

Grado en Ingeniería Informática

Curso 2022-2023

Título

Optimización del proceso de desarrollo de APIs REST mediante la implementación de una herramienta de generación automática de código (APIzzy).

Fabiana Alexandra Calles Pestana

Tutor

Juan Miguel Gómez Berbis

Leganés, Agosto 2023



Esta obra se encuentra sujeta a la licencia Creative Commons Reconocimiento - No Comercial - Sin Obra Derivada

Agradecimientos.

Deseo expresar mi sincero agradecimiento a todas las personas que han contribuido, directa o indirectamente, al desarrollo y finalización de este trabajo de fin de grado.

En primer lugar, no puedo pasar por alto la oportunidad de expresar mi profundo agradecimiento a mis padres y a mi querida hermana. A lo largo de los años, su inquebrantable apoyo ha sido el cimiento sobre el cual he edificado cada uno de mis logros. Sus palabras de aliento en los momentos de duda y sus sonrisas llenas de orgullo en los triunfos compartidos han sido mi mayor fortaleza. Gracias a mi padre por la idea que condujo a la realización de este proyecto.

A Simón y Nacho, su apoyo y consejo han sido invaluable en esta travesía. Gracias por creer en mí, en el potencial de este proyecto y sobre todo, gracias por vuestra amistad y amor.

Mis amigos Carlos De Val, Carlos Díez, Jorge Zazo, Daniel Prieto y Ana Fabiola Vásquez, han sido el apoyo emocional constante durante los altos y bajos de la carrera. Su amistad ha sido un regalo que valoro enormemente.

Mi gratitud a mi tutor Juan Miguel Gómez Berbis por su orientación, paciencia y confianza depositada en mí durante todo el proceso de elaboración de este trabajo.

A Nicole Morante, cuya amistad ha sido una constante desde hace muchos años, gracias por estar siempre allí.

Por último, quiero agradecer a todas las personas con las que he compartido experiencias y conocimientos a lo largo de estos años. Cada uno de ustedes ha enriquecido mi trayectoria académica y personal.

Gracias.

Resumen

APIzzy surge como una respuesta innovadora a un problema persistente en el panorama tecnológico actual: la complejidad de la creación y gestión de APIs. Aunque las APIs se han vuelto fundamentales para la interacción en el mundo digital, su creación sigue siendo un reto, especialmente para aquellos con menor experiencia técnica.

APIzzy, no solo simplifica este proceso sino que introduce una forma revolucionaria de diseñar APIs. A través de un formulario intuitivo, permite a los usuarios, independientemente de su experiencia, construir APIs personalizadas que recolectan datos de diversas fuentes. Esta propuesta innovadora no solo reduce la barrera de entrada al mundo de las APIs, sino que redefine cómo las percibimos y las utilizamos.

Utilizando tecnologías de vanguardia como Python y React, APIzzy garantiza un producto de alto rendimiento con una interfaz amigable. Pero su verdadera aportación va más allá de la técnica: su impacto socioeconómico. Al democratizar la creación de APIs, facilita a empresas y desarrolladores individuales la reducción de tiempos y costos en desarrollo. Esto es especialmente valioso para pequeñas y medianas empresas que buscan digitalizarse pero se ven obstaculizadas por barreras técnicas y económicas.

A lo largo de su desarrollo, adoptamos una metodología ágil, lo que nos permitió adaptarnos y evolucionar según las necesidades. Con una inversión de 10.601,6 euros y seis meses de trabajo, APIzzy no es solo una herramienta, es un cambio de juego en el ecosistema tecnológico.

Palabras clave:

Python, API, Formulario, Json, React, Gestión del proyecto, Flask Restful, Interface de usuario.

Abstract

APIzzy emerges as an innovative response to a persistent issue in the current technological landscape: the complexity of creating and managing APIs. Even though APIs have become essential for interaction in the digital world, their creation remains a challenge, especially for those with less technical experience.

APIzzy not only simplifies this process but introduces a revolutionary way to design APIs. Through an intuitive form, it allows users, regardless of their experience, to build customized APIs that gather data from various sources. This groundbreaking approach not only lowers the entry barrier to the world of APIs but redefines how we perceive and use them.

Utilizing cutting-edge technologies like Python and React, APIzzy ensures a high-performance product with a user-friendly interface. However, its true contribution extends beyond the technical: its socioeconomic impact. By democratizing API creation, it aids businesses and individual developers in reducing development times and costs. This is especially valuable for small and medium-sized businesses looking to digitize but are hindered by technical and economic barriers.

Throughout its development, we adopted an agile methodology, allowing us to adapt and evolve according to needs. With an investment of 10,601.6 euros and six months of work, APIzzy is not just a tool, it's a game-changer in the tech ecosystem.

Keywords:

React, Python, API, interface, Form, Flask Restful, project management.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Problema	3
1.3. Motivación	3
1.4. Objetivos	4
1.5. Marco Regulador	5
1.6. Estructura del Documento	6
 2. Estado del arte	 9
2.1. Historia y Evolución de las API	9
2.2. Tipos de APIs	11
2.3. Protocolos y arquitecturas comunes de APIs	11
2.3.1. REST: Transferencia de Estado Representacional	12
2.3.2. SOAP: <i>Simple Object Acces Protocol</i>	12
2.3.3. RPC: <i>Remote Procedure Call</i>	13
2.4. Protocolo HTTP	13
2.5. Herramientas de Generación Automática de APIs	14
2.5.1. Swagger/OpenAPI	15
2.5.2. DreamFactory	16
2.5.3. Postman	18
2.5.4. Mocky	19

2.6. Comparativa de herramientas actuales con la solución propuesta .	20
2.7. Tecnologías subyacentes	21
3. Solución propuesta / método	25
3.1. APIzzy: especificación de funcionalidades esperadas	25
3.1.1. Frontend	26
3.1.2. Backend	27
3.2. Formato JSON del Formulario	28
3.3. Definición formal de Requisitos y Casos de Uso	30
3.3.1. Solución Propuesta	40
4. Validación y pruebas	51
4.1. Validaciones Frontend:	51
4.1.1. Pruebas unitarios	52
4.1.2. Pruebas de integración	53
4.2. Validaciones Backend	53
4.2.1. Pruebas unitarios	54
4.2.2. Pruebas de integración	55
5. Gestión del proyecto	57
5.1. APIzzy: especificación de funcionalidades esperadas	57
5.1.1. Impacto Socio-Económico	57
5.1.2. Metodología Utilizada	58
5.1.3. Planificación Temporal y Diagrama de Gantt:	60
5.1.4. Presupuesto:	61
6. Conclusiones	63
7. Líneas de trabajo futuro	65
8. Anexo - English content	71
8.1. Introduction	71

8.2. Context	71
8.3. Problem	73
8.4. Objectives	73
8.5. State of the Art	74
8.6. History and Evolution of APIs	74
8.7. Types of APIs	74
8.8. Common API Protocols and Architectures	75
8.8.1. REST: Representational State Transfer	75
8.8.2. SOAP: <i>Simple Object Access Protocol</i>	76
8.8.3. RPC: <i>Remote Procedure Call</i>	76
8.9. HTTP Protocol	76
8.10. Automatic API Generation Tools	77
8.10.1. Swagger/OpenAPI	77
8.10.2. Postman	77
8.10.3. Mocky	77
8.11. Comparison of Current Tools with the Proposed Solution	78
8.12. Underlying Technologies	78
8.13. Proposed Solution / Method	79
8.13.1. Proposed Solution	79
8.14. Validation and Testing	82
8.15. Frontend Validations:	82
8.15.1. Unit tests	83
8.15.2. Integration tests	84
8.16. Backend Validations	84
8.16.1. Unit tests	85
8.16.2. Integration tests	86
8.17. Acceptance tests	86
8.18. Conclusions	87
8.19. Future Work Lines	88

Índice de figuras

1.1. Número de Apis por Categorías desde el 2011 (fuente: ProgrammableWeb, consultado el 28 de Agosto 2023).	2
2.1. Popularidad en los últimos años de Python vs Java en forma logarítmica (fuente: PYPL, consultado el 28 de Agosto 2023).	22
3.1. Carpeta <i>src</i> APIzzy.	41
3.2. Agregado o eliminado	42
3.3. Selección de método.	43
3.4. Selección de formato.	43
3.5. Propiedades para validaciones.	44
3.6. Formato no válido.	44
3.7. Entrada vacía.	45
3.8. Diagrama de clases backend APIzzy.	47
3.9. Interconexión simple de los módulos de APIzzy.	50
5.1. Distribución de tareas y tiempos del proyecto (diagrama de Gantt).	60
8.1. Number of APIs by Categories since 2011 (source: ProgrammableWeb).	72

Índice de tablas

3.1. Estructura empleada en la formalización de requisitos.	30
3.2. Requisito funcional RF01.	31
3.3. Requisito funcional RF02.	31
3.4. Requisito funcional RF03.	32
3.5. Requisito funcional RF04.	32
3.6. Requisito funcional RF05.	32
3.7. Requisito funcional RF06.	32
3.8. Requisito funcional RF07.	33
3.9. Requisito no funcional RNF01.	33
3.10. Requisito no funcional RNF02.	33
3.11. Requisito no funcional RNF03.	34
3.12. Requisito no funcional RNF04.	34
3.13. Requisito no funcional RNF05.	34
3.14. Requisito no funcional RNF06.	34
3.15. Requisito no funcional RNF07.	35
3.16. Requisito no funcional RNF08.	35
3.17. Requisito no funcional RNF09.	35
3.18. Requisito no funcional RNF10.	35
3.19. Requisito no funcional RNF11.	36
3.20. Requisito no funcional RNF12.	36
3.21. Plantilla casos de uso.	36

3.22.Caso de uso CU01.	37
3.23.Caso de uso CU02.	37
3.24.Caso de uso CU03.	37
3.25.Caso de uso CU04.	38
3.26.Caso de uso CU05.	38
3.27.Caso de uso CU06.	39
3.28.Caso de uso CU07.	39
3.29.Matriz de Trazabilidad	40
5.1. Tabla costes de personal.	61
5.2. Tabla costes de software y hardware.	61
5.3. Tabla costes indirectos.	62
5.4. Tabla costes totales.	62

1

Introducción

En el ámbito tecnológico actual, las Interfaces de Programación de Aplicaciones (API), son piezas fundamentales en la interoperabilidad y el intercambio de datos entre sistemas. A pesar de su importancia, la creación y gestión de APIs, sigue siendo una tarea desafiante, que a menudo necesita de habilidades y conocimientos técnicos avanzados. La motivación principal detrás de este esfuerzo radica en la necesidad de proporcionar a los desarrolladores una herramienta sencilla que reduzca la complejidad, el tiempo y el esfuerzo empleado por los mismos a la hora de diseñar y desplegar APIs en un lenguaje de programación concreto como es Python.

En el presente capítulo se introduce el contexto en el que se enmarca el desarrollo de la herramienta apenas mencionada, así como el problema al que pone solución, la motivación que ha llevado al desarrollo de la misma y los objetivos que se pretenden cumplir.

1.1 Contexto

Actualmente, las Interfaces de Programación de Aplicaciones (APIs) se han consolidado como elementos esenciales a la hora de desplegar una Arquitectura Orientada a Servicios (SOA) [1]. Estas herramientas permiten la conexión o interacción entre dos sistemas de software entre sí, posibilitando la creación de soluciones más sólidas y potentes [2]. Las APIs actúan como puentes entre dife-

rentes plataformas y aplicaciones, permitiendo que las mismas 'conversen' para compartir información y funcionalidades de manera efectiva.

En los últimos años muchas organizaciones se han decantado por el uso de APIs en su infraestructuras, convirtiéndose en un recurso esencial en el proceso de transformación digital para cualquier empresa debido a los múltiples beneficios que estas aportan [3]. Además, el auge de tecnologías como son los dispositivos móviles, aplicaciones, e-commerce, startups tecnológicas y de tendencias emergentes como la nube, IoT y BigData han hecho que las APIs sean un componente indispensable en cualquier arquitectura.

Las APIs satisfacen la demanda de agilidad y capacidad de respuesta que las empresas requieren en la era digital. Un dato revelador es que ProgrammableWeb, un reconocido repositorio de APIs, ha registrado un marcado aumento en el uso de APIs, destacando categorías como herramientas, redes sociales, finanzas, e-commerce, entre otras que se pueden observar en la figura 1.1 [4].

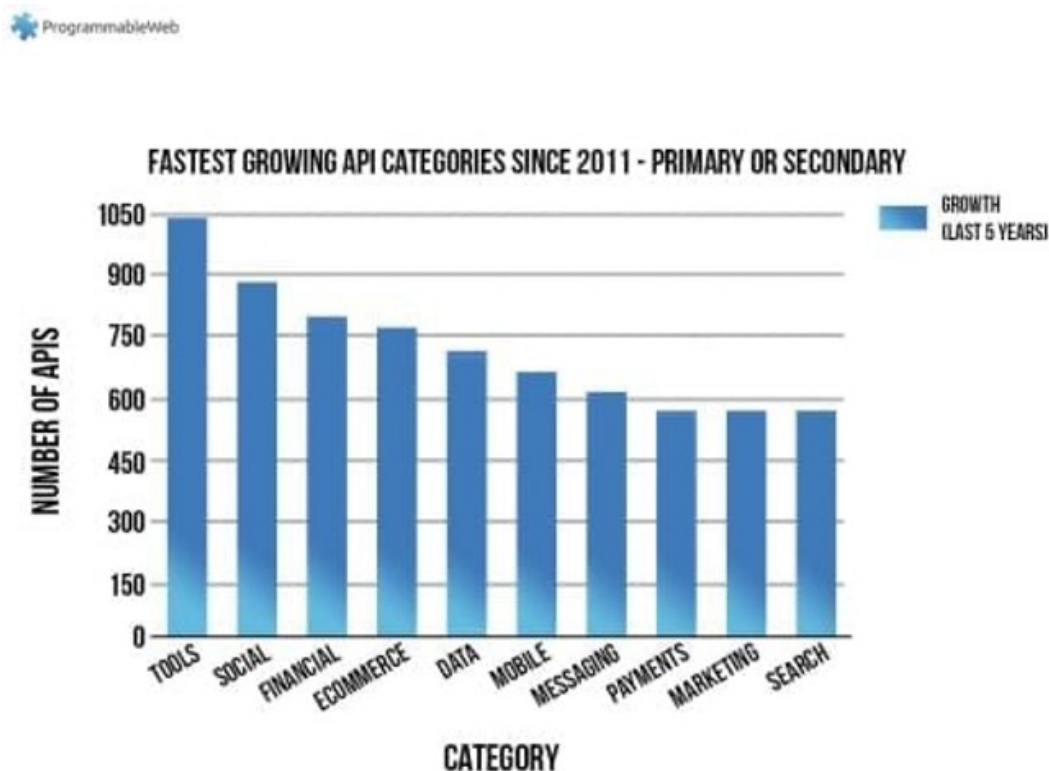


Figura 1.1. Número de Apis por Categorías desde el 2011 (fuente: ProgrammableWeb, consultado el 28 de Agosto 2023).

Las APIs bien gestionadas permiten a las empresas innovar internamente, reducir tiempos de despliegue de infraestructuras, optimizar recursos, desarrollar nuevos modelos de negocio y mejorar la experiencia del cliente. Dada la crecien-

La importancia y complejidad de las APIs en el panorama tecnológico actual, es esencial contar con herramientas que simplifiquen su creación, agilizando la labor en el proceso de desarrollo. Es crucial reconocer que no todos los miembros de una organización poseen las habilidades técnicas, los conocimientos o la familiaridad con la codificación necesaria para implementar una API. Esta realidad se vuelve particularmente evidente cuando las necesidades del negocio requieren la rápida implementación de soluciones basadas en APIs. Es por ello, que la disponibilidad de una herramienta que utilice formularios interactivos y de fácil comprensión no sólo es útil, sino de vital importancia. Tal aplicación minimizaría la barrera técnica, permitiendo a un rango más amplio de profesionales colaborar en la consecución de objetivos organizacional es mediante la implementación eficiente de APIs.

1.2 Problema

Actualmente, existen pocas herramientas interactivas y sencillas de usar por los usuarios que permitan generar APIs con especificaciones determinadas de forma automática. Desarrollar estas APIs puede ser una labor ardua, que no solo consume tiempo valioso, sino que también demanda habilidades específicas en programación. En muchas organizaciones, los equipos son multidisciplinarios y no todos sus integrantes poseen competencias en este ámbito. Esta disparidad en habilidades puede erigirse como un obstáculo, resultando en demoras, incremento de costos y, en ocasiones, en soluciones que no son óptimas o que comprometen la seguridad del sistema del que forman parte. Por ello, surge la pregunta: ¿cómo facilitar la creación y gestión de APIs de manera eficiente, reduciendo la curva de aprendizaje y permitiendo que profesionales sin formación técnica profunda puedan participar activamente en su implementación? El objetivo de este TFG es abordar esta problemática, proponiendo una solución innovadora y práctica mediante el desarrollo de una herramienta que, proporcionados una serie de parámetros, genere una plantilla que implemente una API operativa con las funcionalidades solicitadas.

1.3 Motivación

Como se ha mencionado anteriormente, la Arquitectura Orientada a Servicios es una de las tendencias digitales hoy en día y en las que se basan modelos más modernos como los microservicios. Estas arquitecturas emplean API como elemento de cohesión y comunicación entre las diferentes aplicaciones que conforman la arquitectura completa en su conjunto. Por ello las empresas deben contar con desarrolladores capaces de implementar y mantener este componen-

te indispensable hoy en día.

Debido a la experiencia laboral de la autora durante el desarrollo de sus prácticas, se ha comprobado que la realidad no es así y que la mayoría de los profesionales que hoy en día se encuentran en activo, no tienen una formación sólida en el desarrollo de APIs. Esto se traduce en una problemática innecesaria a la hora de desarrollar estos módulos, por lo que es evidente la necesidad de una herramienta que facilite el desarrollo de los mismos. Su uso se puede extender desde profesionales formados que quieran agilizar el proceso de desarrollo mediante la creación de APIs a partir de parámetro, hasta profesionales con una formación más limitada que requieran de su uso a la hora de desarrollar y formarse en la creación de APIs en Python.

Por otra parte, el desarrollo de la herramienta propuesta supone una gran oportunidad para aplicar y consolidar conocimientos en múltiples disciplinas adquiridas en el grado en Ingeniería Informática cursado. Estas dos razones son la motivación que ha impulsado el desarrollo de la herramienta presentada en esta memoria.

1.4 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es diseñar y desarrollar APIzzy, una herramienta interactiva y original que enfrenta un desafío persistente en el panorama tecnológico: la compleja tarea de crear y gestionar APIs. A pesar de que las APIs son vitales para la interacción digital moderna, su desarrollo sigue siendo una barrera, especialmente para aquellos que no poseen un profundo conocimiento técnico.

Con APIzzy, se introduce una metodología revolucionaria que no solo simplifica el proceso de creación de APIs, sino que también democratiza su desarrollo. Esto permite que individuos, independientemente de su nivel técnico en programación, puedan diseñar APIs con especificaciones concretas.

El valor añadido de APIzzy radica en su capacidad para ofrecer una solución práctica y efectiva apta para entornos profesionales, como empresas. Esta innovadora propuesta tiene el potencial de cambiar el juego, cerrando la brecha entre la creciente demanda de interconexión digital y la insuficiente oferta de habilidades especializadas para desarrollar dichas conexiones vitales. A través de este proyecto, se busca no solo crear una herramienta tecnológica, sino también aportar significativamente al ecosistema digital, redefiniendo cómo abordamos y entendemos el mundo de las APIs.

1.5 Marco Regulator

A continuación se presenta el marco regulator o conjunto de pautas para este proyecto:

- **Derechos de autor y propiedad intelectual:** Dada la naturaleza innovadora del proyecto APIzzy, es esencial proteger cualquier innovación o desarrollo. Esto implica no solo evitar infracciones de derechos de terceros, sino también registrar y proteger los derechos propios de la solución desarrollada. En conforme al Ley de Propiedad Intelectual (Real Decreto Legislativo 1/1996, de 12 de abril) regula los derechos de autor, disponiendo protección en la obra y autores [5].
- **Privacidad y protección de datos:** En conformidad con el Reglamento General de Protección de Datos (GDPR) de la Unión Europea, cualquier dato personal que sea procesado por APIzzy debe ser tratado con la máxima privacidad. Esto incluye la obtención de consentimientos claros, garantizar el anonimato y el cifrado de datos, así como proporcionar mecanismos para la rectificación o eliminación de datos [6].
- **Cumplimiento normativo:** Las APIs creadas a través de APIzzy deben ser compatibles con todas las regulaciones relevantes de la industria específica para la que están destinadas. Esto puede variar desde regulaciones de salud, financieras hasta normativas medioambientales.
- **Seguridad cibernética:** La creciente amenaza de ciberataques hace esencial la incorporación de protocolos de seguridad robustos. Desde la autenticación de usuarios hasta la protección contra ataques de inyección, es vital que las APIs generadas sean seguras.
- **Licencias y uso de software:** La elección de la licencia adecuada determinará cómo y quién puede usar, modificar o distribuir las APIs generadas. Conforme a la Ley 3/2008, del 23 de diciembre, sobre Medidas Fiscales y Administrativas: se aborda regulaciones vinculadas a la licencia y utilización de programas informáticos, definiendo deberes y restricciones tanto para quienes usan el software como para quienes lo suministran [7].
- **Calidad del código:** Siguiendo estándares como ISO/IEC 25010, es posible garantizar que el código producido sea de alta calidad, lo que implica legibilidad, mantenibilidad, eficiencia, entre otros [8].
- **Documentación:** Más allá de una documentación técnica, es esencial ofrecer guías de usuario y casos prácticos que faciliten la adopción y correcta utilización de las APIs por parte de desarrolladores y empresas.

