Deusto

Facultad de Ingeniería
Ingeniaritza Fakultatea

Máster Universitario en Computación y Sistemas Inteligentes

Master's Degree in Computation and Intelligent Systems

Proyecto fin de máster

Master's final project

Diseño e implementación de un marco tecnológico para la adopción de prácticas modernas en proyectos de inteligencia artificial

Asier Villar Marzo

Director: Mikel Emaldi Manrique

Bilbao, julio de 2024

Deusto

Facultad de Ingeniería
Ingeniaritza Fakultatea

Máster Universitario en Computación y Sistemas Inteligentes

Master's Degree in Computation and Intelligent Systems

Proyecto fin de máster

Master's final project

Diseño e implementación de un marco tecnológico para la adopción de prácticas modernas en proyectos de inteligencia artificial

Asier Villar Marzo

Director: Mikel Emaldi Manrique

Bilbao, julio de 2024

RESUMEN

En un mundo empresarial cada vez más dinámico y competitivo, el uso de metodologías y estándares modernos se ha convertido en una necesidad para las organizaciones que buscan mantenerse relevantes y competitivas. En este contexto, la adopción de metodologías ágiles, MLOps (Machine Learning Operations) y otras prácticas modernas se presenta como un elemento fundamental para facilitar la cooperación entre equipos, mejorar la calidad del producto y reducir los tiempos de desarrollo.

Las empresas que logran adaptarse y adoptar estos enfoques modernos experimentan una serie de beneficios significativos. En primer lugar, les permite responder de manera más rápida a los cambios en el entorno empresarial, lo que otorga una ventaja competitiva crucial. Además, la incorporación de prácticas de MLOps permite a las organizaciones gestionar de manera eficiente los modelos de machine learning en producción, garantizando su rendimiento y fiabilidad a lo largo del tiempo. Esto es especialmente relevante en un contexto donde el uso de la inteligencia artificial y el machine learning está cada vez más extendido dentro de la industria.

La implementación de estos estándares no está exenta de desafíos y dificultades. Cambiar la forma en que una empresa opera y se organiza puede ser un proceso complejo que requiere un cambio cultural significativo, así como la adopción de nuevas herramientas y tecnologías. El objetivo de este proyecto es diseñar un stack tecnológico que permita a una empresa adoptar estas metodologías y estándares de manera efectiva, facilitando la transición y reduciendo la complejidad de la misma.

DESCRIPTORES

MLOps, Machine Learning, CI/CD, Automatización, Metodologías ágiles

Contents

List of Tables

List of Figures

1. INTRODUCCIÓN

El presente documento constituye la memoria del Proyecto de Fin de Máster (PFM) del Máster en Computación y Sistemas Inteligentes de la Universidad de Deusto, realizado en colaboración con Tecnalia Research and Innovation, un centro de investigación aplicada. Su propósito es documentar exhaustivamente el trabajo llevado a cabo, desde la conceptualización del problema hasta la implementación de la solución.

En la actualidad, los sistemas basados en inteligencia artificial (IA) desempeñan un papel más significativo en diversos sectores como el de la medicina o la industria. Cada vez son más las empresas que buscan incorporar soluciones IA para mejorar su eficiencia y competitividad. Sin embargo, el desarrollo de modelos de IA es un proceso complejo que demanda considerable esfuerzo y experiencia técnica, así como un mantenimiento continuo y una monitorización constante para garantizar su óptimo rendimiento.

En el contexto de la investigación y desarrollo, es común centrar los esfuerzos en la búsqueda de nuevas soluciones que aplicar a aspectos concretos dentro de una temática. Sin embargo, esta se ve gravemente ralentizada debido a la predominancia de tareas repetitivas y procesos manuales que consumen una gran cantidad de tiempo, pero no aportan ningún tipo de valor. Además, la falta de un buen sistema de gestión del conocimiento, conlleva por parte de las empresas a la pérdida de información valiosa que ha sido descubierta y que podría ser reutilizada en futuros proyectos. Todo ello sumado a la ausencia de un marco de trabajo común, dificulta en gran medida la cooperación, ya que cada miembro del equipo requiere de un tiempo adicional para adaptarse a las especificaciones de cada proyecto.

Durante los últimos años, el concepto de MLOps (Machine Learning Operations) ha ido ganando popularidad hasta convertirse en un elemento disruptivo en cuanto a desarrollo de modelos de IA se refiere. Este nuevo paradigma, que toma como base las prácticas DevOps (Development Operations), busca combinar la IA y el desarrollo de software moderno con el objetivo de tener un mayor control sobre el ciclo de vida de los modelos, permitiendo una entrega continua. Estas prácticas han demostrado ser muy efectivas en la industria, pero en cambio no han sido totalmente acogidas en el ámbito de la investigación. Esto puede deberse a multitud de factores, ya sea por la resistencia al cambio, la falta de comprensión de sus beneficios o la falta de recursos dedicados a su implementación.

Uno de los principales desafíos en la implementación de estas prácticas radica en las diferencias de conocimientos entre los miembros del equipo, lo que genera fricción. Por tanto, resulta crucial considerar el punto de partida del equipo y diseñar un proceso intuitivo. Este proyecto propone un estándar que aborda esta problemática, recopilando las mejores prácticas, tecnologías y procedimientos desarrollados hasta la fecha, y las adapta a un marco de trabajo común que facilite la cooperación y la reutilización de conocimiento.

1.1. MOTIVACIÓN

Mi motivación para empezar este proyecto surge de la necesidad identificada por Tecnalia de explorar el ámbito del software, específicamente enfocándose en la creación de una herramienta que agilice el proceso de desarrollo de modelos de aprendizaje automático. Esta herramienta tiene como objetivo permitir que los investigadores dediquen más tiempo a la investigación en sí y menos a tareas repetitivas, al mismo tiempo que fomenta la cooperación y la reutilización del conocimiento de manera eficiente.

Desde el principio, encontré este tema sumamente interesante, especialmente considerando mi experiencia previa en el desarrollo de software. Percibí la oportunidad de aportar una solución innovadora que podría ser de gran ayuda para abordar los desafíos identificados por Tecnalia.

Además, el hecho de que la integración de buenas prácticas en el desarrollo de modelos de IA sea un tema en constante crecimiento, pero aún poco explorado en el ámbito de la investigación, me motiva aún más. Esta situación me brinda la oportunidad de realizar una contribución significativa que no solo beneficiará a Tecnalia, sino que también podría ayudar a otros equipos de investigación que se enfrenten a problemáticas similares.

1.2. ESTRUCTURA DEL DOCUMENTO

En esta sección, se presenta la estructura del documento de forma clara y organizada. Se brinda una visión general de cómo se han organizado los diferentes capítulos y secciones para abordar de manera coherente y completa todos los aspectos relevantes del proyecto. Además, se proporciona una breve descripción de cada capítulo, destacando su contenido y su contribución al conjunto de la memoria. Esta sección permite al lector tener una guía clara sobre cómo está estructurado el documento y qué puede esperar encontrar en cada sección.

