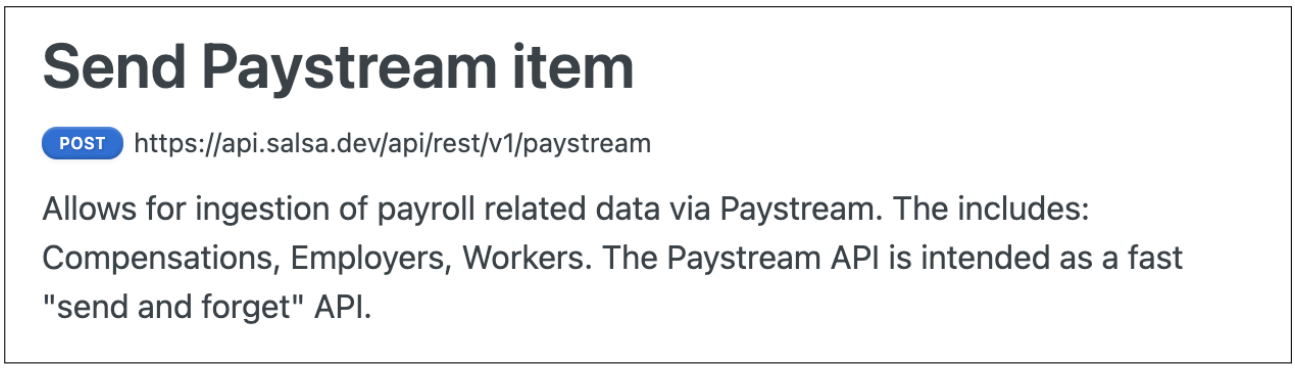


Figura 4.17: "The includes: Compensations, Employers, Workers"Provee información que probablemente el lector no sabe.

Punto de vista

Al escribir descripciones, utiliza el punto de vista en segunda persona cuando sea relevante. Esto significa que debes dirigirte al desarrollador como “tú“ y asumir que el lector es el usuario colaborador (desarrollador) que está realizando las tareas que se están documentando.

Por ejemplo:

“El identificador utilizado para representar este Empleador en el sistema externo“. (Mejorable)

VS

“Utiliza el identificador para representar el objeto en tu sistema y así asignar a este Empleador“. (Ideal)

Ejemplos

Incluye ejemplos cada vez que puedas. Ya que, las herramientas que vamos a usar para interpretar la definición OpenAPI pueden brindar mock-server partiendo de estos ejemplos.

Resultado en las referencias del API:

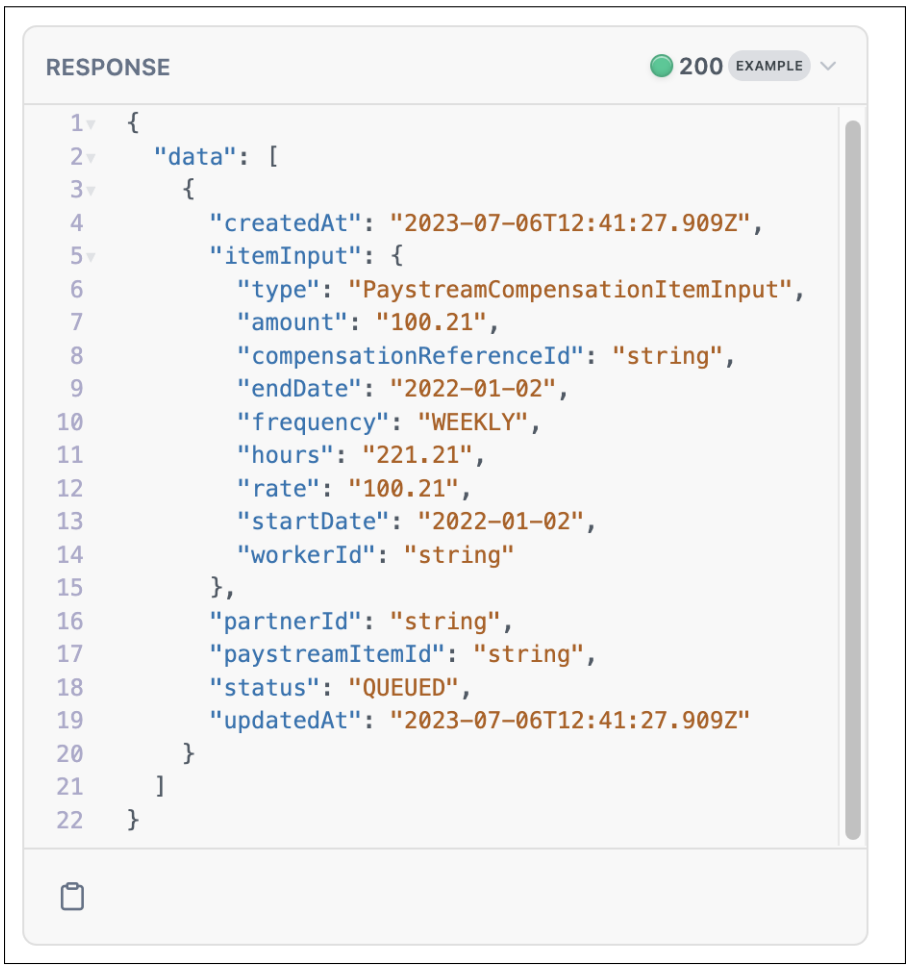


Figura 4.18: Ejemplo de la respuesta de “Send Paystream item“

4.3. Entrega rápido y vencerás

En un correcto flujo de desarrollo existen diferentes prácticas consideradas como fundamentales y el despliegue continuo y la entrega continúa (CICD –Continuous Integration and Continuous Delivery en inglés) es una de ellas. En el ciclo de desarrollo del software se incluye esta práctica para agilizar el tiempo que se gasta entre la realización de una nueva característica del sistema y la puesta en producción.

En esta sección se va a mostrar los resultados obtenidos después de haber incluido en el proceso de CICD la acción de publicar automáticamente la especificación OpenAPI que anteriormente se describió en este trabajo.

Salsa ya contaba con un buen proceso de CICD, como ya se explicó en el ciclo de vida del desarrollo de software, es de suma importancia contar con ello. En este caso, las acciones de construcción, etiquetado y despliegues del software estaban basados en **GitHub actions** [2.2]. Por lo que se propuso crear una nueva acción que fuera capaz de generar la especificación y posteriormente publicarla en ReadMe.

Alineado con las buenas prácticas del software ReadMe nos da la posibilidad de publicar la especificación OpenAPI desde un entorno en la nube como lo es GitHub actions.

Aprovechando esta ventaja se creó una acción que recibiera “o no“ la versión de la API desplegada, dependiendo de eso generaría la especificación del API a partir de una **“tag“** o una rama. En el primer caso, se corresponde con a la llamada de “máquina

a máquina“ desde la acción de DEPLOY y en la segunda casuística con una llamada manual desde la interfaz de GitHub que indica la rama que queremos publicar. Con estos pequeños cambios, el software del API fue capaz de entregar manual y automáticamente la especificación rápidamente.

Esquema del flujo de publicación

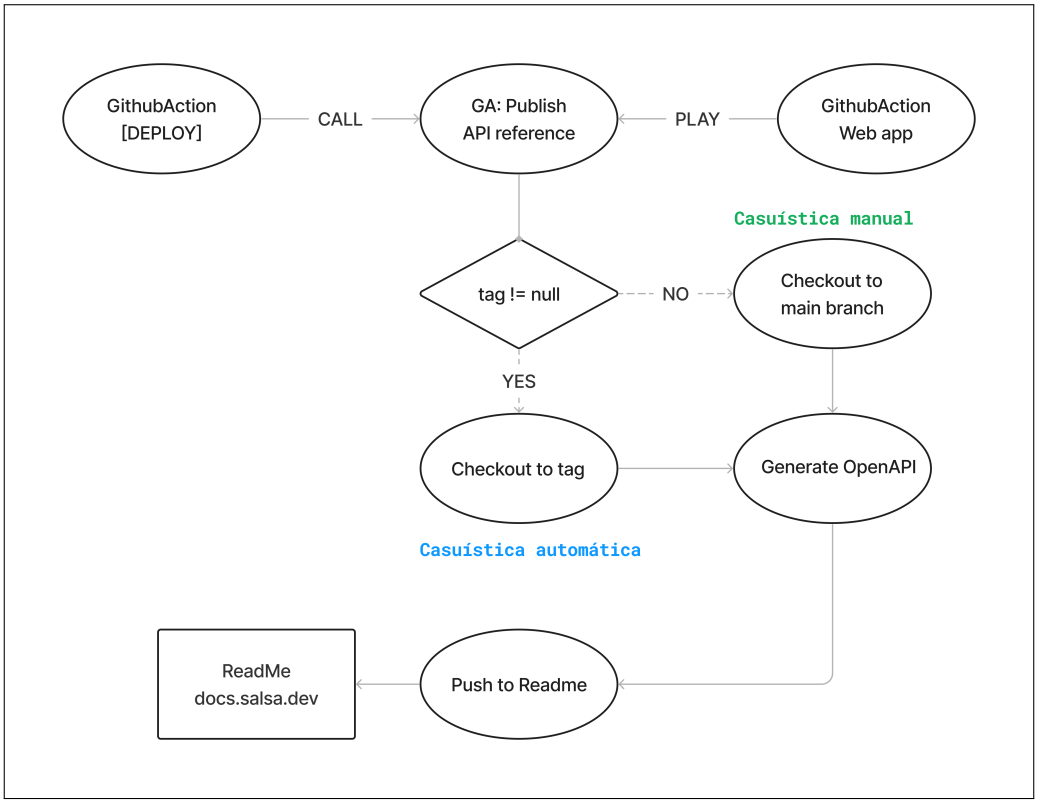


Figura 4.19: Flujo para publicar manual y automáticamente la definición OpenAPI

De esta forma, en cada despliegue del API se comenzó a entregar la definición del API automáticamente en ReadMe.

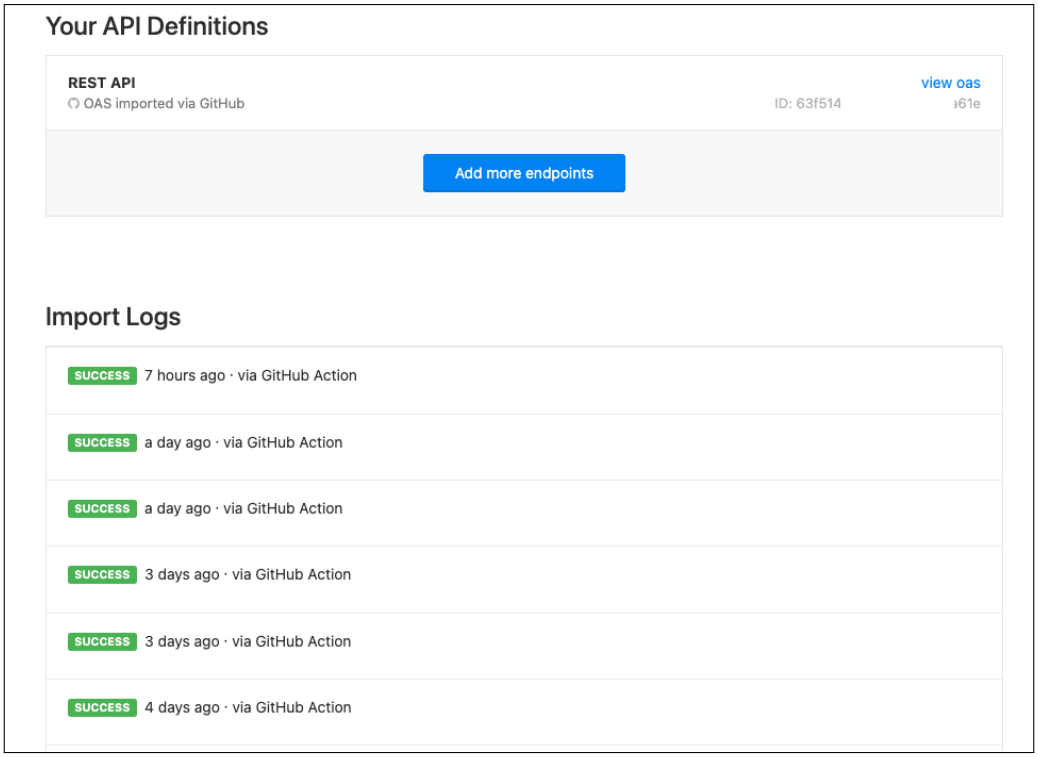


Figura 4.20: ReadMe: Registro de entregas

4.4. Evaluación comparativa

En este punto ya tenemos definidos los cuatro tipos de documentos que definirán la documentación del APIs. Ahora, orientado por los objetivos de este TFM, se debe evaluar y escoger un sistema que pueda servir la documentación que se desea especificar.

En esta sección se mostrarán los resultados de dos iteraciones para buscar, evaluar y comparar las herramientas disponibles en el mercado. En la primera iteración, se buscó y evaluaron las tecnologías de manera **no exhaustiva**. En la segunda iteración, se seleccionaron las tres opciones más prometedoras para ser evaluadas y comparadas **exhaustivamente**; tomando como criterio las características más relevantes y de mayor valor para la empresa. Para ello, se identificaron de manera detallada cada propiedad a partir de los datos cualitativos obtenidos en las entrevistas y se evaluaron en consecuencia.

4.4.1. Primera iteración

Como resultados de esta parte se usó la evaluación comparativa entre las seis pruebas de concepto. Los resultados mostraron que tanto **ReadMe, Docusaurus y Redocly** soportaban muy bien el concepto “portal de desarrolladores“, mientras que el resto, no. Estos ofrecían una experiencia no muy buena de base, como por ejemplo: flujos de desarrollo complicados de entender y por consecuencia difíciles de integrar en el proceso de desarrollo, poca autonomía para modificar la estructura de la documentación y pocas posibilidades de automatización. *ReadMe, Docusaurus, Stoplight y Redocly*, ofrecían una

documentación detallada y orientada a perfiles técnicos, mientras el *Theneo y GitBook* estaban más pensados para perfiles NO-CODE y esto parece que fuera positivo, pero en la toma de decisión se optó por descartar a estas dos herramientas por no presentar una buena documentación y no resolver el problema de la documentación para API REST completamente.