- **Pruebas y validación:** La implementación de pruebas unitarias, integración y de sistema, garantizarán que las APIs generadas estén libres de errores y se comporten según lo previsto.
- **Monitorización y mantenimiento:** Utilizar herramientas de monitorización para detectar problemas en tiempo real, garantizando una rápida respuesta ante fallos y asegurando un funcionamiento óptimo a largo plazo.
- **Transparencia y ética:** Es fundamental mantener una política de transparencia, informando a los usuarios sobre cómo funciona APIzzy, las posibles limitaciones de las APIs generadas y garantizando que no se generen sesgos o discriminaciones no intencionadas.

1.6 Estructura del Documento

Para llevar a cabo este trabajo, hemos adoptado una metodología convencional, y a continuación, se detallan los capítulos que lo integran:

- **Introducción:** En esta sección se presenta el panorama actual de las APIs y su importancia en el mundo digital y empresarial. Se da una breve descripción de la experiencia personal en un banco que sirvió de motivación para el proyecto. Finalmente, se establecen la motivación y el objetivo del TFG, así como cualquier marco regulador o contexto que pueda ser relevante para entender el alcance del trabajo.
- **Estado del Arte:** Esta parte proporciona una revisión detallada de la tecnología actual y las herramientas relacionadas con la creación y gestión de APIs. Se destacarán las soluciones existentes, sus ventajas y limitaciones, y se hará una comparación con la herramienta propuesta en este trabajo.
- **Solución Propuesta/Método:** Aquí se describe en profundidad la herramienta que se propone para facilitar el desarrollo de APIs. Se detallan las características y funcionalidades que ofrecerá, los requisitos que deberá cumplir y los posibles formatos de entrada y salida. Además, se presentarán casos de uso, el diseño de la arquitectura de la solución, parámetros configurables y cómo se gestionarán posibles errores.
- **Validación y Pruebas:** En este capítulo se describen los diferentes métodos de validación utilizados durante el desarrollo de la herramienta. Se hablará sobre pruebas unitarias, pruebas de integración y pruebas de aceptación, señalando cómo se aseguró que la herramienta cumpliera con los requisitos y expectativas establecidas.

1.6 Estructura del Documento

- **Conclusiones:** Se expondrán las conclusiones generales del proyecto, resaltando los logros, desafíos y aprendizajes obtenidos durante el desarrollo del TFG. Además, se compartirán reflexiones personales sobre la experiencia y cómo esta ha enriquecido la formación académica.
- **Líneas de Trabajo Futuro:** Para finalizar, se presentarán ideas y recomendaciones para expandir o mejorar la herramienta en el futuro, identificando oportunidades de desarrollo adicional, posibles extensiones o nuevas funcionalidades que podrían ser de interés.

2

Estado del arte

En este capítulo se analiza el contexto en el que se enmarca el desarrollo de la herramienta presentada en el capítulo introductorio. Se evaluará la situación actual de herramientas similares con el fin de obtener una visión completa sobre los avances más significativos en el campo de estudio relacionado. Mediante este análisis se pretende establecer una base sólida para el sistema a desarrollar, proponiendo mejoras sustanciales con respecto a otras soluciones existentes en el mercado actual.

De la misma manera, será posible identificar puntos débiles y fallos en las plataformas existentes a día de hoy, permitiendo diseñar una plataforma mejorada. Finalmente, se evaluarán los marcos de desarrollo más empleados en herramientas similares, con la finalidad de que el sistema final sea escalable a futuro, mantenible y flexible ante la incorporación de nuevas funcionalidades.

2.1 Historia y Evolución de las API

Las Interfaces de Programación de Aplicaciones, más conocidas por sus siglas en inglés como API, representa uno de los componentes fundamentales en el desarrollo de software moderno, siendo el medio de comunicación principal entre módulos de un mismo sistema. Mediante estas interfaces pueden interconectarse módulos completamente independientes, sin importar el lenguaje de programación o la plataforma en la que estén desplegados. Este intercambio de

información se realiza de manera estructurada, permitiendo que los sistemas puedan entender y procesar los datos que se transmiten, lo que facilita la integración de servicios y la creación de aplicaciones más robustas y versátiles [9].

Desde sus primeros días, las APIs se diseñaron como simples herramientas que servían para conectar funciones específicas dentro de una aplicación o entre aplicaciones estrechamente relacionadas. Sin embargo, el auge de la era digital y la necesidad de una web más conectada llevó a la evolución de estas interfaces. Empezaron a surgir APIs más generalizadas, diseñadas para servir a múltiples aplicaciones y permitir la conexión entre diferentes plataformas y dispositivos.

El comercio electrónico, las redes sociales y, más recientemente, el Internet of Things (IoT) son ejemplos claros de sectores que se han beneficiado enormemente de la utilización de APIs. Estas permiten que aplicaciones móviles, sitios web y dispositivos conectados compartan información en tiempo real, mejorando la experiencia del usuario y ofreciendo servicios más personalizados y eficientes [10].

Actualmente las APIs ya no están limitadas al ámbito de los desarrolladores o a las grandes empresas tecnológicas. Se han infiltrado en casi todos los rincones de la empresa moderna, impulsando la automatización y la eficiencia en todos los departamentos, potenciando el negocio y conectando estos con los clientes. Desde recursos humanos hasta finanzas, pasando por ventas y marketing, llegando a los clientes y optimizando las operaciones, estas interfaces permiten que las aplicaciones se comuniquen entre sí, eliminando la necesidad de procesos manuales repetitivos y propensos a errores.

Por ejemplo, el departamento de recursos humanos podría utilizar una API para conectar su sistema de gestión de personal con una herramienta de nóminas, garantizando que los datos se sincronicen en tiempo real y se reduzcan los errores en el proceso de pago. Del mismo modo, el departamento de ventas podría integrar su software CRM con herramientas de análisis y pronóstico para obtener una visión más clara del rendimiento de ventas y las tendencias futuras [11]. Esta omnipresencia de las APIs en la empresa resalta su importancia, no solo como herramienta de desarrollo, sino también como facilitador esencial de operaciones empresariales eficientes y modernas. Su capacidad para conectar sistemas dispares ha llevado a una revolución en la forma en que las empresas operan, permitiéndoles ser más ágiles, basadas en datos y centradas en el cliente [10].

Es evidente que las APIs no son simplemente una moda pasajera en el mundo tecnológico; son una pieza fundamental en el rompecabezas de la digitalización, permitiendo la creación de un ecosistema digital más interconectado y colaborativo.

2.2 Tipos de APIs

- **Públicas:** una API pública, también conocida como API abierta o API externa, es una interfaz de programación de aplicaciones de código abierto a las que se pueden acceder con el protocolo HTTP y que han sido diseñadas para ser accesibles por cualquier desarrollador [12]. Las ventajas de las APIs públicas incluyen la capacidad de ampliar la funcionalidad de una aplicación o plataforma a una gama más amplia de usuarios, promover la innovación y permitir nuevas formas de colaboración entre aplicaciones y plataformas. Sin embargo, también es crucial gestionar y proteger adecuadamente estas API para garantizar la seguridad y la privacidad de los datos [9].
- **Partner:** a diferencia de las APIs abiertas, las APIs de socio están destinadas a ser utilizadas por un grupo específico de desarrolladores o empresas asociadas. Esto permite una colaboración más estrecha entre las empresas, garantizando que la API se utilice de una manera que sea beneficiosa para ambas partes. Las APIs de socio también pueden tener un nivel de seguridad adicional o características personalizadas específicas para el socio [9], [12].
- **Internas:** las APIs internas son interfaces de programación de aplicaciones que permanecen ocultas a los usuarios externos [12]. Las grandes organizaciones a menudo desarrollan aplicaciones y sistemas propios que deben interactuar entre sí, generalmente diseñadas para el funcionamiento de una empresa particular [13]. Las APIs internas facilitan esta comunicación, permitiendo una mayor eficiencia y cohesión entre los sistemas internos.
- **Compuestas:** son APIs que combinan varias API de datos o servicios [12]. A medida que las aplicaciones se han vuelto más complejas, ha surgido la necesidad de combinar múltiples servicios y funciones en una única operación. Las APIs compuestas satisfacen esta necesidad, permitiendo la ejecución de varias tareas y llamadas en una única solicitud [9].

2.3 Protocolos y arquitecturas comunes de APIs

Las APIs intercambian comandos e información, para ello se necesitan protocolos y arquitecturas claras. Actualmente existen tres categorías principales: REST, RPC Y SOAP. Son formatos con características distintas utilizados para distintos objetivos [9].

2.3.1 REST: Transferencia de Estado Representacional

Es una API que cumple con los principios de diseño del estilo de arquitectura REST o Representational State Transfer [14]. Este estilo arquitectónico ha ganado popularidad debido a su simplicidad y eficiencia, siendo esenciales en el panorama actual de desarrollo de software [13]. El cliente envía solicitudes al servidor mediante Protocolo de Transferencia de Hipertexto HTTP, donde el servidor emplea las solicitudes para realizar funciones internas como GET, PUT, POST, DELETE, entre otras, y devolver los datos de salida al cliente. La principal característica de las API REST es que no tienen estado, o lo que es lo mismo, el servidor no almacena ninguno de los datos al procesar las peticiones [15]. Un beneficio destacado de el uso del paradigma API REST es su adaptabilidad, en particular porque pueden trabajar con diversos formatos de datos ampliamente utilizados como JSON (Notación de Objetos de JavaScript) y XML (eXtensible Markup Language) [16].

Adicionalmente, el diseño sin estado de las API REST garantiza que el rendimiento del sistema sea óptimo, ya que cada solicitud es tratada como una transacción nueva sin relación con cualquier solicitud anterior. También proporciona una mayor escalabilidad, permitiendo que las aplicaciones gestionen un gran número de solicitudes de manera eficiente. Otro aspecto destacable es la facilidad con la que estas APIs pueden ser integradas en diferentes plataformas y lenguajes de programación debido a su naturaleza basada en estándares web. En la práctica, esto significa que aplicaciones móviles, sitios web, y otras aplicaciones de software pueden consumir y trabajar con datos provenientes de una API REST sin importar las tecnologías subyacentes [14].

2.3.2 SOAP: *Simple Object Acces Protocol*

Aunque es más antiguo que REST, SOAP o Simple Object Access Protocol sigue siendo ampliamente utilizado, especialmente en aplicaciones empresariales [9]. Es un protocolo ligero para el intercambio de información en entornos distribuidos y descentralizados. SOAP se basa en el transporte de mensajes de un remitente a un destinatario. Además, ofrece características adicionales, como la seguridad a través de WS-Security y para consultas UDDI (Universal Description, Discovery, and Integration). Sin embargo, es más complejo y requiere más ancho de banda y recursos [17].

2.3.3 RPC: *Remote Procedure Call*

Antes de la popularidad de REST y SOAP, RPC (Simple Object Access Protocol) dominaba el mundo de las APIs. Aunque menos común hoy en día, sigue siendo una opción viable, especialmente cuando se necesita una comunicación directa entre sistemas [9]. Estas APIs permiten a los desarrolladores llamar a funciones remotas en servidores externos [18].

2.4 Protocolo HTTP

HTTP (Protocolo de Transferencia de HiperTexto) es el protocolo de comunicación subyacente utilizado por la World Wide Web (WWW) y forma la base de la mayoría de las interacciones en línea. Desarrollado por el World Wide Web Consortium (W3C) y la Internet Engineering Task Force (IETF), HTTP es la base de cualquier intercambio de datos en la web, permitiendo la comunicación entre clientes y servidores mediante la transferencia de mensajes en formato de texto plano, ya que realiza peticiones de datos y de recursos. El protocolo HTTP trabaja sobre conexiones TCP/IP, donde el protocolo Localizador Uniforme de Recursos (TCP) es el encargado de mantener las comunicaciones y garantizar que se cumpla el intercambio de datos sin errores después de que se establezca la conexión [19].

Las APIs, especialmente las API REST, suelen operar sobre HTTP debido a su facilidad de uso. Cuando un cliente, como una aplicación móvil o un navegador web, desea interactuar con un servicio remoto, hace una solicitud HTTP a una URL (Protocolo de Transferencia de HiperTexto) específica. Esta solicitud puede incluir información adicional en forma de encabezados o datos en el cuerpo de la solicitud. El servidor procesa esta solicitud y devuelve una respuesta, también a través de HTTP.

HTTP opera mediante un conjunto de métodos o funciones, los más comunes son [20]:

- **GET:** este método solicita datos de un recurso. Se utiliza para la recuperación de información y solo recuperación de datos.
- **HEAD:** este método solicita datos de forma idéntica a un GET, la diferencia es que sin el cuerpo de respuesta.
- **POST:** este método envía datos para crear un nuevo recurso, generando un cambio en el estado o efectos secundarios hacia el servidor.

- **PUT:** este método actualiza un recurso existente con nuevos datos, es decir que reemplaza representaciones actuales del recurso al que se desea realizar la petición.
- **DELETE:** este método elimina un recurso específico.
- **CONNECT:** este método establece una conexión o un túnel hacia el servidor con establecido e identificado por el recurso.
- **PATCH:** este método realiza modificaciones parciales en un recurso.

Cada respuesta HTTP tiene un código de estado que indica el resultado de la solicitud. Estos códigos se dividen en varias categorías:

- **Respuestas Informativas (100-199):** proporcionan información sobre el procesamiento en curso.
- **Respuestas Exitosas (200-299):** indican que la solicitud fue procesada correctamente.
- **Redirecciones (300-399):** informan al cliente que debe tomar medidas adicionales, generalmente dirigirse a una URL diferente.
- **Errores del cliente (400-499):** indican que hubo un error en la solicitud enviada por el cliente. Por ejemplo, el error *"404 Not Found"* indica que el recurso solicitado no pudo ser encontrado.
- **Errores del servidor (500-599):** indican que el servidor falló al procesar una solicitud válida. El error *"500 Internal Server Error"* es un ejemplo común y denota un error en el servidor.

Con el auge de las aplicaciones web y móviles, la importancia de las APIs basadas en HTTP ha crecido exponencialmente. Es esencial comprender estos códigos y métodos para diagnosticar y solucionar problemas, así como para desarrollar APIs robustas y eficientes.

2.5 Herramientas de Generación Automática de APIs

Como se ha mencionado anteriormente, en la actualidad las empresas están adoptando como arquitectura principal para sus productos de software los microservicios. Puesto que este tipo de arquitectura requiere del uso de APIs, han comenzado a surgir diferentes herramientas cuyo fin reside en facilitar su implementación [21]. Las APIs, dependiendo de la naturaleza y requerimiento del

2.5 Herramientas de Generación Automática de APIs

negocio, pueden ser creadas manualmente o mediante procesos automatizados. Mientras que muchos optan por servicios REST a través de HTTP para gestionar y obtener datos, otros se inclinan por soluciones de automatización para agilizar tanto el desarrollo como la validación de sus APIs.

Como resultado de la investigación realizada, se han identificado numerosos estudios y herramientas que persiguen objetivos análogos al del presente estudio. En la siguiente sección, se hará un desglose de las principales herramientas identificadas con el fin de contextualizar y contrastar con el enfoque de este trabajo.

2.5.1 Swagger/OpenAPI

Swagger, también conocido como OpenAPI, es una herramienta y un marco de trabajo de código abierto utilizado para diseñar, desarrollar, construir, documentar y consumir APIs RESTful. Se ha consolidado como una de las herramientas más populares en la industria del software para este propósito. La especificación detrás de Swagger se ha convertido ahora en una norma conocida como OpenAPI Specification (OAS). Swagger fue desarrollado por SmartBear Software, una empresa conocida por sus soluciones en desarrollo, testeo y monitoreo de software. Aunque comenzó como una herramienta independiente, Swagger fue adquirido por SmartBear en 2015 y desde entonces ha continuado evolucionando bajo su dirección [22].

Dentro de sus funciones tenemos:

- **Documentación Interactiva:** Swagger proporciona una interfaz de usuario donde los desarrolladores y otros interesados pueden visualizar y probar la API. Esta interfaz permite a los usuarios realizar peticiones y recibir respuestas directamente desde la documentación.
- **Generación Automática de Documentación:** a medida que se desarrolla la API, Swagger genera automáticamente la documentación, asegurando que esté siempre actualizada.
- **Generación de Código:** Swagger ofrece herramientas para generar automáticamente código cliente en múltiples lenguajes de programación a partir de la definición de la API en texto.
- **Validación y Diseño:** ayuda a diseñar una API coherente y válida que cumpla con la especificación OpenAPI.

Sin embargo, este estándar tiene algunas carencias:

- **Complejidad en Grandes Proyectos:** en proyectos extensos con numerosos endpoints, mantener la documentación actualizada y coherente con Swagger puede volverse complejo. La estructura del archivo puede volverse extensa y difícil de manejar.
- **Flexibilidad Limitada:** aunque Swagger proporciona una gran cantidad de funcionalidades, puede ser menos flexible cuando se trata de necesidades específicas o personalizaciones avanzadas en la documentación o en las interfaces de usuario.
- **Curva de Aprendizaje:** a pesar de que Swagger tiene una interfaz intuitiva, puede requerir una curva de aprendizaje alta para aquellos que no están familiarizados con la especificación OpenAPI o con la estructura de documentación de Swagger.
- **Rendimiento:** la generación automática de documentación y la interfaz de usuario pueden añadir una sobrecarga, especialmente en proyectos más grandes, lo que puede afectar el rendimiento general de la API.
- **Seguridad:** expone detalles sobre la estructura y las capacidades de las APIs generadas, facilitando ataques. Si bien esto se puede mitigar con prácticas adecuadas de seguridad y configuración, sigue siendo un punto a considerar.
- **Sincronización:** a medida que una API evoluciona, mantener la documentación de Swagger sincronizada con los cambios en el código puede ser un desafío, especialmente si no se integra adecuadamente en el flujo de trabajo de desarrollo.

2.5.2 DreamFactory

DreamFactory es una plataforma que permite a los desarrolladores crear, desplegar y gestionar APIs RESTful sin necesidad de escribir código. Proporciona una manera automática de generar APIs a partir de múltiples fuentes de datos, como bases de datos, servicios de archivos y servicios web externos. La versatilidad de DreamFactory radica en su capacidad para generar APIs de manera rápida y sencilla, siendo especialmente útil para proyectos que requieren un rápido prototipado o para empresas que desean digitalizar sus recursos sin incurrir en desarrollos largos y costosos [23].

DreamFactory Software, Inc. es la empresa detrás de esta herramienta. Fundada en 2012, se ha consolidado como una solución de referencia en el mundo de las APIs debido a su enfoque en la automatización y la facilidad de uso. Su

2.5 Herramientas de Generación Automática de APIs

plataforma es ofrecida tanto en versiones de código abierto como en soluciones empresariales con características avanzadas.

Algunas de las funcionalidades que implementa DreamFactory son:

- **Generación Automática de APIs:** DreamFactory puede convertir cualquier fuente de datos en una API RESTful totalmente funcional sin necesidad de escribir código.
- **Limitaciones en la Personalización:** en ciertos casos de uso específicos pueden requerirse personalizaciones que no son fáciles de implementar en la plataforma.
- **Precio:** las versiones empresariales de DreamFactory, que ofrecen características más avanzadas y soporte técnico, pueden resultar costosas para pequeñas empresas o startups con presupuestos limitados. Es vital considerar el retorno de inversión al adoptar esta herramienta en contextos empresariales.
- **Integración de Múltiples Fuentes de Datos:** puede conectarse y generar APIs a partir de diversas bases de datos y servicios.
- **Escalabilidad:** DreamFactory es capaz de manejar un gran número de solicitudes simultáneas, adecuándose tanto a proyectos pequeños como a grandes implementaciones empresariales.
- **Monitorización y Caché:** proporciona herramientas para monitorizar el uso de la API y ofrece funcionalidades de caché para mejorar el rendimiento.

De nuevo, no todo son ventajas, si no que también DreamFactory cuenta con algunos puntos que juegan en su contra:

- **Curva de Aprendizaje:** a pesar de su enfoque en la generación automática, algunos usuarios pueden encontrar un desafío inicial al adaptarse a su interfaz y características debido a la gran cantidad de funcionalidades que incluye.
- **Autenticación y Seguridad:** incluye características integradas de autenticación y autorización, soportando JWT (JSON Web Tokens) y OAuth, entre otros.
- **Curva de Aprendizaje:** a pesar de que Swagger tiene una interfaz intuitiva, puede requerir una curva de aprendizaje alta para aquellos que no están familiarizados con la especificación OpenAPI o con la estructura de documentación de Swagger.