- **Introducción.** En este capítulo se presenta de forma breve el objetivo principal del proyecto, su impacto deseado y la motivación detrás de su realización. Además, se realiza una breve descripción del problema a resolver y se enumeran de manera ordenada los capítulos que componen el proyecto.
- **Antecedentes y justificación.** Se proporciona un estudio del estado del arte y las últimas tendencias, y se justifican las antecedentes existentes durante el desarrollo del proyecto.
- **Alcance y objetivos.** Se definen de manera detallada tanto el objetivo principal como los objetivos secundarios del proyecto. También se establece el alcance del proyecto, que se describe mediante una lista concisa de elementos que se encuentran dentro y fuera del proyecto.
- **Metodología.** Se describe la metodología de trabajo utilizada durante el desarrollo del proyecto, así como la metodología creada para la resolución del problema.
- **Memoria técnica.** Se explican en detalle todos los aspectos técnicos del mismo. Se incluyen la arquitectura del sistema integral, las herramientas utilizadas para el desarrollo, los requisitos del sistema y las incidencias encontradas entre otros.
- **Proceso de desarrollo.** En este capitulo se presenta el proceso de desarrollo utilizado en el proyecto. Se describe de manera detallada la metodología y las prácticas empleadas durante la resolución del problema. Proporciona una visión general del enfoque adoptado en el desarrollo del proyecto y cómo se aseguró la calidad y eficiencia en la implementación del sistema. También se discuten posibles limitaciones de los métodos y se proponen recomendaciones para investigaciones futuras.
- **Experimentación.** En este apartado se describe el proceso de experimentación llevado a cabo en el proyecto. Se detallan los experimentos realizados, las diferentes representaciones del problema, los datos recopilados y los resultados obtenidos. Además, se analizan e interpretan los resultados para sacar conclusiones relevantes y respaldar las decisiones tomadas en el proyecto.

- **Planificación y presupuesto.** Se detallan las fases y tareas del proyecto, se organizan cronológicamente indicando su duración. También se incluye un esquema de descomposición del trabajo y el plan de recursos humanos. Además, se incluyen los costes totales del proyecto, incluyendo los materiales y los recursos humanos.
- **Conclusiones y trabajo a futuro.** Se presentan las reflexiones realizadas tras la finalización del proyecto, así como las lecciones aprendidas y los conocimientos adquiridos. Además, se presentan ideas o propuestas que podrían ser utilizadas o implementadas en futuras investigaciones.
- **Abreviaturas, acrónimos y definiciones.** Se proporcionan explicaciones sobre el significado de ciertos términos, acrónimos o abreviaturas mencionadas en la memoria y que se consideran relevantes.
- **Bibliografía.** Se incluye una lista de referencias bibliográficas utilizadas durante el desarrollo de la memoria.
- **Anexos.** Se incluyen documentos independientes a la memoria del proyecto, pero considerados lo suficientemente relevantes como para ser adjuntados en documentos separados.
  - **Anexo I, Manual de usuario.** Se proporcionan las instrucciones necesarias para que cualquier usuario, independientemente de su nivel de conocimiento sobre el tema del proyecto, pueda poner en marcha el sistema inteligente y aprovechar todas sus funcionalidades.
  - **Anexo II, Dimensión ética del proyecto.** Se realiza un análisis ético del proyecto para garantizar que en su conjunto sea considerado éticamente aceptable y una contribución positiva para la sociedad.

2. ANTECEDENTES Y JUSTIFICACIÓN

2.1. ESTADO DEL ARTE

2.1.1 Diseño Atómico

El diseño atómico es una metodología de diseño que se centra en la creación de sistemas de diseño modulares y reutilizables. La idea principal es dividir las diferentes funcionalidades de un sistemas en sus partes más fundamentales, de manera que cada una de estas partes pueda ser reutilizada en diferentes contextos. Este enfoque permite tener un mayor control sobre cada una de las partes del sistema, facilitando su mantenimiento, documentación y reutilización. Originalmente, el diseño atómico ha sido aplicado en el diseño de interfaces de usuario, pero su filosofía puede ser aplicada a cualquier sistema de diseño modular. En el contexto de este proyecto, el diseño atómico se aplicará al diseño de un sistema de componentes para el desarrollo de modelos de aprendizaje automático.

Dentro del diseño atómico, los componentes se dividen en cinco categorías principales, que representan diferentes niveles de abstracción. Estas categorías son: átomos, moléculas, organismos, plantillas y páginas. Cada una de estas categorías representa un nivel de abstracción diferente, y se relaciona con las demás categorías de manera jerárquica. La figura 2.1 muestra la estructura tradicional del diseño atómico.

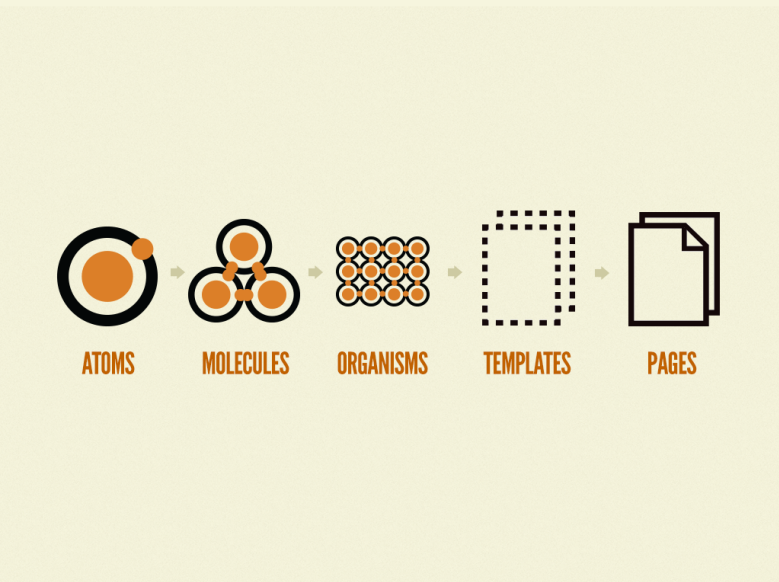

Figure 2.1: Estructura tradicional diseño Atómico

A continuación, se describen brevemente cada una de las categorías:

- **Átomos:** Los átomos son los componentes más básicos de un sistema de diseño. Representan las funcionalidades más fundamentales, solo tienen una responsabilidad y no dependen de otros componentes.
- **Moléculas:** Las moléculas son la combinación de varios átomos para formar una funcionalidad más compleja. Representan la combinación de diferentes funcionalidades básicas para

formar una funcionalidad más compleja.

- **Organismos:** Los organismos son la combinación de varias moléculas y átomos para formar una funcionalidad completa.
- **Plantillas:** Las plantillas son la combinación de varios Organismos para dar forma a un contenido o funcionalidad completa.
- **Páginas:** Las páginas son la combinación de varias plantillas.

Esta estructura jerárquica permite que los componentes sean reutilizados en diferentes contextos, y que cada uno de ellos pueda ser modificado de manera independiente. Además, se facilita la documentación y el mantenimiento de los componentes, ya que cada uno de ellos es independiente de los demás. Podemos ver multitud de ejemplos de diseño atómico en grandes empresas y que nosotros utilizamos a diario, como por ejemplo en la creación de sistemas de diseño Microsoft Fluent Design o Google Material Design entre otros.

Aunque el diseño atómico se ha aplicado tradicionalmente a la creación de interfaces de usuario, su filosofía puede ser aplicada a cualquier sistema de diseño modular. En el contexto de este proyecto, el diseño atómico se aplicará para la creación de un sistema de componentes en el desarrollo de modelos de aprendizaje automático. Traeremos la filosofía del diseño atómico y la adaptaremos a nuestro contexto, con las particularidades y necesidades que requiere el desarrollo de modelos de aprendizaje automático.