Stoplight, por otra parte, demostró ser una herramienta potente con buen soporte y una experiencia de desarrollo buena, pero fue descartada porque frente a sus competidores la configuración inicial era compleja y demandaba mucho esfuerzo de la empresa para conseguir resultados similares que el resto de herramientas de entrada ya ofrecían. es decir, de alguna forma Stoplight fue descartada no por ser mala opción, sino que sus competidores resolvieron mejor el factor “Plug and Play“.

En resumen, en la primera iteración se escogieron ReadMe, Docusaurus y Redocly por ser de las opciones que satisfacían de alguna manera todos los criterios propuestos para esta primera parte. y se descartaron GitBook, Theneo y Stoplight por no llegaron a ofrecer todos los criterios.

4.4.2. Segunda iteración

Tras investigar más sobre las características de los portales de desarrolladores en el mercado y analizar en los datos cualitativos de las entrevistas, se definieron los siguientes criterios fundamentales, los cuales son de importancia para las conclusiones de este trabajo.

En esta sección se muestra una tabla de comparación al final de cada descripción de las características principales o features más importantes para la documentación de Salsa: *Gestión de documentos, referencia del API, editor de guías, gestor de categorías, personalización, seguridad y analítica, monitorización y soporte técnico.* –Importante: En cada apartado que representa una característica y seguidamente se van desglosando los factores evaluados en las POCs de ReadMe, Docusaurus y Redocly.

Gestión de documentos

Los documentos o guías de usuario son recursos que brindan instrucciones y orientación sobre cómo utilizar los diferentes recursos del API; son la base de conocimiento para los clientes de la empresa. La posibilidad de gestionar estos recursos brinda agilidad a la hora de crear y administrar guías o instrucciones.

Características evaluadas:

Versionado

Permite realizar un seguimiento de los cambios en las guías, permitiendo a los usuarios acceder a versiones antiguas si es necesario.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Internacionalización (i18n)

Permite internacionalizar las guías en varios idiomas.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Búsqueda de texto

Permite a los usuarios buscar en las guías por texto, lo que facilita la búsqueda de guías. Comúnmente se representa con un input de texto.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Exportar PDF

Permite a los usuarios descargar guías en formato PDF para acceder sin conexión o imprimir.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Ciclo de vida del artículo

Permite a los editores saber si la guía ha sido publicada, si necesita revisión o está en estado de borrador.

Evaluación:

- **[TAGS] Basado en etiquetas:** Permitir agregar etiquetas al estado (antes o después de la publicación).
- **[GIT] Basado en Git:** Usando Git para controlar cuándo se debe publicar.

Metadatos de la guía

Permite la personalización de la configuración específica, como el título, la descripción, la URL, las etiquetas y la imagen de portada personalizada.

Evaluación:

- **[AUTO] Automático**: Recuperación de la configuración desde sí mismo.

- **[GLOBAL] Configuración global**: Quieres decir que es una solución global utilizando rutas.
- **[ON-FILE] Definido en cada archivo**: Permitir definir encabezados al comienzo.
- **No**: Quiere decir que NO es posible su configuración.

Plantillas

Proporciona plantillas prediseñadas facilitando a los editores la creación de las guías con formato consistente.

Evaluación:

- **[AUTO] Automático**: Quieres decir que las plantillas almacenadas en su plataforma.
- **[DS] Creadas desde un sistema de diseño**: Quieres decir que las Plantillas desde un sistema de diseño.
- **[MODEL-FILE] Creadas desde archivo modelo**: Quieres decir que se crean sus propias plantillas.

Flujos de trabajo

Flujos de trabajo son procesos específicos que mejoran, validan y automatizan la entrega de las guías, básicamente se refiere a los pipelines de las guías.

Evaluación:

- **[AUTO] Administrado por la herramienta**: Quieres decir que la revisión e integración continua es implementada por la herramienta.
- **[ON-PREMISES] Administrado por la empresa**: Quieres decir que la integración continua es administrada internamente por la empresa, en lugar de depender de un servicio de terceros.

Plataforma de discusión

Una plataforma de discusión permite la colaboración entre editores del equipo para escribir o revisar guías publicadas o en borrador. Es similar a la revisión de código.

Evaluación:

- **[GIT] Basado en Git**: Usando Git para comentarios y revisiones en Pull requests.
- **[PLATFORM] Ediciones sugeridas**: Ofrece plataforma o panel para comentar y hacer revisiones.
- **[PLATFORM + GIT] Ediciones sugeridas + Git**: Usa Git para solicitar mejoras sugeridas con un enlace a las guías.

Tabla comparativa

Característica	ReadMe	Redocly	Docusaurus
Versionado	Sí	SDT	SDT
Ubicación	Sí	Sí	SDT
Búsqueda de texto	Sí	Sí	SDT
Exportar PDF	Sí	No	SDT
Ciclo de vida del artículo	TAGS	GIT	GIT
Metadatos guía	AUTO	GLOBAL	ON-FILE
Plantillas	AUTO	DS	MODEL-FILE
Flujos de trabajo	AUTO	AUTO	ON-PREMISES
Plataforma de discusión	PLATFORM	GIT	PLATFORM + GIT

Tabla 4.1: Tabla comparativa: Gestión de documentos

API reference o referencia del API

Es la documentación detallada que proporciona información sobre las diversas funcionalidades, métodos, parámetros y estructura de la API.

Ejemplo de las Respuestas

Esta función permite a los desarrolladores anticipar la estructura de las respuestas, ahorrando tiempo y asegurando la consistencia en todas las respuestas.

Ejemplo de las peticiones o request

Esta función permite a los desarrolladores comprender rápidamente la estructura de la request, asegurando la consistencia en todas las peticiones a la API.

Fragmentos de código en diferentes lenguajes

Esta función permite mostrar fragmentos de código en múltiples lenguajes de programación.

API Playground

Permite a los usuarios ejecutar directamente solicitudes a la API sin el uso de software de terceros.

Búsqueda por texto

Permite a los usuarios buscar en la API utilizando texto, facilitando la búsqueda de recursos específicos.

Definiciones de recursos

Define los endpoints disponibles y su funcionalidad. Proporcionan información sobre parámetros, respuestas esperadas y cualquier información adicional que pueda ser relevante para realizar las peticiones.

Guía de autenticación

Esta guía explica cómo obtener acceso a recursos protegidos utilizando la API. Puede incluir instrucciones para obtener una clave de API, configurar OAuth u otros métodos de autenticación admitidos.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Funcionalidad	ReadMe	Redocly	Docusaurus
Ejemplo de las Respuestas	Sí	Sí	SDT
Ejemplo de las peticiones o request	Sí	Sí	SDT
F. código en diferentes lenguajes	Sí	Sí	SDT
API Playground	Sí	Sí	SDT
Búsqueda por texto	Sí	SDT	SDT
Definiciones de recursos	Sí	Sí	SDT
Guía de autenticación	Sí	SDT	SDT

Tabla 4.2: Tabla comparativa: API reference

NOTA: En esta parte Docusaurus tiene “SDT“en todas las opciones porque no soporta de forma nativa API reference. Pero es posible con SDT.

Editor de texto

Es una herramienta que brinda a los escritores la capacidad de crear y editar contenido de manera eficiente en una interfaz de texto.

Editor de texto

Su objetivo principal es permitir a los escritores enfocarse en el contenido en lugar de preocuparse por el formato.

Vista previa

Permite a los desarrolladores ver una vista previa de cómo se verá su contenido cuando se publique.

Evaluación:

- **[LOCAL] Entorno local**: Permite utilizar un entorno local.
- **[ONLINE] Entorno en línea**: Proporciona un entorno en línea.

Markdown

Markdown permite a los desarrolladores utilizar la sintaxis estándar de Markdown para dar formato al contenido.

MDX

El soporte **MDX** permite a los desarrolladores utilizar la sintaxis MDX dentro de su contenido.

Incrustar imágenes

Permite agregar imágenes a su contenido.

Bloque de código

Permite agregar fragmentos de código con resaltado de sintaxis.

Callouts o alertas

Permiten agregar cuadros informativos o callouts a su contenido.

Hipervínculos

La función de **hipervínculos** permite a los desarrolladores agregar enlaces a otros contenidos o recursos externos.

Tablas

Permite agregar tablas a su contenido.

Pantalla completa

Permite ver y editar el contenido en modo de pantalla completa.

Listas

Permite a los desarrolladores crear listas ordenadas o no ordenadas.

Guardado automático

La función de **guardado automático** guarda automáticamente los cambios realizados en el contenido.

Adjuntos de archivo

Permite a los desarrolladores adjuntar archivos a su contenido.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Funcionalidad	Rearme	Redocly	Docusaurus
Editor de texto	LOCAL + ONLINE	LOCAL	LOCAL
Vista previa	ONLINE	LOCAL	LOCAL
Markdown	Sí	Sí	Sí
MDX	No	Sí	Sí
Incrustar imágenes	Sí	Sí	Sí
Bloque de código	Sí	Sí	SDT
Callouts o alertas	Sí	Sí	Sí
Hipervínculos	Sí	Sí	Sí
Tablas	Sí	Sí	Sí
Pantalla completa	Sí	Sí	Sí
Listas	Sí	Sí	Sí
Guardado automático	Sí	SDT	SDT
Adjuntar archivos	No	No	NO

Tabla 4.3: Tabla comparativa: Editor de texto

Gestor de Categorías

La característica “Gestor de Categorías“ permite mantener y agrupar las guías en una estructura coherente, fácil de entender.

Crear categoría

Permite crear una categoría en la estructura jerárquica de la documentación.

Reordenar guías dentro de las categorías

Permite reorganizar el orden de los guías dentro de una categoría.

Ocultar categoría

Permite ocultar categorías para simplificar la navegación del usuario.

Control de acceso por nivel de categoría

Permite restringir el acceso a ciertas categorías solo a ciertos usuarios, lo que es útil para la documentación privada o confidencial.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.

Funcionalidad	ReadMe	Redocly	Docusaurus
Crear categoría	Sí	Sí	Sí
Reordenar guías dentro de las categorías	Sí	Sí	Sí

Funcionalidad	ReadMe	Redocly	Docusaurus
Ocultar categoría	Sí	No	Sí
Control de acceso por nivel de categoría	No	Sí	No

Tabla 4.4: Tabla comparativa: Gestor de categorías

Imagen corporativa o branding

Se refiere al conjunto de elementos visuales y simbólicos que representan la identidad y de la empresa o servicio. Es la forma en que la empresa se presenta al público y cómo desea ser percibida por sus clientes y empleados.

Nivel de personalización

Indica el grado de personalización posible con la función, con las opciones siendo alto, medio o bajo.

Evaluación:

- **Bajo**: No permite modificar los elementos y solo es posible cambiar el color corporativo y logo.
- **Medio**: Admite modificar los elementos a través del patrón theme.
- **Alto**: Permite sustituir los elementos por elementos de un sistema de diseño.

Página de inicio o landing

Se refiere a la página principal de la documentación.

Logotipo personalizado

Permite al usuario cargar y mostrar un logotipo personalizado en la documentación.

Navegación de encabezado / pie de página

Se refiere al menú de navegación que aparece en la parte superior o inferior de cada página en la documentación.

Tema o esquema de color

Permite al usuario seleccionar un esquema de color para la documentación que esté en línea con su identidad de marca.

Conexión con software de terceros

Se refiere a la capacidad de integrar la documentación con software de terceros para funcionalidad adicional.

Adopción de sistema de diseño

Se refiere a la capacidad de adoptar un sistema de diseño preexistente para la documentación, asegurando consistencia con otros productos o materiales.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que no posible.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Funcionalidad	ReadMe	Redocly	Docusaurus
Nivel de personalización	Bajo	Medio	Alto
Página de inicio o landing	Si	Si	Si
Logotipo personalizado	Si	Si	Si
Navegación de encabezado / pie de página	Si	Si	Si
Tema o esquema de color	No	Si	SDT
Conexión con software de terceros	No	No	Si
Adopción de sistema de diseño	No	No	Si

Tabla 4.5: Tabla comparativa: Imagen corporativa o branding

Seguridad y control

Permite proteger la documentación con roles definidos, restricciones y copias de seguridad.

Copia de seguridad y restauración

Permite realizar copias de seguridad de la documentación y restaurar versiones anteriores si es necesario.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **Parcialmente**: Quiere decir que NO es posible, pero de alguna manera se puede lograr.
- **[GIT] Git:** Quiere decir que se utiliza software de terceros para que sea posible.

Documentación privada

Permite restringir el acceso a la documentación solo a ciertos usuarios, lo que es útil para la documentación privada o confidencial.

Evaluación:

- **[No]**: Quiere decir que NO es posible.
- **[PASSWORD] Con clave**: Quiere decir que NO es posible.
- **[AUTH-PROVIDER] Admite proveedor de autenticación:** Quiere decir que se utiliza software de terceros para que sea posible.

Roles

Permite asignar roles y permisos personalizados a miembros del equipo, restringiendo el acceso a ciertas secciones de la documentación.

Exportación e importación de documentos

Permite exportar e importar la documentación para transferirla entre diferentes plataformas o para hacer una copia de seguridad.

Proveedor de autenticación

Permite cumplir con los proveedores de autenticación al proporcionar un inicio de sesión seguro y protegido.

Dominio personalizado

Permite personalizar el dominio de la documentación para que se ajuste a la identidad de marca de la organización.

Redirección

Permite redireccionar a los usuarios a otra página o sitio web si intentan acceder a una página que no existe.

Restricción de IP

Permite restringir el acceso a la documentación solo a ciertas direcciones IP, lo que es útil para restringir el acceso a la documentación solo a la red interna de la organización.

Consentimiento de cookies

Permite obtener el consentimiento del usuario para el uso de cookies en la documentación.

SSO empresarial

Permite utilizar un sistema de autenticación único (SSO) para proporcionar acceso a la documentación a los empleados de la organización.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que NO es posible
- **Parcialmente**: Quiere decir que NO es posible, pero de alguna manera se puede lograr.
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Funcionalidad	ReadMe	Redocly	Docusaurus
Roles	No	No	Sí
Copia de seguridad y restauración	Sí	GIT	GIT
Exportación e importación de documentos	Sí	Sí	Sí
Proveedor de autenticación	No	No	Sí
Documentación privada	PASSWORD	No	AUTH-PROVIDER
Dominio personalizado	No	Sí	Sí
Redirección	Sí	No	Parcialmente
Restricción de IP	Sí	Sí	Parcialmente
Consentimiento de cookies	Sí	Sí	Sí
SSO empresarial	No	SDT	SDT

Tabla 4.6: Tabla comparativa: Seguridad y control

Analítica, monitorización y soporte

Con estas características se ayuda a comprender el compromiso de los usuarios finales y tomar decisiones basadas en datos.