- **Interfaz de Usuario:** aunque DreamFactory se ha esforzado por tener una interfaz intuitiva, algunos usuarios han señalado que ciertos aspectos de la plataforma no son sencillos de entender, especialmente para usuarios con conocimientos limitados sobre el desarrollo de APIs.
- **Documentación:** DreamFactory no cuenta con una documentación extensa, por lo que en algunos casos es necesario recurrir a foros de terceros para obtener información sobre la herramienta.
- **Monitorización y Caché:** proporciona herramientas para monitorizar el uso de la API y ofrece funcionalidades de caché para mejorar el rendimiento.

2.5.3 Postman

Postman es una plataforma popular utilizada para desarrollar, probar, documentar y monitorear APIs. Originalmente diseñada como una extensión de Chrome para probar APIs, Postman ha evolucionado hasta convertirse en una solución integral para todo el ciclo de vida de desarrollo de APIs. Debido a su amplio conjunto de herramientas y características, Postman se utiliza en una gran variedad de escenarios. Proporciona soluciones para prácticamente todas las etapas del desarrollo y mantenimiento de una API [24].

Postman fue fundado por Abhinav Asthana, Ankit Sobti y Abhijit Kane en 2014, con el objetivo de simplificar el proceso de creación y prueba de APIs. Desde entonces, ha crecido exponencialmente y se ha convertido en una herramienta esencial para muchos desarrolladores de APIs en todo el mundo. La compañía tiene su sede en San Francisco y ha obtenido financiamiento de inversores de renombre [25].

Entre las numerosas funcionalidades que implementa el sistema, destacan:

- **Testing de APIs:** permite a los desarrolladores enviar peticiones a APIs y ver las respuestas en tiempo real. Proporciona un entorno de prueba que incluye la capacidad de ejecutar conjuntos de pruebas y automatizar flujos de trabajo.
- **Documentación Automática:** genera y mantiene documentación de APIs de manera automática, la cual es accesible y fácil de entender para los consumidores de la API.
- **Monitorización:** permite programar pruebas para monitorizar el rendimiento y funcionamiento de las APIs de manera regular.
- **Mock Servers** ayuda a crear servidores simulados para probar endpoints de una API antes de que esté en producción.

2.5 Herramientas de Generación Automática de APIs

- **Colaboración en Equipo:** facilita la colaboración entre equipos con características como compartir colecciones, entornos y otros datos.

A pesar de los beneficios que el uso de Postman conlleva, existen también algunas limitaciones como las enumeradas a continuación:

- **Rendimiento:** en ocasiones, especialmente con grandes conjuntos de datos, Postman puede experimentar problemas de rendimiento.
- **Limitaciones en la Versión Gratuita:** algunas de las características avanzadas solo están disponibles en las versiones de pago.
- **No es un verdadero entorno de desarrollo:** Postman es excelente para probar y documentar APIs, pero no es una herramienta de desarrollo de APIs. Los desarrolladores aún necesitan otras herramientas o plataformas para crear la lógica subyacente de la API.
- **Carece de Integración Continua/Despliegue Continuo** Postman no tiene capacidades integradas de CI/CD. Aunque se puede integrar con otras herramientas, puede requerir esfuerzo adicional para automatizar completamente el flujo de trabajo de la API.
- **Limitaciones de Seguridad:** mientras que Postman proporciona algunas características de seguridad, la herramienta podría no ser adecuada para pruebas de seguridad exhaustivas o para validar la robustez de las APIs en escenarios de seguridad avanzados.

2.5.4 Mocky

Mocky es una herramienta online que permite a los desarrolladores simular respuestas HTTP de una API. Su principal propósito es facilitar la creación de "mocks."° simulacros de respuestas de una API, lo que es especialmente útil durante las etapas de desarrollo y pruebas. A diferencia de otras herramientas que requieren la instalación de software adicional, Mocky opera completamente desde el navegador, lo que simplifica su uso. Además una herramienta de código abierto y su código está disponible en GitHub [26].

Al ser una herramienta online, es mucho más limitada que las mencionadas anteriormente. Algunas de las funcionalidades que cumple son las siguientes:

- **Simulación de Respuestas:** Mocky permite generar enlaces URL que devuelven respuestas HTTP específicas. Estas respuestas pueden ser personalizadas en términos de código de estado, encabezados y cuerpo del mensaje.

- **Configuración Personalizable:** los usuarios pueden definir el tipo de respuesta (como JSON, XML, texto plano, etc.) y configurar otros parámetros según las necesidades del proyecto.
- **Interfaz Intuitiva:** la plataforma brinda una interfaz de usuario simple e intuitiva que facilita la creación de simulacros de API en pocos pasos. rrolladores aún necesitan otras herramientas o plataformas para crear la lógica subyacente de la API.

Por el contrario, a pesar de contar con la ventaja de no necesitarse ningún tipo de instalación para su uso, Mocky tiene las siguientes limitaciones:

- **No Genera Código de la API:** a diferencia de algunas herramientas más avanzadas, Mocky no ofrece la capacidad de generar el código fuente de la API. Es principalmente una herramienta de simulación, lo que significa que es ideal para probar, pero no para construir una API desde cero.
- **Limitaciones en Respuestas Simuladas:** aunque Mocky es poderoso, tiene ciertas limitaciones en cuanto a la complejidad de las respuestas que puede simular en comparación con soluciones más robustas.
- **No Sustituye a APIs Reales:** a pesar de su utilidad en el desarrollo y pruebas, no debe ser considerado como un reemplazo para interactuar y probar con APIs reales en escenarios de producción.

2.6 Comparativa de herramientas actuales con la solución propuesta

Tras analizar diversas herramientas disponibles para la gestión y creación de APIs, se observa una tendencia clara: la mayoría de estas herramientas están orientadas primordialmente hacia la administración y despliegue virtual de APIs más que a la generación automática de código funcional. Estas soluciones, aunque robustas, no proporcionan una ruta sencilla para que usuarios sin profundos conocimientos técnicos generen código de API listo para ser implementado en sus páginas web, proyectos y demás aplicaciones.

Aunque algunas de estas herramientas intentan ofrecer interfaces amigables, como la incorporación de formularios para aquellos usuarios con limitados conocimientos en programación, en muchas instancias, se requiere que el usuario recurra a documentación adicional o posea ciertos conocimientos base para aprovecharlas al máximo.

En este contexto, surge APIzzzy. Este proyecto se concibe con un objetivo claro: ofrecer una experiencia genuinamente orientada al usuario. APIzzzy elimina la barrera técnica al proveer una plataforma donde, gracias a su infraestructura subyacente, los usuarios pueden generar APIs RESTful con operaciones GET, POST, DELETE y PUT sin tener que lidiar con complejidades técnicas. Gracias a la implementación de formularios intuitivos para recopilar la información requerida, APIzzzy se posiciona no solo como una herramienta dinámica, sino también amigable y accesible para cualquier usuario, independientemente de su nivel de conocimientos técnicos.

2.7 Tecnologías subyacentes

Aunque existen diversas herramientas de administración, como Postman, y plataformas de despliegue de APIs, como Mocky, ya destacado anteriormente las diferencias fundamentales entre ellas y APIzzzy, es crucial determinar los lenguajes más adecuados para su creación. En este sentido, se buscó un lenguaje de programación para el Backend que fuese de alto nivel y, preferentemente, orientado a objetos. Esto se debe a que APIzzzy requiere un código que sea dinámico, reutilizable, bien estructurado y fácil de mantener.

De acuerdo con múltiples fuentes [27]-[29], los principales lenguajes orientados a objetos incluyen Java, Python, R, SQL y C/C++. Si bien R y SQL están orientados principalmente al análisis de datos y C/C++ es esencial para tareas computacionalmente intensivas en ciencia de datos, esto nos lleva a considerar a Python y Java como las opciones más viables para el desarrollo de este proyecto.

Aunque Java podría considerarse como una opción viable para el desarrollo del backend, dada su robustez, portabilidad y amplio uso en aplicaciones empresariales, tiene una curva de aprendizaje más pronunciada y un desarrollo más verboso en comparación con otros lenguajes. Además, Java a menudo requiere una mayor configuración y gestión de dependencias. Por otra parte, Python ofrece una serie de ventajas para el desarrollo del backend en un proyecto de creación dinámica de APIs. Algunas de las razones por las cuales Python resulta beneficiosa en este contexto son:

- **Sintaxis Clara y Legible** Python es conocido por su código claro y legible, lo que facilita la escritura, el mantenimiento y la colaboración en proyectos de software.
- **Librerías y Frameworks:** Python cuenta con una vasta colección de librerías y frameworks, lo que permite a los desarrolladores acceder a herramientas preexistentes y soluciones específicas para distintos desafíos. Un

ejemplo es Flask-RESTful, una extensión de Flask, que facilita la creación rápida de APIs RESTful. Esto reduce el tiempo y el esfuerzo necesario para establecer y gestionar endpoints de una API.

- **Versatilidad:** Python es adecuado para una amplia gama de aplicaciones, desde desarrollo web hasta ciencia de datos y aprendizaje automático. Esta versatilidad asegura que el lenguaje pueda adaptarse a futuras expansiones o cambios en las especificaciones del proyecto.
- **Comunidad Activa:** Python tiene una de las comunidades más grandes y activas. Esto asegura soporte continuo, una gran cantidad de recursos educativos y soluciones a problemas comunes.
- **Integración:** Python se integra fácilmente con otras tecnologías y servicios, lo que es vital al desarrollar APIs que pueden necesitar interactuar con otros sistemas o bases de datos.

En años recientes, Python ha visto un auge significativo en su uso, consolidándose como el lenguaje de programación más popular según indicadores como el de PYPL, tal como se refleja en la Figura 2.1 tomada de su sitio oficial [28]. Adicionalmente, la preferencia del cliente se inclina hacia el desarrollo en este lenguaje.

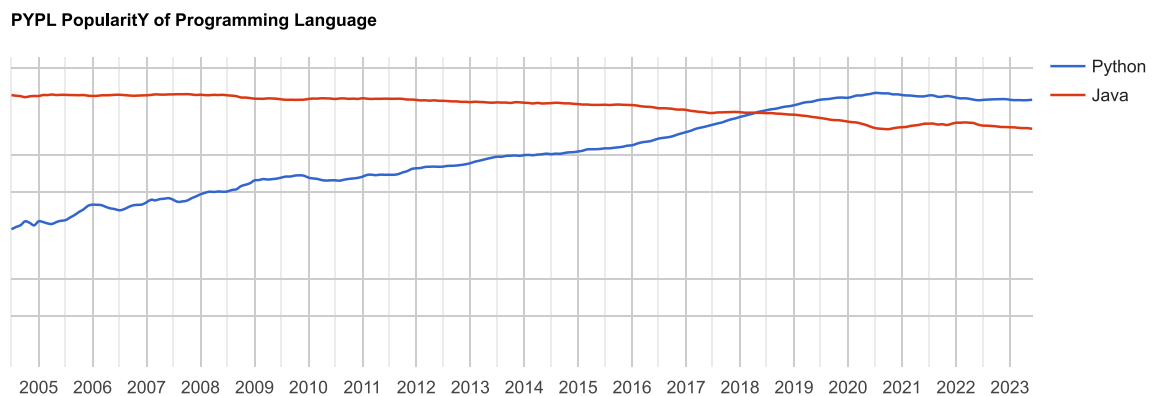


Figura 2.1. Popularidad en los últimos años de Python vs Java en forma logarítmica (fuente: PYPL, consultado el 28 de Agosto 2023).

Uno de los aspectos cruciales en el diseño de APIzzzy es ofrecer una interfaz que resulte intuitiva y amigable para el usuario. Al elegir un framework para la implementación de la interfaz de usuario, es esencial que este proporcione capacidades de interactividad en tiempo real, componentes reutilizables y un alto rendimiento. Es por ello por lo que se optó por la utilización de ReactJS.

ReactJS, desarrollado por Facebook, emerge como una elección líder en este contexto. Esta biblioteca es un marco de JavaScript que permite crear interfaces

2.7 Tecnologías subyacentes

de usuario de manera rápida y eficiente, se ha consolidado como una herramienta esencial para el desarrollo de aplicaciones web modernas debido a su naturaleza basada en componentes, lo que facilita la creación de interfaces dinámicas y modulares [29]. React resulta ideal para el desarrollo del “Front-end” de APIZZY debido a los siguientes puntos:

- **Es isomórfico:** con un mismo código, se puede renderizar HTML tanto en el servidor como en el cliente, disminuyendo así la carga de trabajo para realizar aplicaciones web amigables.
- **Desarrollo rentable:** esta librería ofrece a los desarrolladores una forma económica de generar aplicaciones para distintos sistemas (por ejemplo Android y Iphone) con el mismo código [29].
- **Naturaleza basada en componentes:** permite crear interfaces modulares y reutilizables, lo que se traduce en una mayor eficiencia durante el desarrollo y facilidad para realizar actualizaciones o cambios.
- **Interactividad en tiempo real:** React proporciona una experiencia dinámica para el usuario, reaccionando a sus acciones de manera inmediata y sin necesidad de recargar la página.
- **Alto rendimiento:** gracias al Virtual DOM de React, las actualizaciones en la interfaz se realizan de manera eficiente, optimizando el rendimiento general de la aplicación.
- **Comunidad activa:** la amplia comunidad de React significa que hay un constante flujo de nuevas herramientas, bibliotecas y soluciones a problemas comunes. Esto asegura que el framework siga siendo relevante y se mantenga al día con las últimas tendencias y tecnologías.
- **Flexibilidad y adaptabilidad:** React se puede integrar con diversas bibliotecas y herramientas, lo que lo hace adaptable a diferentes necesidades y retos de desarrollo.

3

Solución propuesta / método

En este apartado se presenta la información pertinente al proceso de desarrollo de la herramienta *APlizzy*. En una primera sección se describen las funcionalidades que el módulo debe implementar para cubrir las necesidades de los usuarios, para posteriormente definir formalmente los requisitos del sistema.

Se concluirá con una descripción detallada de la implementación final de la aplicación, en la que se explica la solución implementada con todos los módulos que la conforman, así como la interconexión entre cada uno de ellos.

3.1 *APlizzy*: especificación de funcionalidades esperadas

Como se ha adelantado en apartados anteriores, la gran necesidad de los desarrolladores de contar con una herramienta que simplifique el desarrollo de APIs y proporcione una interfaz sencilla y lúdica, tanto para usuarios avanzados como para aquellos usuarios con experiencias limitadas, resulta evidente. Por tanto este proyecto pretende simplificar la generación de código mediante la especificación de parámetros fijos por parte de los usuarios a través de un formulario simple y accesible.

Por ello, una parte de gran relevancia en este proyecto reside en el diseño de la interfaz con la que los usuarios finales interactúan, es decir, el *frontend*. Como ya se ha mencionado en el Capítulo 2 de esta memoria, este se va a desarrollar en React debido a los numerosos beneficios que este marco de trabajo ofrece.

Por otra parte, se requiere del diseño de un *backend* capaz de procesar la información generada por el formulario y responder de forma precisa, correcta y en un tiempo razonable. Además, teniendo en cuenta la ventaja con la que cuentan herramientas similares como Mocky, el despliegue del sistema se realizará mediante un servicio web que no requiera ningún tipo de instalación.

Cabe destacar que, con el fin de conectar estos dos módulos, se ha decidido emplear una API REST, siguiendo con la temática principal de este proyecto y demostrando una vez más su amplio uso en el sector del desarrollo de software hoy en día.

A continuación se procede a detallar en profundidad las funcionalidades esperadas de cada uno de los módulos.

3.1.1 Frontend

Mencionado anteriormente, el *frontend* será el encargado en la recopilación de datos por medio de un formulario. Este formulario tiene que ser personalizable para cumplir con las necesidades del usuario, y a partir de estos datos, el *frontend* deberá de guardar y generar un JSON con una estructura específica para ser tratable por el *backend*. Finalmente, se tendrá que recibir la respuesta del *backend* y proporcionar un archivo fuente Python descargable con la API generada.

Esto nos da a entender, que hay tres sub-módulos presentes, el primero recuperación de información, el segundo envío del JSON y el tercero el recibo y proporcionar descarga del API generada en un archivo fuente Python.

- **Módulo recuperación de información:** este módulo debe de recopilar la información a través de los campos del formulario, comprobar que los datos ingresados sean válidos y generar el Json con la estructura específica.
- **Módulo envío de Json:** este módulo debe de ser capaz de enviar el Json con los datos recopilados al *backend*.
- **Módulo descarga del archivo fuente Python:** este módulo debe de ser capaz de recibir la API generada por el *backend* y proporcionar un archivo funete Python con la API personalizada descargable por el usuario.

Dado que se utilizará React para el desarrollo del *frontend*, se implementará la utilización de la biblioteca Reactstrap. Esta biblioteca proporciona componentes

3.1 APIzzy: especificación de funcionalidades esperadas

estilizados y predefinidos, incluyendo una componente formulario llamada Form conformada por FormGroup, que facilitan la generación de formularios HTML, validaciones y personalización de campos.

Por otra parte, el formulario debe de ser lo mas dinámico posible y ser capaz de agregar distintos métodos o el mismo método, para una Api específica, a su vez de eliminar ciertos método generado. También, debe permitir la selección del formato de los estados. Para ello se implementaran en React distintas funciones de agregado y eliminación de componentes del formulario.

Para guardar la información correctamente en una estructura específica, el componente FormGroup debe guardar los datos, para ello se implementará el hook "useState", función que permitirá establecer los datos obtenidos del formulario en cualquier estructura deseada, en este caso un JSON que el *backend* pueda tratar.

Para el envío de información al *backend* una vez el formulario rellenado, se utilizará Axios. Axios es una biblioteca para realizar solicitudes HTTP, que permite el envío del JSON con un método POST y a su vez, permite recibir la respuesta del *backend* .

Una vez recibida la API generada por el *backend*, se implementará un objeto Blob, encargado de representar la API generada en JSON a formato Python, para luego proporcionar la nueva API como elemento de descarga.

3.1.2 Backend

Como se ha introducido anteriormente, el *backend* será el encargado de implementar la generación de código automática a partir de los datos que reciba del formulario implementado en el *frontend*. En este apartado se incluye también el desarrollo de la API REST encargada de la comunicación con la página web.

En primer lugar, las funcionalidades que debe cubrir el módulo de generación de código se recogen a continuación:

- **Soporte para las principales métodos HTTP:** para la primera versión de APIzzy se ha decidido dar soporte a los principales métodos HTTP, es decir, GET, POST, PUT y DELETE. Estas peticiones son las más empleadas en internet y sirven para obtener, crear, modificar y eliminar datos respectivamente.
- **Implementación para principales estructuras de datos:** las operaciones que realizan las API REST son numerosas, pero principalmente modifican estructuras y bases de datos. Por ello, en esta primera versión que se recoge en este documento, se pretende ofrecer la posibilidad de operar con

valores simples, objetos de tipo JSON y con operaciones sencillas en bases de datos.

- **Resistencia a errores:** el generador de código deberá contemplar valores por defecto para cada uno de los valores que espera recibir del *frontend* para poder ofrecer siempre una respuesta al usuario final, aunque consista en una plantilla simple que sirva como de base al desarrollo.
- **Código funcional y estandarizado:** el código que se proporcione al usuario debe ser completamente funcional, y como extra, se ha decidido que cumpla con la normativa PEP-8, una guía de estilo de codificación estandarizada empleada ampliamente por los desarrolladores Python.

La codificación de la API solicitada por el usuario se realizará empleando la librería Flask Restful de Python. Esta sigue un formato muy estandarizado que permite automatizar la generación de código sin emplear modelos de lenguaje avanzados como puede ser Generative Pre-trained Transformer 4 (GPT4). Se pretenden realizar plantillas con parámetros personalizables en las que se inyectarán los datos obtenidos del formulario.

Por otra parte, el programa a desarrollar debe ser lo más modular posible, teniendo en cuenta que en su primera versión APIzzy implementará cuatro de las peticiones HTTP, de tal manera que a futuro pueda ser escalable.

Por otra parte y como se ha adelantado anteriormente, también forma parte del *backend* la API que se encarga de traspasar los datos de un módulo a otro. Al recibir los datos de un formulario, es preciso que implemente un método que responda a una petición POST que contenga en el cuerpo una estructura en formato JSON fácilmente tratable por Python. Una vez recibida la petición, deberá procesar los datos y los volcarlos en el módulo de generación de código para cumplimentar la plantilla correspondiente.

3.2 Formato JSON del Formulario

Para que la generación del código funcione correctamente, es necesario establecer un formato de cerrado e inmutable al que se debe adecuar el JSON que se genera a partir del formulario. Para definir dicha estructura, deben definirse estrictamente cuáles son los parámetros que admiten personalización a la hora de crear la API.