2.2. ANTECEDENTES

3. OBJETIVOS Y ALCANCE

En esta sección se introducen los objetivos del proyecto, habiéndose realizado una division entre el principal y los secundarios. Ademas de ello, se presentan los elementos que forman el alcance, asi como se comentan otros que no.

3.1. OBJETIVOS GENERALES

El objetivo principal del proyecto es diseñar e implementar un estándar tecnológico y operacional que cubra las necesidades más comunes dentro de un equipo de *data science*. Se busca agilizar los tiempos de desarrollo y estandarizar los procesos, con el fin de facilitar la colaboración entre investigadores y la reutilización del conocimiento. A continuación, se detallan los objetivos específicos que guiarán el desarrollo:

- **Agilizar el proceso inicial de proyectos:** Optimizar las primeras etapas de los proyectos, identificando y eliminando aquellos procesos repetitivos que no aportan valor y que puedan retrasar su puesta en marcha.
- **Facilitar la colaboración entre investigadores:** Implementar herramientas y métodos que fomenten una cooperación fluida y efectiva entre los miembros del equipo de investigación, con el fin de potenciar la sinergia y aprovechar al máximo el conocimiento colectivo.
- **Definir procesos mediante buenas prácticas:** Establecer un marco de trabajo basado en buenas prácticas de gestión de proyectos, con el objetivo de estandarizar los procesos y garantizar su eficiencia y calidad.
- **Automatizar el desarrollo de modelos robustos:** Investigar y aplicar técnicas que contribuyan al desarrollo automático de modelos de aprendizaje automático, con el fin de reducir el tiempo y el esfuerzo necesarios para obtener resultados de calidad.
- **Promover la reutilización del conocimiento:** Desarrollar mecanismos y herramientas que faciliten la captura, organización y difusión del conocimiento generado durante el desarrollo de los proyectos, con el propósito de fomentar su reutilización en futuras investigaciones y actividades relacionadas.

El cumplimiento de estos objetivos se espera que no solo mejore la eficiencia y la calidad de los proyectos, sino que también contribuya a la creación de un entorno de trabajo más colaborativo y enriquecedor para los miembros del equipo de investigación.

3.2. ALCANCE

En esta sección se definen los límites del proyecto, estableciendo lo que está incluido y excluido dentro mismo. Se describirá de manera detallada las actividades que forman parte del desarrollo final, así como aquellos elementos que no están incluidos en el alcance del proyecto. Aunque el enfoque de este proyecto podría aplicarse a una amplia variedad de problemas en el ámbito del aprendizaje automático, en el contexto de este TFM nos centraremos en tres de los casos más comunes dentro del marco de las series temporales: Forecasting, Clasificación y Detección de Anomalías. A continuación, se detallan las actividades que forman parte del alcance del proyecto.

3.2.1 Dentro del alcance

- **Integración de sistemas externos:** Se incluirá la configuración de sistemas externos, como plataformas MLOPs o herramientas de visualización, con las plantillas de proyectos base. Esto permitirá una integración más fluida y rápida de estos sistemas con los proyectos, facilitando el flujo de datos y la visualización de resultados.
- **Plantillas de proyectos base:** Desarrollar plantillas para los tres problemas de series temporales comentados anteriormente. Estas plantillas servirán como punto de partida para proyectos específicos dentro de cada uno de estos dominios y definirán desde el principio una estructura y un conjunto de herramientas comunes. Además, se incluirán ejemplos de código y documentación que faciliten su uso y comprensión.
- **Componentes esenciales:** Identificar y almacenar los componentes esenciales de cada proyecto, incluyendo modelos, algoritmos, métricas de evaluación y preprocesamiento de datos. Estos componentes se almacenarán y documentarán de forma que puedan ser reutilizados en futuros proyectos, facilitando la transferencia de conocimiento.
- **Proceso de AutoML:** Diseñará y ejecutar procesos de AutoML (Machine Learning Automatizado) que demostrará cómo se pueden combinar los conocimientos adquiridos de todos los proyectos para desarrollar un sistema de aprendizaje automático automatizado. Este proceso utilizará las plantillas y componentes esenciales almacenados para generar modelos de forma automática.

3.2.2 Fuera del alcance

- **Desarrollo de modelos específicos:** Aunque se incluirán ejemplos de modelos y algoritmos, el desarrollo de modelos específicos para problemas concretos no forma parte del alcance de este proyecto. Se espera que los modelos desarrollados sean generales y puedan ser adaptados a problemas específicos por los usuarios.
- **Despliegue de modelos:** El despliegue de modelos en producción no forma parte del alcance de este proyecto. Se espera que los modelos desarrollados puedan ser desplegados en sistemas de producción, pero no se incluirá en este proyecto. El enfoque se centrará exclusivamente en el desarrollo de los mismos.

4. METODOLOGÍA DEL PROYECTO

En esta sección, se describirá la metodología para abordar tanto el desarrollo del software como la propia metodología de investigación. La metodología de desarrollo de software se basa en el uso de metodologías ágiles [1], concretamente en la metodología Scrum [2]. Por otro lado, la metodología de investigación es una metodología propia diseñada para abordar problemas dentro del marco de trabajo de los problemas combinatorios NP-Hard diseñada por el autor de este trabajo. Esta metodología se basa en el uso de técnicas de IL.

5. DESARROLLO DEL PROYECTO

En esta sección se detallan los aspectos más relevantes del desarrollo del proyecto. Se profundizará en los diferentes aspectos del diseño, la implementación y la prueba de los sistemas y componentes desarrollados. Además, se describirán las herramientas y tecnologías utilizadas, así como los procesos y metodologías empleadas para el desarrollo del proyecto. A continuación, se muestra una vista general de los diferentes elementos que conforman la estructura del marco de trabajo.