Ubicación geografía

Proporciona información sobre la ubicación geográfica de los usuarios que acceden a la documentación.

Rendimiento

Monitoriza el rendimiento de la plataforma de documentación para asegurar una experiencia de usuario óptima.

Panel de búsqueda

Facilita la visualización de métricas en búsquedas, usuarios y palabras clave de búsqueda sin resultados.

Comentarios

Proporciona una forma para que los usuarios dejen comentarios sobre la documentación.

Estado de enlaces

Monitoriza el estado de los enlaces dentro de la documentación para asegurarse de que funcionen correctamente.

Página no encontrada

Proporciona una página de error personalizada cuando un usuario intenta acceder a una página que no existe.

SEO

Optimiza la documentación para motores de búsqueda para mejorar la visibilidad y accesibilidad.

Evaluación:

- **Sí**: Quiere decir que es completamente posible.
- **No**: Quiere decir que NO es posible
- **[SDT] Software de terceros:** Quiere decir que se utiliza software de terceros para que sea posible.

Nombre	ReadMe	Redocly	Docusaurus
Ubicación geografía	Sí	SDT	SDT
Rendimiento	Sí	No	SDT
Panel de búsqueda	Sí	Sí	SDT
Comentarios	No	No	SDT
Estado de enlaces	No	No	SDT
Página no encontrada	Sí	Sí	Si
SEO	Sí	Sí	Sí

Tabla 4.7: Tabla comparativa: Analítica, monitorización y soporte

NOTA: En la siguiente sección se evaluarán los datos obtenidos de todas estas tablas.

4.5. La herramienta más eficiente para publicar la documentación

En esta sección se plantean dos soluciones contando con las ventajas y desventajas de cada una de las herramientas; La primera como solución a corto plazo y adicionalmente se ofrece una alternativa. La segunda como solución a largo plazo.

4.5.1. ReadMe *como solución a corto plazo*

Ventajas

1. ReadMe demostró ser la herramienta más rápida para crear la referencia del API, tutoriales y guías, por mucho.
2. Ofrece todos los beneficios del Software as a Service (SaaS) como por ejemplo: entrega automática, alojamiento de la aplicación (hosting) y acceso privado.
3. Proporciona soporte técnico completo y de pago.
4. Permite la revisión colaborativa (edición y control cambios) del contenido.
5. Incluye un editor en línea, lo que la hace una herramienta NO-CODE[27] ideal para cualquier perfil de la empresa (Desarrolladores o no).

6. Tiene una estructura simple y requiere menos mantenimiento que el resto de opciones.
7. Admite Integración Continua y Despliegue Continuo (CICD) a través de acciones de GitHub.
8. Ofrece integración vía API e interfaz de línea de comandos (CLI) desde la nube.
9. Ofrece un sistema de fácil versionado de las referencias de API.
10. Actualmente, es utilizada para documentar el API de Notion[9], Scale[16] y Miro[8].

Desventajas

1. Baja capacidad para cambiar el aspecto y personalizar los elementos de la documentación.
2. La estructura y organización del contenido no es escalable, por lo que no extiende la posibilidad de mejorarla.
3. El acceso privado es solo para usuarios de pago (Cuesta 99 dólares americanos al mes).
4. No soporta MDX; aunque puede exportar la documentación en MD.
5. En el caso de implementarla en Salsa[15] algunas empresas competidoras también usan ReadMe lo que supone una desventaja en términos de innovación.

4.5.2. Redocly *como alternativa a corto plazo*

Ventajas

1. Ofrece una aplicación del estilo "Portal de desarrolladoresçomo punto de partida.
2. Posibilidades de desplegar el "Portal de desarrolladoresçon unos pocos pasos.
3. El código del "Portal de desarrolladores"debe estar alojado en un controlador de versiones como por ejemplo GitHub; lo que ofrece beneficios para la revisión y control de cambios.
4. Proporciona soporte técnico completo y de pago.
5. Ofrece un sistema de fácil versionado de las referencias de API.
6. Soporte de MDX de forma nativa.
7. Mediana capacidad para cambiar el aspecto y personalizar elementos a través de un tema (theme).

Desventajas

1. El "Portal de desarrolladores"está disponible desde la versión PRO que cuesta 300€.
2. Su implementación al principio puede ser dura; ya que hablamos de un framework 2.2.
3. El editor de texto disponible es a través de la interfaz de usuario de GitHub o un editor de texto local; lo cual añade complejidad para perfiles de usuario sin conocimientos de programación (NO-CODE).[27].
4. El proveedor de elementos MDX solo acepta elementos de su catálogo, lo que significa que la apariencia sigue siendo poco flexible.

4.5.3. Docusaurus *como solución a largo plazo*

Ventajas

1. La documentación generada tiene un aspecto moderno, minimalista y fácil de entender.
2. Es un proyecto Open Source [2.2] lo que significa que no requiere pagar por una licencia de uso.
3. Tiene completo soporte de MDX y permite declarar elementos React [2.2] personalizados.
4. Sigue el patrón de “Plugins“ lo que permite extender sus funcionalidades casi de forma ilimitada.
5. Genera una aplicación de código estático con rutas y componentes de navegación.
6. Es utilizado para exponer la documentación de Redux [14], Supabase [19] y Dyte [4]

Desventajas

1. El API reference 2.2 no es soportado de forma nativa; lo que supone la dependencia de un plugin.
2. Todo el proceso de publicación es on-premise.
3. Los plugins no tienen soporte de pago. Lo que no garantiza el mantenimiento a largo plazo.
4. Los plugins en algunos casos de uso pueden ser escasos.
5. Su implementación y configuración puede ser dura al principio porque estamos hablando de un framework 2.2.
6. El editor de texto puede ser a través de GitHub o un editor de texto local, lo cual añade complejidad para perfiles de usuario sin conocimientos de programación (NO-CODE).[27].

Finalmente y en forma de complemento a los resultados de este trabajo, se añade el apéndice E para mostrar capturas de pantalla del portal de desarrolladores de Salsa.

Capítulo 5

Discusión

5.1. Resolviendo interrogantes de la investigación

- ¿Cuál es la clave para generar una documentación de calidad?
    - Según los resultados obtenidos, no existe una “clave“, pero es evidente que cuidar los detalles en la documentación aumenta significativamente su calidad. Contar con un marco de clasificación para los documentos y comprender que cada uno cumple una función específica en la documentación también influye en su calidad. A menudo se piensa que el desarrollo de software y la documentación están completamente separados, pero esto no es cierto, aprendimos que deben convivir. En los resultados, se observó el enorme esfuerzo requerido para implementar en el software del API los criterios que garantizaran una visualización coherente en la referencia técnica (API reference). En resumen, los detalles en el proceso de desarrollo tienen un impacto positivo o negativo en la calidad de la documentación.
- ¿Existen soluciones en el mercado que aborden las deficiencias identificadas en la documentación técnica?
    - Como es evidente, en el mercado actual existen diversas soluciones que ofrecen enfoques y herramientas para mejorar la calidad y eficiencia de la documentación técnica en las startups. Estas soluciones abordan desafíos comunes, como la estructura y organización de la documentación, la colaboración entre equipos y la facilidad de uso para los usuarios. Al evaluar y seleccionar las soluciones más adecuadas para cada caso, las empresas pueden integrar u optimizar su proceso de documentación y brindar una experiencia mejorada a los desarrolladores y usuarios finales.
- ¿Cuáles son los factores más importantes que pueden agilizar el proceso de creación de la documentación técnica?
    - Disponer de una especificación OpenAPI que defina claramente los endpoints y las estructuras de datos utilizadas en la API. [4.2]
    - Establecer guías bien estructuradas según la tipología de la documentación, que proporcionen instrucciones claras y concisas para los desarrolladores. [4.1]

- Utilizar una plataforma o sistema como ReadMe, Redocly o Docusaurus, que permita representar y mantener actualizada la documentación de forma constante y ágil. 4.5

El factor clave para el éxito radica en la capacidad de integrar de manera efectiva estos elementos en el ciclo de vida del desarrollo de software, asegurando una comunicación fluida entre los equipos y una constante retroalimentación para mejorar la documentación en cada iteración.

- ¿Cómo podemos lograr los objetivos de mejora en un plazo de 3 meses en lugar de 6?

    - Siempre y cuando se cuenten con los factores clave mencionados anteriormente, se puede afirmar que es completamente posible alcanzar los objetivos establecidos en este trabajo en una iteración de corta duración. La experiencia adquirida con Salsa respalda esta afirmación, demostrando que tomar "la opción 1" 4.5.1 es viable y efectiva para lograr una implementación exitosa de la documentación técnica en el ciclo de vida del desarrollo de software. Con una planificación adecuada, una ejecución eficiente y el uso de herramientas y tecnologías apropiadas, se puede obtener un resultado satisfactorio en un tiempo reducido.

- ¿Cuáles son las mejores prácticas para mantener actualizada y relevante la documentación técnica a lo largo del ciclo de vida del producto?

    - La documentación técnica debe considerar las técnicas modernas del desarrollo de software, como la revisión de código, el seguimiento de cambios y el versionado.
    - Establecer los canales para retroalimentación entre los clientes y la empresa constantemente.
    - Integrar la Documentation técnica en el proceso de CI/CD.

- ¿Qué estrategias se pueden implementar para asegurar la colaboración efectiva entre los equipos de desarrollo y los escritores técnicos en el proceso de creación de la documentación técnica?

    - Definir estándares de estilo, estructura y formato para la documentación técnica es esencial. Esto asegura coherencia en el contenido y facilita su comprensión.
    - Emplear herramientas de colaboración como Notion, plataformas de gestión de proyectos como Linear, sistemas de control de versiones y editores colaborativos como ReadMe, facilita el trabajo conjunto.
    - Promover un proceso de revisión y retroalimentación entre los equipos de desarrollo y los escritores técnicos similar a la revisión de código, ayuda a mejorar la calidad de la documentación.
    - Es importante los roles y responsabilidades de cada equipo. Los desarrolladores deben proporcionar la información técnica necesaria en las referencias técnicas y tutoriales, mientras que los escritores no-técnicos se encargarán de darle forma y estructura a la documentación en términos de negocio, principalmente con los casos de uso y las explicaciones.

5.2. Métricas de visualizaciones

Aunque este estudio no se basa en datos cuantitativos, es importante resaltar que se ha observado un aumento significativo en las visualizaciones de la documentación, según las estadísticas proporcionadas por ReadMe. Estos datos respaldan la relevancia y el interés de los usuarios en acceder y utilizar la documentación para obtener información y orientación sobre el producto o servicio.

Las visualizaciones han llegado a tener picos de hasta 1526 visitas en los últimos 3 meses.

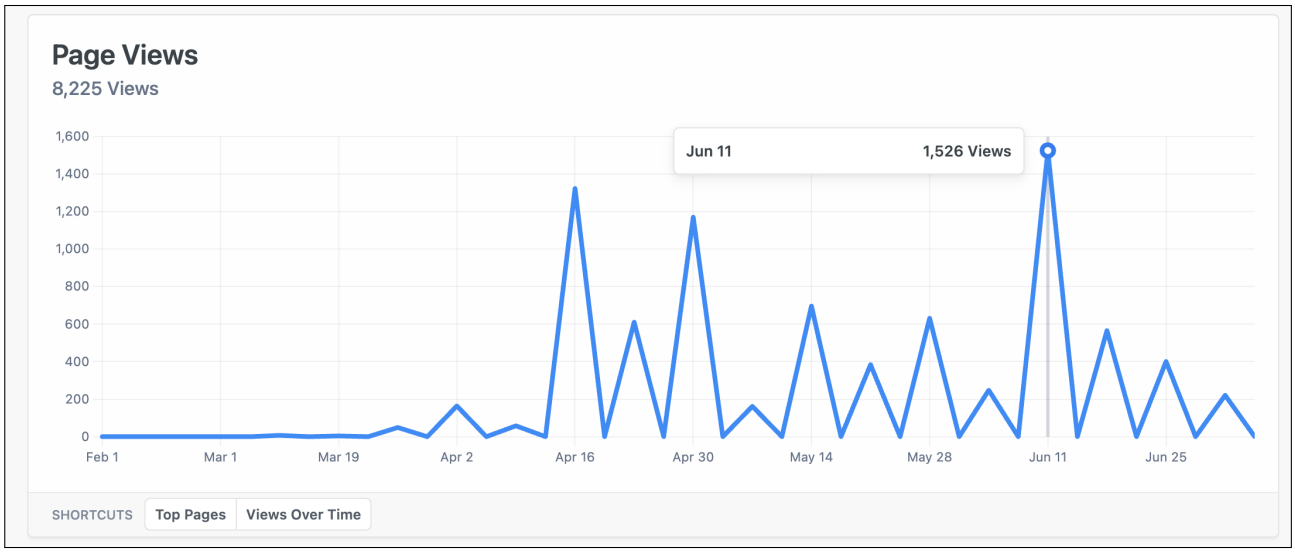


Figura 5.1: Estadísticas de uso: Tutoriales y casos de uso [Hasta 06/07/2023]

5.3. Implicaciones y aplicaciones prácticas:

5.3.1. Portales de documentación según su tipología

Después de varias semanas de investigación exhaustiva en la Evaluación Comparativa (Capítulo IV) 4.4, se ha evidenciado un patrón consistente en cada prueba de concepto (POC): cada tecnología ha demostrado un rendimiento superior en su respectivo escenario.

Es importante destacar que se han identificado tres tipos de soluciones tecnológicas para la documentación técnica de APIs como producto, las cuales abordan la misma problemática pero de manera diferenciada según los recursos y requisitos particulares. Esta clasificación se basa en la naturaleza intrínseca de cada solución, su entorno de aplicación y los responsables de su mantenimiento. A continuación, se presentarán de manera resumida las lecciones aprendidas durante este proceso de investigación.

Aplicaciones SaaS:

Según AWS, una solución tecnológica de tipo **SaaS (Software como servicio)** es un modelo de software basado en la nube que se ofrece a los usuarios finales a través de un navegador de Internet [21]. Este principio las hace idéales para empresas que necesitan avanzar rápido en la publicación de la documentación técnica. Es decir, cuando no se cuenta con mucho tiempo y conocimiento del dominio para desarrollar la solución.