Tras varias iteraciones siguiendo una metodología Ágil, los campos que se solicitan en el formulario son los siguientes:

3.2 Formato JSON del Formulario

- **Type:** tipo de petición que se desea implementar, admite los valores GET, POST, PUT y DELETE.
- **Source:** estructura de datos sobre la que se pretende operar y espera los valores *Raw*, *JSON* y *Sql* que significa tratar con valores en crudo, objetos JSON o bases de datos respectivamente.
- **MethodConfig:** contiene una estructura de configuración para cada uno de las posibles tipologías de datos sobre las que se opera explicadas en punto anterior.

En el caso de tratar con objetos JSON, deben cumplimentarse dos campos, uno obligatorio y otro opcional. El campo obligatorio es la ruta en la que se espera que se encuentre el JSON sobre el que se pretende operar, mientras que el opcional consiste en una llave, en caso de que solo se quiera operar sobre un dato único dentro de toda la estructura.

Si se opera sobre valores en crudo, simplemente debe rellenarse un campo con el nombre del valor que se quiere administrar.

La configuración más compleja es la que se debe transferir cuando se solicita una API que trabaje con una conexión a base de datos. En este caso, los campos requeridos son:

- **DbName:** nombre de la base de datos a la que se pretende acceder.
- **Username y Password:** credenciales del usuario que se desea emplear en la conexión a la base de datos.
- **Ip:** dirección IP en la que se aloja la base de datos.
- **Port:** puerto habilitado para realizar la conexión.
- **Columns:** columnas sobre las cuales se pretende operar.
- **Table:** nombre de la tabla que se desea seleccionar.
- **Values:** valores, en el caso de ser necesarios, que se pretenden insertar/modificar.
- **Conditions:** condiciones de filtrado en la base de datos.

Cabe mencionar que las credenciales se pasan en claro y no es la mejor práctica en términos de seguridad, por lo que es responsabilidad del usuario no comprometer las claves en el código final. Sin embargo esto no es una práctica exclusiva para conexiones a bases de datos, si no que es algo que todo programador debe conocer en un entorno de producción.

3.3 Definición formal de Requisitos y Casos de Uso

Antes de comenzar el desarrollo como tal de la propia aplicación, es fundamental realizar una definición clara de requisitos. Esto permite tener un seguimiento de las funcionalidades a implementar, los formatos que se deben seguir y ayudan a definir una guía clara a seguir. Además, combinando los requisitos con casos de uso, es posible estructurar un plan de pruebas completo para testar el sistema antes de ponerlo en producción. Por otra parte, a la hora de organizar el proceso de desarrollo y calcular tiempos, es vital conocer en profundidad los requisitos del sistema para llevar a cabo una correcta distribución de esfuerzos y recursos.

A lo largo de esta sección se enumeran los requisitos establecidos como base para el desarrollo de APIzzy, formalizando los contenidos explicados en secciones anteriores. Para seguir una definición estándar entre todos los requisitos se ha decidido emplear la siguiente plantilla con la información más comúnmente empleada en proyectos de desarrollo de software:

En este apartado se enumerarán los requisitos funcionales y no funcionales definidos para la realización de este proyecto a partir de la información explicada en apartados anteriores. A continuación se muestra la plantilla que se ha empleado para definir los requisitos, así como una breve descripción de los campos que la conforman:

ID: RXNN			
Nombre			
Prioridad		Verificabilidad	
Descripción			
Dependencias			

Tabla 3.1. Estructura empleada en la formalización de requisitos.

En la tabla 3.1 se pueden identificar los siguientes campos:

- **ID:** es una forma normalizada de identificar los requisitos mediante el format RXNN, donde X puede adoptar los valores F y NF dependiendo de si el requisito es funcional o no. Por otra parte NN es el número asignado a cada requisito.
- **Nombre:** se emplea como breve descripción del contenido del requisito.
- **Prioridad:** indica la prioridad ligada a cada uno de los requisitos y puede adoptar los valores alta, media o baja. Es relevante a la hora de organizar el proyecto.

3.3 Definición formal de Requisitos y Casos de Uso

- **Verificabilidad:** se emplea para medir cuánto de verificable es cada requisito, pudiendo ser alta, media o baja.
- **Descripción:** en este campo se realiza una descripción más extensa del requisito con el fin de aportar información extra.
- **Dependencias:** este último campo contiene los identificadores de otros requisitos que causen dependencias con el tratado, debiendo implementarse en orden para que el sistema funcione correctamente.

Requisitos Funcionales

Un requisito funcional es una declaración detallada y específica que describe una función o capacidad que debe tener un sistema, software o producto para cumplir con ciertas necesidades del usuario o del negocio. Estos requisitos definen cómo debe operar el sistema y qué acciones debe ser capaz de llevar a cabo. Son una parte esencial de la especificación de un proyecto y guían el desarrollo y diseño del producto. A continuación se muestran los requisitos funcionales identificados para este proyecto.

ID: RF01			
Nombre	Recogida de Datos		
Prioridad	Alta	Verificabilidad	Alta
Descripción	Se deberán recoger las preferencias de los usuarios mediante un formulario web con los datos recogidos en apartados anteriores		
Dependencias			

Tabla 3.2. Requisito funcional RF01.

ID: RF02			
Nombre	Petición HTTP POST		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema enviará la información extraída del formulario a través de una petición POST.		
Dependencias	RF01		

Tabla 3.3. Requisito funcional RF02.

3 Solución propuesta / método

ID: RF03			
Nombre	Generación acorde a la petición del usuario		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El usuario indicará qué métodos HTTP desea generar y el sistema deberá devolver el código de forma acorde a dichas peticiones.		
Dependencias	RF01, RF02		

Tabla 3.4. Requisito funcional RF03.

ID: RF04			
Nombre	Método GET		
Prioridad	Media	Verificabilidad	Alta
Descripción	El sistema será capaz de generar métodos GET para estructuras JSON, datos RAW y con conexiones a bases de datos.		
Dependencias	RF03		

Tabla 3.5. Requisito funcional RF04.

ID: RF05			
Nombre	Métodos POST, PUT y DELETE		
Prioridad	Media	Verificabilidad	Alta
Descripción	El sistema será capaz de generar métodos POST, PUT y DELETE para estructuras JSON y con conexiones a bases de datos, siendo posible aplicar filtros WHERE y operar sobre columnas individuales en este último caso.		
Dependencias	RF03		

Tabla 3.6. Requisito funcional RF05.

ID: RF06			
Nombre	El backend devolverá un fichero con extensión py		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema será capaz de generar un archivo formato Python con la API especificada.		
Dependencias	RF04, RF05		

Tabla 3.7. Requisito funcional RF06.

3.3 Definición formal de Requisitos y Casos de Uso

ID: RF07			
Nombre	Descargara automatica de la respuesta		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema descargará a través del navegador del usuario el fichero Python devuelto por el backend.		
Dependencias	RF06		

Tabla 3.8. Requisito funcional RF07.

Requisitos No Funcionales

Los requisitos no funcionales son criterios y características que describen cómo debe comportarse un sistema, software o producto en términos de atributos de calidad, en lugar de describir funciones específicas. Estos requisitos se centran en aspectos como el rendimiento, la seguridad, la usabilidad, la escalabilidad y otros atributos que no se relacionan directamente con las funciones principales del sistema, pero que son cruciales para garantizar su eficacia, confiabilidad y satisfacción del usuario. Los requisitos no funcionales ayudan a establecer estándares de desempeño y calidad que deben cumplirse durante el desarrollo y la implementación.

A continuación se enumeran los requisitos no funcionales identificados para el presente proyecto:

ID: RNF01			
Nombre	Versión de Python		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema se desarrollará en Python 3.10 o superior.		
Dependencias			

Tabla 3.9. Requisito no funcional RNF01.

ID: RNF02			
Nombre	Frontend React		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El frontend se desarrollará empleando el marco de trabajo React.		
Dependencias			

Tabla 3.10. Requisito no funcional RNF02.

3 Solución propuesta / método

ID: RNF03			
Nombre	API REST Flask		
Prioridad	Alta	Verificabilidad	Alta
Descripción	La API REST que conectará el frontend con el backend será desarrollada empleando el marco de trabajo de Flask.		
Dependencias			

Tabla 3.11. Requisito no funcional RNF03.

ID: RNF04			
Nombre	SQLAlchemy		
Prioridad	Alta	Verificabilidad	Alta
Descripción	Las conexiones a las bases de datos se gestionarán con la librería de Python SQLAlchemy.		
Dependencias			

Tabla 3.12. Requisito no funcional RNF04.

ID: RNF05			
Nombre	Resultados Python Flask en PEP-8		
Prioridad	Alta	Verificabilidad	Alta
Descripción	Las APIS generadas por APIzzy serán en Python, implementadas con Flask Restful y siguiendo el estándar de programación PEP-8.		
Dependencias			

Tabla 3.13. Requisito no funcional RNF05.

ID: RNF06			
Nombre	Modularidad y Escalabilidad		
Prioridad	Media	Verificabilidad	Baja
Descripción	El sistema deberá seguir una arquitectura modular y escalable con el fin de poder añadir nuevas funcionalidades a futuro.		
Dependencias			

Tabla 3.14. Requisito no funcional RNF06.

3.3 Definición formal de Requisitos y Casos de Uso

ID: RNF07			
Nombre	Datos por Defecto		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema deberá cumplimentar con datos por defecto las plantillas que contengan datos erróneos.		
Dependencias			

Tabla 3.15. Requisito no funcional RNF07.

ID: RNF08			
Nombre	API con tratamiento de error.		
Prioridad	Alta	Verificabilidad	Alta
Descripción	Las APIs generadas por el sistema tendrán en la plantilla en cuenta un tratamiento de errores básico.		
Dependencias			

Tabla 3.16. Requisito no funcional RNF08.

ID: RNF09			
Nombre	Inclusión de imports necesarios.		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema incluirá en los resultados la importación de librerías auxiliares necesarias para el correcto funcionamiento de la API.		
Dependencias			

Tabla 3.17. Requisito no funcional RNF09.

ID: RNF10			
Nombre	Validación de datos de entrada.		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El formulario tendrá verificación del formato de los datos de entrada.		
Dependencias			

Tabla 3.18. Requisito no funcional RNF10.

ID: RNF11			
Nombre	Múltiples recursos en la misma API.		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema admitirá la generación de APIS con varios recursos.		
Dependencias			

Tabla 3.19. Requisito no funcional RNF11.

ID: RNF12			
Nombre	Múltiples columnas y condiciones.		
Prioridad	Alta	Verificabilidad	Alta
Descripción	El sistema contemplará sentencias sql con múltiples columnas y condiciones.		
Dependencias			

Tabla 3.20. Requisito no funcional RNF12.

Casos de Uso

Esta sección presenta la descripción de los casos de uso del sistema. Estos casos de uso detallan las acciones que los usuarios o el propio sistema deben llevar a cabo para lograr un propósito específico.

Siguiendo el mismo enfoque que en la especificación de requisitos, utilizaremos una estructura en forma de tabla para detallar cada caso de uso. La estructura que usaremos es la siguiente:

CUNN	
Nombre	
Actor	
Propósito	
Condiciones iniciales	
Condiciones finales	

Tabla 3.21. Plantilla casos de uso.

3.3 Definición formal de Requisitos y Casos de Uso

CU01	
Nombre	Ingresar Datos en el Formulario.
Actor	Usuario.
Propósito	El Usuario accede a la interfaz e ingresa en el formulario los datos de los campos deseados para crear una API.
Condiciones iniciales	El usuario ha accedido a la interfaz principal de la aplicación.
Condiciones finales	El usuario ha accedido a la interfaz principal de la aplicación.

Tabla 3.22. Caso de uso CU01.

CU02	
Nombre	Enviar Datos del Formulario.
Actor	Front.
Propósito	Una vez que el usuario ha ingresado la información en el formulario, el frontend recoge estos datos y los envía al backend a través de una petición POST.
Condiciones iniciales	El usuario ha completado el formulario.
Condiciones finales	El backend recibe los datos y los procesa.

Tabla 3.23. Caso de uso CU02.

CU03	
Nombre	Seleccionar Métodos HTTP.
Actor	Usuario.
Propósito	El usuario, a través del formulario de la interfaz, selecciona qué métodos HTTP desea implementar en la API, como GET, POST, PUT o DELETE.
Condiciones iniciales	La interfaz presenta las opciones de métodos HTTP.
Condiciones finales	Las opciones seleccionadas quedan registradas para su posterior procesamiento.

Tabla 3.24. Caso de uso CU03.

3 Solución propuesta / método

CU04	
Nombre	Generar Métodos GET.
Actor	Back.
Propósito	Según las indicaciones dadas por el usuario, el backend procesa los datos y genera el código correspondiente a los métodos GET, considerando estructuras JSON, datos RAW o conexiones a bases de datos según sea necesario.
Condiciones iniciales	El usuario ha seleccionado la generación de métodos GET y ha especificado sus características.
Condiciones finales	El código para los métodos GET está listo para ser incluido en el archivo final de la API.

Tabla 3.25. Caso de uso CU04.

CU05	
Nombre	Generar Métodos POST, PUT y DELETE.
Actor	Back.
Propósito	El backend, con base en las elecciones del usuario, crea el código para los métodos POST, PUT y DELETE. Esto incluye la generación de conexiones a bases de datos, aplicación de filtros WHERE y operaciones sobre columnas individuales si es necesario.
Condiciones iniciales	El usuario ha elegido la generación de métodos POST, PUT y/o DELETE y ha definido sus especificaciones.
Condiciones finales	El código necesario para estos métodos está listo para ser integrado en el archivo .py de la API.

Tabla 3.26. Caso de uso CU05.

3.3 Definición formal de Requisitos y Casos de Uso

CU06	
Nombre	Crear Archivo Python de la API.
Actor	Back.
Propósito	Una vez que todos los métodos y características han sido definidos y generados, el backend compila toda esta información en un único archivo con formato Python, que contiene el código fuente completo de la API definida.
Condiciones iniciales	Todos los métodos y características de la API han sido definidos.
Condiciones finales	Se dispone de un archivo .py listo para ser descargado.

Tabla 3.27. Caso de uso CU06.

CU07	
Nombre	Descargar Archivo Python.
Actor	Front.
Propósito	El frontend recibe el archivo .py del backend y lo presenta al usuario como un archivo descargable, permitiéndole guardar la API generada en su dispositivo.
Condiciones iniciales	El archivo .py de la API ha sido generado.
Condiciones finales	El usuario tiene la opción de descargar el archivo en su dispositivo.

Tabla 3.28. Caso de uso CU07.

Matriz de trazabilidad

Para finalizar esta sección, se elaborará la matriz de trazabilidad que vincula los requisitos funcionales con los casos de uso. Esta matriz facilita la identificación visual de las conexiones entre los requisitos y los casos de uso, ofreciendo una dirección precisa para el desarrollo y la realización de pruebas del sistema.

REQ\UC	CU01	CU02	CU03	CU04	CU05	CU06	CU07
RF01	X						
RF02		X					
RF03			X				
RF04				X			
RF05					X		
RF06						X	
RF07							X

Tabla 3.29. Matriz de Trazabilidad

3.3.1 Solución Propuesta

Una vez definidos los 3.3 y casos de uso, en esta sección se presentará el diseño del frontend y el diseño del backend.

Diseño del Frontend

Teniendo en cuenta los requisitos definidos en el apartado 3.3, a continuación se discute la estructura diseñada para el *frontend* de APIzzy en un proyecto React.

La estructura del *frontend* de APIzzy sigue las de un proyecto React con buenas prácticas, se organiza en base a componentes modulares y reutilizables. Esto permite una mayor flexibilidad y facilidad de mantenimiento. La interfaz de APIzzy, tomando en cuenta estas buenas prácticas:

- **Componentización:** Se dividen las interfaces en componentes más pequeños y reutilizables. Cada componente tiene una función única y claramente definida. APIzzy tiene 14 componentes distintos archivos .js dentro de una carpeta *components* en la carpeta *src*. Estos componentes son desde la barra de navegador hasta el formulario en sí.
- **Jerarquía de carpetas:** Se suele separar los componentes en carpetas como *components* para componentes generales, *containers* para componentes de nivel superior que manejan la lógica de estado, y *assets* para imágenes y otros recursos estáticos.
- **Gestión del Estado:** Se utilizan herramientas como Redux o Context API para gestionar el estado global de la aplicación y garantizar que el flujo de datos sea unidireccional y predecible.

3.3 Definición formal de Requisitos y Casos de Uso

- **Rutas:** Se implementa un sistema de enrutamiento, generalmente con bibliotecas como *react-router*, para facilitar la navegación entre diferentes vistas o páginas de la aplicación.
- **Separación de Estilos:** Los estilos, ya sea usando CSS puro, preprocesadores como SASS o sistemas basados en JavaScript como Styled Components, se mantienen separados de la lógica para mantener una clara distinción entre presentación y funcionalidad.

Estructura del Proyecto

Como resultado de la explicación anterior, se ha determinado una estructura de tres carpetas principales para el proyecto. La primera carpeta llamada *node_modules*, es un directorio donde se almacenan las bibliotecas o paquetes que APIZZY utiliza, contiene paquetes de estilos y librerías de funcionalidades como Axios. La segunda carpeta llamada *public*, en el cual contiene activos públicos estáticos y el archivo principal *index.html* (archivo para poder ejecutar una aplicación React). La tercera carpeta llamada *src*, en esta carpeta se encuentran todos los componentes funcionales de la aplicación como se puede observar en la figura 3.1.

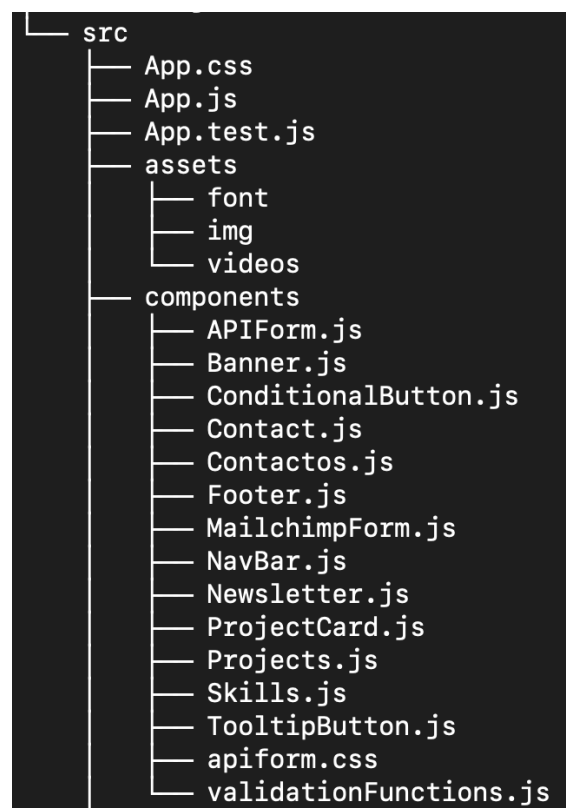


Figura 3.1. Carpeta *src* APIZZY.

Como se ha mencionado, en la carpeta *src* es donde se encuentran todas las

componentes del proyecto. Estas componentes son los distintos objetos y funcionalidades que componen la interfaz. Desde archivos `css` para un diseño visual amigable al usuario, videos e imágenes y por último los elementos que realizan la funcionalidad de la interfaz.

Creación de Formulario

Para la creación de la componente formulario, se ha creado un archivo llamado `APIForm.js`, este archivo tiene todas las funciones, constantes y código JSX el cual permite escribir estructuras similares al HTML dentro del código JavaScript. El formulario ha sido creado gracias Reactstrap con la componente `Form`.

Para estructurar campos de entrada y etiquetas dentro de un formulario, se utilizó `FormGroup`. Gracias a la utilización de estas componentes se pudo personalizar el formulario para cumplir con los requisitos. Estas componentes, permitieron realizar funciones de agregado y eliminado de recursos o de condiciones, estos consistían en que si la API del usuario contaba con dos métodos iguales, se realizaba en el `FormGroup` la llamada de una función *agregar* o *eliminar* elimina o agrega ese grupo de campos de entrada en el formulario como se observa en la figura 3.3.