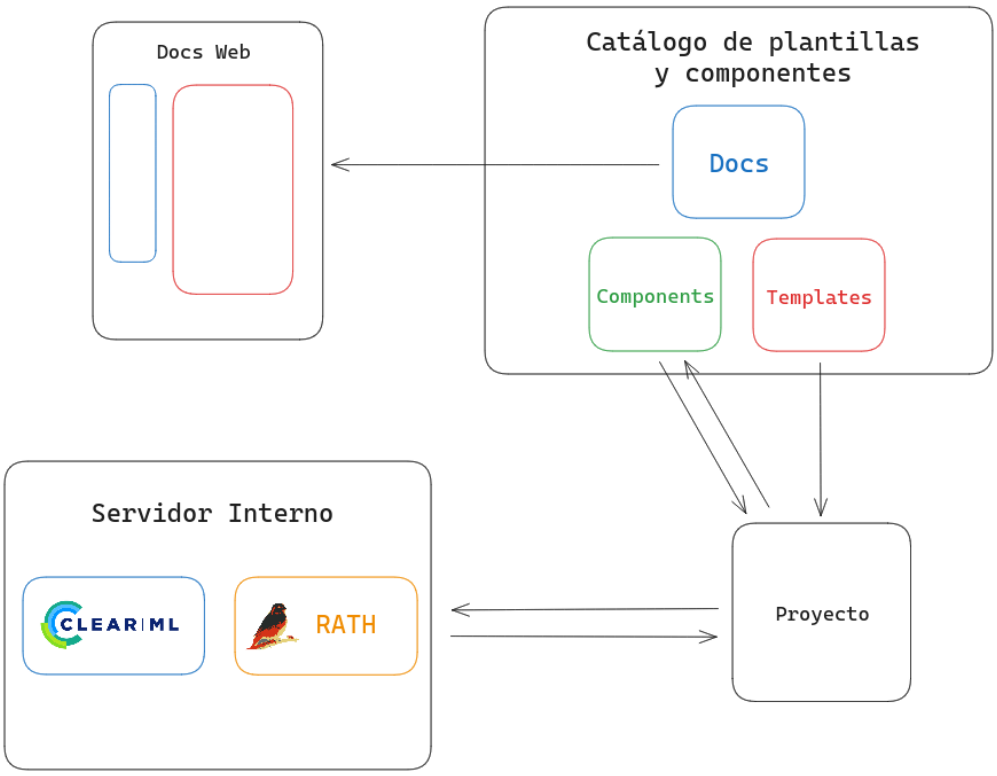

Figure 5.1: Vista general del proyecto

Dentro de la estructura general del proyecto, se pueden identificar tres elementos principales: la infraestructura y herramientas, el catálogo de componentes y la documentación del proyecto. Cada uno de estos elementos se encarga de aspectos diferentes dentro del ecosistema del proyecto, y se relacionan entre sí para formar un sistema completo. La idea principal es que la infraestructura sea complementaria al desarrollo, estando al alcance de los investigadores de una forma sencilla. El catálogo de componentes y plantillas se encarga de proporcionar una base sólida sobre la que construir los diferentes proyectos, automatizando la creación de proyectos siguiendo las mejores prácticas y proporcionando una base sólida sobre la que construir. Por último, la documentación del proyecto se encarga de proporcionar una guía clara y detallada sobre el uso de las herramientas y componentes, así como de los procesos y metodologías empleadas en el

desarrollo del proyecto.

5.1. INFRAESTRUCTURA Y HERRAMIENTAS

5.1.1 Descripción general de la infraestructura

La infraestructura y las herramientas son la base sobre la que se construirán los diferentes proyectos de aprendizaje automático. Se encargan de proporcionar un entorno de desarrollo e investigación eficiente, que permita a los miembros del equipo centrarse en el desarrollo de modelos sin tener que preocuparse por la configuración. Concretamente, se han desplegado dos plataformas que vienen a cubrir varias de las necesidades fundamentales de los proyectos como son la gestión y exploración de datasets, la monitorización de experimentos o el almacenamiento de modelos de inteligencia artificial. Además, se ha añadido un sistema de autenticación para garantizar la seguridad y privacidad de los datos.

Esta infraestructura se ha desplegado en un servidor interno de la empresa utilizando contenedores de Docker. La elección de esta tecnología se debe a que permite la creación de entornos aislados y portables, lo que facilita el despliegue de las aplicaciones. Se ha utilizado Docker Compose para sincronizar el despliegue de los diferentes servicios, lo que permite realizar despliegues automatizados mediante las acciones de GitLab CI. La figura 5.2 muestra una vista general de la infraestructura desplegada en el servidor interno de la empresa, donde se pueden observar como se integran los diferentes servicios y sus respectivas conexiones. Además, se puede observar que todos los servicios están interconectados mediante un proxy inverso mediante Nginx, que se encarga de redirigir las peticiones en función de la URL. Esto permite que todos los servicios sean accesibles desde el exterior a través de un único punto de entrada y que el sistema de autenticación sea común para todos ellos.

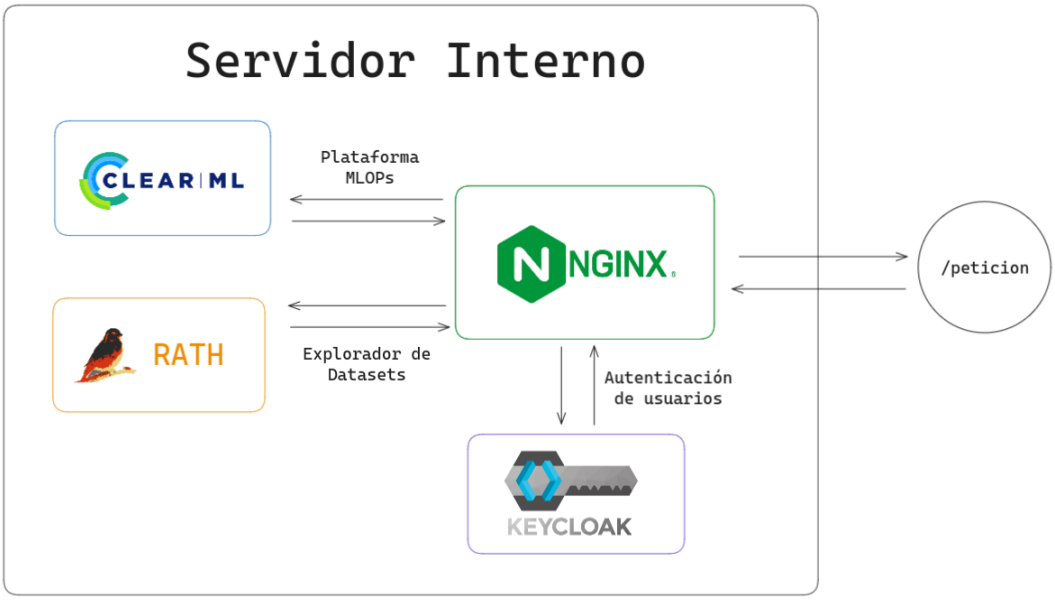

Figure 5.2: Vista general del proyecto

5.1.2 Selección de plataformas

Previo a la elección de las plataformas integradas, se realizó una evaluación a nivel de equipo para determinar las necesidades que se debían cubrir. Se identificaron las siguientes funcionalidades fundamentales:

- **F1:** Versionado y almacenamiento de dataset.
- **F2:** Monitorización de experimentos.
- **F3:** Almacenamiento de modelos de inteligencia artificial.
- **F4:** Integración de métricas en datasets y visualización de resultados.
- **F5:** Exploración de datasets.

Una vez identificadas las necesidades, se consensuó un criterio de selección para las plataformas integradas. Este criterio es un criterio de mínimos, es decir, se seleccionarán las plataformas que cumplan con el criterio mínimo establecido y que, además, ofrezcan funcionalidades adicionales que puedan ser de utilidad para el equipo. El criterio de selección se basa en los siguientes aspectos:

- **C1 (Facilidad de uso):** Se valora muy positivamente la facilidad de uso de las plataformas, ya que se considera que no todo el equipo no tiene experiencia previa en el uso de estas herramientas.
- **C2 (Integración con otras herramientas):** Es fundamental que las plataformas integradas sean compatibles con las librerías y herramientas que se utilizan generalmente en proyectos (TensorFlow, PyTorch, etc.).
- **C3 (Poca dependencia sobre la infraestructura):** Medimos la dependencia sobre una plataforma como el numero de acciones que se deben realizar para migrar un proyecto vanilla, es decir, un proyecto que no ha sido desarrollado con la plataforma en mente. Y penalizando aquellas practicas que sean propias de la plataforma y que no sean comunes en la industria.