Al tratarse de herramientas que normalmente son de pago como **ReadMe** [12] son sistemas de software más confiables porque cuentan con servicios de asistencia técnica (SAT [35]), planes de Garantía de nivel de servicio (SLA [22]) que disminuyen los problemas en producción en este tipo de soluciones tecnológicas. Además, evitan configuraciones complejas para su mantenimiento y mejoran el rendimiento sin sobre coste. Con respecto a la personalización de la imagen corporativa, son poco flexible y no ofrecen la oportunidad de innovar en experiencias de usuarios más complejas. Son capaces de controlar el acceso a su configuración y meta datos, haciéndolas muy atractivas para proyectos que necesitan salir rápido y no cuentan con muchos recursos.

Aplicaciones basadas en generadores de sitios estáticos:

Las aplicaciones basadas en generadores de sitios estáticos es el perfecto equilibrio entre las soluciones basadas en Software como servicio (SaaS)5.3.1 y las que son construidas completamente por la organización **??**. Son ideales para equipos medianos que tienen conocimiento del framework y que requieren una primera instancia como modelo para desde allí construir todo un proyecto.

Este tipo de soluciones tiene un punto de partida bien definido; como se ve en los casos de **Redocly**[13] o **Docusaurus**[3]. Estos generan una aplicación que puede crear rutas de navegación a partir de ficheros MD o MDX con poco esfuerzo. Además, permiten una buena flexibilidad para definir y representar los componentes de la documentación.

Es importante tener en cuenta que los proyectos basados en generadores de sitios estáticos requieren un cierto conocimiento de programación, lo que dificulta su uso en las primeras etapas. Sin embargo, una vez que los desarrolladores se han familiarizado con el framework 2.2, ofrecen una gran ventaja a la hora de implementar la documentación técnica.

Aplicaciones In-house Built:

In-house Built es un enfoque del desarrollo de software que enfatiza las habilidades de producir código desde los propios desarrolladores de software de la organización. Las herramientas de este tipo son fabricadas desde cero, pasando por todas las etapas de definición y diseño. Y usualmente son implementadas por los mismos desarrolladores que las diseñaron; estos proyectos de documentación son los más costosos en tiempo y esfuerzo. Las aplicaciones basadas en soluciones como **Markdoc**[6], **CodeHike**[2] o **Stoplight Elements** requieren primeramente de una base donde integrar, la completa configuración del entorno, despliegues y mantenimiento del software. Ofrecen gran flexibilidad para crear casi cualquier experiencia de usuario. Por lo que las hace ideales para productos con requerimientos complejos y que cuentan con tiempo suficiente para implementar este tipo de soluciones. No son nada recomendadas para equipos sin experiencia en el dominio de documentación de APIs.

5.3.2. ¿Existen alternativas para OpenAPI?:

Sí, si existen alternativas para OpenAPI. Las principales son RAML [11] y API Blueprint [1].

RAML (RESTful API Modeling Language): RAML es un lenguaje de modelado de API que permite describir las API RESTful. Proporciona una sintaxis simple y legible que facilita la comprensión y la colaboración entre los equipos de desarrollo y documentación.

API Blueprint: es otro formato de descripción de API que utiliza una sintaxis en formato Markdown. Permite describir la estructura y los detalles de las API de manera clara y concisa, y se puede convertir fácilmente en documentación legible y amigable para los desarrolladores.

Algunas diferencias comparativas discutidas en el proyecto:

1. **Sintaxis y formato:** OpenAPI utiliza JSON o YAML para describir las especificaciones de la API, mientras que RAML utiliza su propio formato basado en YAML y API Blueprint utiliza Markdown. La elección del formato puede depender de la familiaridad y preferencias del equipo de desarrollo.

2. **Compatibilidad de herramientas:** OpenAPI tiene una amplia compatibilidad con diversas herramientas y frameworks populares, lo que facilita la integración con el ecosistema de desarrollo. RAML y API Blueprint también tienen su conjunto de herramientas y ecosistemas, pero no tienen una adopción tan amplia como OpenAPI.

3. **Madurez y adopción:** OpenAPI (anteriormente Swagger) tiene una mayor madurez y adopción en la industria, ya que ha estado presente durante más tiempo y ha sido ampliamente utilizado por organizaciones y comunidades. RAML y API Blueprint también tienen su base de usuarios, pero no tienen la misma adopción generalizada que OpenAPI.

4. **Enfoque de diseño:** OpenAPI y RAML se centran en la descripción de APIs RESTful, proporcionando una amplia gama de características y funcionalidades para documentar y describir las API. API Blueprint, por otro lado, se enfoca más en proporcionar una sintaxis sencilla y legible para describir las API.

5. **Características específicas:** Cada especificación tiene sus propias características únicas. Por ejemplo, OpenAPI tiene un enfoque robusto en la documentación de API, con soporte para definir esquemas de respuesta y solicitud, autenticación y autorización. RAML se destaca por su enfoque en la reutilización de componentes y la especificación de perfiles de API. API Blueprint se centra en la facilidad de lectura y la generación automática de documentación.

5.3.3. Estándares genéricos de una API REST:

Los resultados revelaron en detalle las directrices aplicadas en Salsa para definir la especificación OpenAPI, partiendo de una base genérica. A continuación, se presentan estas directrices genéricas a tener en cuenta al definir la estructura y los estándares de una API REST:

1. **Nomenclatura de los endpoints:** Definir reglas para nombrar los endpoints de la API.

2. **Métodos HTTP:** Establecer el uso correcto de los métodos HTTP.

3. **Diseños de las URL:** Definir un diseño consistente para las URLs de los endpoints de la API.

4. **Formato de respuesta:** Especificar el formato de respuesta de la API, como JSON, por ejemplo.
5. **Versionado:** Proporcionar directrices sobre cómo manejar el versionado de la API.
6. **Autenticación y autorización:** Establecer prácticas recomendadas para la autenticación y autorización de los usuarios.
7. **Manejo de errores:** Definir estándares para el manejo de errores en la API.
8. **Paginación:** Convenciones para la paginación de resultados en las respuestas de la API, como límites de resultados y parámetros de paginación.
9. **Filtrado, ordenamiento y búsqueda:** Cómo realizar filtros, ordenamientos y búsquedas en los recursos de la API
10. **Documentación:** Documentación de la API, incluyendo descripciones de endpoints, ejemplos de uso, explicación de parámetros y respuestas, y herramientas de generación de documentación.

5.4. Opiniones: Recetas y glosario de ReadMe

Sobre las “Recetas“

Las recetas en la documentación de ReadMe se utilizan para mostrar un bloque de código único y resaltar diferentes secciones a medida que se describen las líneas específicas dentro del bloque de código. Esto puede ser útil para proporcionar una guía detallada sobre cómo realizar una llamada a la API desde la perspectiva de un cliente, o tal vez para algo como la inicialización e invocación de “Salsa.js“. Sin embargo, estas recetas no reemplazan a las Los tutoriales o referencias, sino que pueden ser útiles como "mini-guías"detalladas.

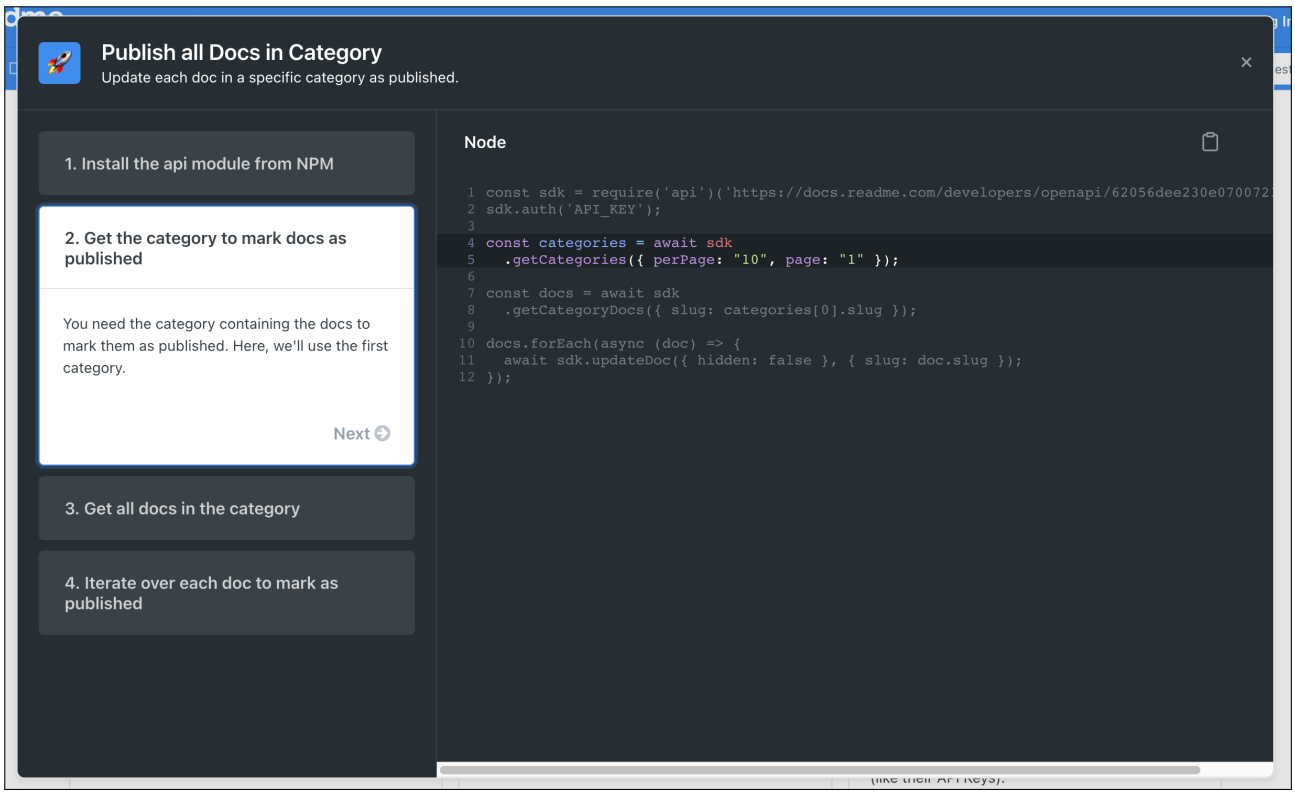


Figura 5.2: ReadMe: Recetas

Sobre el “Glosario“

ReadMe cuenta con un glosario que, a primera vista, parece que podría haber cumplido el papel de ser ese lugar donde se proporcionan los detalles sobre los conceptos/términos, pero al examinarlo más de cerca, parece estar dirigido a definiciones cortas que se muestran a través de información sobre herramientas al pasar el mouse en otras páginas.

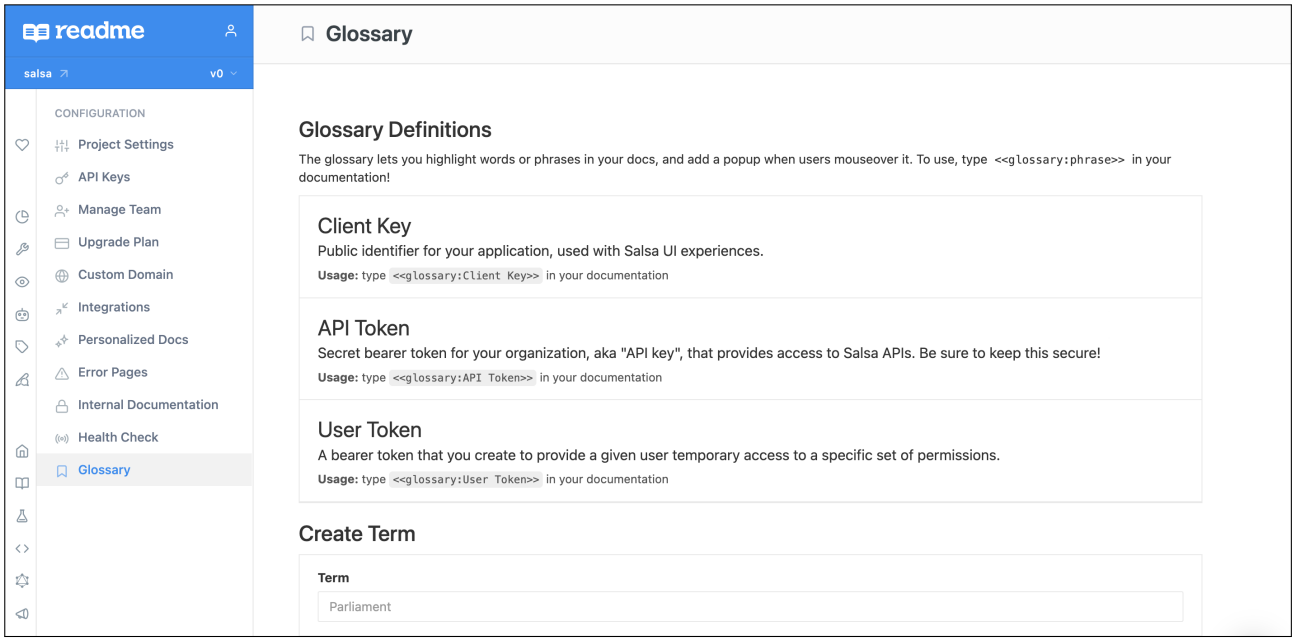


Figura 5.3: ReadMe

5.5. Limitaciones del Estudio:

Limitaciones y reflexiones sobre el mantenimiento de los tutoriales, guías prácticas y explicaciones.

Durante el desarrollo de esta investigación, como es natural, se identificaron limitaciones significativas en relación con el mantenimiento de los tutoriales, casos de uso y explicaciones. Una de las limitaciones destacadas fue la falta de un método eficiente para realizar pruebas y validar la coherencia de los documentos en ReadMe con respecto a la versión actualizada del API. Esto plantea la posibilidad de que algunos documentos se vuelvan obsoletos sin una detección automática. Además, se enfatizó la necesidad de mantener una interfaz intuitiva tanto para ingenieros como para perfiles no técnicos [27]. Estas limitaciones resaltan la importancia de abordar estos desafíos para garantizar la calidad y relevancia de la documentación técnica a lo largo del ciclo de vida del producto.