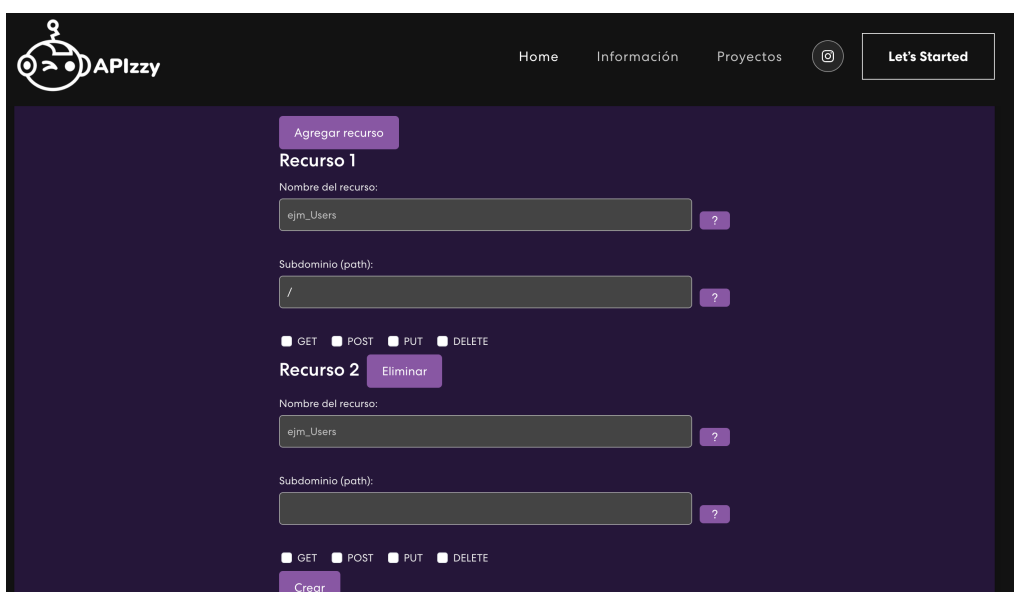
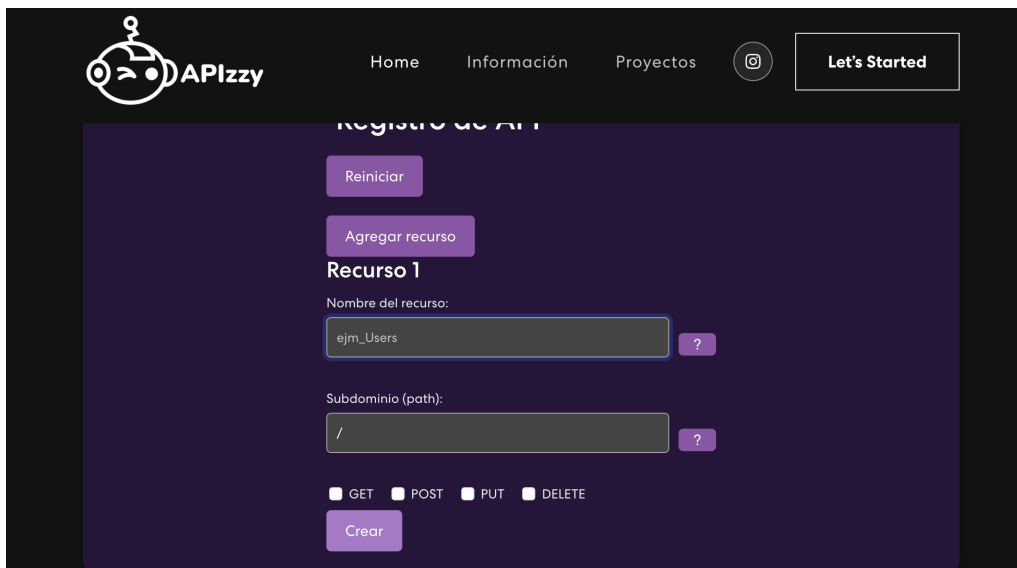
The image shows a web application interface for API management. At the top, there's a navigation bar with the 'APIzzzy' logo and links for 'Home', 'Información', 'Proyectos', and a 'Let's Started' button. The main content area is titled 'Agregar recurso' and contains two identical forms labeled 'Recurso 1' and 'Recurso 2'. Each form has a 'Nombre del recurso:' field with the value 'ejm_Users' and a 'Subdominio (path):' field with the value '/'. Below these fields are four radio buttons for HTTP methods: GET, POST, PUT, and DELETE. A purple 'Eliminar' button is next to the 'Recurso 2' form. At the bottom of the form area is a purple 'Crear' button.

Figura 3.2. Agregado o eliminado

También, dentro de estos `FormGroup` se indica el tipo de entrada deseado, (campo de tipo texto, de tipo *checkbox*, de selección entre varios recursos, entre otros). Con esta amplia selección, se pudo realizar un formulario dinámico y ameno al usuario. A continuación en las figuras 3.4 y 3.5, se mostrar de la interfaz mostrando los distintos tipos de entrada para recolectar la información del

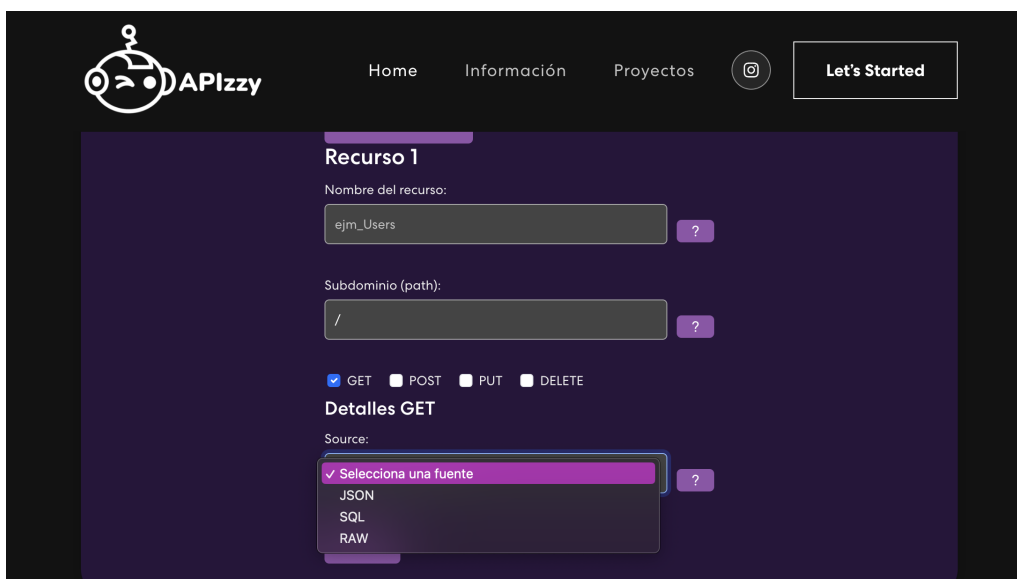
3.3 Definición formal de Requisitos y Casos de Uso

usuario.



The screenshot shows the 'Registro de API' form in the APIzzzy application. At the top, there's a navigation bar with 'Home', 'Información', 'Proyectos', and a 'Let's Started' button. The form has a 'Reiniciar' button at the top left. Below it is an 'Agregar recurso' button. The form is titled 'Recurso 1'. It contains two input fields: 'Nombre del recurso:' with the value 'ejm_Users' and 'Subdominio (path):' with the value '/'. Below these fields are four radio buttons for HTTP methods: GET, POST, PUT, and DELETE. A 'Crear' button is located at the bottom of the form.

Figura 3.3. Selección de método.



The screenshot shows the 'Recurso 1' form in the APIzzzy application. It includes the same fields as Figure 3.3: 'Nombre del recurso:' (ejm_Users) and 'Subdominio (path):' (/). Below these fields, the 'GET' radio button is selected. Underneath the radio buttons is a section titled 'Detalles GET' with a 'Source:' label. A dropdown menu is open, showing the following options: 'Selecciona una fuente' (highlighted), 'JSON', 'SQL', and 'RAW'.

Figura 3.4. Selección de formato.

Se han implementado para ciertos campos, distintas validaciones para el control de entradas. Estas validaciones fueron posible realizarlas gracias a los *props* del componente FormGroup, específicamente del *input*. Los *props* (abreviatura de *properties* o propiedades en español), en React, son una forma de pasar datos de componentes padres a componentes hijos. En este caso se utilizó la propiedad *pattern* la cual se utiliza para establecer una expresión regular contra la cual se valida el valor del campo de entrada. También se utilizó la propiedad *required*, para indicar que el campo de entrada es obligatorio y debe ser llenado antes de que

3 Solución propuesta / método

el formulario pueda ser enviado. A continuación en la figura 3.6 se mostrará una validación realizada, en la imagen 3.7 la notificación de una entrada no válida y en la figura 3.8 una entrada vacía.

```
<Input
  type="text"
  name="ip"
  value={resource.GetMethod.MethodConfig.Sql.Ip}
  onChange={e => methodChange(e, index, 'get', 'ip')}
  pattern="(((0-9){1,4}\.){3}[0-9]{1,4}"
  required
  className='mi-input-customizado'
  placeholder='ejemplo= 192.0.8.32'
  invalid={!new RegExp("(((0-9){1,4}\.){3}[0-9]{1,4}").test(resource.GetMethod.MethodConfig.Sql.Ip) && formSubmitted}
/>
```

Figura 3.5. Propiedades para validaciones.

The screenshot shows the APIzzzy web application interface. At the top, there is a navigation bar with the APIzzzy logo, links for Home, Información, and Proyectos, and a 'Let's Started' button. The main content area is titled 'Registro de API' and contains a 'Reiniciar' button. Below this is a section for 'Agregar recurso' with a sub-header 'Recurso 1'. The form for 'Recurso 1' has a label 'Nombre del recurso:' followed by two input fields. The first input field contains '1.1' and has a red question mark icon to its right. A red tooltip with an exclamation mark icon and the text 'Utiliza un formato que coincida con el solicitado' points to the first input field. The second input field contains '/' and also has a red question mark icon to its right. Below the input fields are radio buttons for GET, POST, PUT, and DELETE, and a 'Crear' button.

Figura 3.6. Formato no válido.

3.3 Definición formal de Requisitos y Casos de Uso

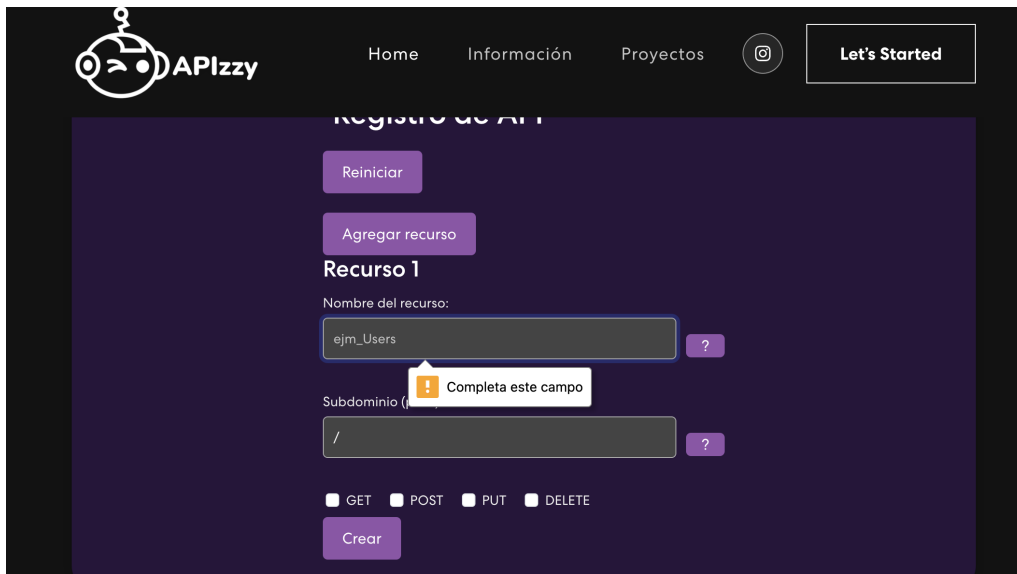


Figura 3.7. Entrada vacía.

Envío de Datos (POST)

Como se ha comentado anteriormente, los datos recuperados por el formulario tienen que cumplir con cierta estructura para ser tratables por el *backend*, para ello se ha optado por la creación de un objeto *initialState*. El objeto *initialState* define un estado inicial, que es común en aplicaciones React, especialmente cuando se utilizan bibliotecas de gestión de estado como Redux o el propio hook *useState* de React, dentro de este objeto se establecen estructuras y valores predeterminados. En general, *initialState* sirve como una plantilla o estructura predeterminada para el estado de una aplicación o componente React. Dada la lógica y la funcionalidad que se ha implementado en la aplicación, este estado inicial se actualiza y cambia a medida que el usuario interactúa con el formulario y a medida que se procesan otros eventos.

Este objeto se utiliza en una función asíncrona llamada *enviarAlaBD* el cual envía el objeto, ejecutando una petición POST a un servidor local en el puerto 5001, enviando los datos que se han recopilado del formulario. Esta petición se logra con la implementación de la biblioteca Axios anteriormente comentada.

Descarga de API

Se implementó una funcionalidad en la que, tras recibir datos del servidor, se crea un Blob, objeto que representa datos en formato binario o texto. Posteriormente, se añade un enlace al documento para permitir la descarga de este Blob. Al simular un clic en dicho enlace, se inicia la descarga del archivo denominado

3 Solución propuesta / método

APlizzy.py. Una vez finalizada la descarga, el enlace se elimina automáticamente del documento.

3.3 Definición formal de Requisitos y Casos de Uso

Diseño del Backend

Teniendo en cuenta los requisitos definidos en el apartado 3.3, a continuación se discute la arquitectura diseñada para el *backend* de APIzzy. Debe destacarse que la modularidad y escalabilidad del sistema han sido prioridad a la hora de organizar y encapsular cada una de las funcionalidades, teniendo en cuenta que a futuro puedan añadirse nuevos módulos para contemplar las peticiones HTTP obviadas en esta primera versión.

La figura 3.8 representa el diagrama de clases planteada para el generador de código automático de APIzzy siguiendo los estándares UML (Unified Modelling Language).

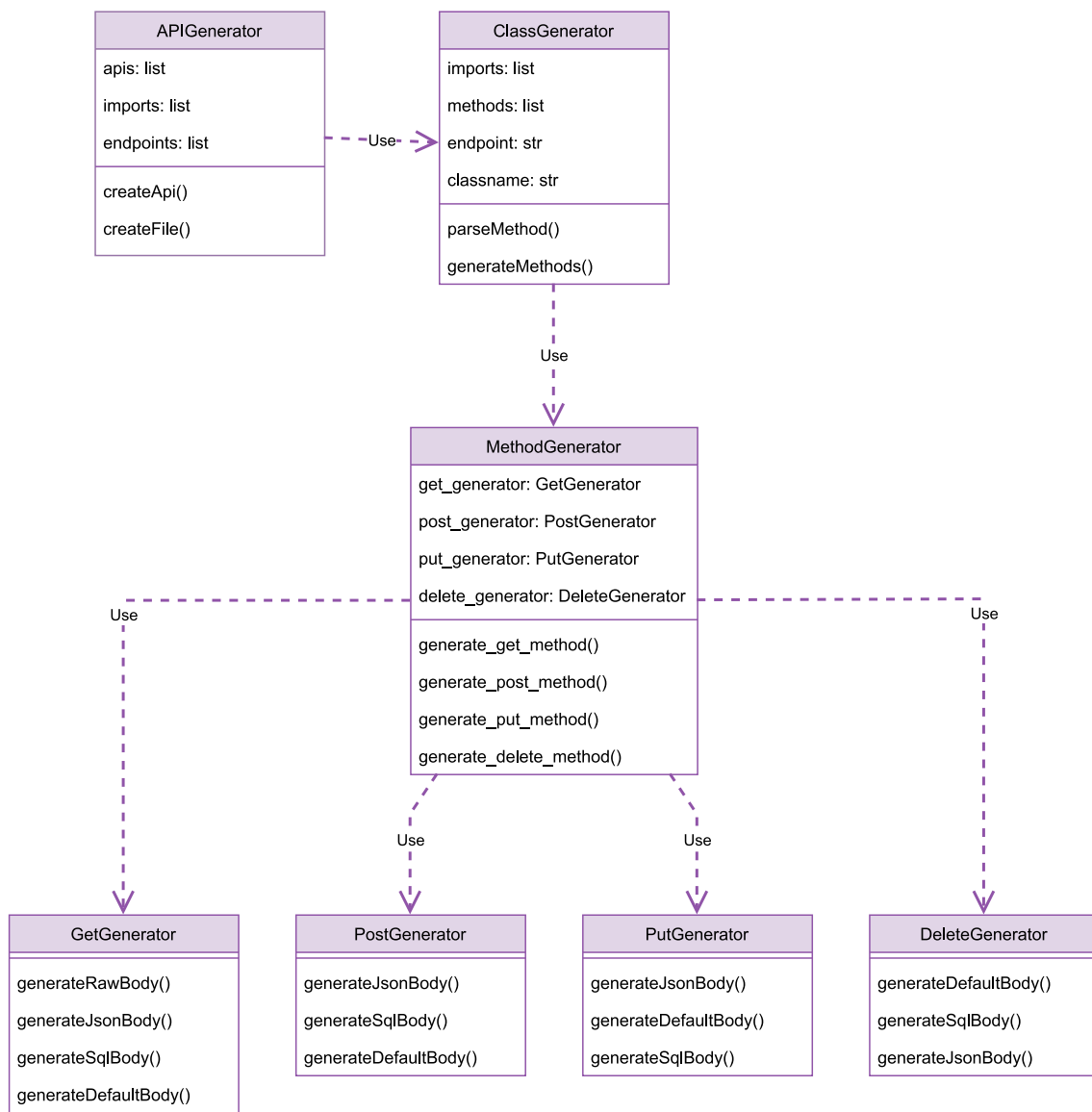


Figura 3.8. Diagrama de clases backend APIzzy.

Como se puede observar, el módulo principal que se encarga de generar el

código es APIGenerator empleando las funcionalidades del resto de clases que conforman el sistema. Primeramente, se puede observar que utiliza directamente el módulo ClassGenerator, el cuál a su vez usa a MethodGenerator, el cuál instancia una clase de cada uno de los generadores específicos para cada método HTTP.

Antes de evaluar en detalle cada uno de los módulos, cabe realizar ciertas puntualizaciones sobre las el formato en el que se produce el código. Como se ha adelantado, Flask Restful permite que para este proyecto se empleen plantillas debido a la estructura similar entre todas las APIs producidas con este framework. Cada API tiene se instancia como una clase de Python, y cada uno de los métodos HTTP se reflejan en métodos asociados a esa clase. Finalmente, se elije el punto de montaje de cada endpoint en la url a la que se pretenden realizar peticiones y se asocia a cada API.

A continuación se detallan las funciones específicas de cada uno de los módulos.

APIGenerator

En primer lugar, esta clase contiene tres atributos. Puesto que el sistema permite generar varias APIs con una petición del formulario, en primer lugar tiene una lista con cada una de las clases a generas. Además hay un atributo para guardar cada uno de los puntos de montaje relacionado con cada API y otro con los imports que deben generarse en la cabecera en el caso de necesitarse librerías adicionales al trabajar con estructuras de datos especiales como JSON.

En cuanto a las funcionalidades que incluye, posee un método que se ocupa de delegar el trabajo de generación de la API a ClassGenerator, guardar las plantillas cumplimentadas de vuelta, así como los imports necesarios y los endpoints pertinentes. El segundo método se encarga de, a partir de las plantillas obtenidas con el método anterior, formar el fichero final uniendo las diferentes APIs, librerías importadas y endpoints.

ClassGenerator

Se trata de la clase que emplea directamente APIGenerator. Posee como atributos, el nombre de la clase o API a implementar, el punto de montaje correspondiente, los métodos HTTP que deben implementarse y las librerías que deben importarse en caso de ser necesarias.

La función principal de este módulo es la de filtrar los métodos HTTP que deben generarse y llamar a las funciones necesarias en las clases subyacentes

3.3 Definición formal de Requisitos y Casos de Uso

para producir dichos métodos. Una vez creada la API con todos los métodos solicitados y el punto de montaje, se devuelven dichos valores a APIGenerator para formar el fichero que se envía al usuario final.

MethodGenerator

MethodGenerator posee una instancia de cada uno de los generadores de métodos específicos que se explicarán en apartados posteriores. A partir de la información filtrada que recibe por parte de ClassGenerator, procede a hacer un segundo filtrado en base a la estructura de datos que se desea emplear en el cuerpo del método y emplea el generador de código pertinente. De nuevo, simplemente aplica un nuevo filtro a los datos obtenidos de la petición del usuario y delega la generación del cuerpo principal de cada uno de los métodos a los generadores individuales apenas mencionados.

GetGenerator, PostGenerator, PutGenerator, DeleteGenerator

Se trata de los generadores de código encargados de los métodos HTTP de tipo GET, POST, PUT y DELETE. Contiene una función para cada tipo de estructuras de datos especificadas que dicho método debe soportar según los requisitos discutidos anteriormente en el apartado 3.3. En estas funciones se forma el código a partir de las plantillas, se establecen valores predeterminados para evitar valores erróneos y se forma con la indentación correcta un script completamente funcional, con un tratamiento de errores sencillo.

En el caso de quererse incluir nuevos generadores para los métodos HTTP no contemplados por APIzzy, sería suficiente con añadir una nueva clase para dicho método e incluir una regla en el filtrado de la clase MethodGenerator, dotando al sistema de una gran flexibilidad a la hora de incorporarse nuevas funcionalidades.