Con estos criterios en mente, se tuvieron en cuenta las siguientes plataformas a la hora de realizar la evaluación: MLflow, ClearML, Kedro, ZenML, Data Version Controller (DVC), Rath, Apache Superset. Cada una de estas plataformas tiene diferentes enfoques y funcionalidades, pero todas ellas cubren una o varias de las necesidades fundamentales identificadas, por lo que se consideraron candidatas para su integración en la infraestructura. La infraestructura final se compondrá de una o varias de estas plataformas en función de los criterios previamente establecidos. A continuación, se muestra un análisis detallado de las plataformas evaluadas y las funcionalidades que ofrecen.

- **MLflow:** MLflow es una plataforma MLOPs de código abierto para la gestión del ciclo de vida de los modelos. Ofrece una interfaz de usuario para el seguimiento de experimentos, la gestión de modelos y la implementación de modelos en diferentes entornos. MLflow es compatible con la mayor parte de librerías de aprendizaje automático, como TensorFlow o PyTorch. Uno de los puntos fuertes de MLflow es su gran comunidad, ya que es una de las plataformas más utilizadas en la industria. Sin embargo, no ofrece funcionalidades relacionadas con la gestión, evaluación o versionado de datasets ni con la exploración de los

mismos. La dependencia sobre la plataforma varía en función de la tarea que se quiera realizar, pero en general, es una plataforma que no ata al usuario a su ella. La documentación se queda corta en cuestión de claridad y ejemplos, lo que puede dificultar la adopción de la plataforma por parte de los miembros del equipo.

- **ClearML:** ClearML al igual que MLflow, es una plataforma MLOps de código abierto que ofrece las mismas funcionalidades que MLflow en relación a la gestión a la gestión de experimentos, pero con la diferencias, que este si cuenta con funcionalidades relacionadas con la gestión, evaluación o versionado de datasets. ClearML también es compatible con la mayor parte de librerías populares aunque no tantas ni tan variadas como MLflow, pero ofrece una API que permite de la integración de estas de forma manual. La dependencia sobre la plataforma es mínima, ya que con pocos cambios se pueden lanzar experimentos sobre un código base. La documentación es clara y ofrece ejemplos en formato tanto de texto como de video, con proyectos sencillos y claros que permiten entender rápidamente el funcionamiento de la plataforma. El principal punto débil de ClearML es que no cuenta con una gran comunidad, lo que puede dificultar a la hora de encontrar soluciones a ciertas problemáticas. Otro de sus puntos débiles es que por defecto no cuenta con un sistema de autenticación robusto, lo que te obliga a implementar uno por tu cuenta.

- **Data Version Controller (DVC):** DVC y DVC Studio son dos herramientas de código abierto que están diseñadas para manejar grandes volúmenes de datos, modelos y experimentos. DVC se centra en el versionado y almacenamiento de datasets, mientras que DVC Studio se centra en la monitorización de experimentos, visualización de resultados y almacenamiento de modelos. Además, DVC Live proporciona integraciones con un número considerable de librerías de aprendizaje automático. La documentación está bien estructurada aunque no es tan clara como la de ClearML, pero cuenta con una comunidad bastante activa. En cuanto a los aspectos negativos de DVC, la principal desventaja es que la curva de aprendizaje es bastante pronunciada, lo que dificulta su adopción. Otro punto en contra es la dependencia gigantesca que tienen los proyectos que usan DVC, ya que se necesita de muchos archivos de configuración, integraciones manuales dentro del código y un dominio completo de los comandos de la herramienta para poder trabajar con ella.

- **Kedro:** TODO:

- **ZenML:** TODO:

- **Rath:** herramienta de exploración de datasets que permite una exploración semi-automática de datasets. Muy fácil de utilizar, ya que cuenta con un modo copilot que te sugiere diferentes gráficos y estadísticas en función de los datos que estés explorando. Además, no requiere de ninguna configuración previa, ya que se puede utilizar directamente desde el navegador.

- **Apache Superset:** Apache Superset es una plataforma de visualización de datos de código abierto que ofrece una amplia gama de características para explorar y visualizar datos de manera interactiva. Con una interfaz intuitiva y basada en web, Superset permite a los usuarios crear paneles de control, gráficos y tablas dinámicas con facilidad. Además, ofrece integraciones con diversas fuentes de datos y admite la creación de paneles de control en tiempo real. Una de las fortalezas de Apache Superset es su comunidad activa y en constante crecimiento, lo que garantiza un soporte sólido y una mejora continua de la plataforma. Sin embargo, la integración con el resto de herramientas es limitada, ya que el propio Superset requiere de un formato de datos concreto para poder subir los datos a

la plataforma, lo que puede dificultar su integración y generar duplicidad en cuanto a los datos almacenados, ya que se necesitaría una copia de los datos en el formato que requiere Superset. Nuestro objetivo es que esta herramienta agilice el proceso de exploración de datos, por lo que no es una opción viable por el momento.

- **Grafana:** Grafana es una plataforma de análisis y visualización de métricas de código abierto que se ha convertido en una opción popular para monitorear sistemas y aplicaciones. Con una amplia gama de complementos y paneles personalizables, Grafana permite a los usuarios crear cuadros de mando y gráficos altamente personalizados para visualizar datos de diferentes fuentes. Además, ofrece características avanzadas como alertas, anotaciones y exploración de datos en tiempo real. Grafana es altamente modular y extensible, lo que facilita su integración con diversas tecnologías y sistemas de monitoreo. El problema de Grafana es el mismo que el de Superset, ya que requiere de un formato de datos concreto para poder subir los datos a la plataforma, lo que puede dificultar su integración y generar duplicidad en cuanto a los datos almacenados. Además, de su curva de aprendizaje, que es bastante pronunciada.

Tecnología	**Funcionalidades**					**Criterios**		
	F1	F2	F3	F4	F5	C1	C2	C3
MLflow	–	✓	✓	–	–	✓	✓	✓
ClearML	✓	✓	✓	✓	–	✓	✓	✓
DVC	✓	✓	✓	✓	–	–	✓	–
Kedro	–	–	–	–	–	–	–	–
ZenML	–	–	–	–	–	–	–	–
Rath	–	–	–	–	✓	✓	–	✓
Apache Superset	✓	–	–	✓	–	–	–	✓
Grafana	✓	–	–	✓	–	–	–	✓

Table 5.1: Tabla comparativa de las plataformas evaluadas

Para finalizar con la evaluación, se puede observar en la tabla 5.1 una comparativa que muestra las funcionalidades y criterios que cumple cada una de ellas de forma resumida. De acuerdo con la evaluación realizada, se ha decidido integrar ClearML y Rath en la infraestructura como las plataformas finales. ClearML cubre la mayoría de las necesidades fundamentales identificadas, ya que ofrece soluciones para la mayoría de casos de uso, mientras que Rath complementa con la exploración de datasets de forma sencilla e intuitiva.

5.1.3 Seguridad y priviacidad

La seguridad y la privacidad de los datos son aspectos fundamentales dentro de un entorno de trabajo empresarial, donde la confidencialidad de los datos es una prioridad para la empresa. Por ello, ya que ClearML y Rath no cuentan con un sistema de autenticación robusto, se ha decidido implementar un sistema de autenticación utilizando Keycloak, que es una solución de código abierto para la gestión de acceso. Keycloak permite a los usuarios autenticarse con cuentas de usuario gestionadas por la empresa, lo que garantiza que solo los usuarios autorizados puedan acceder a los datos. Además, ofrece integraciones con diferentes proveedores de identidad, lo que facilita la gestión de usuarios y la integración con otras herramientas.