A pesar de los esfuerzos realizados para desarrollar un proceso de desarrollo ágil e integrar en el ciclo de vida del desarrollo del API la posibilidad de validar la generación de su especificación OpenAPI, no se aplicó de la misma forma para el resto de los documentos. Si no que más bien se priorizó la validación temprana de la estructura de la documentación con los clientes en lugar de establecer mecanismos automáticos para resolver este problema. Esto plantea la cuestión de si la solución implementada fue la más efectiva en términos de diseño y flujo de trabajo.

A medida que se desarrollaba la investigación, se evidenció que la pregunta de investigación inicial sobre las mejores prácticas para mantener actualizada y relevante la documentación técnica a lo largo del ciclo de vida del producto no fue abordada en su totalidad. Aunque se brindaron recomendaciones y pautas, se reconoce que la respuesta a esta interrogante aún requiere un enfoque más completo y detallado para lograr una comprensión exhaustiva del tema.

Finalmente, es de suma importancia resaltar la limitación identificada en este estudio, que radica en la ausencia de una herramienta o tecnología que aborde las necesidades de prueba (testing) y promueva la colaboración entre los equipos de ingeniería y los perfiles no técnicos. Esta carencia significativa destaca la necesidad de explorar y desarrollar soluciones más integrales que aborden esta brecha y mejoren la experiencia de mantenimiento de los tutoriales, casos de uso y explicaciones, brindando así una documentación actualizada y confiable.

Capítulo 6

Conclusiones, recomendaciones y líneas futuras

Según *Divio*[20] “No importa que tan bueno sea tu producto, si no está bien documentado la gente igualmente no lo usará“.

Existen muchos factores para que una startup tenga éxito y sin duda después de toda esta investigación se puede afirmar que contar con una buena documentación técnica es uno de esos factores.

La integración de la documentación técnica en el SDCL no es una tarea sencilla. Por lo tanto, es crucial que los miembros de la empresa tengan estándares y patrones claros para evitar cualquier incertidumbre sobre qué y dónde deben escribir al documentar el API. Contar con pautas y herramientas que faciliten la creación de la documentación técnica mejorará significativamente la productividad del equipo.

Durante el proceso de integración de la documentación técnica de las APIs, hemos aprendido lo siguiente:

- Es fundamental establecer una estructura clara para los documentos, preferiblemente siguiendo el patrón de Tutorial, Guías prácticas o casos de uso, Referencia técnica y Explicaciones. Esta estructura facilita la comprensión y navegación de la documentación.
- Contar con la especificación OpenAPI del API, aplicando ciertos criterios, es esencial. Sin el OAS, resulta difícil documentar correctamente las referencias técnicas en la documentación.
- Es importante utilizar sistemas como ReadMe, Redocly o Docusaurus, que permiten representar y actualizar la documentación de forma constante. Estas herramientas facilitan la visualización y mantenimiento de la documentación.
- Realizar pruebas de concepto es una estrategia valiosa para evaluar herramientas antes de realizar una implementación más seria. Esto nos permite identificar y seleccionar las herramientas más adecuadas para nuestras necesidades.

En conclusión, al seguir estos pasos, garantizamos que la documentación técnica del API se convierta en un recurso sólido, evitando deficiencias en nuestra base de conocimientos y mejorando la experiencia de los nuevos usuarios al integrarse con nuestras APIs. Al establecer una estructura clara, utilizar la especificación OpenAPI, emplear herramientas adecuadas y realizar pruebas de concepto, nos aseguramos de brindar una

documentación confiable y de calidad. Esto no solo facilita el uso y comprensión del API, sino que también fortalece la confianza y satisfacción de nuestros usuarios.

6.1. Recomendaciones y líneas futuras

Recomendación en relación con las limitaciones:

En relación con las limitaciones de la investigación, se recomienda adoptar un enfoque de programación conocido como "literate programming", "programación literaria"(o "letrada") [24] que promueva la integración directa de fragmentos de código fuente en la documentación. Este enfoque utiliza un lenguaje de marcado o notación específica para combinar la documentación y el código en un formato legible tanto para humanos como para máquinas. Por ejemplo, la herramienta nbdoc-docusaurus basada en nbdev (un sistema de "literate programming") permite la creación y pruebas de guías de usuario y blogs con Jupiter Notebooks. Esto permite estar seguros de que la versión de las guías de usuario, tutoriales y documentación están en línea con el código correspondiente.

Además, se sugiere considerar la implementación de pruebas contractuales (contract testing) [26] como una alternativa para futuras iteraciones. Estas pruebas permiten validar la compatibilidad y el cumplimiento de los contratos establecidos entre el proveedor (especificación del API) y el consumidor (tutoriales, casos de uso y explicaciones). Las pruebas contractuales garantizan que los componentes de la documentación se ajusten adecuadamente a la especificación del API, mejorando la calidad y la confiabilidad de la documentación.

Recomendación para la imagen de marca y la autonomía:

ReadMe ha demostrado ser una buena solución para lograr los objetivos propuestos a corto plazo. Sin embargo, existen opciones para mejorar la documentación.

Como recomendación para futuras iteraciones; sería interesante realizar una prueba de concepto donde se cree un híbrido entre dos herramientas de diferente “tipo“. (Discutidas anteriormente en la sección *Clasificación de la documentación según su tipología“* [5.3.1])

Un buen ejemplo de este tipo de híbridos es el portal de desarrolladores de **Dyte**[4] donde usan **Docusaurus**[3] como el generador del sitio y gestionar los documentos y **Stoplight Elements**[17] para renderizar el API 2.2.

¿Por qué un híbrido?

La pregunta correcta sería ¿Por qué no? **Docusaurus** es un generador de sitios estáticos muy popular con altas posibilidades de personalización y compatible con diferentes complementos. Sin embargo, no soporta de forma nativa OpenAPI, lo hace usando código de tercero. **Stoplight Elements** es un proyecto de código abierto y proporciona la posibilidad renderizar la especificación del API. Y además lo hace usando un sistema de tres columnas muy similar a la de “Stripe“ [18]. La documentación de Dyte [4] puede servir de ejemplo: `https://github.com/dyte-in/docs`

¿Por qué moverse a Docusaurus?

1. Con ReadMe la posibilidad de innovar en la experiencia de usuario es casi imposible, ya que no ofrece tanta flexibilidad. Con Docusaurus las posibilidades aumentan considerablemente.

2. En nuestra experiencia con ReadMe, encontramos algunos problemas con el API reference que no pudimos resolver por nuestra cuenta. Tener un mayor control sobre el proyecto de la documentación provee más oportunidades para solucionar ese tipo de problemas.

3. Al seguir estas recomendaciones, no se pierde ninguna “funcionalidad“ de ReadMe.

4. Es una forma más “barata“ de generar documentación.

5. Posibilidad de incluir una página de inicio (Landing page).

6. Al tratarse de una solución híbrida; la integración con Markdoc [6] y Codehike [2] es posible.

Capítulo 7

Conclusions, recommendations, and Future work

According to Divio[20], “No matter how good your product is, if it is not well-documented, people will not use it.“

Integrating technical documentation into the SDLC (Software Development Life Cycle) is not a straightforward task. Therefore, it is crucial for team members to have clear standards and patterns to avoid any uncertainty about what and where to write when documenting the API. Having guidelines and tools that facilitate the creation of technical documentation significantly improves team productivity.

Throughout the process of integrating API documentation, we have learned the following:

- Establishing a clear structure for the documents is essential, preferably following the pattern of Tutorials, Practical Guides or Use Cases, Technical Reference, and Explanations. This structure enhances the understanding and navigation of the documentation.
- Having the OpenAPI specification of the API, with certain criteria applied, is essential. Without the OAS, it becomes challenging to properly document the technical references in the documentation.
- Utilizing systems such as ReadMe, Redocly, or Docusaurus that allow for constant representation and updating of the documentation is important. These tools facilitate visualization and maintenance of the documentation.
- Conducting proof-of-concept tests is a valuable strategy to evaluate tools before making more significant implementations. This enables us to identify and select the most suitable tools for our needs.

In conclusion, by following these steps, we ensure that the API’s technical documentation becomes a robust resource, avoiding deficiencies in our knowledge base and enhancing the experience of new users when integrating with our APIs. Establishing a clear structure, utilizing the OpenAPI specification, employing appropriate tools, and conducting proof-of-concept tests guarantee reliable and high-quality documentation. This not only facilitates the use and understanding of the API, but also strengthens user trust and satisfaction.

7.1. Recommendations and future work

Recommendation regarding the limitations:

In light of the research limitations, it is recommended to adopt a programming approach known as "literary programming"[24], which promotes the direct integration of source code snippets into the documentation. This approach utilizes a specific markup language or notation to combine documentation and code in a format that is readable by both humans and machines. For example, the tool nbdoc-docusaurus based on nbdev (a system of "literate programming") enables the creation and testing of user guides and blogs with Jupiter Notebooks. This way we have guarantee that the version of the user guides, tutorials and documentation are in line with the corresponding code.

Furthermore, it is suggested to consider implementing contract testing [26] as an alternative for future iterations. Contract testing allows for validating the compatibility and compliance of contracts established between the provider (API specification) and the consumer (tutorials, use cases, and explanations). Contract testing ensures that the components of the documentation align properly with the API specification, thereby improving the quality and reliability of the documentation.

Branding and autonomy recommendations:

ReadMe has proven to be an adequate solution for achieving the short-term objectives. However, there are options to enhance the documentation.

As a recommendation for future iterations, it would be interesting to conduct a proof of concept where a hybrid approach is created using two different types of tools (as discussed earlier in the section “Classification of documentation by type“ [5.3.1]).

A good example of such hybrids is the developer portal of **Dyte** [4], where they utilize **Docusaurus** [3] as the site generator and document management tool, and **Stoplight Elements** [17] for rendering the API reference 2.2.

Why a hybrid approach?

The question should be, why not? **Docusaurus** is a highly popular static site generator with extensive customization possibilities and compatibility with various plugins. However, it does not natively support OpenAPI, relying on third-party code for that functionality. On the other hand, **Stoplight Elements** is an open-source project that provides the capability to render the API specification, using a three-column layout similar to that of "Stripe"[18].

The documentation of Dyte [4] can serve as an example: `https://github.com/dyte-in/docs`

Why move to Docusaurus?

Here are several reasons to consider moving to Docusaurus:

1. With ReadMe, the possibility of innovating in the user experience is limited, as it does not offer as much flexibility. Docusaurus provides significantly more possibilities for customization and innovation.

2. In our experience with ReadMe, we run across some issues with the API reference that we were unable to resolve on our own. Having greater control over the documentation project allows for more opportunities to address such problems.
3. By following these recommendations, you will not lose any functionality provided by ReadMe.
4. Docusaurus offers a more cost-effective solution for generating documentation.
5. The ability to include a landing page is a valuable feature.
6. As a hybrid solution, integration with Markdoc [6] and Codehike [2] is possible.

Bibliografía

[1] Api blueprint: web principal. https://apiblueprint.org/. Accessed: 2023-06-27.

[2] Codehike: Not just a syntax highlighter. https://codehike.org/. Accessed: 2023-06-27.

[3] Docusaurus: Build optimized websites quickly, focus on your content. https://docusaurus.io/. Accessed: 2023-06-27.

[4] Dyte: portal para desarrolladores. https://docs.dyte.io. Accessed: 2023-06-27.

[5] Kotlin: Kotlin is a programming language. https://kotlinlang.org/. Accessed: 2023-06-27.

[6] Markdoc: is a powerful, flexible, markdown-based authoring framework. https://markdoc.dev/. Accessed: 2023-06-27.

[7] Marqeta, página web oficial. https://marqeta.com. Accessed: 2023-05-27.

[8] Miro: portal para desarrolladores. https://developers.miro.com/docs. Accessed: 2023-06-27.

[9] Notion: portal para desarrolladores. https://developers.notion.com/. Accessed: 2023-06-27.

[10] Plaid, página web oficial. https://plaid.com. Accessed: 2023-05-27.

[11] Raml: web principal. https://raml.org/. Accessed: 2023-06-27.

[12] Readme: transforms your api docs into interactive hubs that help developers succeed. https://readme.com/. Accessed: 2023-06-27.

[13] Redocly: Make api docs your superpower. https://redocly.com/. Accessed: 2023-06-27.

[14] Redux: portal para desarrolladores. https://es.redux.js.org/. Accessed: 2023-06-27.

[15] Salsa: Grow revenue and delight your customers by launching payroll. https://www.salsa.dev/. Accessed: 2023-06-27.

[16] Scale: portal para desarrolladores. https://docs.scale.com/reference/introduction. Accessed: 2023-06-27.

[17] Stoplight elements: Beautiful api documentation powered by openapi and markdown. https://stoplight.io/open-source/elements. Accessed: 2023-06-27.

[18] Stripe, página web oficial. https://stripe.com. Accessed: 2023-05-27.

[19] Supabase: portal para desarrolladores. https://supabase.com/docs/guides/resources. Accessed: 2023-06-27.

[20] Divio Technologies AB. Documentation: The problem and the solution. https://shorturl.at/anoFY. Accessed: 2023-06-27.

[21] Amazon AWS. SaaS: Software as a services. https://aws.amazon.com/es/what-is/saas/.

[22] Douglas da Silva. SLA, ¿qué significa sla y para qué sirve? 2021. Accessed: 2023-06-28.

[23] Brajesh De. *API Documentation*, pages 59–80. Apress, Berkeley, CA, 2017.

[24] Fernando Doglio. Literate programming: A radical approach to writing code with documentation. https://blog.bitsrc.io/literate-programming-a-radical-approach-to-writing-code-with-documentation-ebb

[25] The Linux Foundation. OpenAPI, what is OpenAPI? https://www.openapis.org/what-is-openapi.

[26] Martin Fowler. Contracttest. https://martinfowler.com/bliki/ContractTest.html.

[27] Félix Gonzalo. ¿qué es el NO-CODE? https://www.nocoders.academy/blog/que-es-nocode. Accessed: 2023-06-27.

[28] Creswell JW. *Qualitative inquiry and research design: choosing among five traditions.* 2006.

[29] Nafis Kamal. Api documentation generator. Master's thesis, Aalto University, School of Science, Master's Programme in Security and Cloud Computing, Espoo, 2022.

[30] Andreas Schubert Michael Meng, Stephanie M. Steinhardt. Optimizing api documentation. 1, 2020.

[31] Nuttlabs. Bulletproof: methodology draws from popular productivity systems and leading notion. https://uno.notion.vip/bulletproof/.