API de comunicación backend-frontend

Para concluir el apartado relativo a la metodología empleada en el desarrollo del sistema APIzzy, es necesario tratar las comunicaciones entre el formulario empleado de cara al usuario y el generador automático de código. Anteriormente se ha adelantado que esta comunicación se realiza mediante una API REST, la cual solamente tiene responde al método POST en el punto de montaje `/form`.

La funcionalidad de esta API es la de recibir los datos en el formato JSON especificado en la sección 3.2 en el contenido de la petición. Una vez recibidos los datos, procede a instanciar una clase de APIGenerator a la que pasa como

3 Solución propuesta / método

configuración el JSON en cuestión. Esta instancia se encarga de generar el fichero, y una vez se ha generado, la API se encarga de formatear el código según el estándar PEP-8 empleando la herramienta para línea de comandos "black", un formateador de código abierto.

Finalmente, cuando el fichero final está completamente listo y funcional, se envía de vuelta al servicio web desplegado, donde se descarga automáticamente a través del navegador del cliente. En la imagen 3.9 se muestra de forma esquemática el funcionamiento de interconexión simple entre los módulos.

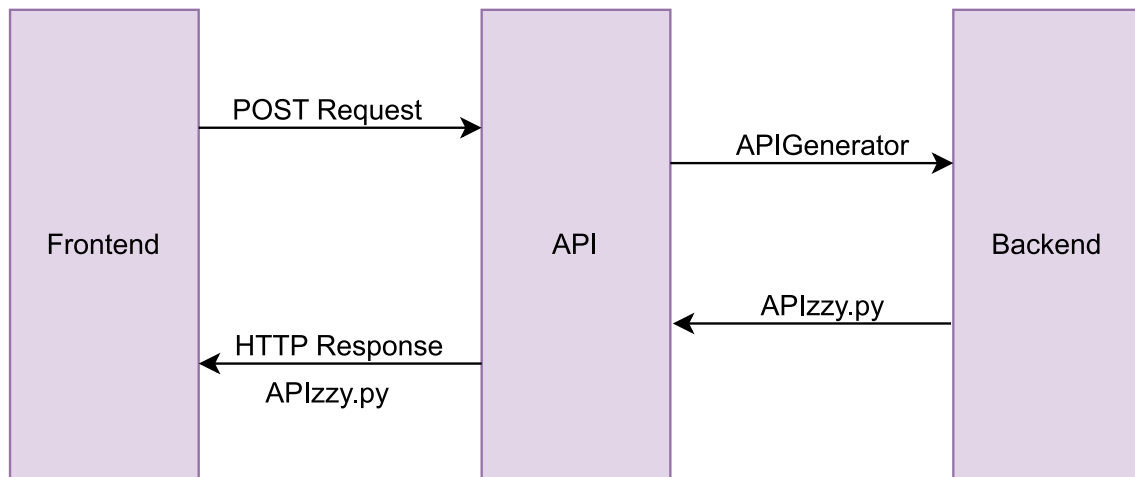


Figura 3.9. Interconexión simple de los módulos de APIzzy.

4

Validación y pruebas

En esta sección se describen las pruebas estratégicas llevadas a cabo para garantizar el correcto funcionamiento de APIzzy. Se abordarán las pruebas unitarias, que confirman la operatividad individual de los módulos; las pruebas de integración, que aseguran la cohesión entre estos módulos; y las pruebas de aceptación, que validan que la solución global cumple con las expectativas del usuario y los requisitos técnicos previstos.

4.1 Validaciones Frontend:

El frontend de una aplicación juega un papel crucial, sirviendo como el punto de interacción directa entre el usuario y el sistema. Su eficiencia, rendimiento y experiencia del usuario determinan en gran medida la percepción y satisfacción del usuario final. Validar adecuadamente el frontend no solo garantiza que los componentes individuales funcionen como se espera, sino que también asegura que los mismos interactúen armónicamente entre sí y con el backend, ofreciendo una experiencia cohesiva y libre de errores. En este contexto, las pruebas unitarias desempeñan un papel fundamental al validar las menores unidades de funcionalidad en el frontend.

4.1.1 Pruebas unitarios

Las pruebas unitarias en el frontend se centran en evaluar y verificar la correcta funcionalidad de los componentes más pequeños y aislados de la aplicación, asegurando que actúen y respondan como se espera. Siguiendo la metodología ágil adoptada a lo largo de todo el desarrollo del proyecto, se han realizado pruebas de componentes individuales del frontend de APIZZY.

■ Componentes Individuales:

- Validar que cada componente (por ejemplo, botones, campos de entrada, listas desplegables) se renderice correctamente en diferentes escenarios.
- Asegurar que los componentes muestren los datos adecuados y manejen correctamente los estados (por ejemplo, un botón debe estar desactivado cuando ciertas condiciones no se cumplen).
- Validar que cada componente (por ejemplo, botones, campos de entrada, listas desplegables) se renderice correctamente en diferentes escenarios.
- Asegurar que los componentes muestren los datos adecuados y manejen correctamente los estados (por ejemplo, un botón debe estar desactivado cuando ciertas condiciones no se cumplen).

■ Manejadores de Eventos:

- Comprobar que los eventos, como los clics del ratón o las entradas del teclado, desencadenen las acciones esperadas. Por ejemplo, al hacer clic en un botón 'Enviar', debe activarse la función correspondiente y los datos del formulario deben ser capturados.
- Asegurar que los eventos de cambio ('onChange') en los campos de entrada actualicen el estado de la aplicación como se espera.

■ Formularios:

- Verificar que los formularios validen la entrada del usuario según las reglas definidas (por ejemplo, comprobar el formato de una IP).
- Asegurar que los mensajes de error se muestren adecuadamente cuando la entrada del usuario no cumpla con los criterios establecidos.

■ Integración con el Estado:

- Comprobar que los cambios en el estado de la aplicación se reflejen adecuadamente en la UI. Por ejemplo, si un usuario selecciona 'SQL'

como fuente en tu aplicación APIZZY, los campos relacionados con SQL deben mostrarse adecuadamente.

- Asegurarse de que las acciones que modifiquen el estado (como seleccionar una opción en una barra de navegación) actualicen el estado correctamente.

Se llevaron a cabo 65 pruebas manuales para componentes, botones, campos, validaciones de entrada, como se ha descrito anteriormente. El porcentaje de cobertura del código alcanzado ha sido del 89

Cabe destacar que, gracias a la metodología ágil adoptada, se llevaron a cabo pruebas constantes a lo largo de todo el desarrollo del proyecto, asegurando así el correcto funcionamiento y rendimiento de la página web.

4.1.2 Pruebas de integración

Estas pruebas buscan garantizar que los componentes individuales del frontend de APIZZY no solo funcionen de manera aislada, sino que también colaboren correctamente cuando se integran en el sistema completo. En concreto se han realizado las siguientes pruebas con resultados satisfactorios:

- Validar que al seleccionar un método (GET, POST, PUT, DELETE) en la interfaz, no solo se muestren los campos adecuados, sino que también se inicialicen y procesen correctamente.
- Confirmar que los datos introducidos en un componente se transmiten, reciben y procesan adecuadamente por otros componentes, manteniendo la integridad de la información.
- Asegurarse de que las transiciones y flujos entre diferentes vistas o componentes de la aplicación sean coherentes y se ejecuten sin interrupciones o errores inesperados.
- Verificar que las dependencias entre componentes, como la llamada de funciones o el acceso a variables compartidas, se gestionen de manera óptima, evitando conflictos o fallos.

4.2 Validaciones Backend

El backend de una aplicación, en especial en herramientas como APIZZY, constituye la columna vertebral que sostiene y procesa todas las operaciones

fundamentales que realiza el sistema. Sirve como puente entre el frontend y la transformación de datos, garantizando que la información sea procesada, almacenada y recuperada de manera eficiente y segura. El rendimiento, la seguridad y la confiabilidad del backend son esenciales para asegurar que la aplicación funcione sin interrupciones y cumpla con las expectativas del usuario. Validar el backend adecuadamente no solo es crucial para asegurar la correcta gestión y procesamiento de los datos, sino también para garantizar una comunicación fluida y eficiente con el frontend, evitando potenciales fallos o vulnerabilidades. En el marco de APIZZY, las pruebas unitarias se convierten en esenciales, ya que permiten validar cada función y método de la API, asegurando que cada endpoint responda y actúe como se espera ante diversas solicitudes y escenarios.

4.2.1 Pruebas unitarios

Siguiendo la metodología ágil adoptada a lo largo de todo el desarrollo del proyecto, con el fin de realizar un código funcional y testado de forma automática y continua, se ha aplicado un proceso de pruebas bajo el paradigma Test Driven Development (TDD). Este paradigma se basa en realizar las pruebas antes que el propio código, de tal manera que las funciones desarrolladas se diseñen estrictamente para cumplir los objetivos definidos en el test.

Las pruebas se han diseñado para probar el caso y medio y los casos extremos de cada función, de tal manera que se compruebe la resistencia del código ante entradas erróneas. La métrica empleada para garantizar un proceso de pruebas completo ha sido el porcentaje de cobertura del código, es decir, el porcentaje de líneas de código, ramas de control y condiciones que se han ejecutado durante las pruebas.

Los módulos que se han probado son los presentados en la figura 3.8:

- **APIGenerator**
- **ClassGenerator**
- **MethodGenerator**
- **GetGenerator**
- **PostGenerator**
- **PutGenerator**
- **DeleteGenerator**

Para las 21 funciones en total que contienen estas clases, se han diseñado un total de 83 pruebas en las que el porcentaje de cobertura del código alcanzado ha sido del 94 %.

4.2.2 Pruebas de integración

Las pruebas de integración de software son un tipo de prueba que se centra en verificar la interacción y la colaboración adecuada entre los diferentes componentes o módulos de un sistema de software. Su objetivo principal es asegurarse de que las diversas partes del software funcionen correctamente cuando se integran y se ejecutan juntas como un sistema completo.

En concreto se han realizado pruebas con resultado satisfactorio siguiendo las conexiones del gráfico UML presentado en la figura 3.8:

- **APIGenerator** con ClassGenerator
- **ClassGenerator** con MethodGenerator
- **MethodGenerator** con GetGenerator, PostGenerator, PutGenerator y DeleteGenerator

A continuación se mostrara una API creada con APIzzy.

La Api generada tiene los métodos GET y POST, el GET se realiza a partir de una base de datos SQL y el post desde un JSON. Después de completar los campos obtenemos un archivo python con:

Código Python APIzzy.py

```
1 import json
  import psycopg2
3 from flask_restful import Resource, Api
  from flask import Flask, jsonify, make_response
5
  app = Flask(__name__)
7 api = Api(app)

9 class EjemploGet(Resource):
    def get(self):
11         try:
                connection = psycopg2.connect(
13                     database="nombreSql_get",
                        user="userSql_get",
```

```

15         password="claveSql_get",
16         host="192.0.9.3",
17         port="500",
18     )
19     cursor = connection.cursor()
20     cursor.execute("Select Columna from Tabla;")
21     ret = cursor.fetchall()
22     connection.close()
23
24     return make_response(jsonify({"Content": ret}), 200)
25
26 except Exception as exc:
27     return make_response(jsonify({"Error": str(exc)}), 400)
28
29 class EjemploPost(Resource):
30     def post(self):
31         http_data = request.get_json()
32
33         try:
34             with open("json_post", "r") as jfile:
35                 jret = json.load(jfile)
36
37                 if isinstance(jret, dict):
38                     jret = [jret]
39                 jret.append(http_data)
40
41             with open("json_post", "w") as jfile:
42                 json.dump(jret, jfile, indent=4)
43
44             return make_response(jsonify(jret), 200)
45
46         except Exception as exc:
47             return make_response(jsonify({"Error": str(exc)}), 400)
48
49 api.add_resource(EjemploGet, "/dominio_get")
50 api.add_resource(EjemploPost, "/dominio_post")
51
52 if __name__ == "__main__":
53     app.run(debug=True)

```

5

Gestión del proyecto

La gestión de un proyecto tecnológico, como es el desarrollo de APlzzy, no sólo se centra en la programación y la implementación de funcionalidades. Es un proceso complejo que requiere una meticulosa planificación, una ejecución estratégica y una visión clara del impacto que tendrá el producto en el mercado y la sociedad. La creación de APlzzy representa una amalgama de habilidades técnicas y administrativas, fusionadas con el objetivo de entregar una herramienta robusta y vanguardista que responda a las necesidades actuales de la industria del software. A lo largo de este capítulo, exploraremos en profundidad cómo se abordó la gestión de APlzzy, desde su concepción hasta su materialización, poniendo de manifiesto las decisiones, desafíos y soluciones que marcaron el camino hacia su culminación exitosa.

5.1 APlzzy: especificación de funcionalidades esperadas

5.1.1 Impacto Socio-Económico

El impacto socio-económico de una herramienta como APlzzy es considerable, especialmente en un mundo donde la digitalización está avanzando a pasos agigantados. Desde una perspectiva social, APlzzy se erige como una solución innovadora que responde a las crecientes demandas del sector tecnológico. La

simplificación en la definición y creación de APIs abre puertas para que desarrolladores, tanto novatos como experimentados, puedan participar activamente en el desarrollo de aplicaciones y servicios sin verse obstaculizados por complejidades técnicas. Este empoderamiento de los desarrolladores puede traducirse en un aumento de la diversidad en el mundo de la tecnología, permitiendo que personas de diferentes trasfondos y niveles de experiencia aporten sus ideas y soluciones al mercado. Además, en una sociedad donde la eficiencia y la inmediatez son altamente valoradas, la posibilidad de acelerar el desarrollo de soluciones tecnológicas puede tener repercusiones significativas en la satisfacción y calidad de vida de los usuarios finales.

Económicamente, APIzzy tiene el potencial de ser un catalizador de crecimiento para empresas y startups. En primer lugar, al reducir el tiempo y el esfuerzo necesarios para desarrollar y lanzar aplicaciones y servicios al mercado, las empresas pueden obtener un retorno de inversión más rápido, lo que es especialmente crucial para startups con recursos limitados. Además, al disminuir la necesidad de contratar especialistas en APIs, las empresas pueden redistribuir esos fondos hacia áreas como marketing, investigación y desarrollo, o incluso capacitación y educación para sus empleados. Por otro lado, esta herramienta puede fomentar un ecosistema más competitivo, donde las pequeñas empresas tienen una mejor oportunidad de competir con grandes corporaciones gracias a la reducción de barreras técnicas. Asimismo, este ambiente competitivo puede conducir a la innovación, ya que las empresas se esforzarán por diferenciarse y ofrecer soluciones únicas al mercado.

En una visión más amplia, APIzzy puede contribuir indirectamente al crecimiento económico de una región o país. Con la digitalización como una de las principales fuerzas motrices de la economía moderna, tener herramientas que faciliten y agilicen el desarrollo tecnológico puede atraer a inversores y talentos a una región. Además, en un mundo donde la economía gira en torno a la información, ser capaz de acceder y gestionar datos a través de APIs eficientes y bien estructuradas es esencial.

Para concluir, APIzzy no es simplemente una herramienta de desarrollo; es un facilitador en el mundo de la tecnología, con el potencial de desencadenar cambios significativos tanto en el panorama social como económico. En una era donde la adaptabilidad y la rapidez son esenciales, soluciones como APIzzy pueden ser el puente hacia un futuro más inclusivo, innovador y próspero.

5.1.2 Metodología Utilizada

El desarrollo de APIzzy implicó enfrentar retos complejos y desafíos dinámicos, lo que exigía una metodología que pudiera adaptarse a los cambios de ma-

5.1 APlzzy: especificación de funcionalidades esperadas

nera efectiva y eficiente. Por esta razón, se adoptó una metodología ágil enfocándose en su carácter iterativo.

- **Análisis del proyecto:** En esta etapa, se profundiza en la idea central de APlzzy, explorando su concepto y finalidad. Se realiza un estudio preliminar sobre soluciones similares existentes y se postula una propuesta preliminar, delineando su factibilidad a nivel teórico, sin entrar en fase de implementación.
- **Investigación del estado del arte:** Una vez consolidada la pertinencia de APlzzy, se efectúa una investigación exhaustiva sobre soluciones preexistentes. Se identifican limitaciones de dichas soluciones y se investigan tecnologías recientes relacionadas con los conceptos que serán aplicados en APlzzy.
- **Definición de requisitos y casos de uso:** Con una perspectiva más clara de lo que será APlzzy, se procede a formalizar los requisitos y casos de uso. Dado el enfoque de APlzzy, se considera la perspectiva y las necesidades de los usuarios, estableciendo requisitos desde un enfoque de utilidad y usabilidad.
- **Diseño y análisis de la solución:** A partir de los requisitos previamente establecidos, se diseñan varias propuestas de solución para APlzzy. Se consideran diferentes enfoques y arquitecturas, y luego se analizan en detalle, seleccionando finalmente la propuesta más adecuada para seguir adelante.
- **Implementación, Iteración 1 - Frontend:** Durante esta fase se implementan las necesidades básicas del frontend de APlzzy. Aunque no se aborda completamente la experiencia del usuario, se garantiza la escalabilidad y adaptabilidad del código.
- **Implementación, Iteración 2 - Backend:** En esta iteración, se centra en el backend de APlzzy, implementando funcionalidades y requisitos esenciales para el buen funcionamiento del sistema.
- **Implementación, Iteración 3 - Integración:** Esta es la fase más compleja. Aquí, se integran las funcionalidades del frontend y backend, asegurando una comunicación fluida entre ambas partes. Se presta especial atención a la experiencia del usuario y se trabajan características opcionales.
- **Validaciones y pruebas:** En paralelo a las iteraciones de implementación, se lleva a cabo un riguroso proceso de validación y pruebas, empleando metodologías específicas para garantizar la calidad y eficacia de APlzzy.
- **Documentación:** A lo largo de todo el proceso de desarrollo, se elabora la documentación pertinente.

5.1.3 Planificación Temporal y Diagrama de Gantt:

La ejecución efectiva de APIzzy se extendió a lo largo de 6 meses, abarcando desde la concepción inicial hasta el lanzamiento en producción. Esta distribución temporal fue cuidadosamente planificada para maximizar la eficiencia y asegurar que cada fase tuviera suficiente tiempo para su desarrollo completo. A continuación se muestra el diagrama de GANTT, ilustrando cada una de las fases, sus tareas asociadas, y cómo estas se solapaban y dependen entre sí a lo largo del tiempo.

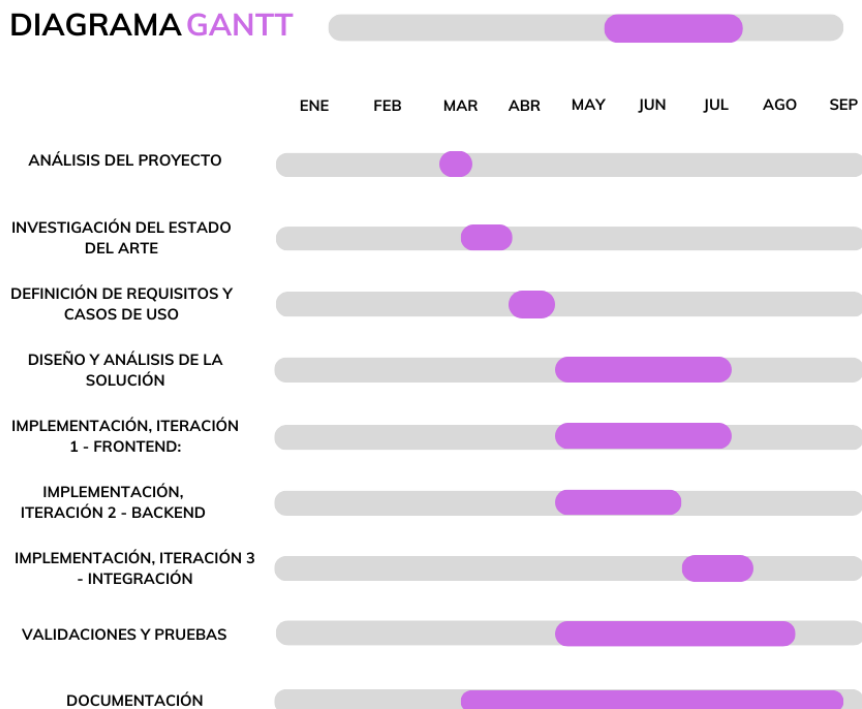


Figura 5.1. Distribución de tareas y tiempos del proyecto (diagrama de Gantt).

El desarrollo de APIzzy se llevó a cabo siguiendo la metodología SCRUM, una de las prácticas ágiles más populares, que permite una planificación y revisión iterativa a través de sprints, en este caso, semanales. Para gestionar el trabajo de manera eficiente, cada tarea fue estimada según su complejidad y tiempo de desarrollo, un ejemplo es el desarrollo del frontend y su constante edición para realizar una interfaz amigable al usuario.

5.1 APlizzy: especificación de funcionalidades esperadas

5.1.4 Presupuesto:

En esta sección, se detallarán los recursos que han sido necesarios para el desarrollo del proyecto APlizzy, así como el coste de cada uno de ellos. Diferenciaremos entre los siguientes costes: coste de personal, coste de Software/Hardware y costes indirectos. Al final de este apartado, se hará un resumen de los costes totales.