Debido a que tanto ClearML como Rath no tienen integraciones directas con Keycloak, se ha decidido utilizar Nginx como proxy inverso para redirigir las peticiones a los servicios de ClearML y Rath. De esta forma, se puede utilizar Keycloak para autenticar a los usuarios y garantizar la seguridad y la privacidad de los datos. La figura 5.2 muestra cómo se ha integrado Keycloak en la infraestructura, así como las conexiones entre los diferentes servicios y el proxy inverso. En el momento actual, se ha implementado un sistema de autenticación básico, pero se espera que en el futuro este sistema se sincronice con el sistema de autenticación de Tecnalia para una mayor comodidad por parte de los usuarios.

5.1.4 Despligue Automatizado

Para la automatización del despliegue de la infraestructura se ha utilizado GitLab CI, que es una herramienta de integración y despliegue continuo que permite automatizar el proceso de despliegue de aplicaciones. GitLab CI permite definir un conjunto de pasos que se ejecutarán cada vez que se realice un cambio en el repositorio, lo que facilita el despliegue de aplicaciones y la gestión de la infraestructura.

Para poder utilizar GitLab CI, es necesario primero definir un runner, que es un agente que se encarga de ejecutar los pasos definidos en el archivo de configuración. En este caso, el runner se ha desplegado en el servidor lo que permite ejecutar los diferentes flujos en el mismo entorno. Además, otro aspecto a tener en cuenta es que por seguridad, se ha configurado el sistema con variables de entorno que enmascaran las credenciales de acceso a los diferentes servicios, lo que garantiza que las credenciales no se almacenen en el repositorio. En cuanto a las pipelines, se han definido tres diferentes flujos para automatizar cada uno de los procesos de instalación y actualización:

- **Instalación de autenticación:** Se encarga de instalar Keycloak en el servidor junto con la base de datos. Este flujo se ejecutará si no se ha instalado Keycloak previamente.
- **Instalación de infraestructura:** Se encarga de instalar ClearML y Rath en el servidor. Además, también se encarga de configurar el proxy inverso para redirigir las peticiones a los servicios de ClearML y Rath. Este flujo se ejecutará si no se ha instalado ClearML previamente.
- **Despliegue de infraestructura:** Se encarga de actualizar el servidor a los nuevos cambios realizados en el repositorio. Este flujo se lanza cada vez que detecta un cambio en el repositorio, pero requiere de una aprobación manual para ejecutarse.

La idea de estos flujos es que se puedan ejecutar de forma independiente y que en caso de que se necesite realizar un cambio en la infraestructura, se pueda lanzar el flujo en otro servidor para instalar la infraestructura de forma automática. Además, facilita el trabajo en local, ya que una vez automatizado el despliegue, los cambios en local se pueden probar lanzar directamente en el servidor.

5.1.5 Problemas durante el Despliegue

A continuación se detallan la lista de problemas que se han encontrado durante el despliegue de la infraestructura y las soluciones que se han aplicado para resolverlos. Estos soluciones se detallan con el fin de que puedan ser de utilidad para futuros despliegues o como registro para mejoras futuras.

- **Clearml no es compatible con servidores no IPv6.** Por defecto, ClearML utiliza direcciones IPv4 e IPv6 para comunicarse con el servidor. Sin embargo, en el servidor interno de la empresa no está habilitado el soporte para direcciones IPv6, lo que provoca que ClearML no pueda comunicarse con el servidor. Internamente, ClearML utiliza una configuración de Nginx para esta comunicación, pero no es posible modificarla directamente ya que el contenedor de Docker lanza un script en bash que sobrescribe la configuración de Nginx y luego lanza el servidor. La solución a este problema es sobrescribe una variable interna llamada *DISABLE_NGINX_IPV6* dentro del contenedor de Webserver de ClearML. Este error es común pero no está documentado en la documentación oficial, la solución es una propuesta personal que ha sido descubierta mediante la lectura del código.
- **ClearML no renderiza correctamente en el navegador.** Otro de los problemas que encontramos al desplegar ClearMl es que el servidor interno de la empresa no era capaz de servir archivos estáticos de un tamaño superior a 3.5MB. Esto provocaba que la interfaz de usuario de ClearML no se renderizara correctamente los gráficos, ya que internamente utiliza plotly para la visualización de los mismos. La solución a este problema fue recompilar la libreria de plotly por nuestra cuenta incluyendo solamente los gráficos que necesitamos, de esta forma reducimos el tamaño de los archivos estáticos y conseguimos que la interfaz de usuario se renderizara correctamente.

5.2. CATALOGO DE COMPONENTES

El catalogo de componentes y plantillas es una herramienta de gestión de conocimiento que permite a los integrantes de un equipo compartir, reutilizar y colaborar en la creación de estándares para una mayor eficiencia en el desarrollo de modelos de IA. Los componentes son elementos que representan pequeñas funcionalidades dentro de un proyecto que pueden ser reutilizados de una forma sencilla. Las plantillas por otro lado son estructuras más complejas que agrupan varios componentes, configuraciones y reglas de negocio para crear un modelo dentro de una temática específica por cada una de ellas. A todos estos elementos los hemos denominado bajo el nombre de STAC (Simple Tecnalia AI Components).

5.2.1 Adaptación del diseño atómico

La metodología de diseño atómico es una técnica que se basa en la creación de componentes, que se pueden reutilizar en diferentes partes de un proyecto. En el desarrollo de modelos de IA se puede adaptar esta técnica para crear componentes que representen funcionalidades específicas, como el preprocesamiento de datos, la selección de características, la evaluación de modelos, entre otros.

Como hemos mencionado anteriormente, los componentes se dividen a su vez en multitud de categorías (atómicos, moleculares, organismos, plantillas, etc.) esta división tiene sentido dentro de su concepción original pero en el caso de los modelos de IA, la división de los componentes se puede hacer de una forma más sencilla, ya que en el caso de la investigación en IA, una abstracción más sencilla puede ser más útil para los investigadores. Es por ello que se propone una división de los componentes en tres categorías:

- **Componentes atómicos:** son los componentes más sencillos que representan una funcionalidad específica.
- **Componentes compuestos:** estos componentes agrupan varios componentes atómicos para realizar una funcionalidad más compleja.

- **Plantillas:** estructuras de proyectos completan que buscan solucionar un problema específico utilizando una técnica concreta. Están formadas a su vez por componentes compuestos y atómicos. Además, las plantillas también agrupan configuraciones, reglas de negocio y diversas integraciones con otros sistemas.

Todos los componentes y plantillas se almacenan en un repositorio compartido, donde los integrantes del equipo pueden colaborar en la creación de nuevos componentes y plantillas, así como en la mejora de los existentes.