[32] S M Sohan, Frank Maurer, Craig Anslow, and Martin P. Robillard. A study of the effectiveness of usage examples in rest api documentation. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 53–61, 2017.

[33] TeoCom. Investigación cualitativa: Definición, características y ejemplos. https://www.youtube.com/watch?v=w8UD6lRAF0E.

[34] Wikipedia. Api, definition. https://es.wikipedia.org/wiki/API.

[35] Wikipedia. Sat, servicio de asistencia técnica. https://shorturl.at/lxFH4. Accessed: 2023-06-28.

Apéndices

Apéndice A

Encuesta de requerimientos

1. Will engineers be heavily authoring and collaborating on the content?
2. How many authors will be authoring directly as contributors in the system?
3. Does your “security group” restrict you from using ‘third-party platforms‘ to host documentation, such as GitHub?
4. Do you already have some system set up internally that you want to use for storing and automatically building your site?
5. Do you have engineering resources available to implement your own continuous delivery publishing workflow?
6. Do you have a strong familiarity with a particular programming language? Could you please enumerate some of them?
7. Approximately how many documentation pages do you have in your project?
8. Do you have an available budget to pay for a third-party hosting and deployment option?
9. Do you need to authenticate documentation for specific users? Is there an existing authentication system already in place at your company?
10. Do you need to localize the content? If so, how many other languages? Are there formatting requirements imposed by your translation vendor and system?
11. Do you need to create PDF deliverables for the content (in addition to web output)?
12. Do you have some web development skills (or access to UX resources) to design or modify your theme?
13. Do you need to integrate your docs directly into your larger company site, with the same branding and appearance?
14. Do you want a lot of control and flexibility to extend or customize the solution with your specific doc’s needs, which might involve time-intensive custom scripting or integration with another system?
15. Can you use an external search service such as Swiftype, Algolia, or Google Custom Search?

16. Do you have to version your content with each new release?

Apéndice B

Salsa REST API Guidelines

This is the definition for Salsa REST APIs. It is based on the Microsoft REST API guidelines. Even though not everything in the Microsoft Guidelines apply to Salsa, we recommend that you read those guidelines before start designing APIs.

Review the REST Coding Practices before start the implementation of the API.

1. Paths

Path naming is snake-case.

Partner APIs are published in `/api/rest` base path.

1.1 Views

If you need different views of the same object, construct different URLs to distinguish views.

For example: to obtain the current pending payroll

```
GET payroll-runs/current
```

2. Response

Follow the Response Format Guidelines

All the responsess from REST APIs are wrapped in the following envelope object:

```
RestApiEnvelope:
  type: object
  properties:
    data:
      $ref: <response object>
```

The response can also be a list:

```
RestApiEnvelopeList:
  type: object
  properties:
    data:
      type: array
      items:
        $ref: <response object>
```

3. Errors

The error envelope has the following structure:

```
ErrorApi:
  required:
  - code
  - message
  type: object
  properties:
    code:
      type: string
      description: One of a Salsa set of error codes
    target:
      type: string
      description: The target of the error
    message:
      type: string
      description: A human-readable representation of the error
  description: An array of details about specific errors that led to this reported

ErrorApiEnvelope:
  type: object
  properties:
    error:
      $ref: '#/components/schemas/ErrorRestResponse'

ErrorRestResponse:
  type: object
  properties:
    code:
      type: string
      description: HTTP Status code
    message:
      type: string
      description: A human-readable representation of the error
    target:
      type: string
      description: The target of the error
    details:
      type: array
      description: An array of details about specific errors that led to this
        reported error
      items:
        $ref: '#/components/schemas/ErrorApi'
```

3.1 Error codes

Use HTTP error codes to identify the type of error.

Generally 4xx errors are client errors (validation, authentication errors, incorrect order of execution, etc), and 5xx are server-side errors (unexpected problems in the system, service is degraded).

- The only way your service should respond with a 500 code is by processing an unhandled exception.
- Captured exceptions should produce 400 errors
- Handle all 400 errors separately and provide each of them with a thorough error message.
- It is important that 400 and 500 errors are well-defined and differentiated. Avoid capturing exceptions that could mask an internal server error behind 400 errors.

4. HTTP Methods

Use HTTP methods to operate on collections and entities

There is one single rule concerning operations performed on collections and entities - **Use HTTP methods**

You can operate on resources using HTTP methods such as `POST`, `GET`, `PUT`, and `DELETE` - to remember them, refer to the CRUD acronym (Create-Read-Update-Delete).

Resource / HTTP method	POST (create)	GET (read)	PUT (update)	PATCH (upsert)	DELETE (delete)
/users	Create new user	List users	Error	Create	Error
/users/{uuid}	Error	Get user	Update user if exists	Create or update	Delete user

4.1 Update and create must return a resource representation

`PUT`, `POST` and `PATCH` methods modify fields of the underlying resource that were not part of the provided parameters (for example: `createdAt` or `updatedAt` timestamps). To prevent an API consumer from having to hit the API again for an updated representation, have the API return the updated (or created) representation as part of the response.

`PATCH` has been standardized by IETF as the method to be used for updating an existing object incrementally (see RFC 5789).

4.2 Creating resources via `PATCH` (UPSERT semantics)

Services that allow callers to specify key values on create SHOULD support UPSERT semantics, and those that do MUST support creating resources using `PATCH`. Because `PUT` is defined as a complete replacement of the content, it is dangerous for clients to use `PUT` to modify data. Clients that do not understand (and hence ignore) properties on a resource are not likely to provide them on a `PUT` when trying to update a resource, hence such properties could be inadvertently removed. Services MAY optionally support `PUT` to update existing resources, but if they do they MUST use replacement semantics (that is, after the `PUT`, the resource's properties MUST match what was provided in the request, including deleting any properties that were not provided).

Under UPSERT semantics, a `PATCH` call to a nonexistent resource is handled by the server as a "create", and a `PATCH` call to an existing resource is handled as an "update". To ensure that an update request is not treated as a create or vice versa, the client MAY

specify precondition HTTP headers in the request. The service MUST NOT treat a `PATCH` request as an insert if it contains an `If-Match` header and MUST NOT treat a `PATCH` request as an update if it contains an `If-None-Match` header with a value of `*`.

If a service does not support UPSERT, then a `PATCH` call against a resource that does not exist MUST result in an HTTP `409 Conflict` error.

5. Versioning

Only major version definition in the URL is supported.

`/api/rest/v?` Where ? is an integer number representing the major version.

All changes must be backward compatible. Breaking changes can only be introduced when increasing the major version.

Major version is increased in all APIs at once.

6. Naming

6.1 Enums

Use SCREAMING_SNAKE_CASE for enums

- FIXED
- RATE
- NOT_PROVIDED
- PERCENTAGE
- FREQUENCY

6.1.1 Exceptions In some cases we may want to break this convention for enums. An example is for the Webhook Event type. There is already a standard that is understood by developers and conforming this enum to our convention would make the API less intuitive for developers who are already accustomed to the existing standards.

Any exception should be considered/discussed and documented appropriately with the motivations.

7. [DRAFT] Sorting

Pending Springboot PoC

8. [DRAFT] Filtering

Pending Springboot PoC

8.1 [DRAFT] Simple filtering

Pending Springboot PoC

8.2 Advanced filtering

Use an action to produce a more complex filter using query by example paradigm, use POST to indicate that this operation is not idempotent, meaning different results will be returned every time that the operation is sent, if this is not the case, consider using PUT:

```
POST payroll-runs/searchByExample
{
    id: [1, 2, 3]
    state: "CONFIRMED"
}
```

9. Pagination

Use offset pagination for fixed lists that won't likely change, eg: A generated report or even the PayrollRun result. Use curosr pagination if changes in the collection could happen while viewing the data, eg: the list of workers of a company.

9.1 Cursor pagination

Use the following parameters for cursor pagination:

```
parameters:
- name: cursor
  in: query
  required: false
  schema:
    type: string
- name: size
  in: query
  required: false
  schema:
    type: integer
    format: int32
    default: 500
```

9.2 Offset pagination

Use the following parameters for pagination:

```
parameters:
- name: page
  in: query
  required: false
  schema:
    type: integer
    format: int32
- name: size
  in: query
  required: false
```

```
  schema:
    type: integer
    format: int32
    default: 500
```

10. Command pattern

Use only when impementing semantics that goes beyond CRUD operation. Avoiding `create`, `read`, `update` or `delete` names when creating a command pattern.

To implement this pattern add a sub-resource with commands to your business resource, for example: `/payroll-run/{payrollRunId}/confirm`. In this example if you want to execute a complex operation on a payrollRun add the operation input data to the appropriate commands collection. Do not use `POST` as `POST` is not idempotent in REST – use the `PUT` method and optionally an UUID generated by the client to support dedupliction if necesary.

11. [DRAFT] Nesting foreign resources relations

Nest foreign resources references, even if the only information is `id` of the object referred to, with a nested object, e.g.:

```
{
   id: "pr_dfs...",
   status: "CONFIRMED",
   employer: {
      id: "em_2342rf..."
   }
}
```

This approach makes it possible to inline more information about the related resource without having to change the structure of the response or introduce more top-level response fields, e.g.:

```
{
   id: "pr_dfs...",
   status: "CONFIRMED",
   employer: {
      id: "em_2342rf..."
    businessName: "business"
   }
}
```

Apéndice C

Capturas de las pruebas de concepto

Capturas de pantalla que documentan y muestran los resultados, de las pruebas de concepto realizadas.