Costes de Personal

Los costes de personal incluyen horas de clases, tutorías o el trabajo individual por parte del alumno. Las horas totales que ha llevado desarrollar este proyecto han sido de 375 horas, que se corresponden con 12 créditos ECTS. Cada crédito se corresponde con 31,25 horas de trabajo.

Cargo	Horas	Coste por hora	Coste Total
Ingeniero	375	20€/hora	7.500€
Director del Proyecto	20	50€/hora	1.000€
Total			8.500€

Tabla 5.1. Tabla costes de personal.

Costes Software y Hardware

Los costes asociados al software engloban los gastos derivados de los programas utilizados durante el desarrollo del proyecto, así como el hardware indispensable para su correcto funcionamiento.

Producto	Coste	Amortización	Coste dentro del proyecto	Coste Total
MacBook Pro M2	1.600€	7 meses	228,6€	1.371,6€
Pc de mesa	2.000€	48 meses	41,6€	250€
React	0	-	0	0
VisualStudio Code	0	-	0	0
Python	0	-	0	0
Linux	0	-	0	0
Total	-	-	-	1.621,6€

Tabla 5.2. Tabla costes de software y hardware.

Costes Indirectos

Los costes indirectos incluyen los costes relacionados con los recursos básicos como la electricidad y el servicio de internet.

Producto	Precio por Mes	Uso en Proyecto	Coste Total
Electricidad	50	6 meses	300€
Servicio de Internet	30	6 meses	180€
Total			480€

Tabla 5.3. Tabla costes indirectos.

Juntando todo lo anterior obtenemos:

Producto	Gasto Total
Personal	8.500€
Material	1.621,6€
Costes indirectos	480€
Total	10.601,6€

Tabla 5.4. Tabla costes totales.

6

Conclusiones

En esta sección, se despliegan las conclusiones resultantes del desarrollo del proyecto presentado en este informe, junto con la explicación detallada de la metodología aplicada a lo largo de su ejecución. Además, se establece una estrecha relación entre los conocimientos adquiridos a lo largo de los años de carrera cursados y su aplicación práctica en la implementación del sistema objeto de estudio. En este sentido, se destaca la importancia de la unión entre la teoría académica y la experiencia real, demostrando cómo los conceptos y habilidades adquiridos durante la formación académica se traducen en soluciones concretas y aplicables en el ámbito profesional.

En primer lugar se procedió a un análisis en profundidad del estado del arte relacionado con el campo de estudio con el fin de sentar las bases del proyecto y proponer una solución novedosa y diferente a las ya existentes en el mercado. De la misma manera, fue posible seleccionar las herramientas que mejor se adaptan a un desarrollo como el que se recoge en esta memoria. Una vez finalizado este análisis se definieron los requisitos que tenía que cumplir la aplicación, teniendo en cuenta las posibles mejoras integrables a futuro, así como la simplicidad y interactividad que se pretendía alcanzar con este proyecto.

A lo largo del desarrollo, se fueron encontrando inconsistencias en la arquitectura del backend, las cuales se fueron refinando en diferentes interacciones derivadas de la aplicación de una metodología ágil, hasta llegar al modelo presentado en este informe. De la misma manera, hasta obtener los requisitos finales fue necesario ir adaptando el código y los formatos de las estructuras de datos que intercambian el formulario y el generador automático de código. El paso final

consistió en interconectar ambos módulos mediante la API REST descrita en el apartado sobre la metodología.

Resumiendo y reflexionando sobre todo el proceso, resulta evidente que, a pesar de los desafíos y obstáculos encontrados a lo largo del camino, se ha logrado implementar una metodología sólida y meticulosa. Esta metodología ha demostrado ser esencial para llevar a cabo con éxito el desarrollo integral del sistema. Desde las etapas iniciales de exploración del estado del arte hasta la definición exhaustiva de los requisitos, pasando por la concepción y diseño de la arquitectura, el proceso de desarrollo del módulo en sí mismo, su rigurosa validación y, por último, la minuciosa elaboración de una documentación completa y detallada.

Este enfoque integral no solo ha permitido la consecución exitosa del proyecto, sino que también ha brindado al estudiante la valiosa oportunidad de aplicar y poner en práctica una amplia gama de conocimientos adquiridos a lo largo de su carrera académica. Es en este punto donde la teoría se transforma en acción concreta y tangible, lo que enriquece significativamente la experiencia educativa y la preparación del alumno para desafíos futuros en el ámbito profesional.

A lo largo del ciclo completo de diseño del sistema se han empleado técnicas aprendidas en la asignatura de Desarrollo de Software, como la modularización del sistema o la creación de módulos de pruebas automatizados (test unitarios en Python). Por otra parte, para el desarrollo del backend han sido necesarios conocimientos avanzados en *Programación* y en *Estructuras de Datos y Algoritmos*. De la misma manera, el poder crear una interfaz con una experiencia de usuario amigable ha requerido conocimientos en *Interfaces de Usuario*. Otros conocimientos como los adquiridos en *Ingeniería de Software* han servido de base para poder acometer una definición clara y formal de los requisitos. Por otro lado, la adaptabilidad y el enfoque iterativo del proceso se vieron influenciados por *Técnicas Ágiles de Desarrollo de Software*.

En conclusión, el desarrollo de este proyecto ha servido para aplicar una gran cantidad de conocimientos adquiridos durante los cursos de Ingeniería Informática y el resultado ha sido satisfactorio.

7

Líneas de trabajo futuro

Durante el proceso de desarrollo del sistema presentado en esta memoria, se contemplaron diversas ideas con el fin de mejorar el mismo. Algunas de ellas constan de una magnitud considerable, por lo que existen limitaciones tanto temporales como a nivel de recursos computacionales disponibles. Sin embargo, resulta interesante mencionar dichas ideas con el fin de considerarlas para futuras versiones de APIzzy, demostrando el potencial y la escalabilidad que tiene la herramienta presentada. Entre estas ideas destacan:

- **Incorporación de los métodos HTTP restantes:** como se ha mencionado a lo largo de la memoria, solamente se han tenido en consideración las cuatro peticiones más comunes, GET, POST, PUT y DELETE. Sin embargo, resultaría de gran utilidad dar soporte al resto de peticiones, como pueden ser HEAD, CONNECT, OPTIONS, TRACE y PATCH. Debido a la modularidad del sistema, sería relativamente sencillo proceder a la integración de dichos métodos.
- **Uso de Despliegue del backend en Docker:** el despliegue del backend empleando contenedores representa una manera sencilla de replicar el entorno de desarrollo, siendo posible desplegar de forma sencilla el sistema al encapsular en el contenedor la solución completa, incluyendo dependencias. De la misma manera, también representa una oportunidad de securización del código fuente del sistema.
- **Dar soporte a otros lenguajes de programación:** a la hora de expandir el sistema, esta es una de las ideas que resulta evidente su incorporación. De

la misma manera que APIZZY genera código en Python, resultaría de gran interés soportar la generación de APIs en otros lenguajes como puede ser JAVA.

- **Modelos de procesamiento de lenguaje natural:** hoy en día la inteligencia artificial ha cobrado un papel sumamente importante en la sociedad actual. Los modelos de procesamiento de lenguaje natural son sumamente precisos a la hora de realizar tareas de traducción y generación de código. Además, con el surgimiento de recientes modelos de licencia abierta como Large Language Model Meta AI 2 o LLAMA2, el cuál permite ser reentrenado para tareas precisas y desplegado en entornos empresariales de forma libre, abre un gran abanico de nuevas posibilidades. Reestructurar el proyecto de cero y abordarlo con una herramienta de este tipo podría elevar la calidad del sistema a un nivel superior, aunque se requiere de numerosos recursos tanto a nivel computacional como a nivel de datos para entrenar el modelo.

7 Líneas de trabajo futuro

Bibliografía

- [1] Grydd, *¿Cómo entender qué es la API?*, <https://grydd.com/es/como-entender-que-es-la-api-ahora/>.
- [2] M. Sampalo, *API: qué es, para qué sirve, cómo funciona y ejemplos*, <https://outvio.com/es/blog/que-es-una-api/>, mar. de 2022.
- [3] C. Valdeolmillos, *crecimiento del número de APIs, un problema para las empresas*, <https://www.muycomputerpro.com/2018/11/20/crecimiento-numero-apis-empresas>, nov. de 2018.
- [4] M. J. Martín, *Apificación y gestión de APIs: el resorte de la economía digital*, <https://profile.es/blog/apificacion-y-gestion-de-apis-el-resorte-de-la-economia-digital/>, sep. de 2017.
- [5] BOE-A-1996-8930, *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*. <https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930>.
- [6] R. (UE), *Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos) (Texto pertinente a efectos del EEE)*, es, <http://data.europa.eu/eli/reg/2016/679/oj/spa>, abr. de 2016.
- [7] BOE-A-2009-4514, *Ley 3/2008, de 29 de diciembre, de Medidas Fiscales y Administrativas*. <https://www.boe.es/buscar/doc.php?id=BOE-A-2009-4514>.
- [8] iso25000, *ISO/IEC 25010*, <https://iso25000.com/index.php/normas-iso-25000/iso-25010>.
- [9] S. J. Bigelow, *What are the types of APIs and their differences?*, <https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences>, ene. de 2023.
- [10] MuleSoft, *Estrategia de APIs para empresas*, <https://www.mulesoft.com/es/resources/api/connected-business-strategy#:~:text=Las%20API%20>.

- [11] HubSpot, *Qué es un software CRM, para qué sirve y características*, <https://blog.hubspot.es/sales/que-es-un-software-crm>.
- [12] IBM, *¿Qué es una interfaz de programación de aplicaciones (API)?*, <https://www.ibm.com/es-es/topics/api#:~:text=Las%20API%20abiertas%20son%20interfaces,formatos%20de%20solicitud%20y%20respuesta>.
- [13] M. E. Coppola, *¿Qué es una API? Definición, tipos y ejemplos*, <https://blog.hubspot.es/website/que-como-usar-api>.
- [14] IBM, *¿Qué es una API REST?*, <https://www.ibm.com/es-es/topics/rest-api>.
- [15] aws, *¿Qué es una API?*, <https://aws.amazon.com/es/what-is/api/>.
- [16] MuleSoft, *Tipos de API y cómo decidir cuál desarrollar*, <https://www.mulesoft.com/es/resources/api/types-of-apis>.
- [17] IBM, SOAP, <https://www.ibm.com/docs/es/rsas/7.5.0?topic=standards-soap>, mar. de 2021.
- [18] aws, *¿Cuál es la diferencia entre RPC y REST?*, <https://aws.amazon.com/es/compare/the-difference-between-rpc-and-rest/#:~:text=Las%20API%20de%20RPC%20permiten,de%20otra%20aplicaci%C3%B3n%20de%20chat>.
- [19] KEEPCODING, *¿Qué es el protocolo HTTP?*, <https://keepcoding.io/blog/que-es-el-protocolo-http>.
- [20] M. web doscs, *Métodos de petición HTTP*, <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>.
- [21] IBM, *Las 10 mejores herramientas de desarrollo y prueba de API*, <https://geekflare.com/es/api-tools/>.
- [22] Swagger.io, *Swagger Editor*, <https://swagger.io/tools/swagger-editor/>.
- [23] DreamFactory, *DreamFactory*, <https://www.dreamfactory.com/>.
- [24] PostMan, *What is Postman?*, <https://www.postman.com/product/what-is-postman/>.
- [25] PostMan, *About Postman*, <https://www.postman.com/company/about-postman/>.
- [26] mocky, *Postman pro and cons*, <https://designer.mocky.io/>.
- [27] readthedocs.io, *Flask-RESTful*, <https://flask-restful.readthedocs.io/en/latest/>.
- [28] PYPL, *PYPL PopularitY of Programming Language*, <https://pypl.github.io/PYPL.html>.
- [29] desarrolloweb.com, *Manual de React*, <https://desarrolloweb.com/manuales/manual-de-react.html>.

8

Anexo - English content

8.1 Introduction

In today's technological landscape, Application Programming Interfaces (APIs) play a pivotal role in interoperability and data exchange between systems. Despite their significance, the creation and management of APIs remain a challenging task, often requiring advanced technical skills and knowledge. The primary motivation behind this effort lies in the need to provide developers with a straightforward tool that reduces complexity, time, and effort when designing and deploying APIs in a specific programming language like Python.

This chapter introduces the context in which the development of the aforementioned tool is situated, along with the problem it addresses, the motivation behind its creation, and the objectives it aims to achieve.

8.2 Context

Currently, Application Programming Interfaces (APIs) have become essential elements when deploying a Service-Oriented Architecture (SOA) [1]. These tools facilitate the connection or interaction between two software systems, enabling the creation of more robust and powerful solutions [2]. APIs act as bridges between different platforms and applications, allowing them to "converse" to effectively share information and functionalities.

In recent years, many organizations have embraced the use of APIs in their infrastructures, making them a critical resource in the digital transformation process for any company due to their numerous benefits [3]. Moreover, the rise of technologies such as mobile devices, applications, e-commerce, technological startups, and emerging trends like cloud computing, the Internet of Things (IoT), and Big Data have made APIs an indispensable component in any architecture.

Well-managed APIs enable companies to innovate internally, reduce infrastructure deployment times, optimize resources, develop new business models, and enhance the customer experience. A telling statistic is that ProgrammableWeb, a renowned API repository, has witnessed a marked increase in API usage, highlighting categories such as tools, social networks, finance, e-commerce, among others [4].

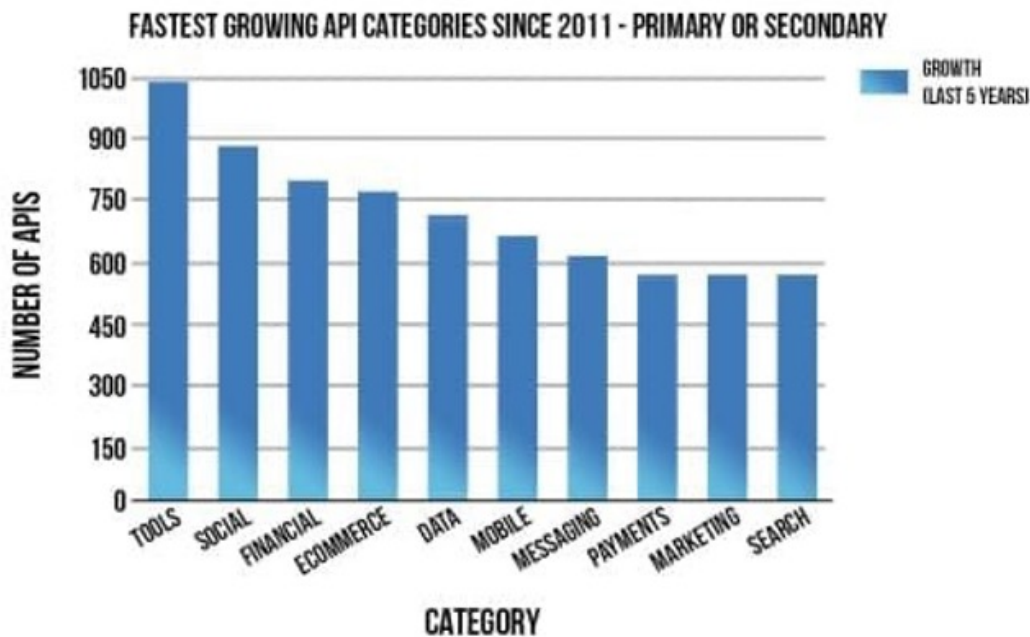


Figura 8.1. Number of APIs by Categories since 2011 (source: ProgrammableWeb).

Given the growing importance and complexity of APIs in today's technological landscape, having tools that simplify their creation while streamlining the development process is essential. It is crucial to recognize that not all members of an organization possess the technical skills, knowledge, or familiarity with coding necessary to implement an API. This reality becomes particularly evident when business needs require the rapid implementation of API-based solutions. Thus,

the availability of a tool that employs interactive and easy-to-understand forms is not only helpful but also vital. Such an application would lower the technical barrier, enabling a wider range of professionals to collaborate in achieving organizational goals through the efficient implementation of APIs.

8.3 Problem

Currently, there are few interactive and user-friendly tools available that allow users to generate APIs with specific specifications automatically. Developing these APIs can be a laborious task, consuming valuable time and demanding specific programming skills. In many organizations, teams are multidisciplinary, and not all members possess competencies in this field. This disparity in skills can become an obstacle, resulting in delays, increased costs, and sometimes solutions that are suboptimal or compromise the security of the system they are part of. Hence, the question arises: How can the creation and management of APIs be facilitated efficiently, reducing the learning curve and enabling non-technically trained professionals to actively participate in their implementation? The objective of this TFG is to address this issue by proposing an innovative and practical solution through the development of a tool that, given a set of parameters, generates a template implementing a functional API with the requested functionalities.

8.4 Objectives

The primary objective of this Final Degree Project is to design and develop APIzzy, an interactive and original tool that addresses a persistent challenge in the current technological landscape: the intricate task of creating and managing APIs. Even though APIs are vital for modern digital interaction, their development remains a hurdle, especially for those lacking deep technical expertise.

With APIzzy, a groundbreaking methodology is introduced that not only simplifies the API creation process but also democratizes its development. This allows individuals, regardless of their programming technical level, to design APIs with specific specifications.

The added value of APIzzy lies in its ability to provide a practical and effective solution suitable for professional environments, such as businesses. This innovative approach has the potential to be a game-changer, bridging the gap between the rising demand for digital interconnection and the scarce supply of specialized skills to develop these crucial connections. Through this project, the aim is not only to create a technological tool but also to make a significant contribution to the digital ecosystem, redefining how we approach and understand the world of

APIs.

8.5 State of the Art

8.6 History and Evolution of APIs

Interfaces de Programación de Aplicaciones, better known by their English acronym as APIs, represent one of the fundamental components in modern software development, serving as the primary means of communication between modules within a system. Through these interfaces, entirely independent modules can be interconnected, regardless of the programming language or platform on which they are deployed. This structured exchange of information allows systems to understand and process transmitted data, facilitating service integration and the creation of more robust and versatile applications [9].

From their early days, APIs were designed as simple tools to connect specific functions within an application or between closely related applications. However, the rise of the digital era and the need for a more connected web led to the evolution of these interfaces. More generalized APIs began to emerge, designed to serve multiple applications and enable connections between different platforms and devices.

Today, APIs are not limited to developers or large tech companies. They have infiltrated almost every corner of the modern enterprise, driving automation and efficiency across departments, boosting businesses, and connecting them with customers. From human resources to finance, sales to marketing, and customer engagement to operations optimization, these interfaces allow applications to communicate, eliminating the need for repetitive manual processes prone to errors.

It's evident that APIs are not just a passing trend in the tech world; they are a fundamental piece of the digitalization puzzle, enabling the creation of a more interconnected and collaborative digital ecosystem.

8.7 Types of APIs

- **Public APIs:** A public API, also known as an open API or external API, is an open-source application programming interface accessible via the HTTP protocol and designed to be accessible by any developer [12]. The advantages of public APIs include expanding application or platform functionality to a broader user base, promoting innovation, and enabling new forms of

8.8 Common API Protocols and Architectures

collaboration between applications and platforms. However, it's crucial to manage and secure these APIs adequately to ensure data privacy and security [9].

- **Partner APIs:** In contrast to open APIs, partner APIs are intended for use by a specific group of developers or associated companies. This allows for closer collaboration between companies, ensuring that the API is used in a way that benefits both parties. Partner APIs can also have additional security or partner-specific custom features [9], [12].
- **Internal APIs:** Internal APIs are application programming interfaces that remain hidden from external users [12]. Larger organizations often develop their own applications and systems that need to interact, usually designed for the operation of a particular company [13]. Internal APIs facilitate this communication, enabling greater efficiency and cohesion among internal systems.
- **Composite APIs:** These APIs combine multiple data or service APIs [12]. As applications have become more complex, the need to combine multiple services and functions into a single operation has arisen. Composite APIs meet this need, allowing the execution of multiple tasks and calls in a single request [9].

8.8 Common API Protocols and Architectures

APIs exchange commands and information, necessitating clear protocols and architectures. Currently, three main categories exist: REST, RPC, and SOAP. These are distinct formats used for different purposes [9].

8.8.1 REST: Representational State Transfer

RESTful APIs adhere to the principles of the Representational State Transfer (REST) architectural style [14]. This architectural style has gained popularity due to its simplicity and efficiency, being essential in today's software development landscape [13]. Clients send requests to the server using the HTTP protocol, where the server employs these requests to perform internal functions such as GET, PUT, POST, DELETE, and more, returning output data to the client. A key feature of REST APIs is that they are stateless, meaning the server does not store any data when processing requests [15]. A notable benefit of using the REST API paradigm is its adaptability, particularly because it can work with various widely used data formats like JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) [16].