5.2.2 Estructura del sistema de componentes

La arquitectura que se ha decidido implementar para el sistema de componentes es conocida como monorepo multi-paquete. Un monorepo es una práctica de desarrollo de software donde todos los proyectos relacionados se almacenan en un único repositorio de código fuente. Esto significa que en lugar de tener múltiples repositorios para cada uno de los componentes, todo se mantienen en un único lugar. Por otro lado, que sea multi-paquete significa que que los diferentes elementos del monorepo se organizan mediante paquetes de software, lo que facilita su distribución de forma independiente. Este tipo de arquitectura cuenta con varias ventajas, entre las que se encuentran:

- **Facilidad de gestión:** Tener todo en un solo lugar simplifica la gestión del código, las dependencias y las versiones. No es necesario alternar entre diferentes repositorios para hacer cambios o resolver problemas.
- **Consistencia:** Todos los proyectos dentro del monorepo pueden seguir las mismas convenciones, estructura de carpetas, y configuraciones, lo que garantiza una mayor consistencia en el código. Esto permite que el código sea más fácil de mantener a largo plazo.

Uno de los principales problemas que puede surgir al utilizar un monorepo es la complejidad que puede acarrear la organización de las diferentes carpetas, ya que al tener todo en un solo lugar, la cantidad de archivos puede llegar a ser muy grande y contar con un nivel de anidamiento muy profundo. Para evitar este problema, se ha decidido tomar una estructura de carpetas basada en *Screaming architecture*, una arquitectura de software que busca anteponer la lógica de negocio sobre las partes técnicas del sistema. En este caso, nuestra lógica de negocio se relaciona en torno a el problema que se busca resolver, es decir los diferentes conjuntos de problemas de las series temporales.

La figura 5.2.4 muestra un ejemplo simplificado de la estructura de carpetas del proyecto. Por cada uno de los diferentes problemas, se ha creado una carpeta principal, que a su vez contiene diferentes subcarpetas que agrupan los diferentes componentes dentro de cada una de las temáticas (procesamiento, modelo, métricas). En caso de que se necesite añadir un nuevo componente, simplemente se crea una nueva carpeta dentro de la temática correspondiente. Por ejemplo, tomando la temática preprocesamiento de datos dentro de clasificación, se buscaría la carpeta *Data-Processing* dentro *TimeSeries-Classification* y en caso de no existir, se crearía una nueva carpeta, y dentro de ella se añadiría el nuevo componente.

Lo que se consigue con este enfoque es que cada uno de los diferentes problemas cuente con una estructura similar pero que a su vez tengan la posibilidad de incluir temáticas propia. En la figura 5.2.4 se puede ver como cada uno de los problemas cuenta con componentes de procesamiento de datos, pero que a su vez, hay algunos de ellos que cuentan con componentes específicos de modelos o métricas. Esto se puede deber a que cada uno de los problemas tiene una necesidades específicas o que no se han encontrado componentes que se puedan reutilizar en este momento.

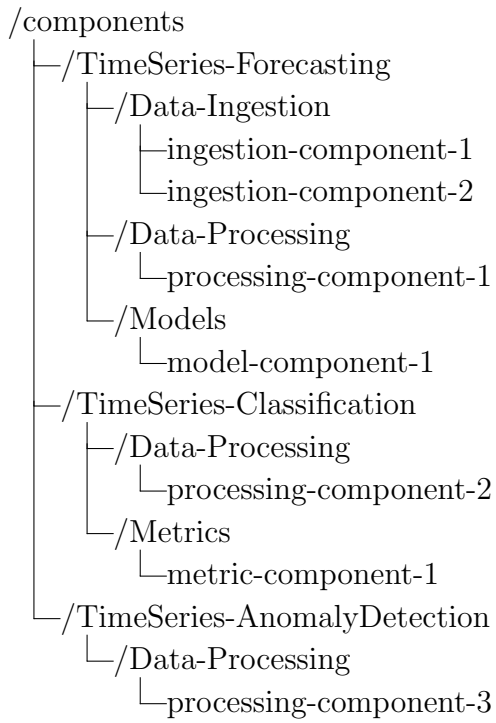

Figure 5.3: Ejemplo de estructura de carpetas basada en Screaming Architecture.

5.2.3 Empaquetado de componentes

La idea de los componentes es que sean fácilmente integrables en cualquier proyecto de una forma sencilla. Para ello, se ha decidido empaquetar cada componente de forma independiente y distribuirlos dentro de un repositorio privado, de forma que se puedan instalar utilizando pip o cualquier otro gestor de paquetes como poetry.

Dentro de nuestro caso de uso, queremos tener la posibilidad de instalar solo aquellos componentes que necesitemos en cada momento, sin tener que descargarnos todo el contenido del repositorio. Esto es especialmente importante ya que las dependencias de las librerías de IA pueden llegar a ser muy pesadas. A su vez, también se quiere que todos nuestros componentes hereden del mismo nombre de paquete, para que se puedan utilizar de una forma transparente a la hora de importarlos en un proyecto. Es decir, si tenemos por ejemplo un componentes de ingesta de datos llamado *get_data* y otro de visualización que imprime una gráfica llamado *show_graph*, la forma de importarlo en un proyecto sería la siguiente:

```
$ pip install stac-show-graph stac-get-data

>> from stac.visualization.show_graph import show_graph
>> from stac.data_ingestion.get_data import get_data
>> data = get_data()
>> show_graph(data)
```

Para conseguir este efecto es necesario comprender como funciona el empaquetado de librerías en Python. En Python, un paquete es una carpeta que contiene, por lo menos, un archivo

__init__.py y un *pyproject.toml* o *setup.py*. La forma en la que se importan los paquetes es a través de un sistema de rutas por archivos, donde se toma la ruta hacia el paquete desde el directorio raíz del proyecto. En la figura 5.4 se puede ver como es el contenido de cada componente.

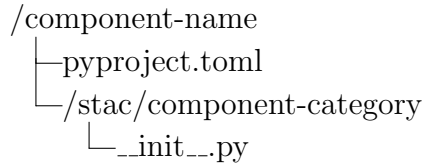

Figure 5.4: Estructura mínima de un componente empaquetado.

Esta estructura se puede repetir para cada uno de los componentes que se quieran añadir al repositorio. No obstante, y aprovechando que cada componente está separada en un proyecto independiente, se pueden añadir más utilidades como tests, documentación o ejemplos de uso. Como estamos utilizando poetry para gestionar las dependencias, y las configuraciones globales del proyecto, también se van a añadir sus archivos correspondientes.

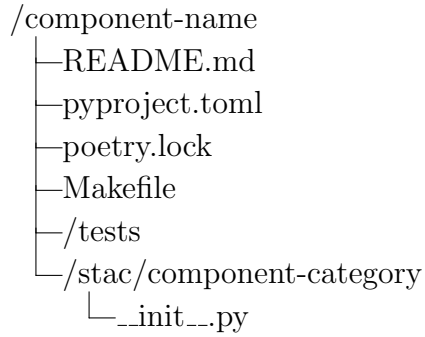

Figure 5.5: Estructura final de un componente empaquetado.

Este empaquetado puede parecer tedioso si se tiene que hacer manualmente, pero en futuras secciones veremos como se puede automatizar este proceso para que sea lo más sencillo posible. Es muy importante que la estructura sea la descrita para optimizar la experiencia de desarrollo de los integrantes del equipo.

5.2.4 Sistema de plantillas

La herramienta que hemos utilizado para la creación de plantillas es Cookiecutter. Cookiecutter es una utilidad para la generación de proyectos que te permite inicializar un proyecto a partir de plantillas predefinidas. Funciona siguiendo un principio básico: en lugar de empezar un nuevo proyecto desde cero y tener que configurar todo manualmente, puedes usar una plantilla predefinida que ya incluya la estructura de directorios, archivos básicos, configuraciones iniciales, y cualquier otro componente necesario para tu proyecto.