ReadMe

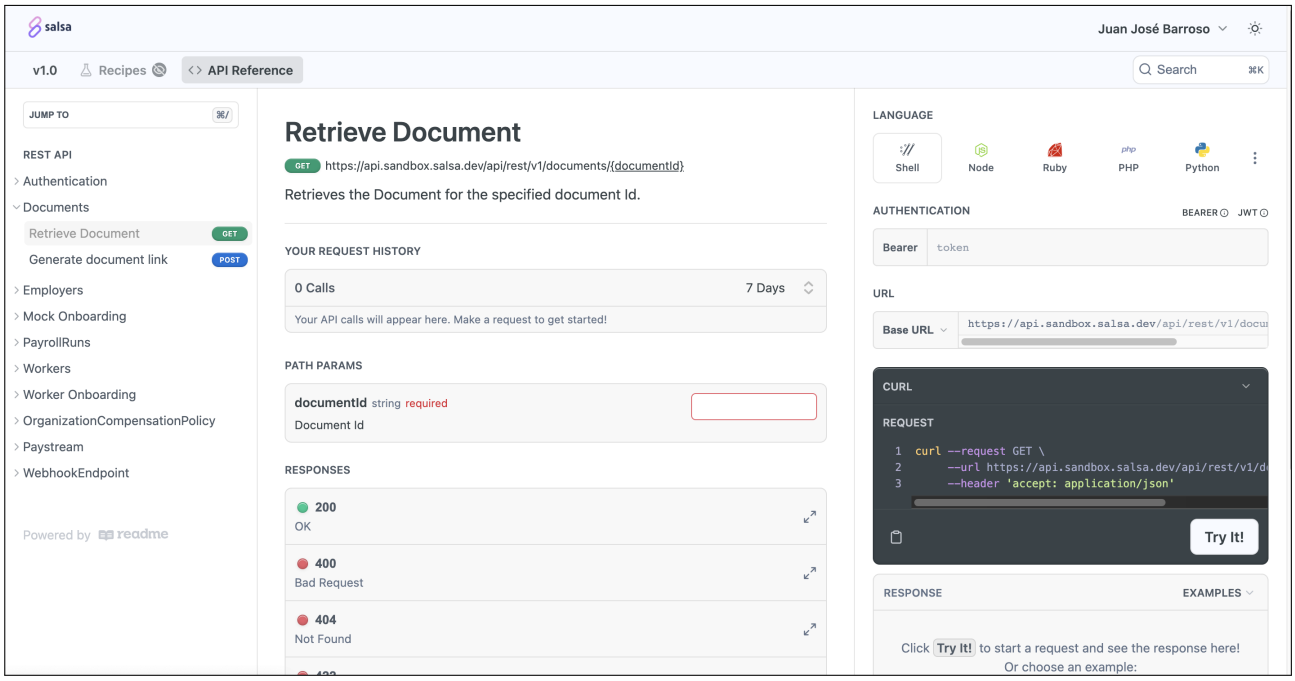


Figura C.1: POC: ReadMe

Docusaurus

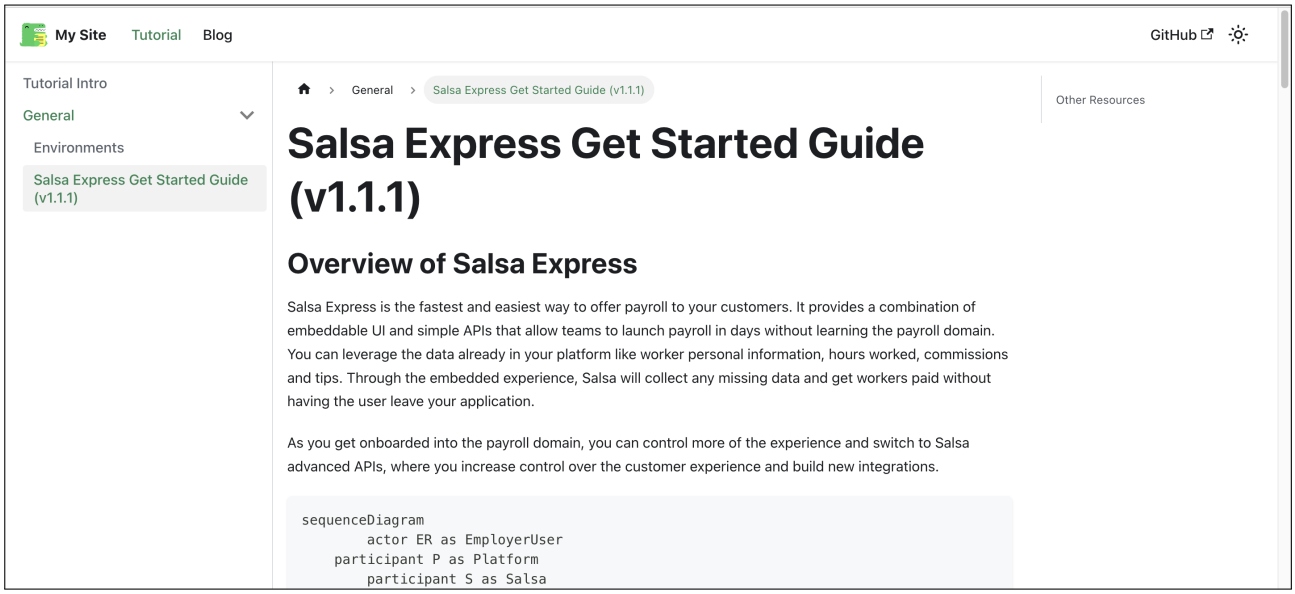


Figura C.2: POC: Docusaurus

Redocly

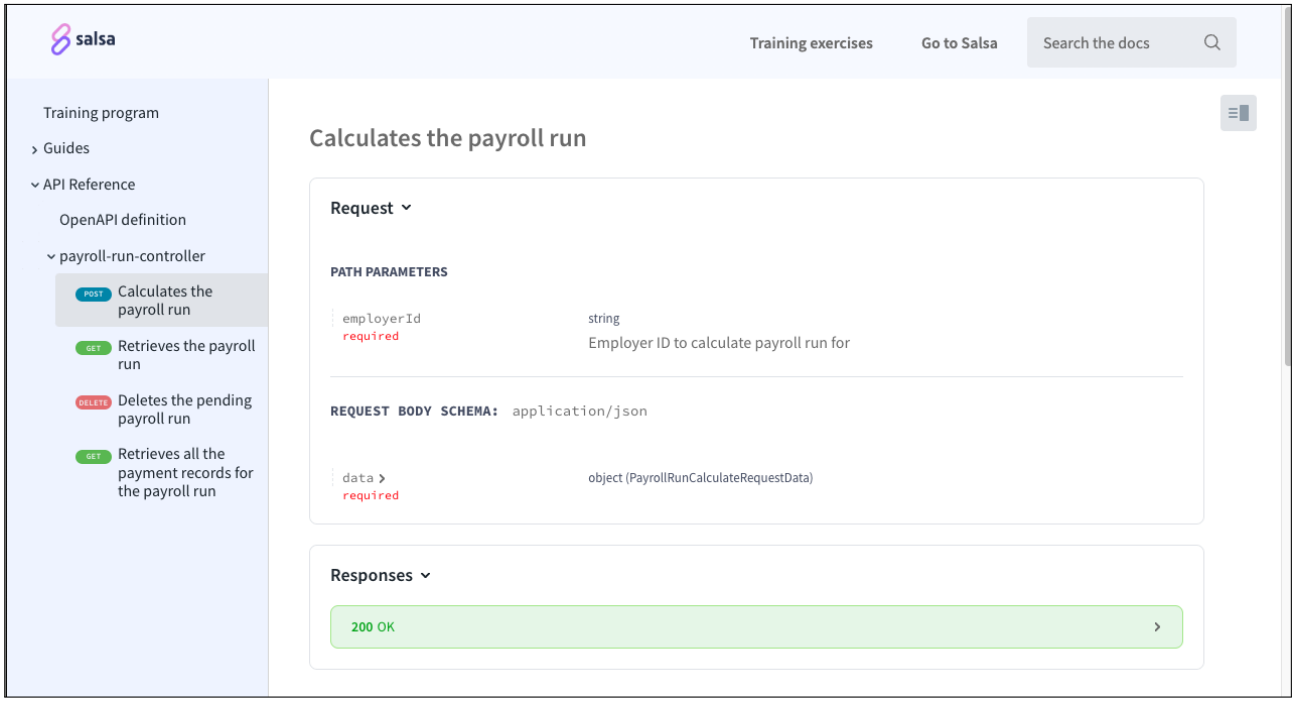


Figura C.3: POC: Redocly

Stoplight

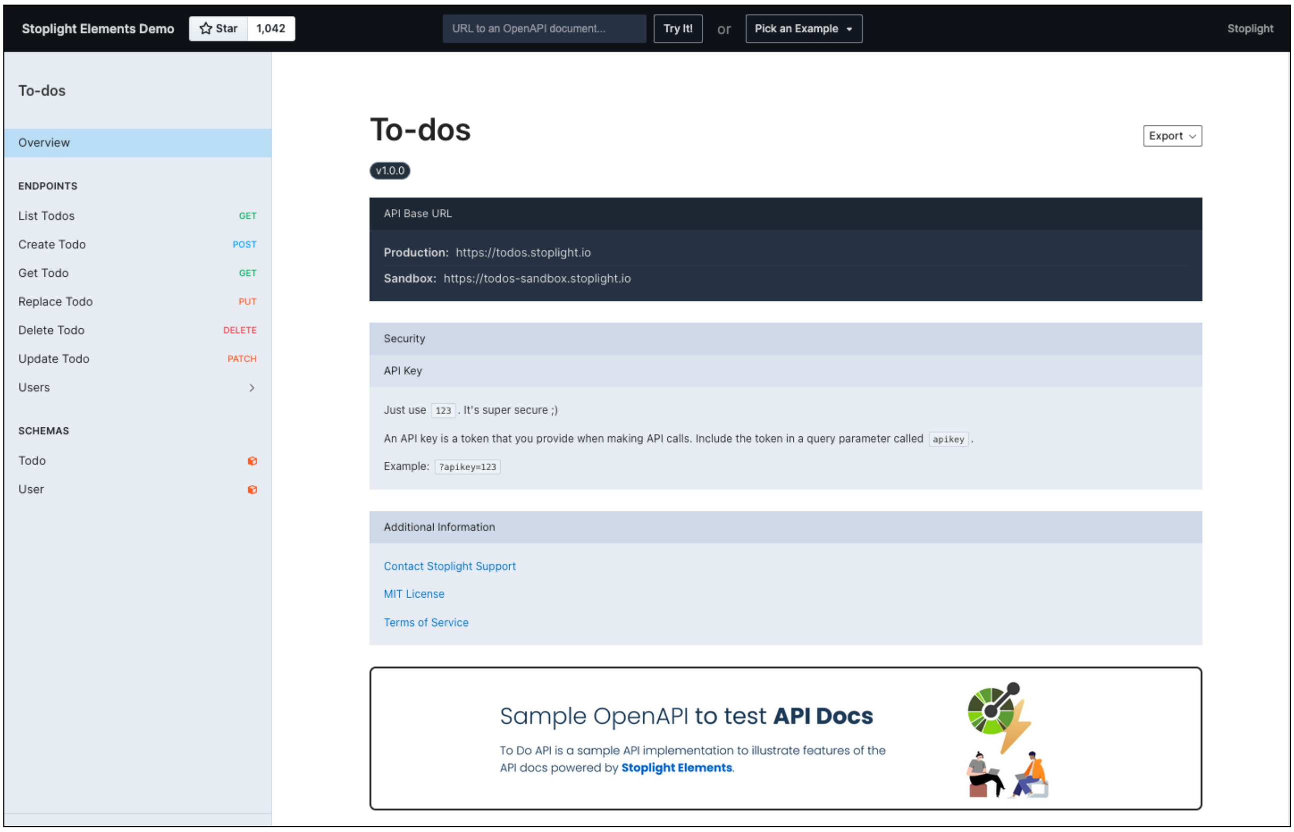


Figura C.4: POC: Stoplight

Theneo

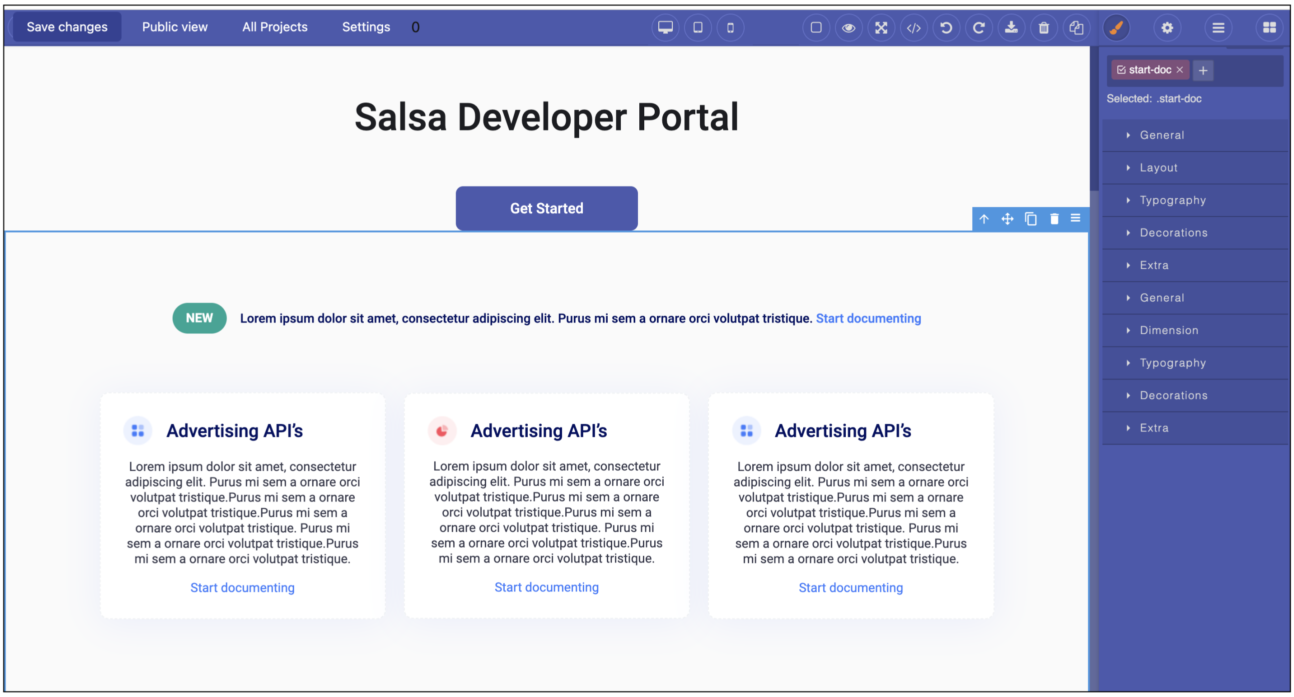


Figura C.5: POC: Theneo

GitBook

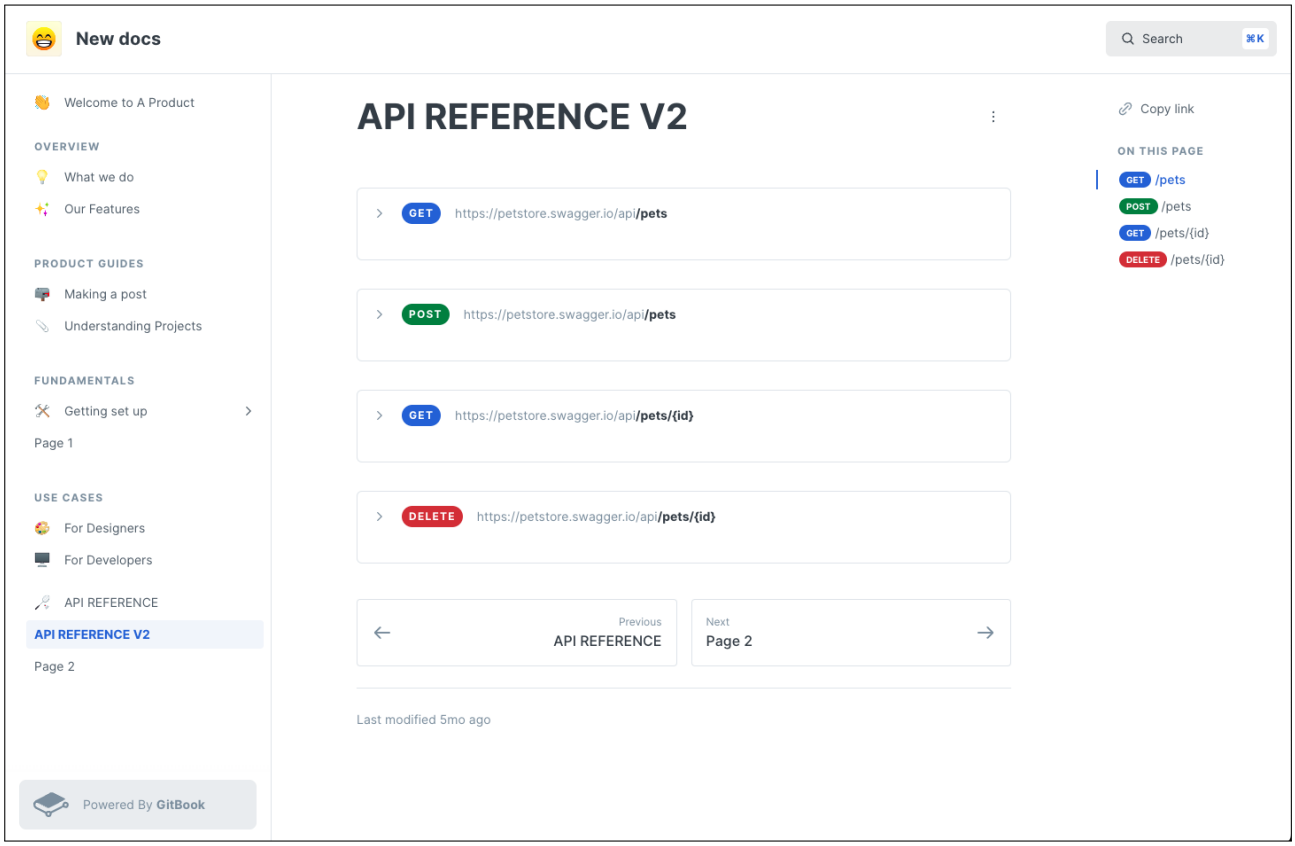


Figura C.6: POC: GitBook

Apéndice D

OpenAPI

The OpenAPI Specification, previously known as the Swagger Specification, is a specification for a machine-readable interface definition language for describing, producing, consuming and visualizing RESTful web services.

An OpenAPI specification will be automatically generated when reaching the OpenApi endpoint e.g. `http://localhost:8080/v3/api-docs.yaml` (for local and similarly for other environments). You will need to set the required AuthorizationHeaders.