Additionally, the stateless design of REST APIs ensures optimal system performance, as each request is treated as a new transaction unrelated to any prior requests. It also provides greater scalability, enabling applications to efficiently handle a large number of requests. Another notable aspect is how easily these APIs can be integrated into different platforms and programming languages due to their web-based nature. In practice, this means that mobile apps, websites, and other software applications can consume and work with data from a REST API regardless of the underlying technologies [14].

8.8.2 SOAP: *Simple Object Access Protocol*

Although it's older than REST, SOAP or Simple Object Access Protocol is still widely used, especially in enterprise applications [9]. It's a lightweight protocol for exchanging information in distributed and decentralized environments. SOAP is based on transmitting messages from a sender to a receiver. Additionally, it offers extra features such as security through WS-Security and support for UDDI (Universal Description, Discovery, and Integration) queries. However, it's more complex and demands more bandwidth and resources [17].

8.8.3 RPC: *Remote Procedure Call*

Before the popularity of REST and SOAP, RPC (Simple Object Access Protocol) dominated the realm of APIs. Even though it's less common in contemporary times, it remains a viable choice, especially when direct communication between systems is needed [9]. These APIs enable developers to invoke remote functions on external servers [18].

8.9 HTTP Protocol

HTTP (Protocolo de Transferencia de HiperTexto) is the underlying communication protocol used by the World Wide Web (WWW) and forms the basis for most online interactions. Developed by World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF), HTTP is the foundation of any data exchange on the web, enabling communication between clients and servers through the transfer of messages in plain text format, as it makes requests for data and resources.

8.10 Automatic API Generation Tools

As a result of the conducted research, numerous studies and tools with objectives similar to those of this study have been identified. In the following section, the main identified tools will be outlined to provide context and contrast with the approach of this work.

8.10.1 Swagger/OpenAPI

Swagger, also known as OpenAPI, is an open-source tool and framework used for designing, developing, building, documenting, and consuming RESTful APIs. It has become one of the most popular tools in the software industry for this purpose. The specification behind Swagger has become a standard known as OpenAPI Specification (OAS). Swagger was developed by SmartBear Software, a company known for its solutions in software development, testing, and monitoring. Although it started as an independent tool, Swagger was acquired by SmartBear in 2015 and has continued to evolve under its direction [22].

8.10.2 Postman

Postman is a popular platform used for developing, testing, documenting, and monitoring APIs. Originally designed as a Chrome extension for API testing, Postman has evolved into a comprehensive solution for the entire lifecycle of API development. Due to its extensive set of tools and features, Postman is used in a wide variety of scenarios. It provides solutions for nearly all stages of API development and maintenance [24].

8.10.3 Mocky

Mocky is an online tool that allows developers to simulate HTTP responses from an API. Its main purpose is to facilitate the creation of "mocks" or simulated responses from an API, which is particularly useful during development and testing stages. Unlike other tools that require additional software installation, Mocky operates entirely from the browser, simplifying its use. Additionally, it is an open-source tool, and its code is available on GitHub [26].

8.11 Comparison of Current Tools with the Proposed Solution

After analyzing various available tools for API management and creation, a clear trend emerges: most of these tools are primarily oriented towards virtual administration and deployment of APIs rather than automatic code generation. These solutions, while robust, don't provide an easy path for non-technical users to generate ready-to-implement API code for their websites, projects, and other applications.

Although some of these tools attempt to offer user-friendly interfaces, such as incorporating forms for users with limited programming knowledge, in many instances, users are still required to refer to additional documentation or possess certain baseline knowledge to fully utilize them.

In this context, APIzzy emerges. This project is conceived with a clear goal: to provide a user-centric experience. APIzzy removes the technical barrier by providing a platform where, thanks to its underlying infrastructure, users can generate RESTful APIs with GET, POST, DELETE, and PUT operations without dealing with technical complexities. Through the implementation of intuitive forms to gather the required information, APIzzy positions itself not only as a dynamic tool but also as a user-friendly and accessible tool for any user, regardless of their level of technical knowledge.

8.12 Underlying Technologies

While there are various administration tools like Postman and API deployment platforms like Mocky, as highlighted earlier, the fundamental differences between them and APIzzy are crucial. It's essential to determine the most suitable programming languages for its creation. In this regard, a high-level, preferably object-oriented programming language for the backend was sought. This is because APIzzy requires code that is dynamic, reusable, well-structured, and easy to maintain.

According to multiple sources [27]–[29], the main object-oriented languages include Java, Python, R, SQL, and C/C++. While R and SQL are primarily oriented towards data analysis and C/C++ is essential for computationally intensive tasks in data science, this leads us to consider Python and Java as the most viable options for developing this project. One crucial aspect in designing APIzzy is to provide a genuinely intuitive and user-friendly interface. When choosing a framework for implementing the user interface, it's essential that it provides real-time interactivity, reusable components, and high performance. Hence, the decision was made to

use ReactJS.

8.13 Proposed Solution / Method

8.13.1 Proposed Solution

Form Creation

For the creation of the form component, a file named *APIForm.js* was created. This file contains all the functions, constants, and JSX code, which allows for writing structures similar to HTML within JavaScript code. The form was created using Reactstrap with the Form component.

To structure input fields and labels within a form, *FormGroup* was used. Thanks to these components, it was possible to customize the form to meet requirements. These components allowed for the addition and removal of resources or conditions. If the user's API had two identical methods, within the *FormGroup*, a function *add* or *delete* was called to remove or add that group of input fields in the form. Also, within these *FormGroup* the desired type of entry is indicated (text field type, *checkbox* type, selection from several resources, among others). With this wide selection, a dynamic and user-friendly form was created. For certain fields, different validations were implemented to control entries. These validations were possible thanks to the *props* of the *FormGroup* component, specifically from *input*. The *props* (short for properties in English) in React, are a way to pass data from parent components to child components. In this case, the *pattern* property was used, which is used to set a regular expression against which the value of the input field is validated. The *required* property was also used to indicate that the input field is mandatory and must be filled out before the form can be submitted.

Data Submission (POST)

As previously mentioned, the data retrieved by the form must adhere to a specific structure to be processed by the *backend*. For this, the creation of an "initial-State" object was chosen. The *initialState* object defines an initial state, common in React applications, especially when using state management libraries like Redux or React's own *useState* hook. Within this object, default structures and values are set. In general, *initialState* serves as a template or default structure for an application or React component's state. Given the logic and functionality implemented in the application, this initial state is updated and changes as the user interacts with the form and as other events are processed.

This object is used in an asynchronous function called *sendToDB* which sends the object, executing a POST request to a local server on port 5001, sending the data collected from the form. This request is achieved with the implementation of the Axios library previously discussed.

API Download

A functionality was implemented in which, after receiving data from the server, a Blob is created, an object that represents data in binary or text format. Subsequently, a link is added to the document to allow the download of this Blob. By simulating a click on said link, the download of the file named *APIzzy.py* begins. Once the download is finished, the link is automatically removed from the document.

Backend Design

Considering the requirements defined in section of requirements, the architecture designed for APIzzy's *backend* is discussed below. It should be emphasized that the modularity and scalability of the system have been priorities when organizing and encapsulating each of the functionalities, taking into account that new modules might be added in the future to cover HTTP requests omitted in this initial version. The main module responsible for generating the code is *APIGenerator*, utilizing the functionalities of the rest of the classes that make up the system. First, it can be observed that it directly uses the *ClassGenerator* module, which in turn uses *MethodGenerator*, which instantiates a class from each of the specific generators for each HTTP method.

Before evaluating each module in detail, certain clarifications about the format in which the code is produced should be made. As mentioned, Flask Restful allows templates to be used for this project due to the similar structure among all APIs produced with this framework. Each API is instantiated as a Python class, and each of the HTTP methods is reflected in methods associated with that class. Finally, the mounting point for each endpoint in the URL where requests are intended to be made is chosen and associated with each API.

The specific functions of each of the modules are detailed below.

APIGenerator

Firstly, this class contains three attributes. Since the system allows for the generation of several APIs with a form request, it first has a list with each of the

classes to generate. There's also an attribute to save each of the mounting points related to each API and another for the imports that must be generated in the header in case additional libraries are needed when working with special data structures like JSON.

Regarding its functionalities, it has a method that is responsible for delegating the API generation work to ClassGenerator, saving the completed templates in return, as well as the necessary imports and relevant endpoints. The second method takes care of forming the final file from the templates obtained with the previous method, joining the different APIs, imported libraries, and endpoints.

ClassGenerator

This is the class directly used by APIGenerator. It possesses attributes such as the name of the class or API to be implemented, the corresponding mounting point, the HTTP methods that need to be implemented, and the libraries that need to be imported if necessary.

The main function of this module is to filter the HTTP methods that need to be generated and call the necessary functions in the underlying classes to produce these methods. Once the API with all the requested methods and mounting point is created, these values are returned to APIGenerator to form the file sent to the end user.

MethodGenerator

MethodGenerator has an instance of each of the specific method generators that will be explained in later sections. Based on the filtered information it receives from ClassGenerator, it proceeds to make a second filter based on the data structure intended for use in the method's body, then uses the relevant code generator. Again, it simply applies a new filter to the data obtained from the user's request and delegates the generation of the main body of each method to the aforementioned individual generators.

GetGenerator, PostGenerator, PutGenerator, DeleteGenerator

These are the code generators responsible for the HTTP methods of type GET, POST, PUT, and DELETE. They contain a function for each type of data structure specified that said method must support according to the requirements discussed earlier in section of requirements. Within these functions, the code is formed from templates, default values are set to prevent incorrect values, and a fully functional

script is formed with the correct indentation and simple error handling.

If there's a desire to include new generators for the HTTP methods not covered by APIzzzy, it would be enough to add a new class for that method and include a rule in MethodGenerator's filtering, giving the system great flexibility when incorporating new functionalities.

Backend-Frontend Communication API

To conclude the section related to the methodology used in the development of the APIzzzy system, it's necessary to address the communications between the user-facing form and the automatic code generator. It has been previously mentioned that this communication is done through a REST API, which only responds to the POST method at the `"/form"` endpoint.

The purpose of this API is to receive data in the JSON format specified in section 3.2 within the content of the request. Upon receiving the data, it instantiates an APIGenerator class to which it passes the JSON configuration. This instance is responsible for generating the file, and once it has been generated, the API formats the code according to the PEP-8 standard using the command-line tool `"black,"` an open-source code formatter.

Finally, when the final file is fully ready and functional, it's sent back to the deployed web service, where it's automatically downloaded through the client's browser.

8.14 Validation and Testing

In this section, we describe the strategic tests carried out to ensure the proper functioning of APIzzzy. We will address unit tests, which confirm the individual operability of the modules; integration tests, which ensure cohesion between these modules; and acceptance tests, which validate that the overall solution meets user expectations and the planned technical requirements.

8.15 Frontend Validations:

The frontend of an application plays a crucial role, serving as the direct interaction point between the user and the system. Its efficiency, performance, and user experience largely determine the perception and satisfaction of the end user. Properly validating the frontend not only ensures that individual components function as expected but also ensures that they interact harmoniously with each other

and the backend, providing a cohesive and error-free experience. In this context, unit tests play a fundamental role in validating the smallest functionality units in the frontend.

8.15.1 Unit tests

Frontend unit tests focus on assessing and verifying the correct functionality of the smallest and isolated components of the application, ensuring that they act and respond as expected. Following the agile methodology adopted throughout the project's development, individual component tests for APIzzy's frontend have been conducted.

■ Individual Components:

- Validate that each component (e.g., buttons, input fields, drop-down lists) renders correctly under different scenarios.
- Ensure that components display appropriate data and handle states correctly (e.g., a button should be disabled when certain conditions are not met).
- Validate that each component (e.g., buttons, input fields, drop-down lists) renders correctly under different scenarios.
- Ensure that components display appropriate data and handle states correctly (e.g., a button should be disabled when certain conditions are not met).

■ Event Handlers:

- Check that events, such as mouse clicks or keyboard inputs, trigger the expected actions. For instance, clicking a 'Send' button should activate the corresponding function, and the form data should be captured.
- Ensure that 'onChange' events in input fields update the application's state as expected.

■ Forms:

- Verify that forms validate user input according to defined rules (e.g., checking the format of an IP address).
- Ensure that error messages display appropriately when user input does not meet established criteria.

■ Integration with State:

- Check that changes in the application's state are adequately reflected in the UI. For example, if a user selects 'SQL' as a source in your APIzzzy application, SQL-related fields should display correctly.
- Ensure that actions that modify the state (like selecting an option on a navigation bar) update the state correctly.

65 manual tests were carried out for components, buttons, fields, input validations, as described above. The code coverage percentage achieved was 89

It is worth noting that, thanks to the agile methodology adopted, constant tests were carried out throughout the entire project development, thus ensuring the correct operation and performance of the website.

8.15.2 Integration tests

These tests seek to ensure that the individual components of APIzzzy's frontend not only function in isolation but also collaborate correctly when integrated into the complete system. Specifically, the following tests were conducted with satisfactory results:

- Validate that when selecting a method (GET, POST, PUT, DELETE) on the interface, not only are the appropriate fields displayed, but they are also initialized and processed correctly.
- Confirm that data entered in one component is transmitted, received, and processed adequately by other components, maintaining information integrity.
- Ensure transitions and flows between different views or application components are coherent and run without interruptions or unexpected errors.
- Verify that dependencies between components, such as function calls or access to shared variables, are managed optimally, avoiding conflicts or failures.

8.16 Backend Validations

The backend of an application, especially in tools like APIzzzy, constitutes the backbone that supports and processes all the fundamental operations that the system performs. It serves as a bridge between the frontend and data transformation, ensuring that information is processed, stored, and retrieved efficiently and safely. The performance, security, and reliability of the backend are essential to ensure that the application operates without interruptions and meets user

expectations. Properly validating the backend is not only crucial to ensure correct data management and processing but also to guarantee smooth and efficient communication with the frontend, avoiding potential failures or vulnerabilities. In the context of APIZZY, unit tests become essential, as they allow validating each function and method of the API, ensuring that each endpoint responds and acts as expected under various requests and scenarios.

8.16.1 Unit tests

Following the agile methodology adopted throughout the project's development, to produce functional code that is automatically and continuously tested, a testing process under the Test Driven Development (TDD) paradigm has been applied.

This method ensures that every new functionality is preceded by its corresponding test, thus certifying its correct operation from the beginning.

In APIZZY's backend unit tests, emphasis was placed on:

■ Endpoints:

- Ensure that each endpoint responds correctly to different HTTP methods (GET, POST, PUT, DELETE) and returns the expected results.
- Verify that data validation on each endpoint is robust and consistent, avoiding potential security vulnerabilities or data corruption.
- Test error handling mechanisms to ensure that when an error occurs, it is correctly captured and an appropriate message is returned to the user.

■ Data Processing:

- Ensure that data transformations and processing operations are carried out correctly and efficiently, maintaining data integrity.
- Verify that database interactions (like CRUD operations) are executed correctly, avoiding potential conflicts or data losses.

■ Integration with Third-party Services:

- Check that connections to external services or APIs are established correctly and that the data exchanged is processed appropriately.
- Ensure that error handling mechanisms are in place for potential connectivity issues or failures in third-party services.

■ Security Mechanisms:

- Ensure that security measures, like authentication and authorization mechanisms, work correctly, safeguarding user data and operations.
- Test potential vulnerabilities to prevent unauthorized access or potential attacks, ensuring that security patches and updates are applied effectively.

The results of the backend unit tests yielded a code coverage of 93%, with a total of 142 manual tests carried out for the various components, functionalities, and endpoints described above.

Constant testing was conducted throughout the entire project development, benefiting from the agile methodology. This approach has ensured the correct operation, performance, security, and reliability of the backend, which plays a vital role in the overall operation of the application.

8.16.2 Integration tests

Integration tests on the backend focused on ensuring the correct interaction between the different components and functionalities of the application. Some of the tests carried out include:

- Confirming that endpoints communicate correctly with the database, correctly processing data requests and responses.
- Ensuring that third-party services or external APIs are integrated correctly and data exchange is consistent.
- Testing that the frontend and backend communicate effectively, ensuring data integrity and correct operation of the entire application.
- Verifying that security mechanisms are integrated effectively into the entire system, safeguarding user information and operations.

The integration tests have been designed to guarantee the smooth operation of the entire system, ensuring that each component interacts optimally with the others, avoiding potential failures or vulnerabilities. The satisfactory results of these tests confirm the correct integration and operation of the various components of APIzzy.

8.17 Acceptance tests

Acceptance tests aim to confirm that the APIzzy solution meets the technical requirements and user expectations. These tests are crucial to ensure that the ap-

plication not only works as expected but also provides an optimal user experience. In the context of APIzzzy, the acceptance tests have been designed based on the initial requirements and specifications of the project. The tests have been carried out by a team of experts, who have simulated the most common and critical use scenarios, verifying that the solution meets the proposed objectives.

Some of the aspects evaluated in the acceptance tests include:

- The correct operation of the frontend and backend, ensuring smooth interaction and optimal user experience.
- The efficiency and reliability of the data processing and transformation mechanisms.
- The robustness and security of the application, avoiding potential vulnerabilities or unauthorized access.
- The adaptability and scalability of the solution, ensuring that it can handle an increasing number of users or data without suffering performance issues.
- The overall satisfaction of users, collecting feedback and suggestions to improve the application in future versions.

The results of the acceptance tests have been highly satisfactory, confirming that the APIzzzy solution meets the technical requirements and user expectations. Feedback collected during these tests has also provided valuable insights for the continuous improvement of the application.

8.18 Conclusions

Initially, an in-depth analysis of the state of the art related to the field of study was carried out to lay the foundation for the project and propose a solution that is innovative and distinct from those already available in the market. Similarly, this process made it possible to select the tools best suited for a development like the one detailed in this report. Once this analysis was completed, the requirements that the application had to meet were defined, taking into account potential future enhancements, as well as the simplicity and interactivity intended for this project. Throughout the development, inconsistencies were found in the backend architecture, which were refined in various iterations resulting from the application of an agile methodology, culminating in the model presented in this report. Similarly, in order to meet the final requirements, it was necessary to adjust the code and the data structure formats exchanged between the form and the automatic code

generator. The final step consisted of interconnecting both modules through the REST API described in the methodology section.

In summary, reflecting on the entire process, it's evident that, despite the challenges and obstacles encountered along the way, we managed to implement a robust and meticulous methodology. This methodology has proven to be essential for the successful comprehensive development of the system. From the early stages of exploring the state of the art to the exhaustive definition of requirements, through the conception and design of the architecture, the development process of the module itself, its rigorous validation, and finally, the careful creation of comprehensive and detailed documentation.

This holistic approach has not only facilitated the successful completion of the project but has also provided the student with a valuable opportunity to apply and practice a broad range of knowledge acquired during their academic career. It is at this juncture where theory becomes tangible action, significantly enriching the educational experience and preparing the student for future challenges in the professional field.

Throughout the complete system design cycle, techniques learned in the Software Development course were employed, such as system modularization or the creation of automated test modules (unit tests in Python). On the other hand, advanced knowledge in Programming and in Data Structures and Algorithms was essential for the backend development. Similarly, the creation of a user-friendly interface required skills in User Interfaces. Other skills, like those acquired in Software Engineering, provided the foundation to undertake a clear and formal definition of requirements. Meanwhile, the adaptability and iterative focus of the process were influenced by Agile Software Development Techniques. In conclusion, the development of this project served to apply a vast array of knowledge gained during the Computer Engineering courses, and the outcome has been satisfactory.

8.19 Future Work Lines

During the development process of the system presented in this report, various ideas were contemplated to enhance it. Some of these have a considerable magnitude, leading to temporal and computational resource constraints. However, it is worth mentioning these ideas with the aim of considering them for future versions of APIzzy, demonstrating the potential and scalability of the presented tool. Notable among these ideas are:

- **Integration of the Remaining HTTP Methods:** As mentioned throughout the report, only the four most common requests, GET, POST, PUT, and

DELETE, have been considered. However, it would be highly useful to support the other requests, such as HEAD, CONNECT, OPTIONS, TRACE, and PATCH. Due to the modularity of the system, integrating these methods would be relatively straightforward.

- **Deployment of the Backend in Docker:** Deploying the backend using containers represents an easy way to replicate the development environment, allowing for straightforward system deployment by encapsulating the complete solution, including dependencies, in the container. Similarly, it also represents an opportunity to secure the system's source code.
- **Support for Other Programming Languages:** As the system expands, incorporating this idea becomes evident. Just as APIZZY generates code in Python, it would be highly interesting to support the generation of APIs in other languages, such as JAVA.
- **Natural Language Processing Models:** Nowadays, artificial intelligence has taken on an extremely important role in today's society. Natural language processing models are incredibly accurate when performing tasks like translation and code generation. Furthermore, with the emergence of recent open-source models such as **LLAMA2g** or **LLAMA2a**, which can be retrained for specific tasks and freely deployed in business environments, a wide range of new possibilities opens up. Reconstructing the project from scratch and approaching it with a tool of this kind could elevate the system's quality to a higher level, although significant computational and data resources are required to train the model.