Esto es especialmente útil en entornos donde necesitas crear múltiples proyectos con una estructura similar donde puede haber proyectos con requisitos comunes, como la configuración de un marco de trabajo específico, estructura de directorios estándar, o incluso configuraciones

de pruebas y documentación. Además, en nuestro caso son especialmente útiles ya que contamos con dos situaciones en las que tener una plantilla puede ser muy útil:

- **Creación de plantillas para problemas específicos:** este es el caso principal de el uso de plantillas. Cada una de las plantillas se crea para resolver un problema específico con una técnica concreta. Por ejemplo, una plantilla para clasificación de series temporales con redes neuronales recurrentes.
- **Creación de nuevos componentes:** en caso de que se quiera añadir un nuevo componente al repositorio, se puede utilizar una plantilla que contenga la estructura de carpetas y los archivos necesarios para empaquetar el componente. Como hemos mencionado anteriormente, esto puede ser especialmente engorroso si no se tiene una plantilla que te ayude a automatizar el proceso.

La razón por la elección de Cookiecutter es por su facilidad de uso, ya que al utilizar el sistema de plantillas Jinja2, se pueden añadir variables a los archivos de la plantilla que se sustituirán por valores concretos al inicializar el proyecto. Este sistema de plantillas es muy conocido especialmente ente los desarrolladores de Python, ya que es ampliamente utilizado por otros frameworks como Django. Además, Cookiecutter cuenta con una gran cantidad de plantillas predefinidas que se pueden utilizar de forma gratuita, lo que puede ser muy útil a la hora de incorporar las primeras plantillas al repositorio.

Estas plantillas se almacenan de la misma forma que podemos ver en la figura pero sustituyendo los componentes por las plantillas. En principio, la estructura de las plantillas puede variar en gran medida entre ellas, ya que cada una de ella busca resolver un problema específico. No obstante, hay una serie de directrices obligatorias que se deben seguir para garantizar el correcto funcionamiento de las plantillas:

- **Automatización de instalación y lanzamiento:** todas las plantillas deben contar con un makefile que permita la instalación de las dependencias y el lanzamiento de la aplicación de forma sencilla. Por convención, el comando de instalación se llamará *make install* y el de lanzamiento *make train*. Es muy importante que estos comandos estén presentes en todas las plantillas para garantizar que con solo dos comandos y sin necesidad de consultar la documentación especifica de cada plantilla se pueda poner en marcha el proyecto en concreto.

5.2.5 Integración continua y despliegue continuo

Dentro de equipos de trabajo donde hay un número elevado de integrantes, es importante no solo tener una buena organización del código, sino también contar con un sistema de CI/CD que asegure los diferentes procesos. En nuestro caso es especialmente critico ya que los componentes son utilizados por diferentes personas en diferentes proyectos, por lo que un fallo en uno de los componentes puede afectar a un gran número de proyectos.

Para la implementación se ha utilizado GitLab CI, con una distribución de tres fases: *test*, *docs* y *deploy*. En la fase de test se ejecutan los test de cada uno de los componentes, en la fase de docs se extraen todos los archivos de documentación y se genera una pagina web estática con toda la documentación del proyecto, y en la fase de deploy hace una comprobación de que paquetes es necesario desplegar y en caso de que haya cambios, se despliegan las nuevas versiones de los paquetes, aclarar que se mantienen las versiones anteriores para evitar problemas de compatibilidad.

Estas fases se ejecutan en diferentes momentos, al estar la rama principal bloqueada por defecto, es necesario crear una rama con las nuevas funcionalidades y/o modificaciones y hacer una petición de merge a la rama principal. En este momento se ejecutan las fases de test y docs, para comprobar que no hay errores en el código y que la documentación se puede generar correctamente, aunque esta última no se despliega hasta que se haya aceptado la petición de merge. En caso de que se acepte la petición de merge, se ejecuta la fase de deploy. Existen usuarios que excepcionalmente pueden realizar modificaciones sobre la rama principal, pero solo en caso de que sea necesario y no como una práctica habitual.

5.3. DOCUMENTACIÓN DEL PROYECTO

La documentación juega un papel crucial en este proyecto, ya que proporciona una guía detallada sobre cómo utilizar los distintos componentes y plantillas disponibles. En este contexto, la adopción de Astro como herramienta para la generación de documentación automática se debe a su facilidad de uso, ya que permite crear paginas de forma sencilla utilizando Markdown, lo que facilita la creación de documentos por los distintos integrantes del equipo. Ya existen proyectos que han utilizado Astro para la generación de documentación, como Microsoft con su proyecto Fluent UI que utiliza Astro.

Concretamente, el proyecto usa Starlight, una plantilla de Astro que proporciona una estructura de documentación predefinida, lo que facilita la creación de documentación de forma rápida y sencilla. Además, Starlight incluye un sistema de búsqueda y organización de documentación, lo que facilita la navegación y búsqueda de información en la documentación. Otra de sus ventajas es la facilidad de customización, ya que permite modificar la apariencia utilizando CSS o Tailwind e incluso incorporar funcionalidades propias mediante JavaScript.

5.3.1 Guías y manuales

Para facilitar la integración de sistemas complejos como el aquí propuesto, es necesario proporcionar guías que expliquen de forma detallada cómo se debe interactuar con los diferentes procesos. En este sentido, la documentación del proyecto incluye un apartado de guías y manuales que proporciona esa información de forma detallada. Además, se incluyen ejemplos de uso y casos prácticos que ayudan a entender cómo se deben utilizar los distintos elementos de forma interactiva y dinámica.

En este momento, la documentación incluye las siguientes guías:

- **Guía de introducción:** proporciona una visión general del proyecto y que partes lo componen.
- **Guía de instalación:** explica cómo instalar las diferentes herramientas en local y conseguir acceso a los servicios.
- **Guía de plantillas y componentes:** proporciona información detallada sobre cómo utilizar los distintos componentes y plantillas disponibles.
- **Guía de ClearMl:** introducción a la plataforma MLOps ClearMl y cómo utilizarla en el proyecto.
- **Guía de contribución:** proporciona información sobre cómo contribuir al proyecto y cómo colaborar con otros miembros del equipo.

En el futuro, se espera ampliar la documentación con nuevas guías que no solo expliquen cómo utilizar los distintos elementos sino que también se centren en aspectos enfocados a la IA y el

aprendizaje automático, como por ejemplo guías sobre cómo solucionar problemas relacionados con el día a día de la empresa.

5.3.2 Construcción automática de documentación

Como se ha mencionado anteriormente, cada componente y plantilla viene con su propia documentación, la cual se genera dentro de su respectiva carpeta mediante un archivo Readme.md.

5.3.3 Documentación de componentes y plantillas

5.3.4 Sistema de búsqueda y organización de documentación

6. CONCLUSIONES Y TRABAJO A FUTURO

6.1. CONCLUSIONES

6.2. TRABAJO A FUTURO

BIBLIOGRAFÍA

[1] "WHAT IS AGILE". en. In: *The Power of the Agile Business Analyst, second edition*. IT Governance Publishing, 2018, pp. 22–25. ISBN: 9781849289955.

[2] Atlassian. *What is Scrum?* 2018.