The endpoint will generate a `yaml` that can be used to read the api documentation. Just paste it in any OpenAPI editor: `https://editor.swagger.io/` or use the internal swagger editor in http://localhost:8080/webjars/swagger-ui/index.html

By default, the apis won't show unless they are exposed in `application.yml` under `springdoc.paths-to-match`

If you are making changes in the definition of the API it is recommended to run

```
./gradlew generateOpenApiDocs
```

to be able to see the changes of the OpenAPI document in the pull request

Apéndice E

Capturas del sistema

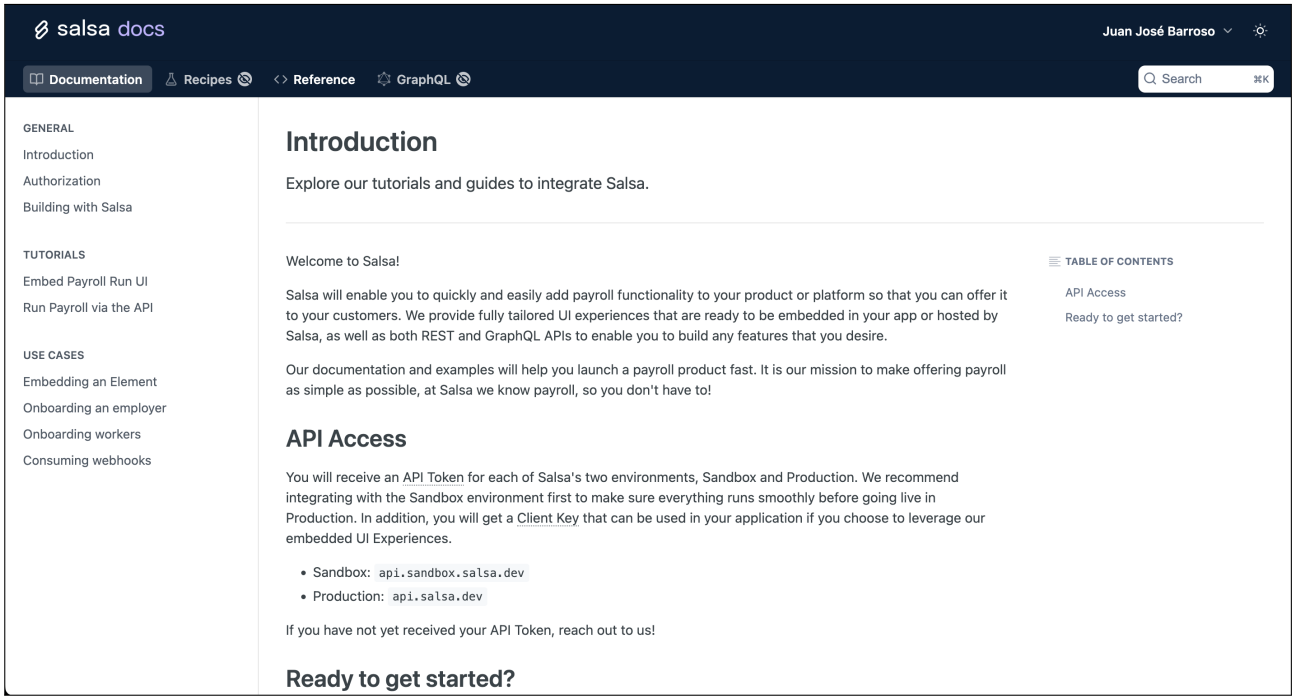


Figura E.1: Introducción

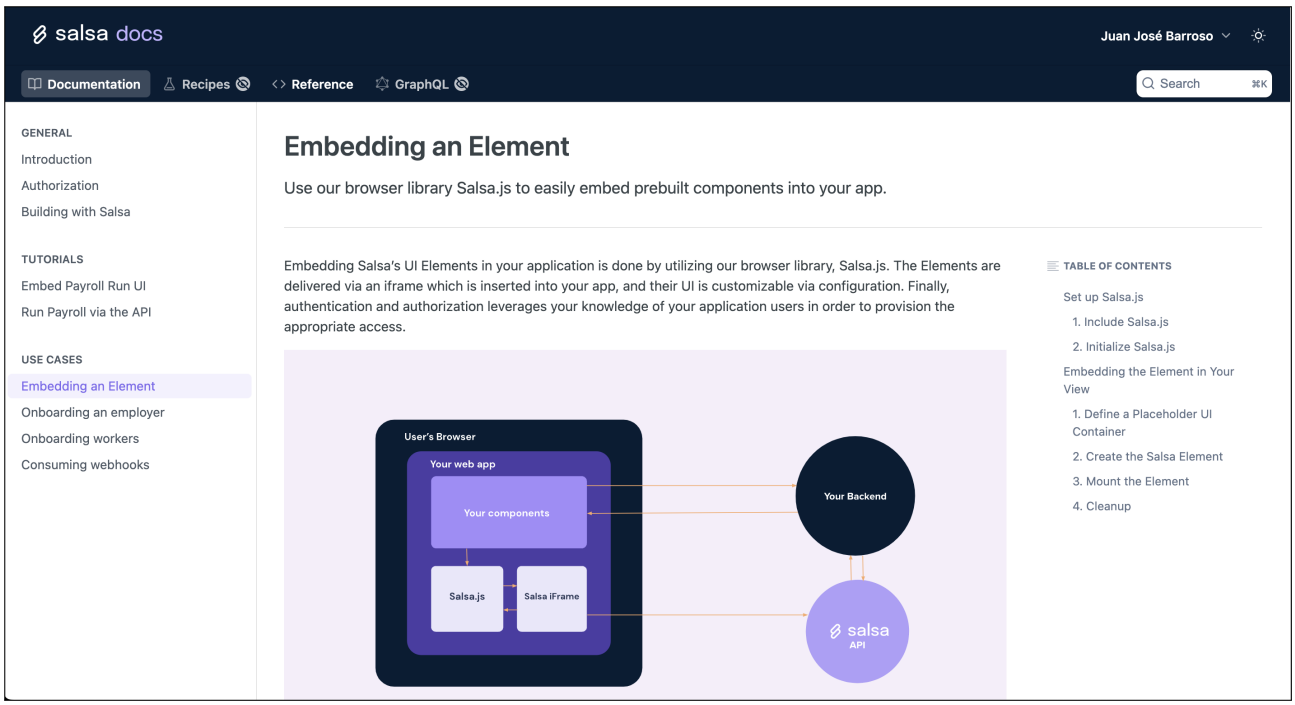


Figura E.2: Caso de uso: "Embedding an Element"

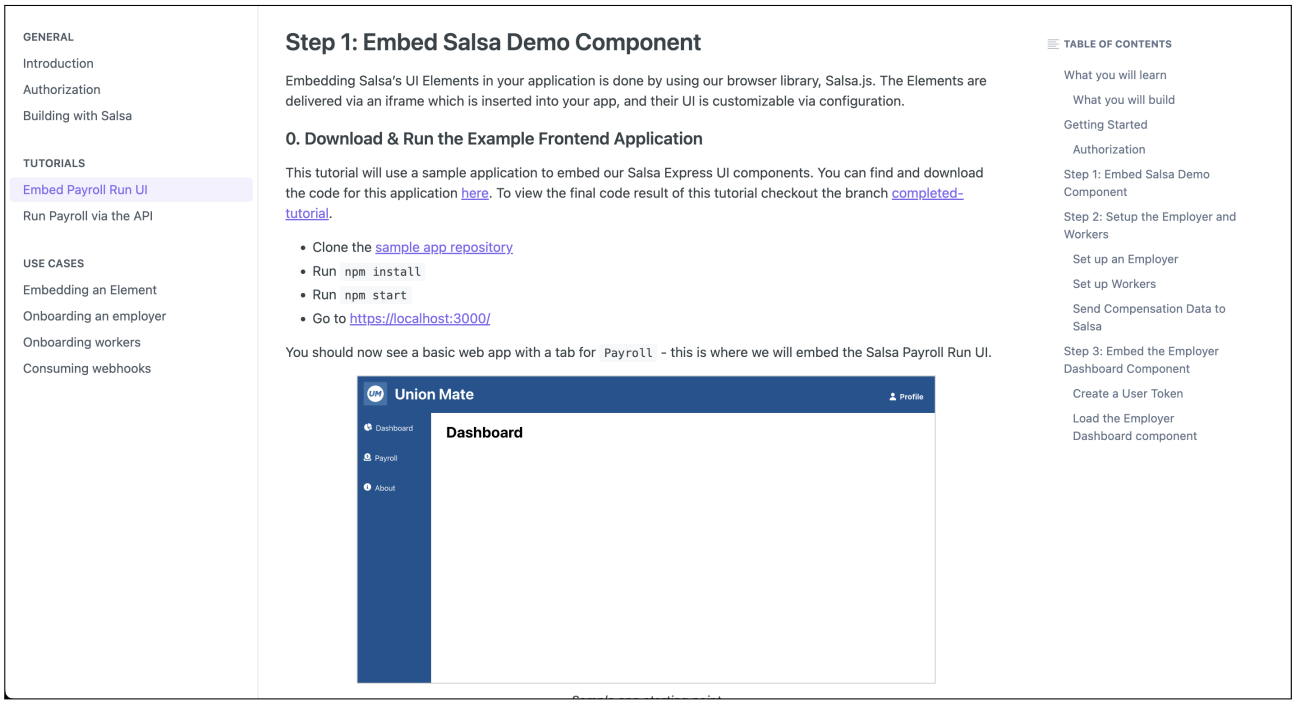


Figura E.3: Tutorial: "Embed Payroll Run UI"

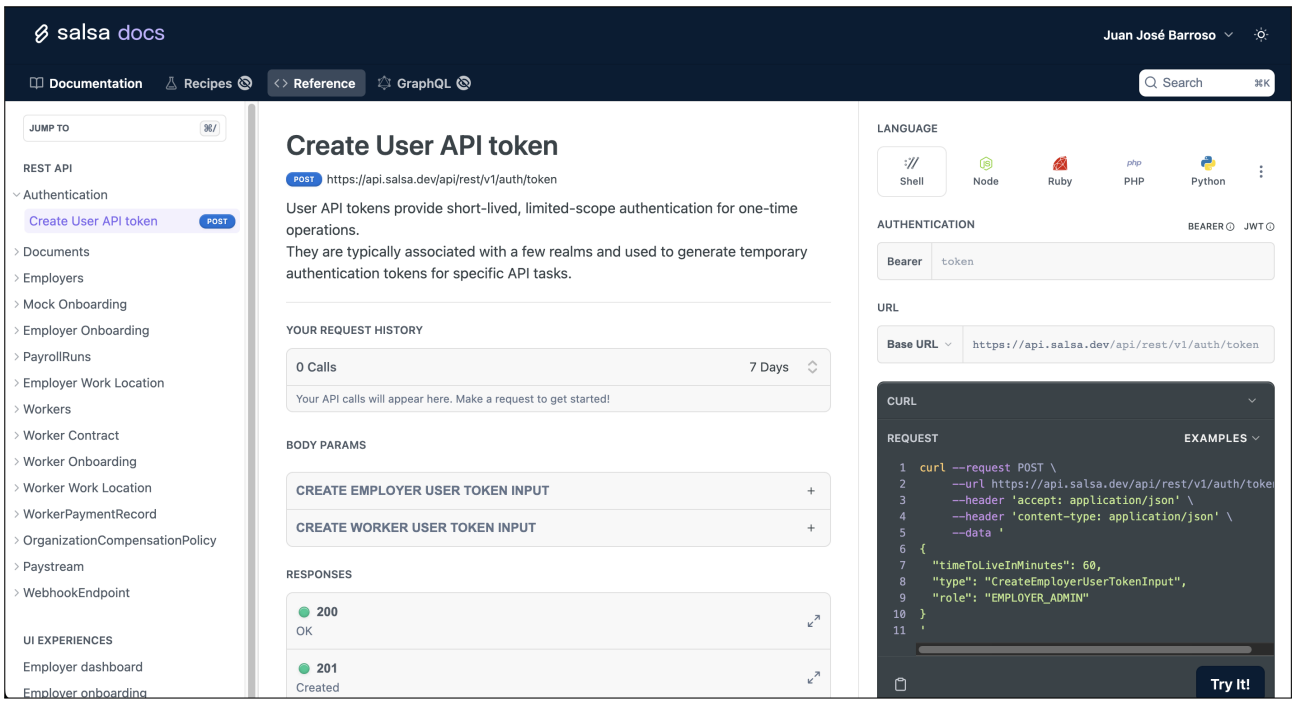


Figura E.4: Referencias técnicas: “API Reference“

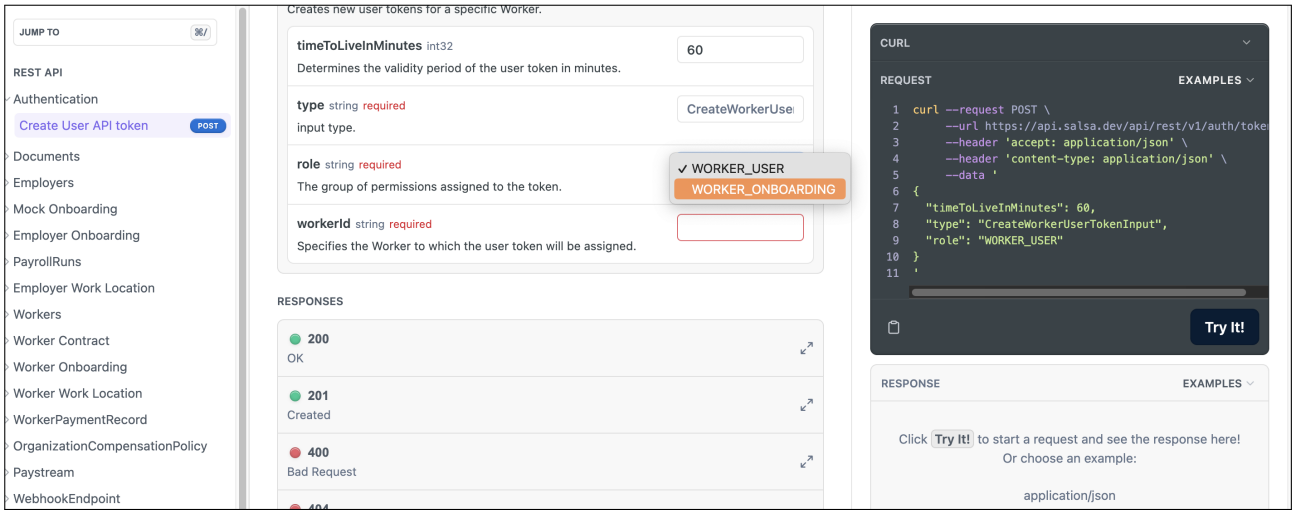


Figura E.5: Referencias técnicas: selectores autogenerados